

## Download PDF

- Backtracking
  - 47. Permutations II
  - 52. N-Queens II
  - 90. Subsets II
  - 1101. The Earliest Moment When Everyone Become Friends
  - 1319. Number of Operations to Make Network Connected
- Binary Search
  - 69. Sqrt(x)
  - 275. H-Index II
  - 367. Valid Perfect Square
  - 441. Arranging Coins
  - 744. Find Smallest Letter Greater Than Target
  - 892. Surface Area of 3D Shapes
  - 976. Largest Perimeter Triangle
  - 1198. Find Smallest Common Element in All Rows
- Array
  - 1. Two Sum
  - 73. Set Matrix Zeroes
  - 167. Two Sum II - Input array is sorted
  - 170. Two Sum III - Data structure design
  - 463. Island Perimeter
  - 509. Fibonacci Number
  - 561. Array Partition I
  - 566. Reshape the Matrix
  - 594. Longest Harmonious Subsequence
  - 624. Maximum Distance in Arrays
  - 653. Two Sum IV - Input is a BST
  - 654. Maximum Binary Tree
  - 623. Add One Row to Tree
  - 695. Max Area of Island
  - 760. Find Anagram Mappings
  - 766. Toeplitz Matrix
  - 811. Subdomain Visit Count
  - 832. Flipping an Image
  - 852. Peak Index in a Mountain Array
  - 867. Transpose Matrix
  - 905. Sort Array By Parity
  - 918. Maximum Sum Circular Subarray
  - 922. Sort Array By Parity II
  - 961. N-Repeated Element in Size 2N Array
  - 977. Squares of a Sorted Array

- 994. Rotting Oranges
- 1051. Height Checker
- 1064. Fixed Point
- 1085. Sum of Digits in the Minimum Number
- 1086. High Five
- 1099. Two Sum Less Than K
- 1122. Relative Sort Array
- 1133. Largest Unique Number
- 1196. How Many Apples Can You Put into the Basket
- 1200. Minimum Absolute Difference
- 1207. Unique Number of Occurrences
- 1213. Intersection of Three Sorted Arrays
- 1217. Play with Chips
- 1243. Array Transformation
- 1252. Cells with Odd Values in a Matrix
- 1266. Minimum Time Visiting All Points
- 1299. Replace Elements with Greatest Element on Right Side
- 1304. Find N Unique Integers Sum up to Zero
- 1305. All Elements in Two Binary Search Trees
- 1313. Decompress Run-Length Encoded List
- 1337. The K Weakest Rows in a Matrix
- 1338. Reduce Array Size to The Half
- 1413. Minimum Value to Get Positive Step by Step Sum
- 1423. Maximum Points You Can Obtain from Cards
- 1425. Constrained Subset Sum
- 1428. Leftmost Column with at Least a One
- 1431. Kids With the Greatest Number of Candies
- 1437. Check If All 1's Are at Least Length K Places Away
- 1442. Count Triplets That Can Form Two Arrays of Equal XOR
- 1450. Number of Students Doing Homework at a Given Time
- 1460. Make Two Arrays Equal by Reversing Sub-arrays
- 1464. Maximum Product of Two Elements in an Array
- 1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts
- 1470. Shuffle the Array
- 1475. Final Prices With a Special Discount in a Shop
- 1481. Least Number of Unique Integers after K Removals
- 1491. Average Salary Excluding the Minimum and Maximum Salary
- Tree
  - 104. Maximum Depth of Binary Tree
  - 111. Minimum Depth of Binary Tree
  - 144. Binary Tree Preorder Traversal
  - 199. Binary Tree Right Side View
  - 513. Find Bottom Left Tree Value
  - 515. Find Largest Value in Each Tree Row
  - 559. Maximum Depth of N-ary Tree
  - 563. Binary Tree Tilt

- 589. N-ary Tree Preorder Traversal
- 590. N-ary Tree Postorder Traversal
- 617. Merge Two Binary Trees
- 655. Print Binary Tree
- 700. Search in a Binary Search Tree
- 872. Leaf-Similar Trees
- 897. Increasing Order Search Tree
- 938. Range Sum of BST
- 965. Univalued Binary Tree
- 993. Cousins in Binary Tree
- 1008. Construct Binary Search Tree from Preorder Traversal
- 1022. Sum of Root To Leaf Binary Numbers
- 1325. Delete Leaves With a Given Value
- 1333. Filter Restaurants by Vegan-Friendly, Price and Distance
- 1443. Minimum Time to Collect All Apples in a Tree
- 1448. Count Good Nodes in Binary Tree
- 1457. Pseudo-Palindromic Paths in a Binary Tree
- 1469. Find All The Lonely Nodes
- 1490. Clone N-ary Tree
- Math
  - 2. Add Two Numbers
  - 7. Reverse Integer
  - 9. Palindrome Number
  - 136. Single Number
  - 172. Factorial Trailing Zeroes
  - 202. Happy Number
  - 217. Contains Duplicate
  - 258. Add Digits
  - 263. Ugly Number
  - 476. Number Complement
  - 645. Set Mismatch
  - 728. Self Dividing Numbers
  - 869. Reordered Power of 2
  - 883. Projection Area of 3D Shapes
  - 908. Smallest Range I
  - 970. Powerful Integers
  - 1103. Distribute Candies to People
  - 1134. Armstrong Number
  - 1185. Day of the Week
  - 1227. Airplane Seat Assignment Probability
  - 1228. Missing Number In Arithmetic Progression
  - 1232. Check If It Is a Straight Line
  - 1237. Find Positive Integer Solution for a Given Equation
  - 1281. Subtract the Product and Sum of Digits of an Integer
  - 1295. Find Numbers with Even Number of Digits
  - 1323. Maximum 69 Number

- String
  - 20. Valid Parentheses
  - 58. Length of Last Word
  - 71. Simplify Path
  - 344. Reverse String
  - 383. Ransom Note
  - 500. Keyboard Row
  - 557. Reverse Words in a String III
  - 599. Minimum Index Sum of Two Lists
  - 657. Robot Return to Origin
  - 709. To Lower Case
  - 771. Jewels and Stones
  - 796. Rotate String
  - 804. Unique Morse Code Words
  - 806. Number of Lines To Write String
  - 821. Shortest Distance to a Character
  - 893. Groups of Special-Equivalent Strings
  - 929. Unique Email Addresses
  - 953. Verifying an Alien Dictionary
  - 1002. Find Common Characters
  - 1021. Remove Outermost Parentheses
  - 1044. Longest Duplicate Substring
  - 1047. Remove All Adjacent Duplicates In String
  - 1078. Occurrences After Bigram
  - 1108. Defanging an IP Address
  - 1119. Remove Vowels from a String
  - 1160. Find Words That Can Be Formed by Characters
  - 1165. Single-Row Keyboard
  - 1180. Count Substrings with Only One Distinct Letter
  - 1221. Split a String in Balanced Strings
  - 1236. Web Crawler
  - 1309. Decrypt String from Alphabet to Integer Mapping
  - 1324. Print Words Vertically
  - 1332. Remove Palindromic Subsequences
  - 1427. Perform String Shifts
  - 1432. Max Difference You Can Get From Changing an Integer
  - 1433. Check If a String Can Break Another String
  - 1436. Destination City
  - 1446. Consecutive Characters
  - 1451. Rearrange Words in a Sentence
  - 1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence
  - 1456. Maximum Number of Vowels in a Substring of Given Length
  - 1487. Making File Names Unique
- Depth-First Search & Breadth-First Search
  - 200. Number of Islands

- 695. Max Area of Island
- 886. Possible Bipartition
- 1222. Queens That Can Attack the King
- 1489. Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
- Linked List
  - 21. Merge Two Sorted Lists
  - 61. Rotate List
  - 876. Middle of the Linked List
  - 1290. Convert Binary Number in a Linked List to Integer
  - 1474. Delete N Nodes After M Nodes of a Linked List
- Bit Manipulation
  - 338. Counting Bits
  - 342. Power of Four
  - 461. Hamming Distance
  - 476. Number Complement
  - 1486. XOR Operation in an Array
- Design Data Structure
  - 359. Logger Rate Limiter
  - 901. Online Stock Span
  - 1429. First Unique Number
  - 1472. Design Browser History
  - 1476. Subrectangle Queries

# Backtracking

## 47. Permutations II

*Description*

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

Example:

Input: [1,1,2]

Output:

```
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

*Solution*

05/05/2020:

```
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        int n = nums.size();
        if (n <= 1) return {nums};
        set<vector<int>> permuted;
        set<vector<int>> sret;
        for (int i = 0; i < n; ++i) {
            int cur = nums[i];
            swap(nums[i], nums[n - 1]);
            nums.pop_back();
            if (permuted.count(nums) > 0) {
                nums.push_back(cur);
                swap(nums[i], nums[n - 1]);
                continue;
            }
            permuted.insert(nums);
            vector<vector<int>> sub = permuteUnique(nums);
            for (auto& s : sub) {
                s.push_back(cur);
                sret.insert(s);
            }
            nums.push_back(cur);
            swap(nums[i], nums[n - 1]);
        }
        vector<vector<int>> ret(sret.begin(), sret.end());
        return ret;
    }
};
```

## 52. N-Queens II

### 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 the number of distinct solutions to the n-queens puzzle.

Example:

Input: 4

Output: 2

Explanation: There are two distinct solutions to the 4-queens puzzle as shown below.

```
[
  [".Q..", // Solution 1
   "...Q",
   "Q...",
   "..Q."],

  ["..Q.", // Solution 2
   "Q...",
   "...Q",
   ".Q.."]
]
```

### Solution

05/27/2020:

```
class Solution {
public:
    int totalNQueens(int n) {
        int ret = 0;
        vector<bool> col(n, false), diag1(2 * n - 1, false), diag2(2 * n - 1, false);
        backtrack(0, n, ret, col, diag1, diag2);
        return ret;
    }

    void backtrack(int r, int n, int& ret, vector<bool>& col, vector<bool>& diag1, vector<bool>& diag2) {
        if (r == n) {
```

```

    ++ret;
    return;
}

for (int c = 0; c < n; ++c) {
    if (!col[c] && !diag1[r + c] && !diag2[r - c + n + 1]) {
        col[c] = diag1[r + c] = diag2[r - c + n + 1] = true;
        backtrack(r + 1, n, ret, col, diag1, diag2);
        col[c] = diag1[r + c] = diag2[r - c + n + 1] = false;
    }
}
}
};

```

## 90. Subsets II

### *Description*

Given a collection of integers that might contain duplicates, `nums`, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

Input: [1,2,2]

Output:

```

[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]

```

### *Solution*

05/26/2020:

```

class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<vector<int>> ret;
        vector<int> subset;
    }
};

```



```

    sort(nums.begin(), nums.end());
    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) {
        if (i != k && nums[i] == nums[i - 1]) continue;
        subset.push_back(nums[i]);
        backtrack(i + 1, nums, subset, ret);
        subset.pop_back();
    }
}
};

```

```

class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        unordered_set<int> seen;
        vector<vector<int>> ret;
        vector<int> subset;
        backtrack(0, nums, subset, ret, seen);
        return ret;
    }

    void backtrack(int k, vector<int>& nums, vector<int>& subset,
vector<vector<int>>& ret, unordered_set<int>& seen) {
        int n = nums.size();
        if (k == n) {
            int h = hash(subset);
            if (seen.count(h) == 0) {
                seen.insert(h);
                ret.push_back(subset);
            }
            return;
        }
        subset.push_back(nums[k]);
        backtrack(k + 1, nums, subset, ret, seen);
        subset.pop_back();
        backtrack(k + 1, nums, subset, ret, seen);
    }

    int hash(vector<int>& nums) {
        long long h = 1;
        const int MOD = 1e9 + 7;

```

```
    for (auto& n : nums) {
        h = (h * 31 + n) % MOD;
    }
    return h;
}
};
```

## 1101. The Earliest Moment When Everyone Become Friends

### Description

In a social group, there are  $N$  people, with unique integer ids from  $0$  to  $N-1$ .

We have a list of logs, where each  $\text{logs}[i] = [\text{timestamp}, \text{id}_A, \text{id}_B]$  contains a non-negative integer timestamp, and the ids of two different people.

Each log represents the time in which two different people became friends. Friendship is symmetric: if  $A$  is friends with  $B$ , then  $B$  is friends with  $A$ .

Let's say that person  $A$  is acquainted with person  $B$  if  $A$  is friends with  $B$ , or  $A$  is a friend of someone acquainted with  $B$ .

Return the earliest time for which every person became acquainted with every other person. Return  $-1$  if there is no such earliest time.

Example 1:

Input:  $\text{logs} = [[20190101, 0, 1], [20190104, 3, 4], [20190107, 2, 3], [20190211, 1, 5], [20190224, 2, 4], [20190301, 0, 3], [20190312, 1, 2], [20190322, 4, 5]]$ ,  $N = 6$

Output: 20190301

Explanation:

The first event occurs at timestamp = 20190101 and after 0 and 1 become friends we have the following friendship groups [0,1], [2], [3], [4], [5].

The second event occurs at timestamp = 20190104 and after 3 and 4 become friends we have the following friendship groups [0,1], [2], [3,4], [5].

The third event occurs at timestamp = 20190107 and after 2 and 3 become friends we have the following friendship groups [0,1], [2,3,4], [5].

The fourth event occurs at timestamp = 20190211 and after 1 and 5 become friends we have the following friendship groups [0,1,5], [2,3,4].

The fifth event occurs at timestamp = 20190224 and as 2 and 4 are already friend anything happens.

The sixth event occurs at timestamp = 20190301 and after 0 and 3 become friends we have that all become friends.

Note:

$2 \leq N \leq 100$

$1 \leq \text{logs.length} \leq 10^4$

$0 \leq \text{logs}[i][0] \leq 10^9$

$0 \leq \text{logs}[i][1], \text{logs}[i][2] \leq N - 1$

It's guaranteed that all timestamps in `logs[i][0]` are different.

`logs` are not necessarily ordered by some criteria.

`logs[i][1] != logs[i][2]`

*Solution*

05/05/2020:

```
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);
    }

    bool full() {
        return sz[find(0)] == (int)sz.size();
    }

    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);
    }

    void merge(int x, int y) {
        int i = find(x);
        int 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];
    }
}
};

class Solution {
public:
    int earliestAcq(vector<vector<int>>& logs, int n) {
        if (n <= 1 || logs.empty() || logs[0].empty()) return -1;
        UnionFind uf(n);
        sort(logs.begin(), logs.end());
        for (auto& log : logs) {
            uf.merge(log[1], log[2]);
            if (uf.full()) {
                return log[0];
            }
        }
        return -1;
    }
};

```

## 1319. Number of Operations to Make Network Connected

### Description

There are  $n$  computers numbered from  $0$  to  $n-1$  connected by ethernet cables connections forming a network where  $\text{connections}[i] = [a, b]$  represents a connection between computers  $a$  and  $b$ . Any computer can reach any other computer directly or indirectly through the network.

Given an initial computer network connections. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected. Return the minimum number of times you need to do this in order to make all the computers connected. If it's not possible, return  $-1$ .

Example 1:

Input:  $n = 4$ ,  $\text{connections} = [[0,1],[0,2],[1,2]]$

Output: 1

Explanation: Remove cable between computer 1 and 2 and place between computers 1 and 3.

Example 2:

Input:  $n = 6$ ,  $\text{connections} = [[0,1],[0,2],[0,3],[1,2],[1,3]]$

Output: 2

Example 3:

Input:  $n = 6$ ,  $\text{connections} = [[0,1],[0,2],[0,3],[1,2]]$

Output: -1

Explanation: There are not enough cables.

Example 4:

Input:  $n = 5$ ,  $\text{connections} = [[0,1],[0,2],[3,4],[2,3]]$

Output: 0

Constraints:

$1 \leq n \leq 10^5$

$1 \leq \text{connections.length} \leq \min(n*(n-1)/2, 10^5)$

$\text{connections}[i].\text{length} == 2$

$0 \leq \text{connections}[i][0], \text{connections}[i][1] < n$

$\text{connections}[i][0] \neq \text{connections}[i][1]$

There are no repeated connections.

No two computers are connected by more than one cable.

*Solution*

05/06/2020:

```
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]);
    }
};
```

```

}

void merge(int x, int y) {
    int i = find(x);
    int 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];
    }
}
};

class Solution {
public:
    int makeConnected(int n, vector<vector<int>>& connections) {
        if (connections.size() < n - 1) return -1;
        UnionFind uf(n);
        for (auto& c : connections)
            uf.merge(c[0], c[1]);
        unordered_set<int> components;
        for (int i = 0; i < n; ++i)
            components.insert(uf.find(i));
        return components.size() - 1;
    }
};

```

# Binary Search

## 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*

04/23/2020:

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

## 275. H-Index II

*Description*

Given an array of citations sorted in ascending order (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 = [0,1,3,5,6]

Output: 3

Explanation: [0,1,3,5,6] means the researcher has 5 papers in total and each of them had

received 0, 1, 3, 5, 6 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.

Follow up:

This is a follow up problem to H-Index, where citations is now guaranteed to be sorted in ascending order.

Could you solve it in logarithmic time complexity?

*Solution*

05/31/2020:

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



## 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

04/23/2020:

```
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;
    }
};
```

## 441. Arranging Coins

### Description

---

You have a total of  $n$  coins that you want to form in a staircase shape, where every  $k$ -th row must have exactly  $k$  coins.

Given  $n$ , find the total number of full staircase rows that can be formed.

$n$  is a non-negative integer and fits within the range of a 32-bit signed integer.

Example 1:

$n = 5$

The coins can form the following rows:

```
✖
✖ ✖
✖ ✖
```

Because the 3rd row is incomplete, we return 2.

Example 2:

$n = 8$

The coins can form the following rows:

```
✖
✖ ✖
✖ ✖ ✖
✖ ✖
```

Because the 4th row is incomplete, we return 3.

*Solution*

04/23/2020:

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

```
};
```

Math:

```
class Solution {  
public:  
    int arrangeCoins(int n) {  
        return floor(sqrt(2.0 * n + 1.0 / 4) - 1.0/2);  
    }  
};
```

## 744. Find Smallest Letter Greater Than Target

### *Description*

Given a list of sorted characters letters containing only lowercase letters, and given a target letter target, find the smallest element in the list that is larger than the given target.

Letters also wrap around. For example, if the target is target = 'z' and letters = ['a', 'b'], the answer is 'a'.

Examples:

Input:  
letters = ["c", "f", "j"]  
target = "a"  
Output: "c"

Input:  
letters = ["c", "f", "j"]  
target = "c"  
Output: "f"

Input:  
letters = ["c", "f", "j"]  
target = "d"  
Output: "f"

Input:  
letters = ["c", "f", "j"]  
target = "g"  
Output: "j"

Input:

```
letters = ["c", "f", "j"]
target = "j"
Output: "c"
```

```
Input:
letters = ["c", "f", "j"]
target = "k"
Output: "c"
Note:
```

letters has a length in range [2, 10000].  
letters consists of lowercase letters, and contains at least 2 unique letters.  
target is a lowercase letter.

*Solution*

04/23/2020:

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

## 892. Surface Area of 3D Shapes

*Description*

On a  $N * N$  grid, we place some  $1 * 1 * 1$  cubes.

Each value  $v = \text{grid}[i][j]$  represents a tower of  $v$  cubes placed on top of grid cell  $(i, j)$ .

Return the total surface area of the resulting shapes.

Example 1:

Input: [[2]]

Output: 10

Example 2:

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

Output: 34

Example 3:

Input: [[1,0],[0,2]]

Output: 16

Example 4:

Input: [[1,1,1],[1,0,1],[1,1,1]]

Output: 32

Example 5:

Input: [[2,2,2],[2,1,2],[2,2,2]]

Output: 46

Note:

1 <= N <= 50

0 <= grid[i][j] <= 50

*Solution*

05/17/2020:

```
class Solution {
public:
    int surfaceArea(vector<vector<int>>& grid) {
        if (grid.empty() && grid[0].empty()) return 0;
        int m = grid.size(), n = grid[0].size(), area = 0;
        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) area += grid[i][j];
                if (i == m - 1) area += grid[i][j];
                if (j == 0) area += grid[i][j];
                if (j == n - 1) area += grid[i][j];
                if (grid[i][j] > 0) area += 2;
                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[i][j] > grid[ni]
[nj]) {
            area += grid[i][j] - grid[ni][nj];
        }
    }
}
return area;
}
};

```

```

class Solution {
public:
    int surfaceArea(vector<vector<int>>& grid) {
        if (grid.empty() && grid[0].empty()) return 0;
        int m = grid.size(), n = grid[0].size(), area = 0;
        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) area += 2;
                for (int d = 0; d < 4; ++d) {
                    int ni = i + dir[d][0], nj = j + dir[d][1], nv = 0;
                    if (0 <= ni && ni < m && 0 <= nj && nj < n) nv = grid[ni][nj];
                    area += max(grid[i][j] - nv, 0);
                }
            }
        }
        return area;
    }
};

```

```

class Solution {
public:
    int surfaceArea(vector<vector<int>>& grid) {
        if (grid.empty() && grid[0].empty()) return 0;
        int m = grid.size(), n = grid[0].size(), area = 0;
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (grid[i][j]) area += grid[i][j] * 4 + 2;
                if (i) area -= 2 * min(grid[i][j], grid[i - 1][j]);
                if (j) area -= 2 * min(grid[i][j], grid[i][j - 1]);
            }
        }
        return area;
    }
};

```

---

## 976. Largest Perimeter Triangle

### *Description*

Given an array A of positive lengths, return the largest perimeter of a triangle with non-zero area, formed from 3 of these lengths.

If it is impossible to form any triangle of non-zero area, return 0.

Example 1:

Input: [2,1,2]

Output: 5

Example 2:

Input: [1,2,1]

Output: 0

Example 3:

Input: [3,2,3,4]

Output: 10

Example 4:

Input: [3,6,2,3]

Output: 8

Note:

$3 \leq A.length \leq 10000$

$1 \leq A[i] \leq 10^6$

### *Solution*

05/17/2020:

```

class Solution {
public:
    int largestPerimeter(vector<int>& A) {
        sort(A.begin(), A.end(), greater<int>());
        for (int i = 0; i < (int)A.size() - 2; ++i)
            if (A[i] < A[i + 1] + A[i + 2])
                return A[i] + A[i + 1] + A[i + 2];
        return 0;
    }
};

```

## 1198. Find Smallest Common Element in All Rows

### Description

Given a matrix `mat` where every row is sorted in increasing order, return the smallest common element in all rows.

If there is no common element, return `-1`.

Example 1:

Input: `mat = [[1,2,3,4,5], [2,4,5,8,10], [3,5,7,9,11], [1,3,5,7,9]]`

Output: 5

Constraints:

`1 <= mat.length, mat[i].length <= 500`

`1 <= mat[i][j] <= 10^4`

`mat[i]` is sorted in increasing order.

### Solution

05/20/2020:

```

class Solution {
public:
    int smallestCommonElement(vector<vector<int>>& mat) {
        if (mat.empty() || mat[0].empty()) return -1;
        int m = mat.size(), n = mat[0].size();
        for (int j = 0; j < n; ++j) {
            bool isCommon = true;

```



```

    for (int i = 1; i < m; ++i) {
        if (!binary_search(mat[i].begin(), mat[i].end(), mat[0][j])) {
            isCommon = false;
            break;
        }
    }
    if (isCommon) return mat[0][j];
}
return -1;
}
};

```

```

class Solution {
public:
    int smallestCommonElement(vector<vector<int>>& mat) {
        if (mat.empty() || mat[0].empty()) return -1;
        int nrow = mat.size();
        unordered_map<int, int> mp;
        for (auto& ma : mat) {
            for (auto& m : ma) {
                if (++mp[m] == nrow) {
                    return m;
                }
            }
        }
        return -1;
    }
};

```

# Array

## 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*

01/25/2020:

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

## 73. Set Matrix Zeroes

*Description*

Given a  $m \times n$  matrix, if an element is 0, set its entire row and column to 0. Do it in-place.

Example 1:

Input:

```
[
  [1,1,1],
  [1,0,1],
```

```
[1,1,1]
]
```

Output:

```
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
```

Example 2:

Input:

```
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
```

Output:

```
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

Follow up:

A straight forward solution using  $O(mn)$  space is probably a bad idea.  
A simple improvement uses  $O(m + n)$  space, but still not the best solution.  
Could you devise a constant space solution?

*Solution*

05/23/2020:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        if (matrix.empty() || matrix[0].empty()) return;
        int nrow = matrix.size(), ncol = matrix[0].size();
        vector<pair<int, int>> zeros;
        vector<bool> isRowZero(nrow, true);
        vector<bool> isColZero(ncol, true);
        for (int i = 0; i < nrow; ++i) {
            for (int j = 0; j < ncol; ++j) {
                if (matrix[i][j] == 0) {
                    zeros.emplace_back(i, j);
                } else {
                    isRowZero[i] = false;
                    isColZero[j] = false;
                }
            }
        }
    }
};
```

```

    }
}
for (auto& p : zeros) {
    int i = p.first, j = p.second;
    if (!isRowZero[i]) {
        for (int k = 0; k < ncol; ++k) matrix[i][k] = 0;
        isRowZero[i] = true;
    }
    if (!isColZero[j]) {
        for (int k = 0; k < nrow; ++k) matrix[k][j] = 0;
        isColZero[j] = true;
    }
}
}
};

```

## 167. Two Sum II - Input array is sorted

### Description

Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

Note:

Your returned answers (both index1 and index2) are not zero-based.

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

Example:

Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.

### Solution

01/25/2020:

```

class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        for (int l = 0, r = numbers.size() - 1; l < r;) {

```

```

    if (numbers[l] + numbers[r] == target) {
        return {l + 1, r + 1};
    } else if (numbers[l] + numbers[r] < target) {
        ++l;
    } else {
        --r;
    }
}
return {-1, -1};
}
};

```

## 170. Two Sum III - Data structure design

### Description

Design and implement a TwoSum class. It should support the following operations: add and find.

add – Add the number to an internal data structure.

find – Find if there exists any pair of numbers which sum is equal to the value.

Example 1:

```

add(1); add(3); add(5);
find(4) -> true
find(7) -> false

```

Example 2:

```

add(3); add(1); add(2);
find(3) -> true
find(6) -> false

```

### Solution

01/25/2020:

```

class TwoSum {
public:
    vector<int> nums;
    /** Initialize your data structure here. */
    TwoSum() {
    }

    /** Add the number to an internal data structure.. */

```

```

void add(int number) {
    nums.push_back(number);
}

/** Find if there exists any pair of numbers which sum is equal to the value.
*/
bool find(int value) {
    unordered_set<int> seen;
    for (int i = 0; i < (int)nums.size(); ++i) {
        if (seen.find(value - nums[i]) == seen.end()) {
            seen.insert(nums[i]);
        } else {
            return true;
        }
    }
    return false;
}
};

/**
 * Your TwoSum object will be instantiated and called as such:
 * TwoSum* obj = new TwoSum();
 * obj->add(number);
 * bool param_2 = obj->find(value);
 */

```

01/25/2020:

```

class TwoSum {
private:
    unordered_map<int, int> m;

public:
    /** Initialize your data structure here. */
    TwoSum() {
    }

    /** Add the number to an internal data structure.. */
    void add(int number) {
        ++m[number];
    }

    /** Find if there exists any pair of numbers which sum is equal to the value.
    */
    bool find(int value) {
        for (auto& n : m) {
            if ((value == n.first * 2 && n.second > 1) || (value != n.first * 2 &&
m.count(value - n.first) > 0))

```

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

/**
 * Your TwoSum object will be instantiated and called as such:
 * TwoSum* obj = new TwoSum();
 * obj->add(number);
 * bool param_2 = obj->find(value);
 */
```

## 463. Island Perimeter

### *Description*

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water.

Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example:

Input:  
[[0,1,0,0],  
 [1,1,1,0],  
 [0,1,0,0],  
 [1,1,0,0]]

Output: 16

Explanation: The perimeter is the 16 yellow stripes in the image below:

### *Solution*

01/26/2020:

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

## 509. Fibonacci Number

### Description

The Fibonacci numbers, commonly denoted  $F(n)$  form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F(0) = 0, \quad F(1) = 1$$

$$F(N) = F(N - 1) + F(N - 2), \text{ for } N > 1.$$

Given  $N$ , calculate  $F(N)$ .

Example 1:

Input: 2

Output: 1

Explanation:  $F(2) = F(1) + F(0) = 1 + 0 = 1$ .

Example 2:



Input: 3  
Output: 2  
Explanation:  $F(3) = F(2) + F(1) = 1 + 1 = 2$ .  
Example 3:

Input: 4  
Output: 3  
Explanation:  $F(4) = F(3) + F(2) = 2 + 1 = 3$ .

Note:

$0 \leq N \leq 30$ .

*Solution*

01/31/2020:

```
class Solution {
public:
    int fib(int N) {
        if (N == 0) return 0;
        if (N == 1) return 1;
        int f0 = 0, f1 = 1;
        for (int i = 2; i <= N; ++i) {
            f0 = f0 + f1;
            swap(f0, f1);
        }
        return f1;
    }
};
```

## 561. Array Partition I

*Description*

Given an array of  $2n$  integers, your task is to group these integers into  $n$  pairs of integer, say  $(a_1, b_1)$ ,  $(a_2, b_2)$ , ...,  $(a_n, b_n)$  which makes sum of  $\min(a_i, b_i)$  for all  $i$  from 1 to  $n$  as large as possible.

Example 1:

Input: [1,4,3,2]

Output: 4

Explanation:  $n$  is 2, and the maximum sum of pairs is  $4 = \min(1, 2) + \min(3, 4)$ .

Note:

$n$  is a positive integer, which is in the range of  $[1, 10000]$ .

All the integers in the array will be in the range of  $[-10000, 10000]$ .

*Solution*

01/30/2020:

```
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int ret = 0;
        for (int i = 0; i < (int)nums.size(); i += 2) {
            ret += nums[i];
        }
        return ret;
    }
};
```

## 566. Reshape the Matrix

*Description*

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two positive integers  $r$  and  $c$  representing the row number and column number of the wanted reshaped matrix, respectively.

The reshaped matrix need to be filled with all the elements of the original matrix in the same row-traversing order as they were.

If the 'reshape' operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

Example 1:

Input:

```
nums =  
[[1,2],  
 [3,4]]  
r = 1, c = 4
```

Output:

```
[[1,2,3,4]]
```

Explanation:

The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 \* 4 matrix, fill it row by row by using the previous list.

Example 2:

Input:

```
nums =  
[[1,2],  
 [3,4]]  
r = 2, c = 4
```

Output:

```
[[1,2],  
 [3,4]]
```

Explanation:

There is no way to reshape a 2 \* 2 matrix to a 2 \* 4 matrix. So output the original matrix.

Note:

The height and width of the given matrix is in range [1, 100].  
The given r and c are all positive.

*Solution*

05/10/2020:

```
class Solution {  
public:  
    vector<vector<int>> matrixReshape(vector<vector<int>>& nums, int r, int c) {  
        if (nums.empty() && nums[0].empty()) return { {} };  
        int m = nums.size(), n = nums[0].size();  
        if (m * n != r * c) return nums;  
        vector<vector<int>> ret(r, vector<int>(c, 0));  
        for (int i = 0; i < r; ++i) {  
            for (int j = 0; j < c; ++j) {  
                int idx = i * c + j;  
                ret[i][j] = nums[idx / n][(idx % n)];  
            }  
        }  
    }  
}
```

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

## 594. Longest Harmonious Subsequence

### Description

We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1.

Now, given an integer array, you need to find the length of its longest harmonious subsequence among all its possible subsequences.

Example 1:

Input: [1,3,2,2,5,2,3,7]

Output: 5

Explanation: The longest harmonious subsequence is [3,2,2,2,3].

### Solution

05/18/2020:

```
class Solution {
public:
    int findLHS(vector<int>& nums) {
        if (nums.empty()) return 0;
        map<int, int> cnt;
        for (auto& n : nums) ++cnt[n];
        int ret = 0;
        for (auto& m : cnt)
            if (cnt[m.first - 1] > 0 && cnt[m.first] > 0)
                ret = max(ret, cnt[m.first - 1] + cnt[m.first]);
        return ret;
    }
};
```

## 624. Maximum Distance in Arrays

### Description

Given  $m$  arrays, and each array is sorted in ascending order. Now you can pick up two integers from two different arrays (each array picks one) and calculate the distance. We define the distance between two integers  $a$  and  $b$  to be their absolute difference  $|a-b|$ . Your task is to find the maximum distance.

Example 1:

Input:

```
[[1,2,3],  
 [4,5],  
 [1,2,3]]
```

Output: 4

Explanation:

One way to reach the maximum distance 4 is to pick 1 in the first or third array and pick 5 in the second array.

Note:

Each given array will have at least 1 number. There will be at least two non-empty arrays.

The total number of the integers in all the  $m$  arrays will be in the range of  $[2, 10000]$ .

The integers in the  $m$  arrays will be in the range of  $[-10000, 10000]$ .

