Top Google Questions – Part 2

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

- 409. Longest Palindrome
- 415. Add Strings
- 429. N-ary Tree Level Order Traversal
- 438. Find All Anagrams in a String
- 447. Number of Boomerangs
- 451. Sort Characters By Frequency
- 468. Validate IP Address
- 470. Implement Rand10() Using Rand7()
- 475. Heaters
- 482. License Key Formatting
- 498. Diagonal Traverse
- 510. Inorder Successor in BST II
- 518. Coin Change 2
- 525. Contiguous Array
- 528. Random Pick with Weight
- 535. Encode and Decode TinyURL
- 540. Single Element in a Sorted Array
- 542. 01 Matrix
- 551. Student Attendance Record I
- 567. Permutation in String

249. Group Shifted Strings

Description

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

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

Example:

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

Solution

05/25/2020:

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

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

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

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

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

250. Count Univalue Subtrees

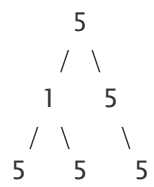
Description

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

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

Example :

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



Output: 4

Solution

05/25/2020:

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

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

```
};
```

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

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

251. Flatten 2D Vector

Description

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

Example:

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

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

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

Notes:

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

Follow up:

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

Solution

05/25/2020:

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

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

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

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

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

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

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

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

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

252. Meeting Rooms

Description

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

Example 1:

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

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

Solution

05/26/2020:

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

253. Meeting Rooms II

Description

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

Example 1:

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

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

Solution

05/27/2020:

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

257. Binary Tree Paths

Description

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

Note: A leaf is a node with no children.

Example:

Input:

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

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

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

Solution

05/26/2020:

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

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

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

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

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

265. Paint House II

Description

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

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

Note:
All costs are positive integers.

Example:

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

Solution

05/28/2020:

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

267. Palindrome Permutation II

Description

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

Example 1:

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

Input: "abc"
Output: []

Solution

05/28/2020:

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

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

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

268. Missing Number

Description

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

Example 1:

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

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

Solution

05/27/2020:

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

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

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

270. Closest Binary Search Tree Value

Description

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

Note:

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

Example:

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

```
    4
   / \
  2   5
```

```
 / \
1   3

Output: 4
```

Solution

04/28/2020:

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

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

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

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

274. H-Index

Description

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

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

Example:

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

Solution

05/30/2020:

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

276. Paint Fence

Description

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

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

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

Note:
n and k are non-negative integers.

Example:

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

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

Solution

01/14/2020 (Dynamic Programming):

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

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

```
class Solution {
```

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

278. First Bad Version

Description

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

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

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

Example:

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

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

Then 4 is the first bad version.

Solution

04/23/2020:

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

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

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

285. Inorder Successor in BST

Description

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

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

Example 1:

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

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

Note:

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

Solution

05/24/2020:

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

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

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

286. Walls and Gates

Description

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

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

Example:

Given the 2D grid:

```
INF  -1  0  INF INF
INF INF -1 INF  -
1  INF  -1
  0   -1  INF  INF
```

After running your function, the 2D grid should be:

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

Solution

05/08/2020 (DFS):

```
class Solution {
```

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

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

05/08/2020 (BFS):

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

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

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

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

287. Find the Duplicate Number

Description

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

Example 1:

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

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

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

Solution

04/28/2020:

Use linear search on sorted array:

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

Use binary search on sorted array:

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

Use hash map:

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

290. Word Pattern

Description

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

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

Example 1:

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

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

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

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

Solution

05/20/2020:

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

```
    }
};
```

299. Bulls and Cows

Description

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

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

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

Example 1:

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

Output: "1A3B"

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

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

Output: "1A1B"

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

Solution

02/05/2020:

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

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

303. Range Sum Query - Immutable

Description

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

Example:

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

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

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

Solution

01/14/2020 (Dynamic Programming):

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

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

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

305. Number of Islands II

Description

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

Example:

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

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

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

1 0 0

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

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

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

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

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

Solution

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

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

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

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

  int find(int x) {
```

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

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

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

```
};
```

311. Sparse Matrix Multiplication

Description

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

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

Example:

Input:

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

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

Output:

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

Solution

05/20/2020:

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

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

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

314. Binary Tree Vertical Order Traversal

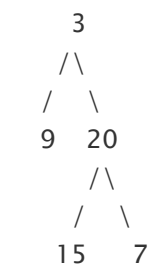
Description

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

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

Examples 1:

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

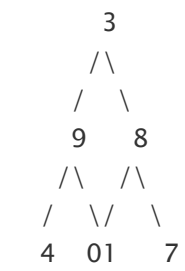


Output:

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

Examples 2:

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

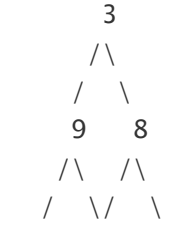


Output:

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

Examples 3:

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



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

Output:

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

Solution

04/26/2020:

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

Using unordered_map:

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

322. Coin Change

Description

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

Example 1:

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

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

Solution

05/27/2020:

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

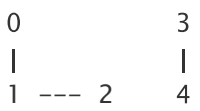
323. Number of Connected Components in an Undirected Graph

Description

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

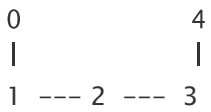
Example 1:

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



Output: 2
Example 2:

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



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

Solution

05/05/2020 (BFS):

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

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

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

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

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

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

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

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

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

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

326. Power of Three

Description

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

Example 1:

Input: 27
Output: true
Example 2:

Input: 0
Output: false
Example 3:

Input: 9
Output: true
Example 4:

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

Solution

06/10/2020:

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

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

328. Odd Even Linked List

Description

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

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

Example 1:

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

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

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

Solution

05/14/2020:

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* oddEvenList(ListNode* head) {
    if (!head || !head->next) return head;
    ListNode *oddHead = head, *evenHead = head->next;
    ListNode *cur = evenHead, *oddCur = oddHead, *evenCur = evenHead;
    while (cur->next) {
      if (cur) {
        oddCur->next = cur->next;
        oddCur = oddCur->next;
        cur = cur->next;
      }
      if (cur) {
        evenCur->next = cur->next;
        evenCur = evenCur->next;
        cur = cur->next;
      }
      if (!cur) break;
    }
    oddCur->next = evenHead;
    return oddHead;
  }
};
```

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
  ListNode* oddEvenList(ListNode* head) {
```

```
    if (!head || !head->next) return head;
    ListNode *evenHead = head->next, *evenCur = evenHead, *cur = head;
    while (cur->next && evenCur->next) {
      cur->next = cur->next->next;
      cur = cur->next;
      evenCur->next = evenCur->next->next;
      evenCur = cur->next;
    }
    cur->next =  evenHead;
    return head;
  }
};
```

344. Reverse String

Description

Write a function that reverses a string. The input string is given as an array of characters char[].

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

You may assume all the characters consist of printable ascii characters.

Example 1:

```
Input:  ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
Example 2:

Input:  ["H","a","n","n","a","h"]
Output:  ["h","a","n","n","a","H"]
```

Solution

05/30/2020:

