目录

[201. Bitwise AND of Numbers Range 4](#_Toc36481255)

[202. Happy Number 6](#_Toc36481256)

[203. Remove Linked List Elements 8](#_Toc36481257)

[204. Count Primes(●ˇ∀ˇ●) 10](#_Toc36481258)

[205. Isomorphic Strings 12](#_Toc36481259)

[206. Reverse Linked List 14](#_Toc36481260)

[207. Course Schedule 16](#_Toc36481261)

[208. Implement Trie (Prefix Tree)(●ˇ∀ˇ●) 18](#_Toc36481262)

[209. Minimum Size Subarray Sum 20](#_Toc36481263)

[210. Course Schedule II 23](#_Toc36481264)

[211. Add and Search WordData structure design(●ˇ∀ˇ●) 25](#_Toc36481265)

[212. Word Search II(●ˇ∀ˇ●) 27](#_Toc36481266)

[213. House Robber II(●ˇ∀ˇ●) 30](#_Toc36481267)

[214. Shortest Palindrome(●ˇ∀ˇ●) 32](#_Toc36481268)

[215. Kth Largest Element in an Array(●ˇ∀ˇ●) 35](#_Toc36481269)

[216. Combination Sum III 37](#_Toc36481270)

[217. Contains Duplicate 39](#_Toc36481271)

[218. The Skyline Problem(●ˇ∀ˇ●) 41](#_Toc36481272)

[219. Contains Duplicate II 43](#_Toc36481273)

[220. Contains Duplicate III(●ˇ∀ˇ●) 45](#_Toc36481274)

[221. Maximal Square(●ˇ∀ˇ●) 47](#_Toc36481275)

[222. Count Complete Tree Nodes(●ˇ∀ˇ●) 49](#_Toc36481276)

[223. Rectangle Area 51](#_Toc36481277)

[224. Basic Calculator 53](#_Toc36481278)

[225. Implement Stack using Queues 56](#_Toc36481279)

[226. Invert Binary Tree 58](#_Toc36481280)

[227. Basic Calculator II(●ˇ∀ˇ●) 60](#_Toc36481281)

[228. Summary Ranges 62](#_Toc36481282)

[229. Majority Element II 64](#_Toc36481283)

[230. Kth Smallest Element in a BST 66](#_Toc36481284)

[231. Power of Two 68](#_Toc36481285)

[232. Implement Queue using Stacks 70](#_Toc36481286)

[233. Number of Digit One 72](#_Toc36481287)

[234. Palindrome Linked List 74](#_Toc36481288)

[235. Lowest Common Ancestor of a Binary Search Tree 76](#_Toc36481289)

[236. Lowest Common Ancestor of a Binary Tree(●ˇ∀ˇ●) 78](#_Toc36481290)

[237. Delete Node in a Linked List 81](#_Toc36481291)

[238. Product of Array Except Self 83](#_Toc36481292)

[239. Sliding Window Maximum(●ˇ∀ˇ●) 85](#_Toc36481293)

[240. Search a 2D Matrix II(●ˇ∀ˇ●) 87](#_Toc36481294)

[241. Different Ways to Add Parentheses(●ˇ∀ˇ●) 89](#_Toc36481295)

[242. Valid Anagram 92](#_Toc36481296)

[243.Shortest Word Distance 94](#_Toc36481297)

[244.Shortest Word Distance II 95](#_Toc36481298)

[245.Shortest Word Distance III 97](#_Toc36481299)

[246.Strobogrammatic Number 99](#_Toc36481300)

[247.Strobogrammatic Number II 101](#_Toc36481301)

[248.Strobogrammatic Number III 103](#_Toc36481302)

[249.Group Shifted Strings 105](#_Toc36481303)

[250.Count Univalue Subtrees 107](#_Toc36481304)

[251. Flatten 2D Vector 109](#_Toc36481305)

[252. Meeting Rooms 112](#_Toc36481306)

[253. Meeting Rooms II 113](#_Toc36481307)

[254. Factor Combinations 116](#_Toc36481308)

[255. Verify Preorder Sequence in Binary Search Tree 118](#_Toc36481309)

[256. Paint House 121](#_Toc36481310)

[257. Binary Tree Paths 123](#_Toc36481311)

[258. Add Digits 125](#_Toc36481312)

[259. Sum Smaller 127](#_Toc36481313)

[260. Single Number III 129](#_Toc36481314)

[261. Graph Valid Tree 131](#_Toc36481315)

[262. Trips and Users（SQL） 134](#_Toc36481316)

[263. Ugly Number 136](#_Toc36481317)

[264. Ugly Number II 138](#_Toc36481318)

[265. Paint House II 141](#_Toc36481319)

[266. Palindrome Permutation 143](#_Toc36481320)

[267. Palindrome Permutation II 145](#_Toc36481321)

[268. Missing Number 147](#_Toc36481322)

[269. Alien Dictionary 150](#_Toc36481323)

[270. Closest Binary Search Tree Value 153](#_Toc36481324)

[271. Encode and Decode Strings 155](#_Toc36481325)

[272. Closest Binary Search Tree Value II 157](#_Toc36481326)

[273. Integer to English Words 160](#_Toc36481327)

[274. H-Index(●ˇ∀ˇ●) 162](#_Toc36481328)

[275. H-Index II 164](#_Toc36481329)

[276. Paint Fence 166](#_Toc36481330)

[277. Find the Celebrity 168](#_Toc36481331)

[278. First Bad Version 171](#_Toc36481332)

[279. Perfect Squares(●ˇ∀ˇ●) 173](#_Toc36481333)

[280. Wiggle Sort 174](#_Toc36481334)

[281. Zigzag Iterator 175](#_Toc36481335)

[282. Expression Add Operators 177](#_Toc36481336)

[283. Move Zeroes 179](#_Toc36481337)

[284. Peeking Iterator 180](#_Toc36481338)

[285. Inorder Successor in BST 182](#_Toc36481339)

[286. Walls and Gates 184](#_Toc36481340)

[287. Find the Duplicate Number(●ˇ∀ˇ●) 186](#_Toc36481341)

[288. Unique Word Abbreviation 188](#_Toc36481342)

[289. Game of Life 190](#_Toc36481343)

[290. Word Pattern 192](#_Toc36481344)

[291. Word Pattern II 194](#_Toc36481345)

[292. Nim Game 196](#_Toc36481346)

[293. Flip Game 197](#_Toc36481347)

[294. Flip Game II 199](#_Toc36481348)

[295. Find Median from Data Stream 200](#_Toc36481349)

[296. Best Meeting Point 202](#_Toc36481350)

[297. Serialize and Deserialize Binary Tree(●ˇ∀ˇ●) 204](#_Toc36481351)

[298. Binary Tree Longest Consecutive Sequence 206](#_Toc36481352)

[299. Bulls and Cows 209](#_Toc36481353)

[300. Longest Increasing Subsequence(●ˇ∀ˇ●) 211](#_Toc36481354)

### 201. Bitwise AND of Numbers Range

Medium

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

**Example 1:**

**Input:** [5,7]

**Output:** 4

**Example 2:**

**Input:** [0,1]

**Output:** 0

class Solution **{**

public**:**

int rangeBitwiseAnd**(**int m**,** int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

int rangeBitwiseAnd**(**int m**,** int n**)** **{**

int x **=** 0x40000000**,** res **=** 0**;**

**while** **(**x**)** **{**

**if ((**m **&** x**)** **==** **(**n **&** x**))** res **+=** **(**m **&** x**);**

**else** **break;**

x **>>=** 1**;**

**}**

**return** res**;**

**}**

**};**

### 202. Happy Number

Easy

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

**Example:**

**Input:** 19

**Output:** true

**Explanation:**

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

class Solution **{**

public**:**

bool isHappy**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isHappy**(**int n**)** **{**

unordered\_set**<**int**>** My\_set**;**

**while** **(**1**)** **{**

int t **=** 0**;**

**while** **(**n**)** **{**

t **+=** **(**n**%**10**)\*(**n**%**10**);**

n **/=** 10**;**

**}**

**if** **(**t **==** 1**)** **return** **true;**

**else** **if** **(**My\_set**.**count**(**t**))** **return** **false;**

My\_set**.**insert**(**n **=** t**);**

**}**

**return** **true;**

**}**

**};**

### 203. Remove Linked List Elements

Easy

Remove all elements from a linked list of integers that have value ***val***.

**Example:**

**Input:** 1->2->6->3->4->5->6, ***val*** = 6

**Output:** 1->2->3->4->5

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** removeElements**(**ListNode**\*** head**,** int val**)** **{**

**}**

**};**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** removeElements**(**ListNode**\*** head**,** int val**)** **{**

**if** **(**head **==** **nullptr)** **return** head**;**

**if** **(**head**->**val **==** val**)** **{**

**return** removeElements**(**head**->**next**,** val**);**

**else** **{**

head**->**next **=** removeElements**(**head**->**next**,** val**);**

**return** head**;**

**}**

**}**

**};**

### 204. Count Primes★★

Easy

Count the number of prime numbers less than a non-negative number, ***n***.