*Solution*

05/20/2020:

```
class Solution {  
public:  
    int maxDistance(vector<vector<int>>& arrays) {  
        int n = arrays.size();  
        vector<pair<int, int>> mins, maxs;  
        int ret = 0;  
        for (int i = 0; i < n; ++i) {  
            mins.emplace_back(arrays[i].front(), i);  
            maxs.emplace_back(arrays[i].back(), i);  
        }  
        sort(mins.begin(), mins.end());  
        sort(maxs.begin(), maxs.end(), greater<pair<int, int>>());  
        ret = maxs.front().second != mins.front().second ? maxs.front().first -  
        mins.front().first : max(maxs[1].first - mins[0].first, maxs[0].first -  
        mins[1].first);  
        return ret;  
    }  
};
```

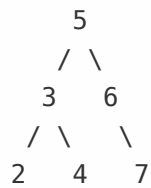
## 653. Two Sum IV - Input is a BST

### Description

Given a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Example 1:

Input:

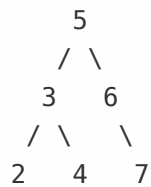


Target = 9

Output: True

Example 2:

Input:



Target = 28

Output: False

### Solution

01/25/2020:

```
/**
 * 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:
    bool findTarget(TreeNode* root, int k) {
        vector<int> nums;
        tree2vector(root, nums);
        for (int l = 0, r = nums.size() - 1; l < r;) {
            int s = nums[l] + nums[r];
            if (s == k) {
                return true;
            } else if (s < k) {
                ++l;
            } else {
                --r;
            }
        }
        return false;
    }
    void tree2vector(TreeNode* root, vector<int>& nums) {
        if (root == nullptr) return;
        tree2vector(root->left, nums);
        nums.push_back(root->val);
        tree2vector(root->right, nums);
    }
};

```

01/25/2019:

```

class Solution {
public:
    unordered_set<int> seen;
    bool findTarget(TreeNode* root, int k) {
        if (root == nullptr) return false;
        if (seen.find(k - root->val) == seen.end()) {
            seen.insert(root->val);
        } else {
            return true;
        }
        return findTarget(root->left, k) || findTarget(root->right, k);
    }
};

```

## 654. Maximum Binary Tree

*Description*

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

The root is the maximum number in the array.

The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.

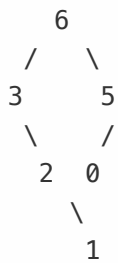
The right subtree is the maximum tree constructed from right part subarray divided by the maximum number.

Construct the maximum tree by the given array and output the root node of this tree.

Example 1:

Input: [3,2,1,6,0,5]

Output: return the tree root node representing the following tree:



Note:

The size of the given array will be in the range [1,1000].

*Solution*

**Discussion:** Mimicking `merge_sort`, we can split the array into left subarray and right subarray which are separated by the maximum of the array. The subtrees constructed by left subarray and right subarray becomes left subtree and right subtree, respectively. Time Complexity:  $O(n^2)$  and Space Complexity:  $O(n^2)$ .



```

class Solution {
public:
    TreeNode* constructMaximumBinaryTree(vector<int>& nums) {
        if (nums.size() == 0) return nullptr;
        auto max_iter = max_element(nums.begin(), nums.end());
        TreeNode* root = new TreeNode(*max_iter);
        vector<int> nums_left(nums.begin(), max_iter);
        vector<int> nums_right(max_iter + 1, nums.end());
        root->left = constructMaximumBinaryTree(nums_left);
        root->right = constructMaximumBinaryTree(nums_right);
        return root;
    }
};

```

It could be more space efficient with Space Complexity  $O(n)$ :

```

class Solution {
public:
    TreeNode* constructMaximumBinaryTree(vector<int>& nums, int start = 0, int
stop = INT_MAX) {
        if (stop == INT_MAX) stop = nums.size() - 1;
        if (start > stop) return nullptr;
        auto middle = max_element(nums.begin() + start, nums.begin() + stop + 1) -
nums.begin();
        TreeNode* root = new TreeNode(nums[middle]);
        root->left = constructMaximumBinaryTree(nums, start, middle - 1);
        root->right = constructMaximumBinaryTree(nums, middle + 1, stop);
        return root;
    }
};

```

## 623. Add One Row to Tree

### *Description*

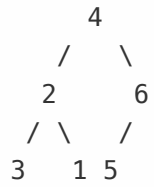
Given the root of a binary tree, then value  $v$  and depth  $d$ , you need to add a row of nodes with value  $v$  at the given depth  $d$ . The root node is at depth 1.

The adding rule is: given a positive integer depth  $d$ , for each NOT null tree nodes  $N$  in depth  $d-1$ , create two tree nodes with value  $v$  as  $N$ 's left subtree root and right subtree root. And  $N$ 's original left subtree should be the left subtree of the new left subtree root, its original right subtree should be the right subtree of the new right subtree root. If depth  $d$  is 1 that means there is no depth  $d-1$  at all, then create a tree node with value  $v$  as the new root of the whole original tree, and the original tree is the new root's left subtree.

Example 1:

Input:

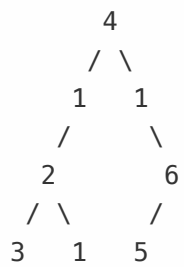
A binary tree as following:



v = 1

d = 2

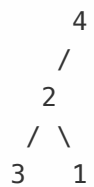
Output:



Example 2:

Input:

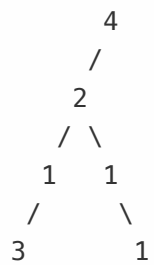
A binary tree as following:



v = 1

d = 3

Output:



Note:

The given d is in range [1, maximum depth of the given tree + 1].

The given binary tree has at least one tree node.

Solution

06/09/2020:

```
/**
 * 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* addOneRow(TreeNode* root, int v, int d) {
        if (d == 1) return new TreeNode(v, root, nullptr);
        queue<TreeNode*> q;
        q.emplace(root);
        while (!q.empty()) {
            --d;
            int sz = q.size();
            for (int s = 0; s < sz; ++s) {
                TreeNode* cur = q.front(); q.pop();
                if (d == 1) {
                    cur->left = cur->left ? new TreeNode(v, cur->left, nullptr) : new
TreeNode(v);
                    cur->right = cur->right ? new TreeNode(v, nullptr, cur->right) : new
TreeNode(v);
                } else {
                    if (cur->left) q.push(cur->left);
                    if (cur->right) q.push(cur->right);
                }
            }
        }
        return root;
    }
};
```

## 695. Max Area of Island

---

*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.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1:

```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

Given the above grid, return 6. Note the answer is not 11, because the island must be connected 4-directionally.

Example 2:

```
[[0,0,0,0,0,0,0,0]]
```

Given the above grid, return 0.

Note: The length of each dimension in the given grid does not exceed 50.

*Solution*

01/26/2020:

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

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

```

    grid[i][j] = 0;
    if (i > 0) dfs(grid, i - 1, j, area);
    if (j > 0) dfs(grid, i, j - 1, area);
    if (i < m - 1) dfs(grid, i + 1, j, area);
    if (j < n - 1) dfs(grid, i, j + 1, area);
}
}
};

```

## 760. Find Anagram Mappings

### Description

Given two lists A and B, and B is an anagram of A. B is an anagram of A means B is made by randomizing the order of the elements in A.

We want to find an index mapping P, from A to B. A mapping  $P[i] = j$  means the  $i$ th element in A appears in B at index  $j$ .

These lists A and B may contain duplicates. If there are multiple answers, output any of them.

For example, given

A = [12, 28, 46, 32, 50]

B = [50, 12, 32, 46, 28]

We should return

[1, 4, 3, 2, 0]

as  $P[0] = 1$  because the 0th element of A appears at B[1], and  $P[1] = 4$  because the 1st element of A appears at B[4], and so on.

Note:

A, B have equal lengths in range [1, 100].

$A[i]$ ,  $B[i]$  are integers in range [0,  $10^5$ ].

### Solution

01/29/2020:

```

class Solution {
public:
    vector<int> anagramMappings(vector<int>& A, vector<int>& B) {
        unordered_map<int, int> m;
        vector<int> ret(A.size(), 0);
        for (int i = 0; i < B.size(); ++i) {

```

```

        m[B[i]] = i;
    }
    for (int i = 0; i < (int)A.size(); ++i) {
        ret[i] = m[A[i]];
    }
    return ret;
}
};

```

## 766. Toeplitz Matrix

### Description

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same element.

Now given an  $M \times N$  matrix, return True if and only if the matrix is Toeplitz.

Example 1:

Input:

```

matrix = [
  [1,2,3,4],
  [5,1,2,3],
  [9,5,1,2]
]

```

Output: True

Explanation:

In the above grid, the diagonals are:

"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".

In each diagonal all elements are the same, so the answer is True.

Example 2:

Input:

```

matrix = [
  [1,2],
  [2,2]
]

```

Output: False

Explanation:

The diagonal "[1, 2]" has different elements.

Note:

matrix will be a 2D array of integers.

matrix will have a number of rows and columns in range [1, 20].  
matrix[i][j] will be integers in range [0, 99].

Follow up:

What if the matrix is stored on disk, and the memory is limited such that you can only load at most one row of the matrix into the memory at once?

What if the matrix is so large that you can only load up a partial row into the memory at once?

*Solution*

02/03/2020:

```
class Solution {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        for (int i = 0; i < (int)matrix.size(); ++i) {
            for (int k = 1; k < (int)matrix[0].size(); ++k) {
                if (i + k < (int)matrix.size() && matrix[i + k][k] != matrix[i + k - 1][k - 1]) {
                    return false;
                }
            }
        }
        for (int j = 0; j < (int)matrix[0].size(); ++j) {
            for (int k = 1; k < (int)matrix.size(); ++k) {
                if (j + k < (int)matrix[0].size() && matrix[k][j + k] != matrix[k - 1][j + k - 1]) {
                    return false;
                }
            }
        }
        return true;
    }
};
```

## 811. Subdomain Visit Count

---

*Description*

A website domain like "discuss.leetcode.com" consists of various subdomains. At the top level, we have "com", at the next level, we have "leetcode.com", and at the lowest level, "discuss.leetcode.com". When we visit a domain like "discuss.leetcode.com", we will also visit the parent domains "leetcode.com" and "com" implicitly.

Now, call a "count-paired domain" to be a count (representing the number of visits this domain received), followed by a space, followed by the address. An example of a count-paired domain might be "9001 discuss.leetcode.com".

We are given a list cpdomains of count-paired domains. We would like a list of count-paired domains, (in the same format as the input, and in any order), that explicitly counts the number of visits to each subdomain.

Example 1:

Input:

```
["9001 discuss.leetcode.com"]
```

Output:

```
["9001 discuss.leetcode.com", "9001 leetcode.com", "9001 com"]
```

Explanation:

We only have one website domain: "discuss.leetcode.com". As discussed above, the subdomain "leetcode.com" and "com" will also be visited. So they will all be visited 9001 times.

Example 2:

Input:

```
["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]
```

Output:

```
["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org","1 intel.mail.com","951 com"]
```

Explanation:

We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com" once and "wiki.org" 5 times. For the subdomains, we will visit "mail.com"  $900 + 1 = 901$  times, "com"  $900 + 50 + 1 = 951$  times, and "org" 5 times.

Notes:

The length of cpdomains will not exceed 100.

The length of each domain name will not exceed 100.

Each address will have either 1 or 2 "." characters.

The input count in any count-paired domain will not exceed 10000.

The answer output can be returned in any order.

*Solution*

01/31/2020:



```

class Solution {
public:
    vector<string> subdomainVisits(vector<string>& cpdomains) {
        unordered_map<string, int> cnt;
        for (auto& s : cpdomains) {
            istringstream iss(s);
            int n;
            string str;
            iss >> n >> str;
            while (str.size() > 0) {
                // cout << n << " " << str << endl;
                cnt[str] += n;
                int i = 0;
                for (; i < (int)str.size(); ++i) {
                    if (str[i] == '.') {
                        break;
                    }
                }
                if (i < str.size()) {
                    str = str.substr(i + 1, str.size() - i - 1);
                } else {
                    str = "";
                }
            }
        }
        vector<string> ret;
        for (auto it = cnt.begin(); it != cnt.end(); ++it) {
            // cout << it->first << " " << it->second << endl;
            ret.push_back(to_string(it->second) + " " + it->first);
        }
        return ret;
    }
};

```

## 832. Flipping an Image

### *Description*

Given a binary matrix A, we want to flip the image horizontally, then invert it, and return the resulting image.

To flip an image horizontally means that each row of the image is reversed. For example, flipping [1, 1, 0] horizontally results in [0, 1, 1].

To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0. For example, inverting [0, 1, 1] results in [1, 0, 0].

Example 1:

Input: [[1,1,0],[1,0,1],[0,0,0]]

Output: [[1,0,0],[0,1,0],[1,1,1]]

Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].

Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]

Example 2:

Input: [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]

Output: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

Explanation: First reverse each row: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]].

Then invert the image: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

Notes:

1 <= A.length = A[0].length <= 20

0 <= A[i][j] <= 1

*Solution*

01/29/2020:

```
class Solution {
public:
    vector<vector<int>> flipAndInvertImage(vector<vector<int>>& A) {
        int m = A.size(), n = A[0].size();
        bool even_col = n % 2 == 1;
        for (int i = 0; i < m; ++i) {
            for (int l = 0, r = n - 1; l < r; ++l, --r) {
                swap(A[i][l], A[i][r]);
                A[i][l] ^= 1;
                A[i][r] ^= 1;
            }
            if (even_col) {
                A[i][n / 2] ^= 1;
            }
        }
        return A;
    }
};
```

852. Peak Index in a Mountain Array

## Description

Let's call an array A a mountain if the following properties hold:

A.length  $\geq$  3

There exists some  $0 < i < A.length - 1$  such that  $A[0] < A[1] < \dots < A[i-1] < A[i] > A[i+1] > \dots > A[A.length - 1]$

Given an array that is definitely a mountain, return any i such that  $A[0] < A[1] < \dots < A[i-1] < A[i] > A[i+1] > \dots > A[A.length - 1]$ .

Example 1:

Input: [0,1,0]

Output: 1

Example 2:

Input: [0,2,1,0]

Output: 1

Note:

$3 \leq A.length \leq 10000$

$0 \leq A[i] \leq 10^6$

A is a mountain, as defined above.

## Solution

01/30/2020:

```
class Solution {
public:
    int peakIndexInMountainArray(vector<int>& A) {
        for (int i = 0; i < (int)A.size() - 1; ++i) {
            if (A[i] > A[i + 1]) {
                return i;
            }
        }
        return A.size() - 1;
    }
};
```

## 867. Transpose Matrix

## Description

Given a matrix A, return the transpose of A.

The transpose of a matrix is the matrix flipped over its main diagonal, switching the row and column indices of the matrix.

Example 1:

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

Example 2:

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

Note:

1 <= A.length <= 1000  
1 <= A[0].length <= 1000

*Solution*

02/03/2020:

```
class Solution {
public:
    vector<vector<int>> transpose(vector<vector<int>>& A) {
        int m = A.size(), n = A[0].size();
        vector<vector<int>> ret(n, vector<int>(m, 0));
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                ret[j][i] = A[i][j];
            }
        }
        return ret;
    }
};
```

## 905. Sort Array By Parity

*Description*

Given an array A of non-negative integers, return an array consisting of all the even elements of A, followed by all the odd elements of A.

You may return any answer array that satisfies this condition.

Example 1:

Input: [3,1,2,4]

Output: [2,4,3,1]

The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.

Note:

$1 \leq A.length \leq 5000$

$0 \leq A[i] \leq 5000$

*Solution*

01/30/2020:

```
class Solution {
public:
    vector<int> sortArrayByParity(vector<int>& A) {
        vector<int> odd;
        vector<int> even;
        for (auto& n : A) {
            if (n % 2 == 1) {
                odd.push_back(n);
            } else {
                even.push_back(n);
            }
        }
        even.insert(even.end(), odd.begin(), odd.end());
        return even;
    }
};
```

## 918. Maximum Sum Circular Subarray

*Description*

Given a circular array C of integers represented by A, find the maximum possible sum of a non-empty subarray of C.

Here, a circular array means the end of the array connects to the beginning of the array. (Formally,  $C[i] = A[i]$  when  $0 \leq i < A.length$ , and  $C[i+A.length] = C[i]$  when  $i \geq 0$ .)

Also, a subarray may only include each element of the fixed buffer A at most once. (Formally, for a subarray  $C[i], C[i+1], \dots, C[j]$ , there does not exist  $i \leq k_1, k_2 \leq j$  with  $k_1 \% A.length = k_2 \% A.length$ .)

Example 1:

Input: [1,-2,3,-2]

Output: 3

Explanation: Subarray [3] has maximum sum 3

Example 2:

Input: [5,-3,5]

Output: 10

Explanation: Subarray [5,5] has maximum sum  $5 + 5 = 10$

Example 3:

Input: [3,-1,2,-1]

Output: 4

Explanation: Subarray [2,-1,3] has maximum sum  $2 + (-1) + 3 = 4$

Example 4:

Input: [3,-2,2,-3]

Output: 3

Explanation: Subarray [3] and [3,-2,2] both have maximum sum 3

Example 5:

Input: [-2,-3,-1]

Output: -1

Explanation: Subarray [-1] has maximum sum -1

Note:

$-30000 \leq A[i] \leq 30000$

$1 \leq A.length \leq 30000$

*Solution*

05/17/2020:

```

class Solution {
public:
    int maxSubarraySumCircular(vector<int>& A) {
        int cur_max = A.front(), s = 0;
        int n = A.size();
        for (int i = 0; i < n; ++i) {
            s = max(s, 0) + A[i];
            cur_max = max(cur_max, s);
        }
        vector<int> prefix_sum(A);
        vector<int> suffix_sum(A);
        for (int i = 1; i < n; ++i) {
            prefix_sum[i] += prefix_sum[i - 1];
            suffix_sum[n - 1 - i] += suffix_sum[n - i];
        }
        for (int i = 1; i < n; ++i) {
            prefix_sum[i] = max(prefix_sum[i], prefix_sum[i - 1]);
            suffix_sum[n - 1 - i] = max(suffix_sum[n - 1 - i], suffix_sum[n - i]);
        }
        for (int i = 1; i < n; ++i) {
            cur_max = max(cur_max, prefix_sum[i - 1] + suffix_sum[i]);
        }
        return cur_max;
    }
};

```

```

class Solution {
public:
    int maxSubarraySumCircular(vector<int>& A) {
        int cur_max = A.front(), s = 0;
        int n = A.size();
        for (int i = 0; i < n; ++i) {
            s = max(s, 0) + A[i];
            cur_max = max(cur_max, s);
        }
        vector<int> prefix_sum(A), suffix_sum(A);
        partial_sum(A.cbegin(), A.cend(), prefix_sum.begin(), plus<int>());
        partial_sum(A.crbegin(), A.crend(), suffix_sum.rbegin(), plus<int>());
        for (int i = 1; i < n; ++i) {
            prefix_sum[i] = max(prefix_sum[i], prefix_sum[i - 1]);
            suffix_sum[n - 1 - i] = max(suffix_sum[n - 1 - i], suffix_sum[n - i]);
            cur_max = max(cur_max, prefix_sum[i - 1] + suffix_sum[i]);
        }
        return cur_max;
    }
};

```

## 922. Sort Array By Parity II

### Description

Given an array A of non-negative integers, half of the integers in A are odd, and half of the integers are even.

Sort the array so that whenever A[i] is odd, i is odd; and whenever A[i] is even, i is even.

You may return any answer array that satisfies this condition.

Example 1:

Input: [4,2,5,7]

Output: [4,5,2,7]

Explanation: [4,7,2,5], [2,5,4,7], [2,7,4,5] would also have been accepted.

Note:

$2 \leq A.length \leq 20000$

$A.length \% 2 == 0$

$0 \leq A[i] \leq 1000$

### Solution

01/30/2020:

```
class Solution {
public:
    vector<int> sortArrayByParityII(vector<int>& A) {
        int n = A.size();
        vector<int> ret(n, 0);
        for (int i = 0, even = 0, odd = 1; i < n; ++i) {
            if (A[i] % 2 == 1) {
                ret[odd] = A[i];
                odd += 2;
            } else {
                ret[even] = A[i];
                even += 2;
            }
        }
        return ret;
    }
}
```



```
};
```

## 961. N-Repeated Element in Size 2N Array

### *Description*

In a array A of size 2N, there are N+1 unique elements, and exactly one of these elements is repeated N times.

Return the element repeated N times.

Example 1:

Input: [1,2,3,3]

Output: 3

Example 2:

Input: [2,1,2,5,3,2]

Output: 2

Example 3:

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

Output: 5

Note:

$4 \leq A.length \leq 10000$

$0 \leq A[i] < 10000$

A.length is even

### *Solution*

01/30/2020:

```

class Solution {
public:
    int repeatedNTimes(vector<int>& A) {
        unordered_map<int, int> m;
        for (auto& n : A) {
            if (++m[n] > 1) {
                return n;
            }
        }
        return -1;
    }
};

```

## 977. Squares of a Sorted Array

### *Description*

Given an array of integers A sorted in non-decreasing order, return an array of the squares of each number, also in sorted non-decreasing order.

Example 1:

Input: [-4,-1,0,3,10]

Output: [0,1,9,16,100]

Example 2:

Input: [-7,-3,2,3,11]

Output: [4,9,9,49,121]

Note:

1 <= A.length <= 10000

-10000 <= A[i] <= 10000

A is sorted in non-decreasing order.

### *Solution*

01/30/2020:

```

class Solution {
public:
    vector<int> sortedSquares(vector<int>& A) {

```

```

vector<int> ret;
for (int l = 0, r = A.size() - 1; l <= r;) {
    if (abs(A[l]) > abs(A[r])) {
        ret.push_back(pow(A[l], 2));
        ++l;
    } else {
        ret.push_back(pow(A[r], 2));
        --r;
    }
}
reverse(ret.begin(), ret.end());
return ret;
}
};

```

## 994. Rotting Oranges

### *Description*

In a given grid, each cell can have one of three values:

the value 0 representing an empty cell;  
the value 1 representing a fresh orange;  
the value 2 representing a rotten orange.

Every minute, any fresh orange that is adjacent (4-directionally) to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return -1 instead.

Example 1:

Input: [[2,1,1],[1,1,0],[0,1,1]]

Output: 4

Example 2:

Input: [[2,1,1],[0,1,1],[1,0,1]]

Output: -1

Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Example 3:

Input: [[0,2]]

Output: 0

Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

Note:

1 <= grid.length <= 10

1 <= grid[0].length <= 10

grid[i][j] is only 0, 1, or 2.

*Solution*

05/07/2020:

```
class Solution {
public:
    int orangesRotting(vector<vector<int>>& grid) {
        if (grid.empty() || grid[0].empty()) return 0;
        int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
        int fresh = 0, count = 0, m = grid.size(), n = grid[0].size();
        queue<pair<int, int>> q;
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (grid[i][j] == 1) {
                    ++fresh;
                } else if (grid[i][j] == 2) {
                    q.emplace(i, j);
                }
            }
        }
        if (fresh == 0) return 0;
        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], nj = j + dir[d][1];
                    if (ni >= 0 && ni < m && nj >= 0 && nj < n && grid[ni][nj] == 1) {
                        --fresh;
                        grid[ni][nj] = 2;
                        q.emplace(ni, nj);
                    }
                }
            }
        }
        ++count;
    }
};
```

```
    }  
    return fresh > 0 ? -1 : count - 1;  
  }  
};
```

## 1051. Height Checker

### Description

Students are asked to stand in non-decreasing order of heights for an annual photo.

Return the minimum number of students that must move in order for all students to be standing in non-decreasing order of height.

Example 1:

Input: heights = [1,1,4,2,1,3]

Output: 3

Constraints:

$1 \leq \text{heights.length} \leq 100$

$1 \leq \text{heights}[i] \leq 100$

### Solution

01/31/2020:

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

```
};
```

## 1064. Fixed Point

### Description

Given an array  $A$  of distinct integers sorted in ascending order, return the smallest index  $i$  that satisfies  $A[i] == i$ . Return  $-1$  if no such  $i$  exists.

Example 1:

Input:  $[-10, -5, 0, 3, 7]$

Output: 3

Explanation:

For the given array,  $A[0] = -10$ ,  $A[1] = -5$ ,  $A[2] = 0$ ,  $A[3] = 3$ , thus the output is 3.

Example 2:

Input:  $[0, 2, 5, 8, 17]$

Output: 0

Explanation:

$A[0] = 0$ , thus the output is 0.

Example 3:

Input:  $[-10, -5, 3, 4, 7, 9]$

Output: -1

Explanation:

There is no such  $i$  that  $A[i] = i$ , thus the output is -1.

Note:

$1 \leq A.length < 10^4$

$-10^9 \leq A[i] \leq 10^9$

### Solution

01/30/2020:

```
class Solution {
public:
    int fixedPoint(vector<int>& A) {
        int lo = 0, hi = A.size() - 1;
```

```

while (lo < hi) {
    int mi = (lo + hi) >> 1;
    if (A[mi] < mi) {
        lo = mi + 1;
    } else {
        hi = mi;
    }
}
return A[lo] == lo ? lo : -1;
}
};

```

```

class Solution {
public:
    int fixedPoint(vector<int>& A) {
        int lo = 0, hi = A.size() - 1, mi = (lo + hi) / 2;
        while (lo < hi) {
            if (A[mi] > mi) {
                hi = mi - 1;
            } else if (A[mi] < mi) {
                lo = mi + 1;
            } else {
                break;
            }
            mi = (lo + hi) / 2;
        }
        for (; mi >= 0 && A[mi] == mi; --mi);
        return mi + 1 >= A.size() || A[mi + 1] != mi + 1 ? -1 : mi + 1;
    }
};

```

## 1085. Sum of Digits in the Minimum Number

### Description

Given an array A of positive integers, let S be the sum of the digits of the minimal element of A.

Return 0 if S is odd, otherwise return 1.

Example 1:

Input: [34,23,1,24,75,33,54,8]

Output: 0

Explanation:

The minimal element is 1, and the sum of those digits is  $S = 1$  which is odd, so the answer is 0.

Example 2:

Input: [99,77,33,66,55]

Output: 1

Explanation:

The minimal element is 33, and the sum of those digits is  $S = 3 + 3 = 6$  which is even, so the answer is 1.

Note:

$1 \leq A.length \leq 100$

$1 \leq A[i].length \leq 100$

*Solution*

01/30/2020:

```
class Solution {
public:
    int sumOfDigits(vector<int>& A) {
        int m = *min_element(A.begin(), A.end());
        string s = to_string(m);
        int t = 0;
        for (auto& c : s) {
            t += c - '0';
        }
        return t % 2 == 1 ? 0 : 1;
    }
};
```

## 1086. High Five

*Description*

Given a list of scores of different students, return the average score of each student's top five scores in the order of each student's id.

Each entry `items[i]` has `items[i][0]` the student's id, and `items[i][1]` the student's score. The average score is calculated using integer division.



Example 1:

Input: [[1,91],[1,92],[2,93],[2,97],[1,60],[2,77],[1,65],[1,87],[1,100],[2,100],[2,76]]

Output: [[1,87],[2,88]]

Explanation:

The average of the student with id = 1 is 87.

The average of the student with id = 2 is 88.6. But with integer division their average converts to 88.

Note:

1 <= items.length <= 1000

items[i].length == 2

The IDs of the students is between 1 to 1000

The score of the students is between 1 to 100

For each student, there are at least 5 scores

*Solution*

01/30/2020:

```
class Solution {
public:
    vector<vector<int>> highFive(vector<vector<int>>& items) {
        sort(items.begin(), items.end(), [](vector<int>& a, vector<int>& b){
            if (a[0] != b[0]) {
                return a[0] < b[0];
            } else {
                return a[1] > b[1];
            }
        });
        // for (auto& item : items) {
        //     cout << item[0] << " " << item[1] << endl;
        // }
        vector<vector<int>> ret;
        for (int i = 0, id = items[0][0]; i < (int)items.size(); id = items[i][0]) {
            int scores = 0;
            for (int j = 0; j < 5; ++j) {
                scores += items[i + j][1];
            }
            ret.push_back({id, scores / 5});
            for (i += 4; i < (int)items.size() && id == items[i][0]; ++i);
            if (i >= items.size()) break;
        }
    }
};
```

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

## 1099. Two Sum Less Than K

### Description

Given an array A of integers and integer K, return the maximum S such that there exists  $i < j$  with  $A[i] + A[j] = S$  and  $S < K$ . If no  $i, j$  exist satisfying this equation, return -1.

Example 1:

Input: A = [34,23,1,24,75,33,54,8], K = 60

Output: 58

Explanation:

We can use 34 and 24 to sum 58 which is less than 60.

Example 2:

Input: A = [10,20,30], K = 15

Output: -1

Explanation:

In this case it's not possible to get a pair sum less than 15.

Note:

$1 \leq A.length \leq 100$

$1 \leq A[i] \leq 1000$

$1 \leq K \leq 2000$

### Solution

01/25/2020:

```
class Solution {
public:
    int twoSumLessThanK(vector<int>& A, int K) {
        sort(A.begin(), A.end());
        int ret = -1;
        for (int l = 0, r = A.size() - 1; l < r; ) {
            int s = A[l] + A[r];
```

```

    if (s < K) {
        ret = max(ret, s);
        ++l;
    } else {
        --r;
    }
}
return ret;
};

```

## 1122. Relative Sort Array

### Description

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that don't appear in arr2 should be placed at the end of arr1 in ascending order.

Example 1:

Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

Output: [2,2,2,1,4,3,3,9,6,7,19]

Constraints:

arr1.length, arr2.length <= 1000

0 <= arr1[i], arr2[i] <= 1000

Each arr2[i] is distinct.

Each arr2[i] is in arr1.

### Solution

01/31/2020:

```

class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        vector<int> ret;
    }
};

```

```

map<int, int> m;
for (auto& n : arr1) {
    ++m[n];
}
for (auto& n : arr2) {
    while (m[n]-- > 0) {
        ret.push_back(n);
    }
}
for (auto it = m.begin(); it != m.end(); ++it) {
    while (it->second-- > 0) {
        ret.push_back(it->first);
    }
}
return ret;
}
};

```

## 1133. Largest Unique Number

### Description

Given an array of integers A, return the largest integer that only occurs once.

If no integer occurs once, return -1.

Example 1:

Input: [5,7,3,9,4,9,8,3,1]

Output: 8

Explanation:

The maximum integer in the array is 9 but it is repeated. The number 8 occurs only once, so it's the answer.

Example 2:

Input: [9,9,8,8]

Output: -1

Explanation:

There is no number that occurs only once.

Note:

$1 \leq A.length \leq 2000$

```
0 <= A[i] <= 1000
```

*Solution*

01/31/2020:

```
class Solution {
public:
    int largestUniqueNumber(vector<int>& A) {
        if (A.size() == 0) return -1;
        if (A.size() == 1) return A[0];
        sort(A.begin(), A.end());
        for (int i = A.size() - 1; i >= 0; --i) {
            if (i == A.size() - 1 && A[i] != A[i - 1]) {
                return A[i];
            } else if (i == 0 && A[i] != A[i + 1]) {
                return A[i];
            } else if (i < A.size() - 1 && i > 0 && A[i] != A[i - 1] && A[i] != A[i + 1]) {
                return A[i];
            }
        }
        return -1;
    }
};
```

## 1196. How Many Apples Can You Put into the Basket

*Description*

You have some apples, where `arr[i]` is the weight of the `i`-th apple. You also have a basket that can carry up to 5000 units of weight.

Return the maximum number of apples you can put in the basket.

Example 1:

Input: `arr = [100,200,150,1000]`

Output: 4

Explanation: All 4 apples can be carried by the basket since their sum of weights is 1450.

Example 2:

Input: arr = [900,950,800,1000,700,800]

Output: 5

Explanation: The sum of weights of the 6 apples exceeds 5000 so we choose any 5 of them.

Constraints:

$1 \leq \text{arr.length} \leq 10^3$

$1 \leq \text{arr}[i] \leq 10^3$

*Solution*

01/30/2020:

```
class Solution {
public:
    int maxNumberOfApples(vector<int>& arr) {
        const int N = 1001;
        vector<int> weights(N, 0);
        int s = 0, ret = 0;
        for (auto& a : arr) {
            ++weights[a];
        }
        for (int i = 0; i < N; ++i) {
            while (weights[i] > 0 && s + i <= 5000) {
                s += i;
                --weights[i];
                ++ret;
            }
        }
        return ret;
    }
};
```

## 1200. Minimum Absolute Difference

*Description*

Given an array of distinct integers arr, find all pairs of elements with the minimum absolute difference of any two elements.

Return a list of pairs in ascending order(with respect to pairs), each pair [a, b] follows

a, b are from arr  
a < b  
b - a equals to the minimum absolute difference of any two elements in arr

Example 1:

Input: arr = [4,2,1,3]

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

Explanation: The minimum absolute difference is 1. List all pairs with difference equal to 1 in ascending order.

Example 2:

Input: arr = [1,3,6,10,15]

Output: [[1,3]]

Example 3:

Input: arr = [3,8,-10,23,19,-4,-14,27]

Output: [[-14,-10],[19,23],[23,27]]

Constraints:

$2 \leq \text{arr.length} \leq 10^5$

$-10^6 \leq \text{arr}[i] \leq 10^6$

*Solution*

02/03/2020:

```
class Solution {
public:
    vector<vector<int>> minimumAbsDifference(vector<int>& arr) {
        sort(arr.begin(), arr.end());
        vector<vector<int>> ret;
        int diff = INT_MAX;
        for (int i = 1; i < (int)arr.size(); ++i) {
            if (diff == arr[i] - arr[i - 1]) {
                ret.push_back({arr[i - 1], arr[i]});
            } else if (diff > arr[i] - arr[i - 1]) {
                ret.clear();
                ret.push_back({arr[i - 1], arr[i]});
                diff = arr[i] - arr[i - 1];
            }
        }
        return ret;
    }
};
```

# 1207. Unique Number of Occurrences

## Description

Given an array of integers `arr`, write a function that returns true if and only if the number of occurrences of each value in the array is unique.

Example 1:

Input: `arr = [1,2,2,1,1,3]`

Output: true

Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

Example 2:

Input: `arr = [1,2]`

Output: false

Example 3:

Input: `arr = [-3,0,1,-3,1,1,1,-3,10,0]`

Output: true

Constraints:

`1 <= arr.length <= 1000`

`-1000 <= arr[i] <= 1000`

## Solution

01/30/2020:

```
class Solution {
public:
    bool uniqueOccurrences(vector<int>& arr) {
        unordered_map<int, int> m;
        unordered_set<int> occurrence;
        for (auto& n : arr) {
            ++m[n];
        }
        for (auto& n : m) {
            occurrence.insert(n.second);
        }
        return occurrence.size() == m.size();
    }
}
```



