Top Google Questions

Download PDF

- 78. Subsets
- 79. Word Search
- 80. Remove Duplicates from Sorted Array II
- 81. Search in Rotated Sorted Array II
- 83. Remove Duplicates from Sorted List
- 84. Largest Rectangle in Histogram
- 88. Merge Sorted Array
- 94. Binary Tree Inorder Traversal
- 96. Unique Binary Search Trees
- 101. Symmetric Tree
- 102. Binary Tree Level Order Traversal
- 103. Binary Tree Zigzag Level Order Traversal
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 109. Convert Sorted List to Binary Search Tree
- 110. Balanced Binary Tree
- 114. Flatten Binary Tree to Linked List
- 118. Pascal's Triangle
- 119. Pascal's Triangle II
- 121. Best Time to Buy and Sell Stock
- 129. Sum Root to Leaf Numbers
- 130. Surrounded Regions
- 136. Single Number
- 137. Single Number II
- 144. Binary Tree Preorder Traversal
- 145. Binary Tree Postorder Traversal
- 146. LRU Cache
- 153. Find Minimum in Rotated Sorted Array
- 162. Find Peak Element
- 163. Missing Ranges
- 169. Majority Element
- 173. Binary Search Tree Iterator
- 174. Dungeon Game
- 198. House Robber
- 200. Number of Islands
- 205. Isomorphic Strings
- 207. Course Schedule
- 208. Implement Trie (Prefix Tree)
- 219. Contains Duplicate II
- 221. Maximal Square
- 222. Count Complete Tree Nodes
- 226. Invert Binary Tree
- 230. Kth Smallest Element in a BST
- 231. Power of Two
- 237. Delete Node in a Linked List
- 242. Valid Anagram
- 246. Strobogrammatic Number

- 249. Group Shifted Strings
- 250. Count Univalue Subtrees
- 251. Flatten 2D Vector
- 252. Meeting Rooms
- 253. Meeting Rooms II
- 257. Binary Tree Paths
- 265. Paint House II
- 267. Palindrome Permutation II
- 268. Missing Number
- 270. Closest Binary Search Tree Value
- 274. H-Index
- 276. Paint Fence
- 278. First Bad Version
- 285. Inorder Successor in BST
- 286. Walls and Gates
- 287. Find the Duplicate Number
- 290. Word Pattern
- 299. Bulls and Cows
- 303. Range Sum Query - Immutable
- 305. Number of Islands II
- 311. Sparse Matrix Multiplication
- 314. Binary Tree Vertical Order Traversal
- 322. Coin Change
- 323. Number of Connected Components in an Undirected Graph
- 326. Power of Three
- 328. Odd Even Linked List
- 344. Reverse String
- 345. Reverse Vowels of a String
- 347. Top K Frequent Elements
- 348. Design Tic-Tac-Toe
- 367. Valid Perfect Square
- 368. Largest Divisible Subset
- 374. Guess Number Higher or Lower
- 379. Design Phone Directory
- 380. Insert Delete GetRandom O(1)
- 384. Shuffle an Array
- 387. First Unique Character in a String
- 389. Find the Difference
- 392. Is Subsequence
- 402. Remove K Digits
- 404. Sum of Left Leaves
- 406. Queue Reconstruction by Height
- 409. Longest Palindrome
- 415. Add Strings
- 429. N-ary Tree Level Order Traversal
- 438. Find All Anagrams in a String

- 447. Number of Boomerangs
- 451. Sort Characters By Frequency
- 468. Validate IP Address
- 470. Implement Rand10() Using Rand7()
- 475. Heaters
- 482. License Key Formatting
- 498. Diagonal Traverse
- 510. Inorder Successor in BST II
- 518. Coin Change 2
- 525. Contiguous Array
- 528. Random Pick with Weight
- 535. Encode and Decode TinyURL
- 540. Single Element in a Sorted Array
- 542. 01 Matrix
- 551. Student Attendance Record I
- 567. Permutation in String
- 609. Find Duplicate File in System
- 616. Add Bold Tag in String
- 658. Find K Closest Elements
- 669. Trim a Binary Search Tree
- 684. Redundant Connection
- 679. 24 Game
- 693. Binary Number with Alternating Bits
- 694. Number of Distinct Islands
- 700. Search in a Binary Search Tree
- 702. Search in a Sorted Array of Unknown Size
- 705. Design HashSet
- 706. Design HashMap
- 711. Number of Distinct Islands II
- 717. 1-bit and 2-bit Characters
- 720. Longest Word in Dictionary
- 721. Accounts Merge
- 733. Flood Fill
- 734. Sentence Similarity
- 737. Sentence Similarity II
- 748. Shortest Completing Word
- 758. Bold Words in String
- 773. Sliding Puzzle
- 787. Cheapest Flights Within K Stops
- 819. Most Common Word
- 844. Backspace String Compare
- 868. Binary Gap
- 917. Reverse Only Letters
- 973. K Closest Points to Origin
- 986. Interval List Intersections
- 988. Smallest String Starting From Leaf

1. Two Sum

Description

```
Given an array of integers, return indices of the two numbers such that they add
up to a specific target.

You may assume that each input would have exactly one solution, and you may not
use the same element twice.

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

Solution

01/02/2020:

```
class Solution {
public:
  vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> m;
    for (int i = 0; i < nums.size(); ++i) {
      if (m.find(target - nums[i]) == m.end()) {
        m[nums[i]] = i;
      } else {
        return {m[target - nums[i]], i};
      }
    }
    return {-1, -1};
  }
};
```

3. Longest Substring Without Repeating Characters

Description

```
Given a string, find the length of the longest substring without repeating
characters.

Example 1:

Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
Example 2:

Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
Example 3:

Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
             Note that the answer must be a substring, "pwke" is a subsequence
and not a substring.
```

Solution

05/21/2020:

```cpp
class Solution {
public:
  int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> seen;
    int ret = 0, slow = 0, n = s.size();
    for (int fast = 0; fast < n; ++fast) {
      if (seen.count(s[fast]) != 0) slow = max(slow, seen[s[fast]] + 1);
      seen[s[fast]] = fast;
      ret = max(ret, fast - slow + 1);
    }
    return ret;
  }
};
```

4. Median of Two Sorted Arrays

Description

```
There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should
be O(log (m+n)).

You may assume nums1 and nums2 cannot be both empty.

Example 1:

nums1 = [1, 3]
nums2 = [2]

The median is 2.0
Example 2:

nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

Solution

01/02/2020:

```cpp
class Solution {
```

```
public:
  double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    int n = nums1.size() + nums2.size();
    vector<int> nums(n, 0);
    auto it1 = nums1.begin();
    auto it2 = nums2.begin();
    auto it = nums.begin();
    for (; it != nums.end(); ++it) {
      *it = it2 == nums2.end() || (it1 != nums1.end() && *it1 < *it2) ? *it1++ :
*it2++;
    }
    if (n % 2 == 0) {
      return ((double) nums[n/2] + nums[n/2-1]) / 2;
    } else {
      return (double) nums[n/2];
    }
  }
};
```

5. Longest Palindromic Substring

Description

```
Given a string s, find the longest palindromic substring in s. You may assume
that the maximum length of s is 1000.

Example 1:

Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
Example 2:

Input: "cbbd"
Output: "bb"
```

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  string longestPalindrome(string s) {
    int n = s.size(), start, max_len = 0;
    if (n == 0) return "";
```

```
    vector<vector<bool>> dp(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i) dp[i][i] = true;
    for (int i = 0; i < n - 1; ++i) dp[i][i + 1] = s[i] == s[i + 1];
    for (int i = n - 3; i >= 0; --i) {
      for (int j = i + 2; j < n; ++j) {
        dp[i][j] = dp[i + 1][j - 1] && s[i] == s[j];
      }
    }
    for (int i = 0; i < n; ++i) {
      for (int j = i; j < n; ++j) {
        if (dp[i][j] && j - i + 1 > max_len) {
          max_len = j - i + 1;
          start = i;
        }
      }
    }
    return s.substr(start, max_len);
  }
};
```

01/13/2020 (Expand Around Center):

```
class Solution {
public:
  string longestPalindrome(string s) {
    int n = s.size(), start = 0, max_len = n > 0 ? 1 : 0;
    for(int i = 0; i < n; ++i) {
      for (int l = i - 1, r = i; l >= 0 && r < n && s[l] == s[r]; --l, ++r) {
        if (r - l + 1 > max_len) {
          max_len = r - l + 1;
          start = l;
        }
      }
      for (int l = i - 1, r = i + 1; l >= 0 && r < n && s[l] == s[r]; --l, ++r)
{
        if (r - l + 1 > max_len) {
          max_len = r - l + 1;
          start = l;
        }
      }
    }
    return max_len == 0 ? "" : s.substr(start, max_len);
  }
};
```

6. ZigZag Conversion

Description

```
The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of
rows like this: (you may want to display this pattern in a fixed font for better
legibility)

P   A   H   N
A P L S I I G
Y   I   R
And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number
of rows:

string convert(string s, int numRows);
Example 1:

Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
Example 2:

Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"
Explanation:

P     I    N
A   L S  I G
Y A   H R
P     I
```

Solution

05/24/2020:

```
class Solution {
public:
  string convert(string s, int numRows) {
    if (s.empty()) return "";
    if (numRows <= 1) return s;
    int k = 0, increment = 1;
    vector<string> strs(numRows);
    for (int i = 0; i < (int)s.size(); ++i) {
      strs[k].push_back(s[i]);
      if (k == numRows - 1) {
        increment = -1;
```

```
      } else if (k == 0) {
        increment = 1;
      }
      k += increment;
    }
    string ret;
    for (auto& s : strs) ret += s;
    return ret;
  }
};
```

8. String to Integer (atoi)

Description

```
Implement atoi which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the
first non-whitespace character is found. Then, starting from this character,
takes an optional initial plus or minus sign followed by as many numerical
digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral
number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid
integral number, or if no such sequence exists because either str is empty or it
contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

Note:

Only the space character ' ' is considered as whitespace character.
Assume we are dealing with an environment which could only store integers within
the 32-bit signed integer range: [−231,  231 − 1]. If the numerical value is out
of the range of representable values, INT_MAX (231 − 1) or INT_MIN (−231) is
returned.
Example 1:

Input: "42"
Output: 42
Example 2:

Input: "   -42"
Output: -42
```

```
Explanation: The first non-whitespace character is '-', which is the minus sign.
             Then take as many numerical digits as possible, which gets 42.
Example 3:

Input: "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next character is not a
numerical digit.
Example 4:

Input: "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w', which is not a numerical
             digit or a +/- sign. Therefore no valid conversion could be
performed.
Example 5:

Input: "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the range of a 32-bit signed
integer.
             Thefore INT_MIN (-231) is returned.
```

Solution

05/24/2020:

```
class Solution {
public:
  int myAtoi(string str) {
    long long ret = atol(str.c_str());
    ret = ret > INT_MAX ? INT_MAX : ret;
    ret = ret < INT_MIN ? INT_MIN : ret;
    return ret;
  }
};
```

```cpp
class Solution {
public:
  int myAtoi(string str) {
    if (str.empty()) return 0;
    long long ret = 0;
    istringstream iss(str);
    iss >> ret;
    ret = ret < INT_MIN ? INT_MIN : ret;
    ret = ret > INT_MAX ? INT_MAX : ret;
    return ret;
  }
};
```

11. Container With Most Water

Description

```
Given n non-negative integers a1, a2, ..., an , where each represents a point at
coordinate (i, ai). n vertical lines are drawn such that the two endpoints of
line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis
forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.




The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this
case, the max area of water (blue section) the container can contain is 49.



Example:

Input: [1,8,6,2,5,4,8,3,7]
Output: 49
```

Solution

05/24/2020:

```cpp
class Solution {
public:
  int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1, ret = 0;
    while (left < right) {
      ret = max(ret, min(height[left], height[right]) * (right - left));
      height[left] < height[right] ? left += 1 : right -= 1;
    }
    return ret;
  }
};
```

14. Longest Common Prefix

Description

```
Write a function to find the longest common prefix string amongst an array of
strings.

If there is no common prefix, return an empty string "".

Example 1:

Input: ["flower","flow","flight"]
Output: "fl"
Example 2:

Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
Note:

All given inputs are in lowercase letters a-z.
```

Solution

05/24/2020:

```cpp
class Solution {
public:
  string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty() || strs[0].empty()) return "";
    int minLength = strs[0].size();
    for (auto& s : strs) minLength = min(minLength, (int)s.size());
    string ret;
```

```
    for (int i = 0; i < minLength; ++i) {
      char cur = strs[0][i];
      for (int j = 1; j < (int)strs.size(); ++j) {
        if (strs[j][i] != cur) {
          return ret;
        }
      }
      ret += cur;
    }
    return ret;
  }
};
```

```
class Solution {
public:
  string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty() || strs[0].empty()) return "";
    sort(strs.begin(), strs.end(), [](const string& s1, const string& s2) {
      if (s1.size() == s2.size()) return s1 < s2;
      return s1.size() < s2.size();
    });
    string ret;
    for (int k = (int)strs[0].size(); k >= 0; --k) {
      bool isCommonPrefix = true;
      string prefix = strs[0].substr(0, k);
      for (int i = 1; i < (int)strs.size(); ++i) {
        if (prefix != strs[i].substr(0, k)) {
          isCommonPrefix = false;
          break;
        }
      }
      if (isCommonPrefix) {
        ret = prefix;
        break;
      }
    }
    return ret;
  }
};
```

```
struct Node {
  bool isWord;
  vector<Node*> children;
  Node() { isWord = false; children.resize(26, nullptr); }
  ~Node() { for (auto& c : children) delete c; }
};
```

```cpp
class Trie {
private:
  Node* root;

  Node* find(const string& word) {
    Node* cur = root;
    for (auto i = 0; i < (int)word.size(); ++i) {
      if (cur->children[word[i] - 'a'] == nullptr) {
        break;
      }
      cur = cur->children[word[i] - 'a'];
    }
    return cur;
  }

public:
  Trie() { root = new Node(); }

  void insert(const string& word) {
    Node* cur = root;
    for (auto i = 0; i < (int)word.size(); ++i) {
      if (cur->children[word[i] - 'a'] == nullptr) {
        cur->children[word[i] - 'a'] = new Node();
      }
      cur = cur->children[word[i] - 'a'];
    }
    cur->isWord = true;
  }

  bool contains(const string& word) {
    Node* cur = find(word);
    return cur && cur->isWord;
  }

  bool startsWith(const string& prefix) {
    Node* cur = find(prefix);
    return cur;
  }

  string commonPrefix() {
    string ret;
    Node* cur = root;
    while (true) {
      int cnt = 0;
      char ch = 'a';
      bool isUnique = false;
      for (int i = 0; i < 26; ++i) {
        if (cur->children[i] != nullptr) {
          isUnique = true;
```

```
          if (++cnt > 1) {
            isUnique = false;
            break;
          }
          ch = i + 'a';
        }
      }
      if (isUnique) {
        ret += ch;
        cur = cur->children[ch - 'a'];
      } else {
        break;
      }
    }
    return ret;
  }
};

class Solution {
public:
  string longestCommonPrefix(vector<string>& strs) {
    Trie t;
    int minLength = INT_MAX;
    for (auto& s : strs) {
      if (s.empty()) return "";
      t.insert(s);
      minLength = min(minLength, (int)s.size());
    }
    return t.commonPrefix().substr(0, minLength);
  }
};
```

19. Remove Nth Node From End of List

Description

```
Given a linked list, remove the n-th node from the end of list and return its
head.

Example:

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
Note:
```

```
Given n will always be valid.

Follow up:

Could you do this in one pass?
```

Solution

05/23/2020:

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* removeNthFromEnd(ListNode* head, int n) {
    ListNode *pre = new ListNode(0, head), *slow = pre, *fast = pre;
    while (fast->next != nullptr && n-- > 0) fast = fast->next;
    while (fast->next != nullptr) {
      fast = fast->next;
      slow = slow->next;
    }
    slow->next = slow->next->next;
    return pre->next;
  }
};
```

23. Merge k Sorted Lists

Description

```
Merge k sorted linked lists and return it as one sorted list. Analyze and
describe its complexity.

Example:

Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
```

Solution

01/02/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  ListNode* mergeKLists(vector<ListNode*>& lists) {
    vector<int> v;
    for (auto& l : lists) {
      vector<int> tmp = ListNode2vector(l);
      v.insert(v.begin(), tmp.begin(), tmp.end());
    }
    sort(v.begin(), v.end());
    return vector2ListNode(v);
  }

  vector<int> ListNode2vector(ListNode* list) {
    vector<int> v;
    for (; list != nullptr; list = list->next)
      v.push_back(list->val);
    return v;
  }

  ListNode* vector2ListNode(vector<int>& v) {
    ListNode *pre = new ListNode(0), *cur = pre;
    for (auto& n : v) {
      cur->next = new ListNode(n);
```

```
      cur = cur->next;
    }
    return pre->next;
  }
};
```

Mimicking `merge_sort` 01/02/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  ListNode* mergeKLists(vector<ListNode*>& lists) {
    int n = lists.size();
    if (n == 0) return nullptr;
    if (n == 1) return lists[0];
    vector<ListNode*> lists1(lists.begin(), lists.begin() + n / 2);
    vector<ListNode*> lists2(lists.begin() + n / 2, lists.end());
    ListNode* l1 = mergeKLists(lists1);
    ListNode* l2 = mergeKLists(lists2);
    if (l1 == nullptr) return l2;
    ListNode* ret = l1;
    while (l2 != nullptr) {
      if (l1->val > l2->val) swap(l1->val, l2->val);
      while(l1->next && l1->next->val < l2->val) l1 = l1->next;
      ListNode* tmp2 = l2->next;
      l2->next = l1->next;
      l1->next = l2;
      l2 = tmp2;
    }
    return ret;
  }
};
```

24. Swap Nodes in Pairs

Description

```
Given a linked list, swap every two adjacent nodes and return its head.

You may not modify the values in the list's nodes, only nodes itself may be
changed.



Example:

Given 1->2->3->4, you should return the list as 2->1->4->3.
```

Solution

05/22/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* swapPairs(ListNode* head) {
    // pre -> A -> B -> C -> D -> null
    // pre -> B (pre->next = pre->next->next)
    // A -> C (A->next = A->next->next)
    // B -> A (B->next = A):
    // (B -> A -> C -> D -> null)
    ListNode* pre = new ListNode(0, head);
    ListNode* cur = pre;
    while (cur->next && cur->next->next) {
      ListNode* tmp = cur->next;
      cur->next = cur->next->next;
      tmp->next = tmp->next->next;
      cur->next->next = tmp;
      cur = tmp;
    }
    return pre->next;
  }
};
```

26. Remove Duplicates from Sorted Array

Description

```
Given a sorted array nums, remove the duplicates in-place such that each element
appear only once and return the new length.

Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.

Example 1:

Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums
being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums
being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.
Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to
the input array will be known to the caller as well.

Internally you can think of this:

// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/27/2020:

```cpp
class Solution {
public:
  int removeDuplicates(vector<int>& nums) {
    if (nums.size() <= 1) return nums.size();
    int slow = 0, fast = 1, n = nums.size();
    for (; fast < n; ++fast) {
      while (nums[fast] == nums[fast - 1]) if (++fast >= n) break;
      if (fast < n) nums[++slow] = nums[fast];
    }
    return slow + 1;
  }
};
```

27. Remove Element

Description

```
Given an array nums and a value val, remove all instances of that value in-place
and return the new length.

Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond
the new length.

Example 1:

Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums
being 2.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums
containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.
Clarification:
```

```
Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to
the input array will be known to the caller as well.

Internally you can think of this:

// nums is passed in by reference. (i.e., without making a copy)
int len = removeElement(nums, val);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/23/2020:

```
class Solution {
public:
  int removeElement(vector<int>& nums, int val) {
    int right = nums.size() - 1, n = nums.size(), cnt = n;
    for (int i = 0; i < n && i <= right; ++i) {
      while (right >= 0 && nums[right] == val) {
        --right;
        --cnt;
      }
      if (nums[i] == val && i <= right) {
        swap(nums[i], nums[right--]);
        --cnt;
      }
    }
    return cnt;
  }
};
```

33. Search in Rotated Sorted Array

Description

```
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.
```

```
(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index,
otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Solution

04/28/2020:

```
class Solution {
public:
  int search(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size() - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] == target) {
        return mid;
      } else {
        if (nums[mid] < nums[lo]) {
          if (target > nums[lo] && target > nums[hi]) {
            hi = mid - 1;
          } else {
            lo = mid + 1;
          }
        } else {
          if (target > nums[lo] && target > nums[hi]) {
            hi = mid - 1;
          } else {
            lo = mid + 1;
          }
        }
      }
    }
    return -1;
```

```
  }
};
```

34. Find First and Last Position of Element in Sorted Array

Description

```
Given an array of integers nums sorted in ascending order, find the starting and
ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
Example 2:

Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Solution

04/28/2020:

```
class Solution {
public:
  vector<int> searchRange(vector<int>& nums, int target) {
    if (nums.empty()) return {-1, -1};
    int n = nums.size();
    int lower = -1, upper = -1;

    // to find lower_bound
    int lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] >= target) {
        hi = mid;
      } else {
        lo = mid + 1;
      }
    }
    if (nums[lo] < target) ++lo;
```

```
    lower = lo;
    if (lower > hi) return {-1, -1};

    // to find the upper bound
    lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] > target) {
        hi = mid;
      } else {
        lo = mid + 1;
      }
    }
    if (nums[hi] > target) --hi;
    upper = hi;
    if (lower > upper) return {-1, -1};
    return {lower, upper};
  }
};
```

```
class Solution {
public:
  vector<int> searchRange(vector<int>& nums, int target) {
    if (binary_search(nums.begin(), nums.end(), target)) {
      auto p = equal_range(nums.begin(), nums.end(), target);
      return {int(p.first - nums.begin()), int(p.second - nums.begin() - 1)};
    } else {
      return {-1, -1};
    }
  }
};
```

35. Search Insert Position

Description

```
Given a sorted array and a target value, return the index if the target is
found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

Input: [1,3,5,6], 5
Output: 2
```

```
Example 2:

Input: [1,3,5,6], 2
Output: 1
Example 3:

Input: [1,3,5,6], 7
Output: 4
Example 4:

Input: [1,3,5,6], 0
Output: 0
```

Solution

04/23/2020:

```
class Solution {
public:
  int searchInsert(vector<int>& nums, int target) {
    int lo = 0, hi = nums.size() - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] == target) {
        return mid;
      } else if (nums[mid] > target) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return lo;
  }
};
```

38. Count and Say

Description

```
The count-and-say sequence is the sequence of integers with the first five terms
as following:

1.     1
2.     11
3.     21
```

```
4.     1211
5.     111221
1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.

Given an integer n where 1 ≤ n ≤ 30, generate the nth term of the count-and-say
sequence. You can do so recursively, in other words from the previous member
read off the digits, counting the number of digits in groups of the same digit.

Note: Each term of the sequence of integers will be represented as a string.



Example 1:

Input: 1
Output: "1"
Explanation: This is the base case.
Example 2:

Input: 4
Output: "1211"
Explanation: For n = 3 the term was "21" in which we have two groups "2" and
"1", "2" can be read as "12" which means frequency = 1 and value = 2, the same
way "1" is read as "11", so the answer is the concatenation of "12" and "11"
which is "1211".
```

Solution

05/24/2020:

```
class Solution {
public:
  string countAndSay(int n) {
    if (n == 1) return "1";
    string s = countAndSay(n - 1);
    string ret;
    int cnt = 1;
    char cur = s[0];
    for (int i = 1; i < (int)s.size(); ++i) {
      if (s[i] == cur) {
        ++cnt;
      } else {
        ret += to_string(cnt) + cur;
        cur = s[i];
        cnt = 1;
      }
```

```
    }
    if (cnt > 0) {
      ret += to_string(cnt) + cur;
    }
    return ret;
  }
};
```

41. First Missing Positive

Description

```
Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

Input: [1,2,0]
Output: 3
Example 2:

Input: [3,4,-1,1]
Output: 2
Example 3:

Input: [7,8,9,11,12]
Output: 1
Note:

Your algorithm should run in O(n) time and uses constant extra space.
```

Solution

05/24/2020:

```
class Solution {
public:
  int firstMissingPositive(vector<int>& nums) {
    unordered_set<int> seen;
    for (auto& i : nums) {
      if (i > 0) {
        seen.insert(i);
      }
    }
    int i = 1, n = nums.size();
    while (i <= n) {
```

```
      if (seen.count(i) == 0) return i;
      ++i;
    }
    return i;
  }
};
```

46. Permutations

Description

```
Given a collection of distinct integers, return all possible permutations.

Example:

Input: [1,2,3]
Output:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

Solution

05/26/2020 (Using `next_permutation`):

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> ret{nums};
    while (next_permutation(nums.begin(), nums.end())) ret.push_back(nums);
    return ret;
  }
};
```

05/26/2020 (Backtracking):

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
```

```
    int n = nums.size(),numOfChosen = 0;
    vector<bool> chosen(n, false);
    vector<vector<int>> ret;
    vector<int> permutation;
    backtrack(nums, chosen, numOfChosen, permutation, ret);
    return ret;
  }

  void backtrack(vector<int>& nums, vector<bool>& chosen, int numOfChosen,
vector<int>& permutation, vector<vector<int>>& ret) {
    if (numOfChosen == (int)nums.size()) {
      ret.push_back(permutation);
      return;
    }
    for (int i = 0; i < (int)nums.size(); ++i) {
      if (chosen[i] == true) continue;
      chosen[i] = true;
      permutation.push_back(nums[i]);
      backtrack(nums, chosen, numOfChosen + 1, permutation, ret);
      chosen[i] = false;
      permutation.pop_back();
    }
  }
};
```

05/05/2020:

```
class Solution {
public:
  vector<vector<int>> permute(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return {nums};
    vector<vector<int>> ret;
    for (int i = 0; i < n; ++i) {
      int cur = nums[i];
      swap(nums[i], nums[n - 1]);
      nums.pop_back();
      vector<vector<int>> sub = permute(nums);
      for (auto& s : sub) {
        s.push_back(cur);
        ret.push_back(s);
      }
      nums.push_back(cur);
      swap(nums[i], nums[n - 1]);
    }
    return ret;
  }
};
```

50. Pow(x, n)

Description

```
Implement pow(x, n), which calculates x raised to the power n (xn).

Example 1:

Input: 2.00000, 10
Output: 1024.00000
Example 2:

Input: 2.10000, 3
Output: 9.26100
Example 3:

Input: 2.00000, -2
Output: 0.25000
Explanation: 2-2 = 1/22 = 1/4 = 0.25
Note:

-100.0 < x < 100.0
n is a 32-bit signed integer, within the range [-231, 231 - 1]
```

Solution

04/23/2020:

- Discussion
- Binary Exponentiation: given a positive power `n`, e.g., `n = 22 (binary: 10110) = 16 (binary: 10000) + 4 (binary: 100)+ 2 (binary: 10)`, then `pow(x, n) = x^n = x^22 = x^(16 + 4 + 2) = x^16 * x^4 * x^2`. For negative powers, we just flip the sign first, and once we calculate the value and return its reciprocal.
- Time complexity: O(n);
- Space complexity: O(1).

```cpp
class Solution {
public:
  double myPow(double x, int n) {
    long double ret = 1.0, pow_x = x;
    for (long m = n >= 0 ? (long)n : -1 * (long)n; m != 0; m >>= 1) {
      if (m & 1) ret *= pow_x;
      pow_x = pow_x * pow_x;
    }
    return n >= 0 ? ret : 1.0L / ret;
  }
};
```

51. N-Queens

Description

```
The n-queens puzzle is the problem of placing n queens on an n×n chessboard such
that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens'
placement, where 'Q' and '.' both indicate a queen and an empty space
respectively.

Example:

Input: 4
Output: [
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown
above.
```

Solution

05/27/2020:

```
class Solution {
public:
  unordered_set<string> seen;
  vector<vector<string>> solveNQueens(int n) {
    vector<vector<string>> ret;
    vector<string> board(n, string(n, '.'));
    vector<bool> col(n, false);
    vector<bool> diag1(2 * n - 1, false);
    vector<bool> diag2(2 * n - 1, false);
    backtrack(0, n, board, col, diag1, diag2, ret);
    return ret;
  }

  void backtrack(int r, int n, vector<string>& board, vector<bool>& col,
vector<bool>& diag1, vector<bool>& diag2, vector<vector<string>>& ret) {
    if (r == n) {
      ret.push_back(board);
      return;
    }
    for (int c = 0; c < n; ++c) {
      if (!col[c] && !diag1[r + c] && !diag2[r - c + n - 1]) {
        board[r][c] = 'Q';
        col[c] = diag1[r + c] = diag2[r -c + n - 1] = true;
        backtrack(r + 1, n, board, col, diag1, diag2, ret);
        col[c] = diag1[r + c] = diag2[r -c + n - 1] = false;
        board[r][c] = '.';
      }
    }
  }
};
```

53. Maximum Subarray

Description

```
Given an integer array nums, find the contiguous subarray (containing at least
one number) which has the largest sum and return its sum.

Example:

Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
Follow up:

If you have figured out the O(n) solution, try coding another solution using the
divide and conquer approach, which is more subtle.
```

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  int maxSubArray(vector<int>& nums) {
    int n = nums.size(), max_sum = nums[0];
    for (int i = 1; i < n; ++i) {
      if (nums[i - 1] > 0) nums[i] += nums[i - 1];
      max_sum = max(max_sum, nums[i]);
    }
    return max_sum;
  }
};
```

54. Spiral Matrix

Description

```
54. Spiral Matrix
Medium

1984

526

Add to List

Share
Given a matrix of m x n elements (m rows, n columns), return all elements of the
matrix in spiral order.
```

```
Example 1:

Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
Output: [1,2,3,6,9,8,7,4,5]
Example 2:

Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

Solution

04/23/2020:

```
class Solution {
public:
  vector<int> spiralOrder(vector<vector<int>>& matrix) {
    int m = matrix.size();
    if (m == 0) return {};
    int n = matrix[0].size();
    if (n == 0) return {};
    vector<vector<int>> dir{ {0, 1}, {1, 0}, {0, -1}, {-1, 0} };
    vector<int> ret;
    int N = m * n, d = 0, i = 0, j = -1;
    while (ret.size() < N) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      while (ni < 0 || ni >= m || nj < 0 || nj >= n || matrix[ni][nj] ==
INT_MIN) {
        d = (d + 1) % 4;
        ni = i + dir[d][0], nj = j + dir[d][1];
      }
      ret.push_back(matrix[ni][nj]);
      matrix[ni][nj] = INT_MIN;
      i = ni;
      j = nj;
    }
    return ret;
  }
```

```
};
```

55. Jump Game

Description

```
Given an array of non-negative integers, you are initially positioned at the
first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

Input: [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:

Input: [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum
             jump length is 0, which makes it impossible to reach the last
index.
```

Solution

04/24/2020:

```
class Solution {
public:
  bool canJump(vector<int>& nums) {
    int ret = 0, n = nums.size();
    for (int i = 0; i < n; ++i) {
      if (i <= ret) {
        ret = max(i + nums[i], ret);
      } else {
        break;
      }
    }
    return ret >= n - 1;
  }
};
```

59. Spiral Matrix II

Description

```
Given a positive integer n, generate a square matrix filled with elements from 1
to n2 in spiral order.

Example:

Input: 3
Output:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

Solution

Discussion:

```
class Solution {
public:
  vector<vector<int>> generateMatrix(int n) {
    if (n <= 0) return {};
    vector<vector<int>> dir{ {0, 1}, {1, 0}, {0, -1}, {-1, 0} };
    vector<vector<int>> matrix(n, vector<int>(n, 0));
    int i = 0, j = -1, d = 0, N = n * n, cnt = 0;
    while (cnt < N) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      while (ni < 0 || ni >= n || nj < 0 || nj >= n || matrix[ni][nj] != 0) {
        d = (d + 1) % 4;
        ni = i + dir[d][0];
        nj = j + dir[d][1];
      }
      i = ni;
      j = nj;
      matrix[i][j] = ++cnt;
    }
    return matrix;
  }
};
```

60. Permutation Sequence

Description

```
The set [1,2,3,...,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following
sequence for n = 3:

"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.

Note:

Given n will be between 1 and 9 inclusive.
Given k will be between 1 and n! inclusive.
Example 1:

Input: n = 3, k = 3
Output: "213"
Example 2:

Input: n = 4, k = 9
Output: "2314"
```

Solution

05/26/2020:

```
class Solution {
public:
  string getPermutation(int n, int k) {
    string s;
    for (int i = 0; i < n; ++i) s += to_string(i + 1);
    while (--k > 0 && next_permutation(s.begin(), s.end()));
    return s;
  }
};
```

66. Plus One

Description

```
Given a non-empty array of digits representing a non-negative integer, plus one
to the integer.

The digits are stored such that the most significant digit is at the head of the
list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number
0 itself.

Example 1:

Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Example 2:

Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

Solution

05/23/2020:

```
class Solution {
public:
  vector<int> plusOne(vector<int>& digits) {
    if (digits.empty()) return {1};
    int carry = 1, n = digits.size();;
    for (int i = n - 1; i >= 0; --i) {
      carry += digits[i];
      digits[i] = carry % 10;
      carry /= 10;
    }
    if (carry > 0) digits.insert(digits.begin(), carry);
    return digits;
  }
};
```

67. Add Binary

Description

```
Given two binary strings, return their sum (also a binary string).
```

```
The input strings are both non-empty and contains only characters 1 or 0.

Example 1:

Input: a = "11", b = "1"
Output: "100"
Example 2:

Input: a = "1010", b = "1011"
Output: "10101"


Constraints:

Each string consists only of '0' or '1' characters.
1 <= a.length, b.length <= 10^4
Each string is either "0" or doesn't contain any leading zero.
```

Solution

05/12/2020:

```cpp
class Solution {
public:
  string addBinary(string a, string b) {
    int i = a.size() - 1, j = b.size() - 1;
    int carry = 0;
    string ret;
    for (; i >= 0 || j >= 0; --i, --j) {
      if (i >= 0) carry += a[i] - '0';
      if (j >= 0) carry += b[j] - '0';
      ret += (carry & 1) + '0';
      carry >>= 1;
    }
    if (carry > 0) ret += carry + '0';
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

69. Sqrt(x)

Description

```
Implement int sqrt(int x).
```

```
Compute and return the square root of x, where x is guaranteed to be a non-
negative integer.

Since the return type is an integer, the decimal digits are truncated and only
the integer part of the result is returned.

Example 1:

Input: 4
Output: 2
Example 2:

Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since
             the decimal part is truncated, 2 is returned.
```

Solution

04/28/2020:

```cpp
class Solution {
public:
  int mySqrt(int x) {
    int lo = 0, hi = x;
    while (lo <= hi) {
      long long mid = lo + (hi - lo) / 2;
      long long sq = mid * mid;
      if (sq == x) {
        return mid;
      } else if (sq > x) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return hi;
  }
};
```

70. Climbing Stairs

Description

```
You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you
climb to the top?

Note: Given n will be a positive integer.

Example 1:

Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
Example 2:

Input: 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

Solution

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
  int climbStairs(int n) {
    int s1 = 1, s2 = 2;
    for (int i = 2; i < n; ++i) {
      s1 = s1 + s2;
      swap(s1, s2);
    }
    return n >= 2 ? s2 : s1;
  }
};
```

72. Edit Distance

Description

```
Given two words word1 and word2, find the minimum number of operations required
to convert word1 to word2.
```

```
You have the following 3 operations permitted on a word:

Insert a character
Delete a character
Replace a character
Example 1:

Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
Example 2:

Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Solution

05/31/2020:

```
class Solution {
public:
  int minDistance(string word1, string word2) {
    int m = word1.size(), n = word2.size();
    if (m == 0) return n;
    if (n == 0) return m;
    const int INF = 1e9 + 5;
    vector<vector<int>> dp(m + 1, vector(n + 1, INF));
    dp[0][0] = 0;
    // dp[i][j]: the edit distance between word1[0..i] and word2[0..j]
    // dp[i][j] = dp[i - 1][j - 1]      if word1[i] == word2[j]: no operations
needed
    // dp[i][j] = min(                   if word1[i] != word2[j]
    //   dp[i - 1][j - 1] + 1,  replace word1[i] by word2[j]
    //   dp[i - 1][j] + 1,      delete character word1[i]
    //   dp[i][j - 1] + 1       delete character word2[j]
    // )
    for (int i = 1; i <= m; ++i) dp[i][0] = dp[i - 1][0] + 1;
    for (int j = 1; j <= n; ++j) dp[0][j] = dp[0][j - 1] + 1;
```

```
    for (int i = 1; i <= m; ++i) {
      for (int j = 1; j <= n; ++j) {
        if (word1[i - 1] == word2[j - 1]) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = min(dp[i - 1][j], min(dp[i][j - 1], dp[i - 1][j - 1])) + 1;
        }
      }
    }
    return dp[m][n];
  }
};
```

```
class Solution {
public:
  int minDistance(string word1, string word2) {
    int m = word1.size(), n = word2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, INT_MAX));
    for (int i = 0; i <= m; ++i) {
      for (int j = 0; j <= n; ++j) {
        if (i == 0) {
          dp[i][j] = j;
        } else if (j == 0) {
          dp[i][j] = i;
        } else if (word1[i - 1] == word2[j - 1]) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;
        }
      }
    }
    return dp[m][n];
  }
};
```

74. Search a 2D Matrix

Description

```
Write an efficient algorithm that searches for a value in an m x n matrix. This
matrix has the following properties:

Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous
row.
```

```
Example 1:

Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
Output: true
Example 2:

Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
Output: false
```

Solution

05/23/2020:

```
class Solution {
public:
  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) return false;
    int m = matrix.size(), n = matrix[0].size();
    int lo = 0, hi = m * n - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      int i = mid / n, j = mid % n;
      if (matrix[i][j] == target) {
        return true;
      } else if (matrix[i][j] < target) {
        lo = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    return false;
  }
};
```

75. Sort Colors

Description

```
Given an array with n objects colored red, white or blue, sort them in-place so
that objects of the same color are adjacent, with the colors in the order red,
white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white,
and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example:

Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite
array with total number of 0's, then 1's and followed by 2's.
Could you come up with a one-pass algorithm using only constant space?
```

Solution

05/12/2020:

```
class Solution {
public:
  void sortColors(vector<int>& nums) {
    sort(nums.begin(), nums.end());
  }
};
```

77. Combinations

Description

```
Given two integers n and k, return all possible combinations of k numbers out of
1 ... n.

Example:
```

```
Input: n = 4, k = 2
Output:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

Solution

05/27/2020:

```
class Solution {
public:
  vector<vector<int>> combine(int n, int k) {
    vector<int> combination;
    vector<vector<int>> ret;
    vector<bool> chosen(n + 1, false);
    backtrack(1, n, k, chosen, combination, ret);
    return ret;
  }

  void backtrack(int j, int n, int k, vector<bool>& chosen, vector<int>&
combination, vector<vector<int>>& ret) {
    if (combination.size() == k) ret.push_back(combination);
    for (int i = j; i <= n; ++i) {
      if (chosen[i] == true || (!combination.empty() && i < combination.back()))
continue;
      chosen[i] = true;
      combination.push_back(i);
      backtrack(j + 1, n, k, chosen, combination, ret);
      chosen[i] = false;
      combination.pop_back();
    }
  }
};
```

78. Subsets

Description

```
Given a set of distinct integers, nums, return all possible subsets (the power
set).