```
class Solution {
public:
  void reverseString(vector<char>& s) {
    for (int lo = 0, hi = s.size() - 1; lo < hi; ++lo, --hi) swap(s[lo], s[hi]);
  }
};
```

345. Reverse Vowels of a String

Description

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

Input: "hello"
Output: "holle"
Example 2:

Input: "leetcode"
Output: "leotcede"
Note:
The vowels does not include the letter "y".

Solution

06/16/2020:

```
class Solution {
public:
  string reverseVowels(string s) {
    unordered_set<char> vowels{'a', 'e', 'i', 'o', 'u'};
    int n = s.size(), l = 0, r = n - 1;
    while (l < r) {
      while (!isalpha(s[l]) && l < n - 1) ++l;
      while (!isalpha(s[l]) && r > 0) --r;
      while (l < n - 1 && vowels.count(tolower(s[l])) == 0) ++l;
      while (r > 0 && vowels.count(tolower(s[r])) == 0) --r;
      if (l < r)
        if (vowels.count(tolower(s[l])) > 0 && vowels.count(tolower(s[r])) > 0)
          swap(s[l++], s[r--]);
    }
    return s;
  }
```

```
};
```

347. Top K Frequent Elements

Description

Given a non-empty array of integers, return the k most frequent elements.

Example 1:

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

Input: nums = [1], k = 1
Output: [1]
Note:

You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
Your algorithm's time complexity must be better than O(n log n), where n is the array's size.
It's guaranteed that the answer is unique, in other words the set of the top k frequent elements is unique.
You can return the answer in any order.

Solution

05/20/2020:

```
class Solution {
public:
  vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> cnt;
    for (auto& n : nums) ++cnt[n];
    priority_queue<pair<int, int>, vector<pair<int, int>>, less<pair<int, int>>>
q;
    for (auto m : cnt) q.emplace(m.second, m.first);
    vector<int> ret;
    while (k-- > 0) {
      ret.push_back(q.top().second);
      q.pop();
    }
    return ret;
  }
};
```

```cpp
class Solution {
public:
  vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> cnt;
    for (auto& n : nums) ++cnt[n];
    vector<pair<int, int>> freq;
    for (auto& m : cnt) freq.emplace_back(m.second, m.first);
    sort(freq.begin(), freq.end(), greater<pair<int, int>>());
    vector<int> ret;
    for (auto& f : freq) {
      if (k-- > 0) {
        ret.push_back(f.second);
      } else {
        break;
      }
    }
    return ret;
  }
};
```

348. Design Tic-Tac-Toe

Description

Design a Tic-tac-toe game that is played between two players on a n x n grid.

You may assume the following rules:

A move is guaranteed to be valid and is placed on an empty block.
Once a winning condition is reached, no more moves is allowed.
A player who succeeds in placing n of their marks in a horizontal, vertical, or diagonal row wins the game.
Example:
Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.

```
TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)
|X| | |
| | | |    // Player 1 makes a move at (0, 0).
| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 2 makes a move at (0, 2).
```

```
| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 1 makes a move at (2, 2).
| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 2 makes a move at (1, 1).
| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 1 makes a move at (2, 0).
|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)
|X| |O|
|O|O| |    // Player 2 makes a move at (1, 0).
|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
|X| |O|
|O|O| |    // Player 1 makes a move at (2, 1).
|X|X|X|
Follow up:
Could you do better than O(n2) per move() operation?
```

Solution

06/15/2020:

```
class TicTacToe {
  vector<vector<int>> board;
  vector<int> rows;
  vector<int> cols;
  int  diag;
  int antiDiag;
  int n;

public:
  /** Initialize your data structure here. */
  TicTacToe(int n) {
    board.assign(n, vector<int>(n, -1));
    rows.assign(n, 0);
    cols.assign(n, 0);
    diag = 0;
```

```
    antiDiag = 0;
    this->n = n;
  }

  /** Player {player} makes a move at ({row}, {col}).
      @param row The row of the board.
      @param col The column of the board.
      @param player The player, can be either 1 or 2.
      @return The current winning condition, can be either:
              0: No one wins.
              1: Player 1 wins.
              2: Player 2 wins. */
  int move(int row, int col, int player) {
    if (board[row][col] != -1) return 0;
    board[row][col] = player;
    if (++rows[row] == n) {
      int countRow = 0;
      for (int i = 0; i < n && board[row][i] == player; ++i) ++countRow;
      if (countRow == n) return player;
    }
    if (++cols[col]) {
      int countCol = 0;
      for (int i = 0; i < n && board[i][col] == player; ++i) ++countCol;
      if (countCol == n) return player;
    }
    if (row == col && ++diag == n) {
      int countDiag = 0;
      for (int i = 0; i < n && board[i][i] == player; ++i) ++countDiag;
      if (countDiag == n) return player;
    }
    if (row == n - 1 - col && ++antiDiag == n) {
      int countAntiDiag = 0;
      for (int i = 0; i < n && board[i][n - i - 1] == player; ++i)
++countAntiDiag;
      if (countAntiDiag == n) return player;
    }
    return 0;
  }
};

/**
 * Your TicTacToe object will be instantiated and called as such:
 * TicTacToe* obj = new TicTacToe(n);
 * int param_1 = obj->move(row,col,player);
 */
```

367. Valid Perfect Square

Description

> Given a positive integer num, write a function which returns True if num is a perfect square else False.
>
> Note: Do not use any built-in library function such as sqrt.
>
> Example 1:
>
> Input: 16
> Output: true
> Example 2:
>
> Input: 14
> Output: false

Solution

05/08/2020:

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

368. Largest Divisible Subset

Description

Given a set of distinct positive integers, find the largest subset such that every pair (Si, Sj) of elements in this subset satisfies:

Si % Sj = 0 or Sj % Si = 0.

If there are multiple solutions, return any subset is fine.

Example 1:

Input: [1,2,3]
Output: [1,2] (of course, [1,3] will also be ok)
Example 2:

Input: [1,2,4,8]
Output: [1,2,4,8]

Solution

06/13/2020:

Dynamic Programming:

```
class Solution {
public:
  vector<int> largestDivisibleSubset(vector<int>& nums) {
    if (nums.empty()) return nums;
    sort(nums.begin(), nums.end());
    unordered_map<int, int> dp;
    unordered_map<int, int> pre;
    for (auto& n : nums) {
      dp[n] = 1;
      pre[n] = n;
    }
    int n = nums.size(), max_count = 1, max_num = nums[0];
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < i; ++j) {
        if (nums[i] % nums[j] == 0) {
          if (dp[nums[j]] + 1 > dp[nums[i]]) {
            pre[nums[i]] = nums[j];
            dp[nums[i]] = dp[nums[j]] + 1;
            if (dp[nums[i]] > max_count) {
              max_count = dp[nums[i]];
              max_num = nums[i];
            }
          }
        }
      }
    }
```

```
    vector<int> ret(1, max_num);
    while (max_num != pre[max_num]) {
      max_num = pre[max_num];
      ret.push_back(max_num);
    }
    return ret;
  }
};
```

Backtracking (TLE):