```
};
```

## 1213. Intersection of Three Sorted Arrays

### Description

Given three integer arrays `arr1`, `arr2` and `arr3` sorted in strictly increasing order, return a sorted array of only the integers that appeared in all three arrays.

Example 1:

Input: `arr1 = [1,2,3,4,5]`, `arr2 = [1,2,5,7,9]`, `arr3 = [1,3,4,5,8]`

Output: `[1,5]`

Explanation: Only 1 and 5 appeared in the three arrays.

Constraints:

`1 <= arr1.length, arr2.length, arr3.length <= 1000`

`1 <= arr1[i], arr2[i], arr3[i] <= 2000`

### Solution

01/29/2020:

```
class Solution {
public:
    vector<int> arraysIntersection(vector<int>& arr1, vector<int>& arr2,
vector<int>& arr3) {
        const int N = 2001;
        vector<int> m(N, 0);
        vector<int> ret;
        for (auto& a : arr1) ++m[a];
        for (auto& a : arr2) ++m[a];
        for (auto& a : arr3) ++m[a];
        for (int i = 0; i < N; ++i) {
            if (m[i] == 3) {
                ret.push_back(i);
            }
        }
        return ret;
    }
};
```

```
};
```

## 1217. Play with Chips

### *Description*

There are some chips, and the  $i$ -th chip is at position `chips[i]`.

You can perform any of the two following types of moves any number of times (possibly zero) on any chip:

Move the  $i$ -th chip by 2 units to the left or to the right with a cost of 0.

Move the  $i$ -th chip by 1 unit to the left or to the right with a cost of 1.

There can be two or more chips at the same position initially.

Return the minimum cost needed to move all the chips to the same position (any position).

Example 1:

Input: `chips = [1,2,3]`

Output: 1

Explanation: Second chip will be moved to position 3 with cost 1. First chip will be moved to position 3 with cost 0. Total cost is 1.

Example 2:

Input: `chips = [2,2,2,3,3]`

Output: 2

Explanation: Both fourth and fifth chip will be moved to position two with cost 1. Total minimum cost will be 2.

Constraints:

$1 \leq \text{chips.length} \leq 100$

$1 \leq \text{chips}[i] \leq 10^9$

### *Solution*

02/03/2020:

```
class Solution {  
public:
```

```

int minCostToMoveChips(vector<int>& chips) {
    int odd = 0, even = 0;
    for (auto& c : chips) {
        if (c % 2 == 1) {
            ++odd;
        } else {
            ++even;
        }
    }
    return min(odd, even);
}
};

```

## 1243. Array Transformation

### *Description*

Given an initial array `arr`, every day you produce a new array using the array of the previous day.

On the  $i$ -th day, you do the following operations on the array of day  $i-1$  to produce the array of day  $i$ :

If an element is smaller than both its left neighbor and its right neighbor, then this element is incremented.

If an element is bigger than both its left neighbor and its right neighbor, then this element is decremented.

The first and last elements never change.

After some days, the array does not change. Return that final array.

Example 1:

Input: `arr = [6,2,3,4]`

Output: `[6,3,3,4]`

Explanation:

On the first day, the array is changed from `[6,2,3,4]` to `[6,3,3,4]`.

No more operations can be done to this array.

Example 2:

Input: `arr = [1,6,3,4,3,5]`

Output: `[1,4,4,4,4,5]`

Explanation:

On the first day, the array is changed from `[1,6,3,4,3,5]` to `[1,5,4,3,4,5]`.

On the second day, the array is changed from `[1,5,4,3,4,5]` to `[1,4,4,4,4,5]`.

No more operations can be done to this array.

Constraints:

$1 \leq \text{arr.length} \leq 100$

$1 \leq \text{arr}[i] \leq 100$

*Solution*

05/20/2020:

```
class Solution {
public:
    vector<int> transformArray(vector<int>& arr) {
        vector<int> newarr(arr);
        while (true) {
            for (int i = 1; i < (int)arr.size() - 1; ++i) {
                if (arr[i - 1] < newarr[i] && newarr[i] > arr[i + 1]) {
                    --newarr[i];
                } else if (arr[i - 1] > newarr[i] && newarr[i] < arr[i + 1]) {
                    ++newarr[i];
                }
            }
            if (arr == newarr) break;
            arr = newarr;
        }
        return newarr;
    }
};
```

## 1252. Cells with Odd Values in a Matrix

*Description*

Given  $n$  and  $m$  which are the dimensions of a matrix initialized by zeros and given an array indices where  $\text{indices}[i] = [\text{ri}, \text{ci}]$ . For each pair of  $[\text{ri}, \text{ci}]$  you have to increment all cells in row  $\text{ri}$  and column  $\text{ci}$  by 1.

Return the number of cells with odd values in the matrix after applying the increment to all indices.

Example 1:

Input:  $n = 2$ ,  $m = 3$ ,  $indices = [[0,1],[1,1]]$

Output: 6

Explanation: Initial matrix =  $[[0,0,0],[0,0,0]]$ .

After applying first increment it becomes  $[[1,2,1],[0,1,0]]$ .

The final matrix will be  $[[1,3,1],[1,3,1]]$  which contains 6 odd numbers.

Example 2:

Input:  $n = 2$ ,  $m = 2$ ,  $indices = [[1,1],[0,0]]$

Output: 0

Explanation: Final matrix =  $[[2,2],[2,2]]$ . There is no odd number in the final matrix.

Constraints:

$1 \leq n \leq 50$

$1 \leq m \leq 50$

$1 \leq indices.length \leq 100$

$0 \leq indices[i][0] < n$

$0 \leq indices[i][1] < m$

*Solution*

01/29/2020:

```
class Solution {
public:
    int oddCells(int n, int m, vector<vector<int>>& indices) {
        vector<bool> row(n, false), col(m, false);
        int nr = 0, nc = 0;
        for (auto& index : indices) {
            row[index[0]] = !row[index[0]];
            col[index[1]] = !col[index[1]];
            nr += row[index[0]] ? 1 : -1;
            nc += col[index[1]] ? 1 : -1;
        }
        return m * nr + n * nc - 2 * nr * nc;
    }
};
```

1266. Minimum Time Visiting All Points

## Description

On a plane there are  $n$  points with integer coordinates  $\text{points}[i] = [x_i, y_i]$ . Your task is to find the minimum time in seconds to visit all points.

You can move according to the next rules:

In one second always you can either move vertically, horizontally by one unit or diagonally (it means to move one unit vertically and one unit horizontally in one second).

You have to visit the points in the same order as they appear in the array.

Example 1:

Input:  $\text{points} = [[1,1],[3,4],[-1,0]]$

Output: 7

Explanation: One optimal path is  $[1,1] \rightarrow [2,2] \rightarrow [3,3] \rightarrow [3,4] \rightarrow [2,3] \rightarrow [1,2] \rightarrow [0,1] \rightarrow [-1,0]$

Time from  $[1,1]$  to  $[3,4] = 3$  seconds

Time from  $[3,4]$  to  $[-1,0] = 4$  seconds

Total time = 7 seconds

Example 2:

Input:  $\text{points} = [[3,2],[-2,2]]$

Output: 5

Constraints:

$\text{points.length} == n$

$1 \leq n \leq 100$

$\text{points}[i].\text{length} == 2$

$-1000 \leq \text{points}[i][0], \text{points}[i][1] \leq 1000$

## Solution

01/29/2020:

```

class Solution {
public:
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {
        int ret = 0;
        for (int i = 1; i < (int)points.size(); ++i) {
            ret += max(abs(points[i][0] - points[i - 1][0]), abs(points[i][1] -
points[i - 1][1]));
        }
        return ret;
    }
};

```

## 1299. Replace Elements with Greatest Element on Right Side

### *Description*

Given an array `arr`, replace every element in that array with the greatest element among the elements to its right, and replace the last element with `-1`.

After doing so, return the array.

Example 1:

Input: `arr = [17,18,5,4,6,1]`

Output: `[18,6,6,6,1,-1]`

Constraints:

$1 \leq \text{arr.length} \leq 10^4$

$1 \leq \text{arr}[i] \leq 10^5$

### *Solution*

01/29/2020:

```

class Solution {
public:
    vector<int> replaceElements(vector<int>& arr) {
        int cur_max = INT_MIN;
        for (int i = arr.size() - 1; i >= 0; --i) {
            int tmp = arr[i];
            arr[i] = cur_max;
            cur_max = max(cur_max, tmp);
        }
        arr.back() = -1;
        return arr;
    }
};

```

## 1304. Find N Unique Integers Sum up to Zero

### Description

Given an integer  $n$ , return any array containing  $n$  unique integers such that they add up to 0.

Example 1:

Input:  $n = 5$

Output:  $[-7, -1, 1, 3, 4]$

Explanation: These arrays also are accepted  $[-5, -1, 1, 2, 3]$  ,  $[-3, -1, 2, -2, 4]$ .

Example 2:

Input:  $n = 3$

Output:  $[-1, 0, 1]$

Example 3:

Input:  $n = 1$

Output:  $[0]$

Constraints:

$1 \leq n \leq 1000$

### Solution

01/29/2020:



```

class Solution {
public:
    vector<int> sumZero(int n) {
        vector<int> ret(n, 0);
        for (int i = 0; i < n - 1; i += 2) {
            ret[i] = (i + 1);
            ret[i + 1] = -(i + 1);
        }
        return ret;
    }
};

```

## 1305. All Elements in Two Binary Search Trees

### Description

Given two binary search trees root1 and root2.

Return a list containing all the integers from both trees sorted in ascending order.

Example 1:

Input: root1 = [2,1,4], root2 = [1,0,3]

Output: [0,1,1,2,3,4]

Example 2:

Input: root1 = [0,-10,10], root2 = [5,1,7,0,2]

Output: [-10,0,0,1,2,5,7,10]

Example 3:

Input: root1 = [], root2 = [5,1,7,0,2]

Output: [0,1,2,5,7]

Example 4:

Input: root1 = [0,-10,10], root2 = []

Output: [-10,0,10]

Example 5:

Input: root1 = [1,null,8], root2 = [8,1]

Output: [1,1,8,8]

Constraints:

Each tree has at most 5000 nodes.  
Each node's value is between  $[-10^5, 10^5]$ .

*Solution*

[Discussion:](#)

Using inorder traversal to obtain two sorted subarrays from the BST's, then merge the subarrays into one sorted array like that in `merge_sort`.

```
class Solution {
public:
    vector<int> getAllElements(TreeNode* root1, TreeNode* root2) {
        vector<int> v1, v2, ret;
        inorder(root1, v1);
        inorder(root2, v2);
        for (auto it1 = v1.begin(), it2 = v2.begin(); it1 != v1.end() || it2 !=
v2.end();)
            ret.push_back(it2 == v2.end() || (it1 != v1.end() && *it1 < *it2) ? *it1++
: *it2++);
        return ret;
    }

    void inorder(TreeNode* root, vector<int>& nums) {
        if (root == nullptr) return;
        inorder(root->left, nums);
        nums.push_back(root->val);
        inorder(root->right, nums);
    }
};
```

## 1313. Decompress Run-Length Encoded List

*Description*

We are given a list `nums` of integers representing a list compressed with run-length encoding.

Consider each adjacent pair of elements  $[a, b] = [\text{nums}[2*i], \text{nums}[2*i+1]]$  (with  $i \geq 0$ ). For each such pair, there are  $a$  elements with value  $b$  in the decompressed list.

Return the decompressed list.

Example 1:

Input: nums = [1,2,3,4]

Output: [2,4,4,4]

Explanation: The first pair [1,2] means we have freq = 1 and val = 2 so we generate the array [2].

The second pair [3,4] means we have freq = 3 and val = 4 so we generate [4,4,4].  
At the end the concatenation [2] + [4,4,4,4] is [2,4,4,4].

Constraints:

$2 \leq \text{nums.length} \leq 100$

$\text{nums.length} \% 2 == 0$

$1 \leq \text{nums}[i] \leq 100$

*Solution*

01/29/2020:

```
class Solution {
public:
    vector<int> decompressRLElist(vector<int>& nums) {
        vector<int> ret;
        for (int i = 0; i < nums.size(); i += 2) {
            for (int j = 0; j < nums[i]; ++j) {
                ret.push_back(nums[i + 1]);
            }
        }
        return ret;
    }
};
```

## 1337. The K Weakest Rows in a Matrix

*Description*

Given a  $m \times n$  matrix mat of ones (representing soldiers) and zeros (representing civilians), return the indexes of the k weakest rows in the matrix ordered from the weakest to the strongest.

A row  $i$  is weaker than row  $j$ , if the number of soldiers in row  $i$  is less than the number of soldiers in row  $j$ , or they have the same number of soldiers but  $i$  is less than  $j$ . Soldiers are always stand in the frontier of a row, that is, always ones may appear first and then zeros.

Example 1:

```
Input: mat =  
[[1,1,0,0,0],  
 [1,1,1,1,0],  
 [1,0,0,0,0],  
 [1,1,0,0,0],  
 [1,1,1,1,1]],  
k = 3
```

```
Output: [2,0,3]
```

Explanation:

The number of soldiers for each row is:

row 0 -> 2

row 1 -> 4

row 2 -> 1

row 3 -> 2

row 4 -> 5

Rows ordered from the weakest to the strongest are [2,0,3,1,4]

Example 2:

```
Input: mat =  
[[1,0,0,0],  
 [1,1,1,1],  
 [1,0,0,0],  
 [1,0,0,0]],  
k = 2
```

```
Output: [0,2]
```

Explanation:

The number of soldiers for each row is:

row 0 -> 1

row 1 -> 4

row 2 -> 1

row 3 -> 1

Rows ordered from the weakest to the strongest are [0,2,3,1]

Constraints:

```
m == mat.length  
n == mat[i].length  
2 <= n, m <= 100
```

```
1 <= k <= m
matrix[i][j] is either 0 or 1.
```

*Solution*

02/02/2020:

```
class Solution {
public:
    vector<int> kWeakestRows(vector<vector<int>>& mat, int k) {
        vector<pair<int, int>> p;
        for (int i = 0; i < (int)mat.size(); ++i) {
            int cnt = 0;
            for (int j = 0; j < (int)mat[0].size(); ++j) {
                ++cnt;
                if (mat[i][j] == 0) {
                    --cnt;
                    break;
                }
            }
            p.push_back({i, cnt});
        }
        sort(p.begin(), p.end(), [](pair<int, int> p1, pair<int, int> p2){
            if (p1.second == p2.second)
                return p1.first < p2.first;
            return p1.second < p2.second;
        });
        vector<int> ret;
        for (int i = 0; i < k; ++i) {
            ret.push_back(p[i].first);
        }
        return ret;
    }
};
```

## 1338. Reduce Array Size to The Half

*Description*

Given an array arr. You can choose a set of integers and remove all the occurrences of these integers in the array.

Return the minimum size of the set so that at least half of the integers of the array are removed.

Example 1:

Input: arr = [3,3,3,3,5,5,5,2,2,7]

Output: 2

Explanation: Choosing {3,7} will make the new array [5,5,5,2,2] which has size 5 (i.e equal to half of the size of the old array).

Possible sets of size 2 are {3,5},{3,2},{5,2}.

Choosing set {2,7} is not possible as it will make the new array [3,3,3,3,5,5,5] which has size greater than half of the size of the old array.

Example 2:

Input: arr = [7,7,7,7,7,7]

Output: 1

Explanation: The only possible set you can choose is {7}. This will make the new array empty.

Example 3:

Input: arr = [1,9]

Output: 1

Example 4:

Input: arr = [1000,1000,3,7]

Output: 1

Example 5:

Input: arr = [1,2,3,4,5,6,7,8,9,10]

Output: 5

Constraints:

$1 \leq \text{arr.length} \leq 10^5$

arr.length is even.

$1 \leq \text{arr}[i] \leq 10^5$

*Solution*

02/02/2020:

```
class Solution {
public:
    int minSetSize(vector<int>& arr) {
        unordered_map<int, int> m;
        for (auto& n : arr) ++m[n];
        vector<int> p;
        for (auto it = m.begin(); it != m.end(); ++it) {
```

```

        p.push_back(it->second);
    }
    sort(p.begin(), p.end(), [](int a, int b) {return a > b;});
    int ret = 0, s = 0, as = arr.size() / 2;
    for (auto& n : p) {
        s += n;
        ++ret;
        if (s >= as)
            break;
    }
    return ret;
}
};

```

## 1413. Minimum Value to Get Positive Step by Step Sum

### Description

Given an array of integers `nums`, you start with an initial positive value `startValue`.

In each iteration, you calculate the step by step sum of `startValue` plus elements in `nums` (from left to right).

Return the minimum positive value of `startValue` such that the step by step sum is never less than 1.

Example 1:

Input: `nums = [-3,2,-3,4,2]`

Output: 5

Explanation: If you choose `startValue = 4`, in the third iteration your step by step sum is less than 1.

step by step sum		
<code>startValue = 4</code>	<code>startValue = 5</code>	<code>nums</code>
<code>(4 -3 ) = 1</code>	<code>(5 -3 ) = 2</code>	<code>-3</code>
<code>(1 +2 ) = 3</code>	<code>(2 +2 ) = 4</code>	<code>2</code>
<code>(3 -3 ) = 0</code>	<code>(4 -3 ) = 1</code>	<code>-3</code>
<code>(0 +4 ) = 4</code>	<code>(1 +4 ) = 5</code>	<code>4</code>
<code>(4 +2 ) = 6</code>	<code>(5 +2 ) = 7</code>	<code>2</code>

Example 2:

Input: `nums = [1,2]`

Output: 1

Explanation: Minimum start value should be positive.

Example 3:

Input: nums = [1,-2,-3]

Output: 5

Constraints:

$1 \leq \text{nums.length} \leq 100$

$-100 \leq \text{nums}[i] \leq 100$

*Solution*

06/11/2020:

```
class Solution {
public:
    int minStartValue(vector<int>& nums) {
        partial_sum(nums.begin(), nums.end(), nums.begin());
        int ret = *min_element(nums.begin(), nums.end());
        return ret >= 0 ? 1 : 1 - ret;
    }
};
```

## 1423. Maximum Points You Can Obtain from Cards

*Description*

There are several cards arranged in a row, and each card has an associated number of points. The points are given in the integer array `cardPoints`.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly `k` cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array `cardPoints` and the integer `k`, return the maximum score you can obtain.

Example 1:

Input: `cardPoints = [1,2,3,4,5,6,1]`, `k = 3`



Output: 12

Explanation: After the first step, your score will always be 1. However, choosing the rightmost card first will maximize your total score. The optimal strategy is to take the three cards on the right, giving a final score of  $1 + 6 + 5 = 12$ .

Example 2:

Input: cardPoints = [2,2,2], k = 2

Output: 4

Explanation: Regardless of which two cards you take, your score will always be 4.

Example 3:

Input: cardPoints = [9,7,7,9,7,7,9], k = 7

Output: 55

Explanation: You have to take all the cards. Your score is the sum of points of all cards.

Example 4:

Input: cardPoints = [1,1000,1], k = 1

Output: 1

Explanation: You cannot take the card in the middle. Your best score is 1.

Example 5:

Input: cardPoints = [1,79,80,1,1,1,200,1], k = 3

Output: 202

Constraints:

$1 \leq \text{cardPoints.length} \leq 10^5$

$1 \leq \text{cardPoints}[i] \leq 10^4$

$1 \leq k \leq \text{cardPoints.length}$

*Solution*

04/25/2020:

```
class Solution {
public:
    int maxScore(vector<int>& cardPoints, int k) {
        int n = cardPoints.size();
        vector<int> sub(2 * k, 0);
        sub[0] = cardPoints[n - k];
        for (int i = -k + 1; i < k; ++i) {
            sub[i + k] = sub[i + k - 1] + cardPoints[(n + i) % n];
        }
        int ret = sub[k - 1];
    }
};
```

```
    for (int i = k; i < 2 * k; ++i) {
        ret = max(sub[i] - sub[i - k], ret);
    }
    return ret;
}
};
```

## 1425. Constrained Subset Sum

### Description

Given an integer array `nums` and an integer `k`, return the maximum sum of a non-empty subset of that array such that for every two consecutive integers in the subset, `nums[i]` and `nums[j]`, where  $i < j$ , the condition  $j - i \leq k$  is satisfied.

A subset of an array is obtained by deleting some number of elements (can be zero) from the array, leaving the remaining elements in their original order.

Example 1:

Input: `nums = [10,2,-10,5,20]`, `k = 2`

Output: 37

Explanation: The subset is `[10, 2, 5, 20]`.

Example 2:

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

Output: -1

Explanation: The subset must be non-empty, so we choose the largest number.

Example 3:

Input: `nums = [10,-2,-10,-5,20]`, `k = 2`

Output: 23

Explanation: The subset is `[10, -2, -5, 20]`.

Constraints:

$1 \leq k \leq \text{nums.length} \leq 10^5$

$-10^4 \leq \text{nums}[i] \leq 10^4$

### Solution

04/25/2020:

---

```

class Solution {
public:
    int constrainedSubsetSum(vector<int>& nums, int k) {
        int n = nums.size();
        int ret = nums[0];
        priority_queue<pair<int, int>> pq; // store the pair (sum, i), the sum at i
        for (int i = 0; i < n; ++i) {
            int tmp = 0;
            while (!pq.empty() && i - pq.top().second > k) {
                pq.pop();
            }
            if (!pq.empty()) {
                tmp = max(tmp, pq.top().first);
            }
            ret = max(ret, tmp + nums[i]);
            pq.emplace(tmp + nums[i], i);
        }
        return ret;
    }
};

```

## 1428. Leftmost Column with at Least a One

### Description

(This problem is an interactive problem.)

A binary matrix means that all elements are 0 or 1. For each individual row of the matrix, this row is sorted in non-decreasing order.

Given a row-sorted binary matrix `binaryMatrix`, return leftmost column index (0-indexed) with at least a 1 in it. If such index doesn't exist, return -1.

You can't access the Binary Matrix directly. You may only access the matrix using a `BinaryMatrix` interface:

`BinaryMatrix.get(row, col)` returns the element of the matrix at index `(row, col)` (0-indexed).

`BinaryMatrix.dimensions()` returns a list of 2 elements `[rows, cols]`, which means the matrix is `rows * cols`.

Submissions making more than 1000 calls to `BinaryMatrix.get` will be judged Wrong Answer. Also, any solutions that attempt to circumvent the judge will result in disqualification.

For custom testing purposes you're given the binary matrix `mat` as input in the following four examples. You will not have access the binary matrix directly.

Example 1:

Input: `mat = [[0,0],[1,1]]`

Output: `0`

Example 2:

Input: `mat = [[0,0],[0,1]]`

Output: `1`

Example 3:

Input: `mat = [[0,0],[0,0]]`

Output: `-1`

Example 4:

Input: `mat = [[0,0,0,1],[0,0,1,1],[0,1,1,1]]`

Output: `1`

Constraints:

`rows == mat.length`

`cols == mat[i].length`

`1 <= rows, cols <= 100`

`mat[i][j]` is either `0` or `1`.

`mat[i]` is sorted in a non-decreasing way.

*Solution*

05/04/2020:

`/**`

```

* // This is the BinaryMatrix's API interface.
* // You should not implement it, or speculate about its implementation
* class BinaryMatrix {
*   public:
*     int get(int x, int y);
*     vector<int> dimensions();
* };
*/

class Solution {
public:
    int leftMostColumnWithOne(BinaryMatrix &binaryMatrix) {
        vector<int> sz = binaryMatrix.dimensions();
        int m = sz[0], n = sz[1], ret = INT_MAX;
        int lo = 0, hi = n - 1, mid;
        for (int i = 0; i < m; ++i) {
            lo = 0;
            int lo_val = binaryMatrix.get(i, lo);
            if (lo_val == 1) {
                ret = min(ret, lo);
                hi = lo;
                continue;
            }
            int hi_val = binaryMatrix.get(i, hi);
            if (hi_val == 0) continue;
            while (lo < hi) {
                mid = (lo + hi) / 2;
                int mid_val = binaryMatrix.get(i, mid);
                if (mid_val == 1) {
                    hi = mid;
                } else {
                    lo = mid + 1;
                }
            }
            ret = min(ret, hi);
        }
        return ret == INT_MAX ? -1 : ret;
    }
};

```

## 1431. Kids With the Greatest Number of Candies

### Description

Given the array `candies` and the integer `extraCandies`, where `candies[i]` represents the number of candies that the `i`th kid has.

For each kid check if there is a way to distribute extraCandies among the kids such that he or she can have the greatest number of candies among them. Notice that multiple kids can have the greatest number of candies.

Example 1:

Input: candies = [2,3,5,1,3], extraCandies = 3

Output: [true,true,true,false,true]

Explanation:

Kid 1 has 2 candies and if he or she receives all extra candies (3) will have 5 candies --- the greatest number of candies among the kids.

Kid 2 has 3 candies and if he or she receives at least 2 extra candies will have the greatest number of candies among the kids.

Kid 3 has 5 candies and this is already the greatest number of candies among the kids.

Kid 4 has 1 candy and even if he or she receives all extra candies will only have 4 candies.

Kid 5 has 3 candies and if he or she receives at least 2 extra candies will have the greatest number of candies among the kids.

Example 2:

Input: candies = [4,2,1,1,2], extraCandies = 1

Output: [true,false,false,false,false]

Explanation: There is only 1 extra candy, therefore only kid 1 will have the greatest number of candies among the kids regardless of who takes the extra candy.

Example 3:

Input: candies = [12,1,12], extraCandies = 10

Output: [true,false,true]

Constraints:

2 <= candies.length <= 100

1 <= candies[i] <= 100

1 <= extraCandies <= 50

*Solution*

05/02/2020:

```

class Solution {
public:
    vector<bool> kidsWithCandies(vector<int>& candies, int extraCandies) {
        int max_val = *max_element(candies.begin(), candies.end());
        int n = candies.size();
        vector<bool> ret(n, false);
        for(int i = 0; i < n; ++i) {
            ret[i] = candies[i] + extraCandies >= max_val;
        }
        return ret;
    }
};

```

## 1437. Check If All 1's Are at Least Length K Places Away

### Description

Given an array `nums` of 0s and 1s and an integer `k`, return `True` if all 1's are at least `k` places away from each other, otherwise return `False`.

Example 1:

Input: `nums = [1,0,0,0,1,0,0,1]`, `k = 2`

Output: `true`

Explanation: Each of the 1s are at least 2 places away from each other.

Example 2:

Input: `nums = [1,0,0,1,0,1]`, `k = 2`

Output: `false`

Explanation: The second 1 and third 1 are only one apart from each other.

Example 3:

Input: `nums = [1,1,1,1,1]`, `k = 0`

Output: `true`

Example 4:

Input: `nums = [0,1,0,1]`, `k = 1`

Output: `true`

Constraints:

```
1 <= nums.length <= 10^5
0 <= k <= nums.length
nums[i] is 0 or 1
```

*Solution*

04/28/2020:

```
class Solution {
public:
    bool kLengthApart(vector<int>& nums, int k) {
        int min_dist = 1e6;
        int lastOne = -1e6;
        for (int i = 0; i < (int)nums.size(); ++i) {
            if (nums[i] == 1) {
                min_dist = min(i - lastOne - 1, min_dist);
                lastOne = i;
                if (min_dist < k) return false;
            }
        }
        return true;
    }
};
```

## 1442. Count Triplets That Can Form Two Arrays of Equal XOR

---

*Description*

Given an array of integers arr.

We want to select three indices i, j and k where (0 <= i < j <= k < arr.length).

Let's define a and b as follows:

$$a = arr[i] \oplus arr[i + 1] \oplus \dots \oplus arr[j - 1]$$
$$b = arr[j] \oplus arr[j + 1] \oplus \dots \oplus arr[k]$$

Note that  $\oplus$  denotes the bitwise-xor operation.

Return the number of triplets (i, j and k) Where  $a == b$ .



Example 1:

Input: arr = [2,3,1,6,7]

Output: 4

Explanation: The triplets are (0,1,2), (0,2,2), (2,3,4) and (2,4,4)

Example 2:

Input: arr = [1,1,1,1,1]

Output: 10

Example 3:

Input: arr = [2,3]

Output: 0

Example 4:

Input: arr = [1,3,5,7,9]

Output: 3

Example 5:

Input: arr = [7,11,12,9,5,2,7,17,22]

Output: 8

Constraints:

1 <= arr.length <= 300

1 <= arr[i] <= 10<sup>8</sup>

*Solution*

05/10/2020:

```
class Solution {
public:
    int countTriplets(vector<int>& arr) {
        int n = arr.size();
        vector<int> nums(n);
        nums[0] = arr[0];
        for (int i = 1; i < n; ++i) nums[i] = nums[i - 1] ^ arr[i];
        // a = arr[i] ^ ... arr[j - 1] = nums[j - 1] ^ nums[i - 1]
        // b = arr[j] ^ ... arr[k] = nums[k] ^ nums[j - 1]
        int a, b, cnt = 0;
        for (int i = 0; i < n - 1; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (i > 0)
                    a = nums[j - 1] ^ nums[i - 1];
```

```

else
    a = nums[j - 1];
for (int k = j; k < n; ++k) {
    int b = nums[k] ^ nums[j - 1];
    if (a == b) {
        ++cnt;
    }
}
}
}
return cnt;
}
};

```

## 1450. Number of Students Doing Homework at a Given Time

### Description

Given two integer arrays `startTime` and `endTime` and given an integer `queryTime`.

The  $i$ th student started doing their homework at the time `startTime[i]` and finished it at time `endTime[i]`.

Return the number of students doing their homework at time `queryTime`. More formally, return the number of students where `queryTime` lays in the interval `[startTime[i], endTime[i]]` inclusive.

Example 1:

Input: `startTime = [1,2,3]`, `endTime = [3,2,7]`, `queryTime = 4`

Output: 1

Explanation: We have 3 students where:

The first student started doing homework at time 1 and finished at time 3 and wasn't doing anything at time 4.

The second student started doing homework at time 2 and finished at time 2 and also wasn't doing anything at time 4.

The third student started doing homework at time 3 and finished at time 7 and was the only student doing homework at time 4.

Example 2:

Input: `startTime = [4]`, `endTime = [4]`, `queryTime = 4`

Output: 1

Explanation: The only student was doing their homework at the queryTime.

Example 3:

Input: startTime = [4], endTime = [4], queryTime = 5

Output: 0

Example 4:

Input: startTime = [1,1,1,1], endTime = [1,3,2,4], queryTime = 7

Output: 0

Example 5:

Input: startTime = [9,8,7,6,5,4,3,2,1], endTime = [10,10,10,10,10,10,10,10,10], queryTime = 5

Output: 5

Constraints:

```
startTime.length == endTime.length
1 <= startTime.length <= 100
1 <= startTime[i] <= endTime[i] <= 1000
1 <= queryTime <= 1000
```

*Solution*

05/17/2020:

```
class Solution {
public:
    int busyStudent(vector<int>& startTime, vector<int>& endTime, int queryTime) {
        int cnt = 0, n = startTime.size();
        for (int i = 0; i < n; ++i) {
            if (queryTime >= startTime[i] && queryTime <= endTime[i]) {
                ++cnt;
            }
        }
        return cnt;
    }
};
```

## 1460. Make Two Arrays Equal by Reversing Sub-arrays

*Description*

Given two integer arrays of equal length target and arr.

In one step, you can select any non-empty sub-array of arr and reverse it. You are allowed to make any number of steps.

Return True if you can make arr equal to target, or False otherwise.

Example 1:

Input: target = [1,2,3,4], arr = [2,4,1,3]

Output: true

Explanation: You can follow the next steps to convert arr to target:

1- Reverse sub-array [2,4,1], arr becomes [1,4,2,3]

2- Reverse sub-array [4,2], arr becomes [1,2,4,3]

3- Reverse sub-array [4,3], arr becomes [1,2,3,4]

There are multiple ways to convert arr to target, this is not the only way to do so.

Example 2:

Input: target = [7], arr = [7]

Output: true

Explanation: arr is equal to target without any reverses.

Example 3:

Input: target = [1,12], arr = [12,1]

Output: true

Example 4:

Input: target = [3,7,9], arr = [3,7,11]

Output: false

Explanation: arr doesn't have value 9 and it can never be converted to target.

Example 5:

Input: target = [1,1,1,1,1], arr = [1,1,1,1,1]

Output: true

Constraints:

target.length == arr.length

1 <= target.length <= 1000

1 <= target[i] <= 1000

1 <= arr[i] <= 1000

*Solution*

05/30/2020:

```
class Solution {
public:
    bool canBeEqual(vector<int>& target, vector<int>& arr) {
        sort(target.begin(), target.end());
        sort(arr.begin(), arr.end());
        return target == arr;
    }
};
```

## 1464. Maximum Product of Two Elements in an Array

### Description

Given the array of integers `nums`, you will choose two different indices `i` and `j` of that array. Return the maximum value of  $(nums[i]-1)*(nums[j]-1)$ .

Example 1:

Input: `nums = [3,4,5,2]`

Output: 12

Explanation: If you choose the indices `i=1` and `j=2` (indexed from 0), you will get the maximum value, that is,  $(nums[1]-1)*(nums[2]-1) = (4-1)*(5-1) = 3*4 = 12$ .

Example 2:

Input: `nums = [1,5,4,5]`

Output: 16

Explanation: Choosing the indices `i=1` and `j=3` (indexed from 0), you will get the maximum value of  $(5-1)*(5-1) = 16$ .

Example 3:

Input: `nums = [3,7]`

Output: 12

Constraints:

$2 \leq \text{nums.length} \leq 500$

$1 \leq \text{nums}[i] \leq 10^3$

### Solution

05/30/2020:

```

class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(), nums.end());
        return (nums[n - 1] - 1) * (nums[n - 2] - 1);
    }
};

```

## 1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

### Description

Given a rectangular cake with height  $h$  and width  $w$ , and two arrays of integers `horizontalCuts` and `verticalCuts` where `horizontalCuts[i]` is the distance from the top of the rectangular cake to the  $i$ th horizontal cut and similarly, `verticalCuts[j]` is the distance from the left of the rectangular cake to the  $j$ th vertical cut.

Return the maximum area of a piece of cake after you cut at each horizontal and vertical position provided in the arrays `horizontalCuts` and `verticalCuts`. Since the answer can be a huge number, return this modulo  $10^9 + 7$ .

Example 1:

Input:  $h = 5$ ,  $w = 4$ , `horizontalCuts = [1,2,4]`, `verticalCuts = [1,3]`

Output: 4

Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green piece of cake has the maximum area.

Example 2:

Input:  $h = 5$ ,  $w = 4$ , `horizontalCuts = [3,1]`, `verticalCuts = [1]`

Output: 6

Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green and yellow pieces of cake have the maximum area.

Example 3:

Input:  $h = 5$ ,  $w = 4$ ,  $\text{horizontalCuts} = [3]$ ,  $\text{verticalCuts} = [3]$

Output: 9

Constraints:

$2 \leq h, w \leq 10^9$

$1 \leq \text{horizontalCuts.length} < \min(h, 10^5)$

$1 \leq \text{verticalCuts.length} < \min(w, 10^5)$

$1 \leq \text{horizontalCuts}[i] < h$

$1 \leq \text{verticalCuts}[i] < w$

It is guaranteed that all elements in  $\text{horizontalCuts}$  are distinct.

It is guaranteed that all elements in  $\text{verticalCuts}$  are distinct.

*Solution*

05/30/2020:

```
class Solution {
public:
    int maxArea(int h, int w, vector<int>& horizontalCuts, vector<int>&
verticalCuts) {
        horizontalCuts.push_back(0);
        horizontalCuts.push_back(h);
        verticalCuts.push_back(0);
        verticalCuts.push_back(w);
        sort(horizontalCuts.begin(), horizontalCuts.end());
        sort(verticalCuts.begin(), verticalCuts.end());
        long long maxH = 0, maxW = 0;
        for (int i = 1; i < (int)horizontalCuts.size(); ++i) {
            maxH = max(maxH, (long long)horizontalCuts[i] - horizontalCuts[i - 1]);
        }
        for (int i = 1; i < (int)verticalCuts.size(); ++i) {
            maxW = max(maxW, (long long)verticalCuts[i] - verticalCuts[i - 1]);
        }
        const int MOD = 1e9 + 7;
        return maxH * maxW % MOD;
    }
};
```

## 1470. Shuffle the Array

*Description*

---

Given the array `nums` consisting of  $2n$  elements in the form `[x1,x2,...,xn,y1,y2,...,yn]`.

Return the array in the form `[x1,y1,x2,y2,...,xn,yn]`.

Example 1:

Input: `nums = [2,5,1,3,4,7]`, `n = 3`

Output: `[2,3,5,4,1,7]`

Explanation: Since `x1=2`, `x2=5`, `x3=1`, `y1=3`, `y2=4`, `y3=7` then the answer is `[2,3,5,4,1,7]`.

Example 2:

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

Output: `[1,4,2,3,3,2,4,1]`

Example 3:

Input: `nums = [1,1,2,2]`, `n = 2`

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

Constraints:

`1 <= n <= 500`

`nums.length == 2n`

`1 <= nums[i] <= 103`

*Solution*

06/06/2020:

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



# 1475. Final Prices With a Special Discount in a Shop

## Description

Given the array `prices` where `prices[i]` is the price of the  $i$ th item in a shop. There is a special discount for items in the shop, if you buy the  $i$ th item, then you will receive a discount equivalent to `prices[j]` where  $j$  is the minimum index such that  $j > i$  and `prices[j] <= prices[i]`, otherwise, you will not receive any discount at all.

Return an array where the  $i$ th element is the final price you will pay for the  $i$ th item of the shop considering the special discount.

Example 1:

Input: `prices = [8,4,6,2,3]`

Output: `[4,2,4,2,3]`

Explanation:

For item 0 with `price[0]=8` you will receive a discount equivalent to `prices[1]=4`, therefore, the final price you will pay is  $8 - 4 = 4$ .

For item 1 with `price[1]=4` you will receive a discount equivalent to `prices[3]=2`, therefore, the final price you will pay is  $4 - 2 = 2$ .

For item 2 with `price[2]=6` you will receive a discount equivalent to `prices[3]=2`, therefore, the final price you will pay is  $6 - 2 = 4$ .

For items 3 and 4 you will not receive any discount at all.

Example 2:

Input: `prices = [1,2,3,4,5]`

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

Explanation: In this case, for all items, you will not receive any discount at all.

Example 3:

Input: `prices = [10,1,1,6]`

Output: `[9,0,1,6]`

Constraints:

$1 \leq \text{prices.length} \leq 500$

$1 \leq \text{prices}[i] \leq 10^3$

## Solution

06/13/2020:

---

```

class Solution {
public:
    vector<int> finalPrices(vector<int>& prices) {
        int n = prices.size();
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (prices[i] >= prices[j]) {
                    prices[i] -= prices[j];
                    break;
                }
            }
        }
        return prices;
    }
};

```

## 1481. Least Number of Unique Integers after K Removals

### Description

Given an array of integers `arr` and an integer `k`. Find the least number of unique integers after removing exactly `k` elements.

Example 1:

Input: `arr = [5,5,4]`, `k = 1`

Output: 1

Explanation: Remove the single 4, only 5 is left.

Example 2:

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

Output: 2

Explanation: Remove 4, 2 and either one of the two 1s or three 3s. 1 and 3 will be left.

Constraints:

$1 \leq arr.length \leq 10^5$

$1 \leq arr[i] \leq 10^9$

$0 \leq k \leq arr.length$

### Solution

06/13/2020:

```
class Solution {
public:
    int findLeastNumOfUniqueInts(vector<int>& arr, int k) {
        unordered_map<int, int> cnt;
        for (auto& a : arr) ++cnt[a];
        vector<pair<int, int>> p;
        for (auto& c : cnt) p.push_back({c.second, c.first});
        sort(p.rbegin(), p.rend());
        while (k > 0 && !p.empty()) {
            k -= p.back().first;
            if (k >= 0) {
                p.pop_back();
            }
        }
        return p.size();
    }
};
```

## 1491. Average Salary Excluding the Minimum and Maximum Salary

### Description

Given an array of unique integers salary where salary[i] is the salary of the employee i.

Return the average salary of employees excluding the minimum and maximum salary.

Example 1:

Input: salary = [4000,3000,1000,2000]

Output: 2500.00000

Explanation: Minimum salary and maximum salary are 1000 and 4000 respectively.

Average salary excluding minimum and maximum salary is  $(2000+3000)/2= 2500$

Example 2:

Input: salary = [1000,2000,3000]

Output: 2000.00000

Explanation: Minimum salary and maximum salary are 1000 and 3000 respectively.

Average salary excluding minimum and maximum salary is  $(2000)/1= 2000$

Example 3:

Input: salary = [6000,5000,4000,3000,2000,1000]

Output: 3500.00000

Example 4:

Input: salary = [8000,9000,2000,3000,6000,1000]

Output: 4750.00000

Constraints:

$3 \leq \text{salary.length} \leq 100$

$10^3 \leq \text{salary}[i] \leq 10^6$

salary[i] is unique.

Answers within  $10^{-5}$  of the actual value will be accepted as correct.

*Solution*

06/27/2020:

```
class Solution {
public:
    double average(vector<int>& salary) {
        double avg = 0;
        int n = salary.size();
        for (auto s : salary) avg += s;
        avg -= *max_element(salary.begin(), salary.end());
        avg -= *min_element(salary.begin(), salary.end());
        return avg / (n - 2);
    }
};
```

# Tree

## 104. Maximum Depth of Binary Tree

*Description*

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

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

```
    3
   / \
  9  20
   \  \
   15  7
return its depth = 3.
```

*Solution*

02/03/2020:

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

## 111. Minimum Depth of Binary Tree

*Description*

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

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

```
    3
   / \
  9  20
   / \
  15  7
return its minimum depth = 2.
```

*Solution*

[Discussion](#): Non-recursive DFS:

```
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (root == nullptr) return 0;
        stack<pair<TreeNode*, int>> st;
        st.emplace(root, 1);
        int min_depth = numeric_limits<int>::max();
        while (!st.empty()) {
            pair<TreeNode*, int> cur = st.top(); st.pop();
            if (cur.first->left == nullptr && cur.first->right == nullptr) min_depth =
min(min_depth, cur.second);
            if (cur.first->left != nullptr) st.emplace(cur.first->left, cur.second +
1);
            if (cur.first->right != nullptr) st.emplace(cur.first->right, cur.second +
1);
        }
        return min_depth;
    }
};
```

Recursive DFS:

```

class Solution {
public:
    int minDepth(TreeNode* root) {
        if (root == nullptr) return 0;
        int min_depth = numeric_limits<int>::max();
        if (root->left != nullptr) min_depth = min(min_depth, minDepth(root->left) +
1);
        if (root->right != nullptr) min_depth = min(min_depth, minDepth(root->right)
+ 1);
        return min_depth == numeric_limits<int>::max() ? 1 : min_depth;
    }
};

```

## 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

01/30/2020:

```

/**
 * 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> 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;
    }
};

```

## 199. Binary Tree Right Side View

### Description

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example:

Input: [1,2,3,null,5,null,4]

Output: [1, 3, 4]

Explanation:

```

      1           <---
     / \
    2   3       <---
     \   \
     5   4     <---

```

### Solution

05/06/2020:

```

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

```

## 513. Find Bottom Left Tree Value

### Description

Given a binary tree, find the leftmost value in the last row of the tree.

Example 1:

Input:

```

    2
   / \
  1   3

```

Output:

1

Example 2:

Input:

```

    1
   / \

```

```

    2   3
   /   / \
  4   5   6
     /
    7

```

Output:

7

Note: You may assume the tree (i.e., the given root node) is not NULL.

*Solution*

05/04/2020:

```

/**
 * 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 findBottomLeftValue(TreeNode* root) {
        if (root == nullptr) return -1;
        queue<TreeNode*> q;
        q.push(root);
        int leftmost = root->val;
        while (!q.empty()) {
            int n = q.size();
            leftmost = q.front()->val;
            for (int i = 0; i < n; ++i) {
                TreeNode* cur = q.front(); q.pop();
                if (cur->left) q.push(cur->left);
                if (cur->right) q.push(cur->right);
            }
        }
        return leftmost;
    }
};

```

## 515. Find Largest Value in Each Tree Row

## Description

You need to find the largest value in each row of a binary tree.

Example:

Input:

```
      1
     / \
    3   2
   / \   \
  5  3   9
```

Output: [1, 3, 9]

## Solution

05/04/2020:

```
/**
 * 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> largestValues(TreeNode* root) {
        if (root == nullptr) return {};
        vector<int> ret;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int n = q.size();
            int maxValue = q.front()->val;
            for (int i = 0; i < n; ++i) {
                TreeNode* cur = q.front(); q.pop();
                maxValue = max(maxValue, cur->val);
                if (cur->left) q.push(cur->left);
                if (cur->right) q.push(cur->right);
            }
            ret.push_back(maxValue);
        }
        return ret;
    }
};
```

```
}  
};
```

## 559. Maximum Depth of N-ary Tree

### Description

Given a n-ary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

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: 3

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: 5

Constraints:

The depth of the n-ary tree is less than or equal to 1000.

The total number of nodes is between [0, 10<sup>4</sup>].

### Solution

01/31/2020:

```
/*  
// 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:
    int maxDepth(Node* root) {
        if (root == nullptr) return 0;
        int depth = 0;
        for (auto& c : root->children) {
            int d = maxDepth(c);
            depth = max(d, depth);
        }
        return depth + 1;
    }
};

```

## 563. Binary Tree Tilt

### *Description*

Given a binary tree, return the tilt of the whole tree.

The tilt of a tree node is defined as the absolute difference between the sum of all left subtree node values and the sum of all right subtree node values. Null node has tilt 0.

The tilt of the whole tree is defined as the sum of all nodes' tilt.

Example:

Input:

```

    1
   / \

```

2 3  
Output: 1  
Explanation:  
Tilt of node 2 : 0  
Tilt of node 3 : 0  
Tilt of node 1 :  $|2-3| = 1$   
Tilt of binary tree :  $0 + 0 + 1 = 1$   
Note:

The sum of node values in any subtree won't exceed the range of 32-bit integer.  
All the tilt values won't exceed the range of 32-bit integer.

*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:
    int findTilt(TreeNode* root) {
        if (!root) return 0;
        return abs(sum(root->left) - sum(root->right)) + findTilt(root->left) +
findTilt(root->right);
    }

    int sum(TreeNode* root) {
        if (!root) return 0;
        return root->val + sum(root->left) + sum(root->right);
    }
};
```

## 589. N-ary Tree Preorder Traversal

*Description*

---

Given an n-ary tree, return the preorder 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).

Follow up:

Recursive solution is trivial, could you do it iteratively?

Example 1:

Input: root = [1,null,3,2,4,null,5,6]

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

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: [1,2,3,6,7,11,14,4,8,12,5,9,13,10]

Constraints:

The height of the n-ary tree is less than or equal to 1000

The total number of nodes is between [0, 10<sup>4</sup>]

*Solution*

[Discussion](#)

01/30/2020:

```
/*
// 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<int> preorder(Node* root) {
        if (root == nullptr) return {};
        vector<int> ret;
        ret.push_back(root->val);
        for (auto& c : root->children) {
            vector<int> t = preorder(c);
            ret.insert(ret.end(), t.begin(), t.end());
        }
        return ret;
    }
};

```

04/26/2020 (Non-recursive):

```

class Solution {
public:
    vector<int> preorder(Node* root) {
        vector<int> ret;
        if (root == nullptr) return ret;
        stack<Node*> st;
        st.push(root);
        while (!st.empty()) {
            Node* cur = st.top(); st.pop();
            ret.push_back(cur->val);
            for (auto it = cur->children.cbegin(); it != cur->children.crend(); ++it)
            {
                st.push(*it);
            }
        }
        return ret;
    }
};

```

04/26/2020 (Space efficient DFS):



```

class Solution {
public:
    vector<int> preorder(Node* root) {
        vector<int> ret;
        dfs(root, ret);
        return ret;
    }

    void dfs(Node* root, vector<int>& ret) {
        if (root == nullptr) return;
        ret.push_back(root->val);
        for (auto& c : root->children) {
            dfs(c, ret);
        }
    }
};

```

## 590. N-ary Tree Postorder Traversal

### *Description*

Given an n-ary tree, return the postorder traversal of its nodes' values.

N-ary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Follow up:

Recursive solution is trivial, could you do it iteratively?

Example 1:

Input: root = [1,null,3,2,4,null,5,6]

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

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: [2,6,14,11,7,3,12,8,4,13,9,10,5,1]
```

Constraints:

The height of the n-ary tree is less than or equal to 1000  
The total number of nodes is between [0, 10<sup>4</sup>]

*Solution*

01/30/2020:

```
/*
// 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<int> postorder(Node* root) {
        if (root == nullptr) return {};
        vector<int> nums;
        for (auto& c : root->children) {
            vector<int> n = postorder(c);
            nums.insert(nums.end(), n.begin(), n.end());
        }
        nums.push_back(root->val);
        return nums;
    }
};
```

04/26/2020 (Non-recursive postorder traversal):

```
class Solution {
public:
    vector<int> postorder(Node* root) {
        if (root == nullptr) return {};
        stack<Node*> st;
        st.push(root);
        vector<int> ret;
        unordered_set<Node*> visited;
        while (!st.empty()) {
            Node* cur = st.top(); st.pop();
            if (cur->children.empty()) {
                visited.insert(cur);
                ret.push_back(cur->val);
            } else {
                bool allChildrenVisited = true;
                for (auto& c : cur->children) {
                    if (visited.count(c) == 0) {
                        allChildrenVisited = false;
                        break;
                    }
                }
                if (visited.count(cur) == 0) {
                    if (allChildrenVisited) {
                        visited.insert(cur);
                        ret.push_back(cur->val);
                    } else {
                        st.push(cur);
                        for (auto it = cur->children.cbegin(); it != cur->children.crend();
++it) {
                            if (visited.count(*it) == 0) {
                                st.push(*it);
                            }
                        }
                    }
                }
            }
        }
        return ret;
    }
};
```

Reverse the process becomes preorder traversal:

```
class Solution {
public:
    vector<int> postorder(Node* root) {
```

```

if (root == nullptr) return {};
vector<int> ret;
stack<Node*> st;
st.push(root);
while (!st.empty()) {
    Node* cur = st.top(); st.pop();
    ret.push_back(cur->val);
    for (auto& c : cur->children) {
        st.push(c);
    }
}
reverse(ret.begin(), ret.end());
return ret;
}
};

```

## 617. Merge Two Binary Trees

### Description

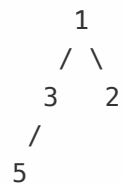
Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

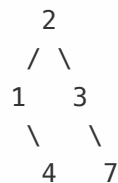
Example 1:

Input:

Tree 1

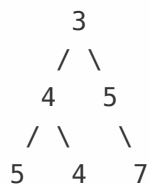


Tree 2



Output:

Merged tree:



Note: The merging process must start from the root nodes of both trees.

*Solution*

01/30/2020:

```
/**
 * 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* mergeTrees(TreeNode* t1, TreeNode* t2) {
        if (t1 == nullptr && t2 == nullptr) return nullptr;
        if (t1 == nullptr && t2 != nullptr) return t2;
        if (t1 != nullptr && t2 == nullptr) return t1;
        if (t1 != nullptr && t2 != nullptr) {
            t1->val += t2->val;
            t1->left = mergeTrees(t1->left, t2->left);
            t1->right = mergeTrees(t1->right, t2->right);
        }
        return t1;
    }
};
```

## 655. Print Binary Tree

*Description*

Print a binary tree in an  $m \times n$  2D string array following these rules:

The row number  $m$  should be equal to the height of the given binary tree.

The column number  $n$  should always be an odd number.

The root node's value (in string format) should be put in the exactly middle of the first row it can be put. The column and the row where the root node belongs will separate the rest space into two parts (left-bottom part and right-bottom part). You should print the left subtree in the left-bottom part and print the right subtree in the right-bottom part. The left-bottom part and the right-bottom part should have the same size. Even if one subtree is none while the other is not, you don't need to print anything for the none subtree but still need to leave the space as large as that for the other subtree. However, if two subtrees are none, then you don't need to leave space for both of them.

Each unused space should contain an empty string "".

Print the subtrees following the same rules.

Example 1:

Input:

```

    1
   /
  2

```

Output:

```

[["", "1", ""],
 ["2", "", ""]]

```

Example 2:

Input:

```

    1
   / \
  2   3
   \
    4

```

Output:

```

[["", "", "", "1", "", "", ""],
 [ "", "2", "", "", "", "3", "" ],
 [ "", "", "4", "", "", "", "" ] ]

```

Example 3:

Input:

```

    1
   / \
  2   5
 /
3
 /
4

```

Output:

```

[["", "", "", "", "", "", "", "1", "", "", "", "", "", ""]]
 [ "", "", "", "2", "", "", "", "", "", "", "5", "", "", ""]
 [ "", "3", "", "", "", "", "", "", "", "", "", "", "", ""]
 [ "4", "", "", "", "", "", "", "", "", "", "", "", "", ""] ]

```

Note: The height of binary tree is in the range of [1, 10].

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:
    vector<vector<string>> printTree(TreeNode* root) {
        if (!root) return { {} };
        int h = height(root), m = h, n = pow(2, h) - 1;
        vector<vector<string>> ret(m, vector<string>(n, ""));
        stack<pair<TreeNode*, pair<int, int>>> st;
        st.push({root, {0, 0}});
        while (!st.empty()) {
            pair<TreeNode*, pair<int, int>> p = st.top(); st.pop();
            TreeNode* cur = p.first;
            int i = p.second.second, j = p.second.first;
            ret[i][n / 2 + j] = to_string(cur->val);
            if (cur->left) st.push({cur->left, {j - pow(2, h - 2 - i), i + 1}});
            if (cur->right) st.push({cur->right, {j + pow(2, h - 2 - i), i + 1}});
        }
        return ret;
    }

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

## 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*

01/30/2020:

```
/**
 * 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* searchBST(TreeNode* root, int val) {
        if (root == nullptr) return nullptr;
        if (root->val > val) return searchBST(root->left, val);
        if (root->val < val) return searchBST(root->right, val);
        return root;
    }
};
```



---

## 872. Leaf-Similar Trees

### Description

Consider all the leaves of a binary tree. From left to right order, the values of those leaves form a leaf value sequence.

For example, in the given tree above, the leaf value sequence is (6, 7, 4, 9, 8).

Two binary trees are considered leaf-similar if their leaf value sequence is the same.

Return true if and only if the two given trees with head nodes root1 and root2 are leaf-similar.

Note:

Both of the given trees will have between 1 and 100 nodes.

### Solution

02/03/2020:

```
/**
 * 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:
    bool leafSimilar(TreeNode* root1, TreeNode* root2) {
        vector<int> nums1 = dfs(root1);
        vector<int> nums2 = dfs(root2);
        if (nums1.size() != nums2.size()) return false;
        for (int i = 0; i < (int)nums1.size(); ++i) {
```

```

        if (nums1[i] != nums2[i]) {
            return false;
        }
    }
    return true;
}
vector<int> dfs(TreeNode* root) {
    if (root == nullptr) return {};
    if (root->left == nullptr && root->right == nullptr) return {root->val};
    vector<int> nums1 = dfs(root->left);
    vector<int> nums2 = dfs(root->right);
    nums1.insert(nums1.end(), nums2.begin(), nums2.end());
    return nums1;
}
};

```

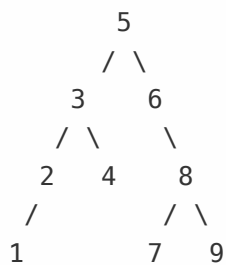
## 897. Increasing Order Search Tree

### Description

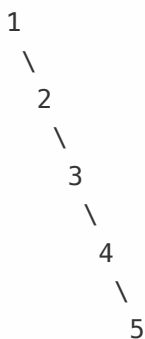
Given a binary search tree, rearrange the tree in in-order so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only 1 right child.

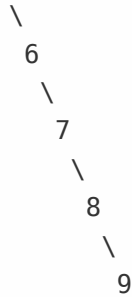
Example 1:

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



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





Note:

The number of nodes in the given tree will be between 1 and 100.  
Each node will have a unique integer value from 0 to 1000.

*Solution*

01/31/2020:

```

/**
 * 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* increasingBST(TreeNode* root) {
        vector<int> nodes = inorder(root);
        TreeNode *pre = new TreeNode(0), *cur = pre;
        for (auto& n : nodes) {
            cur->right = new TreeNode(n);
            cur = cur->right;
        }
        return pre->right;
    }

    vector<int> inorder(TreeNode* root) {
        if (root == nullptr) return {};
        vector<int> nodes;
        vector<int> l = inorder(root->left);
        vector<int> r = inorder(root->right);
        nodes.insert(nodes.end(), l.begin(), l.end());
        nodes.push_back(root->val);
        nodes.insert(nodes.end(), r.begin(), r.end());
        return nodes;
    }
}

```

```
};
```

## 938. Range Sum of BST

### Description

Given the root node of a binary search tree, return the sum of values of all nodes with value between L and R (inclusive).

The binary search tree is guaranteed to have unique values.

Example 1:

Input: root = [10,5,15,3,7,null,18], L = 7, R = 15

Output: 32

Example 2:

Input: root = [10,5,15,3,7,13,18,1,null,6], L = 6, R = 10

Output: 23

Note:

The number of nodes in the tree is at most 10000.

The final answer is guaranteed to be less than  $2^{31}$ .

### Solution

01/29/2020:

```
/**
 * 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 rangeSumBST(TreeNode* root, int L, int R) {
        if (root == nullptr) return 0;
```

```
    if (root->val > R) return rangeSumBST(root->left, L, R);
    if (root->val < L) return rangeSumBST(root->right, L, R);
    return root->val + rangeSumBST(root->left, L, R) + rangeSumBST(root->right,
L, R);
}
};
```

## 965. Univalued Binary Tree

### Description

A binary tree is univalued if every node in the tree has the same value.

Return true if and only if the given tree is univalued.

Example 1:

Input: [1,1,1,1,1,null,1]

Output: true

Example 2:

Input: [2,2,2,5,2]

Output: false

Note:

The number of nodes in the given tree will be in the range [1, 100].

Each node's value will be an integer in the range [0, 99].

### Solution

01/31/2020:

```
/**
 * 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:
    bool isUnivalTree(TreeNode* root) {
        if (root == nullptr) return true;
        return dfs(root, root->val);
    }

    bool dfs(TreeNode* root, int val) {
        if (root == nullptr) return true;
        if (root->val != val) return false;
        return dfs(root->left, val) && dfs(root->right, val);
    }
};

```

## 993. Cousins in Binary Tree

### Description

In a binary tree, the root node is at depth 0, and children of each depth k node are at depth k+1.

Two nodes of a binary tree are cousins if they have the same depth, but have different parents.

We are given the root of a binary tree with unique values, and the values x and y of two different nodes in the tree.

Return true if and only if the nodes corresponding to the values x and y are cousins.

Example 1:

Input: root = [1,2,3,4], x = 4, y = 3

Output: false

Example 2:

Input: root = [1,2,3,null,4,null,5], x = 5, y = 4

Output: true

Example 3:

Input: root = [1,2,3,null,4], x = 2, y = 3  
Output: false

Note:

The number of nodes in the tree will be between 2 and 100.  
Each node has a unique integer value from 1 to 100.

*Solution*

05/09/2020 [Discussion](#):

We can create a new data structure Node to store the parent, depth of each node.

```
class Node {
public:
    TreeNode* node;
    TreeNode* parent;
    int depth;
    Node(TreeNode* n, TreeNode* p, int d) : node(n), parent(p), depth(d) {};
};
```

Then apply the standard BFS to search for x and y, then check the conditions of being cousins nodes.

BFS:

```
class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        Node nx = bfs(root, x);
        Node ny = bfs(root, y);
        return nx.parent != ny.parent && nx.depth == ny.depth;
    }

    Node bfs(TreeNode* root, int x) {
        queue<Node> q;
        q.emplace(root, nullptr, 0);
        while (!q.empty()) {
            int n = q.size();
            for (int i = 0; i < n; ++i) {
                Node cur = q.front(); q.pop();
                if (cur.node->val == x) return cur;
                if (cur.node->left) q.emplace(cur.node->left, cur.node, cur.depth + 1);
                if (cur.node->right) q.emplace(cur.node->right, cur.node, cur.depth +
1);
            }
        }
    }
};
```

```

    }
}
return Node(nullptr, nullptr, -1);
}
};

```

Since we store the information about the depth of each node, it doesn't really matter whether we search in level-order. Thus the code can be simplified to the following: Non-recursive DFS:

```

class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        Node nx = dfs(root, x);
        Node ny = dfs(root, y);
        return nx.parent != ny.parent && nx.depth == ny.depth;
    }

    Node dfs(TreeNode* root, int x) {
        stack<Node> st;
        st.emplace(root, nullptr, 0);
        while (!st.empty()) {
            Node cur = st.top(); st.pop();
            if (cur.node->val == x) return cur;
            if (cur.node->left) st.emplace(cur.node->left, cur.node, cur.depth + 1);
            if (cur.node->right) st.emplace(cur.node->right, cur.node, cur.depth + 1);
        }
        return Node(nullptr, nullptr, -1);
    }
};

```

Recursive DFS:

```

class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        Node nx = dfs(Node(root, nullptr, 0), x);
        Node ny = dfs(Node(root, nullptr, 0), y);
        return nx.parent != ny.parent && nx.depth == ny.depth;
    }

    Node dfs(Node root, int x) {
        if (root.node == nullptr) return Node(nullptr, nullptr, -1);
        if (root.node->val == x) return root;
        Node left = dfs(Node(root.node->left, root.node, root.depth + 1), x);
        Node right = dfs(Node(root.node->right, root.node, root.depth + 1), x);
        if (left.node) return left;
        if (right.node) return right;
    }
};

```



```

        return Node(nullptr, nullptr, -1);
    }
};

```

DFS without using Node:

```

class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        pair<TreeNode*, int> nx = dfs(root, nullptr, x, 0);
        pair<TreeNode*, int> ny = dfs(root, nullptr, y, 0);
        return nx.first != ny.first && nx.second == ny.second;
    }

    pair<TreeNode*, int> dfs(TreeNode* root, TreeNode* parent, int x, int depth) {
        if (root == nullptr) return {nullptr, -1};
        if (root->val == x) return {parent, depth};
        pair<TreeNode*, int> left = dfs(root->left, root, x, depth + 1);
        pair<TreeNode*, int> right = dfs(root->right, root, x, depth + 1);
        if (left.first) return left;
        if (right.first) return right;
        return {nullptr, -1};
    }
};

```

Or we can directly check the parents of the nodes on each level: BFS without using Node:

```

class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        queue<pair<TreeNode*, int>> q;
        q.emplace(root, -1);
        while (!q.empty()) {
            int n = q.size();
            unordered_map<int, int> parent;
            for (int i = 0; i < n; ++i) {
                pair<TreeNode*, int> cur = q.front(); q.pop();
                parent[cur.first->val] = cur.second;
                if (cur.first->left != nullptr) q.emplace(cur.first->left, cur.first->val);
                if (cur.first->right != nullptr) q.emplace(cur.first->right, cur.first->val);
            }
            if (parent.count(x) > 0 && parent.count(y) > 0 && parent[x] != parent[y])
                return true;
        }
        return false;
    }
};

```

```
}  
};
```

## 1008. Construct Binary Search Tree from Preorder Traversal

### Description

Return the root node of a binary search tree that matches the given preorder traversal.

(Recall that a binary search tree is a binary tree where for every node, any descendant of `node.left` has a value  $<$  `node.val`, and any descendant of `node.right` has a value  $>$  `node.val`. Also recall that a preorder traversal displays the value of the node first, then traverses `node.left`, then traverses `node.right`.)

It's guaranteed that for the given test cases there is always possible to find a binary search tree with the given requirements.

Example 1:

Input: [8,5,1,7,10,12]

Output: [8,5,10,1,7,null,12]

Constraints:

$1 \leq \text{preorder.length} \leq 100$

$1 \leq \text{preorder}[i] \leq 10^8$

The values of preorder are distinct.

### 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 Solution {
public:
    TreeNode* bstFromPreorder(vector<int>& preorder, int start = 0, int stop =
INT_MAX) {
        if (preorder.empty()) return nullptr;
        if (stop == INT_MAX) stop = preorder.size() - 1;
        if (start > stop) return nullptr;
        TreeNode* root = new TreeNode(preorder[start]);
        int i = start + 1;
        for (; i <= stop; ++i)
            if (preorder[i] > preorder[start])
                break;
        root->left = bstFromPreorder(preorder, start + 1, i - 1);
        root->right = bstFromPreorder(preorder, i, stop);
        return root;
    }
};

```

## 1022. Sum of Root To Leaf Binary Numbers

### Description

Given a binary tree, each node has value 0 or 1. Each root-to-leaf path represents a binary number starting with the most significant bit. For example, if the path is 0 → 1 → 1 → 0 → 1, then this could represent 01101 in binary, which is 13.

For all leaves in the tree, consider the numbers represented by the path from the root to that leaf.

Return the sum of these numbers.

Example 1:

Input: [1,0,1,0,1,0,1]

Output: 22

Explanation: (100) + (101) + (110) + (111) = 4 + 5 + 6 + 7 = 22

Note:

The number of nodes in the tree is between 1 and 1000.  
node.val is 0 or 1.  
The answer will not exceed  $2^{31} - 1$ .

*Solution*

02/03/2020:

```
/**
 * 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 sumRootToLeaf(TreeNode* root) {
        return dps(root, 0);
    }

    int dps(TreeNode* root, int sum) {
        if (root == nullptr) return 0;
        sum = sum * 2 + root->val;
        if (root->left == nullptr && root->right == nullptr) return sum;
        return dps(root->left, sum) + dps(root->right, sum);
    }
};
```

## 1325. Delete Leaves With a Given Value

*Description*

Given a binary tree root and an integer target, delete all the leaf nodes with value target.

Note that once you delete a leaf node with value target, if its parent node becomes a leaf node and has the value target, it should also be deleted (you need to continue doing that until you can't).

Example 1:

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

Output: [1,null,3,null,4]

Explanation: Leaf nodes in green with value (target = 2) are removed (Picture in left).

After removing, new nodes become leaf nodes with value (target = 2) (Picture in center).

Example 2:

Input: root = [1,3,3,3,2], target = 3

Output: [1,3,null,null,2]

Example 3:

Input: root = [1,2,null,2,null,2], target = 2

Output: [1]

Explanation: Leaf nodes in green with value (target = 2) are removed at each step.

Example 4:

Input: root = [1,1,1], target = 1

Output: []

Example 5:

Input: root = [1,2,3], target = 1

Output: [1,2,3]

Constraints:

$1 \leq \text{target} \leq 1000$

Each tree has at most 3000 nodes.

Each node's value is between [1, 1000].

*Solution*

01/19/2020:

```
/**
 * 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* removeLeafNodes(TreeNode* root, int target) {
        if (root == nullptr) return root;
        root->left = removeLeafNodes(root->left, target);
        root->right = removeLeafNodes(root->right, target);
        if (root->left == nullptr && root->right == nullptr && root->val == target)
            root = nullptr;
        return root;
    }
};

```

## 1333. Filter Restaurants by Vegan-Friendly, Price and Distance

### Description

Given the array restaurants where  $\text{restaurants}[i] = [\text{id}_i, \text{rating}_i, \text{veganFriendly}_i, \text{price}_i, \text{distance}_i]$ . You have to filter the restaurants using three filters.