**Example:**

**Input:** 10

**Output:** 4

**Explanation:** There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

class Solution **{**

public**:**

int countPrimes**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

int countPrimes**(**int n**)** **{**

**if** **(**n **<=** 2**)** **return** 0**;**

vector**<**bool**>** v**(**n**,** **false);**

int Sqrt **=** sqrt**(**n**)+**0.5**,** cnt **=** n**/**2**;**

**for** **(**int i **=** 3**;** i **<=** Sqrt**;** i **+=** 2**)** **{**

**for** **(**int j **=** i**\***i**;** j **<** n**;** j **+=** 2**\***i**)** **{**

**if** **(!**v**[**j**])** **{**

v**[**j**]** **=** **true;**

**--**cnt**;**

**}**

**}**

**}**

**return** cnt**;**

**}**

**};**

### 205. Isomorphic Strings

Easy

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

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

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

**Example 1:**

**Input:** ***s*** = "egg", ***t =*** "add"

**Output:** true

**Example 2:**

**Input:** ***s*** = "foo", ***t =*** "bar"

**Output:** false

**Example 3:**

**Input:** ***s*** = "paper", ***t =*** "title"

**Output:** true

**Note:**  
You may assume both ***s***and ***t***have the same length.

class Solution **{**

public**:**

bool isIsomorphic**(**string s**,** string t**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isIsomorphic**(**string s**,** string t**)** **{**

**if (**s**.**length**()** **!=** t**.**length**())** **return** **false;**

vector**<**int**>** a**(**256**,-**1**),** b**(**256**,-**1**);**

int n **=** s**.**length**();**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**if (**a**[**s**[**i**]]** **!=** b**[**t**[**i**]])** **return** **false;**

**else if (**a**[**s**[**i**]]** **==** **-**1**)** a**[**s**[**i**]]** **=** b**[**t**[**i**]]** **=** i**;**

**}**

**return** **true;**

**}**

**};**

### 206. Reverse Linked List

Easy

Reverse a singly linked list.

**Example:**

**Input:** 1->2->3->4->5->NULL

**Output:** 5->4->3->2->1->NULL

**Follow up:**

A linked list can be reversed either iteratively or recursively. Could you implement both?

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** reverseList**(**ListNode**\*** head**)** **{**

**}**

**};**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** reverseList**(**ListNode**\*** head**)** **{**

ListNode **\***dummy **=** **new** ListNode**(-**1**),** **\***p**,** **\***t**;**

p **=** head**;**

**while** **(**p**)** **{**

t **=** p**->**next**;**

p**->**next **=** dummy**->**next**;**

dummy**->**next **=** p**;**

p **=** t**;**

**}**

**return** dummy**->**next**;**

**}**

**};**

### 207. Course Schedule

Medium

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

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

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

**Example 1:**

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

**Output:** true

**Explanation:** There are a total of 2 courses to take.

  To take course 1 you should have finished course 0. So it is possible.

**Example 2:**

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

**Output:** false

**Explanation:** There are a total of 2 courses to take.

  To take course 1 you should have finished course 0, and to take course 0 you should

  also have finished course 1. So it is impossible.

**Note:**

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs).
2. You may assume that there are no duplicate edges in the input prerequisites.

class Solution **{**

public**:**

bool canFinish**(**int numCourses**,** vector**<**pair**<**int**,** int**>>&** prerequisites**)** **{**

**}**

**};**

class Solution {

public:

    bool canFinish(int numCourses, vector<vector<int>>& prerequisites){

        vector<vector<int>> v(numCourses);

        vector<int> visited(numCourses, 0);

        for (auto &i : prerequisites) {

            v[i[0]].push\_back(i[1]);

        }

        for (int i = 0; i < numCourses; i++) {

            if (visited[i] == 0) {

               if (!dfs(i, visited, v)) return false;

            }

        }

        return true;

    }

private:

    bool dfs(int cur, vector<int> &visited, vector<vector<int>> &v) {

        visited[cur] = -1;

        for (auto &j : v[cur]) {

            if (visited[j] == -1) return false;

            else if (!visited[j] && !dfs(j, visited, v)) return false;

        }

        visited[cur] = 1;

        return true;

    }

};

### 208. Implement Trie (Prefix Tree) ★★

Medium

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

**Example:**

Trie trie = new Trie();

trie.insert("apple");

trie.search("apple"); // returns true

trie.search("app"); // returns false

trie.startsWith("app"); // returns true

trie.insert("app");

trie.search("app"); // returns true

**Note:**

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

class Trie **{**

public**:**

Trie**()** **{}**

void insert**(**string word**)** **{**

Trie **\***node **=** **this;**

**for** **(**char c **:** word**)** **{**

**if** **(!**node**->**next**.**count**(**c**))** **{**node**->**next**[**c**]** **=** **new** Trie**();}**

node **=** node**->**next**[**c**];**

**}**

node**->**isword **=** **true;**

**}**

bool search**(**string word**)** **{**

Trie **\***node **=** **this;**

**for** **(**char c **:** word**)** **{**

**if** **(!**node**->**next**.**count**(**c**))** **return** **false;**

node **=** node**->**next**[**c**];**

**}**

**return** node**->**isword**;**

**}**

bool startsWith**(**string prefix**)** **{**

Trie **\***node **=** **this;**

**for** **(**auto c **:** prefix**)** **{**

**if** **(!**node**->**next**.**count**(**c**))** **return** **false;**

node **=** node**->**next**[**c**];**

**}**

**return** **true;**

**}**

private**:**

unordered\_map**<**char**,** Trie**\*>** next**;**

bool isword **=** **false;**

**};**

### 209. Minimum Size Subarray Sum

Medium

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

**Example:**

**Input:** s = 7, nums = [2,3,1,2,4,3]

**Output:** 2

**Explanation:** the subarray [4,3] has the minimal length under the problem constraint.

**Follow up:**

If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*).

class Solution **{**

public**:**

int minSubArrayLen**(**int s**,** vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution {

public:

    int minSubArrayLen(int s, vector<int>& nums) {

        int n = nums.size();

        int ans = INT\_MAX;

        int left = 0;

        int sum = 0;

        for (int i = 0; i < n; i++) {

            sum += nums[i];

            while (sum >= s) {

                ans = min(ans, i + 1 - left);

                sum -= nums[left++];

            }

        }

        return (ans != INT\_MAX) ? ans : 0;

    }

};

///////////////O(nlogn)/////////////

class Solution {

public:

    int minSubArrayLen(int s, vector<int>& nums) {

        vector<int> sums{0};

        partial\_sum(nums.begin(), nums.end(), back\_inserter(sums));

        int n = sums.size(), len = INT\_MAX;

        for (int i = n-1; i >= 0 && sums[i] >= s; --i) {

            int j = upper\_bound(sums.begin(),sums.begin()+i,sums[i]-s)

  - sums.begin();

            len = min(len, i-j+1);

        }

        return len == INT\_MAX ? 0 : len;

    }

};

### 210. Course Schedule II

Medium

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

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

Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

**Example 1:**

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

**Output:** [0,1]

**Explanation:** There are a total of 2 courses to take. To take course 1 you should have finished

  course 0. So the correct course order is [0,1] .

**Example 2:**

**Input:** 4, [[1,0],[2,0],[3,1],[3,2]]

**Output:** [0,1,2,3] or [0,2,1,3]

**Explanation:** There are a total of 4 courses to take. To take course 3 you should have finished both

courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0.

  So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3] .

**Note:**

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs).
2. You may assume that there are no duplicate edges in the input prerequisites.

class Solution **{**

public**:**

vector**<**int**>** findOrder**(**int numCourses**,** vector**<**vector**<**int**>>** **&**pres**)** **{**

visited**.**resize**(**numCourses**,** 0**);**

v**.**resize**(**numCourses**);**

**for** **(**auto **&**i **:** pres**)** v**[**i**[**0**]].**push\_back**(**i**[**1**]);**

**for** **(**int i **=** 0**;** i **<** numCourses**;** i**++)** **{**

**if** **(!**visited**[**i**]** **&&** **!**dfs**(**i**))**

**return** vector**<**int**>** **();**

**}**

**return** path**;**

**}**

private**:**

vector**<**int**>** path**,** visited**;**

vector**<**vector**<**int**>>** v**;**

bool dfs**(**int i**)** **{**

visited**[**i**]** **=** **-**1**;**

**for** **(**auto **&**j **:** v**[**i**])** **{**

**if** **(**visited**[**j**]** **==** **-**1**)** **return** **false;**

**else** **if** **(!**visited**[**j**]** **&&** **!**dfs**(**j**))** **return** **false;**

**}**

path**.**push\_back**(**i**);**

visited**[**i**]** **=** 1**;**

**return** **true;**

**}**

**};**

### 211. Add and Search WordData structure design★★ Medium

Design a data structure that supports the following two operations:

void addWord(word)

bool search(word)

search(word) can search a literal word or a regular expression string containing only letters a-z or .. A . means it can represent any one letter.

**Example:**

addWord("bad")

addWord("dad")

addWord("mad")

search("pad") -> false

search("bad") -> true

search(".ad") -> true

search("b..") -> true

**Note:**  
You may assume that all words are consist of lowercase letters a-z.

class WordDictionary {

public:

    WordDictionary() {}

    void addWord(string word) {

        WordDictionary \*node = this;

        for (char c : word) {

            if (!node->next.count(c)) {

                node->next[c] = new WordDictionary();

            }

            node = node->next[c];

        }

        node->isword = true;

    }

    bool search(string word) {

        return search(word, 0, word.length(), this);

    }

    bool search(const string &word,int cur,int n,WordDictionary \*node){

        for (int i = cur; i < n; i++) {

            char c = word[i];

            if (c == '.') {

                for (auto [cc, p] : node->next) {

                    if (search(word, i+1, n, p)) return true;

                }

                return false;

            }

            else if (!node->next.count(c))  return false;

            node = node->next[c];

        }

        return node->isword;

    }

private:

    unordered\_map<char, WordDictionary\*> next;

    bool isword = false;

};

### 212. Word Search II★★

Hard

Given a 2D board and a list of words from the dictionary, find all words in the board.