```
class Solution {
public:
  vector<int> ret;
  vector<int> largestDivisibleSubset(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    int n = nums.size();
    vector<int> s;
    backtrack(0, n, s, nums);
    return ret;
  }

  void backtrack(int k, int n, vector<int>& s, vector<int>& nums) {
    if (k == n) {
      if (s.size() > ret.size()) ret = s;
      return;
    }
    bool ok = true;
    for (int i = s.size() - 1; i >= 0 && ok; --i)
      ok = nums[k] % s[i] == 0;
    if (ok) {
      s.push_back(nums[k]);
      backtrack(k + 1, n, s, nums);
      s.pop_back();
    }
    backtrack(k + 1, n, s, nums);
  }
};
```

374. Guess Number Higher or Lower

Description

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):

-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
Example :

Input: n = 10, pick = 6
Output: 6

Solution

04/23/2020:

```
/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return       -1 if num is lower than the guess number
 *               1 if num is higher than the guess number
 *               otherwise return 0
 * int guess(int num);
 */

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

379. Design Phone Directory

Description

Design a Phone Directory which supports the following operations:

get: Provide a number which is not assigned to anyone.
check: Check if a number is available or not.
release: Recycle or release a number.

Example:

```
// Init a phone directory containing a total of 3 numbers: 0, 1, and 2.
PhoneDirectory directory = new PhoneDirectory(3);

// It can return any available phone number. Here we assume it returns 0.
directory.get();

// Assume it returns 1.
directory.get();

// The number 2 is available, so return true.
directory.check(2);

// It returns 2, the only number that is left.
directory.get();

// The number 2 is no longer available, so return false.
directory.check(2);

// Release number 2 back to the pool.
directory.release(2);

// Number 2 is available again, return true.
directory.check(2);
```

Constraints:

1 <= maxNumbers <= 10^4
0 <= number < maxNumbers
The total number of call of the methods is between [0 - 20000]

Solution

06/15/2020:

```
class PhoneDirectory {
private:
  vector<bool> numbers;
  int p, maxNumbers;
  stack<int> st;

public:
  /** Initialize your data structure here
      @param maxNumbers - The maximum numbers that can be stored in the phone
directory. */
  PhoneDirectory(int maxNumbers) {
    numbers.assign(maxNumbers, true);
    this->maxNumbers = maxNumbers;
    p = 0;
  }

  /** Provide a number which is not assigned to anyone.
      @return - Return an available number. Return -1 if none is available. */
  int get() {
    if (!st.empty()) {
      int cur = st.top(); st.pop();
      numbers[cur] = false;
      return cur;
    }
    while (!numbers[p] && p < maxNumbers) p++;
    if (p < maxNumbers) {
      numbers[p] = false;
      return p;
    } else {
      return -1;
    }
  }

  /** Check if a number is available or not. */
  bool check(int number) {
    return numbers[number];
  }

  /** Recycle or release a number. */
  void release(int number) {
    if (!numbers[number]) {
      numbers[number] = true;
      st.push(number);
    }
  }
};
```

```
/**
 * Your PhoneDirectory object will be instantiated and called as such:
 * PhoneDirectory* obj = new PhoneDirectory(maxNumbers);
 * int param_1 = obj->get();
 * bool param_2 = obj->check(number);
 * obj->release(number);
 */
```

```
class PhoneDirectory {
private:
  vector<bool> numbers;
  int p, maxNumbers;

public:
  /** Initialize your data structure here
      @param maxNumbers - The maximum numbers that can be stored in the phone
directory. */
  PhoneDirectory(int maxNumbers) {
    numbers.assign(maxNumbers, true);
    this->maxNumbers = maxNumbers;
    p = 0;
  }

  /** Provide a number which is not assigned to anyone.
      @return - Return an available number. Return -1 if none is available. */
  int get() {
    while (!numbers[p] && p < maxNumbers) p++;
    if (p < maxNumbers) {
      numbers[p] = false;
      return p;
    } else {
      return -1;
    }
  }

  /** Check if a number is available or not. */
  bool check(int number) {
    return numbers[number];
  }

  /** Recycle or release a number. */
  void release(int number) {
    numbers[number] = true;
    p = min(number, p);
  }
};
```

```
/**
 * Your PhoneDirectory object will be instantiated and called as such:
 * PhoneDirectory* obj = new PhoneDirectory(maxNumbers);
 * int param_1 = obj->get();
 * bool param_2 = obj->check(number);
 * obj->release(number);
 */
```

380. Insert Delete GetRandom O(1)

Description

Design a data structure that supports all following operations in average O(1) time.

insert(val): Inserts an item val to the set if not already present.
remove(val): Removes an item val from the set if present.
getRandom: Returns a random element from current set of elements. Each element must have the same probability of being returned.
Example:

```
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

Solution

06/12/2020:

```
class RandomizedSet {
private:
  unordered_map<int, int> cnt;
  vector<int> nums;
  int n;

public:
  /** Initialize your data structure here. */
  RandomizedSet() {
    nums.clear();
    cnt.clear();
    n = 0;
  }

  /** Inserts a value to the set. Returns true if the set did not already
contain the specified element. */
  bool insert(int val) {
    if (cnt.count(val) > 0) return false;
    nums.push_back(val);
    cnt[val] = n++;
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the
specified element. */
  bool remove(int val) {
    if (cnt.count(val) == 0) return false;
    cnt[nums[n - 1]] = cnt[val];
    swap(nums[cnt[val]], nums[n - 1]);
    cnt.erase(val);
    nums.pop_back();
    --n;
    return true;
  }

  /** Get a random element from the set. */
  int getRandom() {
    if (n == 0) return -1;
    return nums[rand() % n];
  }
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * RandomizedSet* obj = new RandomizedSet();
```

```
 * bool param_1 = obj->insert(val);
 * bool param_2 = obj->remove(val);
 * int param_3 = obj->getRandom();
 */
```

```
class RandomizedSet {
private:
  unordered_set<int> cnt;
  vector<int> nums;
  bool ok = false;

public:
  /** Initialize your data structure here. */
  RandomizedSet() {
    nums.clear();
    cnt.clear();
  }

  /** Inserts a value to the set. Returns true if the set did not already
contain the specified element. */
  bool insert(int val) {
    if (cnt.count(val) > 0) return false;
    cnt.insert(val); nums.push_back(val);
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the
specified element. */
  bool remove(int val) {
    if (cnt.count(val) == 0) return false;
    cnt.erase(val);
    return true;
  }

  /** Get a random element from the set. */
  int getRandom() {
    if (nums.empty()) return -1;
    int i;
    for (i = rand() % nums.size(); nums.size() != 0 && cnt.count(nums[i]) == 0;
i = nums.empty() ? 0 : rand() % nums.size())
      nums.erase(nums.begin() + i);
    return nums.empty() ? -1 : nums[i];
  }
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
```

```
 * RandomizedSet* obj = new RandomizedSet();
 * bool param_1 = obj->insert(val);
 * bool param_2 = obj->remove(val);
 * int param_3 = obj->getRandom();
 */
```

384. Shuffle an Array

Description

Shuffle a set of numbers without duplicates.

Example:

```
// Init an array with set 1, 2, and 3.
int[] nums = {1,2,3};
Solution solution = new Solution(nums);

// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3]
must equally likely to be returned.
solution.shuffle();

// Resets the array back to its original configuration [1,2,3].
solution.reset();

// Returns the random shuffling of array [1,2,3].
solution.shuffle();
```

Solution

05/25/2020:

```
class Solution {
private:
  vector<int> nums;
  std::function<int()> mt19937_rand;
public:
  Solution(vector<int>& nums) {
    this->nums = nums;
    mt19937_rand = bind(uniform_int_distribution<int>(0, INT_MAX),
mt19937(time(0)));
  }

  /** Resets the array to its original configuration and return it. */
  vector<int> reset() {
```

```
    return nums;
  }

  /** Returns a random shuffling of the array. */
  vector<int> shuffle() {
    vector<int> shuffled(nums);
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      int j = mt19937_rand() % (i + 1);
      swap(shuffled[i], shuffled[j]);
    }
    return shuffled;
  }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(nums);
 * vector<int> param_1 = obj->reset();
 * vector<int> param_2 = obj->shuffle();
 */
```

```
class Solution {
private:
  vector<int> nums;
  mt19937 mt19937_rand;
  unsigned seed;
public:
  Solution(vector<int>& nums) {
    this->nums = nums;
    seed = std::chrono::system_clock::now().time_since_epoch().count();
    mt19937_rand.seed(seed);
  }

  /** Resets the array to its original configuration and return it. */
  vector<int> reset() {
    seed = std::chrono::system_clock::now().time_since_epoch().count();
    mt19937_rand.seed(seed);
    return nums;
  }

  /** Returns a random shuffling of the array. */
  vector<int> shuffle() {
    vector<int> shuffled(nums);
    int n = nums.size();
    for (int i = 0; i < n; ++i) {
      int j = mt19937_rand() % (i + 1);
      swap(shuffled[i], shuffled[j]);
```

```
    }
    return shuffled;
  }
};
```

387. First Unique Character in a String

Description

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
Note: You may assume the string contain only lowercase letters.

Solution

05/05/2020:

```
class Solution {
public:
  int firstUniqChar(string s) {
    unordered_map<char, int> mp;
    for (auto& c : s) ++mp[c];
    int n = s.size();
    for (int i = 0; i < n; ++i)
      if (mp[s[i]] == 1)
        return i;
    return -1;
  }
};
```

389. Find the Difference

Description

Given two strings s and t which consist of only lowercase letters.

String t is generated by random shuffling string s and then add one more letter at a random position.

Find the letter that was added in t.

Example:

Input:
s = "abcd"
t = "abcde"

Output:
e

Explanation:
'e' is the letter that was added.

Solution

05/17/2020:

```
class Solution {
public:
  char findTheDifference(string s, string t) {
    vector<int> cntS(26, 0);
    vector<int> cntT(26, 0);
    for (auto& c : s) {
      ++cntS[c - 'a'];
    }
    for (auto& c : t) {
      ++cntT[c - 'a'];
    }
    for (int i = 0; i < 26; ++i) {
      if (cntS[i] < cntT[i]) {
        return i + 'a';
      }
    }
    return 'a';
  }
};
```

392. Is Subsequence

Description

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~= 500,000) string, and s is a short string (<=100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:
s = "abc", t = "ahbgdc"

Return true.

Example 2:
s = "axc", t = "ahbgdc"

Return false.

Follow up:
If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

Credits:
Special thanks to @pbrother for adding this problem and creating all test cases.

Solution

01/14/2020 (Dynamic Programming, Memory Limit Exceeded):

```
class Solution {
public:
  bool isSubsequence(string s, string t) {
    if (s.size() >= t.size()) return s == t;
    return s.back() == t.back() ? isSubsequence(s.substr(0, s.size() - 1),
t.substr(0, t.size() - 1)) :
            isSubsequence(s, t.substr(0, t.size() - 1));
  }
};
```

01/14/2020 (Dynamic Programming (bottom-up), Two pointers):

```cpp
class Solution {
public:
  bool isSubsequence(string s, string t) {
    int ps = 0, pt = 0;
    for (; ps < s.size() && pt < t.size(); ++pt)
      if (s[ps] == t[pt]) ++ps;
    return ps == s.size();
  }
};
```

06/09/2020:

```cpp
class Solution {
public:
  bool isSubsequence(string s, string t) {
    auto sit = s.begin(), tit = t.begin();
    for (; sit != s.end() && tit != t.end(); ++tit)
      if (*sit == *tit) ++sit;
    return sit == s.end();
  }
};
```

402. Remove K Digits

Description

Given a non-negative integer num represented as a string, remove k digits from the number so that the new number is the smallest possible.

Note:
The length of num is less than 10002 and will be ≥ k.
The given num does not contain any leading zero.
Example 1:

Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.
Example 2:

Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.
Example 3:

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing
which is 0.
```

Solution

05/12/2020:

```
class Solution {
public:
  string removeKdigits(string num, int k) {
    deque<char> q;
    string ret;
    for (int i = 0; i < num.size(); ++i) {
      while (!q.empty() && q.back() > num[i] && k-- > 0) q.pop_back();
      q.push_back(num[i]);
    }
    while (!q.empty() && k-- > 0) q.pop_back();
    while (!q.empty() && q.front() == '0') q.pop_front();
    while (!q.empty()) ret += q.front(), q.pop_front();
    return ret.empty() ? string(1, '0') : ret;
  }
};
```

```
class Solution {
public:
  string removeKdigits(string num, int k) {
    string ret;
    for (int i = 0; i < num.size(); ++i) {
      while (!ret.empty() && ret.back() > num[i] && k-- > 0)
        ret.pop_back();
      if (!ret.empty() || num[i] != '0')
        ret += num[i];
    }
    while (!ret.empty() && k-- > 0)
      ret.pop_back();
    return ret.empty() ? "0" : ret;
  }
};
```

404. Sum of Left Leaves

Description

Find the sum of all left leaves in a given binary tree.

Example:

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

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.

Solution

Discussion

Recursive:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int sumOfLeftLeaves(TreeNode* root, bool isLeft = false) {
    if (root == nullptr) return 0;
    int s = !root->left && !root->right && isLeft ? root->val : 0;
    return s + sumOfLeftLeaves(root->left, true) + sumOfLeftLeaves(root->right,
false);
  }
};
```

Iterative:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
```

```
 *      TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
  int sumOfLeftLeaves(TreeNode* root) {
    if (root == nullptr) return 0;
    int s = 0;
    stack<pair<TreeNode*, bool>> st;
    st.emplace(root, false);
    while (!st.empty()) {
      pair<TreeNode*, bool> cur = st.top(); st.pop();
      s += cur.first->left == nullptr && cur.first->right == nullptr &&
cur.second ? cur.first->val : 0;
      if (cur.first->left) st.emplace(cur.first->left, true);
      if (cur.first->right) st.emplace(cur.first->right, false);
    }
    return s;
  }
};
```

406. Queue Reconstruction by Height

Description

Suppose you have a random list of people standing in a queue. Each person is described by a pair of integers (h, k), where h is the height of the person and k is the number of people in front of this person who have a height greater than or equal to h. Write an algorithm to reconstruct the queue.