Note: The solution set must not contain duplicate subsets.

Example:

Input: nums = [1,2,3]
Output:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Solution

05/26/2020:

```
class Solution {
public:
  vector<int> subset;
  vector<vector<int>> s;
  int n;
  vector<vector<int>> subsets(vector<int>& nums) {
    n = nums.size();
    search(0);
    vector<vector<int>> ret;
    for (auto& ss : s) {
      vector<int> tmp;
      for (auto& i : ss) {
        tmp.push_back(nums[i]);
      }
      ret.push_back(tmp);
    }
    return ret;
  }

  void search(int k) {
    if (k == n) {
      s.push_back(subset);
    } else {
      subset.push_back(k);
```

```
      search(k + 1);
      subset.pop_back();
      search(k + 1);
    }
  }
};
```

```
class Solution {
public:
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret;
    vector<int> subset;
    backtrack(0, nums, subset, ret);
    return ret;
  }

  void backtrack(int k, vector<int>& nums, vector<int>& subset,
vector<vector<int>>& ret) {
    int n = nums.size();
    if (k == n) {
      ret.push_back(subset);
      return;
    }
    subset.push_back(nums[k]);
    backtrack(k + 1, nums, subset, ret);
    subset.pop_back();
    backtrack(k + 1, nums, subset, ret);
  }
};
```

```
class Solution {
public:
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret{ {} };
    for (auto& n : nums) {
      int sz = ret.size();
      for (int i = 0; i < sz; ++i) {
        ret.push_back(ret[i]);
        ret.back().push_back(n);
      }
    }
    return ret;
  }
};
```

```
public:
```

```
  vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ret;
    vector<int> subset;
    backtrack(0, nums, subset, ret);
    return ret;
  }

  void backtrack(int k, vector<int>& nums, vector<int>& subset,
vector<vector<int>>& ret) {
    ret.push_back(subset);
    for (int i = k; i < (int)nums.size(); ++i) {
      subset.push_back(nums[i]);
      backtrack(i + 1, nums, subset, ret);
      subset.pop_back();
    }
  }
};
```

79. Word Search

Description

```
Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where
"adjacent" cells are those horizontally or vertically neighboring. The same
letter cell may not be used more than once.

Example:

board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]

Given word = "ABCCED", return true.
Given word = "SEE", return true.
Given word = "ABCB", return false.


Constraints:

board and word consists only of lowercase and uppercase English letters.
1 <= board.length <= 200
```

```
1 <= board[i].length <= 200
1 <= word.length <= 10^3
```

Solution

05/27/2020:

```
class Solution {
public:
  int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
  bool exist(vector<vector<char>>& board, string word) {
    if (word.empty()) return true;
    if (board.empty() || board[0].empty()) return false;
    int m = board.size(), n = board[0].size(), k = word.size();
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        if (backtrack(board, word, i, j, 0))
          return true;
    return false;
  }

  bool backtrack(vector<vector<char>>& board, string word, int i, int j, int k)
{
    int m = board.size(), n = board[0].size(), sz = word.size();
    if (board[i][j] != word[k]) return false;
    if (k == sz - 1) return true;
    board[i][j] = '#';
    for (int d = 0; d < 4; ++d) {
      int ni = i + dir[d][0], nj = j + dir[d][1];
      if (ni < 0 || ni >= m || nj < 0 || nj >= n) continue;
      if (backtrack(board, word, ni, nj, k + 1)) return true;
    }
    board[i][j] = word[k];
    return false;
  }
};
```

80. Remove Duplicates from Sorted Array II

Description

```
Given a sorted array nums, remove the duplicates in-place such that duplicates
appeared at most twice and return the new length.
```

```
Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.

Example 1:

Given nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums
being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.
Example 2:

Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of nums
being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.
Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to
the input array will be known to the caller as well.

Internally you can think of this:

// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Solution

05/27/2020:

```
class Solution {
public:
  int removeDuplicates(vector<int>& nums) {
    if (nums.size() <= 2) return nums.size();
    int slow = 0, fast = 1, n = nums.size(), cnt = 1;
    for (; fast < n; ++fast) {
      while (fast < n && nums[fast] == nums[fast - 1]) {
```

```
        if (cnt <= 1) {
          ++cnt;
          nums[++slow] = nums[fast];
        }
        ++fast;
      }
      if (fast < n && nums[fast] != nums[slow]) {
        cnt = 1;
        nums[++slow] = nums[fast];
      }
    }
    return slow + 1;
  }
};
```

81. Search in Rotated Sorted Array II

Description

```
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.

(i.e., [0,0,1,2,2,5,6] might become [2,5,6,0,0,1,2]).

You are given a target value to search. If found in the array return true,
otherwise return false.

Example 1:

Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
Example 2:

Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
Follow up:

This is a follow up problem to Search in Rotated Sorted Array, where nums may
contain duplicates.
Would this affect the run-time complexity? How and why?
```

Solution

05/27/2020:

```
class Solution {
public:
  bool search(vector<int>& nums, int target) {
    sort(nums.begin(), nums.end());
    return binary_search(nums.begin(), nums.end(), target);
  }
};
```

83. Remove Duplicates from Sorted List

Description

```
Given a sorted linked list, delete all duplicates such that each element appear
only once.

Example 1:

Input: 1->1->2
Output: 1->2
Example 2:

Input: 1->1->2->3->3
Output: 1->2->3
```

Solution

05/27/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* deleteDuplicates(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return head;
    ListNode *slow = head, *fast = head->next;
    while (fast->next) {
      while (fast && fast->val == slow->val && fast->next) fast = fast->next;
```

```
      if (fast && fast->val != slow->val) {
        slow->next = fast;
        slow = slow->next;
      }
    }
    if (slow->val == fast->val) {
      slow->next = fast->next;
    }
    return head;
  }
};
```

84. Largest Rectangle in Histogram

Description

```
Given n non-negative integers representing the histogram's bar height where the
width of each bar is 1, find the area of largest rectangle in the histogram.

Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].

The largest rectangle is shown in the shaded area, which has area = 10 unit.
```

Solution

01/14/2020 (Stack):

```
class Solution {
public:
  // Approach 1: Brute Force
  // int largestRectangleArea(vector<int>& heights) {
  //   int max_area = INT_MIN;
  //   for (int i = 0; i < heights.size(); ++i) {
  //     for (int j = i; j < heights.size(); ++j) {
  //       int min_height = *min_element(heights.begin() + i, heights.begin() +
j + 1);
  //       max_area = max(max_area, min_height * (j - i + 1));
  //     }
  //   }
  //   return max_area;
  // }

  // Approach 2: Better Brute Force
  // int largestRectangleArea(vector<int>& heights) {
  //   int n = heights.size(), max_area = n > 0 ? heights[0] : 0;
```

```
  //   for (int i = 0; i < heights.size(); ++i) {
  //     int l = i, r = i;
  //     for (; l >= 0 && heights[l] >= heights[i]; --l);
  //     for (; r < heights.size() && heights[r] >= heights[i]; ++r);
  //     max_area = max(max_area, heights[i] * (--r - ++l + 1));
  //   }
  //   return max_area;
  // }

  // Approach 3: Divide and Conquer
  // int maxRectangleArea(vector<int>& heights, int left, int right) {
  //   if (left > right) return 0;
  //   int min_height_index = left;
  //   for (int i = left; i <= right; ++i) {
  //     if (heights[min_height_index] > heights[i])
  //       min_height_index = i;
  //   }
  //   return max(max(maxRectangleArea(heights, left, min_height_index - 1),
maxRectangleArea(heights, min_height_index + 1, right)),
heights[min_height_index] * (right - left + 1));
  // }
  //
  // int largestRectangleArea(vector<int>& heights) {
  //   return maxRectangleArea(heights, 0, heights.size() - 1);
  // }

  // Approach 4: Stack
  int largestRectangleArea(vector<int>& heights) {
    stack<int> s;
    s.push(-1);
    int max_area = 0;
    for (int i = 0; i < heights.size(); ++i) {
      while (s.top() != -1 && heights[s.top()] >= heights[i]) {
        int k = s.top();
        s.pop();
        max_area = max(max_area, heights[k] * (i - s.top() - 1));
      }
      s.push(i);
    }
    while (s.top() != -1) {
      int k = s.top();
      s.pop();
      max_area = max(max_area, heights[k] * ((int)heights.size() - s.top() -
1));
    }
    return max_area;
  }
};
```

88. Merge Sorted Array

Description

```
Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one
sorted array.

Note:

The number of elements initialized in nums1 and nums2 are m and n respectively.
You may assume that nums1 has enough space (size that is greater or equal to m +
n) to hold additional elements from nums2.
Example:

Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6],       n = 3

Output: [1,2,2,3,5,6]
```

Solution

05/23/2020:

```
class Solution {
public:
  void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    copy(nums2.begin(), nums2.end(), nums1.begin() + m);
    inplace_merge(nums1.begin(), nums1.begin() + m, nums1.end());
  }
};
```

94. Binary Tree Inorder Traversal

Description

```
Given a binary tree, return the inorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [1,3,2]
Follow up: Recursive solution is trivial, could you do it iteratively?
```

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    stack<TreeNode*> st;
    st.push(root);
    unordered_set<TreeNode*> visited;
    vector<int> ret;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->left && visited.count(cur->left) == 0) {
        if (cur->right) st.push(cur->right);
        st.push(cur);
        st.push(cur->left);
      } else if (!cur->left) {
        if (cur->right) st.push(cur->right);
        ret.push_back(cur->val);
        visited.insert(cur);
      } else {
        ret.push_back(cur->val);
        visited.insert(cur);
```

```
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = inorderTraversal(root->left);
    ret.push_back(root->val);
    vector<int> right = inorderTraversal(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> inorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret;
    TreeNode* cur = root;
    stack<TreeNode*> st;
    while (cur || !st.empty()) {
      while (cur) {
        st.push(cur);
        cur = cur->left;
      }
      cur = st.top(); st.pop();
      ret.push_back(cur->val);
      cur = cur->right;
    }
    return ret;
  }
};
```

96. Unique Binary Search Trees

Description

```
Given n, how many structurally unique BST's (binary search trees) that store
values 1 ... n?
```

```
Example:

Input: 3
Output: 5
Explanation:
Given n = 3, there are a total of 5 unique BST's:

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

Solution

06/24/2020:

```
class Solution {
public:
  int numTrees(int n) {
    long C = 1;
    for (int i = 0; i < n; ++i) C = C * 2 * (2 * i + 1) / (i + 2);
    return C;
  }
};
```

101. Symmetric Tree

Description

```
Given a binary tree, check whether it is a mirror of itself (ie, symmetric
around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric:

    1
   / \
  2   2
 / \ / \
3  4 4  3


But the following [1,2,2,null,3,null,3] is not:
```

```
    1
   / \
  2   2
   \   \
   3    3
```

Follow up: Solve it both recursively and iteratively.

Solution

Discussion: This problem is very similar to https://leetcode.com/problems/same-tree/. Non-recursive DFS:

```
class Solution {
public:
  bool isSymmetric(TreeNode* root) {
    stack<pair<TreeNode*, TreeNode*>> st;
    st.emplace(root, root);
    while (!st.empty()) {
      pair<TreeNode*, TreeNode*> cur = st.top(); st.pop();
      if (cur.first == nullptr && cur.second == nullptr) continue;
      if (cur.first != nullptr && cur.second == nullptr) return false;
      if (cur.first == nullptr && cur.second != nullptr) return false;
      if (cur.first->val != cur.second->val) return false;
      st.emplace(cur.first->left, cur.second->right);
      st.emplace(cur.first->right, cur.second->left);
    }
    return true;
  }
};
```

Recursive DFS:

```cpp
class Solution {
public:
  bool isSymmetric(TreeNode* root) {
    if (!root) return true;
    return isSubtreeSymmetric(root->left, root->right);
  }

  bool isSubtreeSymmetric(TreeNode* left, TreeNode* right) {
    if (!left && !right) return true;
    if ((!left && right) || (left && !right)) return false;
    return left->val == right->val && isSubtreeSymmetric(left->left, right->right) && isSubtreeSymmetric(left->right, right->left);
  }
};
```

102. Binary Tree Level Order Traversal

Description

```
Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree [3,9,20,null,null,15,7],

    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:

[
  [3],
  [9,20],
  [15,7]
]
```

Solution

04/26/2020:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<vector<int>> levelOrder(TreeNode* root) {
    if (root == nullptr) return {};
    vector<vector<int>> ret;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      for (int i = 0; i < n; ++i) {
        TreeNode* cur = q.front(); q.pop();
        level.push_back(cur->val);
        if (cur->left) q.push(cur->left);
        if (cur->right) q.push(cur->right);
      }
      ret.push_back(level);
    }
    return ret;
  }
};
```

103. Binary Tree Zigzag Level Order Traversal

Description

```
Given a binary tree, return the zigzag level order traversal of its nodes'
values. (ie, from left to right, then right to left for the next level and
alternate between).

For example:
Given binary tree [3,9,20,null,null,15,7],

    3
   / \
  9  20
    /  \
   15   7
return its zigzag level order traversal as:
```

```
[
  [3],
  [20,9],
  [15,7]
]
```

Solution

Discussion 04/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ret;
    if (root == nullptr) return ret;
    queue<pair<TreeNode*, int>> q;
    q.emplace(root, 0);
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      int depth = q.front().second;
      for (int i = 0; i < n; ++i) {
        pair<TreeNode*, int> cur = q.front(); q.pop();
        level.push_back(cur.first->val);
        if (cur.first->left) q.emplace(cur.first->left, cur.second + 1);
        if (cur.first->right) q.emplace(cur.first->right, cur.second + 1);
      }
      if (depth % 2 != 0) reverse(level.begin(), level.end());
      ret.push_back(level);
    }
    return ret;
  }
};
```

Better:

```
class Solution {
public:
```

```
  vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    vector<vector<int>> ret;
    if (root == nullptr) return ret;
    queue<TreeNode*> q;
    q.push(root);
    for (int depth = 0; !q.empty(); ++depth) {
      int n = q.size();
      vector<int> level;
      for (int j = 0; j < n; ++j) {
        TreeNode* cur = q.front(); q.pop();
        level.push_back(cur->val);
        if (cur->left) q.push(cur->left);
        if (cur->right) q.push(cur->right);
      }
      if (depth % 2 != 0) reverse(level.begin(), level.end());
      ret.push_back(level);
    }
    return ret;
  }
};
```

105. Construct Binary Tree from Preorder and Inorder Traversal

Description

```
Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

For example, given

preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
Return the following binary tree:

    3
   / \
  9  20
    /  \
   15   7
```

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder, int p_start
= 0, int p_stop = INT_MAX, int i_start = 0, int i_stop = INT_MAX) {
      if (p_stop == INT_MAX) p_stop = i_stop = inorder.size() - 1;
      if (i_start > i_stop) return nullptr;
      int i = i_start;
      for (; i <= i_stop; ++i)
        if (inorder[i] == preorder[p_start])
          break;
      TreeNode* root = new TreeNode(preorder[p_start]);
      root->left = buildTree(preorder, inorder, p_start + 1, p_start + 1 + (i -
1) - i_start, i_start, i - 1);
      root->right = buildTree(preorder, inorder, p_start + 1 + (i - 1) - i_start
+ 1, p_stop, i + 1, i_stop);
      return root;
    }
};
```

109. Convert Sorted List to Binary Search Tree

Description

```
Given a singly linked list where elements are sorted in ascending order, convert
it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in
which the depth of the two subtrees of every node never differ by more than 1.

Example:
```

```
Given the sorted linked list: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following
height balanced BST:

      0
     / \
   -3   9
   /   /
 -10  5
```

Solution

05/27/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  TreeNode* sortedListToBST(ListNode* head) {
    if (head == nullptr) return nullptr;
    if (head->next == nullptr) return new TreeNode(head->val);
    ListNode *slow = head, *fast = head, *preSlow = head;
    while (fast && fast->next) {
      preSlow = slow;
      slow = slow->next;
      fast = fast->next->next;
    }
```

```
    preSlow->next = nullptr;
    TreeNode* root = new TreeNode(slow->val);
    root->left = sortedListToBST(head);
    root->right = sortedListToBST(slow->next);
    return root;
  }
};
```

110. Balanced Binary Tree

Description

```
Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

a binary tree in which the left and right subtrees of every node differ in
height by no more than 1.



Example 1:

Given the following tree [3,9,20,null,null,15,7]:

    3
   / \
  9  20
    /  \
   15   7
Return true.

Example 2:

Given the following tree [1,2,2,3,3,null,null,4,4]:

       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
Return false.
```

Solution

05/27/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  bool isBalanced(TreeNode* root) {
    if (root == nullptr) return true;
    return abs(depth(root->left) - depth(root->right)) <= 1 && isBalanced(root-
>left) && isBalanced(root->right);
  }

  int depth(TreeNode* root) {
    if (root == nullptr) return 0;
    return 1 + max(depth(root->left), depth(root->right));
  }
};
```

114. Flatten Binary Tree to Linked List

Description

```
Given a binary tree, flatten it to a linked list in-place.

For example, given the following tree:

    1
   / \
  2   5
 / \   \
3   4   6
The flattened tree should look like:

1
```

```
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

Solution

05/27/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  void flatten(TreeNode* root) {
    if (root == nullptr) return;
    flatten(root->left);
    flatten(root->right);
    if (root->left) {
      TreeNode* cur = root->left;
      while (cur->right) cur = cur->right;
      cur->right = root->right;
      root->right = root->left;
      root->left = nullptr;
    }
  }
};
```

```
class Solution {
public:
  void flatten(TreeNode* root) {
    if (root == nullptr) return;
```

```
    TreeNode *list = new TreeNode(0), *l = list;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->right) st.push(cur->right);
      if (cur->left) st.push(cur->left);
      l->right = cur;
      cur->left = nullptr;
      l = l->right;
    }
    root = list->right;
  }
};
```

118. Pascal's Triangle

Description

```
Given a non-negative integer numRows, generate the first numRows of Pascal's
triangle.


In Pascal's triangle, each number is the sum of the two numbers directly above
it.

Example:

Input: 5
Output:
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

Solution

05/27/2020:

```cpp
class Solution {
public:
  vector<vector<int>> generate(int numRows) {
    if (numRows < 0) return {};
    vector<vector<int>> ret(numRows);
    for (int i = 0; i < numRows; ++i) {
      ret[i].resize(i + 1, 1);
      for (int j = 1; j < i; ++j)
        ret[i][j] = ret[i - 1][j] + ret[i - 1][j - 1];
    }
    return ret;
  }
};
```

119. Pascal's Triangle II

Description

```
Given a non-negative index k where k ≤ 33, return the kth index row of the
Pascal's triangle.

Note that the row index starts from 0.


In Pascal's triangle, each number is the sum of the two numbers directly above
it.

Example:

Input: 3
Output: [1,3,3,1]
Follow up:

Could you optimize your algorithm to use only O(k) extra space?
```

Solution

05/27/2020:

```cpp
class Solution {
public:
  vector<int> getRow(int rowIndex) {
    if (rowIndex <= 1) return vector<int>(rowIndex + 1, 1);
    vector<int> ret(2, 1);
    for (int i = 2; i <= rowIndex; ++i) {
```

```
      vector<int> nextRow(i + 1, 1);
      for (int j = 1; j < i; ++j)
        nextRow[j] = ret[j - 1] + ret[j];
      ret = nextRow;
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> getRow(int rowIndex) {
    int n = rowIndex + 1;
    vector<vector<int>> ret(n);
    for (int i = 0; i < n; ++i) {
      ret[i].resize(i + 1, 1);
      for (int j = 1; j < i; ++j) {
        ret[i][j] = ret[i - 1][j - 1] + ret[i - 1][j];
      }
    }
    return ret.back();
  }
};
```

121. Best Time to Buy and Sell Stock

Description

```
Say you have an array for which the ith element is the price of a given stock on
day i.

If you were only permitted to complete at most one transaction (i.e., buy one
and sell one share of the stock), design an algorithm to find the maximum
profit.

Note that you cannot sell a stock before you buy one.

Example 1:

Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-
1 = 5.
             Not 7-1 = 6, as selling price needs to be larger than buying price.
Example 2:
```

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

Solution

01/13/2020 (Dynamic Programming):

```
class Solution {
public:
  int maxProfit(vector<int>& prices) {
    int n = prices.size();
    if (n <= 1) return 0;
    vector<int> diff(n - 1, 0);
    for (int i = 0; i < n - 1; ++i) {
      diff[i] = prices[i + 1] - prices[i];
    }
    int max_sum = max(0, diff[0]);
    for (int i = 1; i < n - 1; ++i) {
      if (diff[i - 1] > 0) diff[i] += diff[i - 1];
      max_sum = max(diff[i], max_sum);
    }
    return max_sum;
  }
};
```

129. Sum Root to Leaf Numbers

Description

```
Given a binary tree containing digits from 0-9 only, each root-to-leaf path
could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

Example:

Input: [1,2,3]
    1
   / \
```

```
  2   3
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
Example 2:

Input: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Solution

06/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  int sumNumbers(TreeNode* root) {
    long long ret = 0, num = 0;
    backtrack(num, ret, root);
    return ret;
  }

  void backtrack(long long& num, long long& ret, TreeNode* root) {
    if (!root) return;
    if (!root->left && !root->right) {
```

```
      ret += num * 10 + root->val;
      return;
    }
    num = num * 10 + root->val;
    backtrack(num, ret, root->left);
    backtrack(num, ret, root->right);
    num /= 10;
  }
};
```

130. Surrounded Regions

Description

```
Given a 2D board containing 'X' and 'O' (the letter O), capture all regions
surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

Example:

X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X
Explanation:

Surrounded regions shouldn’t be on the border, which means that any 'O' on the
border of the board are not flipped to 'X'. Any 'O' that is not on the border
and it is not connected to an 'O' on the border will be flipped to 'X'. Two
cells are connected if they are adjacent cells connected horizontally or
vertically.
```

Solution

05/08/2020 Discussion:

The idea is borrowed from one of the assignments of the course Algorithms (Part 1):

1. We assume there is a virtual cell (labeled as `m * n`) that the node on the boundary of the board is

automatically connected to it.
2. We only need to deal with the cells that are `'O'`: merge these neighboring `'O'` cells.
3. Traverse `'O'` cells which are stored in `visited`: if the cell is connected to the virtual cell do nothing; otherwise, change it to `'X'`.

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind (int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool connected(int x, int y) { return find(x) == find(y); }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[j] > sz[i]) {
      swap(i, j);
      swap(sz[i], sz[j]);
    }
    id[j] = i;
    sz[i] += sz[j];
    return true;
  }
};

class Solution {
public:
  void solve(vector<vector<char>>& board) {
    if (board.empty() || board[0].empty()) return;
    int m = board.size(), n = board[0].size(), bound = m * n;
    UnionFind uf(m * n + 1);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    unordered_set<int> visited;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (board[i][j] == 'X' || visited.count(i * n + j)) continue;
```

```
        if (i == 0 || i == m - 1 || j == 0 || j == n - 1) uf.merge(i * n + j,
bound);
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1];
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && board[ni][nj] == 'O' &&
visited.count(i * n + j) == 0) {
            uf.merge(i * n + j, ni * n + nj);
          }
        }
        visited.insert(i * n + j);
      }
    }
    for (auto& p : visited) {
      if (!uf.connected(bound, p)) {
        int i = p / n, j = p % n;
        board[i][j] = 'X';
      }
    }
  }
};
```

Update: Inspired by the above, we can actually use a set `uncaptured` to store all the cells that are connected to the bounds. We can always use BFS which starts from the `'O'` cells on the boundary to find the inner cells which are connected to the boundary.

```
class Solution {
public:
  void solve(vector<vector<char>>& board) {
    if (board.empty() || board[0].empty()) return;
    int m = board.size(), n = board[0].size();
    unordered_set<int> uncaptured;
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i == 0 || i == m - 1 || j == 0 || j == n - 1) {
          if (board[i][j] == 'X') continue;
          queue<int> q;
          q.push(i * n + j);
          while (!q.empty()) {
            int sz = q.size();
            for (int s = 0; s < sz; ++s) {
              int cur = q.front(); q.pop();
              if (uncaptured.count(cur) > 0) continue;
              for (int d = 0; d < 4; ++d) {
                int ni = cur / n + dir[d][0], nj = cur % n + dir[d][1];
                if (ni >= 0 && ni < m && nj >= 0 && nj < n && board[ni][nj] ==
'O') {
                  q.push(ni * n + nj);
```

```
                }
              }
              uncaptured.insert(cur);
            }
          }
        }
      }
    }
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (board[i][j] == 'O' && uncaptured.count(i * n + j) == 0) {
          board[i][j] = 'X';
        }
      }
    }
  }
};
```

136. Single Number

Description

```
Given a non-empty array of integers, every element appears twice except for one.
Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it
without using extra memory?

Example 1:

Input: [2,2,1]
Output: 1
Example 2:

Input: [4,1,2,1,2]
Output: 4
```

Solution

05/27/2020:

```cpp
class Solution {
public:
  int singleNumber(vector<int>& nums) {
    int ret = 0;
    for (auto& i : nums) ret ^= i;
    return ret;
  }
};
```

137. Single Number II

Description

```
Given a non-empty array of integers, every element appears three times except
for one, which appears exactly once. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it
without using extra memory?

Example 1:

Input: [2,2,3,2]
Output: 3
Example 2:

Input: [0,1,0,1,0,1,99]
Output: 99
```

Solution

05/27/2020:

```cpp
class Solution {
public:
  int singleNumber(vector<int>& nums) {
    unordered_map<int, int> cnt;
    for (auto& i : nums) ++cnt[i];
    for (auto& [k, v] : cnt)
      if (v == 1)
        return k;
    return 0;
  }
};
```

144. Binary Tree Preorder Traversal

Description

```
Given a binary tree, return the preorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [1,2,3]
Follow up: Recursive solution is trivial, could you do it iteratively?
```

Solution

05/20/2020:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
```

```
  vector<int> preorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<int> ret;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      ret.push_back(cur->val);
      if (cur->right) st.push(cur->right);
      if (cur->left) st.push(cur->left);
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> preorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<int> ret;
    ret.push_back(root->val);
    vector<int> l = preorderTraversal(root->left);
    vector<int> r = preorderTraversal(root->right);
    ret.insert(ret.end(), l.begin(), l.end());
    ret.insert(ret.end(), r.begin(), r.end());
    return ret;
  }
};
```

145. Binary Tree Postorder Traversal

Description

```
Given a binary tree, return the postorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [3,2,1]
Follow up: Recursive solution is trivial, could you do it iteratively?
```

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<int> postorderTraversal(TreeNode* root) {
    vector<int> ret;
    if (root == nullptr) return ret;
    stack<TreeNode*> st;
    st.push(root);
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      ret.push_back(cur->val);
      if (cur->left != nullptr) st.push(cur->left);
      if (cur->right != nullptr) st.push(cur->right);
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

```
class Solution {
public:
```

```
  vector<int> postorderTraversal(TreeNode* root) {
    if (!root) return {};
    stack<TreeNode*> st;
    st.push(root);
    vector<int> ret;
    unordered_set<TreeNode*> visited;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if ((!cur->left || visited.count(cur->left)) && (!cur->right ||
visited.count(cur->right))) {
        ret.push_back(cur->val);
        visited.insert(cur);
      } else {
        st.push(cur);
        if (cur->right) st.push(cur->right);
        if (cur->left) st.push(cur->left);
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> postorderTraversal(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = postorderTraversal(root->left);
    vector<int> right = postorderTraversal(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    ret.push_back(root->val);
    return ret;
  }
};
```

146. LRU Cache

Description

```
Design and implement a data structure for Least Recently Used (LRU) cache. It
should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists
in the cache, otherwise return -1.
```

```
put(key, value) - Set or insert the value if the key is not already present.
When the cache reached its capacity, it should invalidate the least recently
used item before inserting a new item.

The cache is initialized with a positive capacity.

Follow up:
Could you do both operations in O(1) time complexity?

Example:

LRUCache cache = new LRUCache( 2 /* capacity */ );

cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // returns 1
cache.put(3, 3);    // evicts key 2
cache.get(2);       // returns -1 (not found)
cache.put(4, 4);    // evicts key 1
cache.get(1);       // returns -1 (not found)
cache.get(3);       // returns 3
cache.get(4);       // returns 4
```

Solution

Discussion:

- Use an `unordered_map<int, int> cache` to store the key-value pairs.
- Use an `unordered_map<int, int> priority` to store the current `rank` for a certain `key`.
- Use a max-heap (`priority_queue`) to store the priorities of `key`'s. The higher `rank` of the `key`, the earlier that the `key` will be removed from the `cache`.
- When we call `get(key)`, we update the `rank` for the current `key` by `priority[key] = rank` and then push this pair `{priority[key], key}` to the `priority_queue`. Therefore, the new pair would invalidate the original `{rank, key}` in the `priority_queue`. So inside the the `put` method, we use a `while` loop to invalidate all those invalid pairs until the first valid pair with the highest rank.
- Time complexity: `get`: O(1); `put`: O(m), where m is the number of operations of `get`.
- Space complexity: O(n + m), where n is the number of key-value pairs and m is s the number of operations of `get`.

```
class LRUCache {
private:
  unordered_map<int, int> cache;
  unordered_map<int, int> priority;
  priority_queue<pair<int, int>, vector<pair<int, int>>, less<pair<int, int>>>
pq;  // max-heap
  int capacity, rank;

public:
```

```
  LRUCache(int capacity) {
    this->capacity = capacity;
    rank = INT_MAX;
    cache.clear();
    priority.clear();
  }

  int get(int key) {
    if (cache.count(key) == 0) {
      return -1;
    } else {
      priority[key] = rank--;
      priority.emplace(key, priority[key]);
      pq.emplace(priority[key], key);
      return cache[key];
    }
  }

  void put(int key, int value) {
    cache[key] = value;
    priority[key] = rank--;
    pq.emplace(priority[key], key);
    while (cache.size() > capacity) {
      pair<int, int> top = pq.top();
      while (!pq.empty() && priority[top.second] != top.first) {
        pq.pop();
        top = pq.top();
      }
      pq.pop();
      cache.erase(top.second);
      priority.erase(top.second);
    }
  }
};
```

153. Find Minimum in Rotated Sorted Array

Description

```
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.

(i.e.,  [0,1,2,4,5,6,7] might become  [4,5,6,7,0,1,2]).

Find the minimum element.
```

```
You may assume no duplicate exists in the array.

Example 1:

Input: [3,4,5,1,2]
Output: 1
Example 2:

Input: [4,5,6,7,0,1,2]
Output: 0
```

Solution

04/28/2020:

```
class Solution {
public:
  int findMin(vector<int>& nums) {
    if (nums.empty()) return -1;
    int n = nums.size();
    int lo = 0, hi = n - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (nums[mid] > nums[hi]) {
        lo = mid + 1;
      } else {
        hi = mid;
      }
    }
    return nums[hi];
  }
};
```

162. Find Peak Element

Description

```
A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and
return its index.

The array may contain multiple peaks, in that case return the index to any one
of the peaks is fine.
```

```
You may imagine that nums[-1] = nums[n] = -∞.

Example 1:

Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index
number 2.
Example 2:

Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak
element is 2,
             or index number 5 where the peak element is 6.
Note:

Your solution should be in logarithmic complexity.
```

Solution

04/28/2020:

```
class Solution {
public:
  int findPeakElement(vector<int>& nums) {
    if (nums.empty()) return -1;
    int lo = 0, hi = nums.size() - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (mid - 1 >= lo && nums[mid] < nums[mid - 1]) {
        hi = mid - 1;
      } else if (mid + 1 <= hi && nums[mid] < nums[mid + 1]) {
        lo = mid + 1;
      } else {
        return mid;
      }
    }
    return lo;
  }
};
```

163. Missing Ranges

Description

```
Given a sorted integer array nums, where the range of elements are in the
inclusive range [lower, upper], return its missing ranges.

Example:

Input: nums = [0, 1, 3, 50, 75], lower = 0 and upper = 99,
Output: ["2", "4->49", "51->74", "76->99"]
```

Solution

05/24/2020:

```
class Solution {
public:
  vector<string> findMissingRanges(vector<int>& nums, int lower, int upper) {
    if (nums.empty()) {
      string s = lower == upper ? to_string(lower) : to_string(lower) + "->" +
to_string(upper);
      return {s};
    }
    long long cur = nums.back(); nums.pop_back();
    vector<string> ranges;
    string s = cur + 1 < upper ? to_string(cur + 1) + "->" + to_string(upper) :
to_string(cur + 1);
    if (cur - 1 >= lower) ranges = findMissingRanges(nums, lower, cur - 1);
    if (cur + 1 <= upper) ranges.push_back(s);
    return ranges;
  }
};
```

```
class Solution {
public:
  vector<string> findMissingRanges(vector<int>& nums, int lower, int upper) {
    if (lower > upper) return {};
    if (nums.empty()) {
      if (lower == upper)
        return { to_string(lower) };
      else
        return { to_string(lower) + "->" + to_string(upper) };
    }
    long long cur = nums.back(); nums.pop_back();
    vector<string> ranges;
    if (cur - 1 >= INT_MIN) {
       ranges = findMissingRanges(nums, lower, cur - 1);
    }
```

```
    if (cur + 1 < upper) {
      ranges.push_back(to_string(cur + 1) + "->" + to_string(upper));
    } else if (cur + 1 == upper) {
      ranges.push_back(to_string(cur + 1));
    }
    return ranges;
  }
};
```

169. Majority Element

Description

```
Given an array of size n, find the majority element. The majority element is the
element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist
in the array.

Example 1:

Input: [3,2,3]
Output: 3
Example 2:

Input: [2,2,1,1,1,2,2]
Output: 2
```

Solution

05/06/2020:

```
class Solution {
public:
  int majorityElement(vector<int>& nums) {
    int n = nums.size();
    if (n == 0) return -1;
    unordered_map<int, int> s;
    int major = nums[0], cnt = 1;
    for (auto& i : nums) {
      if (++s[i] > cnt) {
        cnt = s[i];
        major = i;
      }
    }
```

```
    return cnt > n / 2 ? major : -1;
  }
};
```

173. Binary Search Tree Iterator

Description

```
Implement an iterator over a binary search tree (BST). Your iterator will be
initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.



Example:



BSTIterator iterator = new BSTIterator(root);
iterator.next();    // return 3
iterator.next();    // return 7
iterator.hasNext(); // return true
iterator.next();    // return 9
iterator.hasNext(); // return true
iterator.next();    // return 15
iterator.hasNext(); // return true
iterator.next();    // return 20
iterator.hasNext(); // return false


Note:

next() and hasNext() should run in average O(1) time and uses O(h) memory, where
h is the height of the tree.
You may assume that next() call will always be valid, that is, there will be at
least a next smallest number in the BST when next() is called.
```

Solution

05/24/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class BSTIterator {
private:
  stack<TreeNode*> st;
  TreeNode* cur;

public:
  BSTIterator(TreeNode* root) {
    cur = root;
  }

  /** @return the next smallest number */
  int next() {
    while (cur) {
      st.push(cur);
      cur = cur->left;
    }
    cur = st.top(); st.pop();
    int ret = cur->val;
    cur = cur->right;
    return ret;
  }

  /** @return whether we have a next smallest number */
  bool hasNext() {
    return cur || !st.empty();
  }
};

/**
 * Your BSTIterator object will be instantiated and called as such:
 * BSTIterator* obj = new BSTIterator(root);
 * int param_1 = obj->next();
 * bool param_2 = obj->hasNext();
 */
```

```
class BSTIterator {
private:
  vector<TreeNode*> inorder;
```

```
  vector<TreeNode*> inorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<TreeNode*> leftTraversal = inorderTraversal(root->left);
    vector<TreeNode*> rightTraversal = inorderTraversal(root->right);
    leftTraversal.push_back(root);
    leftTraversal.insert(leftTraversal.end(), rightTraversal.begin(),
rightTraversal.end());
    return leftTraversal;
  }

public:
  BSTIterator(TreeNode* root) {
    inorder = inorderTraversal(root);
    reverse(inorder.begin(), inorder.end());
  }

  /** @return the next smallest number */
  int next() {
    TreeNode* ret = inorder.back();
    inorder.pop_back();
    return ret->val;
  }

  /** @return whether we have a next smallest number */
  bool hasNext() {
    return !inorder.empty();
  }
};
```

174. Dungeon Game

Description

The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).

```
In order to reach the princess as quickly as possible, the knight decides to
move only rightward or downward in each step.



Write a function to determine the knight's minimum initial health so that he is
able to rescue the princess.

For example, given the dungeon below, the initial health of the knight must be
at least 7 if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

-2 (K)  -3  3
-5  -10 1
10  30  -5 (P)


Note:

The knight's health has no upper bound.
Any room can contain threats or power-ups, even the first room the knight enters
and the bottom-right room where the princess is imprisoned.
```

Solution

06/21/2020:

```
class Solution {
public:
  const int inf = numeric_limits<int>::max();
  vector<vector<int>> dp;
  int rows, cols;

  int getMinHealth(int curCell, int nextRow, int nextCol) {
    if (nextRow >= rows || nextCol >= cols) return inf;
    int nextCell = dp[nextRow][nextCol];
    return max(1, nextCell - curCell);
  }

  int calculateMinimumHP(vector<vector<int>>& dungeon) {
    rows = dungeon.size();
    cols = dungeon[0].size();
    dp.assign(rows, vector<int>(cols, inf));
    int curCell, rightHealth, downHealth, nextHealth, minHealth;
    for (int row = rows - 1; row >= 0; --row) {
      for (int col = cols - 1; col >= 0; --col) {
        curCell = dungeon[row][col];
        rightHealth = getMinHealth(curCell, row, col + 1);
        downHealth = getMinHealth(curCell, row +  1, col);
```

```
        nextHealth = min(rightHealth, downHealth);
        if (nextHealth != inf) {
          minHealth = nextHealth;
        } else {
          minHealth = curCell >= 0 ? 1 : 1 - curCell;
        }
        dp[row][col] = minHealth;
      }
    }
    return dp[0][0];
  }
};
```

198. House Robber

Description

```
You are a professional robber planning to rob houses along a street. Each house
has a certain amount of money stashed, the only constraint stopping you from
robbing each of them is that adjacent houses have security system connected and
it will automatically contact the police if two adjacent houses were broken into
on the same night.