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

**Example:**

**Input:**

**words** = ["oath","pea","eat","rain"] and **board** =

[

['o','a','a','n'],

['e','t','a','e'],

['i','h','k','r'],

['i','f','l','v']

]

**Output:**["eat","oath"]

**Note:**  
You may assume that all inputs are consist of lowercase letters a-z.

//////////////////////注意本题我用了两个class//////////////////////

class Trie **{**

friend class Solution**;**

public**:**

Trie**()** **{}**

void insert**(**string word**)** **{**

Trie **\***node **=** **this;**

**for** **(**char c **:** word**)** **{**

**if** **(!**node**->**next**.**count**(**c**))** **{**node**->**next**[**c**]** **=** **new** Trie**();}**

node **=** node**->**next**[**c**];**

**}**

node**->**isword **=** **true;**

**}**

Trie**\*** search**(**char c**)** **{**

Trie **\***node **=** **this;**

**if** **(!**node**->**next**.**count**(**c**))** **return** **nullptr;**

**else** **return** node**->**next**[**c**];**

**}**

private**:**

unordered\_map**<**char**,** Trie**\*>** next**;**

bool isword **=** **false;**

**};**

class Solution **{**

public**:**

vector**<**string**>** findWords**(**vector**<**vector**<**char**>>&** board**,**

vector**<**string**>&** words**)** **{**

n **=** board**.**size**(),** m **=** board**[**0**].**size**();**

visited**.**resize**(**n**,** vector**<**bool**>(**m**,** **false));**

Trie **\***node **=** **new** Trie**();**

**for** **(**auto i **:** words**)** node**->**insert**(**i**);**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** j**++)** **{**

string s**;**

dfs**(**i**,** j**,** s**,** node**,** board**);**

**}**

**}**

vector**<**string**>** res**;**

**for** **(**auto **&**i **:** My\_set**)** res**.**push\_back**(**i**);**

**return** res**;**

**}**

private**:**

int n**,** m**;**

unordered\_set**<**string**>** My\_set**;**

vector**<**vector**<**bool**>>** visited**;**

void dfs**(**int i**,**int j**,** string s**,** Trie **\***node**,** vector**<**vector**<**char**>>** **&**board**){**

**if** **(**i **<** 0 **||** i **>=** n **||** j **<** 0 **||** j **>=** m**)** **return;**

**else** **if** **(**visited**[**i**][**j**])** **return;**

visited**[**i**][**j**]** **=** **true;**

char c **=** board**[**i**][**j**];**

s **+=** c**;**

**if** **(!(**node **=** node**->**search**(**c**)))** **{**

visited**[**i**][**j**]** **=** **false;**

**return;**

**}** **else** **if** **(**node**->**isword**)** **{**

My\_set**.**insert**(**s**);**

**}**

dfs**(**i**+**1**,** j**,** s**,** node**,** board**);**

dfs**(**i**-**1**,** j**,** s**,** node**,** board**);**

dfs**(**i**,** j**+**1**,** s**,** node**,** board**);**

dfs**(**i**,** j**-**1**,** s**,** node**,** board**);**

visited**[**i**][**j**]** **=** **false;**

**}**

**};**

### 213. House Robber II★★

Medium

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

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

**Example 1:**

**Input:** [2,3,2]

**Output:** 3

**Explanation:** You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),

  because they are adjacent houses.

**Example 2:**

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

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

  Total amount you can rob = 1 + 3 = 4.

class Solution **{**

public**:**

int rob**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

int rob**(**vector**<**int**>&** nums**)** **{**

int n **=** nums**.**size**();**

**if** **(**n **<** 2**)** **return** n **?** nums**[**0**]** **:** 0**;**

**return** max**(**rob**(**nums**,** 0**,** n **-** 1**),** rob**(**nums**,** 1**,** n**));**

**}**

private**:**

int rob**(**vector**<**int**>&** nums**,** int l**,** int r**)** **{**

int pre **=** 0**,** cur **=** 0**;**

**for** **(**int i **=** l**;** i **<** r**;** i**++)** **{**

int temp **=** max**(**pre **+** nums**[**i**],** cur**);**

pre **=** cur**;**

cur **=** temp**;**

**}**

**return** cur**;**

**}**

**};**

### 214. Shortest Palindrome★★

Hard

Given a string ***s***, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

**Example 1:**

**Input:** "aacecaaa"

**Output:** "aaacecaaa"

**Example 2:**

**Input:** "abcd"

**Output:** "dcbabcd"

///////////////////////////////O(n^2)///////////////////////////////////

class Solution **{**

public**:**

string shortestPalindrome**(**string s**)** **{**

int n **=** s**.**size**(),** i **=** 0**;**

**for** **(**int j **=** n **-** 1**;** j **>=** 0**;** j**--)** **{**

**if** **(**s**[**i**]** **==** s**[**j**])**

i**++;**

**}**

**if** **(**i **==** n**)** **return** s**;**

string remain\_rev **=** s**.**substr**(**i**);**

reverse**(**remain\_rev**.**begin**(),** remain\_rev**.**end**());**

**return** remain\_rev **+** shortestPalindrome**(**s**.**substr**(**0**,** i**))** **+** s**.**substr**(**i**);**

**}**

**};**

//////////////////////O(n^2)/////////////////////////////

class Solution **{**

public**:**

string shortestPalindrome**(**string s**)** **{**

int n **=** s**.**size**();**

string rev**(**s**);**

reverse**(**rev**.**begin**(),** rev**.**end**());**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**if** **(**s**.**substr**(**0**,** n **-** i**)** **==** rev**.**substr**(**i**))**

**return** rev**.**substr**(**0**,** i**)** **+** s**;**

**}**

**return** ""**;**

**}**

**};**

//////////////////////////O（n）///////////////////////

class Solution **{**

public**:**

string shortestPalindrome**(**string s**)** **{**

string rev**(**s**);**

reverse**(**rev**.**begin**(),** rev**.**end**());**

string str **=** s **+** "#" **+** rev**;**

int n **=** str**.**size**();**

vector**<**int**>** f**(**n**,** 0**);**

**for** **(**int i **=** 1**,** len **=** 0**;** i **<** n**;)** **{**

**if** **(**str**[**i**]** **==** str**[**len**])** f**[**i**++]** **=** **++**len**;**

**else** **if** **(**len**)** len **=** f**[**len **-** 1**];**

**else** f**[**i**++]** **=** 0**;**

**}**

**return** rev**.**substr**(**0**,** s**.**length**()-** f**[**n **-** 1**])** **+** s**;**

**}**

**}**

### 215. Kth Largest Element in an Array(●ˇ∀ˇ●)

Medium

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

**Example 1:**

**Input:** [3,2,1,5,6,4] and k = 2

**Output:** 5

**Example 2:**

**Input:** [3,2,3,1,2,4,5,5,6] and k = 4

**Output:** 4

**Note:**   
You may assume k is always valid, 1 ≤ k ≤ array's length.

class Solution **{**

public**:**

int findKthLargest**(**vector**<**int**>&** nums**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

int findKthLargest**(**vector**<**int**>&** nums**,** int k**)** **{**

int left **=** 0**,** right **=** nums**.**size**()** **-** 1**;**

**while** **(**1**)** **{**

int idx **=** partition**(**nums**,** left**,** right**);**

**if** **(**idx **==** k**-**1**)** **break;**

**if** **(**idx **<** k**-**1**)** left **=** idx **+** 1**;**

**else** right **=** idx **-** 1**;**

**}**

**return** nums**[**k**-**1**];**

**}**

private**:**

int partition**(**vector**<**int**>&** nums**,** int left**,** int right**)** **{**

int pivot **=** nums**[**left**],** l **=** left **+** 1**,** r **=** right**;**

**while** **(**l **<=** r**)** **{**

**while** **(**l **<=** r **&&** nums**[**l**]** **>=** pivot**)** l**++;**

**while** **(**l **<=** r **&&** nums**[**r**]** **<=** pivot**)** r**--;**

**if** **(**l **<=** r **&&** nums**[**l**]** **<** pivot **&&** nums**[**r**]** **>** pivot**)** **{**

swap**(**nums**[**l**++],** nums**[**r**--]);**

**}**

**}**

swap**(**nums**[**left**],** nums**[**r**]);**

// swap(nums[left], nums[l];会在nums = [1] 时出错

**return** r**;**

**}**

**};**

### 216. Combination Sum III

Medium

Find all possible combinations of ***k*** numbers that add up to a number ***n***, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

**Note:**

* All numbers will be positive integers.
* The solution set must not contain duplicate combinations.

**Example 1:**

**Input:** ***k*** = 3, ***n*** = 7

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

**Example 2:**

**Input:** ***k*** = 3, ***n*** = 9

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

class Solution **{**

public**:**

vector**<**vector**<**int**>>** combinationSum3**(**int k**,** int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**vector**<**int**>>** combinationSum3**(**int k**,** int n**)** **{**

vector**<**int**>** path**;**

vector**<**vector**<**int**>>** res**;**

dfs**(**1**,** 0**,** k**,** n**,** path**,** res**);**

**return** res**;**

**}**

private**:**

void dfs**(**int i**,** int cur**,** int k**,** int n**,** vector**<**int**>** **&**path**,** vector**<**vector**<**int**>>** **&**res**)** **{**

**if** **(**cur **==** k **&&** n **==** 0**)** **{**

res**.**push\_back**(**path**);**

**return;**

**}**

**for** **(**int j **=** i**;** j **<=** 9 **&&** n**-**j **>=** 0**;** j**++)** **{**

path**.**push\_back**(**j**);**

dfs**(**j**+**1**,** cur**+**1**,** k**,** n**-**j**,** path**,** res**);**

path**.**pop\_back**();**

**}**

**}**

**};**

### 217. Contains Duplicate

Easy

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

**Example 1:**

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

**Output:** true

**Example 2:**

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

**Output:** false

**Example 3:**

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

**Output:** true

class Solution **{**

public**:**

bool containsDuplicate**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool containsDuplicate**(**vector**<**int**>&** nums**)** **{**

sort**(**nums**.**begin**(),** nums**.**end**());**

int n **=** nums**.**size**();**

**for** **(**int i **=** 1**;** i **<** n**;** i**++){**

**if** **(**nums**[**i**-**1**]** **==** nums**[**i**])**

**return** **true;**

**}**

**return** **false;**

**}**

**};**

class Solution **{**

public**:**

bool containsDuplicate**(**vector**<**int**>&** nums**)** **{**

unordered\_set**<**int**>** My\_set**;**

**for** **(**auto **&**i**:** nums**)** **{**

**if** **(**My\_set**.**count**(**i**))** **return** **true;**

**else** My\_set**.**insert**(**i**);**

**}**

**return** **false;**

**}**

**};**

### 218. The Skyline Problem(●ˇ∀ˇ●)

Hard

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).