The `veganFriendly` filter will be either `true` (meaning you should only include restaurants with `veganFriendlyi` set to `true`) or `false` (meaning you can include any restaurant). In addition, you have the filters `maxPrice` and `maxDistance` which are the maximum value for price and distance of restaurants you should consider respectively.

Return the array of restaurant IDs after filtering, ordered by rating from highest to lowest. For restaurants with the same rating, order them by id from highest to lowest. For simplicity `veganFriendlyi` and `veganFriendly` take value 1 when it is true, and 0 when it is false.

Example 1:

Input: `restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]]`, `veganFriendly = 1`, `maxPrice = 50`, `maxDistance = 10`

Output: `[3,1,5]`

Explanation:

The restaurants are:

Restaurant 1 [id=1, rating=4, veganFriendly=1, price=40, distance=10]

Restaurant 2 [id=2, rating=8, veganFriendly=0, price=50, distance=5]

Restaurant 3 [id=3, rating=8, veganFriendly=1, price=30, distance=4]

Restaurant 4 [id=4, rating=10, veganFriendly=0, price=10, distance=3]

Restaurant 5 [id=5, rating=1, veganFriendly=1, price=15, distance=1]

After filter restaurants with veganFriendly = 1, maxPrice = 50 and maxDistance = 10 we have restaurant 3, restaurant 1 and restaurant 5 (ordered by rating from highest to lowest).

Example 2:

Input: restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 50, maxDistance = 10

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

Explanation: The restaurants are the same as in example 1, but in this case the filter veganFriendly = 0, therefore all restaurants are considered.

Example 3:

Input: restaurants = [[1,4,1,40,10],[2,8,0,50,5],[3,8,1,30,4],[4,10,0,10,3],[5,1,1,15,1]], veganFriendly = 0, maxPrice = 30, maxDistance = 3

Output: [4,5]

Constraints:

```
1 <= restaurants.length <= 10^4
restaurants[i].length == 5
1 <= idi, ratingi, pricei, distancei <= 10^5
1 <= maxPrice, maxDistance <= 10^5
veganFriendlyi and veganFriendly are 0 or 1.
All idi are distinct.
```

*Solution*

01/25/2020:

```
class Solution {
public:
    vector<int> filterRestaurants(vector<vector<int>>& restaurants, int
veganFriendly, int maxPrice, int maxDistance) {
        vector<vector<int>> filtered;
        for (auto& r : restaurants)
            if (r[3] <= maxPrice && r[4] <= maxDistance)
                if (veganFriendly == 0 || (veganFriendly == 1 && r[2] == 1))
                    filtered.push_back(r);
        sort(filtered.begin(), filtered.end(), [](vector<int>& f1, vector<int>& f2)
-> bool { return f1[1] == f2[1] ? f1[0] > f2[0] : f1[1] > f2[1]; });
        vector<int> ret;
    }
};
```

```

    for (auto& f : filtered) ret.push_back(f[0]);
    return ret;
}
};

```

01/25/2020:

```

class Solution {
public:
    vector<int> filterRestaurants(vector<vector<int>>& restaurants, int
veganFriendly, int maxPrice, int maxDistance) {
        vector<int> ret;
        for (int i = 0; i < restaurants.size(); ++i) {
            if ((veganFriendly == 0 || (veganFriendly == 1 && restaurants[i][2] == 1))
&& restaurants[i][3] <= maxPrice
&& restaurants[i][4] <= maxDistance) {
                ret.push_back(i);
            }
        }
        quicksort(ret, 0, ret.size() - 1, restaurants);
        for (auto& r : ret) r = restaurants[r][0];
        return ret;
    }
    void quicksort(vector<int>& ret, int s1, int s2, vector<vector<int>>&
restaurants) {
        if (s1 >= s2) return;
        int j = s1;
        for (int i = s1; i <= s2; ++i) {
            if (restaurants[ret[i]][1] > restaurants[ret[s2]][1] ||
(restaurants[ret[i]][1] == restaurants[ret[s2]][1] && restaurants[ret[i]][0] >
restaurants[ret[s2]][0])) {
                swap(ret[i], ret[j++]);
            }
        }
        swap(ret[j], ret[s2]);
        quicksort(ret, s1, j - 1, restaurants);
        quicksort(ret, j + 1, s2, restaurants);
    }
};

```

## 1443. Minimum Time to Collect All Apples in a Tree

*Description*



Given an undirected tree consisting of  $n$  vertices numbered from  $0$  to  $n-1$ , which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend in order to collect all apples in the tree starting at vertex  $0$  and coming back to this vertex.

The edges of the undirected tree are given in the array `edges`, where `edges[i] = [fromi, toi]` means that exists an edge connecting the vertices `fromi` and `toi`. Additionally, there is a boolean array `hasApple`, where `hasApple[i] = true` means that vertex `i` has an apple, otherwise, it does not have any apple.

Example 1:

Input: `n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,true,true,false]`

Output: 8

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 2:

Input: `n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,false,true,false]`

Output: 6

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 3:

Input: `n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,false,false,false,false,false]`

Output: 0

Constraints:

```
1 <= n <= 10^5
edges.length == n-1
edges[i].length == 2
0 <= fromi, toi <= n-1
fromi < toi
hasApple.length == n
```

05/10/2020:

```
class Solution {
public:
    int minTime(int n, vector<vector<int>>& edges, vector<bool>& hasApple) {
        if (n == 0 || edges.empty() || edges[0].empty() || hasApple.empty()) return
0;
        unordered_map<int, unordered_set<int>> g;
        for (auto& e : edges) {
            g[e[0]].insert(e[1]);
            g[e[1]].insert(e[0]);
        }
        const int INF = numeric_limits<int>::max();
        vector<int> dist(n, INF);
        vector<int> pred(n);
        iota(pred.begin(), pred.end(), 0);
        dist[0] = 0;
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> q;
        unordered_set<int> visited;
        q.emplace(dist[0], 0);
        while (!q.empty()) {
            pair<int, int> cur = q.top(); q.pop();
            int i = cur.second;
            if (visited.count(i)) continue;
            int j = pred[i];
            for (auto& k : g[i]) {
                if (dist[i] != INF && dist[k] > dist[i] + 1) {
                    dist[k] = dist[i] + 1;
                    pred[k] = i;
                }
                q.emplace(dist[k], k);
            }
            visited.insert(i);
        }
        set<pair<int, int>> paths;
        for (int i = 0; i < n; ++i) {
            if (hasApple[i] == false) continue;
            int cur = i;
            while (cur != 0) {
                paths.insert({pred[cur], cur});
                cur = pred[cur];
            }
        }
        return paths.size() * 2;
    }
};
```

# 1448. Count Good Nodes in Binary Tree

## Description

Given a binary tree root, a node X in the tree is named good if in the path from root to X there are no nodes with a value greater than X.

Return the number of good nodes in the binary tree.

Example 1:

Input: root = [3,1,4,3,null,1,5]

Output: 4

Explanation: Nodes in blue are good.

Root Node (3) is always a good node.

Node 4 → (3,4) is the maximum value in the path starting from the root.

Node 5 → (3,4,5) is the maximum value in the path

Node 3 → (3,1,3) is the maximum value in the path.

Example 2:

Input: root = [3,3,null,4,2]

Output: 3

Explanation: Node 2 → (3, 3, 2) is not good, because "3" is higher than it.

Example 3:

Input: root = [1]

Output: 1

Explanation: Root is considered as good.

Constraints:

The number of nodes in the binary tree is in the range [1, 10<sup>5</sup>].

Each node's value is between [-10<sup>4</sup>, 10<sup>4</sup>].

## Solution

05/17/2020:

```
/**
 * 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 goodNodes(TreeNode* root, int cur_max = INT_MIN) {
        if (!root) return 0;
        int n = root->val >= cur_max ? 1 : 0;
        cur_max = max(root->val, cur_max);
        return n + goodNodes(root->left, cur_max) + goodNodes(root->right, cur_max);
    }
};

```

## 1457. Pseudo-Palindromic Paths in a Binary Tree

### Description

Given a binary tree where node values are digits from 1 to 9. A path in the binary tree is said to be pseudo-palindromic if at least one permutation of the node values in the path is a palindrome.

Return the number of pseudo-palindromic paths going from the root node to leaf nodes.

Example 1:

Input: root = [2,3,1,3,1,null,1]

Output: 2

Explanation: The figure above represents the given binary tree. There are three paths going from the root node to leaf nodes: the red path [2,3,3], the green path [2,1,1], and the path [2,3,1]. Among these paths only red path and green path are pseudo-palindromic paths since the red path [2,3,3] can be rearranged in [3,2,3] (palindrome) and the green path [2,1,1] can be rearranged in [1,2,1] (palindrome).

Example 2:

Input: root = [2,1,1,1,3,null,null,null,null,null,1]

Output: 1

Explanation: The figure above represents the given binary tree. There are three paths going from the root node to leaf nodes: the green path [2,1,1], the path [2,1,3,1], and the path [2,1]. Among these paths only the green path is pseudo-palindromic since [2,1,1] can be rearranged in [1,2,1] (palindrome).

Example 3:

Input: root = [9]

Output: 1

Constraints:

The given binary tree will have between 1 and  $10^5$  nodes.  
Node values are digits from 1 to 9.

*Solution*

05/23/2020:

```
/**
 * 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 ret = 0;
    vector<int> cnt;
    int pseudoPalindromicPaths (TreeNode* root) {
        cnt.resize(10, 0);
        dfs(root, 0);
        return ret;
    }

    void dfs(TreeNode* root, int numberOfOdds) {
        if (root == nullptr) return;
```

```

bool isOdd = ++cnt[root->val] % 2 == 1;
numberOfOdds += isOdd ? 1 : -1;
if (root->left == nullptr && root->right == nullptr) {
    ret += numberOfOdds <= 1 ? 1 : 0;
    --cnt[root->val];
    return;
}
dfs(root->left, numberOfOdds);
dfs(root->right, numberOfOdds);
--cnt[root->val];
}
};

```

```

class Solution {
public:
    vector<vector<int>> nums;
    int pseudoPalindromicPaths (TreeNode* root) {
        int ret = 0;
        vector<int> path;
        dfs(root, path);
        for (auto& path : nums) ret += isPalindromic(path) ? 1 : 0;
        return ret;
    }

    void dfs(TreeNode* root, vector<int>& path) {
        if (root == nullptr) return;
        if (root->left == nullptr && root->right == nullptr) {
            path.push_back(root->val);
            nums.push_back(path);
            path.pop_back();
            return;
        }
        path.push_back(root->val);
        dfs(root->left, path);
        dfs(root->right, path);
        path.pop_back();
    }

    bool isPalindromic(vector<int>& path) {
        vector<int> freq(10, 0);
        int numberOfOdds = 0;
        for (auto& p : path) ++freq[p];
        for (int i = 0; i < 10; ++i) numberOfOdds += freq[i] % 2 ? 1 : 0;
        return numberOfOdds <= 1;
    }
};

```

# 1469. Find All The Lonely Nodes

## Description

In a binary tree, a lonely node is a node that is the only child of its parent node. The root of the tree is not lonely because it does not have a parent node.

Given the root of a binary tree, return an array containing the values of all lonely nodes in the tree. Return the list in any order.

Example 1:

Input: root = [1,2,3,null,4]

Output: [4]

Explanation: Light blue node is the only lonely node.

Node 1 is the root and is not lonely.

Nodes 2 and 3 have the same parent and are not lonely.

Example 2:

Input: root = [7,1,4,6,null,5,3,null,null,null,null,null,2]

Output: [6,2]

Explanation: Light blue nodes are lonely nodes.

Please remember that order doesn't matter, [2,6] is also an acceptable answer.

Example 3:

Input: root = [11,99,88,77,null,null,66,55,null,null,44,33,null,null,22]

Output: [77,55,33,66,44,22]

Explanation: Nodes 99 and 88 share the same parent. Node 11 is the root.

All other nodes are lonely.

Example 4:

Input: root = [197]

Output: []

Example 5:

Input: root = [31,null,78,null,28]

Output: [78,28]

Constraints:

The number of nodes in the tree is in the range [1, 1000].

Each node's value is between  $[1, 10^6]$ .

*Solution*

06/11/2020:

Using iteration:

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

Using recursion:



```

class Solution {
public:
    vector<int> ret;
    vector<int> getLonelyNodes(TreeNode* root) {
        if (root == nullptr || (!root->left && !root->right)) return ret;
        if (!root->left) ret.push_back(root->right->val);
        if (!root->right) ret.push_back(root->left->val);
        getLonelyNodes(root->left);
        getLonelyNodes(root->right);
        return ret;
    }
};

```

## 1490. Clone N-ary Tree

### Description

Given a root of an N-ary tree, return a deep copy (clone) of the tree.

Each node in the n-ary tree contains a val (int) and a list (List[Node]) of its children.

```

class Node {
    public int val;
    public List<Node> children;
}

```

N-ary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Follow up: Can your solution work for the graph problem?

Example 1:

Input: root = [1,null,3,2,4,null,5,6]

Output: [1,null,3,2,4,null,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,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]

Constraints:

The depth of the n-ary tree is less than or equal to 1000.

The total number of nodes is between [0, 10<sup>4</sup>].

*Solution*

06/27/2020:

```
/*
// 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:
    Node* cloneTree(Node* root) {
        if (!root) return nullptr;
        queue<Node*> q, qc;
        q.push(root);
        Node* clone = new Node(root->val);
        qc.push(clone);
        while (!q.empty()) {
            int sz = q.size();
            for (int s = 0; s < sz; ++s) {
```

```

Node* cur = q.front(); q.pop();
Node* ccur = qc.front(); qc.pop();
for (auto& c : cur->children) {
    q.push(c);
    Node* cnode = new Node(c->val);
    qc.push(cnode);
    ccur->children.push_back(cnode);
}
}
}
return clone;
}
};

```

```

class Solution {
public:
    Node* cloneTree(Node* root) {
        if (!root) return nullptr;
        Node* clone = new Node(root->val);
        for (auto& c : root->children) clone->children.push_back(cloneTree(c));
        return clone;
    }
};

```

# Math

## 2. Add Two Numbers

*Description*

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example:

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

*Solution*

01/25/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode *pre = new ListNode(0), *cur = pre;
        for (int carry = 0; l1 || l2 || carry > 0; cur = cur->next, carry /= 10) {
            if (l1) carry += l1->val, l1 = l1->next;
            if (l2) carry += l2->val, l2 = l2->next;
            cur->next = new ListNode(carry % 10);
        }
        return pre->next;
    }
};
```

## 7. Reverse Integer

*Description*

Given a 32-bit signed integer, reverse digits of an integer.

Example 1:

Input: 123  
Output: 321  
Example 2:

Input: -123  
Output: -321  
Example 3:

Input: 120  
Output: 21  
Note:

Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range:  $[-2^{31}, 2^{31} - 1]$ . For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

*Solution*

01/29/2020:

```
class Solution {
public:
    int reverse(int x) {
        string s = x > 0 ? to_string(x) : to_string(-(long)x);
        while (s.size() > 1 && s.back() == '0') s.pop_back();
        std::reverse(s.begin(), s.end());
        try {
            return x > 0 ? stoi(s) : -1 * stoi(s);
        } catch(std::out_of_range& e) {
            return 0;
        }
    }
};
```

## 9. Palindrome Number

---

*Description*

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

Example 1:

Input: 121

Output: true

Example 2:

Input: -121

Output: false

Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3:

Input: 10

Output: false

Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Follow up:

Could you solve it without converting the integer to a string?

*Solution*

01/29/2020:

```
class Solution {
public:
    bool isPalindrome(int x) {
        if (x < 0) return false;
        string s = to_string(x);
        for (int l = 0, r = s.size() - 1; l < r; ++l, --r)
            if (s[l] != s[r]) return false;
        return true;
    }
};
```

## 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*

02/03/2020:

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

## 172. Factorial Trailing Zeroes

*Description*

Given an integer  $n$ , return the number of trailing zeroes in  $n!$ .

Example 1:

Input: 3

Output: 0

Explanation:  $3! = 6$ , no trailing zero.

Example 2:

Input: 5

Output: 1

Explanation:  $5! = 120$ , one trailing zero.

Note: Your solution should be in logarithmic time complexity.

*Solution*

05/20/2020:

```
class Solution {
public:
    int trailingZeroes(int n) {
        int ret = 0;
        for (; n >= 5; ret += n / 5, n /= 5);
        return ret;
    }
};
```

```
class Solution {
public:
    int trailingZeroes(int n) {
        if (n < 5) return 0;
        return n / 5 + trailingZeroes(n / 5);
    }
};
```

## 202. Happy Number

### *Description*

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example:

Input: 19

Output: true

Explanation:

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

### *Solution*

01/26/2020:



```
class Solution {
public:
    bool isHappy(int n) {
        unordered_set<int> s;
        for (int sum = 0; s.find(n) == s.end(); n = sum, sum = 0) {
            s.insert(n);
            for (; n > 0; n /= 10)
                sum += (n % 10) * (n % 10);
        }
        return s.find(1) != s.end();
    }
};
```

## 217. Contains Duplicate

### *Description*

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

Example 1:

Input: [1,2,3,1]

Output: true

Example 2:

Input: [1,2,3,4]

Output: false

Example 3:

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

Output: true

### *Solution*

05/17/2020:

```

class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_set<int> s;
        for (auto& n : nums) {
            if (s.count(n) > 0) {
                return true;
            }
            s.insert(n);
        }
        return false;
    }
};

```

```

class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_map<int, int> mp;
        for (auto& n : nums)
            if (++mp[n] > 1)
                return true;
        return false;
    }
};

```

## 258. Add Digits

### *Description*

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

Example:

Input: 38

Output: 2

Explanation: The process is like:  $3 + 8 = 11$ ,  $1 + 1 = 2$ .

Since 2 has only one digit, return it.

Follow up:

Could you do it without any loop/recursion in  $O(1)$  runtime?

### *Solution*

05/23/2020:

```
class Solution {
public:
    int addDigits(int num) {
        int ret = num;
        while (ret > 9) {
            for (ret = 0; num != 0; ret += num % 10, num /= 10);
            num = ret;
        }
        return ret;
    }
};
```

## 263. Ugly Number

### *Description*

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

Example 1:

Input: 6

Output: true

Explanation:  $6 = 2 \times 3$

Example 2:

Input: 8

Output: true

Explanation:  $8 = 2 \times 2 \times 2$

Example 3:

Input: 14

Output: false

Explanation: 14 is not ugly since it includes another prime factor 7.

Note:

1 is typically treated as an ugly number.

Input is within the 32-bit signed integer range:  $[-2^{31}, 2^{31} - 1]$ .

### *Solution*

05/23/2020:

```
class Solution {
public:
    bool isUgly(int num) {
        if (num <= 0) return false;
        while (num % 2 == 0) num /= 2;
        while (num % 3 == 0) num /= 3;
        while (num % 5 == 0) num /= 5;
        return num == 1;
    }
};
```

## 476. Number Complement

### *Description*

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

#### Note:

The given integer is guaranteed to fit within the range of a 32-bit signed integer.

You could assume no leading zero bit in the integer's binary representation.

Example 1:

Input: 5

Output: 2

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

Example 2:

Input: 1

Output: 0

Explanation: The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

### *Solution*

02/03/2020:

```

class Solution {
public:
    int findComplement(int num) {
        int ret = 0, bits = 0;
        for (; num != 0; num /= 2, ++bits) {
            if (num % 2 == 0) {
                ret += 1 << bits;
            }
        }
        return ret;
    }
};

```

## 645. Set Mismatch

### Description

The set  $S$  originally contains numbers from 1 to  $n$ . But unfortunately, due to the data error, one of the numbers in the set got duplicated to another number in the set, which results in repetition of one number and loss of another number.

Given an array `nums` representing the data status of this set after the error. Your task is to firstly find the number occurs twice and then find the number that is missing. Return them in the form of an array.

Example 1:

Input: `nums = [1,2,2,4]`

Output: `[2,3]`

Note:

The given array size will in the range `[2, 10000]`.

The given array's numbers won't have any order.

### Solution

05/20/2020:

```

class Solution {
public:
    vector<int> findErrorNums(vector<int>& nums) {
        int n = nums.size();
        vector<int> cnt(n, 0);
        vector<int> ret;
        for (auto& i : nums) {

```

```

        if (++cnt[i - 1] == 2) {
            ret.push_back(i);
        }
    }
    for (int i = 1; i <= n; ++i) {
        if (cnt[i - 1] == 0) {
            ret.push_back(i);
        }
    }
    return ret;
}
};

```

## 728. Self Dividing Numbers

### Description

A self-dividing number is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because  $128 \% 1 == 0$ ,  $128 \% 2 == 0$ , and  $128 \% 8 == 0$ .

Also, a self-dividing number is not allowed to contain the digit zero.

Given a lower and upper number bound, output a list of every possible self dividing number, including the bounds if possible.

Example 1:

Input:

left = 1, right = 22

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]

Note:

The boundaries of each input argument are  $1 \leq \text{left} \leq \text{right} \leq 10000$ .

### Solution

01/30/2020:

```

class Solution {
public:
    vector<int> selfDividingNumbers(int left, int right) {
        vector<int> ret;
        for (int i = left; i <= right; ++i) {
            if (self_dividing(i)) {

```

```

        ret.push_back(i);
    }
}
return ret;
}

bool self_dividing(int n) {
    string s = to_string(n);
    for (auto& c : s) {
        if (c == '0' || (c != '0' && n % (c - '0') != 0))
            return false;
    }
    return true;
}
};

```

## 869. Reordered Power of 2

### *Description*

Starting with a positive integer  $N$ , we reorder the digits in any order (including the original order) such that the leading digit is not zero.

Return true if and only if we can do this in a way such that the resulting number is a power of 2.

Example 1:

Input: 1

Output: true

Example 2:

Input: 10

Output: false

Example 3:

Input: 16

Output: true

Example 4:

Input: 24

Output: false

Example 5:

Input: 46  
Output: true

Note:

$1 \leq N \leq 10^9$

*Solution*

06/16/2020:

```
class Solution {
public:
    bool reorderedPowerOf2(int N) {
        unordered_set<string> powerOfTwos;
        for (int i = 0; i < 32; ++i) {
            int n = 1 << i;
            string s = to_string(n);
            sort(s.begin(), s.end());
            powerOfTwos.insert(s);
        }
        string s = to_string(N);
        sort(s.begin(), s.end());
        return powerOfTwos.count(s) > 0;
    }
};
```

```
class Solution {
public:
    bool reorderedPowerOf2(int N) {
        string s = to_string(N);
        sort(s.begin(), s.end());
        unordered_set<string> seen;
        do {
            if (seen.count(s)) continue;
            if (s[0] == '0') continue;
            long n = stol(s);
            if (n <= INT_MAX && isPowerOfTwo(n)) return true;
            seen.insert(s);
        } while (next_permutation(s.begin(), s.end()));
        return false;
    }

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



```
};
```

## 883. Projection Area of 3D Shapes

### *Description*

On a  $N * N$  grid, we place some  $1 * 1 * 1$  cubes that are axis-aligned with the  $x$ ,  $y$ , and  $z$  axes.

Each value  $v = \text{grid}[i][j]$  represents a tower of  $v$  cubes placed on top of grid cell  $(i, j)$ .

Now we view the projection of these cubes onto the  $xy$ ,  $yz$ , and  $zx$  planes.

A projection is like a shadow, that maps our 3 dimensional figure to a 2 dimensional plane.

Here, we are viewing the "shadow" when looking at the cubes from the top, the front, and the side.

Return the total area of all three projections.

Example 1:

Input: `[[2]]`

Output: 5

Example 2:

Input: `[[1,2],[3,4]]`

Output: 17

Explanation:

Here are the three projections ("shadows") of the shape made with each axis-aligned plane.

Example 3:

Input: `[[1,0],[0,2]]`

Output: 8

Example 4:

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

Output: 14

Example 5:

Input: [[2,2,2],[2,1,2],[2,2,2]]

Output: 21

Note:

1 <= grid.length = grid[0].length <= 50

0 <= grid[i][j] <= 50

*Solution*

01/31/2020:

```
class Solution {
public:
    int projectionArea(vector<vector<int>>& grid) {
        int x = 0, y = 0, z = 0;
        vector<int> col_max(grid[0].size(), 0);
        vector<int> row_max(grid.size(), 0);
        for (int i = 0; i < (int)grid.size(); ++i) {
            for (int j = 0; j < (int)grid[0].size(); ++j) {
                if (grid[i][j] > 0) {
                    ++z;
                }
                row_max[i] = max(row_max[i], grid[i][j]);
            }
        }
        for (int j = 0; j < (int)grid[0].size(); ++j) {
            for (int i = 0; i < (int)grid.size(); ++i) {
                col_max[j] = max(col_max[j], grid[i][j]);
            }
        }
        for (auto& r : row_max) {
            y += r;
        }
        for (auto& c : col_max) {
            x += c;
        }
        return x + y + z;
    }
};
```

## 908. Smallest Range I

*Description*

---

Given an array A of integers, for each integer A[i] we may choose any x with  $-K \leq x \leq K$ , and add x to A[i].

After this process, we have some array B.

Return the smallest possible difference between the maximum value of B and the minimum value of B.

Example 1:

Input: A = [1], K = 0

Output: 0

Explanation: B = [1]

Example 2:

Input: A = [0,10], K = 2

Output: 6

Explanation: B = [2,8]

Example 3:

Input: A = [1,3,6], K = 3

Output: 0

Explanation: B = [3,3,3] or B = [4,4,4]

Note:

$1 \leq A.length \leq 10000$

$0 \leq A[i] \leq 10000$

$0 \leq K \leq 10000$

*Solution*

02/03/2020:

```
class Solution {
public:
    int smallestRangeI(vector<int>& A, int K) {
        int r = *max_element(A.begin(), A.end()) - *min_element(A.begin(), A.end());
        return max(0, r - 2 * K);
    }
};
```

# 970. Powerful Integers

## Description

Given two positive integers  $x$  and  $y$ , an integer is powerful if it is equal to  $x^i + y^j$  for some integers  $i \geq 0$  and  $j \geq 0$ .

Return a list of all powerful integers that have value less than or equal to  $bound$ .

You may return the answer in any order. In your answer, each value should occur at most once.

Example 1:

Input:  $x = 2, y = 3, bound = 10$

Output:  $[2,3,4,5,7,9,10]$

Explanation:

$$2 = 2^0 + 3^0$$

$$3 = 2^1 + 3^0$$

$$4 = 2^0 + 3^1$$

$$5 = 2^1 + 3^1$$

$$7 = 2^2 + 3^1$$

$$9 = 2^3 + 3^0$$

$$10 = 2^0 + 3^2$$

Example 2:

Input:  $x = 3, y = 5, bound = 15$

Output:  $[2,4,6,8,10,14]$

Note:

$$1 \leq x \leq 100$$

$$1 \leq y \leq 100$$

$$0 \leq bound \leq 10^6$$

## Solution

05/20/2020:

```
class Solution {
public:
    vector<int> powerfulIntegers(int x, int y, int bound) {
        vector<int> xPowers, yPowers;
```

```

unordered_set<int> ret;
for (int i = 0; pow(x, i) <= bound; ++i) {
    xPowers.push_back(pow(x, i));
    if (x == 1) break;
}
for (int i = 0; pow(y, i) <= bound; ++i) {
    yPowers.push_back(pow(y, i));
    if (y == 1) break;
}
for (int i = 0; i < (int)xPowers.size(); ++i) {
    for (int j = 0; j < (int)yPowers.size(); ++j) {
        int p = xPowers[i] + yPowers[j];
        if (p <= bound) {
            ret.insert(p);
        }
    }
}
return vector<int>(ret.begin(), ret.end());
}
};

```

## 1103. Distribute Candies to People

### *Description*

We distribute some number of candies, to a row of  $n = \text{num\_people}$  people in the following way:

We then give 1 candy to the first person, 2 candies to the second person, and so on until we give  $n$  candies to the last person.

Then, we go back to the start of the row, giving  $n + 1$  candies to the first person,  $n + 2$  candies to the second person, and so on until we give  $2 * n$  candies to the last person.

This process repeats (with us giving one more candy each time, and moving to the start of the row after we reach the end) until we run out of candies. The last person will receive all of our remaining candies (not necessarily one more than the previous gift).

Return an array (of length  $\text{num\_people}$  and sum candies) that represents the final distribution of candies.

Example 1:

Input: candies = 7, num\_people = 4

Output: [1,2,3,1]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0,0].

On the second turn, ans[1] += 2, and the array is [1,2,0,0].

On the third turn, ans[2] += 3, and the array is [1,2,3,0].

On the fourth turn, ans[3] += 1 (because there is only one candy left), and the final array is [1,2,3,1].

Example 2:

Input: candies = 10, num\_people = 3

Output: [5,2,3]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0].

On the second turn, ans[1] += 2, and the array is [1,2,0].

On the third turn, ans[2] += 3, and the array is [1,2,3].

On the fourth turn, ans[0] += 4, and the final array is [5,2,3].

Constraints:

1 <= candies <= 10<sup>9</sup>

1 <= num\_people <= 1000

*Solution*

05/10/2020:

```
class Solution {
public:
    vector<int> distributeCandies(int candies, int num_people) {
        int n = num_people;
        vector<int> ret(n, 0);
        int i = 0;
        while (candies > 0) {
            if (i + 1 <= candies) {
                ret[i % n] += i + 1;
                candies -= i + 1;
            } else {
                ret[i % n] += candies;
                candies = 0;
            }
            ++i;
        }
        return ret;
    }
};
```

# 1134. Armstrong Number

## Description

The k-digit number N is an Armstrong number if and only if the k-th power of each digit sums to N.

Given a positive integer N, return true if and only if it is an Armstrong number.

Example 1:

Input: 153

Output: true

Explanation:

153 is a 3-digit number, and  $153 = 1^3 + 5^3 + 3^3$ .

Example 2:

Input: 123

Output: false

Explanation:

123 is a 3-digit number, and  $123 \neq 1^3 + 2^3 + 3^3 = 36$ .

Note:

$1 \leq N \leq 10^8$

## Solution

01/29/2020:

```
class Solution {
public:
    bool isArmstrong(int N) {
        int n = ceil(log10(N + 1)), t = 0;
        for (int m = N; m != 0; m /= 10) {
            t += pow(m % 10, n);
        }
        return t == N;
    }
};
```

# 1185. Day of the Week

## Description

Given a date, return the corresponding day of the week for that date.

The input is given as three integers representing the day, month and year respectively.

Return the answer as one of the following values {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}.

Example 1:

Input: day = 31, month = 8, year = 2019

Output: "Saturday"

Example 2:

Input: day = 18, month = 7, year = 1999

Output: "Sunday"

Example 3:

Input: day = 15, month = 8, year = 1993

Output: "Sunday"

Constraints:

The given dates are valid dates between the years 1971 and 2100.

## Solution

02/03/2020:

```
class Solution {
public:
    int daysOfMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    string days[7] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};

    string dayOfTheWeek(int day, int month, int year) {
        return days[daysFrom1971(day, month, year) % 7];
    }

    bool isLeapYear(int year) {
```



```

    if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)) return true;
    return false;
}

int daysFrom1971(int day, int month, int year) {
    int days = 0;
    for (int i = 1971; i < year; ++i) {
        days += isLeapYear(i) ? 366 : 365;
    }
    for (int i = 0; i < month - 1; ++i) {
        if (i == 1) {
            days += isLeapYear(year) ? 29 : daysOfMonth[i];
        } else {
            days += daysOfMonth[i];
        }
    }
    days += day + 4;
    return days;
}
};

```

## 1227. Airplane Seat Assignment Probability

### Description

$n$  passengers board an airplane with exactly  $n$  seats. The first passenger has lost the ticket and picks a seat randomly. But after that, the rest of passengers will:

Take their own seat if it is still available,  
 Pick other seats randomly when they find their seat occupied  
 What is the probability that the  $n$ -th person can get his own seat?

Example 1:

Input:  $n = 1$

Output: 1.00000

Explanation: The first person can only get the first seat.

Example 2:

Input:  $n = 2$

Output: 0.50000

Explanation: The second person has a probability of 0.5 to get the second seat (when first person gets the first seat).

Constraints:

$1 \leq n \leq 10^5$

*Solution*

06/17/2020:

```
class Solution {  
public:  
    double nthPersonGetsNthSeat(int n) {  
        return n == 1 ? 1 : 1.0 / 2;  
    }  
};
```

## 1228. Missing Number In Arithmetic Progression

*Description*

In some array `arr`, the values were in arithmetic progression: the values `arr[i+1] - arr[i]` are all equal for every  $0 \leq i < arr.length - 1$ .

Then, a value from `arr` was removed that was not the first or last value in the array.

Return the removed value.

Example 1:

Input: `arr = [5,7,11,13]`

Output: 9

Explanation: The previous array was `[5,7,9,11,13]`.

Example 2:

Input: `arr = [15,13,12]`

Output: 14

Explanation: The previous array was `[15,14,13,12]`.

Constraints:

```
3 <= arr.length <= 1000
0 <= arr[i] <= 10^5
```

*Solution*

06/16/2020:

```
class Solution {
public:
    int missingNumber(vector<int>& arr) {
        adjacent_difference(arr.begin(), arr.end(), arr.begin());
        for (int i = 1; i < (int)arr.size() - 1; ++i)
            if (abs(arr[i]) > abs(arr[i + 1]))
                return arr[0] + i * arr[i + 1];
        return arr[0] + (arr.size() - 1) * arr[1];
    }
};
```

## 1232. Check If It Is a Straight Line

*Description*

You are given an array coordinates,  $\text{coordinates}[i] = [x, y]$ , where  $[x, y]$  represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:

Input: coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

Output: true

Example 2:

Input: coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]

Output: false

Constraints:

```
2 <= coordinates.length <= 1000
coordinates[i].length == 2
-10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4
coordinates contains no duplicate point.
```

*Solution*

05/07/2020:

```
class Solution {
public:
    bool checkStraightLine(vector<vector<int>>& coordinates) {
        double x1 = coordinates[0][0], x2 = coordinates[1][0];
        double y1 = coordinates[0][1], y2 = coordinates[1][1];
        double slope = (y1 - y2) / (x1 - x2);
        int n = coordinates.size();
        for (int i = 1; i < n - 1; ++i){
            x1 = coordinates[i][0], x2 = coordinates[i + 1][0];
            y1 = coordinates[i][1], y2 = coordinates[i + 1][1];
            double newSlope = (y1 - y2) / (x1 - x2);
            if (slope != newSlope) return false;
        }
        return true;
    }
};
```

## 1237. Find Positive Integer Solution for a Given Equation

*Description*

Given a function  $f(x, y)$  and a value  $z$ , return all positive integer pairs  $x$  and  $y$  where  $f(x, y) == z$ .