Given a list of non-negative integers representing the amount of money of each
house, determine the maximum amount of money you can rob tonight without
alerting the police.

Example 1:

Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
Example 2:

Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5
(money = 1).
             Total amount you can rob = 2 + 9 + 1 = 12.
```

Solution

01/14/2020 (Dynamic Programming):

```cpp
class Solution {
public:
  int rob(vector<int>& nums) {
    if (nums.size() >= 2) nums[1] = max(nums[0], nums[1]);
    for (int i = 2; i < nums.size(); ++i)
        nums[i] = max(nums[i - 1], nums[i - 2] + nums[i]);
    return nums.size() > 0 ? nums.back() : 0;
  }
};
```

200. Number of Islands

Description

```
Given a 2d grid map of '1's (land) and '0's (water), count the number of
islands. An island is surrounded by water and is formed by connecting adjacent
lands horizontally or vertically. You may assume all four edges of the grid are
all surrounded by water.

Example 1:

Input:
11110
11010
11000
00000

Output: 1
Example 2:

Input:
11000
11000
00100
00011

Output: 3
```

Solution

05/08/2020 (Union-Find):

```cpp
class UnionFind {
private:
  vector<int> id;
```

```
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      id[j] = i;
      sz[i] += sz[j];
    } else {
      id[i] = j;
      sz[j] += sz[i];
    }
    return true;
  }
};

class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    if (grid.empty() || grid[0].empty()) return 0;
    int m = grid.size(), n = grid[0].size();
    UnionFind uf(m * n);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '0') continue;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1];
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == '1') {
            uf.merge(i * n + j, ni * n + nj);
          }
        }
      }
    }
    unordered_set<int> components;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
```

```
        if (grid[i][j] == '1')
          components.insert(uf.find(i * n + j));
    return components.size();
  }
};
```

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size(), ret = 0;
    stack<pair<int, int>> st;
    int d[4][2] = { {0, -1}, {0, 1}, {-1, 0}, {1, 0} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
          ++ret;
          st.emplace(i, j);
          while (!st.empty()) {
            pair<int, int> cur = st.top(); st.pop();
            grid[cur.first][cur.second] = '0';
            for (int k = 0; k < 4; ++k) {
              int ni = cur.first + d[k][0], nj = cur.second + d[k][1];
              if (ni > -1 && ni < m && nj > -1 && nj < n && grid[ni][nj] == '1')
st.emplace(ni, nj);
            }
          }
        }
      }
    }
    return ret;
  }
};
```

Iterative DFS:

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size(), ret = 0;
    stack<pair<int, int>> st;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
```

```
          ++ret;
          st.emplace(i, j);
          while (!st.empty()) {
            pair<int, int> cur = st.top(); st.pop();
            int ni = cur.first, nj = cur.second;
            grid[ni][nj] = '0';
            if (ni + 1 < m && grid[ni + 1][nj] == '1') st.emplace(ni + 1, nj);
            if (ni - 1 >= 0 && grid[ni - 1][nj] == '1') st.emplace(ni - 1, nj);
            if (nj + 1 < n && grid[ni][nj + 1] == '1') st.emplace(ni, nj + 1);
            if (nj - 1 >= 0 && grid[ni][nj - 1] == '1') st.emplace(ni, nj - 1);
          }
        }
      }
    }
    return ret;
  }
};
```

Recursive DFS:

```
class Solution {
public:
  int numIslands(vector<vector<char>>& grid) {
    int m = grid.size();
    if (m == 0) return 0;
    int n = grid[0].size();
    int ret = 0;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == '1') {
          ++ret;
          dfs(grid, i, j, m, n);
        }
      }
    }
    return ret;
  }

  void dfs(vector<vector<char>>& grid, int i, int j, int m, int n) {
    if (i >= 0 && i < m && j >= 0 && j < n && grid[i][j] == '1') {
      grid[i][j] = '0';
      dfs(grid, i - 1, j, m, n);
      dfs(grid, i + 1, j, m, n);
      dfs(grid, i, j - 1, m, n);
      dfs(grid, i, j + 1, m, n);
    }
  }
};
```

205. Isomorphic Strings

Description

```
Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while
preserving the order of characters. No two characters may map to the same
character but a character may map to itself.

Example 1:

Input: s = "egg", t = "add"
Output: true
Example 2:

Input: s = "foo", t = "bar"
Output: false
Example 3:

Input: s = "paper", t = "title"
Output: true
Note:
You may assume both s and t have the same length.
```

Solution

05/20/2020:

```
class Solution {
public:
  bool isIsomorphic(string s, string t) {
    unordered_map<char, unordered_set<char>> mpST, mpTS;
    for (int i = 0; i < (int)s.size(); ++i) {
      mpST[s[i]].insert(t[i]);
      mpTS[t[i]].insert(s[i]);
    }
    for (auto& m : mpST)
      if (m.second.size() > 1)
        return false;
    for (auto& m : mpTS)
      if (m.second.size() > 1)
        return false;
```

```
    return true;
  }
};
```

207. Course Schedule

Description

```
There are a total of numCourses courses you have to take, labeled from 0 to
numCourses-1.

Some courses may have prerequisites, for example to take course 0 you have to
first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, is it
possible for you to finish all courses?



Example 1:

Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0. So it is
possible.
Example 2:

Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0, and to take
course 0 you should
             also have finished course 1. So it is impossible.


Constraints:

The input prerequisites is a graph represented by a list of edges, not adjacency
matrices. Read more about how a graph is represented.
You may assume that there are no duplicate edges in the input prerequisites.
1 <= numCourses <= 10^5
```

Solution

05/29/2020:

```
class Solution {
public:
  bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    vector<vector<int>> adj(numCourses);
    for (auto& p : prerequisites) adj[p[1]].push_back(p[0]);
    vector<bool> path(numCourses);
    for (int i = 0; i < numCourses; ++i)
      if (isCyclic(i, adj, path))
        return false;
    return true;
  }

  bool isCyclic(int i, vector<vector<int>>& adj, vector<bool>& path) {
    if (path[i]) return true;
    if (adj[i].empty()) return false;
    path[i] = true;
    bool ret = false;
    for (auto& j : adj[i]) {
      ret = isCyclic(j, adj, path);
      if (ret) break;
    }
    path[i] = false;
    return ret;
  }
};
```

208. Implement Trie (Prefix Tree)

Description

```
Implement a trie with insert, search, and startsWith methods.

Example:

Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // returns true
trie.search("app");     // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");     // returns true
Note:
```

```
You may assume that all inputs are consist of lowercase letters a-z.
All inputs are guaranteed to be non-empty strings.
```

Solution

05/13/2020:

```
class Trie {
public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node();
  }

  /** Inserts a word into the trie. */
  void insert(string word) {
    Node* cur = root;
    for (auto& c : word) {
      if (!cur->mp[c])
        cur->mp[c] = new Node();
      cur = cur->mp[c];
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = find(word);
    return cur != nullptr && cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = find(prefix);
    return cur != nullptr;
  }

private:
  struct Node {
    bool isWord;
    unordered_map<char, Node*> mp;
    Node() : isWord(false) {}
  };

  Node* root;
```

```
  Node* find(const string& word) {
    Node* cur = root;
    for (auto& c : word) {
      cur = cur->mp[c];
      if (!cur) break;
    }
    return cur;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

```
class Trie {
private:
  struct Node {
    bool isWord;
    vector<Node*> children;
    Node() { isWord = false; children.resize(26, nullptr); }
    ~Node() { for(auto& c : children) delete c; }
  };

  Node* root;

  Node* find(const string& word) {
    Node* cur = root;
    for (auto& c : word) {
      cur = cur->children[c - 'a'];
      if (!cur) break;
    }
    return cur;
  }

public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node();
  }

  /** Inserts a word into the trie. */
  void insert(string word) {
    Node* cur = root;
    for (auto& c : word) {
```

```
      if (!cur->children[c - 'a'])
        cur->children[c - 'a'] = new Node();
      cur = cur->children[c - 'a'];
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = find(word);
    return cur && cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = find(prefix);
    return cur;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

```
struct Node {
  char character;
  bool isWord;
  vector<Node*> children;
  Node(char c, bool i) : character(c), isWord(i) {}
  ~Node() {
    for (auto& c : children) delete c;
  }
};

class Trie {
private:
  Node *root;

public:
  /** Initialize your data structure here. */
  Trie() {
    root = new Node('r', false);
  }
```

```
  /** Inserts a word into the trie. */
  void insert(string word) {
    int n = word.size();
    Node* cur = root;
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
        if (word[i] == c->character) {
          cur = c;
          foundCharacter = true;
          break;
        }
      }
      if (!foundCharacter) {
        Node* c = new Node(word[i], false);
        cur->children.push_back(c);
        cur = c;
      }
    }
    cur->isWord = true;
  }

  /** Returns if the word is in the trie. */
  bool search(string word) {
    Node* cur = root;
    int n = word.size();
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
        if (c->character == word[i]) {
          foundCharacter = true;
          cur = c;
          break;
        }
      }
      if (!foundCharacter) return false;
    }
    return cur->isWord;
  }

  /** Returns if there is any word in the trie that starts with the given
prefix. */
  bool startsWith(string prefix) {
    Node* cur = root;
    int n = prefix.size();
    for (int i = 0; i < n; ++i) {
      bool foundCharacter = false;
      for (auto& c : cur->children) {
```

```
        if (c->character == prefix[i]) {
          foundCharacter = true;
          cur = c;
          break;
        }
      }
      if (!foundCharacter) return false;
    }
    return true;
  }
};

/**
 * Your Trie object will be instantiated and called as such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```

219. Contains Duplicate II

Description

```
Given an array of integers and an integer k, find out whether there are two
distinct indices i and j in the array such that nums[i] = nums[j] and the
absolute difference between i and j is at most k.

Example 1:

Input: nums = [1,2,3,1], k = 3
Output: true
Example 2:

Input: nums = [1,0,1,1], k = 1
Output: true
Example 3:

Input: nums = [1,2,3,1,2,3], k = 2
Output: false
```

Solution

05/20/2020:

```cpp
class Solution {
public:
  bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_map<int, int> mp;
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      if (mp.count(nums[i]) > 0) {
        if (i - mp[nums[i]] <= k) {
          return true;
        }
      }
      mp[nums[i]] = i;
    }
    return false;
  }
};
```

```cpp
class Solution {
public:
  bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_set<int> visited;
    for (int i = 0; i < (int)nums.size(); ++i) {
      if (visited.count(nums[i]) > 0) return true;
      visited.insert(nums[i]);
      if (i - k >= 0) visited.erase(nums[i - k]);
    }
    return false;
  }
};
```

221. Maximal Square

Description

```
Given a 2D binary matrix filled with 0's and 1's, find the largest square
containing only 1's and return its area.

Example:

Input:

1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0

Output: 4
```

Solution

04/27/2020 (Dynamic Programming):

```
class Solution {
public:
  int maximalSquare(vector<vector<char>>& matrix) {
    int m = matrix.size();
    if (m == 0) return 0;
    int n = matrix[0].size();
    int ret = 0;
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        dp[i + 1][j + 1] = matrix[i][j] - '0';
        if (dp[i + 1][j + 1] > 0) {
          dp[i + 1][j + 1] = min(dp[i][j], min(dp[i][j + 1], dp[i + 1][j])) + 1;
          ret = max(ret, dp[i + 1][j + 1]);
        }
      }
    }
    return ret * ret;
  }
};
```

More space-efficient Dynamic Programming:

```
class Solution {
public:
  int maximalSquare(vector<vector<char>>& matrix) {
    if (matrix.size() == 0 || matrix[0].size() == 0) return 0;
    int m = matrix.size(), n = matrix[0].size();
    int ret = 0;
```

```
    vector<int> dp(n + 1, 0);
    for (int i = 0; i < m; ++i) {
      vector<int> tmp(dp);
      for (int j = 0; j < n; ++j) {
        dp[j + 1] = matrix[i][j] - '0';
        if (dp[j + 1] > 0) {
          dp[j + 1] = min(tmp[j], min(tmp[j + 1], dp[j])) + 1;
          ret = max(ret, dp[j + 1]);
        }
      }
    }
    return ret * ret;
  }
};
```

222. Count Complete Tree Nodes

Description

```
Given a complete binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely
filled, and all nodes in the last level are as far left as possible. It can have
between 1 and 2h nodes inclusive at the last level h.

Example:

Input:
    1
   / \
  2   3
 / \  /
4  5 6

Output: 6
```

Solution

06/23/2020:

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int countNodes(TreeNode* root) {
    if (!root) return 0;
    queue<TreeNode*> q;
    q.push(root);
    int ret = 0;
    while (!q.empty()) {
      int sz = q.size();
      ret += sz;
      for (int i = 0; i < sz; ++i) {
        TreeNode* cur = q.front(); q.pop();
        if (cur->right) q.push(cur->right);
        if (cur->left) q.push(cur->left);
      }
    }
    return ret;
  }
};
```

226. Invert Binary Tree

Description

```
Invert a binary tree.

Example:

Input:

     4
   /   \
  2     7
 / \   / \
1   3 6   9
Output:

     4
   /   \
```

```
  7     2
 / \   / \
9   6 3   1
Trivia:
This problem was inspired by this original tweet by Max Howell:

Google: 90% of our engineers use the software you wrote (Homebrew), but you
can’t invert a binary tree on a whiteboard so f*** off.
```

Solution

06/01/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  TreeNode* invertTree(TreeNode* root) {
    if (root == nullptr) return nullptr;
    swap(root->left, root->right);
    invertTree(root->left);
    invertTree(root->right);
    return root;
  }
};
```

230. Kth Smallest Element in a BST

Description

```
Given a binary search tree, write a function kthSmallest to find the kth
smallest element in it.
```

```
Example 1:

Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
Example 2:

Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
Follow up:
What if the BST is modified (insert/delete operations) often and you need to
find the kth smallest frequently? How would you optimize the kthSmallest
routine?



Constraints:

The number of elements of the BST is between 1 to 10^4.
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.
```

Solution

05/20/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
```

```cpp
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    stack<TreeNode*> st;
    TreeNode* cur = root;
    while (cur || !st.empty()) {
      while (cur) {
        st.push(cur);
        cur = cur->left;
      }
      cur = st.top(); st.pop();
      if (--k == 0) return cur->val;
      cur = cur->right;
    }
    return 0;
  }
};
```

```cpp
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    stack<TreeNode*> st;
    st.push(root);
    unordered_set<TreeNode*> visited;
    while (!st.empty()) {
      TreeNode* cur = st.top(); st.pop();
      if (cur->left && visited.count(cur->left) == 0) {
        if (cur->right) st.push(cur->right);
        st.push(cur);
        st.push(cur->left);
      } else {
        if (!cur->left && cur->right) st.push(cur->right);
        visited.insert(cur);
        if (--k == 0) return cur->val;
      }
    }
    return 0;
  }
};
```

```cpp
class Solution {
public:
  int n, ans;
  int kthSmallest(TreeNode* root, int k) {
    n = k;
    inorder(root);
    return ans;
```

```
  }

  void inorder(TreeNode* root) {
    if (!root) return;
    inorder(root->left);
    if (--n == 0) ans = root->val;
    inorder(root->right);
  }
};
```

```
class Solution {
public:
  int kthSmallest(TreeNode* root, int k) {
    vector<int> list = BST2vector(root);
    return list[k - 1];
  }

  vector<int> BST2vector(TreeNode* root) {
    if (!root) return {};
    vector<int> ret = BST2vector(root->left);
    ret.push_back(root->val);
    vector<int> right = BST2vector(root->right);
    ret.insert(ret.end(), right.begin(), right.end());
    return ret;
  }
};
```

231. Power of Two

Description

```
Given an integer, write a function to determine if it is a power of two.

Example 1:

Input: 1
Output: true
Explanation: 20 = 1
Example 2:

Input: 16
Output: true
Explanation: 24 = 16
Example 3:
```

```
Input: 218
Output: false
```

Solution

06/07/2020:

```
class Solution {
public:
  bool isPowerOfTwo(int n) {
    return n > 0 && __builtin_popcount(n) == 1;
  }
};
```

237. Delete Node in a Linked List

Description

```
Write a function to delete a node (except the tail) in a singly linked list,
given only access to that node.

Given linked list -- head = [4,5,1,9], which looks like following:




Example 1:

Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should
become 4 -> 1 -> 9 after calling your function.
Example 2:

Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should
become 4 -> 5 -> 9 after calling your function.


Note:

The linked list will have at least two elements.
All of the nodes' values will be unique.
```

```
The given node will not be the tail and it will always be a valid node of the
linked list.
Do not return anything from your function.
```

Solution

05/10/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
  void deleteNode(ListNode* node) {
    if (node == nullptr) return;
    if (node->next == nullptr) {
      node = node->next;
    } else {
      node->val = node->next->val;
      node->next = node->next->next;
      node = node->next;
    }
  }
};
```

06/02/2020:

```
class Solution {
public:
  void deleteNode(ListNode* node) {
    *node = *(node->next);
  }
};
```

```
class Solution {
public:
  void deleteNode(ListNode* node) {
    node->val = node->next->val;
    node->next = node->next->next;
  }
};
```

242. Valid Anagram

Description

```
Given two strings s and t , write a function to determine if t is an anagram of
s.

Example 1:

Input: s = "anagram", t = "nagaram"
Output: true
Example 2:

Input: s = "rat", t = "car"
Output: false
Note:
You may assume the string contains only lowercase alphabets.

Follow up:
What if the inputs contain unicode characters? How would you adapt your solution
to such case?
```

Solution

05/17/2020:

```
class Solution {
public:
  bool isAnagram(string s, string t) {
    vector<int> cntS(26, 0), cntT(26, 0);
    for (int i = 0; i < (int)s.size(); ++i) ++cntS[s[i] - 'a'];
    for (int i = 0; i < (int)t.size(); ++i) ++cntT[t[i] - 'a'];
    for (int i = 0; i < 26; ++i) {
      if (cntS[i] != cntT[i]) {
        return false;
      }
    }
    return true;
  }
};
```

```cpp
class Solution {
public:
  bool isAnagram(string s, string t) {
    sort(s.begin(), s.end());
    sort(t.begin(), t.end());
    return s == t;
  }
};
```

246. Strobogrammatic Number

Description

```
A strobogrammatic number is a number that looks the same when rotated 180
degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic. The number is
represented as a string.

Example 1:

Input:  "69"
Output: true
Example 2:

Input:  "88"
Output: true
Example 3:

Input:  "962"
Output: false
```

Solution

05/18/2020:

```
class Solution {
public:
  bool isStrobogrammatic(string num) {
    unordered_map<char, char> mp{ {'0', '0'}, {'1', '1'}, {'6', '9'}, {'8',
'8'}, {'9', '6'} };
    string stro = "";
    for (auto& n : num) {
      if (mp.count(n) == 0) return false;
      stro = mp[n] + stro;
    }
    return stro == num;
  }
};
```

249. Group Shifted Strings

Description

```
Given a string, we can "shift" each of its letter to its successive letter, for
example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

"abc" -> "bcd" -> ... -> "xyz"
Given a list of strings which contains only lowercase alphabets, group all
strings that belong to the same shifting sequence.

Example:

Input: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"],
Output:
[
  ["abc","bcd","xyz"],
  ["az","ba"],
  ["acef"],
  ["a","z"]
]
```

Solution

05/25/2020:

```
class Solution {
public:
  vector<vector<string>> groupStrings(vector<string>& strings) {
    vector<vector<string>> ret;
    unordered_map<string, vector<string>> shiftedStrings;
```

```
    for (auto& s : strings) shiftedStrings[base(s)].push_back(s);
    for (auto& [key, val] : shiftedStrings) ret.push_back(val);
    return ret;
  }

  string base(const string& s) {
    string ts;
    for (auto& c : s) ts += (c - s[0] + 26) % 26 + 'a';
    return ts;
  }
};
```

```
class Solution {
public:
  vector<vector<string>> groupStrings(vector<string>& strings) {
    unordered_map<int, vector<string>> shiftedStrings;
    for (auto& s : strings) shiftedStrings[hash(s)].push_back(s);
    vector<vector<string>> ret;
    for (auto& m : shiftedStrings) ret.push_back(m.second);
    return ret;
  }

  int hash(const string& s) {
    long long h = 1;
    if (s.empty()) return h;
    int offset = s[0] - 'a';
    string ts;
    const int MOD = 1e9 + 7;
    for (auto& c : s) ts += (c - 'a' - offset + 26) % 26 + 'a';
    for (auto& c : ts) h = (h * 31 + c - 'a') % MOD;
    return h;
  }
};
```

250. Count Univalue Subtrees

Description

```
Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same value.

Example :

Input:  root = [5,1,5,5,5,null,5]
```

```
              5
             / \
            1   5
           / \   \
          5   5   5

Output: 4
```

Solution

05/25/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  int countUnivalSubtrees(TreeNode* root) {
    pair<int, bool> ret = dfs(root);
    return ret.first;
  }

  pair<int, bool> dfs(TreeNode* root) {
    if (root == nullptr) return {0, true};
    if (root->left == nullptr && root->right == nullptr) return {1, true};
    pair<int, bool> countLeft = dfs(root->left);
    pair<int, bool> countRight = dfs(root->right);
    bool isUniqueTree = true;
    if (countLeft.first > 0) {
      isUniqueTree = isUniqueTree && root->val == root->left->val &&
countLeft.second;
    }
    if (countRight.first > 0) {
      isUniqueTree = isUniqueTree && root->val == root->right->val &&
countRight.second;
    }
    return {isUniqueTree + countLeft.first + countRight.first, isUniqueTree};
  }
```

```
};
```

```
class Solution {
public:
  int countUnivalSubtrees(TreeNode* root) {
    int cnt = 0;
    dfs(root, cnt);
    return cnt;
  }

  bool dfs(TreeNode* root, int& cnt) {
    if (root == nullptr) return true;
    if (root->left == nullptr && root->right == nullptr) {
      ++cnt;
      return true;
    }
    bool isUniqueTree = true;
    bool checkLeft = dfs(root->left, cnt);
    bool checkRight = dfs(root->right, cnt);
    if (root->left) isUniqueTree = isUniqueTree && root->val == root->left->val
&& checkLeft;
    if (root->right) isUniqueTree = isUniqueTree && root->val == root->right-
>val && checkRight;
    if (isUniqueTree) ++cnt;
    return isUniqueTree;
  }
};
```

251. Flatten 2D Vector

Description

```
Design and implement an iterator to flatten a 2d vector. It should support the
following operations: next and hasNext.



Example:

Vector2D iterator = new Vector2D([[1,2],[3],[4]]);

iterator.next(); // return 1
iterator.next(); // return 2
iterator.next(); // return 3
iterator.hasNext(); // return true
```

```
iterator.hasNext(); // return true
iterator.next(); // return 4
iterator.hasNext(); // return false


Notes:

Please remember to RESET your class variables declared in Vector2D, as
static/class variables are persisted across multiple test cases. Please see here
for more details.
You may assume that next() call will always be valid, that is, there will be at
least a next element in the 2d vector when next() is called.


Follow up:

As an added challenge, try to code it using only iterators in C++ or iterators
in Java.
```

Solution

05/25/2020:

```
class Vector2D {
private:
  vector<vector<int>> v;
  int i, j;

public:
  Vector2D(vector<vector<int>>& v) {
    this->v = v;
    i = j = 0;
    while (i < (int)v.size() && v[i].empty()) ++i;
  }

  int next() {
    int ret = v[i][j];
    if (i < (int)v.size() && ++j >= (int)v[i].size()) {
      j = 0, ++i;
      while (i < (int)v.size() && v[i].empty()) ++i;
    }
    return ret;
  }

  bool hasNext() {
    return i < (int)v.size() && j < (int)v[i].size();
  }
};
```

```
/**
 * Your Vector2D object will be instantiated and called as such:
 * Vector2D* obj = new Vector2D(v);
 * int param_1 = obj->next();
 * bool param_2 = obj->hasNext();
 */
```

```
class Vector2D {
private:
  vector<int> flattenedVector;

public:
  Vector2D(vector<vector<int>>& v) {
    flattenedVector.clear();
    for (auto& nums : v) {
      for (auto& i : nums) {
        flattenedVector.push_back(i);
      }
    }
    reverse(flattenedVector.begin(), flattenedVector.end());
  }

  int next() {
    int ret = flattenedVector.back();
    flattenedVector.pop_back();
    return ret;
  }

  bool hasNext() {
    return !flattenedVector.empty();
  }
};
```

252. Meeting Rooms

Description

```
Given an array of meeting time intervals consisting of start and end times
[[s1,e1],[s2,e2],...] (si < ei), determine if a person could attend all
meetings.

Example 1:

Input: [[0,30],[5,10],[15,20]]
Output: false
Example 2:

Input: [[7,10],[2,4]]
Output: true
NOTE: input types have been changed on April 15, 2019. Please reset to default
code definition to get new method signature.
```

Solution

05/26/2020:

```
class Solution {
public:
  bool canAttendMeetings(vector<vector<int>>& intervals) {
    if (intervals.empty()) return true;
    sort(intervals.begin(), intervals.end());
    for (int i = 1; i < (int)intervals.size(); ++i)
      if (intervals[i - 1][1] > intervals[i][0])
        return false;
    return true;
  }
};
```

253. Meeting Rooms II

Description

```
Given an array of meeting time intervals consisting of start and end times
[[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms
required.

Example 1:

Input: [[0, 30],[5, 10],[15, 20]]
Output: 2
Example 2:

Input: [[7,10],[2,4]]
Output: 1
NOTE: input types have been changed on April 15, 2019. Please reset to default
code definition to get new method signature.
```

Solution

05/27/2020:

```
class Solution {
public:
  int minMeetingRooms(vector<vector<int>>& intervals) {
    vector<int> startTime, endTime;
    for (auto& i : intervals) {
      startTime.push_back(i[0]);
      endTime.push_back(i[1]);
    }
    sort(startTime.begin(), startTime.end());
    sort(endTime.begin(), endTime.end());
    int ret = 0, cnt = 0;
    auto startIt = startTime.begin();
    auto endIt = endTime.begin();
    for (; startIt != startTime.end() || endIt != endTime.end(); ) {
      if (endIt == endTime.end() || (startIt != startTime.end() && *startIt <
*endIt)) {
        ret = max(ret, ++cnt);
        ++startIt;
      } else {
        --cnt;
        ++endIt;
      }
    }
    return ret;
  }
};
```

257. Binary Tree Paths

Description

```
Given a binary tree, return all root-to-leaf paths.

Note: A leaf is a node with no children.

Example:

Input:

   1
 /   \
2     3
 \
  5

Output: ["1->2->5", "1->3"]

Explanation: All root-to-leaf paths are: 1->2->5, 1->3
```

Solution

05/26/2020:

```
class Solution {
public:
  vector<string> binaryTreePaths(TreeNode* root) {
    string path;
    vector<string> ret;
    backtrack(root, path, ret);
    return ret;
  }

  void backtrack(TreeNode* root, const string& path, vector<string>& ret) {
    if (root == nullptr) return;
    if (root->left == nullptr && root->right == nullptr) {
      path.empty() ? ret.push_back(to_string(root->val)) : ret.push_back(path +
"->" + to_string(root->val));
      return;
    }
    string addend = (path.empty() ? "" : "->") + to_string(root->val);
    backtrack(root->left, path + addend, ret);
    backtrack(root->right, path + addend, ret);
  }
};
```

```cpp
class Solution {
public:
  vector<string> binaryTreePaths(TreeNode* root) {
    if (root == nullptr) return {};
    stack<pair<TreeNode*, string>> st;
    st.emplace(root, "");
    vector<string> ret;
    while (!st.empty()) {
      pair<TreeNode*, string> cur = st.top(); st.pop();
      cur.second += "->" + to_string(cur.first->val);
      if (cur.first->left == nullptr && cur.first->right == nullptr)
ret.push_back(cur.second.substr(2, cur.second.size() - 2));
      if (cur.first->right != nullptr) st.emplace(cur.first->right, cur.second);
      if (cur.first->left != nullptr) st.emplace(cur.first->left, cur.second);
    }
    return ret;
  }
};
```

```cpp
class Solution {
public:
  vector<string> binaryTreePaths(TreeNode* root) {
    vector<string> ret;
    string p = "";
    dfs(root, p, ret);
    return ret;
  }

  void dfs(TreeNode* root, string p, vector<string>& ret) {
    if (root == nullptr) return;
    p += "->" + to_string(root->val);
    if (root->left != nullptr) dfs(root->left, p, ret);
    if (root->right != nullptr) dfs(root->right, p, ret);
    if (root->left == nullptr && root->right == nullptr)
ret.push_back(p.substr(2, (int)p.size() - 2));
  }
};
```

265. Paint House II

Description

```
There are a row of n houses, each house can be painted with one of the k colors.
The cost of painting each house with a certain color is different. You have to
paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x k
cost matrix. For example, costs[0][0] is the cost of painting house 0 with color
0; costs[1][2] is the cost of painting house 1 with color 2, and so on... Find
the minimum cost to paint all houses.

Note:
All costs are positive integers.

Example:

Input: [[1,5,3],[2,9,4]]
Output: 5
Explanation: Paint house 0 into color 0, paint house 1 into color 2. Minimum
cost: 1 + 4 = 5;
             Or paint house 0 into color 2, paint house 1 into color 0. Minimum
cost: 3 + 2 = 5.
Follow up:
Could you solve it in O(nk) runtime?
```

Solution

05/28/2020:

```
class Solution {
public:
  int minCostII(vector<vector<int>>& costs) {
    if (costs.empty() || costs[0].empty()) return 0;
    int m = costs.size(), n = costs[0].size();
    for (int i = 1; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        int prevMinCost = INT_MAX;
        for (int k = 0; k < n; ++k)
          if (k != j)
            prevMinCost = min(prevMinCost, costs[i - 1][k]);
        costs[i][j] += prevMinCost;
      }
    }
    return *min_element(costs.back().begin(), costs.back().end());
  }
};
```

267. Palindrome Permutation II

Description

```
Given a string s, return all the palindromic permutations (without duplicates)
of it. Return an empty list if no palindromic permutation could be form.

Example 1:

Input: "aabb"
Output: ["abba", "baab"]
Example 2:

Input: "abc"
Output: []
```

Solution

05/28/2020:

```
class Solution {
public:
  unordered_map<char, int> cnt;
  vector<string> generatePalindromes(string s) {
    if (!isPalindromic(s)) return {};
    unordered_set<string> permutation;
    string prefix, oddChar;
    for (auto& [k, v] : cnt) {
      prefix += string(v / 2, k);
      if (v % 2 == 1) oddChar = string(1, k);
    }
    sort(prefix.begin(), prefix.end());
    do {
      string suffix(prefix);
      reverse(suffix.begin(), suffix.end());
      permutation.insert(prefix + oddChar + suffix);
    } while (next_permutation(prefix.begin(), prefix.end()));
    return vector<string>(permutation.begin(), permutation.end());
  }

  bool isPalindromic(const string& s) {
    int numOfOdds = 0;
    for (auto& c : s) ++cnt[c];
    for (auto& [k, v] : cnt)
      if (v % 2 == 1)
        if (++numOfOdds > 1)
          return false;
```

```
    return true;
  }
};
```

268. Missing Number

Description

```
Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find
the one that is missing from the array.

Example 1:

Input: [3,0,1]
Output: 2
Example 2:

Input: [9,6,4,2,3,5,7,0,1]
Output: 8
Note:
Your algorithm should run in linear runtime complexity. Could you implement it
using only constant extra space complexity?
```

Solution

05/27/2020:

```
class Solution {
public:
  int missingNumber(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      if (nums[i] != i) {
        return i;
      }
    }
    return n;
  }
};
```

```cpp
class Solution {
public:
  int missingNumber(vector<int>& nums) {
    unordered_set<int> seen;
    for (auto& i : nums) seen.insert(i);
    for (int i = 0; i <= (int)nums.size(); ++i)
      if (seen.count(i) == 0)
        return i;
    return -1;
  }
};
```

```cpp
class Solution {
public:
  int missingNumber(vector<int>& nums) {
    int n = nums.size();
    bitset<200000000> s;
    for (auto& i : nums) s[i] = 1;
    for (int i = 0; i <= n; ++i)
      if (s[i] == 0)
        return i;
    return -1;
  }
};
```

270. Closest Binary Search Tree Value

Description

```
Given a non-empty binary search tree and a target value, find the value in the
BST that is closest to the target.

Note:

Given target value is a floating point.
You are guaranteed to have only one unique value in the BST that is closest to
the target.
Example:

Input: root = [4,2,5,1,3], target = 3.714286

    4
   / \
  2   5
```

```
 / \
1   3

Output: 4
```

Solution

04/28/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int closestValue(TreeNode* root, double target) {
    if (root == nullptr) return INT_MAX;
    int n = root->val;
    int n_left = closestValue(root->left, target);
    int n_right = closestValue(root->right, target);
    n = !root->left || fabs(n - target) < fabs(n_left - target) ? n : n_left;
    n = !root->right || fabs(n - target) < fabs(n_right - target) ? n : n_right;
    return n;
  }
};
```

```
class Solution {
public:
  int closestValue(TreeNode* root, double target) {
    int n = root->val;
    double d = fabs(root->val - target);
    if (root->left && target < root->val) {
      int n_left = closestValue(root->left, target);
      double d_left = fabs(n_left - target);
      if (d_left < d) {
        n = n_left;
        d = d_left;
      }
    }
    if (root->right && target > root->val) {
      int n_right = closestValue(root->right, target);
      double d_right = fabs(n_right - target);
```

```
      if (d_right < d) {
        n = n_right;
        d = d_right;
      }
    }
    return n;
  }
};
```

```
class Solution {
public:
  int closestValue(TreeNode* root, double target) {
    int n = root->val;
    double d = fabs(root->val - target);
    if (root->left && target < root->val) {
      int n_left = closestValue(root->left, target);
      double d_left = fabs(n_left - target);
      if (d_left < d) {
        n = n_left;
        d = d_left;
      }
    }
    if (root->right && target > root->val) {
      int n_right = closestValue(root->right, target);
      double d_right = fabs(n_right - target);
      if (d_right < d) {
        n = n_right;
        d = d_right;
      }
    }
    return n;
  }
};
```

274. H-Index

Description

```
Given an array of citations (each citation is a non-negative integer) of a
researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if
h of his/her N papers have at least h citations each, and the other N − h papers
have no more than h citations each."

Example:

Input: citations = [3,0,6,1,5]
Output: 3
Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of
them had
             received 3, 0, 6, 1, 5 citations respectively.
             Since the researcher has 3 papers with at least 3 citations each
and the remaining
             two with no more than 3 citations each, her h-index is 3.
Note: If there are several possible values for h, the maximum one is taken as
the h-index.
```

Solution

05/30/2020:

```cpp
class Solution {
public:
  int hIndex(vector<int>& citations) {
    sort(citations.rbegin(), citations.rend());
    int lo = 0, hi = citations.size() - 1;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (citations[mid] >= mid + 1) {
        lo = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    return lo;
  }
};
```

276. Paint Fence

Description

```
There is a fence with n posts, each post can be painted with one of the k
colors.

You have to paint all the posts such that no more than two adjacent fence posts
have the same color.

Return the total number of ways you can paint the fence.

Note:
n and k are non-negative integers.

Example:

Input: n = 3, k = 2
Output: 6
Explanation: Take c1 as color 1, c2 as color 2. All possible ways are:

            post1  post2  post3
 -----      -----  -----  -----
   1         c1     c1     c2
   2         c1     c2     c1
   3         c1     c2     c2
   4         c2     c1     c1
   5         c2     c1     c2
   6         c2     c2     c1
```

Solution

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
  int numWays(int n, int k) {
    if (n == 0 || k == 0) return 0;
    vector<vector<int>> dp(n, vector<int>(2, 0));
    dp[0][0] = k;
    for (int i = 1; i < n; ++i) {
      dp[i][0] = (dp[i - 1][0] + dp[i - 1][1]) * (k - 1);
      dp[i][1] = dp[i - 1][0];
    }
    return dp.back()[0] + dp.back()[1];
  }
};
```

01/14/2020: (Dynamic Programming, Improve Space Complexity):

```
class Solution {
```

```
public:
  int numWays(int n, int k) {
    if (n == 0 || k == 0) return 0;
    vector<int> dp(2, 0);
    dp[0] = k;
    for (int i = 1; i < n; ++i) {
      int tmp = dp[0];
      dp[0] = (dp[0] + dp[1]) * (k - 1);
      dp[1] = tmp;
    }
    return dp[0] + dp[1];
  }
};
```

278. First Bad Version

Description

```
You are a product manager and currently leading a team to develop a new product.
Unfortunately, the latest version of your product fails the quality check. Since
each version is developed based on the previous version, all the versions after
a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first
bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether
version is bad. Implement a function to find the first bad version. You should
minimize the number of calls to the API.

Example:

Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true

Then 4 is the first bad version.
```

Solution

04/23/2020:

```
// The API isBadVersion is defined for you.
```

```cpp
// bool isBadVersion(int version);

class Solution {
public:
  int firstBadVersion(int n) {
    int lo = 1, hi = n;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (isBadVersion(mid) == true) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return lo;
  }
};
```

285. Inorder Successor in BST

Description

```
Given a binary search tree and a node in it, find the in-order successor of that
node in the BST.

The successor of a node p is the node with the smallest key greater than p.val.



Example 1:


Input: root = [2,1,3], p = 1
Output: 2
Explanation: 1's in-order successor node is 2. Note that both p and the return
value is of TreeNode type.
Example 2:


Input: root = [5,3,6,2,4,null,null,1], p = 6
Output: null
Explanation: There is no in-order successor of the current node, so the answer
is null.