[](https://leetcode.com/static/images/problemset/skyline1.jpg)[](https://leetcode.com/static/images/problemset/skyline2.jpg)

The geometric information of each building is represented by a triplet of integers [Li, Ri, Hi], where Li and Ri are the x coordinates of the left and right edge of the ith building, respectively, and Hi is its height. It is guaranteed that 0 ≤ Li, Ri ≤ INT\_MAX, 0 < Hi ≤ INT\_MAX, and Ri - Li > 0. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as: [ [2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8] ] .

The output is a list of "**key points**" (red dots in Figure B) in the format of [ [x1,y1], [x2, y2], [x3, y3], ... ] that uniquely defines a skyline. **A key point is the left endpoint of a horizontal line segment**. Note that the last key point, where the rightmost building ends, is merely used to mark the termination of the skyline, and always has zero height. Also, the ground in between any two adjacent buildings should be considered part of the skyline contour.

For instance, the skyline in Figure B should be represented as:[ [2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0] ].

**Notes:**

* The number of buildings in any input list is guaranteed to be in the range [0, 10000].
* The input list is already sorted in ascending order by the left x position Li.
* The output list must be sorted by the x position.
* There must be no consecutive horizontal lines of equal height in the output skyline. For instance, [...[2 3], [4 5], [7 5], [11 5], [12 7]...] is not acceptable; the three lines of height 5 should be merged into one in the final output as such: [...[2 3], [4 5], [12 7], ...]

class Solution **{**

public**:**

vector**<**vector**<**int**>>** getSkyline**(**vector**<**vector**<**int**>>&** buildings**)** **{**

vector**<**pair**<**int**,** int**>>** edges**;**

**for(**const auto **&**v **:** buildings**)** **{**

edges**.**push\_back**(**make\_pair**(**v**[**0**],** **-**v**[**2**]));**

edges**.**push\_back**(**make\_pair**(**v**[**1**],** v**[**2**]));**

**}**

sort**(**edges**.**begin**(),** edges**.**end**());**

vector**<**vector**<**int**>>** res**;**

multiset**<**int**>** My\_set**{**0**};**

int pre\_top **=** 0**;**

**for (**const auto **&**e **:** edges**){**

**if** **(**e**.**second **<** 0**)** My\_set**.**insert**(-**e**.**second**);**

**else** My\_set**.**erase**(**My\_set**.**find**(**e**.**second**));**

int cur\_top **=** **\*(**My\_set**.**rbegin**());**

**if** **(**cur\_top **!=** pre\_top**){**

res**.**push\_back**({**e**.**first**,** pre\_top **=** cur\_top**});**

**}**

**}**

**return** res**;**

**}**

**};**

### 219. Contains Duplicate II

Easy

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

**Example 1:**

**Input:** nums = [1,2,3,1], k = 3

**Output:** true

**Example 2:**

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

**Output:** true

**Example 3:**

**Input:** nums = [1,2,3,1,2,3], k = 2

**Output:** false

class Solution **{**

public**:**

bool containsNearbyDuplicate**(**vector**<**int**>&** nums**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool containsNearbyDuplicate**(**vector**<**int**>&** nums**,** int k**)** **{**

unordered\_set**<**int**>** My\_set**;**

**for** **(**int i **=** 0**;** i **<** nums**.**size**();** i**++){**

**if** **(**My\_set**.**count**(**nums**[**i**]))** **return** **true;**

My\_set**.**insert**(**nums**[**i**]);**

**if (**i **>=** k**)** My\_set**.**erase**(**nums**[**i**-**k**]);**

**}**

**return** **false;**

**}**

**};**

### 220. Contains Duplicate III(●ˇ∀ˇ●)

Medium

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

**Example 1:**

**Input:** nums = [1,2,3,1], k = 3, t = 0

**Output:** true

**Example 2:**

**Input:** nums = [1,0,1,1], k = 1, t = 2

**Output:** true

**Example 3:**

**Input:** nums = [1,5,9,1,5,9], k = 2, t = 3

**Output:** false

class Solution **{**

public**:**

bool containsNearbyAlmostDuplicate**(**vector**<**int**>&** nums**,** int k**,** int t**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool containsNearbyAlmostDuplicate**(**vector**<**int**>&** nums**,** int k**,** int t**)** **{**

multiset**<**long long**>** My\_set**;**

**for** **(**int i **=** 0**;** i **<** nums**.**size**();** i**++){**

auto pos **=** My\_set**.**lower\_bound**((**long long**)(**nums**[**i**])-**t**);**

**if (**pos **!=** My\_set**.**end**()** **&&** **\***pos **- (**long long**)(**nums**[**i**])** **<=** t**) {**

**return** **true;**

**}**

My\_set**.**insert**(**nums**[**i**]);**

**if** **(**i **>=** k**)** My\_set**.**erase**(**nums**[**i**-**k**]);**

**}**

**return** **false;**

**}**

**};**

### 221. Maximal Square(●ˇ∀ˇ●)

Medium

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

**Example:**

**Input:**

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

**Output:** 4

class Solution **{**

public**:**

int maximalSquare**(**vector**<**vector**<**char**>>&** matrix**)** **{**

**}**

**};**

class Solution **{**

public**:**

int maximalSquare**(**vector**<**vector**<**char**>>&** matrix**)** **{**

**if** **(**matrix**.**empty**())** **return** 0**;**

int n **=** matrix**.**size**(),** m **=** matrix**[**0**].**size**(),** res **=** 0**;**

vector**<**vector**<**int**>>** dp**(**2**,** vector**<**int**>(**m**,** 0**));**

int k **=** 0**;**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** **++**j**)** **{**

**if** **(**i **==** 0 **&&** matrix**[**i**][**j**]** **==** '1'**)** dp**[**k**][**j**]** **=** 1**;**

**else** **if** **(**matrix**[**i**][**j**]** **==** '0'**)** dp**[**k**][**j**]** **=** 0**;**

**else** **if** **(**j **==** 0**)** dp**[**k**][**j**]** **=** 1**;**

**else** dp**[**k**][**j**]** **=** 1 + min**(**min**(**dp**[**k**][**j**-**1**],** dp**[**k**^**1**][**j**]),**

dp**[**k**^**1**][**j**-**1**]);**

res **=** max**(**res**,** dp**[**k**][**j**]);**

**}**

k **^=** 1**;**

**}**

**return** res**\***res**;**

**}**

**};**

### 222. Count Complete Tree Nodes(●ˇ∀ˇ●)

Medium

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

**Note:**

**Definition of a complete binary tree from** [**Wikipedia**](http://en.wikipedia.org/wiki/Binary_tree#Types_of_binary_trees)**:**  
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

**Example:**

**Input:**

1

/ \

2 3

/ \ /

4 5 6

**Output:** 6

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution **{**

public**:**

int countNodes**(**TreeNode**\*** root**)** **{**

**}**

**};**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution **{**

public**:**

int countNodes**(**TreeNode**\*** root**)** **{**

**if** **(!**root**)** **return** 0**;**

int hl **=** 0**,** hr **=** 0**;**

TreeNode **\***l **=** root**,** **\***r **=** root**;**

**while(**l**)** **{**hl**++;** l **=** l**->**left**;}**

**while(**r**)** **{**hr**++;** r **=** r**->**right**;}**

**if (**hl **==** hr**)** **return** **(**1 **<<** hl**)** **-** 1**;**

**return** 1 **+** countNodes**(**root**->**left**)** **+** countNodes**(**root**->**right**);**

**}**

**};**

### 223. Rectangle Area

Medium

Find the total area covered by two **rectilinear** rectangles in a **2D** plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



**Example:**

**Input:** A = -3, B = 0, C = 3, D = 4, E = 0, F = -1, G = 9, H = 2

**Output:** 45

**Note:**

Assume that the total area is never beyond the maximum possible value of **int**.

class Solution **{**

public**:**

int computeArea**(**int A**,** int B**,** int C**,** int D**,** int E**,** int F**,** int G**,** int H**)** **{**

**}**

**};**

class Solution **{**

public**:**

int computeArea**(**int A**,** int B**,** int C**,** int D**,** int E**,** int F**,** int G**,** int H**)** **{**

int sum **=** **(**C **-** A**)** **\*** **(**D **-** B**)** **+** **(**H **-** F**)** **\*** **(**G **-** E**);**

**if** **(**E **>=** C **||** F **>=** D **||** B **>=** H **||** A **>=** G**)** **return** sum**;**

**return** sum **-** **((**min**(**G**,** C**)** **-** max**(**A**,** E**))** **\*** **(**min**(**D**,** H**)** **-** max**(**B**,** F**)));**

**}**

**};**

### 224. Basic Calculator

Hard

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, **non-negative** integers and empty spaces .