The function is constantly increasing, i.e.:

$$f(x, y) < f(x + 1, y)$$

$$f(x, y) < f(x, y + 1)$$

The function interface is defined like this:

```
interface CustomFunction {
public:
    // Returns positive integer f(x, y) for any given positive integer x and y.
    int f(int x, int y);
};
```

For custom testing purposes you're given an integer `function_id` and a target `z` as input, where `function_id` represent one function from an secret internal list, on the examples you'll know only two functions from the list.

You may return the solutions in any order.

Example 1:

Input: `function_id = 1, z = 5`

Output: `[[1,4],[2,3],[3,2],[4,1]]`

Explanation: `function_id = 1` means that  $f(x, y) = x + y$

Example 2:

Input: `function_id = 2, z = 5`

Output: `[[1,5],[5,1]]`

Explanation: `function_id = 2` means that  $f(x, y) = x * y$

Constraints:

`1 <= function_id <= 9`

`1 <= z <= 100`

It's guaranteed that the solutions of  $f(x, y) == z$  will be on the range  $1 <= x, y <= 1000$

It's also guaranteed that  $f(x, y)$  will fit in 32 bit signed integer if  $1 <= x, y <= 1000$

*Solution*

01/30/2020:

```
/*
 * // This is the custom function interface.
 * // You should not implement it, or speculate about its implementation
 * class CustomFunction {
 * public:
 *     // Returns f(x, y) for any given positive integers x and y.
 *     // Note that f(x, y) is increasing with respect to both x and y.
 *     // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
 *     int f(int x, int y);
 * };
 */

class Solution {
public:
    vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {
```

```

vector<vector<int>> ret;
for (int x = 1, y = 1000; x <= 1000 && y > 0;) {
    if (customfunction.f(x, y) > z) {
        --y;
    } else if (customfunction.f(x, y) < z) {
        ++x;
    } else {
        ret.push_back({x, y});
        ++x;
    }
}
return ret;
}
};

```

```

/*
 * // This is the custom function interface.
 * // You should not implement it, or speculate about its implementation
 * class CustomFunction {
 * public:
 *     // Returns f(x, y) for any given positive integers x and y.
 *     // Note that f(x, y) is increasing with respect to both x and y.
 *     // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
 *     int f(int x, int y);
 * };
 */

class Solution {
public:
    vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {
        vector<vector<int>> ret;
        for (int x = 1; x <= 1000; ++x) {
            if (customfunction.f(x, 1) > z) break;
            for (int y = 1; y <= 1000; ++y) {
                if (customfunction.f(x, y) == z) {
                    ret.push_back({x, y});
                } else if (customfunction.f(x, y) > z) {
                    break;
                }
            }
        }
        return ret;
    }
};

```

# 1281. Subtract the Product and Sum of Digits of an Integer

## Description

Given an integer number  $n$ , return the difference between the product of its digits and the sum of its digits.

Example 1:

Input:  $n = 234$

Output: 15

Explanation:

Product of digits =  $2 * 3 * 4 = 24$

Sum of digits =  $2 + 3 + 4 = 9$

Result =  $24 - 9 = 15$

Example 2:

Input:  $n = 4421$

Output: 21

Explanation:

Product of digits =  $4 * 4 * 2 * 1 = 32$

Sum of digits =  $4 + 4 + 2 + 1 = 11$

Result =  $32 - 11 = 21$

Constraints:

$1 \leq n \leq 10^5$

## Solution

01/29/2020:

```
class Solution {
public:
    int subtractProductAndSum(int n) {
        int ret = 1;
        for (int m = n; m != 0; m /= 10) {
            ret *= m % 10;
        }
        for (int m = n; m != 0; m /= 10) {
            ret -= m % 10;
        }
        return ret;
    }
};
```

# 1295. Find Numbers with Even Number of Digits

## Description

Given an array `nums` of integers, return how many of them contain an even number of digits.

Example 1:

Input: `nums = [12,345,2,6,7896]`

Output: 2

Explanation:

12 contains 2 digits (even number of digits).

345 contains 3 digits (odd number of digits).

2 contains 1 digit (odd number of digits).

6 contains 1 digit (odd number of digits).

7896 contains 4 digits (even number of digits).

Therefore only 12 and 7896 contain an even number of digits.

Example 2:

Input: `nums = [555,901,482,1771]`

Output: 1

Explanation:

Only 1771 contains an even number of digits.

Constraints:

$1 \leq \text{nums.length} \leq 500$

$1 \leq \text{nums}[i] \leq 10^5$

## Solution

01/29/2020:

```
class Solution {
public:
    int findNumbers(vector<int>& nums) {
        int ret = 0;
        for (auto& n : nums) {
            ret += int(ceil(log10(n + 1))) % 2 == 0 ? 1 : 0;
        }
        return ret;
    }
};
```



## 1323. Maximum 69 Number

### Description

Given a positive integer num consisting only of digits 6 and 9.

Return the maximum number you can get by changing at most one digit (6 becomes 9, and 9 becomes 6).

Example 1:

Input: num = 9669

Output: 9969

Explanation:

Changing the first digit results in 6669.

Changing the second digit results in 9969.

Changing the third digit results in 9699.

Changing the fourth digit results in 9666.

The maximum number is 9969.

Example 2:

Input: num = 9996

Output: 9999

Explanation: Changing the last digit 6 to 9 results in the maximum number.

Example 3:

Input: num = 9999

Output: 9999

Explanation: It is better not to apply any change.

Constraints:

$1 \leq \text{num} \leq 10^4$

num's digits are 6 or 9.

### Solution

01/19/2020:

```
class Solution {
public:
    int maximum69Number (int num) {
        string s = to_string(num);
        for (auto& c : s) {
            if (c == '6') {
                c = '9';
                break;
            }
        }
        return stoi(s);
    }
};
```

# String

## 20. Valid Parentheses

### *Description*

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Example 1:

Input: "()"

Output: true

Example 2:

Input: "()[]{}"

Output: true

Example 3:

Input: "[]"

Output: false

Example 4:

Input: "([])"  
Output: false  
Example 5:

Input: "{[]}"  
Output: true

*Solution*

01/25/2020:

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> p, q;
        for (auto it = s.begin(); it != s.end(); ++it) {
            if (*it == '(' || *it == '{' || *it == '[') {
                p.push(*it);
            } else if (p.size() > 0) {
                switch (*it) {
                    case ')': if (p.top() == '(') p.pop(); break;
                    case ']': if (p.top() == '[') p.pop(); break;
                    case '}': if (p.top() == '{') p.pop(); break;
                }
            }
        }
        for (auto it = s.rbegin(); it != s.rend(); ++it) {
            if (*it == ')' || *it == '}' || *it == ']') {
                q.push(*it);
            } else if (q.size() > 0) {
                switch (*it) {
                    case '(': if (q.top() == ')') q.pop(); break;
                    case '[': if (q.top() == ']') q.pop(); break;
                    case '{': if (q.top() == '}') q.pop(); break;
                }
            }
        }
        return p.size() == 0 && q.size() == 0;
    }
};
```

## 58. Length of Last Word

*Description*

Given a string `s` consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word (last word means the last appearing word if we loop from left to right) in the string.

If the last word does not exist, return 0.

Note: A word is defined as a maximal substring consisting of non-space characters only.

Example:

Input: "Hello World"

Output: 5

*Solution*

05/27/2020:

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        if (s.empty()) return 0;
        string ss, last;
        istringstream iss(s);
        while (getline(iss, ss, ' '))
            if (!ss.empty())
                last = ss;
        return last.size();
    }
};
```

## 71. Simplify Path

*Description*

Given an absolute path for a file (Unix-style), simplify it. Or in other words, convert it to the canonical path.

In a UNIX-style file system, a period `.` refers to the current directory. Furthermore, a double period `..` moves the directory up a level.

Note that the returned canonical path must always begin with a slash `/`, and there must be only a single slash `/` between two directory names. The last directory name (if it exists) must not end with a trailing `/`. Also, the canonical path must be the shortest string representing the absolute path.

Example 1:

Input: `"/home/"`

Output: `"/home"`

Explanation: Note that there is no trailing slash after the last directory name.

Example 2:

Input: `"/../"`

Output: `"/"`

Explanation: Going one level up from the root directory is a no-op, as the root level is the highest level you can go.

Example 3:

Input: `"/home//foo/"`

Output: `"/home/foo"`

Explanation: In the canonical path, multiple consecutive slashes are replaced by a single one.

Example 4:

Input: `"/a./b/././c/"`

Output: `"/c"`

Example 5:

Input: `"/a/././b/./c//./"`

Output: `"/c"`

Example 6:

Input: `"/a//b///c/d//././.."`

Output: `"/a/b/c"`

*Solution*

05/27/2020:

```
class Solution {
public:
    string simplifyPath(string path) {
        stack<string> st;
        istringstream iss(path);
        string s;
        while (getline(iss, s, '/')) {
            if (s == "..") {
                if (!st.empty())
                    st.pop();
            } else if (s.size() > 0 && s != ".") {
```

```
        st.push(s);
    }
}
string ret;
while (!st.empty()) {
    ret = "/" + st.top() + ret;
    st.pop();
}
return ret.empty() ? "/" : ret;
}
};
```

## 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*

02/03/2020:

```

class Solution {
public:
    void reverseString(vector<char>& s) {
        for (int l = 0, r = (int)s.size() - 1; l < r; ++l, --r) {
            swap(s[l], s[r]);
        }
    }
};

```

## 383. Ransom Note

### *Description*

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

#### Note:

You may assume that both strings contain only lowercase letters.

```

canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true

```

### *Solution*

05/04/2020:

```

class Solution {
public:
    bool canConstruct(string ransomNote, string magazine) {
        unordered_map<char, int> mp;
        for (auto& c : magazine) ++mp[c];
        for (auto& c : ransomNote)
            if (--mp[c] < 0) return false;
        return true;
    }
};

```

## 500. Keyboard Row

## Description

Given a List of words, return the words that can be typed using letters of alphabet on only one row's of American keyboard like the image below.

Example:

Input: ["Hello", "Alaska", "Dad", "Peace"]

Output: ["Alaska", "Dad"]

Note:

You may use one character in the keyboard more than once.

You may assume the input string will only contain letters of alphabet.

## Solution

02/03/2020:

```
class Solution {
public:
    vector<int> row{1,2,2,1,0,1,1,1,0,1,1,1,2,2,0,0,0,0,1,0,0,2,0,2,0,2};
    vector<string> findWords(vector<string>& words) {
        vector<string> ret;
        for (auto& word : words) {
            bool valid = true;
            for (int i = 1; i < (int)word.size(); ++i) {
                if (row[tolower(word[i]) - 'a'] != row[tolower(word[i - 1]) - 'a']) {
                    valid = false;
                    break;
                }
            }
            if (valid) ret.push_back(word);
        }
        return ret;
    }
};
```



## Description

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1:

Input: "Let's take LeetCode contest"

Output: "s'teL ekat edoCteeL tsetnoc"

Note: In the string, each word is separated by single space and there will not be any extra space in the string.

## Solution

01/31/2020:

```
class Solution {
public:
    string reverseWords(string s) {
        string ret;
        vector<string> t;
        for (int i = 0; i < (int)s.size(); ++i) {
            int j = i;
            for (; j < (int)s.size() && s[j] != ' '; ++j);
            t.push_back(s.substr(i, j - i));
            i = j;
        }
        for (auto& tt : t) {
            reverse(tt.begin(), tt.end());
            ret += tt + " ";
        }
        ret.pop_back();
        return ret;
    }
};
```

## 599. Minimum Index Sum of Two Lists

### Description

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their common interest with the least list index sum. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

Example 1:

Input:

```
["Shogun", "Tapioca Express", "Burger King", "KFC"]
```

```
["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
```

Output: ["Shogun"]

Explanation: The only restaurant they both like is "Shogun".

Example 2:

Input:

```
["Shogun", "Tapioca Express", "Burger King", "KFC"]
```

```
["KFC", "Shogun", "Burger King"]
```

Output: ["Shogun"]

Explanation: The restaurant they both like and have the least index sum is "Shogun" with index sum 1 (0+1).

Note:

The length of both lists will be in the range of [1, 1000].

The length of strings in both lists will be in the range of [1, 30].

The index is starting from 0 to the list length minus 1.

No duplicates in both lists.

*Solution*

05/20/2020:

```
class Solution {
public:
    vector<string> findRestaurant(vector<string>& list1, vector<string>& list2) {
        if (list1.empty() || list2.empty()) return {};
        unordered_map<string, vector<int>> index;
        for (int i = 0; i < (int)list1.size(); ++i) index[list1[i]].push_back(i);
        for (int i = 0; i < (int)list2.size(); ++i) index[list2[i]].push_back(i);
        int min_index_sum = INT_MAX;
        for (auto& i : index) {
            if ((int)i.second.size() != 2) continue;
            min_index_sum = min(min_index_sum, i.second.front() + i.second.back());
        }
        vector<string> ret;
        for(auto& i : index) {
            if ((int)i.second.size() != 2) continue;
            if (i.second.front() + i.second.back() == min_index_sum) {
                ret.push_back(i.first);
            }
        }
        return ret;
    }
}
```

```
};
```

## 657. Robot Return to Origin

### Description

There is a robot starting at position  $(0, 0)$ , the origin, on a 2D plane. Given a sequence of its moves, judge if this robot ends up at  $(0, 0)$  after it completes its moves.

The move sequence is represented by a string, and the character `moves[i]` represents its *i*th move. Valid moves are R (right), L (left), U (up), and D (down). If the robot returns to the origin after it finishes all of its moves, return true. Otherwise, return false.

Note: The way that the robot is "facing" is irrelevant. "R" will always make the robot move to the right once, "L" will always make it move left, etc. Also, assume that the magnitude of the robot's movement is the same for each move.

Example 1:

Input: "UD"

Output: true

Explanation: The robot moves up once, and then down once. All moves have the same magnitude, so it ended up at the origin where it started. Therefore, we return true.

Example 2:

Input: "LL"

Output: false

Explanation: The robot moves left twice. It ends up two "moves" to the left of the origin. We return false because it is not at the origin at the end of its moves.

### Solution

01/30/2020:

```
class Solution {
public:
    bool judgeCircle(string moves) {
        int h = 0, v = 0;
        for (auto& m : moves) {
```

```
    switch (m) {
        case 'L': --h; break;
        case 'R': ++h; break;
        case 'D': --v; break;
        case 'U': ++v; break;
    }
}
return h == 0 && v == 0;
}
};
```

## 709. To Lower Case

### *Description*

Implement function ToLowerCase() that has a string parameter str, and returns the same string in lowercase.

Example 1:

Input: "Hello"

Output: "hello"

Example 2:

Input: "here"

Output: "here"

Example 3:

Input: "LOVELY"

Output: "lovely"

### *Solution*

01/29/2020:

```
class Solution {
public:
    string toLowerCase(string str) {
        for (auto& c : str) {
            c = 'A' <= c && 'Z' >= c ? c + ('a' - 'A') : c;
        }
        return str;
    }
};
```

## 771. Jewels and Stones

### *Description*

You're given strings *J* representing the types of stones that are jewels, and *S* representing the stones you have. Each character in *S* is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in *J* are guaranteed distinct, and all characters in *J* and *S* are letters. Letters are case sensitive, so "a" is considered a different type of stone from "A".

Example 1:

Input: *J* = "aA", *S* = "aAAbbbb"

Output: 3

Example 2:

Input: *J* = "z", *S* = "ZZ"

Output: 0

Note:

*S* and *J* will consist of letters and have length at most 50.

The characters in *J* are distinct.

### *Solution*

01/29/2020:

```

class Solution {
public:
    int numJewelsInStones(string J, string S) {
        int ret = 0, chars[128] = {0};
        for (auto& j : J) {
            ++chars[j];
        }
        for (auto& s : S) {
            ret += chars[s];
        }
        return ret;
    }
};

```

## 796. Rotate String

### *Description*

We are given two strings, A and B.

A shift on A consists of taking string A and moving the leftmost character to the rightmost position. For example, if A = 'abcde', then it will be 'bcdea' after one shift on A. Return True if and only if A can become B after some number of shifts on A.

Example 1:

Input: A = 'abcde', B = 'cdeab'

Output: true

Example 2:

Input: A = 'abcde', B = 'abcd'

Output: false

Note:

A and B will have length at most 100.

### *Solution*

06/16/2020:

```

class Solution {
public:
    bool rotateString(string A, string B) {
        if (A.size() != B.size()) return false;
        if (A.empty()) return true;
    }
};

```

```

int n = A.size();
for (int i = 0; i < n; ++i) {
    bool isShift = true;
    while (i < n && A[i] != B[i]) ++i;
    for (int j = 0; j < n && isShift; ++j)
        if (A[j] != B[(i + j) % n])
            isShift = false;
    if (isShift) return true;
}
return false;
}
};

```

## 804. Unique Morse Code Words

### Description

International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows: "a" maps to ".-", "b" maps to "-...", "c" maps to "-.-.", and so on.

For convenience, the full table for the 26 letters of the English alphabet is given below:

```

[".-","-...","-.-.", "-..", ".","...-","--.", "...", ".-", ".---", "-.-", ".-..", "--",
 "-.", "----", ".--.", "-.-.", ".-.", "...", "-.", ".-.-", ".---", ".--", "-..-", ".-.-", "-.."]

```

Now, given a list of words, each word can be written as a concatenation of the Morse code of each letter. For example, "cba" can be written as "-.-..---..", (which is the concatenation "-.-." + "-..." + ".-"). We'll call such a concatenation, the transformation of a word.

Return the number of different transformations among all words we have.

Example:

Input: words = ["gin", "zen", "gig", "msg"]

Output: 2

Explanation:

The transformation of each word is:

"gin" -> "--...-."

"zen" -> "--...-."

"gig" -> "--...--."

"msg" -> "--...--."

There are 2 different transformations, "--...-." and "--...--."

Note:





```
widths =
[10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]
S = "abcdefghijklmnopqrstuvwxy"
Output: [3, 60]
```

Explanation:

All letters have the same length of 10. To write all 26 letters, we need two full lines and one line with 60 units.

Example :

Input:

```
widths =
[4,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]
S = "bbbccdddaaa"
Output: [2, 4]
```

Explanation:

All letters except 'a' have the same length of 10, and "bbbccdddaa" will cover  $9 * 10 + 2 * 4 = 98$  units.

For the last 'a', it is written on the second line because there is only 2 units left in the first line.

So the answer is 2 lines, plus 4 units in the second line.

Note:

The length of S will be in the range [1, 1000].

S will only contain lowercase letters.

widths is an array of length 26.

widths[i] will be in the range of [2, 10].

*Solution*

02/03/2020:

```
class Solution {
public:
    vector<int> numberOfLines(vector<int>& widths, string S) {
        int lines = S.size() > 0 ? 1 : 0, cur = 0;
        for (auto& c : S) {
            if (cur + widths[c - 'a'] > 100) {
                ++lines;
                cur = widths[c - 'a'];
            } else {
                cur += widths[c - 'a'];
            }
        }
        return {lines, cur};
    }
};
```

## 821. Shortest Distance to a Character

### Description

Given a string *S* and a character *C*, return an array of integers representing the shortest distance from the character *C* in the string.

Example 1:

Input: *S* = "loveleetcode", *C* = 'e'

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

Note:

*S* string length is in [1, 10000].

*C* is a single character, and guaranteed to be in string *S*.

All letters in *S* and *C* are lowercase.

### Solution

02/03/2020:

```
class Solution {
public:
    vector<int> shortestToChar(string S, char C) {
        int n = S.size();
        vector<int> loc, ret(n, INT_MAX);
        for (int i = 0; i < n; ++i) {
            if (S[i] == C) {
                loc.push_back(i);
            }
        }
        for (int j = 0; j < loc.front(); ++j) ret[j] = loc.front() - j;
        for (int j = loc.back(); j < n; ++j) ret[j] = j - loc.back();
        for (int i = 1; i < loc.size(); ++i) {
            for (int j = loc[i - 1]; j <= loc[i]; ++j) {
                ret[j] = min(j - loc[i - 1], loc[i] - j);
            }
        }
        return ret;
    }
};
```

## 893. Groups of Special-Equivalent Strings

### Description

You are given an array  $A$  of strings.

A move onto  $S$  consists of swapping any two even indexed characters of  $S$ , or any two odd indexed characters of  $S$ .

Two strings  $S$  and  $T$  are special-equivalent if after any number of moves onto  $S$ ,  $S == T$ .

For example,  $S = "zzxy"$  and  $T = "xyzz"$  are special-equivalent because we may make the moves  $"zzxy" \rightarrow "xzyz" \rightarrow "xyzz"$  that swap  $S[0]$  and  $S[2]$ , then  $S[1]$  and  $S[3]$ .

Now, a group of special-equivalent strings from  $A$  is a non-empty subset of  $A$  such that:

Every pair of strings in the group are special equivalent, and;

The group is the largest size possible (ie., there isn't a string  $S$  not in the group such that  $S$  is special equivalent to every string in the group)

Return the number of groups of special-equivalent strings from  $A$ .

Example 1:

Input: ["abcd","cdab","cbad","xyzz","zzxy","zzyx"]

Output: 3

Explanation:

One group is ["abcd", "cdab", "cbad"], since they are all pairwise special equivalent, and none of the other strings are all pairwise special equivalent to these.

The other two groups are ["xyzz", "zzxy"] and ["zzyx"]. Note that in particular, "zzxy" is not special equivalent to "zzyx".

Example 2:

Input: ["abc","acb","bac","bca","cab","cba"]

Output: 3

Note:

$1 \leq A.length \leq 1000$

$1 \leq A[i].length \leq 20$

All  $A[i]$  have the same length.

All  $A[i]$  consist of only lowercase letters.

Solution

02/03/2020:

```
class Solution {
public:
    int numSpecialEquivGroups(vector<string>& A) {
        unordered_set<string> us;
        for (auto& s : A) {
            vector<int> cnt(52, 0);
            for (int i = 0; i < (int)s.size(); ++i) {
                if (i % 2 == 0) {
                    ++cnt[s[i] - 'a'];
                } else {
                    ++cnt[s[i] - 'a' + 26];
                }
            }
            string ss;
            for (auto& n : cnt) {
                if (n < 10) {
                    ss += "0" + to_string(n);
                } else {
                    ss += to_string(n);
                }
            }
            us.insert(ss);
        }
        return us.size();
    }
};
```

## 929. Unique Email Addresses

*Description*

Every email consists of a local name and a domain name, separated by the @ sign.

For example, in alice@leetcode.com, alice is the local name, and leetcode.com is the domain name.

Besides lowercase letters, these emails may contain '.'s or '+'s.

If you add periods ('.') between some characters in the local name part of an email address, mail sent there will be forwarded to the same address without dots in the local name. For example, "alice.z@leetcode.com" and "alicez@leetcode.com" forward to the same email address. (Note that this rule does not apply for domain names.)

If you add a plus ('+') in the local name, everything after the first plus sign will be ignored. This allows certain emails to be filtered, for example m.y+name@email.com will be forwarded to my@email.com. (Again, this rule does not apply for domain names.)

It is possible to use both of these rules at the same time.

Given a list of emails, we send one email to each address in the list. How many different addresses actually receive mails?

Example 1:

Input:

```
["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","testemail+  
david@lee.tcode.com"]
```

Output: 2

Explanation: "testemail@leetcode.com" and "testemail@lee.tcode.com" actually receive mails

Note:

1 <= emails[i].length <= 100

1 <= emails.length <= 100

Each emails[i] contains exactly one '@' character.

All local and domain names are non-empty.

Local names do not start with a '+' character.

*Solution*

01/30/2020:

```
class Solution {  
public:  
    int numUniqueEmails(vector<string>& emails) {  
        unordered_set<string> unique_emails;  
        for (auto& email : emails) {  
            cout << email << " " << processEmail(email) << endl;  
            unique_emails.insert(processEmail(email));  
        }  
    }  
};
```

```

    return unique_emails.size();
}

string processEmail(string& email) {
    string ret;
    bool isLocal = true, doNotIgnore = true;
    for (int i = 0; i < email.size(); ++i) {
        if (isLocal) {
            if (doNotIgnore) {
                if (email[i] == '+') {
                    doNotIgnore = false;
                } else if (email[i] == '.') {
                    continue;
                } else if (email[i] == '@') {
                    ret.push_back(email[i]);
                    isLocal = false;
                    doNotIgnore = true;
                } else {
                    ret.push_back(email[i]);
                }
            } else {
                if (email[i] == '@') {
                    ret.push_back(email[i]);
                    isLocal = false;
                    doNotIgnore = true;
                }
            }
        } else {
            ret.push_back(email[i]);
        }
    }
    return ret;
}
};

```

## 953. Verifying an Alien Dictionary

### *Description*

In an alien language, surprisingly they also use english lowercase letters, but possibly in a different order. The order of the alphabet is some permutation of lowercase letters.

Given a sequence of words written in the alien language, and the order of the alphabet, return true if and only if the given words are sorted lexicographically in this alien language.

Example 1:

Input: words = ["hello","leetcode"], order = "hlabcdefgijklmnopqrstuvwxyz"

Output: true

Explanation: As 'h' comes before 'l' in this language, then the sequence is sorted.

Example 2:

Input: words = ["word","world","row"], order = "worldabcefghijklmnopqstuvwxyz"

Output: false

Explanation: As 'd' comes after 'l' in this language, then words[0] > words[1], hence the sequence is unsorted.

Example 3:

Input: words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"

Output: false

Explanation: The first three characters "app" match, and the second string is shorter (in size.) According to lexicographical rules "apple" > "app", because 'l' > 'ø', where 'ø' is defined as the blank character which is less than any other character (More info).

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 20

order.length == 26

All characters in words[i] and order are English lowercase letters.

*Solution*

05/18/2020:

```
class Solution {
public:
    bool isAlienSorted(vector<string>& words, string order) {
        vector<int> o(26, 0);
        for (int i = 0; i < (int)order.size(); ++i) {
            o[order[i] - 'a'] = i;
        }
        for (int i = 1; i < (int)words.size(); ++i) {
            int n = min(words[i].size(), words[i - 1].size());
            bool all = true;
            for (int j = 0; j < n; ++j) {
                if (o[words[i][j] - 'a'] > o[words[i - 1][j] - 'a']) {
```

```

        all = false;
        break;
    } else if (o[words[i][j] - 'a'] < o[words[i - 1][j] - 'a']) {
        return false;
    }
}
if (all && n == words[i].size() && words[i].size() < words[i - 1].size())
return false;
}
return true;
}
};

```

## 1002. Find Common Characters

### *Description*

Given an array A of strings made only from lowercase letters, return a list of all characters that show up in all strings within the list (including duplicates). For example, if a character occurs 3 times in all strings but not 4 times, you need to include that character three times in the final answer.

You may return the answer in any order.

Example 1:

Input: ["bella","label","roller"]

Output: ["e","l","l"]

Example 2:

Input: ["cool","lock","cook"]

Output: ["c","o"]

Note:

1 <= A.length <= 100

1 <= A[i].length <= 100

A[i][j] is a lowercase letter

### *Solution*

02/03/2020:

---



```

class Solution {
public:
    vector<string> commonChars(vector<string>& A) {
        int n = A.size();
        vector<vector<int>> cnt(n, vector<int>(26, 0));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < (int)A[i].size(); ++j) {
                ++cnt[i][A[i][j] - 'a'];
            }
        }
        vector<string> ret;
        for (int i = 0; i < 26; ++i) {
            int m = INT_MAX;
            for (int j = 0; j < n; ++j) {
                m = min(m, cnt[j][i]);
            }
            for (; m > 0; --m) {
                ret.push_back(string(1, 'a' + i));
            }
        }
        return ret;
    }
};

```

## 1021. Remove Outermost Parentheses

### Description

A valid parentheses string is either empty (`""`), `"(" + A + ")"`, or `A + B`, where `A` and `B` are valid parentheses strings, and `+` represents string concatenation. For example, `""`, `"()"`, `"(())()"`, and `"(()(()))"` are all valid parentheses strings.

A valid parentheses string `S` is primitive if it is nonempty, and there does not exist a way to split it into `S = A+B`, with `A` and `B` nonempty valid parentheses strings.

Given a valid parentheses string `S`, consider its primitive decomposition: `S = P1 + P2 + ... + Pk`, where `Pi` are primitive valid parentheses strings.

Return `S` after removing the outermost parentheses of every primitive string in the primitive decomposition of `S`.

Example 1:

Input: "(()())(())"

Output: "()()()"

Explanation:

The input string is "(()())(())", with primitive decomposition "(()())" + "()()".

After removing outer parentheses of each part, this is "()()()".

Example 2:

Input: "(()())(())(())(())"

Output: "()()()()()()"

Explanation:

The input string is "(()())(())(())(())", with primitive decomposition "(()())" + "()" + "(()())" + "(()())".

After removing outer parentheses of each part, this is "()()()()()()".

Example 3:

Input: "()()"

Output: ""

Explanation:

The input string is "()()", with primitive decomposition "()" + "()".

After removing outer parentheses of each part, this is "" + "" = "".

Note:

S.length <= 10000

S[i] is "(" or ")"

S is a valid parentheses string

*Solution*

01/29/2020:

```
class Solution {
public:
    string removeOuterParentheses(string S) {
        string ret;
        int status = 0;
        for (auto& c : S) {
            if (c == '(') {
                if (status++ > 0) {
                    ret += c;
                }
            } else {
                if (--status > 0) {
                    ret += c;
                }
            }
        }
    }
};
```

```
        ret += c;
    }
}
}
return ret;
}
};
```

## 1044. Longest Duplicate Substring

### Description

Given a string  $S$ , consider all duplicated substrings: (contiguous) substrings of  $S$  that occur 2 or more times. (The occurrences may overlap.)

Return any duplicated substring that has the longest possible length. (If  $S$  does not have a duplicated substring, the answer is `""`.)

Example 1:

Input: "banana"

Output: "ana"

Example 2:

Input: "abcd"

Output: ""

Note:

$2 \leq S.length \leq 10^5$

$S$  consists of lowercase English letters.

### Solution

06/19/2020:

```
class Solution {
public:
    string longestDupSubstring(string S) {
        std::string_view ret;
        std::unordered_set<std::string_view> set;
        size_t lo = 1;
```

```

size_t hi = S.size() - 1;
while (lo <= hi) {
    auto mid = lo + (hi - lo) / 2;
    bool ok = false;
    for (size_t i = 0; i != S.size() - mid + 1; ++i) {
        const auto [it, inserted] = set.emplace(S.data() + i, mid);
        if (!inserted) {
            ok = true;
            ret = *it;
            break;
        }
    }
    if (ok)
        lo = mid + 1;
    else
        hi = mid - 1;
    set.clear();
}
return {ret.begin(), ret.end()};
}
};

```

TLE:

```

class Solution {
public:
    string longestDupSubstring(string S) {
        const int MOD = 1e9 + 7;
        const int m = 31;
        int n = S.size();
        vector<long long> p(n + 1, 1);
        vector<long long> h(n + 1, 0);
        for (int i = 0; i < n; ++i) {
            p[i + 1] = (p[i] * m) % MOD;
            h[i + 1] = (h[i] + (S[i] - 'a' + 1) * p[i]) % MOD;
        }
        int lo = 0, hi = n, start = 0;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2; // length of the substring
            bool ok = false;
            for (int i = 0; i + mid <= n && !ok; ++i) {
                // pattern: S[i..i + mid]
                long long hp = (h[i + mid] - h[i] + MOD) % MOD;
                for (int j = i + 1; j + mid <= n && !ok; ++j) {
                    // substring: S[j..j + mid]
                    long long cur_h = (h[j + mid] - h[j] + MOD) % MOD;
                    long long new_h = (hp * p[j - i] + MOD) % MOD;
                    ok = cur_h == new_h;
                }
            }
            if (ok) start = i + mid;
            if (ok) lo = i + mid;
            else hi = i;
        }
        return S.substr(start, lo - start);
    }
};

```

```

        if (ok) start = i;
    }
}
if (ok)
    lo = mid + 1;
else
    hi = mid - 1;
}
bool exists = false;
long long hp = (h[start + lo] - h[start] + MOD) % MOD;
for (int j = start + 1; j + lo <= n && !exists; ++j) {
    long long cur_h = (h[j + lo] - h[j] + MOD) % MOD;
    long long new_h = (hp * p[j - start] + MOD) % MOD;
    exists = cur_h == new_h;
}
if (!exists) --lo;
return S.substr(start, lo);
}
};

```

## 1047. Remove All Adjacent Duplicates In String

### Description

Given a string *S* of lowercase letters, a duplicate removal consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on *S* until we no longer can.

Return the final string after all such duplicate removals have been made. It is guaranteed the answer is unique.

Example 1:

Input: "abbaca"

Output: "ca"

Explanation:

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Note:

```
1 <= S.length <= 20000
S consists only of English lowercase letters.
```

*Solution*

01/31/2020:

```
class Solution {
public:
    string removeDuplicates(string S) {
        string ret;
        for (auto& c : S) {
            if (ret.empty() || ret.back() != c) {
                ret.push_back(c);
            } else {
                ret.pop_back();
            }
        }
        return ret;
    }
};
```

## 1078. Occurrences After Bigram

*Description*

Given words `first` and `second`, consider occurrences in some text of the form "`first second third`", where `second` comes immediately after `first`, and `third` comes immediately after `second`.