Note:
The number of people is less than 1,100.

Example

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

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

Solution

06/06/2020:

```
class Solution {
public:
  vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
    if (people.empty() || people[0].empty()) return people;
    sort(people.begin(), people.end(), [](vector<int>& p1, vector<int>& p2) {
      if (p1[0] == p2[0]) return p1[1] < p2[1];
      return p1[0] > p2[0];
    });
    list<vector<int>> ret;
    for (int i = 0; i < (int)people.size(); ++i)
      ret.insert(next(ret.begin(), people[i][1]), people[i]);
    return vector<vector<int>>(ret.begin(), ret.end());
  }
};
```

```
class Solution {
public:
  vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
    if (people.empty() || people[0].empty()) return people;
    sort(people.begin(), people.end(), [](vector<int>& p1, vector<int>& p2) {
      if (p1[0] == p2[0]) return p1[1] < p2[1];
      return p1[0] > p2[0];
    });
    vector<vector<int>> ret;
    for (int i = 0; i < (int)people.size(); ++i)
      ret.insert(ret.begin() + people[i][1], people[i]);
    return ret;
  }
};
```

409. Longest Palindrome

Description

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example "Aa" is not considered a palindrome here.

Note:
Assume the length of given string will not exceed 1,010.

Example:

```
Input:
"abccccdd"

Output:
7

Explanation:
One longest palindrome that can be built is "dccaccd", whose length is 7.
```

Solution

05/18/2020:

```
class Solution {
public:
  int longestPalindrome(string s) {
    vector<int> cnt(128, 0);
    for (auto& c : s) {
      ++cnt[c];
    }
    bool odd = false;
    int even = 0;
    for (int i = 0; i < 128; ++i) {
      if (cnt[i] % 2 == 0) {
        even += cnt[i];
      } else {
        odd = true;
        even += max(cnt[i] - 1, 0);
      }
    }
    return odd ? even + 1 : even;
  }
};
```

415. Add Strings

Description

Given two non-negative integers num1 and num2 represented as string, return the sum of num1 and num2.

Note:

The length of both num1 and num2 is < 5100.
Both num1 and num2 contains only digits 0-9.
Both num1 and num2 does not contain any leading zero.
You must not use any built-in BigInteger library or convert the inputs to integer directly.

Solution

06/09/2020:

```
class Solution {
public:
  string addStrings(string num1, string num2) {
    int carry = 0;
    int n1 = num1.size(), n2 = num2.size();
    string ret;
    for (int i1 = n1 - 1, i2 = n2 - 1; i1 >= 0 || i2 >= 0 || carry > 0; --i1, --i2)
{
      if (i1 >= 0) carry += num1[i1] - '0';
      if (i2 >= 0) carry += num2[i2] - '0';
      ret.push_back(carry % 10 + '0');
      carry /= 10;
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

429. N-ary Tree Level Order Traversal

Description

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

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

Example 1:

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



Input: root =
[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null
,null,14]
Output: [[1],[2,3,4,5],[6,7,8,9,10],[11,12,13],[14]]


Constraints:

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

Solution

04/26/2020:

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {
public:
  vector<vector<int>> levelOrder(Node* root) {
    if (root == nullptr) return {};
```

```
    queue<Node*> q;
    q.push(root);
    vector<vector<int>> ret;
    ret.push_back({root->val});
    while (!q.empty()) {
      int n = q.size();
      vector<int> level;
      for (int i = 0; i < n; ++i) {
        Node* cur = q.front(); q.pop();
        for (auto& c : cur->children) {
          level.push_back(c->val);
          q.push(c);
        }
      }
      if (!level.empty()) ret.push_back(level);
    }
    return ret;
  }
};
```

438. Find All Anagrams in a String

Description

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

Strings consists of lowercase English letters only and the length of both strings s and p will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input:
s: "cbaebabacd" p: "abc"

Output:
[0, 6]

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc". The substring with start index = 6 is "bac", which is an anagram of "abc". Example 2:

Input:

s: "abab" p: "ab"

Output:
[0, 1, 2]

Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab". The substring with start index = 1 is "ba", which is an anagram of "ab". The substring with start index = 2 is "ab", which is an anagram of "ab".

Solution

05/16/2020:

```
class Solution {
public:
  vector<int> findAnagrams(string s, string p) {
    vector<int> sch(26, 0), pch(26, 0), ret;
    for (auto& c : p) ++pch[c - 'a'];
    int m = p.size(), n = s.size();
    if (m > n) return {};
    for (int i = 0; i < m - 1; ++i) ++sch[s[i] - 'a'];
    bool ok = false;
    for (int i = m - 1; i < n; ++i) {
      if (ok) {
        if (s[i] == s[i - m]) {
          ret.push_back(i - m + 1);
        } else {
          --sch[s[i - m] - 'a'];
          ++sch[s[i] - 'a'];
          ok = false;
        }
      } else {
        if (i >= m) --sch[s[i - m] - 'a'];
        ++sch[s[i] - 'a'];
        ok = true;
        for (int j = 0; j < 26; ++j) {
          if (pch[j] != sch[j]) {
            ok = false;
            break;
          }
        }
        if (ok) ret.push_back(i - m + 1);
      }
    }
    return ret;
  }
};
```

```
class Solution {
public:
  vector<int> findAnagrams(string s, string p) {
    vector<int> sch(26, 0), pch(26, 0), ret;
    for (auto& c : p) ++pch[c - 'a'];
    int m = p.size(), n = s.size();
    for (int i = 0; i < n; ++i) {
      if (i - m >= 0) --sch[s[i - m] - 'a'];
      ++sch[s[i] - 'a'];
      bool ok = true;
      for (int j = 0; j < 26; ++j) {
        if (pch[j] != sch[j]) {
          ok = false;
          break;
        }
      }
      if (ok) ret.push_back(i - m + 1);
    }
    return ret;
  }
};
```

447. Number of Boomerangs

Description

Given n points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points (i, j, k) such that the distance between i and j equals the distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and coordinates of points are all in the range [-10000, 10000] (inclusive).

Example:

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

Output:
2

Explanation:
The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]

Solution

05/20/2020:

```cpp
class Solution {
public:
  int numberOfBoomerangs(vector<vector<int>>& points) {
    if (points.empty() || points[0].empty()) return 0;
    int n = points.size(), cnt = 0;
    vector<unordered_map<double, int>> mp(n);
    for (int i = 0; i < n - 1; ++i) {
      for (int j = i + 1; j < n; ++j) {
        int x1 = points[i][0], x2 = points[j][0];
        int y1 = points[i][1], y2 = points[j][1];
        double d = hypot(x1 - x2, y1 - y2);
        ++mp[i][d];
        ++mp[j][d];
      }
    }
    for (int i = 0; i < n; ++i)
      for (auto& m : mp[i])
        cnt += nchoosek(m.second, 2) * 2;
    return cnt;
  }

  int nchoosek(int n, int k) {
    if (n < k) return 0;
    // n! / (k!) (n - k)!;
    // n * (n - 1) * ... * (n - k + 1) / k!
    long long num = 1, dem = 1;
    int kk = min(n - k, k);
    for (int i = 1; i <= kk; ++i) {
      num *= (n - i + 1);
      dem *= i;
    }
    return num / dem;
  }
};
```

451. Sort Characters By Frequency

Description

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

Example 2:

Input:
"cccaaa"

Output:
"cccaaa"

Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.
Note that "cacaca" is incorrect, as the same characters must be together.