Note:
```

```
If the given node has no in-order successor in the tree, return null.
It's guaranteed that the values of the tree are unique.
```

Solution

05/24/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
    stack<TreeNode*> st;
    TreeNode* cur = root;
    while (cur || !st.empty()) {
      while (cur) {
        st.push(cur);
        cur = cur->left;
      }
      cur = st.top(); st.pop();
      if (cur->val > p->val) return cur;
      cur = cur->right;
    }
    return nullptr;
  }
};
```

```
class Solution {
public:
  TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
    vector<TreeNode*> inorder = inorderTraversal(root);
    for (auto& i : inorder) {
      if (i->val > p->val) {
        return i;
      }
    }
    return nullptr;
  }
```

```
  vector<TreeNode*> inorderTraversal(TreeNode* root) {
    if (root == nullptr) return {};
    vector<TreeNode*> leftTraversal = inorderTraversal(root->left);
    vector<TreeNode*> rightTraversal = inorderTraversal(root->right);
    leftTraversal.push_back(root);
    leftTraversal.insert(leftTraversal.end(), rightTraversal.begin(),
rightTraversal.end());
    return leftTraversal;
  }
};
```

286. Walls and Gates

Description

```
You are given a m x n 2D grid initialized with these three possible values.

-1 - A wall or an obstacle.
0 - A gate.
INF - Infinity means an empty room. We use the value 231 - 1 = 2147483647 to
represent INF as you may assume that the distance to a gate is less than
2147483647.
Fill each empty room with the distance to its nearest gate. If it is impossible
to reach a gate, it should be filled with INF.

Example:

Given the 2D grid:

INF  -1  0  INF
INF INF INF  -1
INF  -1 INF  -1
  0  -1 INF INF
After running your function, the 2D grid should be:

  3  -1   0   1
  2   2   1  -1
  1  -1   2  -1
  0  -1   3   4
```

Solution

05/08/2020 (DFS):

```
class Solution {
```

```
public:
  void wallsAndGates(vector<vector<int>>& rooms) {
    if (rooms.empty() || rooms[0].empty()) return;
    for (int i = 0; i < rooms.size(); ++i) {
      for (int j = 0; j < rooms[0].size(); ++j) {
        if (rooms[i][j] == 0) {
          dfs(rooms, i, j, rooms[i][j]);
        }
      }
    }
  }

  void dfs(vector<vector<int>>& rooms, int i, int j, int d) {
    int m = rooms.size(), n = rooms[0].size();
    if (!(i >= 0 && i < m && j >= 0 && j < n) || rooms[i][j] < d) return;
    rooms[i][j] = min(rooms[i][j], d);
    dfs(rooms, i - 1, j, d + 1);
    dfs(rooms, i + 1, j, d + 1);
    dfs(rooms, i, j - 1, d + 1);
    dfs(rooms, i, j + 1, d + 1);
  }
};
```

05/08/2020 (BFS):

```
class Solution {
public:
  void wallsAndGates(vector<vector<int>>& rooms) {
    if (rooms.empty() || rooms[0].empty()) return;

    int m = rooms.size(), n = rooms[0].size();
    queue<pair<int, int>> q;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (rooms[i][j] == 0) {
          q.emplace(i, j);
        }
      }
    }

    const int INF = numeric_limits<int>::max();
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    while (!q.empty()) {
      int sz = q.size();
      for (int k = 0; k < sz; ++k) {
        pair<int, int> cur = q.front(); q.pop();
        int i = cur.first, j = cur.second;
        for (int d = 0; d < 4; ++d) {
```

```
            int ni = i + dir[d][0];
            int nj = j + dir[d][1];
            if (ni >= 0 && ni < m && nj >= 0 && nj < n && rooms[ni][nj] == INF) {
              rooms[ni][nj] = rooms[i][j] + 1;
              q.emplace(ni, nj);
            }
          }
        }
      }
    }
};
```

287. Find the Duplicate Number

Description

```
Given an array nums containing n + 1 integers where each integer is between 1
and n (inclusive), prove that at least one duplicate number must exist. Assume
that there is only one duplicate number, find the duplicate one.

Example 1:

Input: [1,3,4,2,2]
Output: 2
Example 2:

Input: [3,1,3,4,2]
Output: 3
Note:

You must not modify the array (assume the array is read only).
You must use only constant, O(1) extra space.
Your runtime complexity should be less than O(n2).
There is only one duplicate number in the array, but it could be repeated more
than once.
```

Solution

04/28/2020:

Use linear search on sorted array:

```cpp
class Solution {
public:
  int findDuplicate(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    for (int i = 1; i < (int)nums.size(); ++i) {
      if (nums[i] == nums[i - 1]) {
        return nums[i];
      }
    }
    return -1;
  }
};
```

Use binary search on sorted array:

```cpp
class Solution {
public:
  int findDuplicate(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    for (int i = 0; i < nums.size(); ++i) {
      auto p = equal_range(nums.begin() + i, nums.end(), nums[i]);
      if (p.second - p.first >= 2) return *p.first;
    }
    return -1;
  }
};
```

Use hash map:

```cpp
class Solution {
public:
  int findDuplicate(vector<int>& nums) {
    unordered_map<int, int> mp;
    for (auto& n : nums) {
      if (++mp[n] > 1) {
        return n;
      }
    }
    return -1;
  }
};
```

290. Word Pattern

Description

```
Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter
in pattern and a non-empty word in str.

Example 1:

Input: pattern = "abba", str = "dog cat cat dog"
Output: true
Example 2:

Input:pattern = "abba", str = "dog cat cat fish"
Output: false
Example 3:

Input: pattern = "aaaa", str = "dog cat cat dog"
Output: false
Example 4:

Input: pattern = "abba", str = "dog dog dog dog"
Output: false
Notes:
You may assume pattern contains only lowercase letters, and str contains
lowercase letters that may be separated by a single space.
```

Solution

05/20/2020:

```
class Solution {
public:
  bool wordPattern(string pattern, string str) {
    istringstream iss(str);
    string s;
    vector<string> words;
    while (iss >> s) words.push_back(s);
    if (words.size() != pattern.size()) return false;
    unordered_map<char, unordered_set<string>> mp1;
    unordered_map<string, unordered_set<char>> mp2;
    for (int i = 0; i < (int)pattern.size(); ++i) {
      mp1[pattern[i]].insert(words[i]);
      mp2[words[i]].insert(pattern[i]);
    }
    for (auto& m : mp1) if (m.second.size() > 1) return false;
    for (auto& m : mp2) if (m.second.size() > 1) return false;
    return true;
```

```
  }
};
```

299. Bulls and Cows

Description

```
You are playing the following Bulls and Cows game with your friend: You write
down a number and ask your friend to guess what the number is. Each time your
friend makes a guess, you provide a hint that indicates how many digits in said
guess match your secret number exactly in both digit and position (called
"bulls") and how many digits match the secret number but locate in the wrong
position (called "cows"). Your friend will use successive guesses and hints to
eventually derive the secret number.

Write a function to return a hint according to the secret number and friend's
guess, use A to indicate the bulls and B to indicate the cows.

Please note that both secret number and friend's guess may contain duplicate
digits.

Example 1:

Input: secret = "1807", guess = "7810"

Output: "1A3B"

Explanation: 1 bull and 3 cows. The bull is 8, the cows are 0, 1 and 7.
Example 2:

Input: secret = "1123", guess = "0111"

Output: "1A1B"

Explanation: The 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow.
Note: You may assume that the secret number and your friend's guess only contain
digits, and their lengths are always equal.
```

Solution

02/05/2020:

```
class Solution {
public:
  string getHint(string secret, string guess) {
```

```
    vector<int> arr1(10, 0), arr2(10, 0);
    for (auto& n : secret) {
      ++arr1[n - '0'];
    }
    for (auto& n : guess) {
      ++arr2[n - '0'];
    }
    int match = 0, bull = 0;
    for (int i = 0; i < 10; ++i) {
      match += min(arr1[i], arr2[i]);
    }
    for (int i = 0; i < (int)secret.size(); ++i) {
      if (secret[i] == guess[i]) {
        ++bull;
      }
    }
    string ret;
    ret += to_string(bull) + "A" + to_string(match - bull) + "B";
    return ret;
  }
};
```

303. Range Sum Query - Immutable

Description

```
Given an integer array nums, find the sum of the elements between indices i and
j (i ≤ j), inclusive.

Example:

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
Note:

You may assume that the array does not change.
There are many calls to sumRange function.
```

Solution

01/14/2020 (Dynamic Programming):

```cpp
class NumArray {
public:
  vector<int> nums;
  NumArray(vector<int>& nums) {
    for (int i = 1; i < nums.size(); ++i) nums[i] += nums[i - 1];
    this->nums = nums;
  }

  int sumRange(int i, int j) {
    return i > 0 ? nums[j] - nums[i - 1] : nums[j];
  }
};

/**
 * Your NumArray object will be instantiated and called as such:
 * NumArray* obj = new NumArray(nums);
 * int param_1 = obj->sumRange(i,j);
 */
```

305. Number of Islands II

Description

```
A 2d grid map of m rows and n columns is initially filled with water. We may
perform an addLand operation which turns the water at position (row, col) into a
land. Given a list of positions to operate, count the number of islands after
each addLand operation. An island is surrounded by water and is formed by
connecting adjacent lands horizontally or vertically. You may assume all four
edges of the grid are all surrounded by water.

Example:

Input: m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]
Output: [1,1,2,3]
Explanation:

Initially, the 2d grid grid is filled with water. (Assume 0 represents water and
1 represents land).

0 0 0
0 0 0
0 0 0
Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

1 0 0
```

```
0 0 0   Number of islands = 1
0 0 0
Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

1 1 0
0 0 0   Number of islands = 1
0 0 0
Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

1 1 0
0 0 1   Number of islands = 2
0 0 0
Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

1 1 0
0 0 1   Number of islands = 3
0 1 0
Follow up:

Can you do it in time complexity O(k log mn), where k is the length of the
positions?
```

Solution

05/08/2020 (Union-Find) Discussion:

1. Use a set `islands` to store all the islands being added so far.
2. Use a set `components` to keep track of roots of current connected components (the size of `components` is the desired number of islands).
3. Whenever a new `island` is added, check whether the neighboring islands (in all four directions) exist. If there is a such `neighbor`, remove its root from the `components` and merge current `island` with that `neighbor`.
4. Then insert the root of `island` into `components`.
5. Store the size of `components`.

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
```

```
    while(x != id[x]) {
      id[x] = id[id[x]];
      x = id[x];
    }
    return x;
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
      id[i] = j;
    }
    return true;
  }
};

class Solution {
public:
  vector<int> numIslands2(int m, int n, vector<vector<int>>& positions) {
    if (m == 0 || n == 0 || positions.empty() || positions[0].empty()) return
{};
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    UnionFind uf(m * n);
    unordered_set<int> islands;
    unordered_set<int> components;
    vector<int> ret;
    for (auto& p : positions) {
      int i = p[0], j = p[1], island = i * n + j;
      islands.insert(island);
      for (int d = 0; d < 4; ++d) {
        int ni = i + dir[d][0], nj = j + dir[d][1], neighbor = ni * n + nj;
        if (ni >= 0 && ni < m && nj >= 0 && nj < n && islands.count(neighbor) >
0) {
          if (components.count(uf.find(neighbor))) {
            components.erase(uf.find(neighbor));
          }
          uf.merge(island, neighbor);
        }
      }
      components.insert(uf.find(island));
      ret.push_back(components.size());
    }
    return ret;
  }
```

```
};
```

311. Sparse Matrix Multiplication

Description

```
Given two sparse matrices A and B, return the result of AB.

You may assume that A's column number is equal to B's row number.

Example:

Input:

A = [
  [ 1, 0, 0],
  [-1, 0, 3]
]

B = [
  [ 7, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 1 ]
]

Output:

     |  1 0 0 |   | 7 0 0 |   |  7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
                  | 0 0 1 |
```

Solution

05/20/2020:

```
class Solution {
public:
  vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B) {
    if (A.empty() || A[0].empty()) return {};
    int nrowA = A.size(), ncolA = A[0].size(), ncolB = B[0].size();
    vector<vector<int>> ret(nrowA, vector<int>(ncolB, 0));
    for (int i = 0; i < nrowA; ++i) {
      for (int k = 0; k < ncolA; ++k) {
        if (A[i][k] == 0) continue;
        for (int j = 0; j < ncolB; ++j) {
```

```
          if (B[k][j] == 0) continue;
          ret[i][j] += A[i][k] * B[k][j];
        }
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B) {
    unordered_map<int, unordered_set<int>> matA, matB;
    int nrowA = A.size(), ncolA = A[0].size(), ncolB = B[0].size();
    for (int i = 0; i < nrowA; ++i)
      for (int j = 0; j < ncolA; ++j)
        if (A[i][j] != 0) matA[i].insert(j);
    for (int i = 0; i < ncolA; ++i)
      for (int j = 0; j < ncolB; ++j)
        if (B[i][j] != 0) matB[i].insert(j);
    vector<vector<int>> ret(nrowA, vector<int>(ncolB, 0));
    for (auto& mA : matA) {
      int i = mA.first;
      for (auto& k : mA.second)
        if (matB.count(k) > 0)
          for (auto& j : matB[k])
            ret[i][j] += A[i][k] * B[k][j];
    }
    return ret;
  }
};
```

314. Binary Tree Vertical Order Traversal

Description

```
Given a binary tree, return the vertical order traversal of its nodes' values.
(ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from left to
right.

Examples 1:

Input: [3,9,20,null,null,15,7]
```

```
   3
  /\
 /  \
 9  20
    /\
   /  \
  15   7
```

Output:

```
[
  [9],
  [3,15],
  [20],
  [7]
]
```

Examples 2:

Input: [3,9,8,4,0,1,7]

```
     3
    /\
   /  \
   9   8
  /\  /\
 /  \/  \
 4  01   7
```

Output:

```
[
  [4],
  [9],
  [3,0,1],
  [8],
  [7]
]
```

Examples 3:

Input: [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5)

```
     3
    /\
   /  \
   9   8
  /\  /\
 /  \/  \
```

```
 4  01   7
    /\
   /  \
   5   2

Output:

[
  [4],
  [9,5],
  [3,0,1],
  [8,2],
  [7]
]
```

Solution

04/26/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  vector<vector<int>> verticalOrder(TreeNode* root) {
    map<int, vector<int>> mp;
    queue<pair<TreeNode*, int>> q;
    q.emplace(root, 0);
    while (!q.empty()) {
      int n = q.size();
      for (int i = 0; i < n; ++i) {
        pair<TreeNode*, int> cur = q.front(); q.pop();
        mp[cur.second].push_back(cur.first->val);
        if (cur.first->left) q.emplace(cur.first->left, cur.second - 1);
        if (cur.first->right) q.emplace(cur.first->right, cur.second + 1);
      }
    }
    vector<vector<int>> ret;
    for (auto& m : mp) ret.push_back(m.second);
    return ret;
  }
};
```

Using `unordered_map`:

```cpp
class Solution {
public:
  vector<vector<int>> verticalOrder(TreeNode* root) {
    if (root == nullptr) return {};
    int minCol = 0, maxCol = 0;
    unordered_map<int, vector<int>> mp;
    queue<pair<TreeNode*, int>> q;
    q.emplace(root, 0);
    while (!q.empty()) {
      int n = q.size();
      for (int i = 0; i < n; ++i) {
        pair<TreeNode*, int> cur = q.front(); q.pop();
        mp[cur.second].push_back(cur.first->val);
        minCol = min(minCol, cur.second);
        maxCol = max(maxCol, cur.second);
        if (cur.first->left) q.emplace(cur.first->left, cur.second - 1);
        if (cur.first->right) q.emplace(cur.first->right, cur.second + 1);
      }
    }
    vector<vector<int>> ret;
    for (int i = minCol; i <= maxCol; ++i)
      if (mp.count(i) > 0)
        ret.push_back(mp[i]);
    return ret;
  }
};
```

322. Coin Change

Description

```
You are given coins of different denominations and a total amount of money
amount. Write a function to compute the fewest number of coins that you need to
make up that amount. If that amount of money cannot be made up by any
combination of the coins, return -1.

Example 1:

Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
Example 2:

Input: coins = [2], amount = 3
Output: -1
Note:
You may assume that you have an infinite number of each kind of coin.
```

Solution

05/27/2020:

```
class Solution {
public:
  int coinChange(vector<int>& coins, int amount) {
    vector<int> dp(amount + 1, INT_MAX);
    dp[0] = 0;
    for (int i = 1; i <= amount; ++i)
      for (auto& c : coins)
        if (i - c >= 0 && dp[i - c] != INT_MAX)
          dp[i] = min(dp[i - c] + 1, dp[i]);
    return dp.back() == INT_MAX ? -1 : dp.back();
  }
};
```

323. Number of Connected Components in an Undirected Graph

Description

```
Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge
is a pair of nodes), write a function to find the number of connected components
in an undirected graph.

Example 1:
```

```
Input: n = 5 and edges = [[0, 1], [1, 2], [3, 4]]

     0          3
     |          |
     1 --- 2    4

Output: 2
Example 2:

Input: n = 5 and edges = [[0, 1], [1, 2], [2, 3], [3, 4]]

     0           4
     |           |
     1 --- 2 --- 3

Output:  1
Note:
You can assume that no duplicate edges will appear in edges. Since all edges are
undirected, [0, 1] is the same as [1, 0] and thus will not appear together in
edges.
```

Solution

05/05/2020 (BFS):

```
class Solution {
public:
  int countComponents(int n, vector<vector<int>>& edges) {
    if (n == 0) return 0;
    unordered_map<int, unordered_set<int>> next;
    for (auto& e : edges) {
      next[e[0]].insert(e[1]);
      next[e[1]].insert(e[0]);
    }
    unordered_set<int> visited;
    int ret = 0;
    for (int i = 0; i < n; ++i) {
      if (visited.count(i) > 0) continue;
      queue<int> q;
      q.push(i);
      visited.insert(i);
      while (!q.empty()) {
        int cur = q.front(); q.pop();
        for (auto& c : next[cur]) {
          if (visited.count(c) == 0) q.push(c);
          visited.insert(c);
        }
```

```
      }
      ++ret;
    }
    return ret;
  }
};
```

03/05/2020 (Union-Find Set):

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    while (x != id[x]) {
      id[x] = id[id[x]];
      x = id[x];
    }
    return x;
    // if (x == id[x]) return x;
    // return id[x] = find(id[x]);
  }

  bool connected(int x, int y) {
    return find(x) == find(y);
  }

  void merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return;
    if (sz[i] > sz[j]) {
      id[j] = i;
      sz[i] += sz[j];
    } else {
      id[i] = j;
      sz[j] = sz[i];
    }
  }
};
```

```cpp
class Solution {
public:
  int countComponents(int n, vector<vector<int>>& edges) {
    UnionFind uf(n);
    for (auto& e : edges) {
      uf.merge(e[0], e[1]);
    }

    unordered_set<int> roots;
    for (int i = 0; i < n; ++i) {
      roots.insert(uf.find(i));
    }
    return roots.size();
  }
};
```

326. Power of Three

Description

```
Given an integer, write a function to determine if it is a power of three.

Example 1:

Input: 27
Output: true
Example 2:

Input: 0
Output: false
Example 3:

Input: 9
Output: true
Example 4:

Input: 45
Output: false
Follow up:
Could you do it without using any loop / recursion?
```

Solution

06/10/2020:

```cpp
class Solution {
public:
  bool isPowerOfThree(int n) {
    return n > 0 && (1162261467 % n == 0);
  }
};
```

```cpp
class Solution {
public:
  bool isPowerOfThree(int n) {
    if (n <= 0) return false;
    for (; n != 1 && n % 3 == 0; n /= 3);
    return n == 1;
  }
};
```

328. Odd Even Linked List

Description

```
Given a singly linked list, group all odd nodes together followed by the even
nodes. Please note here we are talking about the node number and not the value
in the nodes.

You should try to do it in place. The program should run in O(1) space
complexity and O(nodes) time complexity.

Example 1:

Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
Example 2:

Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
Note:

The relative order inside both the even and odd groups should remain as it was
in the input.
The first node is considered odd, the second node even and so on ...
```

Solution

05/14/2020:

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* oddEvenList(ListNode* head) {
    if (!head || !head->next) return head;
    ListNode *oddHead = head, *evenHead = head->next;
    ListNode *cur = evenHead, *oddCur = oddHead, *evenCur = evenHead;
    while (cur->next) {
      if (cur) {
        oddCur->next = cur->next;
        oddCur = oddCur->next;
        cur = cur->next;
      }
      if (cur) {
        evenCur->next = cur->next;
        evenCur = evenCur->next;
        cur = cur->next;
      }
      if (!cur) break;
    }
    oddCur->next = evenHead;
    return oddHead;
  }
};
```

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* oddEvenList(ListNode* head) {
```

```
    if (!head || !head->next) return head;
    ListNode *evenHead = head->next, *evenCur = evenHead, *cur = head;
    while (cur->next && evenCur->next) {
      cur->next = cur->next->next;
      cur = cur->next;
      evenCur->next = evenCur->next->next;
      evenCur = cur->next;
    }
    cur->next = evenHead;
    return head;
  }
};
```

344. Reverse String

Description

```
Write a function that reverses a string. The input string is given as an array
of characters char[].

Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.



Example 1:

Input: ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
Example 2:

Input: ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

Solution

05/30/2020:

```cpp
class Solution {
public:
  void reverseString(vector<char>& s) {
    for (int lo = 0, hi = s.size() - 1; lo < hi; ++lo, --hi) swap(s[lo], s[hi]);
  }
};
```

345. Reverse Vowels of a String

Description

```
Write a function that takes a string as input and reverse only the vowels of a
string.

Example 1:

Input: "hello"
Output: "holle"
Example 2:

Input: "leetcode"
Output: "leotcede"
Note:
The vowels does not include the letter "y".
```

Solution

06/16/2020:

```cpp
class Solution {
public:
  string reverseVowels(string s) {
    unordered_set<char> vowels{'a', 'e', 'i', 'o', 'u'};
    int n = s.size(), l = 0, r = n - 1;
    while (l < r) {
      while (!isalpha(s[l]) && l < n - 1) ++l;
      while (!isalpha(s[l]) && r > 0) --r;
      while (l < n - 1 && vowels.count(tolower(s[l])) == 0) ++l;
      while (r > 0 && vowels.count(tolower(s[r])) == 0) --r;
      if (l < r)
        if (vowels.count(tolower(s[l])) > 0 && vowels.count(tolower(s[r])) > 0)
          swap(s[l++], s[r--]);
    }
    return s;
  }
```

```
};
```

347. Top K Frequent Elements

Description

```
Given a non-empty array of integers, return the k most frequent elements.

Example 1:

Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
Example 2:

Input: nums = [1], k = 1
Output: [1]
Note:

You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
Your algorithm's time complexity must be better than O(n log n), where n is the
array's size.
It's guaranteed that the answer is unique, in other words the set of the top k
frequent elements is unique.
You can return the answer in any order.
```

Solution

05/20/2020:

```
class Solution {
public:
  vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> cnt;
    for (auto& n : nums) ++cnt[n];
    priority_queue<pair<int, int>, vector<pair<int, int>>, less<pair<int, int>>>
q;
    for (auto m : cnt) q.emplace(m.second, m.first);
    vector<int> ret;
    while (k-- > 0) {
      ret.push_back(q.top().second);
      q.pop();
    }
    return ret;
  }
};
```

```cpp
class Solution {
public:
  vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> cnt;
    for (auto& n : nums) ++cnt[n];
    vector<pair<int, int>> freq;
    for (auto& m : cnt) freq.emplace_back(m.second, m.first);
    sort(freq.begin(), freq.end(), greater<pair<int, int>>());
    vector<int> ret;
    for (auto& f : freq) {
      if (k-- > 0) {
        ret.push_back(f.second);
      } else {
        break;
      }
    }
    return ret;
  }
};
```

348. Design Tic-Tac-Toe

Description

```
Design a Tic-tac-toe game that is played between two players on a n x n grid.

You may assume the following rules:

A move is guaranteed to be valid and is placed on an empty block.
Once a winning condition is reached, no more moves is allowed.
A player who succeeds in placing n of their marks in a horizontal, vertical, or
diagonal row wins the game.
Example:
Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)
|X| | |
| | | |    // Player 1 makes a move at (0, 0).
| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 2 makes a move at (0, 2).
```

```
| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 1 makes a move at (2, 2).
| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 2 makes a move at (1, 1).
| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 1 makes a move at (2, 0).
|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)
|X| |O|
|O|O| |    // Player 2 makes a move at (1, 0).
|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
|X| |O|
|O|O| |    // Player 1 makes a move at (2, 1).
|X|X|X|
Follow up:
Could you do better than O(n2) per move() operation?
```

Solution

06/15/2020:

```
class TicTacToe {
  vector<vector<int>> board;
  vector<int> rows;
  vector<int> cols;
  int diag;
  int antiDiag;
  int n;

public:
  /** Initialize your data structure here. */
  TicTacToe(int n) {
    board.assign(n, vector<int>(n, -1));
    rows.assign(n, 0);
    cols.assign(n, 0);
    diag = 0;
```

```
    antiDiag = 0;
    this->n = n;
  }

  /** Player {player} makes a move at ({row}, {col}).
      @param row The row of the board.
      @param col The column of the board.
      @param player The player, can be either 1 or 2.
      @return The current winning condition, can be either:
              0: No one wins.
              1: Player 1 wins.
              2: Player 2 wins. */
  int move(int row, int col, int player) {
    if (board[row][col] != -1) return 0;
    board[row][col] = player;
    if (++rows[row] == n) {
      int countRow = 0;
      for (int i = 0; i < n && board[row][i] == player; ++i) ++countRow;
      if (countRow == n) return player;
    }
    if (++cols[col]) {
      int countCol = 0;
      for (int i = 0; i < n && board[i][col] == player; ++i) ++countCol;
      if (countCol == n) return player;
    }
    if (row == col && ++diag == n) {
      int countDiag = 0;
      for (int i = 0; i < n && board[i][i] == player; ++i) ++countDiag;
      if (countDiag == n) return player;
    }
    if (row == n - 1 - col && ++antiDiag == n) {
      int countAntiDiag = 0;
      for (int i = 0; i < n && board[i][n - i - 1] == player; ++i)
++countAntiDiag;
      if (countAntiDiag == n) return player;
    }
    return 0;
  }
};

/**
 * Your TicTacToe object will be instantiated and called as such:
 * TicTacToe* obj = new TicTacToe(n);
 * int param_1 = obj->move(row,col,player);
 */
```

367. Valid Perfect Square

Description

```
Given a positive integer num, write a function which returns True if num is a
perfect square else False.

Note: Do not use any built-in library function such as sqrt.

Example 1:

Input: 16
Output: true
Example 2:

Input: 14
Output: false
```

Solution

05/08/2020:

```
class Solution {
public:
  bool isPerfectSquare(int num) {
    int lo = 0, hi = num;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      long long s = (long long)mid * mid;
      if (s == num) {
        return true;
      } else if (s > num) {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return (long long)hi * hi == num;
  }
};
```

368. Largest Divisible Subset

Description

```
Given a set of distinct positive integers, find the largest subset such that
every pair (Si, Sj) of elements in this subset satisfies:

Si % Sj = 0 or Sj % Si = 0.

If there are multiple solutions, return any subset is fine.

Example 1:

Input: [1,2,3]
Output: [1,2] (of course, [1,3] will also be ok)
Example 2:

Input: [1,2,4,8]
Output: [1,2,4,8]
```

Solution

06/13/2020:

Dynamic Programming:

```
class Solution {
public:
  vector<int> largestDivisibleSubset(vector<int>& nums) {
    if (nums.empty()) return nums;
    sort(nums.begin(), nums.end());
    unordered_map<int, int> dp;
    unordered_map<int, int> pre;
    for (auto& n : nums) {
      dp[n] = 1;
      pre[n] = n;
    }
    int n = nums.size(), max_count = 1, max_num = nums[0];
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < i; ++j) {
        if (nums[i] % nums[j] == 0) {
          if (dp[nums[j]] + 1 > dp[nums[i]]) {
            pre[nums[i]] = nums[j];
            dp[nums[i]] = dp[nums[j]] + 1;
            if (dp[nums[i]] > max_count) {
              max_count = dp[nums[i]];
              max_num = nums[i];
            }
          }
        }
      }
    }
```

```
    vector<int> ret(1, max_num);
    while (max_num != pre[max_num]) {
      max_num = pre[max_num];
      ret.push_back(max_num);
    }
    return ret;
  }
};
```

Backtracking (TLE):

```
class Solution {
public:
  vector<int> ret;
  vector<int> largestDivisibleSubset(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    int n = nums.size();
    vector<int> s;
    backtrack(0, n, s, nums);
    return ret;
  }

  void backtrack(int k, int n, vector<int>& s, vector<int>& nums) {
    if (k == n) {
      if (s.size() > ret.size()) ret = s;
      return;
    }
    bool ok = true;
    for (int i = s.size() - 1; i >= 0 && ok; --i)
      ok = nums[k] % s[i] == 0;
    if (ok) {
      s.push_back(nums[k]);
      backtrack(k + 1, n, s, nums);
      s.pop_back();
    }
    backtrack(k + 1, n, s, nums);
  }
};
```

374. Guess Number Higher or Lower

Description

```
We are playing the Guess Game. The game is as follows:
```

```
I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results (-1,
1, or 0):

-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
Example :

Input: n = 10, pick = 6
Output: 6
```

Solution

04/23/2020:

```cpp
/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return       -1 if num is lower than the guess number
 *            1 if num is higher than the guess number
 *               otherwise return 0
 * int guess(int num);
 */

class Solution {
public:
  int guessNumber(int n) {
    int lo = 1, hi = n;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      int ans = guess(mid);
      if (ans == 0) {
        return mid;
      } else if (ans == 1) {
        lo = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    return lo;
  }
};
```

379. Design Phone Directory

Description

```
Design a Phone Directory which supports the following operations:



get: Provide a number which is not assigned to anyone.
check: Check if a number is available or not.
release: Recycle or release a number.


Example:

// Init a phone directory containing a total of 3 numbers: 0, 1, and 2.
PhoneDirectory directory = new PhoneDirectory(3);

// It can return any available phone number. Here we assume it returns 0.
directory.get();

// Assume it returns 1.
directory.get();

// The number 2 is available, so return true.
directory.check(2);

// It returns 2, the only number that is left.
directory.get();

// The number 2 is no longer available, so return false.
directory.check(2);

// Release number 2 back to the pool.
directory.release(2);

// Number 2 is available again, return true.
directory.check(2);


Constraints:

1 <= maxNumbers <= 10^4
0 <= number < maxNumbers
The total number of call of the methods is between [0 - 20000]
```

Solution

06/15/2020:

```
class PhoneDirectory {
private:
  vector<bool> numbers;
  int p, maxNumbers;
  stack<int> st;

public:
  /** Initialize your data structure here
      @param maxNumbers - The maximum numbers that can be stored in the phone
directory. */
  PhoneDirectory(int maxNumbers) {
    numbers.assign(maxNumbers, true);
    this->maxNumbers = maxNumbers;
    p = 0;
  }

  /** Provide a number which is not assigned to anyone.
      @return - Return an available number. Return -1 if none is available. */
  int get() {
    if (!st.empty()) {
      int cur = st.top(); st.pop();
      numbers[cur] = false;
      return cur;
    }
    while (!numbers[p] && p < maxNumbers) p++;
    if (p < maxNumbers) {
      numbers[p] = false;
      return p;
    } else {
      return -1;
    }
  }

  /** Check if a number is available or not. */
  bool check(int number) {
    return numbers[number];
  }

  /** Recycle or release a number. */
  void release(int number) {
    if (!numbers[number]) {
      numbers[number] = true;
      st.push(number);
    }
  }
};
```

```
/**
 * Your PhoneDirectory object will be instantiated and called as such:
 * PhoneDirectory* obj = new PhoneDirectory(maxNumbers);
 * int param_1 = obj->get();
 * bool param_2 = obj->check(number);
 * obj->release(number);
 */
```

```
class PhoneDirectory {
private:
  vector<bool> numbers;
  int p, maxNumbers;

public:
  /** Initialize your data structure here
      @param maxNumbers - The maximum numbers that can be stored in the phone
directory. */
  PhoneDirectory(int maxNumbers) {
    numbers.assign(maxNumbers, true);
    this->maxNumbers = maxNumbers;
    p = 0;
  }

  /** Provide a number which is not assigned to anyone.
      @return - Return an available number. Return -1 if none is available. */
  int get() {
    while (!numbers[p] && p < maxNumbers) p++;
    if (p < maxNumbers) {
      numbers[p] = false;
      return p;
    } else {
      return -1;
    }
  }

  /** Check if a number is available or not. */
  bool check(int number) {
    return numbers[number];
  }

  /** Recycle or release a number. */
  void release(int number) {
    numbers[number] = true;
    p = min(number, p);
  }
};
```

```
/**
 * Your PhoneDirectory object will be instantiated and called as such:
 * PhoneDirectory* obj = new PhoneDirectory(maxNumbers);
 * int param_1 = obj->get();
 * bool param_2 = obj->check(number);
 * obj->release(number);
 */
```

380. Insert Delete GetRandom O(1)

Description

```
Design a data structure that supports all following operations in average O(1)
time.

insert(val): Inserts an item val to the set if not already present.
remove(val): Removes an item val from the set if present.
getRandom: Returns a random element from current set of elements. Each element
must have the same probability of being returned.
Example:

// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

Solution

06/12/2020:

```
class RandomizedSet {
private:
  unordered_map<int, int> cnt;
  vector<int> nums;
  int n;

public:
  /** Initialize your data structure here. */
  RandomizedSet() {
    nums.clear();
    cnt.clear();
    n = 0;
  }

  /** Inserts a value to the set. Returns true if the set did not already
contain the specified element. */
  bool insert(int val) {
    if (cnt.count(val) > 0) return false;
    nums.push_back(val);
    cnt[val] = n++;
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the
specified element. */
  bool remove(int val) {
    if (cnt.count(val) == 0) return false;
    cnt[nums[n - 1]] = cnt[val];
    swap(nums[cnt[val]], nums[n - 1]);
    cnt.erase(val);
    nums.pop_back();
    --n;
    return true;
  }

  /** Get a random element from the set. */
  int getRandom() {
    if (n == 0) return -1;
    return nums[rand() % n];
  }
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * RandomizedSet* obj = new RandomizedSet();
```

```
 * bool param_1 = obj->insert(val);
 * bool param_2 = obj->remove(val);
 * int param_3 = obj->getRandom();
 */
```

```
class RandomizedSet {
private:
  unordered_set<int> cnt;
  vector<int> nums;
  bool ok = false;

public:
  /** Initialize your data structure here. */
  RandomizedSet() {
    nums.clear();
    cnt.clear();
  }

  /** Inserts a value to the set. Returns true if the set did not already
contain the specified element. */
  bool insert(int val) {
    if (cnt.count(val) > 0) return false;
    cnt.insert(val);
    nums.push_back(val);
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the
specified element. */
  bool remove(int val) {
    if (cnt.count(val) == 0) return false;
    cnt.erase(val);
    return true;
  }

  /** Get a random element from the set. */
  int getRandom() {
    if (nums.empty()) return -1;
    int i;
    for (i = rand() % nums.size(); nums.size() != 0 && cnt.count(nums[i]) == 0;
i = nums.empty() ? 0 : rand() % nums.size())
      nums.erase(nums.begin() + i);
    return nums.empty() ? -1 : nums[i];
  }
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
```

```
 * RandomizedSet* obj = new RandomizedSet();
 * bool param_1 = obj->insert(val);
 * bool param_2 = obj->remove(val);
 * int param_3 = obj->getRandom();
 */
```

384. Shuffle an Array

Description

```
Shuffle a set of numbers without duplicates.

Example:

// Init an array with set 1, 2, and 3.
int[] nums = {1,2,3};
Solution solution = new Solution(nums);

// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3]
must equally likely to be returned.
solution.shuffle();

// Resets the array back to its original configuration [1,2,3].
solution.reset();

// Returns the random shuffling of array [1,2,3].
solution.shuffle();
```

Solution

05/25/2020:

```
class Solution {
private:
  vector<int> nums;
  std::function<int()> mt19937_rand;
public:
  Solution(vector<int>& nums) {
    this->nums = nums;
    mt19937_rand = bind(uniform_int_distribution<int>(0, INT_MAX),
mt19937(time(0)));
  }

  /** Resets the array to its original configuration and return it. */
  vector<int> reset() {
```

```
    return nums;
  }

  /** Returns a random shuffling of the array. */
  vector<int> shuffle() {
    vector<int> shuffled(nums);
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      int j = mt19937_rand() % (i + 1);
      swap(shuffled[i], shuffled[j]);
    }
    return shuffled;
  }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(nums);
 * vector<int> param_1 = obj->reset();
 * vector<int> param_2 = obj->shuffle();
 */
```

```
class Solution {
private:
  vector<int> nums;
  mt19937 mt19937_rand;
  unsigned seed;
public:
  Solution(vector<int>& nums) {
    this->nums = nums;
    seed = std::chrono::system_clock::now().time_since_epoch().count();
    mt19937_rand.seed(seed);
  }

  /** Resets the array to its original configuration and return it. */
  vector<int> reset() {
    seed = std::chrono::system_clock::now().time_since_epoch().count();
    mt19937_rand.seed(seed);
    return nums;
  }

  /** Returns a random shuffling of the array. */
  vector<int> shuffle() {
    vector<int> shuffled(nums);
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      int j = mt19937_rand() % (i + 1);
      swap(shuffled[i], shuffled[j]);
```

```
    }
    return shuffled;
  }
};
```

387. First Unique Character in a String

Description

```
Given a string, find the first non-repeating character in it and return it's
index. If it doesn't exist, return -1.

Examples:

s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
Note: You may assume the string contain only lowercase letters.
```

Solution

05/05/2020:

```
class Solution {
public:
  int firstUniqChar(string s) {
    unordered_map<char, int> mp;
    for (auto& c : s) ++mp[c];
    int n = s.size();
    for (int i = 0; i < n; ++i)
      if (mp[s[i]] == 1)
        return i;
    return -1;
  }
};
```

389. Find the Difference

Description

```
Given two strings s and t which consist of only lowercase letters.

String t is generated by random shuffling string s and then add one more letter
at a random position.

Find the letter that was added in t.