For each such occurrence, add "`third`" to the answer, and return the answer.

Example 1:

Input: `text = "alice is a good girl she is a good student", first = "a", second = "good"`

Output: `["girl","student"]`

Example 2:

Input: `text = "we will we will rock you", first = "we", second = "will"`

Output: `["we","rock"]`

Note:

1 <= text.length <= 1000

text consists of space separated words, where each word consists of lowercase English letters.

1 <= first.length, second.length <= 10

first and second consist of lowercase English letters.

*Solution*

02/03/2020:

```
class Solution {
public:
    vector<string> findOccurrences(string text, string first, string second) {
        vector<string> words, ret;
        istringstream iss(text);
        string word;
        while (iss >> word) {
            words.push_back(word);
        }
        for (int i = 0; i < words.size() - 2; ++i) {
            if (words[i] == first && words[i + 1] == second) {
                ret.push_back(words[i + 2]);
            }
        }
        return ret;
    }
};
```

## 1108. Defanging an IP Address

*Description*

Given a valid (IPv4) IP address, return a defanged version of that IP address.

A defanged IP address replaces every period "." with "[.]".

Example 1:

Input: address = "1.1.1.1"

Output: "1[.]1[.]1[.]1"

Example 2:

Input: address = "255.100.50.0"  
Output: "255[.]100[.]50[.]0"

Constraints:

The given address is a valid IPv4 address.

*Solution*

01/29/2020:

```
class Solution {
public:
    string defangIPaddr(string address) {
        string ret;
        for (auto& c : address) {
            if (c == '.') {
                ret += "[.]";
            } else {
                ret += c;
            }
        }
        return ret;
    }
};
```

## 1119. Remove Vowels from a String

*Description*

Given a string *S*, remove the vowels 'a', 'e', 'i', 'o', and 'u' from it, and return the new string.

Example 1:

Input: "leetcodeisacommunityforcoders"

Output: "ltdscmmntyfrcdrs"

Example 2:

Input: "aeiou"

Output: ""



Note:

S consists of lowercase English letters only.

$1 \leq S.length \leq 1000$

*Solution*

01/29/2020:

```
class Solution {
public:
    string removeVowels(string S) {
        string ret;
        for (auto& c : S) {
            if (c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u') {
                ret.push_back(c);
            }
        }
        return ret;
    }
};
```

## 1160. Find Words That Can Be Formed by Characters

*Description*

You are given an array of strings words and a string chars.

A string is good if it can be formed by characters from chars (each character can only be used once).

Return the sum of lengths of all good strings in words.

Example 1:

Input: words = ["cat","bt","hat","tree"], chars = "atach"

Output: 6

Explanation:

The strings that can be formed are "cat" and "hat" so the answer is  $3 + 3 = 6$ .

Example 2:

Input: words = ["hello","world","leetcode"], chars = "welldonehoneyr"

Output: 10

Explanation:

The strings that can be formed are "hello" and "world" so the answer is 5 + 5 = 10.

Note:

1 <= words.length <= 1000

1 <= words[i].length, chars.length <= 100

All strings contain lowercase English letters only.

*Solution*

01/31/2020:

```
class Solution {
public:
    int countCharacters(vector<string>& words, string chars) {
        int ret = 0;
        vector<int> cnt(26, 0);
        for (auto& c : chars) {
            ++cnt[c - 'a'];
        }
        for (auto& w : words) {
            vector<int> req(26, 0);
            for (auto& c : w) {
                ++req[c - 'a'];
            }
            bool fit = true;
            for (int i = 0; i < 26; ++i) {
                if (req[i] > cnt[i]) {
                    fit = false;
                    break;
                }
            }
            if (fit) {
                ret += w.size();
            }
        }
        return ret;
    }
};
```

# 1165. Single-Row Keyboard

## Description

There is a special keyboard with all keys in a single row.

Given a string `keyboard` of length 26 indicating the layout of the keyboard (indexed from 0 to 25), initially your finger is at index 0. To type a character, you have to move your finger to the index of the desired character. The time taken to move your finger from index  $i$  to index  $j$  is  $|i - j|$ .

You want to type a string `word`. Write a function to calculate how much time it takes to type it with one finger.

Example 1:

Input: `keyboard = "abcdefghijklmnopqrstuvwxyz"`, `word = "cba"`

Output: 4

Explanation: The index moves from 0 to 2 to write 'c' then to 1 to write 'b' then to 0 again to write 'a'.

Total time = 2 + 1 + 1 = 4.

Example 2:

Input: `keyboard = "pqrstuvwxyzabcdefghijklmnop"`, `word = "leetcode"`

Output: 73

Constraints:

`keyboard.length == 26`

`keyboard` contains each English lowercase letter exactly once in some order.

`1 <= word.length <= 10^4`

`word[i]` is an English lowercase letter.

## Solution

01/29/2020:

```
class Solution {
public:
    int calculateTime(string keyboard, string word) {
        unordered_map<char, int> m;
        int last = 0, ret = 0;
        for (auto i = 0; i < (int)keyboard.size(); ++i) {
            m[keyboard[i]] = i;
        }
    }
};
```

```
    }  
    for (auto& c : word) {  
        ret += abs(m[c] - last);  
        last = m[c];  
    }  
    return ret;  
}  
};
```

## 1180. Count Substrings with Only One Distinct Letter

### *Description*

Given a string  $S$ , return the number of substrings that have only one distinct letter.

Example 1:

Input:  $S = \text{"aaaba"}$

Output: 8

Explanation: The substrings with one distinct letter are "aaa", "aa", "a", "b".

"aaa" occurs 1 time.

"aa" occurs 2 times.

"a" occurs 4 times.

"b" occurs 1 time.

So the answer is  $1 + 2 + 4 + 1 = 8$ .

Example 2:

Input:  $S = \text{"aaaaaaaaa"}$

Output: 55

Constraints:

$1 \leq S.length \leq 1000$

$S[i]$  consists of only lowercase English letters.

### *Solution*

01/30/2020:

```
class Solution {  
public:
```

```

int countLetters(string S) {
    int ret = 0;
    int start = 0;
    for (int i = 0; i < S.size() - 1; ++i) {
        if (S[i] != S[i + 1]) {
            ret += (i - start + 1) * (i - start + 2) / 2;
            start = i + 1;
        }
    }
    ret += (S.size() - start) * (S.size() - start + 1) / 2;
    return ret;
}
};

```

## 1221. Split a String in Balanced Strings

### *Description*

Balanced strings are those who have equal quantity of 'L' and 'R' characters.

Given a balanced string *s* split it in the maximum amount of balanced strings.

Return the maximum amount of splitted balanced strings.

Example 1:

Input: *s* = "RLRLLRLRL"

Output: 4

Explanation: *s* can be split into "RL", "RRLL", "RL", "RL", each substring contains same number of 'L' and 'R'.

Example 2:

Input: *s* = "RLLLLRRRLR"

Output: 3

Explanation: *s* can be split into "RL", "LLLRRR", "LR", each substring contains same number of 'L' and 'R'.

Example 3:

Input: *s* = "LLLLRRRR"

Output: 1

Explanation: *s* can be split into "LLLLRRRR".

Example 4:

Input: *s* = "RLRRRLRLRL"

Output: 2

Explanation: s can be split into "RL", "RRRLLRLL", since each substring contains an equal number of 'L' and 'R'

Constraints:

1 <= s.length <= 1000  
s[i] = 'L' or 'R'

*Solution*

01/29/2020:

```
class Solution {
public:
    int balancedStringSplit(string s) {
        int ret = 0, running_sum = 0;
        for (auto& c : s) {
            running_sum += c == 'R' ? 1 : -1;
            ret += running_sum == 0 ? 1 : 0;
        }
        return ret;
    }
};
```

## 1236. Web Crawler

*Description*

Given a url `startUrl` and an interface `HtmlParser`, implement a web crawler to crawl all links that are under the same hostname as `startUrl`.

Return all urls obtained by your web crawler in any order.

Your crawler should:

Start from the page: `startUrl`

Call `HtmlParser.getUrls(url)` to get all urls from a webpage of given url.

Do not crawl the same link twice.

Explore only the links that are under the same hostname as `startUrl`.

As shown in the example url above, the hostname is example.org. For simplicity sake, you may assume all urls use http protocol without any port specified. For example, the urls `http://leetcode.com/problems` and `http://leetcode.com/contest` are under the same hostname, while urls `http://example.org/test` and `http://example.com/abc` are not under the same hostname.

The `HtmlParser` interface is defined as such:

```
interface HtmlParser {
    // Return a list of all urls from a webpage of given url.
    public List<String> getUrls(String url);
}
```

Below are two examples explaining the functionality of the problem, for custom testing purposes you'll have three variables `urls`, `edges` and `startUrl`. Notice that you will only have access to `startUrl` in your code, while `urls` and `edges` are not directly accessible to you in code.

Example 1:

Input:

```
urls = [
    "http://news.yahoo.com",
    "http://news.yahoo.com/news",
    "http://news.yahoo.com/news/topics/",
    "http://news.google.com",
    "http://news.yahoo.com/us"
]
edges = [[2,0],[2,1],[3,2],[3,1],[0,4]]
startUrl = "http://news.yahoo.com/news/topics/"
```

Output: [

```
    "http://news.yahoo.com",
    "http://news.yahoo.com/news",
    "http://news.yahoo.com/news/topics/",
    "http://news.yahoo.com/us"
]
```

Example 2:

Input:

```
urls = [
    "http://news.yahoo.com",
    "http://news.yahoo.com/news",
    "http://news.yahoo.com/news/topics/",
    "http://news.google.com"
```

```
]
edges = [[0,2],[2,1],[3,2],[3,1],[3,0]]
startUrl = "http://news.google.com"
Output: ["http://news.google.com"]
Explanation: The startUrl links to all other pages that do not share the same
hostname.
```

Constraints:

```
1 <= urls.length <= 1000
1 <= urls[i].length <= 300
startUrl is one of the urls.
Hostname label must be from 1 to 63 characters long, including the dots, may
contain only the ASCII letters from 'a' to 'z', digits from '0' to '9' and the
hyphen-minus character ('-').
The hostname may not start or end with the hyphen-minus character ('-').
See: https://en.wikipedia.org/wiki/Hostname#Restrictions\_on\_valid\_hostnames
You may assume there're no duplicates in url library.
```

*Solution*

05/04/2020:

```
/**
 * // This is the HtmlParser's API interface.
 * // You should not implement it, or speculate about its implementation
 * class HtmlParser {
 *   public:
 *     vector<string> getUrls(string url);
 * };
 */

class Solution {
public:
    vector<string> crawl(string startUrl, HtmlParser htmlParser) {
        string hostname = startUrl.substr(0, startUrl.find('/', 7));
        unordered_set<string> visited;
        vector<string> ret;
        queue<string> q;
        q.push(startUrl);
        while (!q.empty()) {
            string url = q.front(); q.pop();
            if (url.find(hostname) == string::npos || visited.count(url) != 0)
                continue;
            visited.insert(url);
            ret.push_back(url);
            vector<string> urls = htmlParser.getUrls(url);
```



```
        for (auto& url : urls) q.push(url);
    }
    return ret;
}
};
```

## 1309. Decrypt String from Alphabet to Integer Mapping

### Description

Given a string  $s$  formed by digits ('0' - '9') and '#' . We want to map  $s$  to English lowercase characters as follows:

Characters ('a' to 'i') are represented by ('1' to '9') respectively.  
Characters ('j' to 'z') are represented by ('10#' to '26#') respectively.  
Return the string formed after mapping.

It's guaranteed that a unique mapping will always exist.

Example 1:

Input:  $s = "10\#11\#12"$

Output: "jkab"

Explanation: "j" -> "10#" , "k" -> "11#" , "a" -> "1" , "b" -> "2".

Example 2:

Input:  $s = "1326\#"$

Output: "acz"

Example 3:

Input:  $s = "25\#"$

Output: "y"

Example 4:

Input:  $s = "12345678910\#11\#12\#13\#14\#15\#16\#17\#18\#19\#20\#21\#22\#23\#24\#25\#26\#"$

Output: "abcdefghijklmnopqrstuvwxyz"

Constraints:

$1 \leq s.length \leq 1000$

$s[i]$  only contains digits letters ('0'-'9') and '#' letter.

$s$  will be valid string such that mapping is always possible.

Solution

01/29/2020:

```
class Solution {
public:
    string freqAlphabets(string s) {
        string ret;
        for (int i = s.size() - 1; i >= 0; i--) {
            if (s[i] == '#') {
                ret.push_back(stoi(s.substr(i - 2, 2)) - 1 + 'a');
                i -= 3;
            } else {
                ret.push_back(s[i] - '1' + 'a');
                --i;
            }
        }
        reverse(ret.begin(), ret.end());
        return ret;
    }
};
```

## 1324. Print Words Vertically

*Description*

Given a string *s*. Return all the words vertically in the same order in which they appear in *s*.  
Words are returned as a list of strings, complete with spaces when is necessary. (Trailing spaces are not allowed).  
Each word would be put on only one column and that in one column there will be only one word.

Example 1:

Input: *s* = "HOW ARE YOU"

Output: ["HAY","ORO","WEU"]

Explanation: Each word is printed vertically.

"HAY"

"ORO"

"WEU"

Example 2:

Input: *s* = "TO BE OR NOT TO BE"

```
Output: ["TBONTB","OEROOE"," T"]
Explanation: Trailing spaces is not allowed.
"TBONTB"
"OEROOE"
" T"
```

Example 3:

Input: s = "CONTEST IS COMING"

```
Output: ["CIC","OSO","N M","T I","E N","S G","T"]
```

Constraints:

1 <= s.length <= 200

s contains only upper case English letters.

It's guaranteed that there is only one space between 2 words.

*Solution*

01/19/2020:

```
class Solution {
public:
    vector<string> printVertically(string s) {
        istringstream is(s);
        string str;
        vector<string> words;
        int max_len = INT_MIN;
        while (is >> str) {
            words.push_back(str);
            max_len = max(max_len, (int)str.size());
        }
        vector<string> ret(max_len, string(words.size(), ' '));
        for (int i = 0; i < words.size(); ++i)
            for (int j = 0; j < ret.size(); ++j)
                if (j < words[i].size())
                    ret[j][i] = words[i][j];
        for (auto& r : ret)
            while (r.back() == ' ')
                r.pop_back();
        return ret;
    }
};
```

## 1332. Remove Palindromic Subsequences

## Description

Given a string  $s$  consisting only of letters 'a' and 'b'. In a single step you can remove one palindromic subsequence from  $s$ .

Return the minimum number of steps to make the given string empty.

A string is a subsequence of a given string, if it is generated by deleting some characters of a given string without changing its order.

A string is called palindrome if is one that reads the same backward as well as forward.

Example 1:

Input:  $s = \text{"ababa"}$

Output: 1

Explanation: String is already palindrome

Example 2:

Input:  $s = \text{"abb"}$

Output: 2

Explanation:  $\text{"abb"} \rightarrow \text{"bb"} \rightarrow \text{""}.$

Remove palindromic subsequence "a" then "bb".

Example 3:

Input:  $s = \text{"baabb"}$

Output: 2

Explanation:  $\text{"baabb"} \rightarrow \text{"b"} \rightarrow \text{""}.$

Remove palindromic subsequence "baab" then "b".

Example 4:

Input:  $s = \text{""}.$

Output: 0

Constraints:

$0 \leq s.length \leq 1000$

$s$  only consists of letters 'a' and 'b'

## Solution

01/25/2020:

```

class Solution {
public:
    int removePalindromeSub(string s) {
        if (s.size() == 0) return 0;
        for (int l = 0, r = s.size() - 1; l < r; ++l, --r)
            if (s[l] != s[r]) return 2;
        return 1;
    }
};

```

## 1427. Perform String Shifts

### Description

You are given a string  $s$  containing lowercase English letters, and a matrix  $shift$ , where  $shift[i] = [direction, amount]$ :

$direction$  can be  $0$  (for left shift) or  $1$  (for right shift).

$amount$  is the amount by which string  $s$  is to be shifted.

A left shift by  $1$  means remove the first character of  $s$  and append it to the end.

Similarly, a right shift by  $1$  means remove the last character of  $s$  and add it to the beginning.

Return the final string after all operations.

Example 1:

Input:  $s = "abc"$ ,  $shift = [[0,1],[1,2]]$

Output: "cab"

Explanation:

$[0,1]$  means shift to left by 1. "abc" → "bca"

$[1,2]$  means shift to right by 2. "bca" → "cab"

Example 2:

Input:  $s = "abcdefg"$ ,  $shift = [[1,1],[1,1],[0,2],[1,3]]$

Output: "efgabcd"

Explanation:

$[1,1]$  means shift to right by 1. "abcdefg" → "gabcdef"

$[1,1]$  means shift to right by 1. "gabcdef" → "fgabcde"

$[0,2]$  means shift to left by 2. "fgabcde" → "abcdefg"

$[1,3]$  means shift to right by 3. "abcdefg" → "efgabcd"

Constraints:

```
1 <= s.length <= 100
s only contains lower case English letters.
1 <= shift.length <= 100
shift[i].length == 2
0 <= shift[i][0] <= 1
0 <= shift[i][1] <= 100
```

*Solution*

05/04/2020:

```
class Solution {
public:
    string stringShift(string s, vector<vector<int>>& shift) {
        int m = 0, n = s.size();
        for (auto& h : shift) m += h[0] == 0 ? -h[1] : h[1];
        m = (m % n + n) % n;
        return s.substr(n - m, m) + s.substr(0, n - m);
    }
};
```

## 1432. Max Difference You Can Get From Changing an Integer

*Description*

You are given an integer `num`. You will apply the following steps exactly two times:

Pick a digit `x` ( $0 \leq x \leq 9$ ).

Pick another digit `y` ( $0 \leq y \leq 9$ ). The digit `y` can be equal to `x`.

Replace all the occurrences of `x` in the decimal representation of `num` by `y`.

The new integer cannot have any leading zeros, also the new integer cannot be `0`.

Let `a` and `b` be the results of applying the operations to `num` the first and second times, respectively.

Return the max difference between `a` and `b`.

Example 1:

Input: num = 555

Output: 888

Explanation: The first time pick  $x = 5$  and  $y = 9$  and store the new integer in a. The second time pick  $x = 5$  and  $y = 1$  and store the new integer in b.

We have now  $a = 999$  and  $b = 111$  and max difference = 888

Example 2:

Input: num = 9

Output: 8

Explanation: The first time pick  $x = 9$  and  $y = 9$  and store the new integer in a. The second time pick  $x = 9$  and  $y = 1$  and store the new integer in b.

We have now  $a = 9$  and  $b = 1$  and max difference = 8

Example 3:

Input: num = 123456

Output: 820000

Example 4:

Input: num = 10000

Output: 80000

Example 5:

Input: num = 9288

Output: 8700

Constraints:

$1 \leq \text{num} \leq 10^8$

*Solution*

05/02/2020:

```
class Solution {
public:
    int maxDiff(int num) {
        string s1 = to_string(num);
        string s2 = to_string(num);
        char c, t;
        bool found = false;
        int n = s1.size();
        for (int i = 0; i < n; ++i) {
            if (!found && s1[i] != '9') {
                c = s1[i];
                s1[i] = '9';
                found = true;
            } else if (found && s1[i] == c) {
```

```

        s1[i] = '9';
    }
}
found = false;
for (int i = 0; i < n; ++i) {
    if (!found) {
        if (s2[0] != '1') {
            c = s2[0];
            t = '1';
            s2[0] = '1';
            found = true;
        } else {
            if (i > 0 && s2[i] != s2[0] && s2[i] != '0') {
                c = s2[i];
                t = '0';
                s2[i] = '0';
                found = true;
            }
        }
    } else {
        if (s2[i] == c) {
            s2[i] = t;
        }
    }
}
return stoi(s1) - stoi(s2);
};

```

## 1433. Check If a String Can Break Another String

### Description

Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa (in other words s2 can break s1).

A string x can break string y (both of size n) if  $x[i] \geq y[i]$  (in alphabetical order) for all i between 0 and n-1.

Example 1:

Input: s1 = "abc", s2 = "xya"

Output: true



Explanation: "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc".

Example 2:

Input: s1 = "abe", s2 = "acd"

Output: false

Explanation: All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa.

Example 3:

Input: s1 = "leetcode", s2 = "interview"

Output: true

Constraints:

s1.length == n

s2.length == n

1 <= n <= 10<sup>5</sup>

All strings consist of lowercase English letters.

*Solution*

05/02/2020:

```
class Solution {
public:
    bool checkIfCanBreak(string s1, string s2) {
        sort(s1.begin(), s1.end());
        sort(s2.begin(), s2.end());
        int n = s1.size();
        bool greaterThan = true, lessThan = true;
        for (int i = 0; i < n; ++i) {
            if (s1[i] > s2[i]) {
                lessThan = false;
            }
            if (s1[i] < s2[i]) {
                greaterThan = false;
            }
        }
        return lessThan || greaterThan;
    }
};
```

# 1436. Destination City

## Description

You are given the array `paths`, where `paths[i] = [cityAi, cityBi]` means there exists a direct path going from `cityAi` to `cityBi`. Return the destination city, that is, the city without any path outgoing to another city.

It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city.

Example 1:

Input: `paths = [{"London","New York"}, {"New York","Lima"}, {"Lima","Sao Paulo"}]`

Output: `"Sao Paulo"`

Explanation: Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo".

Example 2:

Input: `paths = [{"B","C"}, {"D","B"}, {"C","A"}]`

Output: `"A"`

Explanation: All possible trips are:

`"D" -> "B" -> "C" -> "A"`.

`"B" -> "C" -> "A"`.

`"C" -> "A"`.

`"A"`.

Clearly the destination city is "A".

Example 3:

Input: `paths = [{"A","Z"}]`

Output: `"Z"`

Constraints:

`1 <= paths.length <= 100`

`paths[i].length == 2`

`1 <= cityAi.length, cityBi.length <= 10`

`cityAi != cityBi`

All strings consist of lowercase and uppercase English letters and the space character.

## Solution

05/02/2020:

```
class Solution {
public:
    string destCity(vector<vector<string>>& paths) {
        unordered_set<string> cities;
        unordered_set<string> departs;
        for (auto& p : paths) {
            cities.insert(p[0]);
            cities.insert(p[1]);
            departs.insert(p[0]);
        }
        for (auto& d : departs) {
            cities.erase(d);
        }
        string ret;
        for (auto& c : cities) {
            ret = c;
        }
        return ret;
    }
};
```

## 1446. Consecutive Characters

### *Description*

Given a string  $s$ , the power of the string is the maximum length of a non-empty substring that contains only one unique character.

Return the power of the string.

Example 1:

Input:  $s = \text{"leetcode"}$

Output: 2

Explanation: The substring "ee" is of length 2 with the character 'e' only.

Example 2:

Input:  $s = \text{"abbccdddeeeedcba"}$

Output: 5

Explanation: The substring "eeeee" is of length 5 with the character 'e' only.

Example 3:

Input: s = "triplepillloooooow"

Output: 5

Example 4:

Input: s = "hooraaaaaaaaaaaay"

Output: 11

Example 5:

Input: s = "tourist"

Output: 1

Constraints:

1 <= s.length <= 500

s contains only lowercase English letters.

*Solution*

05/17/2020:

```
class Solution {
public:
    int maxPower(string s) {
        if (s.empty()) return 0;
        int cnt = 1;
        int ret = 1;
        for (int i = 1; i < (int)s.size(); ++i) {
            if (s[i] == s[i - 1]) {
                ret = max(ret, ++cnt);
            } else {
                cnt = 1;
            }
        }
        return ret;
    }
};
```

## 1451. Rearrange Words in a Sentence

*Description*

Given a sentence text (A sentence is a string of space-separated words) in the following format:

First letter is in upper case.

Each word in text are separated by a single space.

Your task is to rearrange the words in text such that all words are rearranged in an increasing order of their lengths. If two words have the same length, arrange them in their original order.

Return the new text following the format shown above.

Example 1:

Input: text = "Leetcode is cool"

Output: "Is cool leetcode"

Explanation: There are 3 words, "Leetcode" of length 8, "is" of length 2 and "cool" of length 4.

Output is ordered by length and the new first word starts with capital letter.

Example 2:

Input: text = "Keep calm and code on"

Output: "On and keep calm code"

Explanation: Output is ordered as follows:

"On" 2 letters.

"and" 3 letters.

"keep" 4 letters in case of tie order by position in original text.

"calm" 4 letters.

"code" 4 letters.

Example 3:

Input: text = "To be or not to be"

Output: "To be or to be not"

Constraints:

text begins with a capital letter and then contains lowercase letters and single space between words.

$1 \leq \text{text.length} \leq 10^5$

*Solution*

05/17/2020:

```
class Solution {
public:
    string arrangeWords(string text) {
        if (text.empty()) return "";
```

```

text.front() = tolower(text.front());
istringstream iss(text);
string ret, s;
vector<pair<string, int>> words;
int i = 0;
while (iss >> s) words.emplace_back(s, ++i);
sort(words.begin(), words.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();
});
bool first = true;
for (auto& w : words) {
    if (first) {
        first = false;
        ret += w.first;
    } else {
        ret += " ";
        ret += w.first;
    }
}
ret.front() = toupper(ret.front());
return ret;
}
};

```

## 1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence

### *Description*

Given a sentence that consists of some words separated by a single space, and a searchWord.

You have to check if searchWord is a prefix of any word in sentence.

Return the index of the word in sentence where searchWord is a prefix of this word (1-indexed).

If searchWord is a prefix of more than one word, return the index of the first word (minimum index). If there is no such word return -1.

A prefix of a string S is any leading contiguous substring of S.

Example 1:

Input: sentence = "i love eating burger", searchWord = "burg"

Output: 4

Explanation: "burg" is prefix of "burger" which is the 4th word in the sentence.

Example 2:

Input: sentence = "this problem is an easy problem", searchWord = "pro"

Output: 2

Explanation: "pro" is prefix of "problem" which is the 2nd and the 6th word in the sentence, but we return 2 as it's the minimal index.

Example 3:

Input: sentence = "i am tired", searchWord = "you"

Output: -1

Explanation: "you" is not a prefix of any word in the sentence.

Example 4:

Input: sentence = "i use triple pillow", searchWord = "pill"

Output: 4

Example 5:

Input: sentence = "hello from the other side", searchWord = "they"

Output: -1

Constraints:

1 <= sentence.length <= 100

1 <= searchWord.length <= 10

sentence consists of lowercase English letters and spaces.

searchWord consists of lowercase English letters.

*Solution*

05/23/2020:

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

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;
}

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

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

class Solution {
public:
    int isPrefixOfWord(string sentence, string searchWord) {
        Trie t;
        string s;
        vector<string> words;
        istringstream iss(sentence);
        int i = 0;
        while (iss >> s) {
            t.insert(s);
            ++i;
            if (t.startsWith(searchWord)) return i;
        }
    }
};

```



```
    return -1;
}
};
```

## 1456. Maximum Number of Vowels in a Substring of Given Length

### *Description*

Given a string  $s$  and an integer  $k$ .

Return the maximum number of vowel letters in any substring of  $s$  with length  $k$ .

Vowel letters in English are (a, e, i, o, u).

Example 1:

Input:  $s = \text{"abciidef"}, k = 3$

Output: 3

Explanation: The substring "iii" contains 3 vowel letters.

Example 2:

Input:  $s = \text{"aeiou"}, k = 2$

Output: 2

Explanation: Any substring of length 2 contains 2 vowels.

Example 3:

Input:  $s = \text{"leetcode"}, k = 3$

Output: 2

Explanation: "lee", "eet" and "ode" contain 2 vowels.

Example 4:

Input:  $s = \text{"rhythms"}, k = 4$

Output: 0

Explanation: We can see that  $s$  doesn't have any vowel letters.

Example 5:

Input:  $s = \text{"tryhard"}, k = 4$

Output: 1

Constraints:

```
1 <= s.length <= 10^5
s consists of lowercase English letters.
1 <= k <= s.length
```

*Solution*

05/23/2020:

```
class Solution {
public:
    int maxVowels(string s, int k) {
        unordered_set<char> vow{'a', 'e', 'i', 'o', 'u'};
        int ret = 0, cnt = 0, slow = 0;
        for (int fast = 0; fast < (int)s.size(); ++fast) {
            if (fast - slow >= k) cnt += vow.count(s[slow++]) > 0 ? -1 : 0;
            if (vow.count(s[fast]) > 0) ret = max(ret, ++cnt);
        }
        return ret;
    }
};
```

## 1487. Making File Names Unique

*Description*

Given an array of strings names of size n. You will create n folders in your file system such that, at the ith minute, you will create a folder with the name names[i].

Since two files cannot have the same name, if you enter a folder name which is previously used, the system will have a suffix addition to its name in the form of (k), where, k is the smallest positive integer such that the obtained name remains unique.

Return an array of strings of length n where ans[i] is the actual name the system will assign to the ith folder when you create it.

Example 1:

Input: names = ["pes","fifa","gta","pes(2019)"]

Output: ["pes","fifa","gta","pes(2019)"]

Explanation: Let's see how the file system creates folder names:

"pes" --> not assigned before, remains "pes"

```
"fifa" --> not assigned before, remains "fifa"  
"gta" --> not assigned before, remains "gta"  
"pes(2019)" --> not assigned before, remains "pes(2019)"  
Example 2:
```

```
Input: names = ["gta","gta(1)","gta","avalon"]  
Output: ["gta","gta(1)","gta(2)","avalon"]  
Explanation: Let's see how the file system creates folder names:  
"gta" --> not assigned before, remains "gta"  
"gta(1)" --> not assigned before, remains "gta(1)"  
"gta" --> the name is reserved, system adds (k), since "gta(1)" is also  
reserved, systems put k = 2. it becomes "gta(2)"  
"avalon" --> not assigned before, remains "avalon"  
Example 3:
```

```
Input: names = ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece"]  
Output: ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece(4)"]  
Explanation: When the last folder is created, the smallest positive valid k is  
4, and it becomes "onepiece(4)".  
Example 4:
```

```
Input: names = ["wano","wano","wano","wano"]  
Output: ["wano","wano(1)","wano(2)","wano(3)"]  
Explanation: Just increase the value of k each time you create folder "wano".  
Example 5:
```

```
Input: names = ["kaido","kaido(1)","kaido","kaido(1)"]  
Output: ["kaido","kaido(1)","kaido(2)","kaido(1)(1)"]  
Explanation: Please note that system adds the suffix (k) to current name even it  
contained the same suffix before.
```

Constraints:

```
1 <= names.length <= 5 * 10^4  
1 <= names[i].length <= 20  
names[i] consists of lower case English letters, digits and/or round brackets.
```

*Solution*

06/20/2020:

```
class Solution {  
public:  
    vector<string> getFolderNames(vector<string>& names) {  
        unordered_set<string> seen;  
        unordered_map<string, int> mp;  
        vector<string> ret;
```

```

for (auto& name : names ) {
    if (seen.count(name) == 0) {
        ret.push_back(name);
        seen.insert(name);
    } else {
        int i = mp.count(name) == 0 ? 1 : mp[name];
        string new_name;
        for (new_name = name + "(" + to_string(i) + ")"; seen.count(new_name) >
0; ++i)
            new_name = name + "(" + to_string(i + 1) + ")";
        ret.push_back(new_name);
        mp[name] = i;
        seen.insert(new_name);
    }
}
return ret;
};

```

# Depth-First Search & Breadth-First Search

## 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*

01/25/2020:

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

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

## 695. Max Area of Island

*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.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1:

```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

Given the above grid, return 6. Note the answer is not 11, because the island must be connected 4-directionally.

Example 2:

```
[[0,0,0,0,0,0,0,0,0]]
```

Given the above grid, return 0.

Note: The length of each dimension in the given grid does not exceed 50.

*Solution*

01/29/2020:

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

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

```

    grid[i][j] = 0;
    if (i > 0) dfs(grid, i - 1, j, area);
    if (j > 0) dfs(grid, i, j - 1, area);
    if (i < m - 1) dfs(grid, i + 1, j, area);
    if (j < n - 1) dfs(grid, i, j + 1, area);
  }
}
};

```

## 886. Possible Bipartition

### Description

Given a set of  $N$  people (numbered  $1, 2, \dots, N$ ), we would like to split everyone into two groups of any size.

Each person may dislike some other people, and they should not go into the same group.

Formally, if  $\text{dislikes}[i] = [a, b]$ , it means it is not allowed to put the people numbered  $a$  and  $b$  into the same group.

Return true if and only if it is possible to split everyone into two groups in this way.