**Example 1:**

**Input:** "1 + 1"

**Output:** 2

**Example 2:**

**Input:** " 2-1 + 2 "

**Output:** 3

**Example 3:**

**Input:** "(1+(4+5+2)-3)+(6+8)"

**Output:** 23

**Note:**

* You may assume that the given expression is always valid.
* **Do not** use the eval built-in library function.

class Solution **{**

public**:**

int calculate**(**string s**)** **{** //加强版考虑 + - \* /

s **=** '(' **+** s **+** ')'**;**

stack**<**long**>** digit**;**

stack**<**char**>** punct**;**

int i **=** 0**,** n **=** s**.**length**();**

**while** **(**i **<** n**)** **{**

**if** **(**isspace**(**s**[**i**]))** i**++;**

**else** **if** **(**isdigit**(**s**[**i**]))** **{**

long temp **=** 0**;**

**while** **(**isdigit**(**s**[**i**]))** **{**

temp **=** temp**\***10 **+** s**[**i**++]-**'0'**;**

**}**

digit**.**push**(**temp**);**

**}else** **if** **(**s**[**i**]** **==** '('**)** **{**

punct**.**push**(**s**[**i**++]);**

**}else** **if** **(**s**[**i**]** **==** ')'**)** **{**

char c**;**

**while** **((**c **=** punct**.**top**())** **!=** '('**)** **{**

punct**.**pop**();**

digit**.**push**(**fun**(**digit**,** c**));**

**}**

punct**.**pop**();**

i**++;**

**}else** **if** **(**s**[**i**]** **==** '\*' **||** s**[**i**]** **==** '/'**)** **{**

char c**;**

**while** **(!**punct**.**empty**()** **&&** **(**c **=** punct**.**top**())** **!=** '('

**&&** c **!=** '+' **&&** c **!=** '-'**)** **{**

punct**.**pop**();**

digit**.**push**(**fun**(**digit**,** c**));**

**}**

punct**.**push**(**s**[**i**++]);**

**}else** **{**

char c**;**

**while** **(!**punct**.**empty**()** **&&** **(**c **=** punct**.**top**())** **!=** '('**)** **{**

punct**.**pop**();**

digit**.**push**(**fun**(**digit**,** c**));**

**}**

punct**.**push**(**s**[**i**++]);**

**}**

**}**

**return** digit**.**top**();**

**}**

private**:**

long fun**(**stack**<**long**>** **&**digit**,** char c**)** **{**

long b **=** digit**.**top**();** digit**.**pop**();**

long a **=** digit**.**top**();** digit**.**pop**();**

**switch** **(**c**)** **{**

**case** '+' **:** **return** a**+**b**;**

**case** '-' **:** **return** a**-**b**;**

**case** '\*' **:** **return** a**\***b**;**

**case** '/' **:** **return** a**/**b**;**

**}**

**return** **-**1**;**

**}**

**};**

### 225. Implement Stack using Queues

Easy

Implement the following operations of a stack using queues.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* empty() -- Return whether the stack is empty.

**Example:**

MyStack stack = new MyStack();

stack.push(1);

stack.push(2);

stack.top(); // returns 2

stack.pop(); // returns 2

stack.empty(); // returns false

**Notes:**

* You must use *only* standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
* Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
* You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

class MyStack **{**

public**:**

queue**<**int**>** q**;**

MyStack**()** **{}**

void push**(**int x**)** **{**

int sz **=** q**.**size**();**

q**.**push**(**x**);**

**while** **(**sz**--)** **{**

q**.**push**(**q**.**front**());**

q**.**pop**();**

**}**

**}**

int pop**()** **{**

int t **=** q**.**front**();**

q**.**pop**();**

**return** t**;**

**}**

int top**()** **{**

**return** q**.**front**();**

**}**

bool empty**()** **{**

**return** q**.**empty**();**

**}**

**};**

### 226. Invert Binary Tree

Easy

Invert a binary tree.

**Example:**

Input:

4

/ \

2 7

/ \ / \

1 3 6 9

Output:

4

/ \

7 2

/ \ / \

9 6 3 1

**Trivia:**  
This problem was inspired by [this original tweet](https://twitter.com/mxcl/status/608682016205344768) by [Max Howell](https://twitter.com/mxcl):

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

class Solution **{**

public**:**

TreeNode**\*** invertTree**(**TreeNode**\*** root**)** **{**

**}**

**};**

class Solution **{**

public**:**

TreeNode**\*** invertTree**(**TreeNode**\*** root**)** **{**

**if** **(**root **==** **nullptr)** **return** **nullptr;**

swap**(**root**->**left**,** root**->**right**);**

invertTree**(**root**->**left**);**

invertTree**(**root**->**right**);**

**return** root**;**

**}**

**};**

### 227. Basic Calculator II(●ˇ∀ˇ●)

Medium

Implement a basic calculator to evaluate a simple expression string.

The expression string contains only **non-negative** integers, +, -, \*, / operators and empty spaces . The integer division should truncate toward zero.

**Example 1:**

**Input:** "3+2\*2"

**Output:** 7

**Example 2:**

**Input:** " 3/2 "

**Output:** 1

**Example 3:**

**Input:** " 3+5 / 2 "

**Output:** 5

**Note:**

* You may assume that the given expression is always valid.
* **Do not** use the eval built-in library function.

class Solution **{**

public**:**

int calculate**(**string s**)** **{**

**}**

**};**

class Solution **{**

public**:**

int calculate**(**string s**)** **{**

stringstream ss**(**"+" **+** s**);**

char op**;**

int n**,** last**,** ans **=** 0**;**

**while** **(**ss **>>** op **>>** n**)** **{**

**if** **(**op **==** '+' **||** op **==** '-'**)** **{**

n **=** op **==** '+' **?** n **:** **-**n**;**

ans **+=** n**;**

**}** **else** **{**

n **=** op **==** '\*' **?** last **\*** n **:** last **/** n**;**

ans **=** ans **-** last **+** n**;**

// simulate a stack by recovering last values

**}**

last **=** n**;**

**}**

**return** ans**;**

**}**

**};**

### 228. Summary Ranges

Medium

Given a sorted integer array without duplicates, return the summary of its ranges.

**Example 1:**

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

**Output:** ["0->2","4->5","7"]

**Explanation:** 0,1,2 form a continuous range; 4,5 form a continuous range.

**Example 2:**

**Input:** [0,2,3,4,6,8,9]

**Output:** ["0","2->4","6","8->9"]

**Explanation:** 2,3,4 form a continuous range; 8,9 form a continuous range.

class Solution **{**

public**:**

vector**<**string**>** summaryRanges**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**string**>** summaryRanges**(**vector**<**int**>&** nums**)** **{**

vector**<**string**>** res**;**

**if** **(**nums**.**empty**())** **return** res**;**

nums**.**push\_back**(**nums**.**back**());**

long low**,** st**,** i **=** 0**;**

string s**;**

**while** **(**i **<** nums**.**size**())** **{**

**if** **(**s**.**empty**())** **{**

s **=** to\_string**(**st **=** low **=** nums**[**i**++]);**

**}**

**else** **if** **(**nums**[**i**]** **!=** **++**low**)** **{**

**if** **(**low**-**1 **!=** st**)** res**.**push\_back**(**s **+** "->" **+** to\_string**(**low**-**1**));**

**else** res**.**push\_back**(**s**);**

s**.**clear**();**

**}**

**else** i**++;**

**}**

**return** res**;**

**}**

**};**

### 229. Majority Element II

Medium

Given an integer array of size *n*, find all elements that appear more than ⌊ n/3 ⌋ times.

**Note:** The algorithm should run in linear time and in O(1) space.

**Example 1:**

**Input:** [3,2,3]

**Output:** [3]

**Example 2:**

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

**Output:** [1,2]

class Solution **{**

public**:**

vector**<**int**>** majorityElement**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** majorityElement**(**vector**<**int**>&** nums**)** **{**

vector**<**int**>** res**;**

int n **=** 0**,** m **=** 0**,** cnt\_n **=** 0**,** cnt\_m **=** 0**;**

**for** **(**auto **&**i **:** nums**)** **{**

**if** **(**i **==** n**)** cnt\_n**++;**

**else** **if** **(**i **==** m**)** cnt\_m**++;**

**else** **if (!**cnt\_n**)** **{**n **=** i**;** cnt\_n **=** 1**;}**

**else** **if (!**cnt\_m**)** **{**m **=** i**;** cnt\_m **=** 1**;}**

**else** **{**cnt\_n**--;** cnt\_m**--;}**

**}**

cnt\_n **=** cnt\_m **=** 0**;**

**for** **(**auto **&**i **:** nums**){**

**if (**i **==** n**)** cnt\_n**++;**

**else** **if (**i **==** m**)** cnt\_m**++;**

**}**

**if (**cnt\_n **>** floor**(**nums**.**size**()/**3**))** res**.**push\_back**(**n**);**

**if (**cnt\_m **>** floor**(**nums**.**size**()/**3**))** res**.**push\_back**(**m**);**

**return** res**;**

**}**

**};**

### 230. Kth Smallest Element in a BST

Medium

Given a binary search tree, write a function kthSmallest to find the **k**th smallest element in it.