Example:

Input:
s = "abcd"
t = "abcde"

Output:
e

Explanation:
'e' is the letter that was added.
```

Solution

05/17/2020:

```
class Solution {
public:
  char findTheDifference(string s, string t) {
    vector<int> cntS(26, 0);
    vector<int> cntT(26, 0);
    for (auto& c : s) {
      ++cntS[c - 'a'];
    }
    for (auto& c : t) {
      ++cntT[c - 'a'];
    }
    for (int i = 0; i < 26; ++i) {
      if (cntS[i] < cntT[i]) {
        return i + 'a';
      }
    }
    return 'a';
  }
};
```

392. Is Subsequence

Description

```
Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t
is potentially a very long (length ~= 500,000) string, and s is a short string
(<=100).

A subsequence of a string is a new string which is formed from the original
string by deleting some (can be none) of the characters without disturbing the
relative positions of the remaining characters. (ie, "ace" is a subsequence of
"abcde" while "aec" is not).

Example 1:
s = "abc", t = "ahbgdc"

Return true.

Example 2:
s = "axc", t = "ahbgdc"

Return false.

Follow up:
If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you
want to check one by one to see if T has its subsequence. In this scenario, how
would you change your code?

Credits:
Special thanks to @pbrother for adding this problem and creating all test cases.
```

Solution

01/14/2020 (Dynamic Programming, Memory Limit Exceeded):

```
class Solution {
public:
  bool isSubsequence(string s, string t) {
    if (s.size() >= t.size()) return s == t;
    return s.back() == t.back() ? isSubsequence(s.substr(0, s.size() - 1),
t.substr(0, t.size() - 1)) :
           isSubsequence(s, t.substr(0, t.size() - 1));
  }
};
```

01/14/2020 (Dynamic Programming (bottom-up), Two pointers):

```cpp
class Solution {
public:
  bool isSubsequence(string s, string t) {
    int ps = 0, pt = 0;
    for (; ps < s.size() && pt < t.size(); ++pt)
      if (s[ps] == t[pt]) ++ps;
    return ps == s.size();
  }
};
```

06/09/2020:

```cpp
class Solution {
public:
  bool isSubsequence(string s, string t) {
    auto sit = s.begin(), tit = t.begin();
    for (; sit != s.end() && tit != t.end(); ++tit)
      if (*sit == *tit) ++sit;
    return sit == s.end();
  }
};
```

402. Remove K Digits

Description

```
Given a non-negative integer num represented as a string, remove k digits from
the number so that the new number is the smallest possible.

Note:
The length of num is less than 10002 and will be ≥ k.
The given num does not contain any leading zero.
Example 1:

Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219
which is the smallest.
Example 2:

Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output
must not contain leading zeroes.
Example 3:
```

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing
which is 0.
```

Solution

05/12/2020:

```cpp
class Solution {
public:
  string removeKdigits(string num, int k) {
    deque<char> q;
    string ret;
    for (int i = 0; i < num.size(); ++i) {
      while (!q.empty() && q.back() > num[i] && k-- > 0) q.pop_back();
      q.push_back(num[i]);
    }
    while (!q.empty() && k-- > 0) q.pop_back();
    while (!q.empty() && q.front() == '0') q.pop_front();
    while (!q.empty()) ret += q.front(), q.pop_front();
    return ret.empty() ? string(1, '0') : ret;
  }
};
```

```cpp
class Solution {
public:
  string removeKdigits(string num, int k) {
    string ret;
    for (int i = 0; i < num.size(); ++i) {
      while (!ret.empty() && ret.back() > num[i] && k-- > 0)
        ret.pop_back();
      if (!ret.empty() || num[i] != '0')
        ret += num[i];
    }
    while (!ret.empty() && k-- > 0)
      ret.pop_back();
    return ret.empty() ? "0" : ret;
  }
};
```

404. Sum of Left Leaves

Description

```
Find the sum of all left leaves in a given binary tree.

Example:

    3
   / \
  9  20
    /  \
   15   7

There are two left leaves in the binary tree, with values 9 and 15 respectively.
Return 24.
```

Solution

Discussion

Recursive:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int sumOfLeftLeaves(TreeNode* root, bool isLeft = false) {
    if (root == nullptr) return 0;
    int s = !root->left && !root->right && isLeft ? root->val : 0;
    return s + sumOfLeftLeaves(root->left, true) + sumOfLeftLeaves(root->right,
false);
  }
};
```

Iterative:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
```

```
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int sumOfLeftLeaves(TreeNode* root) {
    if (root == nullptr) return 0;
    int s = 0;
    stack<pair<TreeNode*, bool>> st;
    st.emplace(root, false);
    while (!st.empty()) {
      pair<TreeNode*, bool> cur = st.top(); st.pop();
      s += cur.first->left == nullptr && cur.first->right == nullptr &&
cur.second ? cur.first->val : 0;
      if (cur.first->left) st.emplace(cur.first->left, true);
      if (cur.first->right) st.emplace(cur.first->right, false);
    }
    return s;
  }
};
```

406. Queue Reconstruction by Height

Description

```
Suppose you have a random list of people standing in a queue. Each person is
described by a pair of integers (h, k), where h is the height of the person and
k is the number of people in front of this person who have a height greater than
or equal to h. Write an algorithm to reconstruct the queue.

Note:
The number of people is less than 1,100.


Example

Input:
[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

Output:
[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]
```

Solution

06/06/2020:

```
class Solution {
public:
  vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
    if (people.empty() || people[0].empty()) return people;
    sort(people.begin(), people.end(), [](vector<int>& p1, vector<int>& p2) {
      if (p1[0] == p2[0]) return p1[1] < p2[1];
      return p1[0] > p2[0];
    });
    list<vector<int>> ret;
    for (int i = 0; i < (int)people.size(); ++i)
      ret.insert(next(ret.begin(), people[i][1]), people[i]);
    return vector<vector<int>>(ret.begin(), ret.end());
  }
};
```

```
class Solution {
public:
  vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
    if (people.empty() || people[0].empty()) return people;
    sort(people.begin(), people.end(), [](vector<int>& p1, vector<int>& p2) {
      if (p1[0] == p2[0]) return p1[1] < p2[1];
      return p1[0] > p2[0];
    });
    vector<vector<int>> ret;
    for (int i = 0; i < (int)people.size(); ++i)
      ret.insert(ret.begin() + people[i][1], people[i]);
    return ret;
  }
};
```

409. Longest Palindrome

Description

```
Given a string which consists of lowercase or uppercase letters, find the length
of the longest palindromes that can be built with those letters.

This is case sensitive, for example "Aa" is not considered a palindrome here.

Note:
Assume the length of given string will not exceed 1,010.

Example:
```

```
Input:
"abccccdd"

Output:
7

Explanation:
One longest palindrome that can be built is "dccaccd", whose length is 7.
```

Solution

05/18/2020:

```cpp
class Solution {
public:
  int longestPalindrome(string s) {
    vector<int> cnt(128, 0);
    for (auto& c : s) {
      ++cnt[c];
    }
    bool odd = false;
    int even = 0;
    for (int i = 0; i < 128; ++i) {
      if (cnt[i] % 2 == 0) {
        even += cnt[i];
      } else {
        odd = true;
        even += max(cnt[i] - 1, 0);
      }
    }
    return odd ? even + 1 : even;
  }
};
```

415. Add Strings

Description

```
Given two non-negative integers num1 and num2 represented as string, return the
sum of num1 and num2.

Note:

The length of both num1 and num2 is < 5100.
Both num1 and num2 contains only digits 0-9.
Both num1 and num2 does not contain any leading zero.
You must not use any built-in BigInteger library or convert the inputs to
integer directly.
```

Solution

06/09/2020:

```
class Solution {
public:
  string addStrings(string num1, string num2) {
    int carry = 0;
    int n1 = num1.size(), n2 = num2.size();
    string ret;
    for (int i1 = n1 - 1, i2 = n2 - 1; i1 >= 0 || i2 >= 0 || carry > 0; --i1, --
i2) {
      if (i1 >= 0) carry += num1[i1] - '0';
      if (i2 >= 0) carry += num2[i2] - '0';
      ret.push_back(carry % 10 + '0');
      carry /= 10;
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

429. N-ary Tree Level Order Traversal

Description

```
Given an n-ary tree, return the level order traversal of its nodes' values.

Nary-Tree input serialization is represented in their level order traversal,
each group of children is separated by the null value (See examples).



Example 1:
```

```
Input: root = [1,null,3,2,4,null,5,6]
Output: [[1],[3,2,4],[5,6]]
Example 2:



Input: root =
[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null
,null,14]
Output: [[1],[2,3,4,5],[6,7,8,9,10],[11,12,13],[14]]


Constraints:

The height of the n-ary tree is less than or equal to 1000
The total number of nodes is between [0, 10^4]
```

Solution

04/26/2020:

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {
public:
  vector<vector<int>> levelOrder(Node* root) {
    if (root == nullptr) return {};
```

```
    queue<Node*> q;
    q.push(root);
    vector<vector<int>> ret;
    ret.push_back({root->val});
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      for (int i = 0; i < n; ++i) {
        Node* cur = q.front(); q.pop();
        for (auto& c : cur->children) {
          level.push_back(c->val);
          q.push(c);
        }
      }
      if (!level.empty()) ret.push_back(level);
    }
    return ret;
  }
};
```

438. Find All Anagrams in a String

Description

```
Given a string s and a non-empty string p, find all the start indices of p's
anagrams in s.

Strings consists of lowercase English letters only and the length of both
strings s and p will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input:
s: "cbaebabacd" p: "abc"

Output:
[0, 6]

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
Example 2:

Input:
```

```
s: "abab" p: "ab"

Output:
[0, 1, 2]

Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```

Solution

05/16/2020:

```
class Solution {
public:
  vector<int> findAnagrams(string s, string p) {
    vector<int> sch(26, 0), pch(26, 0), ret;
    for (auto& c : p) ++pch[c - 'a'];
    int m = p.size(), n = s.size();
    if (m > n) return {};
    for (int i = 0; i < m - 1; ++i) ++sch[s[i] - 'a'];
    bool ok = false;
    for (int i = m - 1; i < n; ++i) {
      if (ok) {
        if (s[i] == s[i - m]) {
          ret.push_back(i - m + 1);
        } else {
          --sch[s[i - m] - 'a'];
          ++sch[s[i] - 'a'];
          ok = false;
        }
      } else {
        if (i >= m) --sch[s[i - m] - 'a'];
        ++sch[s[i] - 'a'];
        ok = true;
        for (int j = 0; j < 26; ++j) {
          if (pch[j] != sch[j]) {
            ok = false;
            break;
          }
        }
        if (ok) ret.push_back(i - m + 1);
      }
    }
    return ret;
  }
};
```

```cpp
class Solution {
public:
  vector<int> findAnagrams(string s, string p) {
    vector<int> sch(26, 0), pch(26, 0), ret;
    for (auto& c : p) ++pch[c - 'a'];
    int m = p.size(), n = s.size();
    for (int i = 0; i < n; ++i) {
      if (i - m >= 0) --sch[s[i - m] - 'a'];
      ++sch[s[i] - 'a'];
      bool ok = true;
      for (int j = 0; j < 26; ++j) {
        if (pch[j] != sch[j]) {
          ok = false;
          break;
        }
      }
      if (ok) ret.push_back(i - m + 1);
    }
    return ret;
  }
};
```

447. Number of Boomerangs

Description

```
Given n points in the plane that are all pairwise distinct, a "boomerang" is a
tuple of points (i, j, k) such that the distance between i and j equals the
distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and
coordinates of points are all in the range [-10000, 10000] (inclusive).

Example:

Input:
[[0,0],[1,0],[2,0]]

Output:
2

Explanation:
The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]
```

Solution

05/20/2020:

```
class Solution {
public:
  int numberOfBoomerangs(vector<vector<int>>& points) {
    if (points.empty() || points[0].empty()) return 0;
    int n = points.size(), cnt = 0;
    vector<unordered_map<double, int>> mp(n);
    for (int i = 0; i < n - 1; ++i) {
      for (int j = i + 1; j < n; ++j) {
        int x1 = points[i][0], x2 = points[j][0];
        int y1 = points[i][1], y2 = points[j][1];
        double d = hypot(x1 - x2, y1 - y2);
        ++mp[i][d];
        ++mp[j][d];
      }
    }
    for (int i = 0; i < n; ++i)
      for (auto& m : mp[i])
        cnt += nchoosek(m.second, 2) * 2;
    return cnt;
  }

  int nchoosek(int n, int k) {
    if (n < k) return 0;
    // n! / (k!) (n - k)!;
    // n * (n - 1) * ... * (n - k + 1) / k!
    long long num = 1, dem = 1;
    int kk = min(n - k, k);
    for (int i = 1; i <= kk; ++i) {
      num *= (n - i + 1);
      dem *= i;
    }
    return num / dem;
  }
};
```

451. Sort Characters By Frequency

Description

```
Given a string, sort it in decreasing order based on the frequency of
characters.

Example 1:
```

```
Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid
answer.
Example 2:

Input:
"cccaaa"

Output:
"cccaaa"

Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.
Note that "cacaca" is incorrect, as the same characters must be together.
Example 3:

Input:
"Aabb"

Output:
"bbAa"

Explanation:
"bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

Solution

05/20/2020:

```cpp
class Solution {
public:
  string frequencySort(string s) {
    unordered_map<char, int> mp;
    for (auto& c : s) ++mp[c];
    vector<pair<int, char>> str;
    for (auto& m : mp) str.emplace_back(m.second, m.first);
    sort(str.begin(), str.end(), greater<pair<int, char>>());
    string ret = "";
    for (auto& s : str) ret += string(s.first, s.second);
    return ret;
  }
};
```

468. Validate IP Address

Description

Write a function to check whether an input string is a valid IPv4 address or IPv6 address or neither.

IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots ("."), e.g.,172.16.254.1;

Besides, leading zeros in the IPv4 is invalid. For example, the address 172.16.254.01 is invalid.

IPv6 addresses are represented as eight groups of four hexadecimal digits, each group representing 16 bits. The groups are separated by colons (":"). For example, the address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 is a valid one. Also, we could omit some leading zeros among four hexadecimal digits and some low-case characters in the address to upper-case ones, so 2001:db8:85a3:0:0:8A2E:0370:7334 is also a valid IPv6 address(Omit leading zeros and using upper cases).

However, we don't replace a consecutive group of zero value with a single empty group using two consecutive colons (::) to pursue simplicity. For example, 2001:0db8:85a3::8A2E:0370:7334 is an invalid IPv6 address.

Besides, extra leading zeros in the IPv6 is also invalid. For example, the address 02001:0db8:85a3:0000:0000:8a2e:0370:7334 is invalid.

Note: You may assume there is no extra space or special characters in the input string.

```
Example 1:
Input: "172.16.254.1"

Output: "IPv4"

Explanation: This is a valid IPv4 address, return "IPv4".
Example 2:
Input: "2001:0db8:85a3:0:0:8A2E:0370:7334"

Output: "IPv6"

Explanation: This is a valid IPv6 address, return "IPv6".
Example 3:
Input: "256.256.256.256"

Output: "Neither"

Explanation: This is neither a IPv4 address nor a IPv6 address.
```

Solution

06/16/2020:

```
class Solution {
public:
  string validIPAddress(string IP) {
    int n = IP.size();
    vector<string> ipv4 = split(IP, '.');
    vector<string> ipv6 = split(IP, ':');
    bool isIpv4 = ipv4.size() == 4 && isdigit(IP.front()) && isdigit(IP.back());
    bool isIpv6 = ipv6.size() == 8 && (isalpha(IP.front()) ||
isdigit(IP.front())) && (isalpha(IP.back()) || isdigit(IP.back()));
    bool hasLeadingZero = false;
    if (isIpv4) {
      for (auto& s : ipv4) {
        isIpv4 = isIpv4 && s.size() <= 3 && s.size() > 0;
        for (auto& c : s) isIpv4 = isIpv4 && isdigit(c);
        if (!isIpv4) break;
        int n = stoi(s);
        isIpv4 = isIpv4 && (0 <= n && n <= 255 && to_string(n) == s);
      }
    } else if (isIpv6) {
      for (auto& s : ipv6) {
        isIpv6 = isIpv6 && s.size() > 0 && s.size() < 5;
        for (auto& c : s) isIpv6 = isIpv6 && (('0' <= c && c <= '9') || ('a' <=
c && c <= 'f') || ('A' <= c && c <= 'F'));
        if (!isIpv6) break;
```

```
      }
    }
    return isIpv4 ? "IPv4" : isIpv6 ? "IPv6" : "Neither";
  }

  vector<string> split(string IP, char delimiter) {
    istringstream iss(IP);
    string s;
    vector<string> ret;
    while (getline(iss, s, delimiter)) ret.push_back(s);
    return ret;
  }
};
```

470. Implement Rand10() Using Rand7()

Description

```
Given a function rand7 which generates a uniform random integer in the range 1
to 7, write a function rand10 which generates a uniform random integer in the
range 1 to 10.

Do NOT use system's Math.random().



Example 1:

Input: 1
Output: [7]
Example 2:

Input: 2
Output: [8,4]
Example 3:

Input: 3
Output: [8,1,10]


Note:

rand7 is predefined.
Each testcase has one argument: n, the number of times that rand10 is called.
```

```
Follow up:

What is the expected value for the number of calls to rand7() function?
Could you minimize the number of calls to rand7()?
```

Solution

05/27/2020:

```cpp
// The rand7() API is already defined for you.
// int rand7();
// @return a random integer in the range 1 to 7

class Solution {
public:
  int rand10() {
    int s = 41;
    while (s > 40) s = (rand7() - 1) * 7 + rand7();
    return (s - 1) % 10 + 1;
  }
};
```

475. Heaters

Description

```
Winter is coming! Your first job during the contest is to design a standard
heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find
out minimum radius of heaters so that all houses could be covered by those
heaters.

So, your input will be the positions of houses and heaters seperately, and your
expected output will be the minimum radius standard of heaters.

Note:

Numbers of houses and heaters you are given are non-negative and will not exceed
25000.
Positions of houses and heaters you are given are non-negative and will not
exceed 10^9.
As long as a house is in the heaters' warm radius range, it can be warmed.
All the heaters follow your radius standard and the warm radius will the same.
```

```
Example 1:

Input: [1,2,3],[2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the
radius 1 standard, then all the houses can be warmed.


Example 2:

Input: [1,2,3,4],[1,4]
Output: 1
Explanation: The two heater was placed in the position 1 and 4. We need to use
radius 1 standard, then all the houses can be warmed.
```

Solution

04/23/2020:

```
class Solution {
public:
  int findRadius(vector<int>& houses, vector<int>& heaters) {
    if ((int)houses.size() <= 0) return 0;
    int n = heaters.size();
    if (n <= 0) return INT_MAX;
    sort(heaters.begin(), heaters.end());
    for (auto& h : houses) {
      int lo = 0, hi = n - 1;
      while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (heaters[mid] <= h) {
          lo = mid + 1;
        } else {
          hi = mid - 1;
        }
      }
      int left = INT_MAX, right = INT_MAX;
      if (lo - 1 >= 0) left = min(right, h - heaters[lo - 1]);
      if (lo < n) right = min(left, heaters[lo] - h);
      h = min(left, right);
    }
    return *max_element(houses.begin(), houses.end());
  }
};
```

482. License Key Formatting

Description

```
You are given a license key represented as a string S which consists only
alphanumeric character and dashes. The string is separated into N+1 groups by N
dashes.

Given a number K, we would want to reformat the strings such that each group
contains exactly K characters, except for the first group which could be shorter
than K, but still must contain at least one character. Furthermore, there must
be a dash inserted between two groups and all lowercase letters should be
converted to uppercase.

Given a non-empty string S and a number K, format the string according to the
rules described above.

Example 1:

Input: S = "5F3Z-2e-9-w", K = 4

Output: "5F3Z-2E9W"

Explanation: The string S has been split into two parts, each part has 4
characters.
Note that the two extra dashes are not needed and can be removed.
Example 2:

Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

Explanation: The string S has been split into three parts, each part has 2
characters except the first part as it could be shorter as mentioned above.
Note:

The length of string S will not exceed 12,000, and K is a positive integer.
String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9)
and dashes(-).
String S is non-empty.
```

Solution

02/05/2020:

```
class Solution {
public:
```

```
  string licenseKeyFormatting(string S, int K) {
    string ret;
    int cnt = 0;
    for (auto it = S.rbegin(); it != S.rend(); ++it) {
      if (*it == '-') {
        continue;
      } else {
        ret += toupper(*it);
      }
      if (++cnt % K == 0) {
        cnt == 0;
        ret += '-';
      }
    }
    if (ret.back() == '-') {
      ret.pop_back();
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

498. Diagonal Traverse

Description

```
Given a matrix of M x N elements (M rows, N columns), return all elements of the
matrix in diagonal order as shown in the below image.



Example:

Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]

Output:  [1,2,4,7,5,3,6,8,9]

Explanation:
```

```
Note:

The total number of elements of the given matrix will not exceed 10,000.
```

Solution

05/20/2020:

```
class Solution {
public:
  vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return {};
    int m = matrix.size(), n = matrix[0].size();
    vector<int> ret;
    bool ascending = false;
    for (int k = 0; k < m + n; ++k) {
      if (ascending)
        for (int i = max(0, k - n + 1); i <= min(m - 1, k); ++i)
ret.push_back(matrix[i][k - i]);
      else
        for (int i = min(m - 1, k); i >= max(0, k - n + 1); --i)
ret.push_back(matrix[i][k - i]);
      ascending = !ascending;
    }
    return ret;
  }
};
```

510. Inorder Successor in BST II

Description

```
Given a node in a binary search tree, find the in-order successor of that node
in the BST.

If that node has no in-order successor, return null.

The successor of a node is the node with the smallest key greater than node.val.

You will have direct access to the node but not to the root of the tree. Each
node will have a reference to its parent node. Below is the definition for Node:

class Node {
    public int val;
    public Node left;
```

```
    public Node right;
    public Node parent;
}
```

Follow up:

Could you solve it without looking up any of the node's values?

Example 1:

```
Input: tree = [2,1,3], node = 1
Output: 2
Explanation: 1's in-order successor node is 2. Note that both the node and the
return value is of Node type.
```

Example 2:

```
Input: tree = [5,3,6,2,4,null,null,1], node = 6
Output: null
Explanation: There is no in-order successor of the current node, so the answer
is null.
```

Example 3:

```
Input: tree =
[15,6,18,3,7,17,20,2,4,null,13,null,null,null,null,null,null,null,null,9], node
= 15
Output: 17
```

Example 4:

```
Input: tree =
[15,6,18,3,7,17,20,2,4,null,13,null,null,null,null,null,null,null,null,9], node
= 13
Output: 15
```

Example 5:

```
Input: tree = [0], node = 0
Output: null
```

Constraints:

```
-10^5 <= Node.val <= 10^5
1 <= Number of Nodes <= 10^4
```

```
All Nodes will have unique values.
```

Solution

05/24/2020:

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* parent;
};
*/

class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    if (node->right != nullptr) return leftmostChild(node->right);
    Node* parent = smallestParentGreaterThanNode(node);
    return parent == nullptr || parent->val < node->val ? nullptr : parent;
  }

  Node* leftmostChild(Node* root) {
    if (root == nullptr) return nullptr;
    if (root->left == nullptr) return root;
    return leftmostChild(root->left);
  }

  Node* smallestParentGreaterThanNode(Node* node) {
    if (node == nullptr || node->parent == nullptr) return nullptr;
    if (node->parent->val > node->val) return node->parent;
    return smallestParentGreaterThanNode(node->parent);
  }
};
```

```
class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    if (node->right != nullptr) return leftmost(node->right);
    Node* parent = node->parent;
    while (true) {
      if (parent != nullptr && parent->val < node->val && parent->parent !=
nullptr) {
        parent = parent->parent;
```

```
      } else {
        break;
      }
    }
    return parent != nullptr && parent->val < node->val ? nullptr : parent;
  }

  Node* leftmost(Node* root) {
    if (root == nullptr) return nullptr;
    if (root->left == nullptr) return root;
    return leftmost(root->left);
  }
};
```

```
class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    Node* root = node;
    while (true) {
      if (root->parent != nullptr)
        root = root->parent;
      else
        break;
    }
    vector<Node*> inorder = inorderTraversal(root);
    auto it = find(inorder.begin(), inorder.end(), node);
    return it != inorder.end() && it + 1 != inorder.end() ? *(it + 1) : nullptr;
  }

  vector<Node*> inorderTraversal(Node* root) {
    if (root == nullptr) return {};
    vector<Node*> leftTraversal = inorderTraversal(root->left);
    vector<Node*> rightTraversal = inorderTraversal(root->right);
    leftTraversal.insert(leftTraversal.end(), root);
    leftTraversal.insert(leftTraversal.end(), rightTraversal.begin(),
rightTraversal.end());
    return leftTraversal;
  }
};
```

518. Coin Change 2

Description

```
You are given coins of different denominations and a total amount of money.
Write a function to compute the number of combinations that make up that amount.
You may assume that you have infinite number of each kind of coin.



Example 1:

Input: amount = 5, coins = [1, 2, 5]
Output: 4
Explanation: there are four ways to make up the amount:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
Example 2:

Input: amount = 3, coins = [2]
Output: 0
Explanation: the amount of 3 cannot be made up just with coins of 2.
Example 3:

Input: amount = 10, coins = [10]
Output: 1


Note:

You can assume that

0 <= amount <= 5000
1 <= coin <= 5000
the number of coins is less than 500
the answer is guaranteed to fit into signed 32-bit integer
```

Solution

05/27/2020:

```
class Solution {
public:
  int change(int amount, vector<int>& coins) {
    vector<int> dp(amount + 1, 0);
    dp[0] = 1;
    for (auto& c : coins) {
      for (int i = c; i <= amount; ++i) {
        if (i - c >= 0) {
          dp[i] += dp[i - c];
```

```
        }
      }
    }
    return dp.back();
  }
};
```

525. Contiguous Array

Description

```
Given a binary array, find the maximum length of a contiguous subarray with
equal number of 0 and 1.

Example 1:
Input: [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with equal number of 0
and 1.
Example 2:
Input: [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal
number of 0 and 1.
Note: The length of the given binary array will not exceed 50,000.
```

Solution

05/26/2020:

```
class Solution {
public:
  int findMaxLength(vector<int>& nums) {
    if (nums.empty()) return 0;
    unordered_map<int, int> firstOccurence{ {0, -1} };
    int n = nums.size(), ret = 0, prefix = 0;
    for (int i = 0; i < n; ++i) {
      prefix = prefix + (nums[i] == 0 ? 1 : -1);
      if (firstOccurence.count(prefix) == 0)
        firstOccurence[prefix] = i;
      else
        ret = max(ret, i - firstOccurence[prefix]);
    }
    return ret;
  }
```

```
};
```

528. Random Pick with Weight

Description

```
Given an array w of positive integers, where w[i] describes the weight of index
i, write a function pickIndex which randomly picks an index in proportion to its
weight.

Note:

1 <= w.length <= 10000
1 <= w[i] <= 10^5
pickIndex will be called at most 10000 times.
Example 1:

Input:
["Solution","pickIndex"]
[[[1]],[]]
Output: [null,0]
Example 2:

Input:
["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"]
[[[1,3]],[],[],[],[],[]]
Output: [null,0,1,1,1,0]
Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. Solution's
constructor has one argument, the array w. pickIndex has no arguments. Arguments
are always wrapped with a list, even if there aren't any.
```

Solution

06/05/2020:

```
class Solution {
private:
  vector<int> weights;
  mt19937 rand;

public:
  Solution(vector<int>& w) {
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
```

```
    rand.seed(seed);
    weights.resize(w.size());
    partial_sum(w.begin(), w.end(), weights.begin(), plus<int>());
  }

  int pickIndex() {
    long long n = (double) rand() / rand.max() * weights.back();
    return upper_bound(weights.begin(), weights.end(), n) - weights.begin();
  }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(w);
 * int param_1 = obj->pickIndex();
 */
```

535. Encode and Decode TinyURL

Description

```
Note: This is a companion problem to the System Design problem: Design TinyURL
(https://leetcode.com/discuss/interview-question/124658/Design-a-URL-Shortener-
(-TinyURL-)-System/).
TinyURL is a URL shortening service where you enter a URL such as
https://leetcode.com/problems/design-tinyurl and it returns a short URL such as
http://tinyurl.com/4e9iAk.

Design the encode and decode methods for the TinyURL service. There is no
restriction on how your encode/decode algorithm should work. You just need to
ensure that a URL can be encoded to a tiny URL and the tiny URL can be decoded
to the original URL.
```

Solution

05/20/2020:

```
class Solution {
private:
  unordered_map<string, string> decodeTable;
  unordered_map<string, string> encodeTable;
  string base = "http://tinyurl.com/";
  const int MOD = 1e9 + 7;

  int hash(string s) {
```

```
    long long h = 1;
    for (auto& c : s) h = (h * 131 + c) % MOD;
    return h;
  }

public:

  // Encodes a URL to a shortened URL.
  string encode(string longUrl) {
    if (encodeTable.count(longUrl) == 0) {
      string shortUrl = base + to_string(hash(longUrl));
      encodeTable[longUrl] = shortUrl;
      decodeTable[shortUrl] = longUrl;
    }
    return encodeTable[longUrl];
  }

  // Decodes a shortened URL to its original URL.
  string decode(string shortUrl) {
    return decodeTable[shortUrl];
  }
};

// Your Solution object will be instantiated and called as such:
// Solution solution;
// solution.decode(solution.encode(url));
```

540. Single Element in a Sorted Array

Description

```
You are given a sorted array consisting of only integers where every element
appears exactly twice, except for one element which appears exactly once. Find
this single element that appears only once.

Follow up: Your solution should run in O(log n) time and O(1) space.



Example 1:

Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
Example 2:

Input: nums = [3,3,7,7,10,11,11]
```

```
Output: 10


Constraints:

1 <= nums.length <= 10^5
0 <= nums[i] <= 10^5
```

Solution

05/12/2020:

```
class Solution {
public:
  int singleNonDuplicate(vector<int>& nums) {
    int lo = 0, hi = nums.size() - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if (mid % 2 == 0) {
        if (nums[mid] != nums[mid + 1]) {
          hi = mid;
        } else {
          lo = mid + 1;
        }
      } else {
        if (nums[mid] != nums[mid - 1]) {
          hi = mid;
        } else {
          lo = mid + 1;
        }
      }
    }
    return nums[lo];
  }
};
```

```
class Solution {
public:
  int singleNonDuplicate(vector<int>& nums) {
    int lo = 0, hi = nums.size() - 1;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      if ((!(mid & 1) && nums[mid] != nums[mid + 1]) || ((mid & 1) && nums[mid]
!= nums[mid - 1])) {
        hi = mid;
      } else {
        lo = mid + 1;
```

```
      }
    }
    return nums[lo];
  }
};
```

```
class Solution {
public:
  int singleNonDuplicate(vector<int>& nums) {
    int ret = 0;
    for (auto& n : nums) {
      ret ^= n;
    }
    return ret;
  }
};
```

542. 01 Matrix

Description

```
Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each
cell.

The distance between two adjacent cells is 1.



Example 1:

Input:
[[0,0,0],
 [0,1,0],
 [0,0,0]]

Output:
[[0,0,0],
 [0,1,0],
 [0,0,0]]
Example 2:

Input:
[[0,0,0],
 [0,1,0],
 [1,1,1]]
```

```
Output:
[[0,0,0],
 [0,1,0],
 [1,2,1]]


Note:

The number of elements of the given matrix will not exceed 10,000.
There are at least one 0 in the given matrix.
The cells are adjacent in only four directions: up, down, left and right.
```

Solution

06/09/2020: BFS:

```
class Solution {
public:
  vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return matrix;
    int m = matrix.size(), n = matrix[0].size();
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    queue<pair<int, int >> q;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        if (matrix[i][j] == 0)
          q.emplace(i, j);
        else
          matrix[i][j] = INT_MAX;

    while (!q.empty()) {
      pair<int, int> cur = q.front(); q.pop();
      int i = cur.first, j = cur.second;
      for (int d = 0; d < 4; ++d) {
        int ni = i + dir[d][0], nj = j + dir[d][1];
        if (ni < 0 || ni >= m || nj < 0 || nj >=n) continue;
        if (matrix[ni][nj] > matrix[i][j] + 1) {
          matrix[ni][nj] = matrix[i][j] + 1;
          q.emplace(ni, nj);
        }
      }
    }
    return matrix;
  }
};
```

```
class Solution {
public:
  vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return matrix;
    int m = matrix.size(), n = matrix[0].size();
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (matrix[i][j] == 0) continue;
        int cur_min = INT_MAX;
        queue<pair<int, int>> q;
        q.emplace(i, j);
        bool found = false;
        unordered_set<string> visited;
        while (!q.empty() && !found) {
          int sz = q.size();
          for (int s = 0; s < sz && !found; ++s) {
            pair<int, int> cur = q.front(); q.pop();
            string key = to_string(cur.first) + "," + to_string(cur.second);
            if (visited.count(key) > 0) continue;
            for (int d = 0; d < 4; ++d) {
              int ni = cur.first + dir[d][0], nj = cur.second + dir[d][1];
              string key = to_string(i) + "," + to_string(j);
              if (ni < 0 || ni >= m || nj < 0 || nj >= n) continue;
              if (matrix[ni][nj] == 0) {
                found = true;
                matrix[i][j] = (abs(i - ni) + abs(j - nj));
                break;
              } else {
                q.emplace(ni, nj);
              }
            }
            visited.insert(key);
          }
        }
      }
    }
    return matrix;
  }
};
```

Dynamic Programming:

```
class Solution {
public:
  vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return matrix;
    int m = matrix.size(), n = matrix[0].size();
```

```
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    queue<pair<int, int >> q;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        if (matrix[i][j] != 0)
          matrix[i][j] = INT_MAX;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i > 0 && matrix[i - 1][j] != INT_MAX) matrix[i][j] = min(matrix[i]
[j], matrix[i - 1][j] + 1);
        if (j > 0 && matrix[i][j - 1] != INT_MAX) matrix[i][j] = min(matrix[i]
[j], matrix[i][j - 1] + 1);
      }
    }
    for (int i = m - 1; i >= 0; --i) {
      for (int j = n - 1; j >= 0; --j) {
        if (i < m - 1 && matrix[i + 1][j] != INT_MAX) matrix[i][j] =
min(matrix[i][j], matrix[i + 1][j] + 1);
        if (j < n - 1 && matrix[i][j + 1] != INT_MAX) matrix[i][j] =
min(matrix[i][j], matrix[i][j + 1] + 1);
      }
    }
    return matrix;
  }
};
```

551. Student Attendance Record I

Description

```
You are given a string representing an attendance record for a student. The
record only contains the following three characters:
'A' : Absent.
'L' : Late.
'P' : Present.
A student could be rewarded if his attendance record doesn't contain more than
one 'A' (absent) or more than two continuous 'L' (late).

You need to return whether the student could be rewarded according to his
attendance record.

Example 1:

Input: "PPALLP"
Output: True
;xample 2:
```

```
Input: "PPALLL"
Output: False
```

Solution

02/05/2020:

```
class Solution {
public:
  bool checkRecord(string s) {
    int cntA = 0, cntLL = 0;
    for (auto i = 0; i < s.size(); ++i) {
      if (s[i] == 'A') {
        ++cntA;
      } else if (s[i] == 'L') {
        if (i > 0 && s[i - 1] == 'L') {
          ++cntLL;
        } else {
          cntLL = 1;
        }
      }
      if (cntA > 1 || cntLL > 2) {
        return false;
      }
    }
    return true;
  }
};
```

567. Permutation in String

Description

```
Given two strings s1 and s2, write a function to return true if s2 contains the
permutation of s1. In other words, one of the first string's permutations is the
substring of the second string.



Example 1:

Input: s1 = "ab" s2 = "eidbaooo"
Output: True
Explanation: s2 contains one permutation of s1 ("ba").
```

```
Example 2:

Input:s1= "ab" s2 = "eidboaoo"
Output: False


Note:

The input strings only contain lower case letters.
The length of both given strings is in range [1, 10,000].
```

Solution

05/18/2020:

```
class Solution {
public:
  bool checkInclusion(string s, string t) {
    vector<int> cntS(26, 0), cntT(26, 0);
    int n = s.size(), m = t.size();
    for (auto& c : s) ++cntS[c - 'a'];
    for (int i = 0; i < m; ++i) {
      ++cntT[t[i] - 'a'];
      bool ok = true;
      if (i - n >= 0) --cntT[t[i - n] - 'a'];
      for (int j = 0; j < 26; ++j)
        if (cntS[j] != cntT[j]) ok = false;
      if (ok) return true;
    }
    return false;
  }
};
```

609. Find Duplicate File in System

Description

```
Given a list of directory info including directory path, and all the files with
contents in this directory, you need to find out all the groups of duplicate
files in the file system in terms of their paths.

A group of duplicate files consists of at least two files that have exactly the
same content.

A single directory info string in the input list has the following format:
```

```
"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ... fn.txt(fn_content)"

It means there are n files (f1.txt, f2.txt ... fn.txt with content f1_content,
f2_content ... fn_content, respectively) in directory root/d1/d2/.../dm. Note
that n >= 1 and m >= 0. If m = 0, it means the directory is just the root
directory.

The output is a list of group of duplicate file paths. For each group, it
contains all the file paths of the files that have the same content. A file path
is a string that has the following format:

"directory_path/file_name.txt"

Example 1:

Input:
["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)", "root/c/d 4.txt(efgh)",
"root 4.txt(efgh)"]
Output:
[["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]


Note:

No order is required for the final output.
You may assume the directory name, file name and file content only has letters
and digits, and the length of file content is in the range of [1,50].
The number of files given is in the range of [1,20000].
You may assume no files or directories share the same name in the same
directory.
You may assume each given directory info represents a unique directory.
Directory path and file info are separated by a single blank space.


Follow-up beyond contest:
Imagine you are given a real file system, how will you search files? DFS or BFS?
If the file content is very large (GB level), how will you modify your solution?
If you can only read the file by 1kb each time, how will you modify your
solution?
What is the time complexity of your modified solution? What is the most time-
consuming part and memory consuming part of it? How to optimize?
How to make sure the duplicated files you find are not false positive?
```

Solution

05/20/2020:

```cpp
class Solution {
public:
  vector<vector<string>> findDuplicate(vector<string>& paths) {
    unordered_map<string, vector<string>> files;
    for (auto& p : paths) {
      istringstream iss(p);
      string base, file_content;
      iss >> base;
      while (iss >> file_content) {
        int i = 0, n = file_content.size();
        for (; file_content[i] != '(' && i < n; ++i);
        string filename = file_content.substr(0, i);
        string content = file_content.substr(i, n - i);
        files[content].push_back(base + "/" + filename);
      }
    }
    vector<vector<string>> ret;
    for (auto& f : files)
      if (f.second.size() > 1)
        ret.push_back(f.second);
    return ret;
  }
};
```