Example 3:

Input:
"Aabb"

Output:
"bbAa"

Explanation:
"bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.

Solution

05/20/2020:

```cpp
class Solution {
public:
  string frequencySort(string s) {
    unordered_map<char, int> mp;
    for (auto& c : s) ++mp[c];
    vector<pair<int, char>> str;
    for (auto& m : mp) str.emplace_back(m.second, m.first);
    sort(str.begin(), str.end(), greater<pair<int, char>>());
    string ret = "";
    for (auto& s : str) ret += string(s.first, s.second);
    return ret;
  }
};
```

468. Validate IP Address

Description

Write a function to check whether an input string is a valid IPv4 address or IPv6 address or neither.

IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots ("."), e.g.,172.16.254.1;

Besides, leading zeros in the IPv4 is invalid. For example, the address 172.16.254.01 is invalid.

IPv6 addresses are represented as eight groups of four hexadecimal digits, each group representing 16 bits. The groups are separated by colons (":"). For example, the address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 is a valid one. Also, we could omit some leading zeros among four hexadecimal digits and some low-case characters in the address to upper-case ones, so 2001:db8:85a3:0:0:8A2E:0370:7334 is also a valid IPv6 address(Omit leading zeros and using upper cases).

However, we don't replace a consecutive group of zero value with a single empty group using two consecutive colons (::) to pursue simplicity. For example, 2001:0db8:85a3::8A2E:0370:7334 is an invalid IPv6 address.

Besides, extra leading zeros in the IPv6 is also invalid. For example, the address 02001:0db8:85a3:0000:0000:8a2e:0370:7334 is invalid.

Note: You may assume there is no extra space or special characters in the input string.

Example 1:
Input: "172.16.254.1"

Output: "IPv4"

Explanation: This is a valid IPv4 address, return "IPv4".
Example 2:
Input: "2001:0db8:85a3:0:0:8A2E:0370:7334"

Output: "IPv6"

Explanation: This is a valid IPv6 address, return "IPv6".
Example 3:
Input: "256.256.256.256"

Output: "Neither"

Explanation: This is neither a IPv4 address nor a IPv6 address.

Solution

06/16/2020:

```
class Solution {
public:
  string validIPAddress(string IP) {
    int n = IP.size();
    vector<string> ipv4 = split(IP, '.');
    vector<string> ipv6 = split(IP, ':');
    bool isIpv4 = ipv4.size() == 4 && isdigit(IP.front()) && isdigit(IP.back());
    bool isIpv6 = ipv6.size() == 8 && (isalpha(IP.front()) ||
isdigit(IP.front())) && (isalpha(IP.back()) || isdigit(IP.back()));
    bool hasLeadingZero = false;
    if (isIpv4) {
      for (auto& s : ipv4) {
        isIpv4 = isIpv4 && s.size() <= 3 && s.size() > 0;
        for (auto& c : s) isIpv4 = isIpv4 && isdigit(c);
        if (!isIpv4) break;
        int n = stoi(s);
        isIpv4 = isIpv4 && (0 <= n && n <= 255 && to_string(n) == s);
      }
    } else if (isIpv6) {
      for (auto& s : ipv6) {
        isIpv6 = isIpv6 && s.size() > 0 && s.size() < 5;
        for (auto& c : s) isIpv6 = isIpv6 && (('0' <= c && c <= '9') || ('a' <=
c && c <= 'f') || ('A' <= c && c <= 'F'));
        if (!isIpv6) break;
```

```
        }
      }
      return isIpv4 ? "IPv4" : isIpv6 ? "IPv6" : "Neither";
    }

    vector<string> split(string IP, char delimiter) {
      istringstream iss(IP);
      string s;
      vector<string> ret;
      while (getline(iss, s, delimiter)) ret.push_back(s);
      return ret;
    }
};
```

470. Implement Rand10() Using Rand7()

Description

Given a function rand7 which generates a uniform random integer in the range 1 to 7, write a function rand10 which generates a uniform random integer in the range 1 to 10.

Do NOT use system's Math.random().

Example 1:

Input: 1
Output: [7]
Example 2:

Input: 2
Output: [8,4]
Example 3:

Input: 3
Output: [8,1,10]

Note:

rand7 is predefined.
Each testcase has one argument: n, the number of times that rand10 is called.

Follow up:

What is the expected value for the number of calls to rand7() function?
Could you minimize the number of calls to rand7()?

Solution

05/27/2020:

```
// The rand7() API is already defined for you.
// int rand7();
// @return a random integer in the range 1 to 7

class Solution {
public:
  int rand10() {
    int s = 41;
    while (s > 40) s = (rand7() - 1) * 7 + rand7();
    return (s - 1) % 10 + 1;
  }
};
```

475. Heaters

Description

Winter is coming! Your first job during the contest is to design a standard heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find out minimum radius of heaters so that all houses could be covered by those heaters.

So, your input will be the positions of houses and heaters seperately, and your expected output will be the minimum radius standard of heaters.

Note:

Numbers of houses and heaters you are given are non-negative and will not exceed 25000.
Positions of houses and heaters you are given are non-negative and will not exceed 10^9.
As long as a house is in the heaters' warm radius range, it can be warmed.
All the heaters follow your radius standard and the warm radius will the same.

Example 1:

Input: [1,2,3],[2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.

Example 2:

Input: [1,2,3,4],[1,4]
Output: 1
Explanation: The two heater was placed in the position 1 and 4. We need to use radius 1 standard, then all the houses can be warmed.

Solution

04/23/2020:

```
class Solution {
public:
  int findRadius(vector<int>& houses, vector<int>& heaters) {
    if ((int)houses.size() <= 0) return 0;
    int n = heaters.size();
    if (n <= 0) return INT_MAX;
    sort(heaters.begin(), heaters.end());
    for (auto& h : houses) {
      int lo = 0, hi = n - 1;
      while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (heaters[mid] <= h) {
          lo = mid + 1;
        } else {
          hi = mid - 1;
        }
      }
      int left = INT_MAX, right = INT_MAX;
      if (lo - 1 >= 0) left = min(right, h - heaters[lo - 1]);
      if (lo < n) right = min(left, heaters[lo] - h);
      h = min(left, right);
    }
    return *max_element(houses.begin(), houses.end());
  }
};
```

482. License Key Formatting

Description

You are given a license key represented as a string S which consists only alphanumeric character and dashes. The string is separated into N+1 groups by N dashes.