**Note:**   
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

**Example 1:**

**Input:** root = [3,1,4,null,2], k = 1

3

/ \

1 4

\

  2

**Output:** 1

**Example 2:**

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

5

/ \

3 6

/ \

2 4

/

1

**Output:** 3

**Follow up:**  
What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

class Solution **{**

public**:**

int kthSmallest**(**TreeNode**\*** root**,** int k**)** **{**

stack**<**TreeNode**\*>** stk**;**

TreeNode **\***p **=** root**;**

**while** **(**p **||** **!**stk**.**empty**())** **{**

**while(**p**)** **{**

stk**.**push**(**p**);**

p **=** p**->**left**;**

**}**

p **=** stk**.**top**();**

stk**.**pop**();**

**if(--**k **==** 0**)** **return** p**->**val**;**

p **=** p**->**right**;**

**}**

**return** **-**1**;**

**}**

**};**

### 231. Power of Two

Easy

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

**Example 1:**

**Input:** 1

**Output:** true

**Explanation:** 20 = 1

**Example 2:**

**Input:** 16

**Output:** true

**Explanation:** 24 = 16

**Example 3:**

**Input:** 218

**Output:** false

class Solution **{**

public**:**

bool isPowerOfTwo**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isPowerOfTwo**(**int n**)** **{**

int cnt **=** 0**;**

**while** **(**n**)** **{**

**if** **(**n **&** 0x00000001**)** **++**cnt**;**

**if** **(**cnt **>=** 2**)** **return** **false;**

n **>>=** 1**;**

**}**

**return** cnt **==** 1**;**

**}**

**};**

class Solution **{**

public**:**

bool isPowerOfTwo**(**int n**)** **{**

**if** **(**n **<=** 0**)** **return** **false;**

**else** **return** **(**n **&** **(**n**-**1**))** **==** 0**;**

**}**

**};**

### 232. Implement Queue using Stacks

Easy

Implement the following operations of a queue using stacks.

* push(x) -- Push element x to the back of queue.
* pop() -- Removes the element from in front of queue.
* peek() -- Get the front element.
* empty() -- Return whether the queue is empty.

**Example:**

MyQueue queue = new MyQueue();

queue.push(1);

queue.push(2);

queue.peek(); // returns 1

queue.pop(); // returns 1

queue.empty(); // returns false

**Notes:**

* You must use *only* standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.
* Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
* You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

class MyQueue **{**

public**:**

/\*\* Initialize your data structure here. \*/

stack**<**int**>** s1**,** s2**;**

MyQueue**()** **{}**

/\*\* Push element x to the back of queue. \*/

void push**(**int x**)** **{**

s2**.**push**(**x**);**

**}**

/\*Removes the element from in front of queue and returns that element. \*/

int pop**()** **{**

**if(**s1**.**empty**())** **{**

**while(!**s2**.**empty**()){**

s1**.**push**(**s2**.**top**());**

s2**.**pop**();**

**}**

**}**

int t **=** s1**.**top**();**

s1**.**pop**();**

**return** t**;**

**}**

/\*\* Get the front element. \*/

int peek**()** **{**

**if(**s1**.**empty**())** **{**

**while(!**s2**.**empty**()){**

s1**.**push**(**s2**.**top**());**

s2**.**pop**();**

**}**

**}**

**return** s1**.**top**();**

**}**

/\*\* Returns whether the queue is empty. \*/

bool empty**()** **{**

**return** s1**.**empty**()** **&&** s2**.**empty**();**

**}**

**};**

### 233. Number of Digit One

Hard

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

**Example:**

**Input:** 13

**Output:** 6

**Explanation:** Digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

class Solution **{**

public**:**

int countDigitOne**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

int countDigitOne**(**int n**)** **{**

int res **=** 0**;**

**if** **(**n **<=** 0**)** **return** res**;**

long int high **=** n**,** low **=** 0**,** fact **=** 1**,** cur**;**

**while** **(**high**)** **{**

cur **=** high**%**10**;**

high **/=** 10**;**

res **+=** **(**high**+** **(**cur **>** 1 **?** 1**:** 0**))\***fact**+(**cur **==** 1 **?** low**+**1 **:** 0**);**

low **+=** fact**\***cur**;**

fact **\*=** 10**;**

**}**

**return** res**;**

**}**

**};**

### 234. Palindrome Linked List

Easy

Given a singly linked list, determine if it is a palindrome.

**Example 1:**

**Input:** 1->2

**Output:** false

**Example 2:**

**Input:** 1->2->2->1

**Output:** true

**Follow up:**  
Could you do it in O(n) time and O(1) space?

class Solution **{**

public**:**

bool isPalindrome**(**ListNode**\*** head**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isPalindrome**(**ListNode**\*** head**)** **{**

**if (**head **==** **nullptr** **||** head**->**next **==** **nullptr)** **return** **true;**

ListNode **\***fast **=** head**,** **\***slow **=** head**;**

**while** **(**fast **&&** fast**->**next **&&** fast**->**next**->**next**)** **{**

fast **=** fast**->**next**->**next**;**

slow **=** slow**->**next**;**

**}**

ListNode **\***p **=** slow**->**next**,** **\***q**;**

slow**->**next **=** **nullptr;**

**while** **(**p**)** **{**

q **=** p**->**next**;**

p**->**next **=** slow**->**next**;**

slow**->**next **=** p**;**

p **=** q**;**

**}**

p **=** head**;** q **=** slow**->**next**;**

**while** **(**p **&&** q**)** **{**

**if** **(**p**->**val **!=** q**->**val**)** **return** **false;**

p **=** p**->**next**;**

q **=** q**->**next**;**

**}**

**return** **true;**

**}**

**};**

### 235. Lowest Common Ancestor of a Binary Search Tree

Easy

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

Given binary search tree:  root = [6,2,8,0,4,7,9,null,null,3,5]



**Example 1:**

**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

**Output:** 6

**Explanation:** The LCA of nodes 2 and 8 is 6.

**Example 2:**

**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

**Output:** 2

**Explanation:** The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

**Note:**

* All of the nodes' values will be unique.
* p and q are different and both values will exist in the BST.

class Solution **{**

public**:**

TreeNode**\*** lowestCommonAncestor**(**TreeNode**\*** root**,** TreeNode**\*** p**,** TreeNode**\*** q**){**

**if** **(**root**->**val **>** p**->**val **&&** root**->**val **>** q**->**val**)**

**return** lowestCommonAncestor**(**root**->**left**,** p**,** q**);**

**else** **if** **(**root**->**val **<** p**->**val **&&** root**->**val **<** q**->**val**)**

**return** lowestCommonAncestor**(**root**->**right**,** p**,** q**);**

**else** **return** root**;**

**}**

**};**

### 236. Lowest Common Ancestor of a Binary Tree(●ˇ∀ˇ●)

Medium

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

Given the following binary tree:  root = [3,5,1,6,2,0,8,null,null,7,4]



**Example 1:**

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

**Output:** 3

**Explanation:** The LCA of nodes 5 and 1 is 3.

**Example 2:**

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

**Output:** 5

**Explanation:** The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

**Note:**

* All of the nodes' values will be unique.
* p and q are different and both values will exist in the binary tree.

/\*\*

\* 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**\*** lowestCommonAncestor**(**TreeNode**\*** root**,** TreeNode**\*** p**,** TreeNode**\*** q**){**

TreeNode **\***cur **=** root**,** **\***last **=** **nullptr;**

vector**<**TreeNode**\*>** pathp**,** pathq**,** temp**;**

**while** **(**pathp**.**empty**()** **||** pathq**.**empty**())** **{**

**while** **(**cur**)** **{**

temp**.**push\_back**(**cur**);**

**if** **(**cur **==** p**)** pathp **=** temp**;** // check and set path for p

**if** **(**cur **==** q**)** pathq **=** temp**;** // check and set path for q

cur **=** cur**->**left**;**

**}**

**if** **(**temp**.**back**()->**right **&&** temp**.**back**()->**right **!=** last**)**

cur **=** temp**.**back**()->**right**;**

**else** **{**

last **=** temp**.**back**();**

temp**.**pop\_back**();**

**}**

**}**

// compare paths and get lowest common ancestor

int n **=** min**(**pathp**.**size**(),** pathq**.**size**());**

**for** **(**int i **=** 1**;** i **<** n**;** i**++)** **{**

**if** **(**pathp**[**i**]** **!=** pathq**[**i**])** **return** pathp**[**i**-**1**];**

**}**

**return** pathp**[**n**-**1**];**

**}**

**};**

class Solution **{**

public**:**

TreeNode**\*** lowestCommonAncestor**(**TreeNode**\*** root**,** TreeNode**\*** p**,** TreeNode**\*** q**){**

**if** **(!**root **||** root **==** p **||** root **==** q**)** **return** root**;**

TreeNode **\***left **=** lowestCommonAncestor**(**root**->**left**,** p**,** q**);**

TreeNode **\***right **=** lowestCommonAncestor**(**root**->**right**,** p**,** q**);**

**return** **!**left **?** right **:** **!**right **?** left **:** root**;**

**}**

**};**

### 237. Delete Node in a Linked List

Easy

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

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



**Example 1:**

**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

**Example 2:**

**Input:** head = [4,5,1,9], node = 1

**Output:** [4,5,9]

**Explanation:** You are given the third node with value 1, the linked list should become 4 -> 5 -> 9 after calling your function.

**Note:**

* The linked list will have at least two elements.
* All of the nodes' values will be unique.
* The given node will not be the tail and it will always be a valid node of the linked list.
* Do not return anything from your function.