616. Add Bold Tag in String

Description

```
Given a string s and a list of strings dict, you need to add a closed pair of
bold tag <b> and </b> to wrap the substrings in s that exist in dict. If two
such substrings overlap, you need to wrap them together by only one pair of
closed bold tag. Also, if two substrings wrapped by bold tags are consecutive,
you need to combine them.
Example 1:

Input:
s = "abcxyz123"
dict = ["abc","123"]
Output:
"<b>abc</b>xyz<b>123</b>"


Example 2:

Input:
```

```
s = "aaabbcc"
dict = ["aaa","aab","bc"]
Output:
"<b>aaabbc</b>c"


Constraints:

The given dict won't contain duplicates, and its length won't exceed 100.
All the strings in input have length in range [1, 1000].
Note: This question is the same as 758: https://leetcode.com/problems/bold-words-in-string/
```

Solution

06/09/2020:

```
class Solution {
public:
  string addBoldTag(string s, vector<string>& dict) {
    unordered_set<string> dicts(dict.begin(), dict.end());
    vector<int> lengths;
    for (auto& d : dict) lengths.push_back(d.size());
    sort(lengths.rbegin(), lengths.rend());
    int n = s.size();
    vector<bool> isBold(n, false);
    for (int i = 0; i < n; ++i) {
      bool wrap = false;
      int wrapLength = 0;
      for (auto& len : lengths) {
        if (i + len <= n && dicts.count(s.substr(i, len)) > 0) {
          wrap = true;
          wrapLength = len;
          break;
        }
      }
      if (wrap) {
        for (int j = i; j < i + wrapLength; ++j)
          isBold[j] = true;
      }
    }
    string ret;
    int last = 0;
    for (int i = 0; i < n;) {
      if (isBold[i]) {
        while (i + 1 < n && isBold[i + 1]) ++i;
        ret += "<b>";
        ret += s.substr(last, i - last + 1);
```

```
        ret += "</b>";
        last = ++i;
      } else {
        ret += s[i];
        last = ++i;
      }
    }
    return ret;
  }
};
```

658. Find K Closest Elements

Description

```
Given a sorted array, two integers k and x, find the k closest elements to x in
the array. The result should also be sorted in ascending order. If there is a
tie, the smaller elements are always preferred.

Example 1:

Input: [1,2,3,4,5], k=4, x=3
Output: [1,2,3,4]
Example 2:

Input: [1,2,3,4,5], k=4, x=-1
Output: [1,2,3,4]
Note:

The value k is positive and will always be smaller than the length of the sorted
array.
Length of the given array is positive and will not exceed 104
Absolute value of elements in the array and x will not exceed 104
```

Solution

04/28/2020:

```
class Solution {
public:
  vector<int> findClosestElements(vector<int>& arr, int k, int x) {
    if (arr.empty()) return {};
    int n = arr.size();
    int lo = 0, hi = n - 1;
    int l, r;
```

```
    if (x < arr.front()) {
      l = 0;
      r = 0;
    } else if (x > arr.back()) {
      l = n - 1;
      r = n - 1;
    } else {
      while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (arr[mid] == x) {
          l = mid;
          r = mid;
          break;
        } else if (arr[mid] > x) {
          hi = mid - 1;
          r = hi;
        } else {
          lo = mid + 1;
          l = lo;
        }
      }
      if (abs(arr[l] - x) < abs(arr[r] - x)) {
        r = l;
      } else {
        l = r;
      }
    }
    deque<int> q;
    if (k-- > 0) {
      q.push_front(arr[l]);
      l--;
      r++;
    }
    while (k > 0) {
      if (l < 0) {
        q.push_back(arr[r++]);
      } else if (r > n - 1) {
        q.push_front(arr[l--]);
      } else if (abs(arr[l] - x) <= abs(arr[r] - x)) {
        q.push_front(arr[l--]);
      } else if (abs(arr[l] - x) > abs(arr[r] - x)) {
        q.push_back(arr[r++]);
      }
      --k;
    }
    return vector<int>(q.begin(), q.end());
  }
};
```

669. Trim a Binary Search Tree

Description

```
Given a binary search tree and the lowest and highest boundaries as L and R,
trim the tree so that all its elements lies in [L, R] (R >= L). You might need
to change the root of the tree, so the result should return the new root of the
trimmed binary search tree.

Example 1:
Input:
    1
   / \
  0   2

  L = 1
  R = 2

Output:
    1
      \
       2
Example 2:
Input:
    3
   / \
  0   4
   \
    2
   /
  1

  L = 1
  R = 3

Output:
      3
     /
   2
  /
 1
```

Solution

05/10/2020:

```
/**
```

```
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  TreeNode* trimBST(TreeNode* root, int L, int R) {
    if (root == nullptr) return root;
    if (root->val < L) return trimBST(root->right, L, R);
    if (root->val > R) return trimBST(root->left, L, R);
    root->left = trimBST(root->left, L, R);
    root->right = trimBST(root->right, L, R);
    return root;
  }
};
```

684. Redundant Connection

Description

```
In this problem, a tree is an undirected graph that is connected and has no
cycles.

The given input is a graph that started as a tree with N nodes (with distinct
values 1, 2, ..., N), with one additional edge added. The added edge has two
different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges. Each element of edges is a
pair [u, v] with u < v, that represents an undirected edge connecting nodes u
and v.

Return an edge that can be removed so that the resulting graph is a tree of N
nodes. If there are multiple answers, return the answer that occurs last in the
given 2D-array. The answer edge [u, v] should be in the same format, with u < v.

Example 1:
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given undirected graph will be like this:
  1
 / \
2 - 3
```

```
Example 2:
Input: [[1,2], [2,3], [3,4], [1,4], [1,5]]
Output: [1,4]
Explanation: The given undirected graph will be like this:
5 - 1 - 2
    |   |
    4 - 3
Note:
The size of the input 2D-array will be between 3 and 1000.
Every integer represented in the 2D-array will be between 1 and N, where N is
the size of the input array.
```

Solution

06/10/2020:

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool connected(int x, int y) {
    return find(x) == find(y);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
      id[i] = j;
    }
    return true;
```

```
  }
};

class Solution {
public:
  vector<int> findRedundantConnection(vector<vector<int>>& edges) {
    int n = edges.size();
    UnionFind uf(n + 1);
    vector<int> ret;
    for (auto& e : edges)
      if (uf.connected(e[0], e[1]))
        ret = e;
      else
        uf.merge(e[0], e[1]);
    return ret;
  }
};
```

679. 24 Game

Description

```
You have 4 cards each containing a number from 1 to 9. You need to judge whether
they could operated through *, /, +, -, (, ) to get the value of 24.

Example 1:
Input: [4, 1, 8, 7]
Output: True
Explanation: (8-4) * (7-1) = 24
Example 2:
Input: [1, 2, 1, 2]
Output: False
Note:
The division operator / represents real division, not integer division. For
example, 4 / (1 - 2/3) = 12.
Every operation done is between two numbers. In particular, we cannot use - as a
unary operator. For example, with [1, 1, 1, 1] as input, the expression -1 - 1 -
1 - 1 is not allowed.
You cannot concatenate numbers together. For example, if the input is [1, 2, 1,
2], we cannot write this as 12 + 12.
```

Solution

06/10/2020:

```
class Solution {
public:
  bool judgePoint24(vector<int>& nums) {
    unordered_set<string> seen;
    string operators = "+-*/";
    sort(nums.begin(), nums.end());
    do {
      int a = nums[0], b = nums[1], c = nums[2], d = nums[3];
      string key = to_string(a) + "," + to_string(b) + "," + to_string(c) + ","
+ to_string(d);
      if (seen.count(key) > 0) continue;
      seen.insert(key);
      for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
          for (int k = 0; k < 4; ++k) {
            vector<double> results;
            results.push_back(op(op(op(a, b, operators[i]), c, operators[j]), d,
operators[k]));
            results.push_back(op(op(a, b, operators[i]), op(c, d, operators[k]),
operators[j]));
            results.push_back(op(op(a, op(b, c, operators[j]), operators[i]), d,
operators[k]));
            results.push_back(op(a, op(op(b, c, operators[j]), d, operators[k]),
operators[i]));
            results.push_back(op(a, op(b, op(c, d, operators[k]),
operators[j]),operators[i]));
            for (auto& r : results)
              if (fabs(r - 24) < 1e-6)
                return true;
          }
        }
      }
    } while (next_permutation(nums.begin(), nums.end()));
    return false;
  }

  double op(double a, double b, char op) {
    switch(op) {
      case '+': return a + b;
      case '-': return a - b;
      case '*': return a * b;
      case '/': return b != 0 ? a / b : -1e6;
    }
    return 0;
  }
};
```

```cpp
class Solution {
public:
  bool judgePoint24(vector<int>& nums) {
    bool res = false;
    double eps = 0.001;
    vector<double> arr(nums.begin(), nums.end());
    dfs(arr, eps, res);
    return res;
  }

  void dfs(vector<double>& nums, double eps, bool& res) {
    if(res) return;
    if(nums.size()==1){
      if(abs(nums[0]-24)<eps) res=true;
      return;
    }
    for(int i=0;i<nums.size();i++){
      for(int j=0;j<i;j++){
        double p=nums[i], q=nums[j];
        vector<double>t{p+q, p-q, q-p, p*q};
        if (p > eps) t.push_back(q / p);
        if (q > eps) t.push_back(p / q);
        nums.erase(nums.begin() + i);
        nums.erase(nums.begin() + j);
        for (double d : t) {
          nums.push_back(d);
          dfs(nums, eps, res);
          nums.pop_back();
        }
        nums.insert(nums.begin() + j, q);
        nums.insert(nums.begin() + i, p);
      }
    }
  }
};
```

693. Binary Number with Alternating Bits

Description

```
Given a positive integer, check whether it has alternating bits: namely, if two
adjacent bits will always have different values.

Example 1:
```

```
Input: 5
Output: True
Explanation:
The binary representation of 5 is: 101
Example 2:

Input: 7
Output: False
Explanation:
The binary representation of 7 is: 111.
Example 3:

Input: 11
Output: False
Explanation:
The binary representation of 11 is: 1011.
Example 4:

Input: 10
Output: True
Explanation:
The binary representation of 10 is: 1010.
```

Solution

05/10/2020:

```cpp
class Solution {
public:
  bool hasAlternatingBits(int n) {
    int cnt = n & 1 ? 1 : 0;
    while (n != 0) {
      n >>= 1;
      if (n & 1) {
        ++cnt;
      } else {
        --cnt;
      }
      if (cnt < 0 || cnt > 1) return false;
    }
    return true;
  }
};
```

694. Number of Distinct Islands

Description

```
Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's
(representing land) connected 4-directionally (horizontal or vertical.) You may
assume all four edges of the grid are surrounded by water.

Count the number of distinct islands. An island is considered to be the same as
another if and only if one island can be translated (and not rotated or
reflected) to equal the other.

Example 1:

11000
11000
00011
00011
Given the above grid map, return 1.
Example 2:

11011
10000
00001
11011
Given the above grid map, return 3.

Notice that:
11
1
and
 1
11
are considered different island shapes, because we do not consider reflection /
rotation.
Note: The length of each dimension in the given grid does not exceed 50.
```

Solution

05/08/2020 Discussion:

1. Use a `UnionFind` set to find all the islands.
2. Eliminate the islands that are the "same" by hashing: translate all the islands to the top-left corner (subtract `offset` from the id for each cell). Then hash the sorted sequence of the cell. Note: the `offset` is obtained from the minimum of the row index `mini` and column index `minj`: `offset = mini * n + minj`.

```
class UnionFind {
private:
  vector<int> id;
```

```cpp
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    while (x != id[x]) {
      id[x] = id[id[x]];
      x = id[x];
    }
    return x;
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
      id[i] = j;
    }
    return true;
  }
};

class Solution {
public:
  int numDistinctIslands(vector<vector<int>>& grid) {
    if (grid.empty() || grid[0].empty()) return 0;
    int m = grid.size(), n = grid[0].size();

    UnionFind uf(m * n);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        int island = i * n + j;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1], neighbor = ni * n + nj;
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == 1) {
            bool merged = uf.merge(island, neighbor);
          }
        }
```

```
      }
    }

    unordered_map<int, vector<int>> components;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        int island = i * n + j;
        components[uf.find(island)].push_back(island);
      }
    }

    unordered_set<int> islands;
    for (auto& c : components) {
      sort(c.second.begin(), c.second.end());
      int mini = c.second.front() / n, minj = n;
      for (auto& i : c.second) minj = min(minj, i % n);
      int offset = mini * n + minj;
      long long hash = 1;
      const int MOD = 1e9 + 7;
      for (auto& i : c.second) hash = (hash * 31 + i - offset) % MOD;
      islands.insert(hash);
    }
    return islands.size();
  }
};
```

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
```

```cpp
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
      id[i] = j;
    }
    return true;
  }
};

class Solution {
private:
  int m, n;
public:
  int numDistinctIslands(vector<vector<int>>& grid) {
    if (grid.empty() || grid[0].empty()) return 0;
    m = grid.size(), n = grid[0].size();
    UnionFind uf(m * n);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        int island = i * n + j;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1], neighbor = ni * n + nj;
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == 1) {
            uf.merge(island, neighbor);
          }
        }
      }
    }

    unordered_map<int, vector<pair<int, int>>> components;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        components[uf.find(i * n + j)].emplace_back(i, j);
      }
    }

    unordered_set<int> seen;
    int cnt = 0;
    for (auto& c : components) {
      vector<int> hashes = hash(c.second);
      bool exists = false;
      for (auto& h : hashes) {
        if (seen.count(h)) {
          exists = true;
          break;
```

```
        }
      }
      for (auto& h : hashes) seen.insert(h);
      if (!exists) ++cnt;
    }
    return cnt;
  }

  vector<int> hash(vector<pair<int, int>> nums) {
    const int MOD = 1e9 + 7;
    int ops[4][2] = { {1, 1}, {-1, 1}, {-1, -1}, {1, -1} };
    int N = 1;
    vector<long long> hashes(N, 1);
    for (int d = 0; d < N; ++d) {
      int offset_x = INT_MAX, offset_y = INT_MAX;
      vector<pair<int, int>> t_nums(nums);
      for (auto& p : t_nums) {
        int& x = p.first;
        int& y = p.second;
        if (d >= 4) swap(x, y);
        x *= ops[d % 4][0];
        y *= ops[d % 4][1];
        offset_x = min(offset_x, x);
        offset_y = min(offset_y, y);
      }
      sort(t_nums.begin(), t_nums.end());
      for (auto& p : t_nums) {
        int& x = p.first;
        int& y = p.second;
        hashes[d] = (hashes[d] * 31 + (x - offset_x) * n + (y - offset_y)) %
MOD;
      }
    }
    return vector<int>(hashes.begin(), hashes.end());
  }
};
```

700. Search in a Binary Search Tree

Description

Given the root node of a binary search tree (BST) and a value. You need to find the node in the BST that the node's value equals the given value. Return the subtree rooted with that node. If such node doesn't exist, you should return NULL.

```
For example,

Given the tree:
        4
       / \
      2   7
     / \
    1   3

And the value to search: 2
You should return this subtree:

      2
     / \
    1   3
In the example above, if we want to search the value 5, since there is no node
with value 5, we should return NULL.

Note that an empty tree is represented by NULL, therefore you would see the
expected output (serialized tree format) as [], not null.
```

Solution

06/15/2020:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  TreeNode* searchBST(TreeNode* root, int val) {
    if (!root) return nullptr;
    if (root->val == val) return root;
    if (root->val > val) return searchBST(root->left, val);
    if (root->val < val) return searchBST(root->right, val);
    return nullptr;
  }
};
```

702. Search in a Sorted Array of Unknown Size

Description

```
Given an integer array sorted in ascending order, write a function to search
target in nums.  If target exists, then return its index, otherwise return -1.
However, the array size is unknown to you. You may only access the array using
an ArrayReader interface, where ArrayReader.get(k) returns the element of the
array at index k (0-indexed).

You may assume all integers in the array are less than 10000, and if you access
the array out of bounds, ArrayReader.get will return 2147483647.



Example 1:

Input: array = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
Example 2:

Input: array = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1


Note:

You may assume that all elements in the array are unique.
The value of each element in the array will be in the range [-9999, 9999].
```

Solution

04/26/2020:

```
/**
 * // This is the ArrayReader's API interface.
 * // You should not implement it, or speculate about its implementation
 * class ArrayReader {
 *   public:
 *     int get(int index);
 * };
 */

class Solution {
public:
```

```
  int search(const ArrayReader& reader, int target) {
    int lo = 0, hi = INT_MAX;
    while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      int val = reader.get(mid);
      if (val == target) {
        return mid;
      } else if (val == INT_MAX || val > target)  {
        hi = mid - 1;
      } else {
        lo = mid + 1;
      }
    }
    return -1;
  }
};
```

705. Design HashSet

Description

```
Design a HashSet without using any built-in hash table libraries.

To be specific, your design should include these functions:

add(value): Insert a value into the HashSet.
contains(value) : Return whether the value exists in the HashSet or not.
remove(value): Remove a value in the HashSet. If the value does not exist in the
HashSet, do nothing.

Example:

MyHashSet hashSet = new MyHashSet();
hashSet.add(1);
hashSet.add(2);
hashSet.contains(1);    // returns true
hashSet.contains(3);    // returns false (not found)
hashSet.add(2);
hashSet.contains(2);    // returns true
hashSet.remove(2);
hashSet.contains(2);    // returns false (already removed)

Note:

All values will be in the range of [0, 1000000].
The number of operations will be in the range of [1, 10000].
```

```
Please do not use the built-in HashSet library.
```

Solution

05/10/2020:

```
struct Node {
  int key;
  Node* next;
  Node (int k) : key(k), next(nullptr) {}
};

class MyHashSet {
private:
  vector<Node*> hashset;
  int N;

public:
  /** Initialize your data structure here. */
  MyHashSet() {
    N = 100003;
    hashset.resize(N, nullptr);
  }

  void add(int key) {
    Node* cur = hashset[key % N];
    if (cur == nullptr) {
      hashset[key % N] = new Node(key);
    } else {
      while (cur->key != key && cur->next != nullptr) {
        cur = cur->next;
      }
      if (cur->key != key) {
        cur->next = new Node(key);
      }
    }
  }

  void remove(int key) {
    Node* cur = hashset[key % N];
    if (cur == nullptr) return;
    if (cur->key == key) {
      hashset[key % N] = cur->next;
    } else {
      while (cur->next != nullptr && cur->next->key != key) {
        cur = cur->next;
      }
      if (cur->next != nullptr && cur->next->key == key) {
```

```
        cur->next = cur->next->next;
      }
    }
  }

  /** Returns true if this set contains the specified element */
  bool contains(int key) {
    Node* cur = hashset[key % N];
    if (cur == nullptr) return false;
    if (cur->key == key) {
      return true;
    } else {
      while (cur->key != key && cur->next != nullptr) {
        cur = cur->next;
      }
      return cur->key == key;
    }
  }
};

/**
 * Your MyHashSet object will be instantiated and called as such:
 * MyHashSet* obj = new MyHashSet();
 * obj->add(key);
 * obj->remove(key);
 * bool param_3 = obj->contains(key);
 */
```

706. Design HashMap

Description

```
Design a HashMap without using any built-in hash table libraries.

To be specific, your design should include these functions:

put(key, value) : Insert a (key, value) pair into the HashMap. If the value
already exists in the HashMap, update the value.
get(key): Returns the value to which the specified key is mapped, or -1 if this
map contains no mapping for the key.
remove(key) : Remove the mapping for the value key if this map contains the
mapping for the key.

Example:

MyHashMap hashMap = new MyHashMap();
```

```
hashMap.put(1, 1);
hashMap.put(2, 2);
hashMap.get(1);            // returns 1
hashMap.get(3);            // returns -1 (not found)
hashMap.put(2, 1);          // update the existing value
hashMap.get(2);            // returns 1
hashMap.remove(2);          // remove the mapping for 2
hashMap.get(2);            // returns -1 (not found)

Note:

All keys and values will be in the range of [0, 1000000].
The number of operations will be in the range of [1, 10000].
Please do not use the built-in HashMap library.
```

Solution

05/10/2020:

```
struct Node {
  int key, val;
  Node* next;
  Node (int k, int v) : key(k), val(v), next(nullptr) {}
};

class MyHashMap {
private:
  vector<Node*> hashmap;
  int N, n;

public:
  /** Initialize your data structure here. */
  MyHashMap() {
    N = 100003;
    n = 0;
    hashmap.resize(N, nullptr);
  }

  /** value will always be non-negative. */
  void put(int key, int value) {
    Node* cur = hashmap[key % N];
    if (cur == nullptr) {
      hashmap[key % N] = new Node(key, value);
    } else {
      while (cur->key != key && cur->next != nullptr) {
        cur = cur->next;
      }
      if (cur->key == key) {
```

```
        cur->val = value;
      } else if (cur->next == nullptr) {
        cur->next = new Node(key, value);
      }
    }
  }

  /** Returns the value to which the specified key is mapped, or -1 if this map
contains no mapping for the key */
  int get(int key) {
    Node* cur = hashmap[key % N];
    if (cur == nullptr) {
      return -1;
    } else if (cur->key == key) {
      return cur->val;
    } else {
      while (cur->key != key && cur->next != nullptr) {
        cur = cur->next;
      }
      return cur->key == key ? cur->val : -1;
    }
  }

  /** Removes the mapping of the specified value key if this map contains a
mapping for the key */
  void remove(int key) {
    Node* cur = hashmap[key % N];
    if (cur == nullptr) return;
    if (cur->key == key) {
      hashmap[key % N] = cur->next;
    } else {
      while (cur->next != nullptr && cur->next->key != key) {
        cur = cur->next;
      }
      if (cur->next != nullptr && cur->next->key == key) {
        cur->next = cur->next->next;
      }
    }
  }
};

/**
 * Your MyHashMap object will be instantiated and called as such:
 * MyHashMap* obj = new MyHashMap();
 * obj->put(key,value);
 * int param_2 = obj->get(key);
 * obj->remove(key);
 */
```

711. Number of Distinct Islands II

Description

Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

Count the number of distinct islands. An island is considered to be the same as another if they have the same shape, or have the same shape after rotation (90, 180, or 270 degrees only) or reflection (left/right direction or up/down direction).

Example 1:

```
11000
10000
00001
00011
```

Given the above grid map, return 1.

Notice that:

```
11
1
```

and

```
 1
11
```

are considered same island shapes. Because if we make a 180 degrees clockwise rotation on the first island, then two islands will have the same shapes.

Example 2:

```
11100
10001
01001
01110
```

Given the above grid map, return 2.

Here are the two distinct islands:

```
111
1
```

and

```
1
1
```

Notice that:

```
111
1
```

and

```
1
111
are considered same island shapes. Because if we flip the first array in the
up/down direction, then they have the same shapes.
Note: The length of each dimension in the given grid does not exceed 50.
```

Solution

05/08/2020 Discussion:

1. Use a `UnionFind` set to find all the islands.
2. Eliminate the islands that are the "same" by hashing: translate all the transformed islands to the top-left corner (subtract `offset_x` from the `x` and substract `offset_y` from `y` for each cell). Then hash the sorted sequence of the cell. Note: the `offset_x` and `offset_y` are obtained from the minimum of the row index and column index.

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
      id[i] = j;
    }
    return true;
  }
};

class Solution {
```

```cpp
private:
  int m, n;
public:
  int numDistinctIslands2(vector<vector<int>>& grid) {
    if (grid.empty() || grid[0].empty()) return 0;
    m = grid.size(), n = grid[0].size();
    UnionFind uf(m * n);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        int island = i * n + j;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1], neighbor = ni * n + nj;
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == 1) {
            uf.merge(island, neighbor);
          }
        }
      }
    }

    unordered_map<int, vector<pair<int, int>>> components;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (grid[i][j] == 0) continue;
        components[uf.find(i * n + j)].emplace_back(i, j);
      }
    }

    unordered_set<int> seen;
    int cnt = 0;
    for (auto& c : components) {
      vector<int> hashes = hash(c.second);
      bool exists = false;
      for (auto& h : hashes) {
        if (seen.count(h)) {
          exists = true;
          break;
        }
      }
      for (auto& h : hashes) seen.insert(h);
      if (!exists) ++cnt;
    }
    return cnt;
  }

  vector<int> hash(vector<pair<int, int>> nums) {
    const int MOD = 1e9 + 7;
    int ops[4][2] = { {1, 1}, {-1, 1}, {-1, -1}, {1, -1} };
```

```
    vector<long long> hashes(8, 1);
    for (int d = 0; d < 8; ++d) {
      int offset_x = INT_MAX, offset_y = INT_MAX;
      vector<pair<int, int>> t_nums(nums);
      for (auto& p : t_nums) {
        int& x = p.first;
        int& y = p.second;
        if (d >= 4) swap(x, y);
        x *= ops[d % 4][0];
        y *= ops[d % 4][1];
        offset_x = min(offset_x, x);
        offset_y = min(offset_y, y);
      }
      sort(t_nums.begin(), t_nums.end());
      for (auto& p : t_nums) {
        int& x = p.first;
        int& y = p.second;
        hashes[d] = (hashes[d] * 31 + (x - offset_x) * n + (y - offset_y)) %
MOD;
      }
    }
    return vector<int>(hashes.begin(), hashes.end());
  }
};
```

717. 1-bit and 2-bit Characters

Description

```
We have two special characters. The first character can be represented by one
bit 0. The second character can be represented by two bits (10 or 11).

Now given a string represented by several bits. Return whether the last
character must be a one-bit character or not. The given string will always end
with a zero.

Example 1:
Input:
bits = [1, 0, 0]
Output: True
Explanation:
The only way to decode it is two-bit character and one-bit character. So the
last character is one-bit character.
Example 2:
Input:
bits = [1, 1, 1, 0]
```

```
Output: False
Explanation:
The only way to decode it is two-bit character and two-bit character. So the
last character is NOT one-bit character.
Note:

1 <= len(bits) <= 1000.
bits[i] is always 0 or 1.
```

Solution

06/09/2020:

```cpp
class Solution {
public:
  bool isOneBitCharacter(vector<int>& bits) {
    int i = 0, n = bits.size();
    for (; i < n - 1; ++i) {
      if (bits[i] == 1) {
        ++i;
      }
    }
    return i == n - 1;
  }
};
```

720. Longest Word in Dictionary

Description

```
Given a list of strings words representing an English Dictionary, find the
longest word in words that can be built one character at a time by other words
in words. If there is more than one possible answer, return the longest word
with the smallest lexicographical order.

If there is no answer, return the empty string.
Example 1:
Input:
words = ["w","wo","wor","worl", "world"]
Output: "world"
Explanation:
The word "world" can be built one character at a time by "w", "wo", "wor", and
"worl".
Example 2:
Input:
```

```
words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]
Output: "apple"
Explanation:
Both "apply" and "apple" can be built from other words in the dictionary.
However, "apple" is lexicographically smaller than "apply".
Note:

All the strings in the input will only contain lowercase letters.
The length of words will be in the range [1, 1000].
The length of words[i] will be in the range [1, 30].
```

Solution

05/18/2020:

```
class Solution {
public:
  string longestWord(vector<string>& words) {
    unordered_set<string> s(words.begin(), words.end());
    sort(words.begin(), words.end(), [](const string& s1, const string& s2) {
      if (s1.size() == s2.size()) {
        return s1 < s2;
      }
      return s1.size() > s2.size();
    });
    for (auto& w : words) {
      bool ok = true;
      for (int i = 1; i < (int)w.size(); ++i) {
        if (s.count(w.substr(0, i)) == 0) {
          ok = false;
          break;
        }
      }
      if (ok) return w;
    }
    return "";
  }
};
```

721. Accounts Merge

Description

```
Given a list accounts, each element accounts[i] is a list of strings, where the
first element accounts[i][0] is a name, and the rest of the elements are emails
representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to
the same person if there is some email that is common to both accounts. Note
that even if two accounts have the same name, they may belong to different
people as people could have the same name. A person can have any number of
accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the
first element of each account is the name, and the rest of the elements are
emails in sorted order. The accounts themselves can be returned in any order.

Example 1:
Input:
accounts = [["John", "johnsmith@mail.com", "john00@mail.com"], ["John",
"johnnybravo@mail.com"], ["John", "johnsmith@mail.com",
"john_newyork@mail.com"], ["Mary", "mary@mail.com"]]
Output: [["John", 'john00@mail.com', 'john_newyork@mail.com',
'johnsmith@mail.com'],  ["John", "johnnybravo@mail.com"], ["Mary",
"mary@mail.com"]]
Explanation:
The first and third John's are the same person as they have the common email
"johnsmith@mail.com".
The second John and Mary are different people as none of their email addresses
are used by other accounts.
We could return these lists in any order, for example the answer [['Mary',
'mary@mail.com'], ['John', 'johnnybravo@mail.com'],
['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']]
would still be accepted.
Note:

The length of accounts will be in the range [1, 1000].
The length of accounts[i] will be in the range [1, 10].
The length of accounts[i][j] will be in the range [1, 30].
```

Solution

06/10/2020:

Define `UnionFind`:

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;
```

```
public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool connected(int x, int y) {
    return find(x) == find(y);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      id[j] = i;
      sz[i] += sz[j];
    } else {
      id[i] = j;
      sz[j] += sz[i];
    }
    return true;
  }
};
```

```
class Solution {
public:
  vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {
    int id = 0;
    UnionFind uf(10000);
    unordered_map<string, string> email_user;
    unordered_map<string, int> email_id;
    unordered_map<int, string> id_email;
    for (auto& emails : accounts) {
      for (int i = 1; i < (int)emails.size(); ++i) {
        email_user[emails[i]] = emails[0];
        if (email_id.count(emails[i]) == 0) {
          email_id[emails[i]] = id++;
          id_email[id - 1] = emails[i];
        }
        if (i > 1) uf.merge(email_id[emails[i - 1]], email_id[emails[i]]);
      }
    }
```

```
    unordered_map<int, vector<int>> connected_components;
    for (int i = 0; i < id; ++i) connected_components[uf.find(i)].push_back(i);

    vector<vector<string>> ret;
    for (auto& c : connected_components) {
      vector<string> account;
      account.push_back(email_user[id_email[c.first]]);
      for (auto& i : c.second) account.push_back(id_email[i]);
      sort(account.begin() + 1, account.end());
      ret.push_back(account);
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {
    int n = accounts.size();
    UnionFind uf(n);
    for (int i = 0; i < n; ++i) {
      unordered_set<string> accounts_set(accounts[i].begin() + 1,
accounts[i].end());
      for (int j = i + 1; j < n; ++j)
        if (accounts[i][0] == accounts[j][0])
          for (int k = 1; !uf.connected(i, j) && k < (int)accounts[j].size();
++k)
            if (accounts_set.count(accounts[j][k]) > 0)
              uf.merge(i, j);
    }

    unordered_map<int, vector<int>> connected_components;
    for (int i = 0; i < n; ++i)
      if (connected_components.count(uf.find(i)) == 0)
        connected_components[uf.find(i)] = {i};
      else
        connected_components[uf.find(i)].push_back(i);

    vector<vector<string>> ret;
    for (auto& c : connected_components) {
      unordered_set<string> acct;
      for (auto i : c.second)
        for (int j = 1; j < (int)accounts[i].size(); ++j)
          acct.insert(accounts[i][j]);
      vector<string> sorted_acct(acct.begin(), acct.end());
      sort(sorted_acct.rbegin(), sorted_acct.rend());
      sorted_acct.push_back(accounts[c.first][0]);
```

```
      reverse(sorted_acct.begin(), sorted_acct.end());
      ret.push_back(sorted_acct);
    }
    return ret;
  }
};
```

733. Flood Fill

Description

```
An image is represented by a 2-D array of integers, each integer representing
the pixel value of the image (from 0 to 65535).

Given a coordinate (sr, sc) representing the starting pixel (row and column) of
the flood fill, and a pixel value newColor, "flood fill" the image.

To perform a "flood fill", consider the starting pixel, plus any pixels
connected 4-directionally to the starting pixel of the same color as the
starting pixel, plus any pixels connected 4-directionally to those pixels (also
with the same color as the starting pixel), and so on. Replace the color of all
of the aforementioned pixels with the newColor.

At the end, return the modified image.

Example 1:
Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation:
From the center of the image (with position (sr, sc) = (1, 1)), all pixels
connected
by a path of the same color as the starting pixel are colored with the new
color.
Note the bottom corner is not colored 2, because it is not 4-directionally
connected
to the starting pixel.
Note:

The length of image and image[0] will be in the range [1, 50].
The given starting pixel will satisfy 0 <= sr < image.length and 0 <= sc <
image[0].length.
The value of each color in image[i][j] and newColor will be an integer in [0,
65535].
```

Solution

05/11/2020:

```
class Solution {
public:
  vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int
newColor) {
    if (image.empty() || image[0].empty()) return image;
    if (image[sr][sc] == newColor) return image;
    int m = image.size(), n = image[0].size();
    int oldColor = image[sr][sc];
    image[sr][sc] = newColor;
    queue<pair<int, int>> q;
    q.emplace(sr, sc);
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    while (!q.empty()) {
      int sz = q.size();
      for (int s = 0; s < sz; ++s) {
        pair<int, int> cur = q.front(); q.pop();
        int i = cur.first, j = cur.second;
        for (int d = 0; d < 4; ++d) {
          int ni = i + dir[d][0], nj = j + dir[d][1];
          if (ni >= 0 && ni < m && nj >= 0 && nj < n && image[ni][nj] ==
oldColor) {
            image[ni][nj] = newColor;
            q.emplace(ni, nj);
          }
        }
      }
    }
    return image;
  }
};
```

734. Sentence Similarity

Description

```
Given two sentences words1, words2 (each represented as an array of strings),
and a list of similar word pairs pairs, determine if two sentences are similar.

For example, "great acting skills" and "fine drama talent" are similar, if the
similar word pairs are pairs = [["great", "fine"], ["acting","drama"],
["skills","talent"]].
```

```
Note that the similarity relation is not transitive. For example, if "great" and
"fine" are similar, and "fine" and "good" are similar, "great" and "good" are
not necessarily similar.

However, similarity is symmetric. For example, "great" and "fine" being similar
is the same as "fine" and "great" being similar.

Also, a word is always similar with itself. For example, the sentences words1 =
["great"], words2 = ["great"], pairs = [] are similar, even though there are no
specified similar word pairs.

Finally, sentences can only be similar if they have the same number of words. So
a sentence like words1 = ["great"] can never be similar to words2 =
["doubleplus","good"].

Note:

The length of words1 and words2 will not exceed 1000.
The length of pairs will not exceed 2000.
The length of each pairs[i] will be 2.
The length of each words[i] and pairs[i][j] will be in the range [1, 20].
```

Solution

06/10/2020:

```
class Solution {
public:
  bool areSentencesSimilar(vector<string>& words1, vector<string>& words2,
vector<vector<string>>& pairs) {
    if (words1.size() != words2.size()) return false;
    bool isSimilar = true;
    unordered_map<string, unordered_set<string>> similarWords;
    for (auto& p : pairs) {
      similarWords[p[0]].insert(p[1]);
      similarWords[p[1]].insert(p[0]);
    }
    for (int i = 0; i < words1.size(); ++i)
      if (words1[i] != words2[i])
        if (similarWords[words1[i]].count(words2[i]) == 0 &&
similarWords[words2[i]].count(words1[i]) == 0)
          return false;
    return true;
  }
};
```

05/20/2020:

```
class Solution {
public:
  bool areSentencesSimilar(vector<string>& words1, vector<string>& words2,
vector<vector<string>>& pairs) {
    if (words1.empty() && words2.empty()) return true;
    if (words1.size() != words2.size()) return false;
    unordered_map<string, unordered_set<string>> similarWords;
    for (auto& p : pairs) {
      if (p.empty()) continue;
      similarWords[p[0]].insert(p[1]);
      similarWords[p[1]].insert(p[0]);
    }
    int n = words1.size();
    for (int i = 0; i < n; ++i) {
      bool isSimilar = false;
      if (words1[i] == words2[i]) {
        isSimilar = true;
      } else {
        if (similarWords[words1[i]].count(words2[i]) > 0)
          isSimilar = true;
      }
      if (isSimilar == false) return false;
    }
    return true;
  }
};
```

737. Sentence Similarity II

Description

Given two sentences words1, words2 (each represented as an array of strings), and a list of similar word pairs pairs, determine if two sentences are similar.

For example, words1 = ["great", "acting", "skills"] and words2 = ["fine", "drama", "talent"] are similar, if the similar word pairs are pairs = [["great", "good"], ["fine", "good"], ["acting","drama"], ["skills","talent"]].

Note that the similarity relation is transitive. For example, if "great" and "good" are similar, and "fine" and "good" are similar, then "great" and "fine" are similar.

Similarity is also symmetric. For example, "great" and "fine" being similar is the same as "fine" and "great" being similar.

```
Also, a word is always similar with itself. For example, the sentences words1 =
["great"], words2 = ["great"], pairs = [] are similar, even though there are no
specified similar word pairs.

Finally, sentences can only be similar if they have the same number of words. So
a sentence like words1 = ["great"] can never be similar to words2 =
["doubleplus","good"].

Note:

The length of words1 and words2 will not exceed 1000.
The length of pairs will not exceed 2000.
The length of each pairs[i] will be 2.
The length of each words[i] and pairs[i][j] will be in the range [1, 20].
```

Solution

06/10/2020:

```
class UnionFind {
private:
  vector<int> id;
  vector<int> sz;

public:
  UnionFind(int n) {
    id.resize(n);
    iota(id.begin(), id.end(), 0);
    sz.resize(n, 1);
  }

  int find(int x) {
    if (x == id[x]) return x;
    return id[x] = find(id[x]);
  }

  bool connected(int x, int y) {
    return find(x) == find(y);
  }

  bool merge(int x, int y) {
    int i = find(x), j = find(y);
    if (i == j) return false;
    if (sz[i] > sz[j]) {
      sz[i] += sz[j];
      id[j] = i;
    } else {
      sz[j] += sz[i];
```

```
      id[i] = j;
    }
    return true;
  }
};

class Solution {
public:
  bool areSentencesSimilarTwo(vector<string>& words1, vector<string>& words2,
vector<vector<string>>& pairs) {
    if (words1.size() != words2.size()) return false;
    UnionFind uf(6000);
    unordered_map<string, int> word_id;
    int id = 0;
    for (auto& w : words1) assign_id(w, id, word_id);
    for (auto& w : words2) assign_id(w, id, word_id);
    for (auto& p : pairs) {
      assign_id(p[0], id, word_id);
      assign_id(p[1], id, word_id);
      uf.merge(word_id[p[0]], word_id[p[1]]);
    }
    for (int i = 0; i < (int)words1.size(); ++i)
      if (words1[i] != words2[i] && !uf.connected(word_id[words1[i]],
word_id[words2[i]]))
        return false;
    return true;
  }

  void assign_id(const string& word, int& id, unordered_map<string, int>&
word_id) {
    if (word_id.count(word) == 0) word_id[word] = id++;
  }
};
```

748. Shortest Completing Word

Description

Find the minimum length word from a given dictionary words, which has all the letters from the string licensePlate. Such a word is said to complete the given string licensePlate

Here, for letters we ignore case. For example, "P" on the licensePlate still matches "p" on the word.