Example 1:

Input:  $N = 4$ ,  $\text{dislikes} = [[1,2],[1,3],[2,4]]$

Output: true

Explanation: group1  $[1,4]$ , group2  $[2,3]$

Example 2:

Input:  $N = 3$ ,  $\text{dislikes} = [[1,2],[1,3],[2,3]]$

Output: false

Example 3:

Input:  $N = 5$ ,  $\text{dislikes} = [[1,2],[2,3],[3,4],[4,5],[1,5]]$

Output: false

Note:

$1 \leq N \leq 2000$

$0 \leq \text{dislikes.length} \leq 10000$

```
1 <= dislikes[i][j] <= N
dislikes[i][0] < dislikes[i][1]
There does not exist i != j for which dislikes[i] == dislikes[j].
```

*Solution*

05/27/2020: Using BFS:

```
class Solution {
public:
    bool possibleBipartition(int n, vector<vector<int>>& dislikes) {
        if (n == 0 || dislikes.empty() || dislikes[0].empty()) return true;
        vector<vector<int>> adj(n + 1);
        for (auto& d : dislikes) {
            adj[d[0]].push_back(d[1]);
            adj[d[1]].push_back(d[0]);
        }
        unordered_map<int, bool> color;
        bool isRed = true;
        unordered_set<int> visited;
        for (int i = 0; i <= n; ++i) {
            queue<int> q;
            q.push(i);
            if (color.count(i) == 0) color[i] = isRed;
            while (!q.empty()) {
                int sz = q.size();
                for (int k = 0; k < sz; ++k) {
                    int cur = q.front(); q.pop();
                    if (visited.count(cur)) continue;
                    for (auto& j : adj[cur]) {
                        if (color.count(j) == 0) {
                            color[j] = !color[cur];
                        } else {
                            if (color[j] == color[cur]) {
                                return false;
                            }
                        }
                    }
                    q.push(j);
                }
                visited.insert(cur);
            }
        }
        return true;
    }
};
```

Using 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) {
        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 possibleBipartition(int n, vector<vector<int>>& dislikes) {
        if (n == 0 || dislikes.empty() || dislikes[0].empty()) return true;
        vector<vector<int>> adj(n);
        for (auto& d : dislikes) {
            adj[d[0] - 1].push_back(d[1] - 1);
            adj[d[1] - 1].push_back(d[0] - 1);
        }
        UnionFind uf(n);
        for (auto& a : adj)
            for (int i = 0; i < (int)a.size() - 1; ++i)
                uf.merge(a[i], a[i + 1]);
    }
};

```

```
    for (auto& d : dislikes)
        if (uf.connected(d[0] - 1, d[1] - 1))
            return false;
    return true;
}
};
```

## 1222. Queens That Can Attack the King

### *Description*

On an 8x8 chessboard, there can be multiple Black Queens and one White King.

Given an array of integer coordinates queens that represents the positions of the Black Queens, and a pair of coordinates king that represent the position of the White King, return the coordinates of all the queens (in any order) that can attack the King.

Example 1:

Input: queens = [[0,1],[1,0],[4,0],[0,4],[3,3],[2,4]], king = [0,0]

Output: [[0,1],[1,0],[3,3]]

Explanation:

The queen at [0,1] can attack the king cause they're in the same row.

The queen at [1,0] can attack the king cause they're in the same column.

The queen at [3,3] can attack the king cause they're in the same diagonal.

The queen at [0,4] can't attack the king cause it's blocked by the queen at [0,1].

The queen at [4,0] can't attack the king cause it's blocked by the queen at [1,0].

The queen at [2,4] can't attack the king cause it's not in the same row/column/diagonal as the king.

Example 2:

Input: queens = [[0,0],[1,1],[2,2],[3,4],[3,5],[4,4],[4,5]], king = [3,3]

Output: [[2,2],[3,4],[4,4]]

Example 3:

```
Input: queens = [[5,6],[7,7],[2,1],[0,7],[1,6],[5,1],[3,7],[0,3],[4,0],[1,2],
[6,3],[5,0],[0,4],[2,2],[1,1],[6,4],[5,4],[0,0],[2,6],[4,5],[5,2],[1,4],[7,5],
[2,3],[0,5],[4,2],[1,0],[2,7],[0,1],[4,6],[6,1],[0,6],[4,3],[1,7]], king = [3,4]
Output: [[2,3],[1,4],[1,6],[3,7],[4,3],[5,4],[4,5]]
```

Constraints:

```
1 <= queens.length <= 63
queens[0].length == 2
0 <= queens[i][j] < 8
king.length == 2
0 <= king[0], king[1] < 8
At most one piece is allowed in a cell.
```

*Solution*

05/27/2020:

```
class Solution {
public:
    vector<vector<int>> queensAttacktheKing(vector<vector<int>>& queens,
vector<int>& king) {
        int dir[8][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1}, {-1, -1}, {-1, 1}, {1,
-1}, {1, 1} };
        int i = king[0], j = king[1], r = 8, c = 8;
        vector<vector<int>> ret;
        unordered_set<string> queenSet;
        for (auto& q : queens) queenSet.insert(to_string(q[0]) + "," +
to_string(q[1]));
        for (int d = 0; d < 8; ++d) {
            for (int ni = i + dir[d][0], nj = j + dir[d][1]; 0 <= ni && ni < r && 0 <=
nj && nj < c; ni += dir[d][0], nj += dir[d][1]) {
                if (queenSet.count(to_string(ni) + "," + to_string(nj)) > 0) {
                    ret.push_back({ni, nj});
                    break;
                }
            }
        }
        return ret;
    }
};
```

# 1489. Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree

## Description

Given a weighted undirected connected graph with  $n$  vertices numbered from  $0$  to  $n-1$ , and an array `edges` where `edges[i] = [fromi, toi, weighti]` represents a bidirectional and weighted edge between nodes `fromi` and `toi`. A minimum spanning tree (MST) is a subset of the edges of the graph that connects all vertices without cycles and with the minimum possible total edge weight.

Find all the critical and pseudo-critical edges in the minimum spanning tree (MST) of the given graph. An MST edge whose deletion from the graph would cause the MST weight to increase is called a critical edge. A pseudo-critical edge, on the other hand, is that which can appear in some MSTs but not all.

Note that you can return the indices of the edges in any order.

Example 1:

Input:  $n = 5$ , `edges = [[0,1,1],[1,2,1],[2,3,2],[0,3,2],[0,4,3],[3,4,3],[1,4,6]]`

Output: `[[0,1],[2,3,4,5]]`

Explanation: The figure above describes the graph.

The following figure shows all the possible MSTs:

Notice that the two edges  $0$  and  $1$  appear in all MSTs, therefore they are critical edges, so we return them in the first list of the output.

The edges  $2$ ,  $3$ ,  $4$ , and  $5$  are only part of some MSTs, therefore they are considered pseudo-critical edges. We add them to the second list of the output.

Example 2:

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

Output: `[[],[0,1,2,3]]`

Explanation: We can observe that since all 4 edges have equal weight, choosing any 3 edges from the given 4 will yield an MST. Therefore all 4 edges are pseudo-critical.

Constraints:

$2 \leq n \leq 100$

```
1 <= edges.length <= min(200, n * (n - 1) / 2)
edges[i].length == 3
0 <= fromi < toi < n
1 <= weighti <= 1000
All pairs (fromi, toi) are distinct.
```

*Solution*

06/20/2020:

```
typedef pair<int, int> pii;
typedef pair<pii, int> metadata;
typedef pair<int, metadata> edge;

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

public:
    UnionFind(int n) {
        init(n);
    }

    void init(int n) {
        id.assign(n, 0);
        iota(id.begin(), id.end(), 0);
        sz.assign(n, 1);
    }

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

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

```

class Solution {
public:
    vector<vector<int>> findCriticalAndPseudoCriticalEdges(int n,
vector<vector<int>>& redges) {
        vector<edge> edges;
        int m = redges.size();
        for (int i = 0; i < m; ++i) {
            auto out = redges[i];
            edges.emplace_back(out[2], metadata(pii(out[0], out[1]), i));
        }
        sort(edges.begin(), edges.end());
        int realweight = 0;
        UnionFind uf(n);
        for (auto out : edges)
            if (uf.merge(out.second.first.first, out.second.first.second))
                realweight += out.first;
        vector<int> lhs, rhs;
        for (int a = 0; a < m; ++a) {
            uf.init(n);
            int nmerge = 0;
            int nowweight = 0;
            for (int i = 0; i < m; ++i) {
                if (i == a) continue;
                if (uf.merge(edges[i].second.first.first, edges[i].second.first.second))
{
                    nmerge++;
                    nowweight += edges[i].first;
                }
            }
            if (nmerge != n - 1 || nowweight > realweight) {
                lhs.push_back(edges[a].second.second);
            }
        }
        vector<int> canappear;
        uf.init(n);
        for (int i = 0; i < m;) {
            int j = i;
            while (j < m && edges[i].first == edges[j].first) ++j;
            for (int k = i; k < j; ++k)
                if (uf.find(edges[k].second.first.first) !=
uf.find(edges[k].second.first.second))
                    canappear.push_back(edges[k].second.second);
            for (int k = i; k < j; ++k) uf.merge(edges[k].second.first.first,
edges[k].second.first.second);
            i = j;
        }
        for (auto& out: canappear) {
            bool in = false;
            for (auto& out2 : lhs) in |= out == out2;

```

```

        if (!in) rhs.push_back(out);
    }
    vector<vector<int>> ret;
    ret.push_back(lhs);
    ret.push_back(rhs);
    return ret;
}
};

```

# Linked List

## 21. Merge Two Sorted Lists

### *Description*

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Example:

Input: 1->2->4, 1->3->4

Output: 1->1->2->3->4->4

### *Solution*

01/25/2020:

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode *pre = new ListNode(0);
        for (auto cur = pre; l1 || l2; cur = cur->next) {
            if (l2 == nullptr || (l1 != nullptr && l1->val <= l2->val)) {
                cur->next = new ListNode(l1->val);
            }
        }
    }
};

```

```

        l1 = l1->next;
    } else {
        cur->next = new ListNode(l2->val);
        l2 = l2->next;
    }
}
return pre->next;
}
};

```

## 61. Rotate List

### *Description*

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Example 1:

Input: 1->2->3->4->5->NULL, k = 2

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

Explanation:

rotate 1 steps to the right: 5->1->2->3->4->NULL

rotate 2 steps to the right: 4->5->1->2->3->NULL

Example 2:

Input: 0->1->2->NULL, k = 4

Output: 2->0->1->NULL

Explanation:

rotate 1 steps to the right: 2->0->1->NULL

rotate 2 steps to the right: 1->2->0->NULL

rotate 3 steps to the right: 0->1->2->NULL

rotate 4 steps to the right: 2->0->1->NULL

### *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* rotateRight(ListNode* head, int k) {
        if (head == nullptr || k == 0) return head;
        int n = 1;
        ListNode* cur = head;
        for (ListNode* cur = head; cur->next; cur = cur->next, ++n);
        k %= n;
        if (k == 0) return head;
        ListNode *slow = head, *fast = head;
        while (k-- > 0 && fast->next) fast = fast->next;
        while (fast->next) {
            slow = slow->next;
            fast = fast->next;
        }
        ListNode* ret = slow->next;
        slow->next = fast->next;
        fast->next = head;
        return ret;
    }
};

```

## 876. Middle of the Linked List

### Description

Given a non-empty, singly linked list with head node head, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

Example 1:

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

Output: Node 3 from this list (Serialization: [3,4,5])

The returned node has value 3. (The judge's serialization of this node is [3,4,5]).

Note that we returned a ListNode object ans, such that:

ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, and ans.next.next.next = NULL.

Example 2:

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

Output: Node 4 from this list (Serialization: [4,5,6])

Since the list has two middle nodes with values 3 and 4, we return the second one.

Note:

The number of nodes in the given list will be between 1 and 100.

*Solution*

01/31/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        int cnt = 0;
        int target = 0;
        for (ListNode* cur = head; cur != nullptr; ++cnt, cur = cur->next);
        target = cnt / 2 - 1;
        ListNode* cur = head;
        for (; target >= 0; cur = cur->next, --target);
        return cur;
    }
};
```

## 1290. Convert Binary Number in a Linked List to Integer

*Description*

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list.

Example 1:

Input: head = [1,0,1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10

Example 2:

Input: head = [0]

Output: 0

Example 3:

Input: head = [1]

Output: 1

Example 4:

Input: head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0]

Output: 18880

Example 5:

Input: head = [0,0]

Output: 0

Constraints:

The Linked List is not empty.

Number of nodes will not exceed 30.

Each node's value is either 0 or 1.

*Solution*

01/29/2020:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    int getDecimalValue(ListNode* head) {
```

```

int ret = 0;
for (; head != nullptr; head = head->next) {
    ret <<= 1;
    ret += head->val;
}
return ret;
}
};

```

## 1474. Delete N Nodes After M Nodes of a Linked List

### Description

Given the head of a linked list and two integers m and n. Traverse the linked list and remove some nodes in the following way:

Start with the head as the current node.

Keep the first m nodes starting with the current node.

Remove the next n nodes

Keep repeating steps 2 and 3 until you reach the end of the list.

Return the head of the modified list after removing the mentioned nodes.

Follow up question: How can you solve this problem by modifying the list in-place?

Example 1:

Input: head = [1,2,3,4,5,6,7,8,9,10,11,12,13], m = 2, n = 3

Output: [1,2,6,7,11,12]

Explanation: Keep the first (m = 2) nodes starting from the head of the linked List (1 → 2) show in black nodes.

Delete the next (n = 3) nodes (3 → 4 → 5) show in red nodes.

Continue with the same procedure until reaching the tail of the Linked List.

Head of linked list after removing nodes is returned.

Example 2:

Input: head = [1,2,3,4,5,6,7,8,9,10,11], m = 1, n = 3

Output: [1,5,9]

Explanation: Head of linked list after removing nodes is returned.

Example 3:

Input: head = [1,2,3,4,5,6,7,8,9,10,11], m = 3, n = 1

Output: [1,2,3,5,6,7,9,10,11]

Example 4:

Input: head = [9,3,7,7,9,10,8,2], m = 1, n = 2

Output: [9,7,8]

Constraints:

The given linked list will contain between 1 and  $10^4$  nodes.

The value of each node in the linked list will be in the range [1,  $10^6$ ].

$1 \leq m, n \leq 1000$

*Solution*

06/11/2020:

```
/**
 * 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* deleteNodes(ListNode* head, int m, int n) {
        ListNode *cur = head, *fast;
        while (cur) {
            for (int skipCount = m; --skipCount > 0 && cur && cur->next; cur = cur->next);
            fast = cur->next;
            for (int removeCount = n; --removeCount > 0 && fast && fast->next; fast = fast->next);
            if (fast) cur->next = fast->next;
            cur = cur->next;
        }
        return head;
    }
};
```

# Bit Manipulation

## 338. Counting Bits

### Description

Given a non negative integer number num. For every numbers i in the range  $0 \leq i \leq \text{num}$  calculate the number of 1's in their binary representation and return them as an array.

Example 1:

Input: 2

Output: [0,1,1]

Example 2:

Input: 5

Output: [0,1,1,2,1,2]

Follow up:

It is very easy to come up with a solution with run time  $O(n * \text{sizeof}(\text{integer}))$ . But can you do it in linear time  $O(n)$  /possibly in a single pass? Space complexity should be  $O(n)$ . Can you do it like a boss? Do it without using any builtin function like `__builtin_popcount` in c++ or in any other language.

### Solution

05/28/2020:

```
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> ret(num + 1, 0);
        for (int i = 1; i <= num; ++i) {
            ret[i] = ret[i & (i - 1)] + 1;
        }
        return ret;
    }
};
```

```

class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> ret;
        for (int i = 0; i <= num; ++i) {
            ret.push_back(__builtin_popcount(i));
        }
        return ret;
    }
};

```

```

class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> dp(num + 1, 0);
        for (int i = 1, m = 1; i <= num; ++i) {
            if (i == m << 1) m <<= 1;
            dp[i] = dp[i % m] + 1;
        }
        return dp;
    }
};

```

## 342. Power of Four

### Description

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example 1:

Input: 16

Output: true

Example 2:

Input: 5

Output: false

Follow up: Could you solve it without loops/recursion?

### Solution

06/16/2020:

```
class Solution {
public:
    bool isPowerOfFour(int num) {
        int n = __builtin_popcount(num);
        if (n != 1 || n <= 0) return false;
        while (num != 0 && (num & 1) != 1) num >>= 2;
        return num == 1;
    }
};
```

## 461. Hamming Distance

### *Description*

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers  $x$  and  $y$ , calculate the Hamming distance.

Note:

$0 \leq x, y < 231$ .

Example:

Input:  $x = 1, y = 4$

Output: 2

Explanation:

```
1  (0 0 0 1)
4  (0 1 0 0)
   ↑  ↑
```

The above arrows point to positions where the corresponding bits are different.

### *Solution*

01/30/2020:



```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int ret = 0, m = x ^ y;
        for (int m = x ^ y; m != 0; m /= 2) {
            ret += m % 2;
        }
        return ret;
    }
};
```

## 476. Number Complement

### *Description*

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Example 1:

Input: 5

Output: 2

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

Example 2:

Input: 1

Output: 0

Explanation: The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

Note:

The given integer is guaranteed to fit within the range of a 32-bit signed integer.

You could assume no leading zero bit in the integer's binary representation.

This question is the same as 1009: <https://leetcode.com/problems/complement-of-base-10-integer/>

### *Solution*

05/04/2020:

```
class Solution {
public:
    int findComplement(int num) {
        int ret = 0, bits = 0;
        for (; num != 0; num /= 2, ++bits) {
            if (num % 2 == 0) {
                ret += 1 << bits;
            }
        }
        return ret;
    }
};
```

## 1486. XOR Operation in an Array

### Description

Given an integer  $n$  and an integer  $start$ .

Define an array  $nums$  where  $nums[i] = start + 2 * i$  ( $0$ -indexed) and  $n == nums.length$ .

Return the bitwise XOR of all elements of  $nums$ .

Example 1:

Input:  $n = 5, start = 0$

Output: 8

Explanation: Array  $nums$  is equal to  $[0, 2, 4, 6, 8]$  where  $(0 \wedge 2 \wedge 4 \wedge 6 \wedge 8) = 8$ .

Where " $\wedge$ " corresponds to bitwise XOR operator.

Example 2:

Input:  $n = 4, start = 3$

Output: 8

Explanation: Array  $nums$  is equal to  $[3, 5, 7, 9]$  where  $(3 \wedge 5 \wedge 7 \wedge 9) = 8$ .

Example 3:

Input:  $n = 1, start = 7$

Output: 7

Example 4:

Input: n = 10, start = 5  
Output: 2

Constraints:

```
1 <= n <= 1000  
0 <= start <= 1000  
n == nums.length
```

*Solution*

06/20/2020:

```
class Solution {  
public:  
    int xorOperation(int n, int start) {  
        int ret = 0;  
        for (int i = 0; i < n; ++i) ret ^= start + 2 * i;  
        return ret;  
    }  
};
```

# Design Data Structure

## 359. Logger Rate Limiter

*Description*

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is not printed in the last 10 seconds.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

Example:

```
Logger logger = new Logger();
```

```
// logging string "foo" at timestamp 1
logger.shouldPrintMessage(1, "foo"); returns true;

// logging string "bar" at timestamp 2
logger.shouldPrintMessage(2,"bar"); returns true;

// logging string "foo" at timestamp 3
logger.shouldPrintMessage(3,"foo"); returns false;

// logging string "bar" at timestamp 8
logger.shouldPrintMessage(8,"bar"); returns false;

// logging string "foo" at timestamp 10
logger.shouldPrintMessage(10,"foo"); returns false;

// logging string "foo" at timestamp 11
logger.shouldPrintMessage(11,"foo"); returns true;
```

*Solution*

01/31/2020:

```
class Logger {
private:
    unordered_map<string, int> logger;
public:
    /** Initialize your data structure here. */
    Logger() {
    }

    /** Returns true if the message should be printed in the given timestamp,
    otherwise returns false.
        If this method returns false, the message will not be printed.
        The timestamp is in seconds granularity. */
    bool shouldPrintMessage(int timestamp, string message) {
        if (logger.count(message) == 0) {
            logger[message] = timestamp;
            return true;
        } else if (timestamp - logger[message] >= 10) {
            logger[message] = timestamp;
            return true;
        } else {
            return false;
        }
    }
};
```

```
/**
 * Your Logger object will be instantiated and called as such:
 * Logger* obj = new Logger();
 * bool param_1 = obj->shouldPrintMessage(timestamp,message);
 */
```

## 901. Online Stock Span

### *Description*

Write a class StockSpanner which collects daily price quotes for some stock, and returns the span of that stock's price for the current day.

The span of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backwards) for which the price of the stock was less than or equal to today's price.

For example, if the price of a stock over the next 7 days were [100, 80, 60, 70, 60, 75, 85], then the stock spans would be [1, 1, 1, 2, 1, 4, 6].

Example 1:

Input: ["StockSpanner","next","next","next","next","next","next","next"], [[], [100],[80],[60],[70],[60],[75],[85]]

Output: [null,1,1,1,2,1,4,6]

Explanation:

First, S = StockSpanner() is initialized. Then:

S.next(100) is called and returns 1,

S.next(80) is called and returns 1,

S.next(60) is called and returns 1,

S.next(70) is called and returns 2,

S.next(60) is called and returns 1,

S.next(75) is called and returns 4,

S.next(85) is called and returns 6.

Note that (for example) S.next(75) returned 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.

Note:

Calls to StockSpanner.next(int price) will have  $1 \leq \text{price} \leq 10^5$ .

There will be at most 10000 calls to StockSpanner.next per test case.

There will be at most 150000 calls to StockSpanner.next across all test cases.

The total time limit for this problem has been reduced by 75% for C++, and 50% for all other languages.

*Solution*

05/20/2020:

```
class StockSpanner {
private:
    stack<pair<int, int>> st;

public:
    StockSpanner() {
    }

    int next(int price) {
        int w = 1;
        while (!st.empty() && st.top().first <= price) {
            w += st.top().second;
            st.pop();
        }
        st.emplace(price, w);
        return w;
    }
};
```

```
class StockSpanner {
private:
    vector<int> prices;
    vector<int> upper_bounds;
    int i, last;

public:
    StockSpanner() {
        prices.resize(100001);
        upper_bounds.resize(100001, INT_MAX);
        i = 0;
        last = 0;
    }

    int next(int price) {
        prices[price] = i++;
        while (last != INT_MAX && last <= price) last = upper_bounds[last];
        upper_bounds[price] = last == INT_MAX ? min(upper_bounds[price], last) :
last > price ? last : INT_MAX;
        last = price;
        if (upper_bounds[price] == INT_MAX) return prices[price] + 1;
    }
};
```

```

        return prices[price] - prices[upper_bounds[price]];
    }
};

```

```

class StockSpanner {
    vector<vector<int>> prices;
    vector<int> upper_bounds;
    int i;
    int last;
    bool first;
public:
    StockSpanner() {
        prices.resize(100001);
        upper_bounds.resize(100001, INT_MAX);
        i = 0;
        last = 0;
    }

    int next(int price) {
        prices[price].push_back(i++);
        while (last != INT_MAX && last <= price) {
            last = upper_bounds[last];
        }
        if (last == INT_MAX) {
            upper_bounds[price] = min(upper_bounds[price], last);
        } else {
            if (last > price) {
                upper_bounds[price] = last;
            }
        }
        // cout << price << " " << upper_bounds[price] << endl;
        last = price;
        if (upper_bounds[price] == INT_MAX) {
            return prices[price].back() + 1;
        } else {
            return prices[price].back() - prices[upper_bounds[price]].back();
        }
    }
};

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner* obj = new StockSpanner();
 * int param_1 = obj->next(price);
 */

```

# 1429. First Unique Number

## Description

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the FirstUnique class:

FirstUnique(int[] nums) Initializes the object with the numbers in the queue.  
int showFirstUnique() returns the value of the first unique integer of the queue, and returns -1 if there is no such integer.  
void add(int value) insert value to the queue.

Example 1:

Input:

```
["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique",  
,"add","showFirstUnique"]  
[[[2,3,5]],[],[5],[],[2],[],[3],[]]
```

Output:

```
[null,2,null,2,null,3,null,-1]
```

Explanation:

```
FirstUnique firstUnique = new FirstUnique([2,3,5]);  
firstUnique.showFirstUnique(); // return 2  
firstUnique.add(5);           // the queue is now [2,3,5,5]  
firstUnique.showFirstUnique(); // return 2  
firstUnique.add(2);           // the queue is now [2,3,5,5,2]  
firstUnique.showFirstUnique(); // return 3  
firstUnique.add(3);           // the queue is now [2,3,5,5,2,3]  
firstUnique.showFirstUnique(); // return -1
```

Example 2:

Input:

```
["FirstUnique","showFirstUnique","add","add","add","add","add","showFirstUnique"  
]  
[[[7,7,7,7,7,7]],[],[7],[3],[3],[7],[17],[]]
```

Output:

```
[null,-1,null,null,null,null,null,17]
```

Explanation:

```
FirstUnique firstUnique = new FirstUnique([7,7,7,7,7,7]);  
firstUnique.showFirstUnique(); // return -1  
firstUnique.add(7);           // the queue is now [7,7,7,7,7,7]  
firstUnique.add(3);           // the queue is now [7,7,7,7,7,7,3]
```



```

firstUnique.add(3);           // the queue is now [7,7,7,7,7,7,7,3,3]
firstUnique.add(7);         // the queue is now [7,7,7,7,7,7,7,3,3,7]
firstUnique.add(17);        // the queue is now [7,7,7,7,7,7,7,3,3,7,17]
firstUnique.showFirstUnique(); // return 17

```

Example 3:

Input:

```

["FirstUnique","showFirstUnique","add","showFirstUnique"]
[[[809]],[],[809],[]]

```

Output:

```
[null,809,null,-1]
```

Explanation:

```

FirstUnique firstUnique = new FirstUnique([809]);
firstUnique.showFirstUnique(); // return 809
firstUnique.add(809);         // the queue is now [809,809]
firstUnique.showFirstUnique(); // return -1

```

Constraints:

$1 \leq \text{nums.length} \leq 10^5$

$1 \leq \text{nums}[i] \leq 10^8$

$1 \leq \text{value} \leq 10^8$

At most 50000 calls will be made to showFirstUnique and add.

\* Hide Hint #1: Use doubly Linked list with hashmap of pointers to linked list nodes. add unique number to the linked list. When add is called check if the added number is unique then it have to be added to the linked list and if it is repeated remove it from the linked list if exists. When showFirstUnique is called retrieve the head of the linked list.

\* Hide Hint #2: Use queue and check that first element of the queue is always unique.

\* Hide Hint #3: Use set or heap to make running time of each function  $O(\log n)$ .

*Solution*

Using vector:

```

class FirstUnique {
    unordered_map<int, int> mp;
    vector<int> unique_nums;
    int i;

public:
    FirstUnique(vector<int>& nums) {
        i = 0;
        for (auto& n : nums) ++mp[n];
        for (auto& n : nums)

```

```

        if (mp[n] == 1)
            unique_nums.push_back(n);
    }

    int showFirstUnique() {
        if (i >= unique_nums.size()) return -1;
        return unique_nums[i];
    }

    void add(int value) {
        if (++mp[value] == 1) unique_nums.push_back(value);
        for (; i < unique_nums.size() && mp[unique_nums[i]] > 1; ++i);
    }
};

/**
 * Your FirstUnique object will be instantiated and called as such:
 * FirstUnique* obj = new FirstUnique(nums);
 * int param_1 = obj->showFirstUnique();
 * obj->add(value);
 */

```

Using queue:

```

class FirstUnique {
    unordered_map<int, int> mp;
    queue<int> unique_nums;

public:
    FirstUnique(vector<int>& nums) {
        for (auto& n : nums) ++mp[n];
        for (auto& n : nums)
            if (mp[n] == 1)
                unique_nums.push(n);
    }

    int showFirstUnique() {
        if (unique_nums.empty()) return -1;
        return unique_nums.front();
    }

    void add(int value) {
        if (++mp[value] == 1) unique_nums.push(value);
        while (!unique_nums.empty() && mp[unique_nums.front()] > 1)
            unique_nums.pop();
    }
};

```

```
/**
 * Your FirstUnique object will be instantiated and called as such:
 * FirstUnique* obj = new FirstUnique(nums);
 * int param_1 = obj->showFirstUnique();
 * obj->add(value);
 */
```

## 1472. Design Browser History

### Description

You have a browser of one tab where you start on the homepage and you can visit another url, get back in the history number of steps or move forward in the history number of steps.

Implement the BrowserHistory class:

BrowserHistory(string homepage) Initializes the object with the homepage of the browser.

void visit(string url) visits url from the current page. It clears up all the forward history.

string back(int steps) Move steps back in history. If you can only return x steps in the history and steps > x, you will return only x steps. Return the current url after moving back in history at most steps.

string forward(int steps) Move steps forward in history. If you can only forward x steps in the history and steps > x, you will forward only x steps. Return the current url after forwarding in history at most steps.

Example:

Input:

```
["BrowserHistory","visit","visit","visit","back","back","forward","visit","forward","back","back"]
[["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]
```

Output:

```
[null,null,null,null,"facebook.com","google.com","facebook.com",null,"linkedin.com","google.com","leetcode.com"]
```

Explanation:

```
BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com"); // You are in "leetcode.com". Visit "google.com"
browserHistory.visit("facebook.com"); // You are in "google.com". Visit "facebook.com"
```

```

browserHistory.visit("youtube.com");    // You are in "facebook.com". Visit
"youtube.com"
browserHistory.back(1);                 // You are in "youtube.com", move back
to "facebook.com" return "facebook.com"
browserHistory.back(1);                 // You are in "facebook.com", move
back to "google.com" return "google.com"
browserHistory.forward(1);              // You are in "google.com", move
forward to "facebook.com" return "facebook.com"
browserHistory.visit("linkedin.com");    // You are in "facebook.com". Visit
"linkedin.com"
browserHistory.forward(2);               // You are in "linkedin.com", you
cannot move forward any steps.
browserHistory.back(2);                  // You are in "linkedin.com", move
back two steps to "facebook.com" then to "google.com". return "google.com"
browserHistory.back(7);                  // You are in "google.com", you can
move back only one step to "leetcode.com". return "leetcode.com"

```

Constraints:

```

1 <= homepage.length <= 20
1 <= url.length <= 20
1 <= steps <= 100
homepage and url consist of '.' or lower case English letters.
At most 5000 calls will be made to visit, back, and forward.

```

*Solution*

06/06/2020:

```

class BrowserHistory {
private:
    list<string> history;
    int x;

public:
    BrowserHistory(string homepage) {
        history.push_back(homepage);
        x = 0;
    }

    void visit(string url) {
        history.erase(next(history.begin(), x + 1), history.end());
        history.push_back(url);
        ++x;
    }

    string back(int steps) {

```

```

    x = steps > x ? 0 : x - steps;
    return *next(history.begin(), x);
}

string forward(int steps) {
    int n = history.size();
    x = steps > n - 1 - x ? n - 1 : x + steps;
    return *next(history.begin(), x);
}
};

/**
 * Your BrowserHistory object will be instantiated and called as such:
 * BrowserHistory* obj = new BrowserHistory(homepage);
 * obj->visit(url);
 * string param_2 = obj->back(steps);
 * string param_3 = obj->forward(steps);
 */

```

## 1476. Subrectangle Queries

### Description

Implement the class `SubrectangleQueries` which receives a rows x cols rectangle as a matrix of integers in the constructor and supports two methods:

1. `updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)`

Updates all values with `newValue` in the subrectangle whose upper left coordinate is `(row1,col1)` and bottom right coordinate is `(row2,col2)`.

2. `getValue(int row, int col)`

Returns the current value of the coordinate `(row,col)` from the rectangle.

Example 1:

Input

```

["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue","getValue"]
[[[1,2,1],[4,3,4],[3,2,1],[1,1,1]], [0,2],[0,0,3,2,5],[0,2],[3,1],[3,0,3,2,10],[3,1],[0,2]]

```

Output

```

[null,1,null,5,5,null,10,5]

```

Explanation

```

SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,2,1],
[4,3,4],[3,2,1],[1,1,1]]);
// The initial rectangle (4x3) looks like:
// 1 2 1
// 4 3 4
// 3 2 1
// 1 1 1
subrectangleQueries.getValue(0, 2); // return 1
subrectangleQueries.updateSubrectangle(0, 0, 3, 2, 5);
// After this update the rectangle looks like:
// 5 5 5
// 5 5 5
// 5 5 5
// 5 5 5
subrectangleQueries.getValue(0, 2); // return 5
subrectangleQueries.getValue(3, 1); // return 5
subrectangleQueries.updateSubrectangle(3, 0, 3, 2, 10);
// After this update the rectangle looks like:
// 5 5 5
// 5 5 5
// 5 5 5
// 10 10 10
subrectangleQueries.getValue(3, 1); // return 10
subrectangleQueries.getValue(0, 2); // return 5
Example 2:

```

Input

```

["SubrectangleQueries","getValue","updateSubrectangle","getValue","getValue","updateSubrectangle","getValue"]
[[[[1,1,1],[2,2,2],[3,3,3]], [0,0], [0,0,2,2,100], [0,0], [2,2], [1,1,2,2,20], [2,2]]

```

Output

```

[null,1,null,100,100,null,20]

```

Explanation

```

SubrectangleQueries subrectangleQueries = new SubrectangleQueries([[1,1,1],
[2,2,2],[3,3,3]]);
subrectangleQueries.getValue(0, 0); // return 1
subrectangleQueries.updateSubrectangle(0, 0, 2, 2, 100);
subrectangleQueries.getValue(0, 0); // return 100
subrectangleQueries.getValue(2, 2); // return 100
subrectangleQueries.updateSubrectangle(1, 1, 2, 2, 20);
subrectangleQueries.getValue(2, 2); // return 20

```

Constraints:

There will be at most 500 operations considering both methods:  
updateSubrectangle and getValue.  
1 <= rows, cols <= 100  
rows == rectangle.length

```
cols == rectangle[i].length
0 <= row1 <= row2 < rows
0 <= col1 <= col2 < cols
1 <= newValue, rectangle[i][j] <= 10^9
0 <= row < rows
0 <= col < cols
```

*Solution*

06/13/2020:

```
class SubrectangleQueries {
private:
    vector<vector<int>> rectangle;
public:
    SubrectangleQueries(vector<vector<int>>& rectangle) {
        this->rectangle = rectangle;
    }

    void updateSubrectangle(int row1, int col1, int row2, int col2, int newValue)
    {
        for (int i = row1; i <= row2; ++i) {
            for (int j = col1; j <= col2; ++j) {
                rectangle[i][j] = newValue;
            }
        }
    }

    int getValue(int row, int col) {
        return rectangle[row][col];
    }
};

/**
 * Your SubrectangleQueries object will be instantiated and called as such:
 * SubrectangleQueries* obj = new SubrectangleQueries(rectangle);
 * obj->updateSubrectangle(row1,col1,row2,col2,newValue);
 * int param_2 = obj->getValue(row,col);
 */
```