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

void deleteNode**(**ListNode**\*** node**)** **{**

**\***node **=** **\*(**node**->**next**);**

**}**

**};**

### 238. Product of Array Except Self

Medium

Given an array nums of *n* integers where *n* > 1,  return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

**Example:**

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

**Output:** [24,12,8,6]

**Note:** Please solve it **without division** and in O(*n*).

**Follow up:**  
Could you solve it with constant space complexity? (The output array **does not** count as extra space for the purpose of space complexity analysis.)

class Solution **{**

public**:**

vector**<**int**>** productExceptSelf**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** productExceptSelf**(**vector**<**int**>&** nums**)** **{**

int n **=** nums**.**size**();**

vector**<**int**>** res**(**n**);**

res**[**0**]** **=** 1**;**

**for** **(**int i **=** 1**;** i **<** n**;** i**++)** res**[**i**]** **=** nums**[**i**-**1**]\***res**[**i**-**1**];**

int a **=** nums**[**n**-**1**];**

**for** **(**int i **=** n**-**2**;** i **>=** 0**;** i**--)** **{**

res**[**i**]** **\*=** a**;**

a **\*=** nums**[**i**];**

**}**

**return** res**;**

**}**

**};**

### 239. Sliding Window Maximum(●ˇ∀ˇ●)

Hard

Given an array *nums*, there is a sliding window of size *k* which is moving from the very left of the array to the very right. You can only see the *k* numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.

**Example:**

**Input:** *nums* = [1,3,-1,-3,5,3,6,7], and *k* = 3

**Output:** [3,3,5,5,6,7]

**Explanation:**

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 **3**

1 [3 -1 -3] 5 3 6 7 **3**

1 3 [-1 -3 5] 3 6 7  **5**

1 3 -1 [-3 5 3] 6 7 **5**

1 3 -1 -3 [5 3 6] 7 **6**

1 3 -1 -3 5 [3 6 7] **7**

**Note:**   
You may assume *k* is always valid, 1 ≤ k ≤ input array's size for non-empty array.

**Follow up:**  
Could you solve it in linear time?

class Solution **{**

public**:**

vector**<**int**>** maxSlidingWindow**(**vector**<**int**>&** nums**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** maxSlidingWindow**(**vector**<**int**>&** nums**,** int k**)** **{**

int n **=** nums**.**size**(),** cnt **=** 0**;**

vector**<**int**>** res**;**

deque**<**int**>** deq**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**while (!**deq**.**empty**()** **&&** nums**[**deq**.**back**()]** **<=** nums**[**i**])** **{**

deq**.**pop\_back**();**

**}**

deq**.**push\_back**(**i**);**

**if (**i**-**deq**.**front**()** **>=** k**)** deq**.**pop\_front**();**

**if (**i **>=** k**-**1**)** res**.**push\_back**(**nums**[**deq**.**front**()]);**

**}**

**return** res**;**

**}**

**};**

### 240. Search a 2D Matrix II(●ˇ∀ˇ●)

Medium

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

* Integers in each row are sorted in ascending from left to right.
* Integers in each column are sorted in ascending from top to bottom.

**Example:**

Consider the following matrix:

[

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

Given target = 5, return true.

Given target = 20, return false.

class Solution **{**

public**:**

bool searchMatrix**(**vector**<**vector**<**int**>>&** matrix**,** int target**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool searchMatrix**(**vector**<**vector**<**int**>>&** matrix**,** int target**)** **{**

**if** **(**matrix**.**empty**())** **return** **false;**

int n **=** matrix**.**size**(),** m **=** matrix**[**0**].**size**();**

int i **=** 0**,** j **=** m**-**1**;**

**while** **(**i **<** n **&&** j **>=** 0**)** **{**

**if** **(**matrix**[**i**][**j**]** **==** target**)** **return** **true;**

**else** **if** **(**matrix**[**i**][**j**]** **<** target**)** i**++;**

**else** j**--;**

**}**

**return** **false;**

**}**

**};**

### 241. Different Ways to Add Parentheses(●ˇ∀ˇ●)

Medium

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are +, - and \*.

**Example 1:**

**Input:** "2-1-1"

**Output:** [0, 2]

**Explanation:**

((2-1)-1) = 0

(2-(1-1)) = 2

**Example 2:**

**Input:** "2\*3-4\*5"

**Output:** [-34, -14, -10, -10, 10]

**Explanation:**

(2\*(3-(4\*5))) = -34

((2\*3)-(4\*5)) = -14

((2\*(3-4))\*5) = -10

(2\*((3-4)\*5)) = -10

(((2\*3)-4)\*5) = 10

class Solution **{**

public**:**

vector**<**int**>** diffWaysToCompute**(**string input**)** **{**

vector**<**int**>** output**;**

**for** **(**int i **=** 0**;** i **<** input**.**size**();** i**++)** **{**

char c **=** input**[**i**];**

**if** **(**ispunct**(**c**))** **{**

**for** **(**int a **:** diffWaysToCompute**(**input**.**substr**(**0**,** i**)))** **{**

**for** **(**int b **:** diffWaysToCompute**(**input**.**substr**(**i**+**1**)))** **{**

output**.**push\_back**(**c **==** '+' **?** a**+**b

**:** c **==** '-' **?** a**-**b **:** a**\***b**);**

**}**

**}**

**}**

**}**

**return** output**.**size**()** **?** output **:** vector**<**int**>** **{**stoi**(**input**)};**

**}**

**};**

////////////////////DP////////////////////////////////////

class Solution **{**

public**:**

vector**<**int**>** diffWaysToCompute**(**string input**)** **{**

**return** computeWithDP**(**input**);**

**}**

private**:**

unordered\_map**<**string**,** vector**<**int**>>** dp**;**

vector**<**int**>** computeWithDP**(**string input**)** **{**

vector**<**int**>** result**;**

int sz **=** input**.**size**();**

**for** **(**int i **=** 0**;** i **<** sz**;** i**++)** **{**

char c **=** input**[**i**];**

**if** **(**ispunct**(**c**))** **{**

string substr **=** input**.**substr**(**0**,** i**);**

auto res1 **=** dp**.**find**(**substr**)** **!=** dp**.**end**()**

**?** dp**[**substr**]:** computeWithDP**(**substr**);**

substr **=** input**.**substr**(**i**+**1**);**

auto res2 **=** dp**.**find**(**substr**)** **!=** dp**.**end**()**

**?** dp**[**substr**]:** computeWithDP**(**substr**);**

**for** **(**auto n1 **:** res1**)** **{**

**for** **(**auto n2 **:** res2**)** **{**

result**.**push\_back**(**c **==** '+' **?** n1**+**n2

**:** c **==** '-' **?** n1**-**n2 **:** n1**\***n2**);**

**}**

**}**

**}**

**}**

**if** **(**result**.**empty**())** result**.**push\_back**(**atoi**(**input**.**c\_str**()));**

**return** dp**[**input**]** **=** result**;**

**}**

**};**

### 242. Valid Anagram

Easy

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

**Example 1:**

**Input:** *s* = "anagram", *t* = "nagaram"

**Output:** true

**Example 2:**

**Input:** *s* = "rat", *t* = "car"

**Output:** false

**Note:**  
You may assume the string contains only lowercase alphabets.

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

class Solution **{**

public**:**

bool isAnagram**(**string s**,** string t**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isAnagram**(**string s**,** string t**)** **{**

**if** **(**s**.**length**()** **!=** t**.**length**())** **return** **false;**

int a**[**26**]** **=** **{**0**};**

**for (**auto **&**i **:** s**)** a**[**i**-**'a'**]++;**

**for (**auto **&**i **:** t**){**

**if(--**a**[**i**-**'a'**]** **<** 0**)**

**return** **false;**

**}**

**return** **true;**

**}**

**};**

### 243.Shortest Word Distance

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

**Example:**  
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

**Input:** word1 = “coding”, word2 = “practice”

**Output:** 3

**Input:** word1 = "makes", word2 = "coding"

**Output:** 1

**Note:**  
You may assume that word1 **does not equal to** word2, and word1 and word2 are both in the list.

class Solution **{**

public**:**

int shortestDistance**(**vector**<**string**>&** words**,** string word1**,** string word2**)** **{**

int p1 **=** **-**1**,** p2 **=** **-**1**;**

int n **=** words**.**size**(),** res **=** INT\_MAX**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**if** **(**words**[**i**]** **==** word1**)** p1 **=** i**;**

**else** **if** **(**words**[**i**]** **==** word2**)** p2 **=** i**;**

**if** **(**p1 **!=** **-**1 **&&** p2 **!=** **-**1**)** res **=** min**(**res**,** abs**(**p1 **-** p2**));**

**}**

**return** res**;**

**}**

**};**

### 244.Shortest Word Distance II

Design a class which receives a list of words in the constructor, and implements a method that takes two words word1 and word2 and return the shortest distance between these two words in the list. Your method will be called repeatedly many times with different parameters.

**Example:**  
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

**Input:** word1 = “coding”, word2 = “practice”

**Output:** 3

**Input:** word1 = "makes", word2 = "coding"

**Output:** 1

**Note:**  
You may assume that word1 **does not equal to** word2, and word1 and word2 are both in the list.

class WordDistance **{**

public**:**

WordDistance**(**vector**<**string**>&** words**)** **{**

**}**

int shortest**(**string word1**,** string word2**)** **{**

**}**

**};**

class WordDistance **{**

public**:**

WordDistance**(**vector**<**string**>&** words**)** **{**

**for** **(**int i **=** 0**;** i **<** words**.**size**();** i**++)** **{**

m**[**words**[**i**]].**push\_back**(**i**);**

**}**

**}**

int shortest**(**string word1**,** string word2**)** **{**

vector**<**int**>** **&**u **=** m**[**word1**];**

vector**<**int**>** **&**v **=** m**[**word2**];**

int n **=** u**.**size**(),** m **=** v**.**size**();**

int i **=** 0**,** j **=** 0**,** dist **=** INT\_MAX**;**

**while** **(**i **<** n **&&** j **<** m**)** **{**

**if** **(**u**[**i**]** **<** v**[**j**])** dist **=** min**(**dist**,** v**[**j**]-**u**[**i**++]);**

**else** dist **=** min**(**dist**,** u**[**i**]-**v**[**j**++]);**

**}**

**return** dist**;**

**}**

private**:**

unordered\_map**<**string**,** vector**<**int**>>** m**;**

**};**

### 245.Shortest Word Distance III

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

word1 and word2 may be the same and they represent two individual words in the list.