Given a number K, we would want to reformat the strings such that each group contains exactly K characters, except for the first group which could be shorter than K, but still must contain at least one character. Furthermore, there must be a dash inserted between two groups and all lowercase letters should be converted to uppercase.

Given a non-empty string S and a number K, format the string according to the rules described above.

Example 1:

Input: S = "5F3Z-2e-9-w", K = 4

Output: "5F3Z-2E9W"

Explanation: The string S has been split into two parts, each part has 4 characters.
Note that the two extra dashes are not needed and can be removed.
Example 2:

Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

Explanation: The string S has been split into three parts, each part has 2 characters except the first part as it could be shorter as mentioned above.
Note:

The length of string S will not exceed 12,000, and K is a positive integer.
String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-).
String S is non-empty.

Solution

02/05/2020:

```
class Solution {
public:
```

```
  string licenseKeyFormatting(string S, int K) {
    string ret;
    int cnt = 0;
    for (auto it = S.rbegin(); it != S.rend(); ++it) {
      if (*it == '-') {
        continue;
      } else {
        ret += toupper(*it);
      }
      if (++cnt % K == 0) {
        cnt == 0;
        ret += '-';
      }
    }
    if (ret.back() == '-') {
      ret.pop_back();
    }
    reverse(ret.begin(), ret.end());
    return ret;
  }
};
```

498. Diagonal Traverse

Description

Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example:

Input:

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

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

Explanation:

Note:

The total number of elements of the given matrix will not exceed 10,000.

Solution

05/20/2020:

```
class Solution {
public:
  vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return {};
    int m = matrix.size(), n = matrix[0].size();
    vector<int> ret;
    bool ascending = false;
    for (int k = 0; k < m + n; ++k) {
      if (ascending)
        for (int i = max(0, k - n + 1); i <= min(m - 1, k); ++i)
ret.push_back(matrix[i][k - i]);
      else
        for (int i = min(m - 1, k); i >= max(0, k - n + 1); --i)
ret.push_back(matrix[i][k - i]);
      ascending = !ascending;
    }
    return ret;
  }
};
```

510. Inorder Successor in BST II

Description

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

If that node has no in-order successor, return null.

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

You will have direct access to the node but not to the root of the tree. Each node will have a reference to its parent node. Below is the definition for Node:

```
class Node {
    public int val;
    public Node left;
```

```
    public Node right;
    public Node parent;
}
```

Follow up:

Could you solve it without looking up any of the node's values?

Example 1:

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

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

Input: tree = [15,6,18,3,7,17,20,2,4,null,13,null,null,null,null,null,null,null,null,9], node = 15
Output: 17
Example 4:

Input: tree = [15,6,18,3,7,17,20,2,4,null,13,null,null,null,null,null,null,null,null,9], node = 13
Output: 15
Example 5:

Input: tree = [0], node = 0
Output: null

Constraints:

-10^5 <= Node.val <= 10^5
1 <= Number of Nodes <= 10^4

All Nodes will have unique values.

Solution

05/24/2020:

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* parent;
};
*/

class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    if (node->right != nullptr) return leftmostChild(node->right);
    Node* parent = smallestParentGreaterThanNode(node);
    return parent == nullptr || parent->val < node->val ? nullptr : parent;
  }

  Node* leftmostChild(Node* root) {
    if (root == nullptr) return nullptr;
    if (root->left == nullptr) return root;
    return leftmostChild(root->left);
  }

  Node* smallestParentGreaterThanNode(Node* node) {
    if (node == nullptr || node->parent == nullptr) return nullptr;
    if (node->parent->val > node->val) return node->parent;
    return smallestParentGreaterThanNode(node->parent);
  }
};
```

```
class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    if (node->right != nullptr) return leftmost(node->right);
    Node* parent = node->parent;
    while (true) {
      if (parent != nullptr && parent->val < node->val && parent->parent !=
nullptr) {
        parent = parent->parent;
```

```
      } else {
        break;
      }
    }
    return parent != nullptr && parent->val < node->val ? nullptr : parent;
  }

  Node* leftmost(Node* root) {
    if (root == nullptr) return nullptr;
    if (root->left == nullptr) return root;
    return leftmost(root->left);
  }
};
```

```
class Solution {
public:
  Node* inorderSuccessor(Node* node) {
    Node* root = node;
    while (true) {
      if (root->parent != nullptr)
        root = root->parent;
      else
        break;
    }
    vector<Node*> inorder = inorderTraversal(root);
    auto it = find(inorder.begin(), inorder.end(), node);
    return it != inorder.end() && it + 1 != inorder.end() ? *(it + 1) : nullptr;
  }

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

518. Coin Change 2

Description

You are given coins of different denominations and a total amount of money. Write a function to compute the number of combinations that make up that amount. You may assume that you have infinite number of each kind of coin.

Example 1:

Input: amount = 5, coins = [1, 2, 5]
Output: 4
Explanation: there are four ways to make up the amount:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
Example 2:

Input: amount = 3, coins = [2]
Output: 0
Explanation: the amount of 3 cannot be made up just with coins of 2.
Example 3:

Input: amount = 10, coins = [10]
Output: 1

Note:

You can assume that

0 <= amount <= 5000
1 <= coin <= 5000
the number of coins is less than 500
the answer is guaranteed to fit into signed 32-bit integer

Solution

05/27/2020:

```
class Solution {
public:
  int change(int amount, vector<int>& coins) {
    vector<int> dp(amount + 1, 0);
    dp[0] = 1;
    for (auto& c : coins) {
      for (int i = c; i <= amount; ++i) {
        if (i - c >= 0) {
          dp[i] += dp[i - c];
```

```
        }
      }
    }
    return dp.back();
  }
};
```

525. Contiguous Array

Description

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Example 1:
Input: [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with equal number of 0 and 1.
Example 2:
Input: [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.
Note: The length of the given binary array will not exceed 50,000.

Solution

05/26/2020:

```
class Solution {
public:
  int findMaxLength(vector<int>& nums) {
    if (nums.empty()) return 0;
    unordered_map<int, int> firstOccurence{ {0, -1} };
    int n = nums.size(), ret = 0, prefix = 0;
    for (int i = 0; i < n; ++i) {
      prefix = prefix + (nums[i] == 0 ? 1 : -1);
      if (firstOccurence.count(prefix) == 0)
        firstOccurence[prefix] = i;
      else
        ret = max(ret, i - firstOccurence[prefix]);
    }
    return ret;
  }
```

```
};
```

528. Random Pick with Weight

Description

Given an array w of positive integers, where w[i] describes the weight of index i, write a function pickIndex which randomly picks an index in proportion to its weight.

Note:

1 <= w.length <= 10000
1 <= w[i] <= 10^5
pickIndex will be called at most 10000 times.
Example 1:

Input:
["Solution","pickIndex"]
[[[1]],[]]
Output: [null,0]
Example 2:

Input:
["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"]
[[[1,3]],[],[],[],[],[]]
Output: [null,0,1,1,1,0]
Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. Solution's constructor has one argument, the array w. pickIndex has no arguments. Arguments are always wrapped with a list, even if there aren't any.