```
It is guaranteed an answer exists. If there are multiple answers, return the one
that occurs first in the array.

The license plate might have the same letter occurring multiple times. For
example, given a licensePlate of "PP", the word "pair" does not complete the
licensePlate, but the word "supper" does.

Example 1:
Input: licensePlate = "1s3 PSt", words = ["step", "steps", "stripe", "stepple"]
Output: "steps"
Explanation: The smallest length word that contains the letters "S", "P", "S",
and "T".
Note that the answer is not "step", because the letter "s" must occur in the
word twice.
Also note that we ignored case for the purposes of comparing whether a letter
exists in the word.
Example 2:
Input: licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
Output: "pest"
Explanation: There are 3 smallest length words that contains the letters "s".
We return the one that occurred first.
Note:
licensePlate will be a string with length in range [1, 7].
licensePlate will contain digits, spaces, or letters (uppercase or lowercase).
words will have a length in the range [10, 1000].
Every words[i] will consist of lowercase letters, and have length in range [1,
15].
```

Solution

05/17/2020:

```
class Solution {
public:
  string shortestCompletingWord(string licensePlate, vector<string>& words) {
    vector<int> trimmedLicensePlate(26, 0);
    for (auto& l : licensePlate) {
      if (isalpha(l)) {
        ++trimmedLicensePlate[tolower(l) - 'a'];
      }
    }
    vector<pair<string, int>> validWords;
    for (int k = 0; k < (int)words.size(); ++k) {
      string w = words[k];
      vector<int> cnt(26, 0);
      for (auto& c : w) {
        ++cnt[c - 'a'];
      }
```

```
      bool isComplete = true;
      for (int i = 0; i < 26; ++i) {
        if (cnt[i] < trimmedLicensePlate[i]) {
          isComplete = false;
          break;
        }
      }
      if (isComplete) validWords.emplace_back(w, k);
    }
    sort(validWords.begin(), validWords.end(), [](pair<string, int>& p1,
pair<string, int>& p2) {
      if (p1.first.size() == p2.first.size()) {
        return p1.second < p2.second;
      }
      return p1.first.size() < p2.first.size();
    });
    return validWords.front().first;
  }
};
```

758. Bold Words in String

Description

```
Given a set of keywords words and a string S, make all appearances of all
keywords in S bold. Any letters between <b> and </b> tags become bold.

The returned string should use the least number of tags possible, and of course
the tags should form a valid combination.

For example, given that words = ["ab", "bc"] and S = "aabcd", we should return
"a<b>abc</b>d". Note that returning "a<b>a<b>b</b>c</b>d" would use more tags,
so it is incorrect.

Constraints:

words has length in range [0, 50].
words[i] has length in range [1, 10].
S has length in range [0, 500].
All characters in words[i] and S are lowercase letters.
Note: This question is the same as 616: https://leetcode.com/problems/add-bold-
tag-in-string/
```

Solution

06/09/2020:

```cpp
class Solution {
public:
  string boldWords(vector<string>& dict, string s) {
    unordered_set<string> dicts(dict.begin(), dict.end());
    vector<int> lengths;
    for (auto& d : dict) lengths.push_back(d.size());
    sort(lengths.rbegin(), lengths.rend());
    int n = s.size();
    vector<bool> isBold(n, false);
    for (int i = 0; i < n; ++i) {
      bool wrap = false;
      int wrapLength = 0;
      for (auto& len : lengths) {
        if (i + len <= n && dicts.count(s.substr(i, len)) > 0) {
          wrap = true;
          wrapLength = len;
          break;
        }
      }
      if (wrap) {
        for (int j = i; j < i + wrapLength; ++j)
          isBold[j] = true;
      }
    }
    string ret;
    int last = 0;
    for (int i = 0; i < n;) {
      if (isBold[i]) {
        while (i + 1 < n && isBold[i + 1]) ++i;
        ret += "<b>";
        ret += s.substr(last, i - last + 1);
        ret += "</b>";
        last = ++i;
      } else {
        ret += s[i];
        last = ++i;
      }
    }
    return ret;
  }
};
```

773. Sliding Puzzle

Description

```
On a 2x3 board, there are 5 tiles represented by the integers 1 through 5, and
an empty square represented by 0.

A move consists of choosing 0 and a 4-directionally adjacent number and swapping
it.

The state of the board is solved if and only if the board is [[1,2,3],[4,5,0]].

Given a puzzle board, return the least number of moves required so that the
state of the board is solved. If it is impossible for the state of the board to
be solved, return -1.

Examples:

Input: board = [[1,2,3],[4,0,5]]
Output: 1
Explanation: Swap the 0 and the 5 in one move.
Input: board = [[1,2,3],[5,4,0]]
Output: -1
Explanation: No number of moves will make the board solved.
Input: board = [[4,1,2],[5,0,3]]
Output: 5
Explanation: 5 is the smallest number of moves that solves the board.
An example path:
After move 0: [[4,1,2],[5,0,3]]
After move 1: [[4,1,2],[0,5,3]]
After move 2: [[0,1,2],[4,5,3]]
After move 3: [[1,0,2],[4,5,3]]
After move 4: [[1,2,0],[4,5,3]]
After move 5: [[1,2,3],[4,5,0]]
Input: board = [[3,2,4],[1,5,0]]
Output: 14
Note:

board will be a 2 x 3 array as described above.
board[i][j] will be a permutation of [0, 1, 2, 3, 4, 5].
```

Solution

05/08/2020 Discussion:

This is the C++ version solution for one of the assignments of the course Algorithms (Part 1):

```
class Board {
public:
  vector<vector<int>> board;
  pair<int, int> zeroTile;
  int rows, cols, manhattanDist, moves, priority;

  Board(const vector<vector<int>>& board) {
    this->board = board;
    rows = board.size();
    cols = board[0].size();
    zeroTile = findZeroTile();
    manhattanDist = manhattan();
    moves = 0;
  }

  bool operator<(const Board& that) const { return this->priority <
that.priority; }

  bool operator>(const Board& that) const { return this->priority >
that.priority; }

  bool operator!=(const Board& that) const {
    for (int i = 0; i < rows; ++i)
      for (int j = 0; j < cols; ++j)
        if (that.board[i][j] != board[i][j])
          return true;
    return false;
  }

  pair<int, int> findZeroTile() {
    for (int i = 0; i < rows; ++i)
      for (int j = 0; j < cols; ++j)
        if (board[i][j] == 0)
          return {i, j};
    return {-1, -1};
  }

  int manhattan() {
    int d = 0;
    for (int i = 0; i < rows; ++i)
      for (int j = 0; j < cols; ++j)
        if (board[i][j] != 0)
          d += abs((board[i][j] - 1) / cols - i) + abs((board[i][j] - 1) % cols
- j);
    return d;
  }

  void updateMoves(int moves) {
```

```
    this->moves = moves;
    priority = manhattanDist + moves;
  }

  vector<Board> neighbor() {
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    int i = zeroTile.first, j = zeroTile.second;
    vector<Board> neighborBoards;
    for (int d = 0; d < 4; ++d) {
      int ni = dir[d][0] + i, nj = dir[d][1] + j;
      vector<vector<int>> newBoard(board);
      if (ni >= 0 && ni < rows && nj >= 0 && nj < cols) {
        swap(newBoard[i][j], newBoard[ni][nj]);
        neighborBoards.push_back(Board(newBoard));
        neighborBoards.back().updateMoves(moves + 1);
      }
    }
    return neighborBoards;
  }

  Board twin () {
    for (int i = 0; i < rows; ++i) {
      for (int j = 0; j < cols - 1; ++j) {
        if (board[i][j] != 0 && board[i][j + 1] != 0) {
          vector<vector<int>> twinBoard(board);
          swap(twinBoard[i][j], twinBoard[i][j + 1]);
          return Board(twinBoard);
        }
      }
    }
    return Board(board);
  }
};

class Solution {
public:
  int slidingPuzzle(vector<vector<int>>& board) {
    priority_queue<pair<Board, Board>, vector<pair<Board, Board>>,
greater<pair<Board, Board>>> pq, pqTwin;
    Board initialBoard(board), twinBoard = initialBoard.twin();
    pq.emplace(initialBoard, initialBoard);
    pqTwin.emplace(twinBoard, twinBoard);
    while (!pq.empty() && pq.top().first.manhattanDist > 0 && !pqTwin.empty() &&
pqTwin.top().first.manhattanDist > 0) {
      pair<Board, Board> cur = pq.top(); pq.pop();
      for(auto& p : cur.first.neighbor())
        if (p != cur.second) pq.emplace(p, cur.first);
      cur = pqTwin.top(); pqTwin.pop();
      for(auto& p : cur.first.neighbor())
```

```
        if (p != cur.second) pqTwin.emplace(p, cur.first);
    }
    return pqTwin.top().first.manhattanDist == 0 ? -1 : pq.top().first.moves;
  }
};
```

787. Cheapest Flights Within K Stops

Description

```
There are n cities connected by m flights. Each flight starts from city u and
arrives at v with a price w.

Now given all the cities and flights, together with starting city src and the
destination dst, your task is to find the cheapest price from src to dst with up
to k stops. If there is no such route, output -1.

Example 1:
Input:
n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]]
src = 0, dst = 2, k = 1
Output: 200
Explanation:
The graph looks like this:


The cheapest price from city 0 to city 2 with at most 1 stop costs 200, as
marked red in the picture.
Example 2:
Input:
n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]]
src = 0, dst = 2, k = 0
Output: 500
Explanation:
The graph looks like this:


The cheapest price from city 0 to city 2 with at most 0 stop costs 500, as
marked blue in the picture.


Constraints:

The number of nodes n will be in range [1, 100], with nodes labeled from 0 to n
- 1.
The size of flights will be in range [0, n * (n - 1) / 2].
```

```
The format of each flight will be (src, dst, price).
The price of each flight will be in the range [1, 10000].
k is in the range of [0, n - 1].
There will not be any duplicated flights or self cycles.
```

Solution

06/14/2020:

```
typedef tuple<int, int, int> ti;
class Solution {
public:
  int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst,
int K) {
    vector<vector<pair<int, int>>> adj(n);
    for(const auto& f : flights) adj[f[0]].emplace_back(f[1], f[2]);
    priority_queue<ti, vector<ti>, greater<ti>> pq;
    pq.emplace(0, src, K + 1);
    while(!pq.empty()) {
      auto [cost, u, stops] = pq.top(); pq.pop();
      if(u == dst) return cost;
      if(!stops) continue;
      for(auto& to : adj[u]) {
        auto [v, w] = to;
        pq.emplace(cost + w, v, stops - 1);
      }
    }
    return -1;
  }
};
```

819. Most Common Word

Description

```
Given a paragraph and a list of banned words, return the most frequent word that
is not in the list of banned words.  It is guaranteed there is at least one word
that isn't banned, and that the answer is unique.

Words in the list of banned words are given in lowercase, and free of
punctuation.  Words in the paragraph are not case sensitive.  The answer is in
lowercase.
```

```
Example:

Input:
paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
banned = ["hit"]
Output: "ball"
Explanation:
"hit" occurs 3 times, but it is a banned word.
"ball" occurs twice (and no other word does), so it is the most frequent non-
banned word in the paragraph.
Note that words in the paragraph are not case sensitive,
that punctuation is ignored (even if adjacent to words, such as "ball,"),
and that "hit" isn't the answer even though it occurs more because it is banned.


Note:

1 <= paragraph.length <= 1000.
0 <= banned.length <= 100.
1 <= banned[i].length <= 10.
The answer is unique, and written in lowercase (even if its occurrences in
paragraph may have uppercase symbols, and even if it is a proper noun.)
paragraph only consists of letters, spaces, or the punctuation symbols !?',;.
There are no hyphens or hyphenated words.
Words only consist of letters, never apostrophes or other punctuation symbols.
```

Solution

05/10/2020:

```
class Solution {
public:
  string mostCommonWord(string paragraph, vector<string>& banned) {
    for (auto& c : paragraph) c = isalpha(c) ? tolower(c) : ' ';
    unordered_set<string> b(banned.begin(), banned.end());
    unordered_map<string, int> mp;
    string ret, s;
    int freq = 0;
    istringstream iss(paragraph);
    while (iss >> s) {
      if (b.count(s) == 0 && ++mp[s] > freq) {
        freq = mp[s];
        ret = s;
      }
    }
    return ret;
  }
};
```

844. Backspace String Compare

Description

```
Given two strings S and T, return if they are equal when both are typed into
empty text editors. # means a backspace character.

Example 1:

Input: S = "ab#c", T = "ad#c"
Output: true
Explanation: Both S and T become "ac".
Example 2:

Input: S = "ab##", T = "c#d#"
Output: true
Explanation: Both S and T become "".
Example 3:

Input: S = "a##c", T = "#a#c"
Output: true
Explanation: Both S and T become "c".
Example 4:

Input: S = "a#c", T = "b"
Output: false
Explanation: S becomes "c" while T becomes "b".
Note:

1 <= S.length <= 200
1 <= T.length <= 200
S and T only contain lowercase letters and '#' characters.
Follow up:

Can you solve it in O(N) time and O(1) space?
```

Solution

02/05/2020:

```
class Solution {
public:
  bool backspaceCompare(string S, string T) {
    string s, t;
    for (auto& c : S) {
      if (c == '#') {
        if (!s.empty()) {
```

```
          s.pop_back();
        }
      } else {
        s.push_back(c);
      }
    }
    for (auto& c : T) {
      if (c == '#') {
        if (!t.empty()) {
          t.pop_back();
        }
      } else {
        t.push_back(c);
      }
    }
    return s == t;
  }
};
```

868. Binary Gap

Description

```
Given a positive integer N, find and return the longest distance between two
consecutive 1's in the binary representation of N.

If there aren't two consecutive 1's, return 0.



Example 1:

Input: 22
Output: 2
Explanation:
22 in binary is 0b10110.
In the binary representation of 22, there are three ones, and two consecutive
pairs of 1's.
The first consecutive pair of 1's have distance 2.
The second consecutive pair of 1's have distance 1.
The answer is the largest of these two distances, which is 2.
Example 2:

Input: 5
Output: 2
Explanation:
```

```
5 in binary is 0b101.
Example 3:

Input: 6
Output: 1
Explanation:
6 in binary is 0b110.
Example 4:

Input: 8
Output: 0
Explanation:
8 in binary is 0b1000.
There aren't any consecutive pairs of 1's in the binary representation of 8, so
we return 0.


Note:

1 <= N <= 10^9
```

Solution

05/10/2020:

```
class Solution {
public:
  int binaryGap(int N) {
    if (N == 0) return 0;
    int ret = INT_MIN;
    while ((N & 1) == 0) {
      N >>= 1;
    }
    int cnt = 0;
    while (N != 0) {
      N >>= 1;
      if ((N & 1) == 1) {
        ret = max(ret, cnt);
        cnt = 0;
      } else {
        ++cnt;
      }
    }
    return ret == INT_MIN ? 0 : ret + 1;
  }
};
```

917. Reverse Only Letters

Description

```
Given a string S, return the "reversed" string where all characters that are not
a letter stay in the same place, and all letters reverse their positions.



Example 1:

Input: "ab-cd"
Output: "dc-ba"
Example 2:

Input: "a-bC-dEf-ghIj"
Output: "j-Ih-gfE-dCba"
Example 3:

Input: "Test1ng-Leet=code-Q!"
Output: "Qedo1ct-eeLg=ntse-T!"


Note:

S.length <= 100
33 <= S[i].ASCIIcode <= 122
S doesn't contain \ or "
```

Solution

05/11/2020:

```
class Solution {
public:
  string reverseOnlyLetters(string S) {
    int n = S.size(), l = 0, r = n - 1;
    while (l < r) {
      if (isalpha(S[l]) && isalpha(S[r])) {
        swap(S[l++], S[r--]);
      } else if (!isalpha(S[l])) {
        ++l;
      } else { // if (!isalpha(S[r]))
        --r;
      }
    }
    return S;
```

```
  }
};
```

```
class Solution {
public:
  string reverseOnlyLetters(string S) {
    int n = S.size(), l = 0, r = n - 1;
    string ret(S);
    while (l < n && r >= 0) {
      for (; l < n && !isalpha(ret[l]); ++l);
      for (; r >= 0 && !isalpha(S[r]); --r);
      if (l < n && r >= 0) ret[l++] = S[r--];
    }
    return ret;
  }
};
```

973. K Closest Points to Origin

Description

```
We have a list of points on the plane.  Find the K closest points to the origin
(0, 0).

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order.  The answer is guaranteed to be unique
(except for the order that it is in.)



Example 1:

Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just
[[-2,2]].
Example 2:

Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
```

```
(The answer [[-2,4],[3,3]] would also be accepted.)


Note:

1 <= K <= points.length <= 10000
-10000 < points[i][0] < 10000
-10000 < points[i][1] < 10000
```

Solution

05/28/2020:

```
class Solution {
public:
  vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {
    priority_queue<pair<double, vector<int>>, vector<pair<double, vector<int>>>,
greater<pair<double, vector<int>>>> q;
    for (auto& p : points) q.push({hypot(p[0], p[1]), p});
    vector<vector<int>> ret;
    while (K-- > 0) {
      pair<double, vector<int>> cur = q.top(); q.pop();
      ret.push_back(cur.second);
    }
    return ret;
  }
};
```

986. Interval List Intersections

Description

```
Given two lists of closed intervals, each list of intervals is pairwise disjoint
and in sorted order.

Return the intersection of these two interval lists.

(Formally, a closed interval [a, b] (with a <= b) denotes the set of real
numbers x with a <= x <= b.  The intersection of two closed intervals is a set
of real numbers that is either empty, or can be represented as a closed
interval.  For example, the intersection of [1, 3] and [2, 4] is [2, 3].)



Example 1:
```

```
Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],[15,24],[25,26]]
Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
Reminder: The inputs and the desired output are lists of Interval objects, and
not arrays or lists.


Note:

0 <= A.length < 1000
0 <= B.length < 1000
0 <= A[i].start, A[i].end, B[i].start, B[i].end < 10^9
NOTE: input types have been changed on April 15, 2019. Please reset to default
code definition to get new method signature.
```

Solution

05/23/2020:

```
class Solution {
public:
  vector<vector<int>> intervalIntersection(vector<vector<int>>& A,
vector<vector<int>>& B) {
    auto ia = A.begin();
    auto ib = B.begin();
    vector<vector<int>> ret;
    for (; ia != A.end() && ib != B.end();) {
      if ((*ia)[1] <= (*ib)[1]) {
        if ((*ib)[0] <= (*ia)[1]) {
          ret.push_back({max((*ia)[0], (*ib)[0]), (*ia)[1]});
        }
        ++ia;
      } else {
        if ((*ib)[1] >= (*ia)[0]) {
          ret.push_back({max((*ia)[0], (*ib)[0]), (*ib)[1]});
        }
        ++ib;
      }
    }
    return ret;
  }
};
```

988. Smallest String Starting From Leaf

Description

```
Given the root of a binary tree, each node has a value from 0 to 25 representing
the letters 'a' to 'z': a value of 0 represents 'a', a value of 1 represents
'b', and so on.

Find the lexicographically smallest string that starts at a leaf of this tree
and ends at the root.

(As a reminder, any shorter prefix of a string is lexicographically smaller: for
example, "ab" is lexicographically smaller than "aba".  A leaf of a node is a
node that has no children.)



Example 1:



Input: [0,1,2,3,4,3,4]
Output: "dba"
Example 2:



Input: [25,1,3,1,3,0,2]
Output: "adz"
Example 3:



Input: [2,2,1,null,1,0,null,0]
Output: "abc"


Note:

The number of nodes in the given tree will be between 1 and 8500.
Each node in the tree will have a value between 0 and 25.
```

Solution

06/15/2020:

```
/**
```

```
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
  string smallestFromLeaf(TreeNode* root) {
    if (!root) return "";
    string ret(8500, 'z'), s;
    backtrack(s, ret, root);
    return ret;
  }

  void backtrack(string& s, string& ret, TreeNode* root) {
    if (!root) return;
    s.push_back(root->val + 'a');
    if (!root->left && !root->right) ret = min(string(s.rbegin(), s.rend()),
ret);
    backtrack(s, ret, root->left);
    backtrack(s, ret, root->right);
    s.pop_back();
  }
};
```

997. Find the Town Judge

Description

```
In a town, there are N people labelled from 1 to N.  There is a rumor that one
of these people is secretly the town judge.

If the town judge exists, then:

The town judge trusts nobody.
Everybody (except for the town judge) trusts the town judge.
There is exactly one person that satisfies properties 1 and 2.
You are given trust, an array of pairs trust[i] = [a, b] representing that the
person labelled a trusts the person labelled b.
```

```
If the town judge exists and can be identified, return the label of the town
judge.  Otherwise, return -1.



Example 1:

Input: N = 2, trust = [[1,2]]
Output: 2
Example 2:

Input: N = 3, trust = [[1,3],[2,3]]
Output: 3
Example 3:

Input: N = 3, trust = [[1,3],[2,3],[3,1]]
Output: -1
Example 4:

Input: N = 3, trust = [[1,2],[2,3]]
Output: -1
Example 5:

Input: N = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]]
Output: 3


Note:

1 <= N <= 1000
trust.length <= 10000
trust[i] are all different
trust[i][0] != trust[i][1]
1 <= trust[i][0], trust[i][1] <= N
```

Solution

05/10/2020:

```
class Solution {
public:
  int findJudge(int N, vector<vector<int>>& trust) {
    if (N == 1 && trust.empty()) return 1;
    if (trust.empty() || trust[0].empty()) return -1;
    unordered_map<int, vector<int>> r;
    unordered_set<int> truster;
    for (auto& t : trust) {
      r[t[1]].push_back(t[0]);
```

```
      truster.insert(t[0]);
    }
    int judge = -1;
    int cnt = 0;
    for (auto& s : r) {
      if ((int)s.second.size() == N - 1 && truster.count(s.first) == 0) {
        ++cnt;
        judge = s.first;
      }
    }
    return cnt == 1 ? judge : -1;
  }
};
```

1029. Two City Scheduling

Description

```
There are 2N people a company is planning to interview. The cost of flying the
i-th person to city A is costs[i][0], and the cost of flying the i-th person to
city B is costs[i][1].

Return the minimum cost to fly every person to a city such that exactly N people
arrive in each city.



Example 1:

Input: [[10,20],[30,200],[400,50],[30,20]]
Output: 110
Explanation:
The first person goes to city A for a cost of 10.
The second person goes to city A for a cost of 30.
The third person goes to city B for a cost of 50.
The fourth person goes to city B for a cost of 20.

The total minimum cost is 10 + 30 + 50 + 20 = 110 to have half the people
interviewing in each city.


Note:

1 <= costs.length <= 100
It is guaranteed that costs.length is even.
1 <= costs[i][0], costs[i][1] <= 1000
```

Solution

06/03/2020:

Backtracking (TLE):

```
class Solution {
public:
  int ret, numOfCityA;
  int twoCitySchedCost(vector<vector<int>>& costs) {
    int n = costs.size(), totalCost = 0;
    ret = INT_MAX, numOfCityA = 0;
    backtrack(0, n, numOfCityA, totalCost, costs);
    return ret;
  }

  void backtrack(int k, int n, int numOfCityA, int totalCost,
vector<vector<int>>& costs) {
    if (k == n) {
      if (numOfCityA == n / 2) ret = min(ret, totalCost);
      return;
    }
    if (numOfCityA > n / 2) return;
    backtrack(k + 1, n, numOfCityA + 1, totalCost + costs[k][0], costs);
    backtrack(k + 1, n, numOfCityA, totalCost + costs[k][1], costs);
  }
};
```

Dynamic Programming:

```
class Solution {
public:
  int twoCitySchedCost(vector<vector<int>>& costs) {
    int n = costs.size() / 2;
    vector<vector<int>> dp(2 * n + 1, vector<int>(n + 1, INT_MAX));
    dp[0][0] = 0;
    // dp[i][j]: the total cost among costs[0..i - 1]: pick j to city A.
    // dp[i][j] = min(dp[i - 1][j - 1] + costs[i - 1][0], dp[i - 1][j] + costs[i
- 1][1])
    for (int i = 1; i <= 2 * n; ++i) {
      for (int j = 0; j <= i && j <= n; ++j) {
        if (i - j > n) continue;
        if (dp[i - 1][j] != INT_MAX) {
          dp[i][j] = min(dp[i][j], dp[i - 1][j] + costs[i - 1][1]);
        }
        if (j > 0 && dp[i - 1][j - 1] != INT_MAX) {
          dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + costs[i - 1][0]);
```

```
        }
      }
    }
    return dp[2 * n][n];
  }
};
```

Greedy:

```
class Solution {
public:
  int twoCitySchedCost(vector<vector<int>>& costs) {
    sort(costs.begin(), costs.end(), [](vector<int>& nums1, vector<int>& nums2)
{
      return nums1[0] - nums1[1] < nums2[0] - nums2[1];
    });
    int n = costs.size(), ret = 0;
    for (int i = 0; i < n; ++i) ret += i < n / 2 ? costs[i][0] : costs[i][1];
    return ret;
  }
};
```

1057. Campus Bikes

Description

```
On a campus represented as a 2D grid, there are N workers and M bikes, with N <=
M. Each worker and bike is a 2D coordinate on this grid.

Our goal is to assign a bike to each worker. Among the available bikes and
workers, we choose the (worker, bike) pair with the shortest Manhattan distance
between each other, and assign the bike to that worker. (If there are multiple
(worker, bike) pairs with the same shortest Manhattan distance, we choose the
pair with the smallest worker index; if there are multiple ways to do that, we
choose the pair with the smallest bike index). We repeat this process until
there are no available workers.

The Manhattan distance between two points p1 and p2 is Manhattan(p1, p2) = |p1.x
- p2.x| + |p1.y - p2.y|.

Return a vector ans of length N, where ans[i] is the index (0-indexed) of the
bike that the i-th worker is assigned to.
```

```
Example 1:



Input: workers = [[0,0],[2,1]], bikes = [[1,2],[3,3]]
Output: [1,0]
Explanation:
Worker 1 grabs Bike 0 as they are closest (without ties), and Worker 0 is
assigned Bike 1. So the output is [1, 0].
Example 2:



Input: workers = [[0,0],[1,1],[2,0]], bikes = [[1,0],[2,2],[2,1]]
Output: [0,2,1]
Explanation:
Worker 0 grabs Bike 0 at first. Worker 1 and Worker 2 share the same distance to
Bike 2, thus Worker 1 is assigned to Bike 2, and Worker 2 will take Bike 1. So
the output is [0,2,1].


Note:

0 <= workers[i][j], bikes[i][j] < 1000
All worker and bike locations are distinct.
1 <= workers.length <= bikes.length <= 1000
```

Solution

06/11/2020: Using bucket-sort:

```
class Solution {
public:
  vector<int> assignBikes(vector<vector<int>>& workers, vector<vector<int>>&
bikes) {
    vector<vector<pair<int, int>>> bucket(2001);
    int m = workers.size(), n = bikes.size();
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        bucket[abs(workers[i][0] - bikes[j][0]) + abs(workers[i][1] - bikes[j]
[1])].emplace_back(i, j);
    vector<bool> chosen_workers(m, false), chosen_bikes(n, false);
    vector<int> ret(m, -1);
    for (int i = 0; i < 2001; ++i) {
      for (int j = 0; j < (int)bucket[i].size(); ++j) {
        int w = bucket[i][j].first, b = bucket[i][j].second;
        if (!chosen_workers[w] && !chosen_bikes[b]) {
          ret[w] = b;
```

```
          chosen_workers[w] = chosen_bikes[b] = true;
        }
      }
    }
    return ret;
  }
};
```

Using `sort`:

```
class Solution {
public:
  vector<int> assignBikes(vector<vector<int>>& workers, vector<vector<int>>&
bikes) {
    int m = workers.size(), n = bikes.size();
    vector<pair<int, pair<int, int>>> dist;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        dist.push_back({abs(workers[i][0] - bikes[j][0]) + abs(workers[i][1] -
bikes[j][1]), {i, j}});
    sort(dist.begin(), dist.end());
    vector<bool> chosen_workers(m, false), chosen_bikes(n, false);
    vector<int> ret(m, -1);
    for (auto& d : dist) {
      int i = d.second.first, j = d.second.second;
      if (!chosen_workers[i] && !chosen_bikes[j]) {
        ret[i] = j;
        chosen_workers[i] = chosen_bikes[j] = true;
      }
    }
    return ret;
  }
};
```

Using `prioirty_queue` (much slower than `sort`):

```
class Solution {
public:
  vector<int> assignBikes(vector<vector<int>>& workers, vector<vector<int>>&
bikes) {
    int m = workers.size(), n = bikes.size();
    priority_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>,
greater<pair<int, pair<int, int>>>> pq;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        pq.push({abs(workers[i][0] - bikes[j][0]) + abs(workers[i][1] - bikes[j]
[1]), {i, j}});
```

```
    vector<bool> chosen_workers(m, false), chosen_bikes(n, false);
    vector<int> ret(m, -1);
    while (!pq.empty()) {
      pair<int, pair<int, int>> p = pq.top(); pq.pop();
      int i = p.second.first, j = p.second.second;
      if (!chosen_workers[i] && !chosen_bikes[j]) {
        ret[i] = j;
        chosen_workers[i] = chosen_bikes[j] = true;
      }
    }
    return ret;
  }
};
```

1065. Index Pairs of a String

Description

```
Given a text string and words (a list of strings), return all index pairs [i, j]
so that the substring text[i]...text[j] is in the list of words.



Example 1:

Input: text = "thestoryofleetcodeandme", words = ["story","fleet","leetcode"]
Output: [[3,7],[9,13],[10,17]]
Example 2:

Input: text = "ababa", words = ["aba","ab"]
Output: [[0,1],[0,2],[2,3],[2,4]]
Explanation:
Notice that matches can overlap, see "aba" is found in [0,2] and [2,4].


Note:

All strings contains only lowercase English letters.
It's guaranteed that all strings in words are different.
1 <= text.length <= 100
1 <= words.length <= 20
1 <= words[i].length <= 50
Return the pairs [i,j] in sorted order (i.e. sort them by their first coordinate
in case of ties sort them by their second coordinate).
```

Solution

05/10/2020:

```
class Solution {
public:
  vector<vector<int>> indexPairs(string text, vector<string>& words) {
    vector<vector<int>> ret;
    int n = text.size();
    for (auto& w : words) {
      int m = w.size();
      if (m > n) continue;
      for (int i = 0; i < n - m + 1; ++i) {
        if (text.substr(i, m) == w) {
          ret.push_back({i, i + m - 1});
        }
      }
    }
    sort(ret.begin(), ret.end());
    return ret;
  }
};
```

1066. Campus Bikes II

Description

```
On a campus represented as a 2D grid, there are N workers and M bikes, with N <=
M. Each worker and bike is a 2D coordinate on this grid.

We assign one unique bike to each worker so that the sum of the Manhattan
distances between each worker and their assigned bike is minimized.

The Manhattan distance between two points p1 and p2 is Manhattan(p1, p2) = |p1.x
- p2.x| + |p1.y - p2.y|.

Return the minimum possible sum of Manhattan distances between each worker and
their assigned bike.



Example 1:



Input: workers = [[0,0],[2,1]], bikes = [[1,2],[3,3]]
```

```
Output: 6
Explanation:
We assign bike 0 to worker 0, bike 1 to worker 1. The Manhattan distance of both
assignments is 3, so the output is 6.
Example 2:



Input: workers = [[0,0],[1,1],[2,0]], bikes = [[1,0],[2,2],[2,1]]
Output: 4
Explanation:
We first assign bike 0 to worker 0, then assign bike 1 to worker 1 or worker 2,
bike 2 to worker 2 or worker 1. Both assignments lead to sum of the Manhattan
distances as 4.


Note:

0 <= workers[i][0], workers[i][1], bikes[i][0], bikes[i][1] < 1000
All worker and bike locations are distinct.
1 <= workers.length <= bikes.length <= 10
```

Solution

06/11/2020:

Using Dynamic Programming:

```
class Solution {
public:
  int assignBikes(vector<vector<int>>& workers, vector<vector<int>>& bikes) {
    function dist = [](vector<int>& worker, vector<int>& bike) {
      return abs(worker[0] - bike[0]) + abs(worker[1] - bike[1]);
    };
    int m = workers.size(), n = bikes.size();
    vector<vector<int>> dp(m + 1, vector<int>(1 << n, INT_MAX - 4000));
    dp[0][0] = 0;
    int minDist = INT_MAX;
    for (int i = 1; i <= m; ++i) {
      for (int s = 1; s < (1 << n); ++s) {
        for (int j = 0; j < n; ++j) {
          if ((s & (1 << j)) != 0) {
            int prev = s ^ (1 << j);
            dp[i][s] = min(dp[i][s], dp[i - 1][prev] + dist(workers[i - 1],
bikes[j]));
            minDist = i == m ? min(minDist, dp[i][s]) : minDist;
          }
        }
```

```
      }
    }
    return minDist;
  }
};
```

Using backtracking (TLE):

```
class Solution {
public:
  int minDist = INT_MAX;
  vector<bool> chosen_bikes;
  int assignBikes(vector<vector<int>>& workers, vector<vector<int>>& bikes) {
    int m = workers.size(), n = bikes.size(), dist = 0;
    chosen_bikes.assign(1000, false);
    backtrack(0, dist, workers, bikes);
    return minDist;
  }
  void backtrack(int i, int dist, vector<vector<int>>& workers,
vector<vector<int>>& bikes) {
    if (i == workers.size()) {
      minDist = min(dist, minDist);
      return;
    }
    if (dist > minDist) return;  // pruning
    for (int j = 0; j < (int)bikes.size(); ++j) {
      if (chosen_bikes[j] == false) {
        chosen_bikes[j] = true;
        backtrack(i + 1, dist + abs(workers[i][0] - bikes[j][0]) +
abs(workers[i][1] - bikes[j][1]), workers, bikes);
        chosen_bikes[j] = false;
      }
    }
  }
};
```

1143. Longest Common Subsequence

Description

```
Given two strings text1 and text2, return the length of their longest common
subsequence.
```

A subsequence of a string is a new string generated from the original string with some characters(can be none) deleted without changing the relative order of the remaining characters. (eg, "ace" is a subsequence of "abcde" while "aec" is not). A common subsequence of two strings is a subsequence that is common to both strings.

If there is no common subsequence, return 0.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- 1 <= text1.length <= 1000
- 1 <= text2.length <= 1000
- The input strings consist of lowercase English characters only.

Solution

Discussion:

```cpp
class Solution {
public:
  int longestCommonSubsequence(string text1, string text2) {
    int m = text1.size(), n = text2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        dp[i + 1][j + 1] = text1[i] == text2[j] ? dp[i][j] + 1 : max(dp[i][j +
1], dp[i + 1][j]);
    return dp.back().back();
  }
};
```

```cpp
class Solution {
public:
  int longestCommonSubsequence(string text1, string text2) {
    int m = text1.size(), n = text2.size();
    vector<int> dp(n + 1, 0);
    for (int i = 0; i < m; ++i) {
      vector<int> tmp(dp);
      for (int j = 0; j < n; ++j) {
        dp[j + 1] = text1[i] == text2[j] ? tmp[j] + 1 : max(dp[j + 1], dp[j]);
      }
    }
    return dp.back();
  }
};
```

1170. Compare Strings by Frequency of the Smallest Character

Description

```
Let's define a function f(s) over a non-empty string s, which calculates the
frequency of the smallest character in s. For example, if s = "dcce" then f(s) =
2 because the smallest character is "c" and its frequency is 2.

Now, given string arrays queries and words, return an integer array answer,
where each answer[i] is the number of words such that f(queries[i]) < f(W),
where W is a word in words.
```

```
Example 1:

Input: queries = ["cbd"], words = ["zaaaz"]
Output: [1]
Explanation: On the first query we have f("cbd") = 1, f("zaaaz") = 3 so f("cbd")
< f("zaaaz").
Example 2:

Input: queries = ["bbb","cc"], words = ["a","aa","aaa","aaaa"]
Output: [1,2]
Explanation: On the first query only f("bbb") < f("aaaa"). On the second query
both f("aaa") and f("aaaa") are both > f("cc").


Constraints:

1 <= queries.length <= 2000
1 <= words.length <= 2000
1 <= queries[i].length, words[i].length <= 10
queries[i][j], words[i][j] are English lowercase letters.
```

Solution

02/05/2020:

```
class Solution {
public:
  vector<int> numSmallerByFrequency(vector<string>& queries, vector<string>&
words) {
    int n = queries.size(), N = 11;
    vector<int> ret(n, 0), cnt(N, 0);
    for (auto& w : words) ++cnt[f(w) - 1];
    for (int i = N - 2; i > -1; --i) cnt[i] += cnt[i + 1];
    for (int i = 0; i < n; ++i) ret[i] = cnt[f(queries[i])];
    return ret;
  }

  int f(string& s) {
    char ch = *min_element(s.begin(), s.end());
    return count(s.begin(), s.end(), ch);
  }
};
```

1277. Count Square Submatrices with All Ones

Description

```
Given a m * n matrix of ones and zeros, return how many square submatrices have
all ones.



Example 1:

Input: matrix =
[
  [0,1,1,1],
  [1,1,1,1],
  [0,1,1,1]
]
Output: 15
Explanation:
There are 10 squares of side 1.
There are 4 squares of side 2.
There is  1 square of side 3.
Total number of squares = 10 + 4 + 1 = 15.
Example 2:

Input: matrix =
[
  [1,0,1],
  [1,1,0],
  [1,1,0]
]
Output: 7
Explanation:
There are 6 squares of side 1.
There is 1 square of side 2.
Total number of squares = 6 + 1 = 7.