Solution

06/05/2020:

```
class Solution {
private:
  vector<int> weights;
  mt19937 rand;

public:
  Solution(vector<int>& w) {
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
```

```
    rand.seed(seed);
    weights.resize(w.size());
    partial_sum(w.begin(), w.end(), weights.begin(), plus<int>());
  }

  int pickIndex() {
    long long n = (double) rand() / rand.max() * weights.back();
    return upper_bound(weights.begin(), weights.end(), n) - weights.begin();
  }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(w);
 * int param_1 = obj->pickIndex();
 */
```

535. Encode and Decode TinyURL

Description

Note: This is a companion problem to the System Design problem: Design TinyURL (https://leetcode.com/discuss/interview-question/124658/Design-a-URL-Shortener-(-TinyURL-)-System/).
TinyURL is a URL shortening service where you enter a URL such as https://leetcode.com/problems/design-tinyurl and it returns a short URL such as http://tinyurl.com/4e9iAk.

Design the encode and decode methods for the TinyURL service. There is no restriction on how your encode/decode algorithm should work. You just need to ensure that a URL can be encoded to a tiny URL and the tiny URL can be decoded to the original URL.

Solution

05/20/2020:

```
class Solution {
private:
  unordered_map<string, string> decodeTable;
  unordered_map<string, string> encodeTable;
  string base = "http://tinyurl.com/";
  const int MOD = 1e9 + 7;

  int hash(string s) {
```

```
    long long h = 1;
    for (auto& c : s) h = (h * 131 + c) % MOD;
    return h;
  }

public:

  // Encodes a URL to a shortened URL.
  string encode(string longUrl) {
    if (encodeTable.count(longUrl) == 0) {
      string shortUrl = base + to_string(hash(longUrl));
      encodeTable[longUrl] = shortUrl;
      decodeTable[shortUrl] = longUrl;
    }
    return encodeTable[longUrl];
  }

  // Decodes a shortened URL to its original URL.
  string decode(string shortUrl) {
    return decodeTable[shortUrl];
  }
};

// Your Solution object will be instantiated and called as such:
// Solution solution;
//  solution.decode(solution.encode(url));
```

540. Single Element in a Sorted Array

Description

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Find this single element that appears only once.

Follow up: Your solution should run in O(log n) time and O(1) space.

Example 1:

Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
Example 2:

Input: nums = [3,3,7,7,10,11,11]

```
Output: 10


Constraints:

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

Solution

05/12/2020:

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

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

```
      }
    }
    return nums[lo];
  }
};
```

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

542. 01 Matrix

Description

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

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

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

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

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

Note:

The number of elements of the given matrix will not exceed 10,000.
There are at least one 0 in the given matrix.
The cells are adjacent in only four directions: up, down, left and right.

Solution

06/09/2020: BFS:

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

    while (!q.empty()) {
      pair<int, int> cur = q.front(); q.pop();
      int i = cur.first, j = cur.second;
      for (int d = 0; d < 4; ++d) {
        int ni = i + dir[d][0], nj = j + dir[d][1];
        if (ni < 0 || ni >= m || nj < 0 || nj >=n) continue;
        if (matrix[ni][nj] > matrix[i][j] + 1) {
          matrix[ni][nj] = matrix[i][j] + 1;
          q.emplace(ni, nj);
        }
      }
    }
    return matrix;
  }
};
```

```cpp
class Solution {
public:
  vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
    if (matrix.empty() || matrix[0].empty()) return matrix;
    int m = matrix.size(), n = matrix[0].size();
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (matrix[i][j] == 0) continue;
        int cur_min = INT_MAX;
        queue<pair<int, int>> q;
        q.emplace(i, j);
        bool found = false;
        unordered_set<string> visited;
        while (!q.empty() && !found) {
          int sz = q.size();
          for (int s = 0; s < sz && !found; ++s) {
            pair<int, int> cur = q.front(); q.pop();
            string key = to_string(cur.first) + "," + to_string(cur.second);
            if (visited.count(key) > 0) continue;
            for (int d = 0; d < 4; ++d) {
              int ni = cur.first + dir[d][0], nj = cur.second + dir[d][1];
              string key = to_string(i) + "," + to_string(j);
              if (ni < 0 || ni >= m || nj < 0 || nj >= n) continue;
              if (matrix[ni][nj] == 0) {
                found = true;
                matrix[i][j] = (abs(i - ni) + abs(j - nj));
                break;
              } else {
                q.emplace(ni, nj);
              }
            }
            visited.insert(key);
          }
        }
      }
    }
    return matrix;
  }
};
```

Dynamic Programming:

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

```
    int dir[4][2] = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };
    queue<pair<int, int >> q;
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j)
        if (matrix[i][j] != 0)
          matrix[i][j] = INT_MAX;
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) {
        if (i > 0 && matrix[i - 1][j] != INT_MAX) matrix[i][j] = min(matrix[i]
[j], matrix[i - 1][j] + 1);
        if (j > 0 && matrix[i][j - 1] != INT_MAX) matrix[i][j] = min(matrix[i]
[j], matrix[i][j - 1] + 1);
      }
    }
    for (int i = m - 1; i >= 0; --i) {
      for (int j = n - 1; j >= 0; --j) {
        if (i < m - 1 && matrix[i + 1][j] != INT_MAX) matrix[i][j] =
min(matrix[i][j], matrix[i + 1][j] + 1);
        if (j < n - 1 && matrix[i][j + 1] != INT_MAX) matrix[i][j] =
min(matrix[i][j], matrix[i][j + 1] + 1);
      }
    }
    return matrix;
  }
};
```

551. Student Attendance Record I

Description

You are given a string representing an attendance record for a student. The record only contains the following three characters:
'A' : Absent.
'L' : Late.
'P' : Present.
A student could be rewarded if his attendance record doesn't contain more than one 'A' (absent) or more than two continuous 'L' (late).

You need to return whether the student could be rewarded according to his attendance record.

Example 1:

Input: "PPALLP"
Output: True
;xample 2:

```
Input: "PPALLL"
Output: False
```

Solution

02/05/2020:

```
class Solution {
public:
  bool checkRecord(string s) {
    int cntA = 0, cntLL = 0;
    for (auto i = 0; i < s.size(); ++i) {
      if (s[i] == 'A') {
        ++cntA;
      } else if (s[i] == 'L') {
        if (i > 0 && s[i - 1] == 'L') {
          ++cntLL;
        } else {
          cntLL = 1;
        }
      }
      if (cntA > 1 || cntLL > 2) {
        return false;
      }
    }
    return true;
  }
};
```

567. Permutation in String

Description

Given two strings s1 and s2, write a function to return true if s2 contains the permutation of s1. In other words, one of the first string's permutations is the substring of the second string.

Example 1:

```
Input: s1 = "ab" s2 = "eidbaooo"
Output: True
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

Input:s1= "ab" s2 = "eidboaoo"
Output: False

Note:

The input strings only contain lower case letters.
The length of both given strings is in range [1, 10,000].

Solution

05/18/2020:

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