Constraints:

1 <= arr.length <= 300
1 <= arr[0].length <= 300
0 <= arr[i][j] <= 1
```

Solution

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
```

```
  int countSquares(vector<vector<int>>& matrix) {
    int m = matrix.size(), n = matrix[0].size(), ret = 0;;
    vector<vector<int>> dp(m, vector<int>(n, 0));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i == 0 || j == 0)
          dp[i][j] = matrix[i][j];
        else
          dp[i][j] = matrix[i][j] == 0 ? 0 : min(min(dp[i - 1][j - 1], dp[i - 1]
[j]), dp[i][j - 1]) + 1;
        ret += dp[i][j];
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  int countSquares(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    int ret = 0;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i >= 1 && j >= 1 && matrix[i][j] == 1)
          matrix[i][j] = min(min(matrix[i - 1][j], matrix[i][j - 1]), matrix[i -
1][j - 1]) + 1;
        ret += matrix[i][j];
      }
    }
    return ret;
  }
};
```

1287. Element Appearing More Than 25% In Sorted Array

Description

```
Given an integer array sorted in non-decreasing order, there is exactly one
integer in the array that occurs more than 25% of the time.

Return that integer.
```

```
Example 1:

Input: arr = [1,2,2,6,6,6,6,7,10]
Output: 6


Constraints:

1 <= arr.length <= 10^4
0 <= arr[i] <= 10^5
```

Solution

05/10/2020:

```
class Solution {
public:
  int findSpecialInteger(vector<int>& arr) {
    int n = arr.size();
    int m = (n + 1) / 4;
    for (int i = 0; i < n - m; ++i) {
      if (arr[i] == arr[i + m]) {
        return arr[i];
      }
    }
    return -1;
  }
};
```

1314. Matrix Block Sum

Description

```
Given a m * n matrix mat and an integer K, return a matrix answer where each
answer[i][j] is the sum of all elements mat[r][c] for i - K <= r <= i + K, j - K
<= c <= j + K, and (r, c) is a valid position in the matrix.


Example 1:

Input: mat = [[1,2,3],[4,5,6],[7,8,9]], K = 1
Output: [[12,21,16],[27,45,33],[24,39,28]]
Example 2:
```

```
Input: mat = [[1,2,3],[4,5,6],[7,8,9]], K = 2
Output: [[45,45,45],[45,45,45],[45,45,45]]


Constraints:

m == mat.length
n == mat[i].length
1 <= m, n, K <= 100
1 <= mat[i][j] <= 100
```

Solution

01/14/2020 (Definition):

```
class Solution {
public:
  vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int K) {
    int m = mat.size(), n = mat[0].size();
    vector<vector<int>> ret(m, vector<int>(n, 0));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        for (int k = -K; k <= K; ++k) {
          for (int l = -K; l <= K; ++l) {
            if (i + k >= 0 && i + k < m && j + l >= 0 && j + l < n) {
              ret[i][j] += mat[i + k][j + l];
            }
          }
        }
      }
    }
    return ret;
  }
};
```

01/14/2020 (Dynamic Programming):

```
class Solution {
public:
  vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int K) {
    int m = mat.size(), n = mat[0].size();
    vector<vector<int>> dp(m, vector<int>(n, 0));
    vector<vector<int>> ret(m, vector<int>(n, 0));
    dp[0][0] = mat[0][0];
    for (int i = 1; i < m; ++i) dp[i][0] += dp[i - 1][0] + mat[i][0];
    for (int j = 1; j < n; ++j) dp[0][j] += dp[0][j - 1] + mat[0][j];
    for (int i = 1; i < m; ++i)
```

```
      for (int j = 1; j < n; ++j)
        dp[i][j] = mat[i][j] + dp[i][j - 1] + dp[i - 1][j] - dp[i - 1][j - 1];
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        int r1 = max(i - K, 0);
        int c1 = max(j - K, 0);
        int r2 = min(i + K, m - 1);
        int c2 = min(j + K, n - 1);
        ret[i][j] = dp[r2][c2];
        if (r1 > 0) ret[i][j] -= dp[r1 - 1][c2];
        if (c1 > 0) ret[i][j] -= dp[r2][c1 - 1];
        if (r1 > 0 && c1 > 0) ret[i][j] += dp[r1 - 1][c1 - 1];
      }
    }
    return ret;
  }
};
```

1331. Rank Transform of an Array

Description

```
Given an array of integers arr, replace each element with its rank.

The rank represents how large the element is. The rank has the following rules:

Rank is an integer starting from 1.
The larger the element, the larger the rank. If two elements are equal, their
rank must be the same.
Rank should be as small as possible.


Example 1:

Input: arr = [40,10,20,30]
Output: [4,1,2,3]
Explanation: 40 is the largest element. 10 is the smallest. 20 is the second
smallest. 30 is the third smallest.
Example 2:

Input: arr = [100,100,100]
Output: [1,1,1]
Explanation: Same elements share the same rank.
Example 3:

Input: arr = [37,12,28,9,100,56,80,5,12]
```

```
Output: [5,3,4,2,8,6,7,1,3]


Constraints:

0 <= arr.length <= 105
-109 <= arr[i] <= 109
```

Solution

05/10/2020:

```cpp
class Solution {
public:
  vector<int> arrayRankTransform(vector<int>& arr) {
    if (arr.empty()) return {};
    set<int> keys(arr.begin(), arr.end());
    unordered_map<int, int> mp;
    int i = 1;
    for (auto& k : keys) {
      mp[k] = i++;
    }
    vector<int> ret;
    for (auto& a : arr) {
      ret.push_back(mp[a]);
    }
    return ret;
  }
};
```

1422. Maximum Score After Splitting a String

Description

```
Given a string s of zeros and ones, return the maximum score after splitting the
string into two non-empty substrings (i.e. left substring and right substring).

The score after splitting a string is the number of zeros in the left substring
plus the number of ones in the right substring.



Example 1:

Input: s = "011101"
```

```
Output: 5
Explanation:
All possible ways of splitting s into two non-empty substrings are:
left = "0" and right = "11101", score = 1 + 4 = 5
left = "01" and right = "1101", score = 1 + 3 = 4
left = "011" and right = "101", score = 1 + 2 = 3
left = "0111" and right = "01", score = 1 + 1 = 2
left = "01110" and right = "1", score = 2 + 1 = 3
Example 2:

Input: s = "00111"
Output: 5
Explanation: When left = "00" and right = "111", we get the maximum score = 2 +
3 = 5
Example 3:

Input: s = "1111"
Output: 3


Constraints:

2 <= s.length <= 500
The string s consists of characters '0' and '1' only.
```

Solution

04/25/2020:

```
class Solution {
public:
  int maxScore(string s) {
    int zeros = 0, ones = 0;
    for (auto& c : s) {
      if (c == '0') {
        ++zeros;
      } else {
        ++ones;
      }
    }
    int n = s.size();
    int cur_zeros = 0;
    int ret = 0;
    for (int i = 0; i < n - 1; ++i) {
      if (s[i] == '0') {
        ++cur_zeros;
      }
      ret = max(ret, cur_zeros + (n - (i + 1) - (zeros - cur_zeros)));
```

```
    }
    return ret;
  }
};
```

1424. Diagonal Traverse II

Description

```
Given a list of lists of integers, nums, return all elements of nums in diagonal
order as shown in the below images.


Example 1:



Input: nums = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,4,2,7,5,3,8,6,9]
Example 2:



Input: nums = [[1,2,3,4,5],[6,7],[8],[9,10,11],[12,13,14,15,16]]
Output: [1,6,2,8,7,3,9,4,12,10,5,13,11,14,15,16]
Example 3:

Input: nums = [[1,2,3],[4],[5,6,7],[8],[9,10,11]]
Output: [1,4,2,5,3,8,6,9,7,10,11]
Example 4:

Input: nums = [[1,2,3,4,5,6]]
Output: [1,2,3,4,5,6]


Constraints:

1 <= nums.length <= 10^5
1 <= nums[i].length <= 10^5
1 <= nums[i][j] <= 10^9
There at most 10^5 elements in nums.
```

Solution

04/25/2020:

```cpp
class Solution {
public:
  vector<int> findDiagonalOrder(vector<vector<int>>& nums) {
    int m = nums.size();
    if (m <= 0) return {};
    map<int, vector<int>> mp;
    for (int i = 0; i < m; ++i) {
      int n = nums[i].size();
      for (int j = 0; j < n; ++j) {
        mp[i + j].push_back(nums[i][j]);
      }
    }
    vector<int> ret;
    for (auto& m : mp) {
      ret.insert(ret.end(), m.second.rbegin(), m.second.rend());
    }
    return ret;
  }
};
```

1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Description

```
Given an array of integers nums and an integer limit, return the size of the
longest continuous subarray such that the absolute difference between any two
elements is less than or equal to limit.

In case there is no subarray satisfying the given condition return 0.



Example 1:

Input: nums = [8,2,4,7], limit = 4
Output: 2
Explanation: All subarrays are:
[8] with maximum absolute diff |8-8| = 0 <= 4.
[8,2] with maximum absolute diff |8-2| = 6 > 4.
[8,2,4] with maximum absolute diff |8-2| = 6 > 4.
[8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.
[2] with maximum absolute diff |2-2| = 0 <= 4.
[2,4] with maximum absolute diff |2-4| = 2 <= 4.
```

```
[2,4,7] with maximum absolute diff |2-7| = 5 > 4.
[4] with maximum absolute diff |4-4| = 0 <= 4.
[4,7] with maximum absolute diff |4-7| = 3 <= 4.
[7] with maximum absolute diff |7-7| = 0 <= 4.
Therefore, the size of the longest subarray is 2.
Example 2:

Input: nums = [10,1,2,4,7,2], limit = 5
Output: 4
Explanation: The subarray [2,4,7,2] is the longest since the maximum absolute
diff is |2-7| = 5 <= 5.
Example 3:

Input: nums = [4,2,2,2,4,4,2,2], limit = 0
Output: 3


Constraints:

1 <= nums.length <= 10^5
1 <= nums[i] <= 10^9
0 <= limit <= 10^9
```

Solution

05/03/2020:

```
class Solution {
public:
  int longestSubarray(vector<int>& nums, int limit, int start = 0, int stop =
INT_MAX) {
    if (stop == INT_MAX) stop = nums.size() - 1;
    if (start > stop) return 0;
    auto min_it = min_element(nums.begin() + start, nums.begin() + stop + 1);
    auto max_it = max_element(nums.begin() + start, nums.begin() + stop + 1);
    if (*max_it - *min_it <= limit) return stop - start + 1;
    auto lo = min_it, hi = min_it;
    for (; lo - nums.begin() > start && *(lo - 1) - *min_it <= limit; --lo);
    for (; hi - nums.begin() < stop && *(hi + 1) - *min_it <= limit; ++hi);
    int left = int(min_it - nums.begin() - 1) - start  + 1 >= hi - lo + 1 ?
longestSubarray(nums, limit, start, int(min_it - nums.begin() - 1)) : 0;
    int right = stop - int(min_it - nums.begin() - 1) + 1 >= hi - lo + 1 ?
longestSubarray(nums, limit, int(min_it - nums.begin() + 1), stop) : 0;
    return max(int(hi - lo + 1), max(left, right));
  }
};
```

1441. Build an Array With Stack Operations

Description

Given an array target and an integer n. In each iteration, you will read a number from list = {1,2,3..., n}.

Build the target array using the following operations:

Push: Read a new element from the beginning list, and push it in the array.
Pop: delete the last element of the array.
If the target array is already built, stop reading more elements.
You are guaranteed that the target array is strictly increasing, only containing numbers between 1 to n inclusive.

Return the operations to build the target array.

You are guaranteed that the answer is unique.

Example 1:

Input: target = [1,3], n = 3
Output: ["Push","Push","Pop","Push"]
Explanation:
Read number 1 and automatically push in the array -> [1]
Read number 2 and automatically push in the array then Pop it -> [1]
Read number 3 and automatically push in the array -> [1,3]
Example 2:

Input: target = [1,2,3], n = 3
Output: ["Push","Push","Push"]
Example 3:

Input: target = [1,2], n = 4
Output: ["Push","Push"]
Explanation: You only need to read the first 2 numbers and stop.
Example 4:

Input: target = [2,3,4], n = 4
Output: ["Push","Pop","Push","Push","Push"]

Constraints:

1 <= target.length <= 100
1 <= target[i] <= 100

```
1 <= n <= 100
target is strictly increasing.
```

Solution

05/10/2020:

```cpp
class Solution {
public:
  vector<string> buildArray(vector<int>& target, int n) {
    int m = *max_element(target.begin(), target.end());
    unordered_set<int> s(target.begin(), target.end());
    vector<string> ret;
    for (int i = 1; i <= m; ++i) {
      ret.push_back("Push");
      if (s.count(i) == 0) {
        ret.push_back("Pop");
      }
    }
    return ret;
  }
};
```

1447. Simplified Fractions

Description

```
Given an integer n, return a list of all simplified fractions between 0 and 1
(exclusive) such that the denominator is less-than-or-equal-to n. The fractions
can be in any order.



Example 1:

Input: n = 2
Output: ["1/2"]
Explanation: "1/2" is the only unique fraction with a denominator less-than-or-
equal-to 2.
Example 2:

Input: n = 3
Output: ["1/2","1/3","2/3"]
Example 3:
```

```
Input: n = 4
Output: ["1/2","1/3","1/4","2/3","3/4"]
Explanation: "2/4" is not a simplified fraction because it can be simplified to
"1/2".
Example 4:

Input: n = 1
Output: []


Constraints:

1 <= n <= 100
```

Solution

05/16/2020:

```
class Solution {
public:
  vector<string> simplifiedFractions(int n) {
    unordered_set<string> ret;
    for (int i = 1; i < n; ++i) {
      for (int j = i + 1; j <= n; ++j) {
        int gcd = __gcd(i, j);
        int numerator = i / gcd;
        int denominator = j / gcd;
        ret.insert(to_string(numerator) + "/" + to_string(denominator));
      }
    }
    return vector<string>(ret.begin(), ret.end());
  }
};
```

1452. People Whose List of Favorite Companies Is Not a Subset of Another List

Description

```
Given the array favoriteCompanies where favoriteCompanies[i] is the list of
favorites companies for the ith person (indexed from 0).
```

```
Return the indices of people whose list of favorite companies is not a subset of any other list of favorites companies. You must return the indices in increasing order.



Example 1:

Input: favoriteCcompanies = [["leetcode","google","facebook"],["google","microsoft"],["google","facebook"],["google"],["amazon"]]
Output: [0,1,4]
Explanation:
Person with index=2 has favoriteCompanies[2]=["google","facebook"] which is a subset of favoriteCompanies[0]=["leetcode","google","facebook"] corresponding to the person with index 0.
Person with index=3 has favoriteCompanies[3]=["google"] which is a subset of favoriteCompanies[0]=["leetcode","google","facebook"] and favoriteCompanies[1]=["google","microsoft"].
Other lists of favorite companies are not a subset of another list, therefore, the answer is [0,1,4].
Example 2:

Input: favoriteCompanies = [["leetcode","google","facebook"],["leetcode","amazon"],["facebook","google"]]
Output: [0,1]
Explanation: In this case favoriteCompanies[2]=["facebook","google"] is a subset of favoriteCompanies[0]=["leetcode","google","facebook"], therefore, the answer is [0,1].
Example 3:

Input: favoriteCompanies = [["leetcode"],["google"],["facebook"],["amazon"]]
Output: [0,1,2,3]


Constraints:

1 <= favoriteCompanies.length <= 100
1 <= favoriteCompanies[i].length <= 500
1 <= favoriteCompanies[i][j].length <= 20
All strings in favoriteCompanies[i] are distinct.
All lists of favorite companies are distinct, that is, If we sort alphabetically each list then favoriteCompanies[i] != favoriteCompanies[j].
All strings consist of lowercase English letters only.
```

Solution

05/16/2020:

```
class Solution {
public:
  vector<int> peopleIndexes(vector<vector<string>>& favoriteCcompanies) {
    vector<pair<unordered_set<string>, int>> companies;
    int n = favoriteCompanies.size();
    for (int i = 0; i < n; ++i) {
      companies.emplace_back(unordered_set<string>(favoriteCompanies[i].begin(),
favoriteCompanies[i].end()), i);
    }
    sort(companies.begin(), companies.end(), [](pair<unordered_set<string>,
int>& p1, pair<unordered_set<string>, int>& p2) {
      return p1.first.size() > p2.first.size();
    });
    vector<int> ret;
    ret.push_back(companies[0].second);
    for (int i = 1; i < n; ++i) {
      int cnt = 0;
      for (int j = 0; j < i; ++j) {
        bool isSubset = true;
        for (auto& s : companies[i].first) {
          if (companies[j].first.count(s) == 0) {
            isSubset = false;
            break;
          }
        }
        if (!isSubset) {
          ++cnt;
        }
      }
      if (cnt == i) ret.push_back(companies[i].second);
    }
    sort(ret.begin(), ret.end());
    return ret;
  }
};
```

1461. Check If a String Contains All Binary Codes of Size K

Description

```
Given a binary string s and an integer k.

Return True if all binary codes of length k is a substring of s. Otherwise,
return False.
```

```
Example 1:

Input: s = "00110110", k = 2
Output: true
Explanation: The binary codes of length 2 are "00", "01", "10" and "11". They
can be all found as substrings at indicies 0, 1, 3 and 2 respectively.
Example 2:

Input: s = "00110", k = 2
Output: true
Example 3:

Input: s = "0110", k = 1
Output: true
Explanation: The binary codes of length 1 are "0" and "1", it is clear that both
exist as a substring.
Example 4:

Input: s = "0110", k = 2
Output: false
Explanation: The binary code "00" is of length 2 and doesn't exist in the array.
Example 5:

Input: s = "0000000001011100", k = 4
Output: false


Constraints:

1 <= s.length <= 5 * 10^5
s consists of 0's and 1's only.
1 <= k <= 20
```

Solution

05/30/2020:

```
class Solution {
public:
  bool hasAllCodes(string s, int k) {
    bool ret = true;
    int n = 1 << k;
    int sz = s.size() - k;
    unordered_set<string> required;
    for (int i = 0; i <= sz; ++i) {
      string sub = s.substr(i, k);
```

```
      required.insert(sub);
    }
    return required.size() == n;
  }
};
```

1463. Cherry Pickup II

Description

Given a rows x cols matrix grid representing a field of cherries. Each cell in grid represents the number of cherries that you can collect.

You have two robots that can collect cherries for you, Robot #1 is located at the top-left corner (0,0) , and Robot #2 is located at the top-right corner (0, cols-1) of the grid.

Return the maximum number of cherries collection using both robots by following the rules below:

From a cell (i,j), robots can move to cell (i+1, j-1) , (i+1, j) or (i+1, j+1).
When any robot is passing through a cell, It picks it up all cherries, and the cell becomes an empty cell (0).
When both robots stay on the same cell, only one of them takes the cherries.
Both robots cannot move outside of the grid at any moment.
Both robots should reach the bottom row in the grid.

Example 1:

Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]
Output: 24
Explanation: Path of robot #1 and #2 are described in color green and blue respectively.
Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12.
Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12.
Total of cherries: 12 + 12 = 24.
Example 2:

Input: grid = [[1,0,0,0,0,0,1],[2,0,0,0,0,3,0],[2,0,9,0,0,0,0],[0,3,0,5,4,0,0],[1,0,2,3,0,0,6]]
Output: 28

```
Explanation: Path of robot #1 and #2 are described in color green and blue
respectively.
Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17.
Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11.
Total of cherries: 17 + 11 = 28.
Example 3:

Input: grid = [[1,0,0,3],[0,0,0,3],[0,0,3,3],[9,0,3,3]]
Output: 22
Example 4:

Input: grid = [[1,1],[1,1]]
Output: 4


Constraints:

rows == grid.length
cols == grid[i].length
2 <= rows, cols <= 70
0 <= grid[i][j] <= 100
```

Solution

06/07/2020:

```
int dp[71][71][71];

class Solution {
public:
  int cherryPickup(vector<vector<int>>& grid) {
    fill_n(dp[0][0], 71 * 71 * 71, -1);
    int m = grid.size(), n = grid[0].size();
    dp[0][0][n - 1] = grid[0][0] + grid[0][n - 1];
    for (int i = 1; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        for (int k = 0; k < n; ++k) {
          int cur = -1;
          for (int u = -1; u <= 1; ++u) {
            int nj = j + u;
            for (int v = -1; v <= 1; ++v) {
              int nk = k + v;
              if (nj >= 0 && nj < n && nk >= 0 && nk < n && dp[i - 1][nj][nk] !=
-1)
                cur = max(cur, dp[i - 1][nj][nk] + grid[i][j] + grid[i][k]);
            }
          }
          if (j == k) cur -= grid[i][j];
```

```
          dp[i][j][k] = max(dp[i][j][k], cur);
        }
      }
    }
    int ret = -1;
    for (int j = 0; j < n; ++j) ret = max(ret, *max_element(dp[m - 1][j], dp[m -
1][j] + n));
    return ret;
  }
};
```

1471. The k Strongest Values in an Array

Description

```
Given an array of integers arr and an integer k.

A value arr[i] is said to be stronger than a value arr[j] if |arr[i] - m| >
|arr[j] - m| where m is the median of the array.
If |arr[i] - m| == |arr[j] - m|, then arr[i] is said to be stronger than arr[j]
if arr[i] > arr[j].

Return a list of the strongest k values in the array. return the answer in any
arbitrary order.

Median is the middle value in an ordered integer list. More formally, if the
length of the list is n, the median is the element in position ((n - 1) / 2) in
the sorted list (0-indexed).

For arr = [6, -3, 7, 2, 11], n = 5 and the median is obtained by sorting the
array arr = [-3, 2, 6, 7, 11] and the median is arr[m] where m = ((5 - 1) / 2) =
2. The median is 6.
For arr = [-7, 22, 17, 3], n = 4 and the median is obtained by sorting the array
arr = [-7, 3, 17, 22] and the median is arr[m] where m = ((4 - 1) / 2) = 1. The
median is 3.


Example 1:

Input: arr = [1,2,3,4,5], k = 2
Output: [5,1]
Explanation: Median is 3, the elements of the array sorted by the strongest are
[5,1,4,2,3]. The strongest 2 elements are [5, 1]. [1, 5] is also accepted
answer.
Please note that although |5 - 3| == |1 - 3| but 5 is stronger than 1 because 5
> 1.
```

```
Example 2:

Input: arr = [1,1,3,5,5], k = 2
Output: [5,5]
Explanation: Median is 3, the elements of the array sorted by the strongest are
[5,5,1,1,3]. The strongest 2 elements are [5, 5].
Example 3:

Input: arr = [6,7,11,7,6,8], k = 5
Output: [11,8,6,6,7]
Explanation: Median is 7, the elements of the array sorted by the strongest are
[11,8,6,6,7,7].
Any permutation of [11,8,6,6,7] is accepted.
Example 4:

Input: arr = [6,-3,7,2,11], k = 3
Output: [-3,11,2]
Example 5:

Input: arr = [-7,22,17,3], k = 2
Output: [22,17]


Constraints:

1 <= arr.length <= 10^5
-10^5 <= arr[i] <= 10^5
1 <= k <= arr.length
```

Solution

06/06/2020:

```
int m;
class Solution {
public:
  vector<int> getStrongest(vector<int>& arr, int k) {
    int n = arr.size();
    sort(arr.begin(), arr.end());
    m = arr[(n - 1) / 2];
    sort(arr.begin(), arr.end(), [](int a, int b) {
      if (abs(a - m) == abs(b - m)) return a > b;
      return abs(a - m) > abs(b - m);
    });
    vector<int> ret;
    for (int i = 0; i < k; ++i)
      ret.push_back(arr[i]);
    return ret;
```

```
  }
};
```

Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

Description

```
Given a binary tree where each path going from the root to any leaf form a valid
sequence, check if a given string is a valid sequence in such binary tree.

We get the given string from the concatenation of an array of integers arr and
the concatenation of all values of the nodes along a path results in a sequence
in the given binary tree.



Example 1:



Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1]
Output: true
Explanation:
The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure).
Other valid sequences are:
0 -> 1 -> 1 -> 0
0 -> 0 -> 0
Example 2:



Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,0,1]
Output: false
Explanation: The path 0 -> 0 -> 1 does not exist, therefore it is not even a
sequence.
Example 3:



Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,1]
Output: false
Explanation: The path 0 -> 1 -> 1 is a sequence, but it is not a valid sequence.
```

```
Constraints:

1 <= arr.length <= 5000
0 <= arr[i] <= 9
Each node's value is between [0 - 9].
```

Solution

Discussion

- Using `unordered_map` is straightforward: convert numbers visited to a string and store seen strings in an `unordered_map`. Then just check whether the target string generated by the `arr` is in the `unordered_map` or not.
- Time complexity: O(n)
- Space complexity: O(n^2)

```
class Solution {
private:
  unordered_set<string> s;
  string target;
public:
  bool isValidSequence(TreeNode* root, vector<int>& arr) {
    for (auto& n : arr) target += to_string(n);
    dfs(root);
    return s.count(target);
  }

  void dfs(TreeNode* root, string str="") {
    if (root == nullptr) return;
    if (root->left == nullptr && root->right == nullptr) {
      str += to_string(root->val);
      if (!str.empty()) s.insert(str);
      return;
    }
    dfs(root->left, str + to_string(root->val));
    dfs(root->right, str + to_string(root->val));
  }
};
```

- Another way is more straightforward, we check the elements in the `arr` while traversing the nodes in the tree:
- Time complexity: O(n)
- Space complexity: O(n)

```cpp
class Solution {
public:
  bool isValidSequence(TreeNode* root, vector<int>& arr, int i = 0) {
    if (root == nullptr) return false;
    if (i >= (int)arr.size()) return false;
    if (root->left == nullptr && root->right == nullptr && i == (int)arr.size()
- 1) return arr.back() == root->val;
    return root->val == arr[i] && (isValidSequence(root->left, arr, i + 1) ||
isValidSequence(root->right, arr, i + 1));
  }
};
```

1477. Find Two Non-overlapping Sub-arrays Each With Target Sum

Description

```
Given an array of integers arr and an integer target.

You have to find two non-overlapping sub-arrays of arr each with sum equal
target. There can be multiple answers so you have to find an answer where the
sum of the lengths of the two sub-arrays is minimum.

Return the minimum sum of the lengths of the two required sub-arrays, or return
-1 if you cannot find such two sub-arrays.



Example 1:

Input: arr = [3,2,2,4,3], target = 3
Output: 2
Explanation: Only two sub-arrays have sum = 3 ([3] and [3]). The sum of their
lengths is 2.
Example 2:

Input: arr = [7,3,4,7], target = 7
Output: 2
Explanation: Although we have three non-overlapping sub-arrays of sum = 7 ([7],
[3,4] and [7]), but we will choose the first and third sub-arrays as the sum of
their lengths is 2.
Example 3:

Input: arr = [4,3,2,6,2,3,4], target = 6
```

```
Output: -1
Explanation: We have only one sub-array of sum = 6.
Example 4:

Input: arr = [5,5,4,4,5], target = 3
Output: -1
Explanation: We cannot find a sub-array of sum = 3.
Example 5:

Input: arr = [3,1,1,1,5,1,2,1], target = 3
Output: 3
Explanation: Note that sub-arrays [1,2] and [2,1] cannot be an answer because
they overlap.


Constraints:

1 <= arr.length <= 10^5
1 <= arr[i] <= 1000
1 <= target <= 10^8
```

Solution

06/13/2020:

```
class Solution {
public:
  int minSumOfLengths(vector<int>& arr, int target) {
    queue<pair<int, int>> q;
    int ret = INT_MAX, preMin = INT_MAX;
    int lo = 0, hi = 0, cur = 0;
    while (hi < arr.size()) {
      cur += arr[hi];
      ++hi;
      while (cur > target && lo < hi) {
        cur -= arr[lo];
        ++lo;
      }
      while (!q.empty() && q.front().second <= lo) {
        preMin = min(preMin, q.front().second - q.front().first);
        q.pop();
      }
      if (cur == target) {
        if (preMin != INT_MAX) {
          ret = min(ret, preMin + hi - lo);
        }
        q.emplace(lo, hi);
      }
```

```
    }
    return ret == INT_MAX ? -1 : ret;
  }
};
```

```
void solve(vector<int>& v, int t, vector<int>& ret) {
  int n = v.size();
  ret.resize(n + 1);
  fill(ret.begin(), ret.end(), 1e9);
  int cur = 0;
  int lhs = 0;
  for (int i = 1; i <= n; ++i) {
    cur += v[i - 1];
    while (cur > t) cur -= v[lhs++];
    if (cur == t) ret[i] = i - lhs;
  }
}

class Solution {
public:
  int minSumOfLengths(vector<int>& arr, int target) {
    vector<int> lhs, rhs;
    solve(arr, target, lhs);
    reverse(arr.begin(), arr.end());
    solve(arr, target, rhs);
    for (int i = 1; i < (int)lhs.size(); ++i) lhs[i] = min(lhs[i], lhs[i - 1]);
    for (int i = 1; i < (int)rhs.size(); ++i) rhs[i] = min(rhs[i], rhs[i - 1]);
    int ret = 1e9;
    int n = arr.size();
    for (int i = 1; i < n; ++i) ret = min(ret, lhs[i] + rhs[n - i]);
    if (ret >= 1e9) ret = -1;
    return ret;
  }
};
```

1480. Running Sum of 1d Array

Description

```
Given an array nums. We define a running sum of an array as runningSum[i] =
sum(nums[0]…nums[i]).

Return the running sum of nums.
```

```
Example 1:

Input: nums = [1,2,3,4]
Output: [1,3,6,10]
Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].
Example 2:

Input: nums = [1,1,1,1,1]
Output: [1,2,3,4,5]
Explanation: Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1,
1+1+1+1+1].
Example 3:

Input: nums = [3,1,2,10,1]
Output: [3,4,6,16,17]


Constraints:

1 <= nums.length <= 1000
-10^6 <= nums[i] <= 10^6
```

Solution

06/13/2020:

```
class Solution {
public:
  vector<int> runningSum(vector<int>& nums) {
    partial_sum(nums.begin(), nums.end(), nums.begin());
    return nums;
  }
};
```

1482. Minimum Number of Days to Make m Bouquets

Description

```
Given an integer array bloomDay, an integer m and an integer k.

We need to make m bouquets. To make a bouquet, you need to use k adjacent
flowers from the garden.
```

The garden consists of n flowers, the ith flower will bloom in the bloomDay[i] and then can be used in exactly one bouquet.

Return the minimum number of days you need to wait to be able to make m bouquets from the garden. If it is impossible to make m bouquets return -1.

Example 1:

Input: bloomDay = [1,10,3,10,2], m = 3, k = 1
Output: 3
Explanation: Let's see what happened in the first three days. x means flower bloomed and _ means flower didn't bloom in the garden.
We need 3 bouquets each should contain 1 flower.
After day 1: [x, _, _, _, _]   // we can only make one bouquet.
After day 2: [x, _, _, _, x]   // we can only make two bouquets.
After day 3: [x, _, x, _, x]   // we can make 3 bouquets. The answer is 3.
Example 2:

Input: bloomDay = [1,10,3,10,2], m = 3, k = 2
Output: -1
Explanation: We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We only have 5 flowers so it is impossible to get the needed bouquets and we return -1.
Example 3:

Input: bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3
Output: 12
Explanation: We need 2 bouquets each should have 3 flowers.
Here's the garden after the 7 and 12 days:
After day 7: [x, x, x, x, _, x, x]
We can make one bouquet of the first three flowers that bloomed. We cannot make another bouquet from the last three flowers that bloomed because they are not adjacent.
After day 12: [x, x, x, x, x, x, x]
It is obvious that we can make two bouquets in different ways.
Example 4:

Input: bloomDay = [1000000000,1000000000], m = 1, k = 1
Output: 1000000000
Explanation: You need to wait 1000000000 days to have a flower ready for a bouquet.
Example 5:

Input: bloomDay = [1,10,2,9,3,8,4,7,5,6], m = 4, k = 2
Output: 9

```
Constraints:

bloomDay.length == n
1 <= n <= 10^5
1 <= bloomDay[i] <= 10^9
1 <= m <= 10^6
1 <= k <= n
```

Solution

06/13/2020:

```
class Solution {
public:
  int minDays(vector<int>& bloomDay, int m, int k) {
    if (bloomDay.size() < (long long)m * k) return -1;
    int lo = 0, hi = 1e9;
    while (lo < hi) {
      int mid = lo + (hi - lo) / 2;
      int amount = 0, candidate = 0;
      for (auto& b : bloomDay) {
        if (b <= mid)
          ++candidate;
        else
          candidate = 0;
        if (candidate == k) {
          candidate = 0;
          ++amount;
        }
      }
      if (amount >= m)
        hi = mid;
      else
        lo = mid + 1;
    }
    return lo;
  }
};
```

1483. Kth Ancestor of a Tree Node

Description

```
You are given a tree with n nodes numbered from 0 to n-1 in the form of a parent
array where parent[i] is the parent of node i. The root of the tree is node 0.
```

```
Implement the function getKthAncestor(int node, int k) to return the k-th
ancestor of the given node. If there is no such ancestor, return -1.

The k-th ancestor of a tree node is the k-th node in the path from that node to
the root.



Example:



Input:
["TreeAncestor","getKthAncestor","getKthAncestor","getKthAncestor"]
[[7,[-1,0,0,1,1,2,2]],[3,1],[5,2],[6,3]]

Output:
[null,1,0,-1]

Explanation:
TreeAncestor treeAncestor = new TreeAncestor(7, [-1, 0, 0, 1, 1, 2, 2]);

treeAncestor.getKthAncestor(3, 1);  // returns 1 which is the parent of 3
treeAncestor.getKthAncestor(5, 2);  // returns 0 which is the grandparent of 5
treeAncestor.getKthAncestor(6, 3);  // returns -1 because there is no such
ancestor


Constraints:

1 <= k <= n <= 5*10^4
parent[0] == -1 indicating that 0 is the root node.
0 <= parent[i] < n for all 0 < i < n
0 <= node < n
There will be at most 5*10^4 queries.
```

Solution

06/13/2020:

```
const int MAXD = 16;
int dp[50005][MAXD];

class TreeAncestor {
public:
  TreeAncestor(int n, vector<int>& parent) {
    for (int i = 0; i < n; ++i) dp[i][0] = parent[i];
```

```
    for (int d = 1; d < MAXD; ++d) {
      for (int i = 0; i < n; ++i) {
        if (dp[i][d - 1] == -1)
          dp[i][d] = -1;
        else
          dp[i][d] = dp[dp[i][d - 1]][d - 1];
      }
    }
  }

  int getKthAncestor(int node, int k) {
    for (int d = MAXD - 1; d >= 0; --d) {
      if ((1 << d) <= k) {
        k -= 1 << d;
        node = dp[node][d];
        if (node == -1) return node;
      }
    }
    return node;
  }
};

/**
 * Your TreeAncestor object will be instantiated and called as such:
 * TreeAncestor* obj = new TreeAncestor(n, parent);
 * int param_1 = obj->getKthAncestor(node,k);
 */
```

1488. Avoid Flood in The City

Description

```
Your country has an infinite number of lakes. Initially, all the lakes are
empty, but when it rains over the nth lake, the nth lake becomes full of water.
If it rains over a lake which is full of water, there will be a flood. Your goal
is to avoid the flood in any lake.

Given an integer array rains where:

rains[i] > 0 means there will be rains over the rains[i] lake.
rains[i] == 0 means there are no rains this day and you can choose one lake this
day and dry it.
Return an array ans where:

ans.length == rains.length
ans[i] == -1 if rains[i] > 0.
```

ans[i] is the lake you choose to dry in the ith day if rains[i] == 0.
If there are multiple valid answers return any of them. If it is impossible to avoid flood return an empty array.

Notice that if you chose to dry a full lake, it becomes empty, but if you chose to dry an empty lake, nothing changes. (see example 4)

Example 1:

Input: rains = [1,2,3,4]
Output: [-1,-1,-1,-1]
Explanation: After the first day full lakes are [1]
After the second day full lakes are [1,2]
After the third day full lakes are [1,2,3]
After the fourth day full lakes are [1,2,3,4]
There's no day to dry any lake and there is no flood in any lake.
Example 2:

Input: rains = [1,2,0,0,2,1]
Output: [-1,-1,2,1,-1,-1]
Explanation: After the first day full lakes are [1]
After the second day full lakes are [1,2]
After the third day, we dry lake 2. Full lakes are [1]
After the fourth day, we dry lake 1. There is no full lakes.
After the fifth day, full lakes are [2].
After the sixth day, full lakes are [1,2].
It is easy that this scenario is flood-free. [-1,-1,1,2,-1,-1] is another acceptable scenario.
Example 3:

Input: rains = [1,2,0,1,2]
Output: []
Explanation: After the second day, full lakes are [1,2]. We have to dry one lake in the third day.
After that, it will rain over lakes [1,2]. It's easy to prove that no matter which lake you choose to dry in the 3rd day, the other one will flood.
Example 4:

Input: rains = [69,0,0,0,69]
Output: [-1,69,1,1,-1]
Explanation: Any solution on one of the forms [-1,69,x,y,-1], [-1,x,69,y,-1] or [-1,x,y,69,-1] is acceptable where 1 <= x,y <= 10^9
Example 5:

Input: rains = [10,20,20]
Output: []

```
Explanation: It will rain over lake 20 two consecutive days. There is no chance
to dry any lake.


Constraints:

1 <= rains.length <= 10^5
0 <= rains[i] <= 10^9
```

Solution

06/20/2020:

```
class Solution {
public:
  vector<int> avoidFlood(vector<int>& rains) {
    int n = rains.size();
    unordered_map<int, int> bad;
    set<int> canfix;
    vector<int> ret(n, -1);
    for (int i = 0; i < n; ++i) {
      if (rains[i] == 0) {
        canfix.insert(i);
        ret[i] = 1;
      } else {
        if (bad.count(rains[i]) == 0) {
          bad[rains[i]] = i;
        } else {
          int must = bad[rains[i]];
          auto it = canfix.upper_bound(must);
          if (it == canfix.end()) return {};
          ret[*it] = rains[i];
          canfix.erase(it);
          bad[rains[i]] = i;
        }
      }
    }
    return ret;
  }
};
```