**Example:**  
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

**Input:** word1 = “makes”, word2 = “coding”

**Output:** 1

**Input:** word1 = "makes", word2 = "makes"

**Output:** 3

**Note:**  
You may assume word1 and word2 are both in the list.

class Solution **{**

public**:**

int shortestWordDistance**(**vector**<**string**>&** words**,** string word1**,** string word2**)** **{**

**}**

**};**

class Solution **{**

public**:**

int shortestWordDistance**(**vector**<**string**>&** words**,** string word1**,** string word2**)** **{**

int res **=** INT\_MAX**;**

vector**<**int**>** v**[**2**];**

**for** **(**int i **=** 0**;** i **<** words**.**size**();** i**++)** **{**

**if** **(**words**[**i**]** **==** word1**)** v**[**0**].**push\_back**(**i**);**

**if** **(**words**[**i**]** **==** word2**)** v**[**1**].**push\_back**(**i**);**

**}**

int n **=** v**[**0**].**size**(),** m **=** v**[**1**].**size**();**

int i **=** 0**,** j **=** 0**;**

**while** **(**i **<** n **&&** j **<** m**)** **{**

**if** **(**v**[**0**][**i**]** **==** v**[**1**][**j**])** i**++;**

**else** **if** **(**v**[**0**][**i**]** **<** v**[**1**][**j**])** res **=** min**(**res**,** v**[**1**][**j**]-**v**[**0**][**i**++]);**

**else** res **=** min**(**res**,** v**[**0**][**i**]-**v**[**1**][**j**++]);**

**}**

**return** res**;**

**}**

**};**

### 246.Strobogrammatic Number

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

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

**Example 1:**

**Input:** "69"

**Output:** true

**Example 2:**

**Input:** "88"

**Output:** true

**Example 3:**

**Input:** "962"

**Output:** false

class Solution **{**

public**:**

bool isStrobogrammatic**(**string s**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isStrobogrammatic**(**string s**)** **{**

unordered\_map**<**char**,** char**>** m**{{**'6'**,**'9'**},** **{**'9'**,**'6'**},** **{**'8'**,**'8'**},**

**{**'1'**,**'1'**},** **{**'0'**,**'0'**}};**

int n **=** s**.**size**();**

**for** **(**int i **=** 0**;** i **<=** n**/**2**;** i**++)** **{**

**if** **(**m**[**s**[**i**]]** **!=** s**[**n**-**1**-**i**])** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

**};**

### 247.Strobogrammatic Number II

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

Find all strobogrammatic numbers that are of length = n.

**Example:**

**Input:** n = 2

**Output:** ["11","69","88","96"]

class Solution **{**

public**:**

vector**<**string**>** findStrobogrammatic**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**string**>** findStrobogrammatic**(**int n**)** **{**

**return** f**(**n**,** n**);**

**}**

vector**<**string**>** f**(**int m**,** int n**)** **{**

**if** **(**m **==** 0**)** **return** **{**""**};**

**if** **(**m **==** 1**)** **return** **{**"0"**,** "1"**,** "8"**};**

vector**<**string**>** v**(**f**(**m**-**2**,** n**)),** res**;**

**for** **(**auto **&**a **:** v**)** **{**

**if** **(**m **!=** n**)** res**.**push\_back**(**"0" **+** a **+** "0"**);**

res**.**push\_back**(**"1" **+** a **+** "1"**);**

res**.**push\_back**(**"6" **+** a **+** "9"**);**

res**.**push\_back**(**"8" **+** a **+** "8"**);**

res**.**push\_back**(**"9" **+** a **+** "6"**);**

**}**

**return** res**;**

**}**

**};**

### 248.Strobogrammatic Number III

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

Write a function to count the total strobogrammatic numbers that exist in the range of low <= num <= high.

**Example:**

**Input:** low = "50", high = "100"

**Output:** 3

**Explanation:** 69, 88, and 96 are three strobogrammatic numbers.

**Note:**  
Because the range might be a large number, the low and high numbers are represented as string.

class Solution **{**

public**:**

int strobogrammaticInRange**(**string low**,** string high**)** **{**

**}**

**};**

class Solution **{**

public**:**

int strobogrammaticInRange**(**string low**,** string high**)** **{**

int len0 **=** low**.**length**(),** len1 **=** high**.**length**();**

int res **=** 0**;**

**for** **(**int n **=** len0**;** n **<=** len1**;** n**++)** **{**

vector**<**string**>** v **=** f**(**n**,** n**);**

**if** **(**n **!=** len0 **&&** n **!=** len1**)** res **+=** v**.**size**();**

**else** **for** **(**auto **&**num **:** v**)** **{**

**if** **(**len0 **==** len1**)** **{**

**if** **(**num **>=** low **&&** num **<=** high**)** res**++;**

**}**

**else** **if** **(**n **==** len0 **&&** num **>=** low**)** res**++;**

**else** **if** **(**n **==** len1 **&&** num **<=** high**)** res**++;**

**}**

**}**

**return** res**;**

**}**

private**:**

vector**<**string**>** f**(**int m**,** int n**)** **{**

**if** **(**m **==** 0**)** **return** **{**""**};**

**if** **(**m **==** 1**)** **return** **{**"0"**,** "1"**,** "8"**};**

vector**<**string**>** v**(**f**(**m**-**2**,** n**)),** res**;**

**for** **(**auto **&**a **:** v**)** **{**

**if** **(**m **!=** n**)** res**.**push\_back**(**"0" **+** a **+** "0"**);**

res**.**push\_back**(**"1" **+** a **+** "1"**);**

res**.**push\_back**(**"6" **+** a **+** "9"**);**

res**.**push\_back**(**"8" **+** a **+** "8"**);**

res**.**push\_back**(**"9" **+** a **+** "6"**);**

**}**

**return** res**;**

**}**

**};**

### 249.Group Shifted Strings

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"]

]

class Solution **{**

public**:**

vector**<**vector**<**string**>>** groupStrings**(**vector**<**string**>&** strings**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**vector**<**string**>>** groupStrings**(**vector**<**string**>&** strings**)** **{**

unordered\_map**<**string**,** vector**<**string**>>** m**;**

**for** **(**auto **&**s **:** strings**)** **{**

string t**;**

**for** **(**auto **&**c **:** s**)** **{**

t **+=** char**((**c**-**s**[**0**]+**26**)** **%** 26**);**

**}**

m**[**t**].**push\_back**(**s**);**

**}**

vector**<**vector**<**string**>>** res**;**

**for** **(**auto **&**i **:** m**)** res**.**push\_back**(**i**.**second**);**

**return** res**;**

**}**

**};**

### 250.Count Univalue Subtrees

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

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

**Example :**

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

5

/ \

1 5

/ \ \

5 5 5

**Output:** 4

class Solution **{**

public**:**

int countUnivalSubtrees**(**TreeNode**\*** root**)** **{**

**}**

**};**

class Solution **{**

public**:**

int countUnivalSubtrees**(**TreeNode**\*** root**)** **{**

**if** **(!**root**)** **return** 0**;**

**return** f**(**root**,** root**->**val**).**second**;**

**}**

private**:**

pair**<**bool**,** int**>** f**(**TreeNode **\***root**,** int val**)** **{**

**if** **(!**root**)** **return** **{true,** 0**};**

auto l **=** f**(**root**->**left**,** root**->**val**);**

auto r **=** f**(**root**->**right**,** root**->**val**);**

int res **=** l**.**second **+** r**.**second**;**

**if** **(**l**.**first **&&** r**.**first**)** **return** **{**root**->**val **==** val**,** res**+**1**};**

**else** **return** **{false,** res**};**

**}**

**};**

### 251. Flatten 2D Vector

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:**

1. Please remember to **RESET** your class variables declared in Vector2D, as static/class variables are **persisted across multiple test cases**. Please see [here](https://leetcode.com/faq/) for more details.
2. 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++](http://www.cplusplus.com/reference/iterator/iterator/) or [iterators in Java](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html).

class Vector2D **{**

public**:**

Vector2D**(**vector**<**vector**<**int**>>** **&**v**):** x**(**v**.**begin**()),** end**(**v**.**end**())** **{}**

int next**()** **{**

hasNext**();**

**return** **(\***x**)[**y**++];**

**}**

bool hasNext**()** **{**

**while** **(**x **!=** end **&&** y **==** **(\***x**).**size**())** **{**

**++**x**;**

y **=** 0**;**

**}**

**return** x **!=** end**;**

**}**

private**:**

int y **=** 0**;**

vector**<**vector**<**int**>>::**iterator x**,** end**;**

**};**

class Vector2D **{**

public**:**

Vector2D**(**vector**<**vector**<**int**>>&** v**)** **{**

**this->**v **=** v**;**

**}**

int next**()** **{**

hasNext**();**

**return** v**[**i**][**j**++];**

**}**

bool hasNext**()** **{**

**if** **(**i **>=** v**.**size**())** **return** **false;**

**else** **if** **(**j **==** v**[**i**].**size**())** **{**

i**++;**

j **=** 0**;**

**return** hasNext**();**

**}**

**return** **true;**

**}**

private**:**

int i **=** 0**,** j **=** 0**;**

vector**<**vector**<**int**>>** v**;**

**};**

### 252. Meeting Rooms

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.

class Solution **{**

public**:**

bool canAttendMeetings**(**vector**<**vector**<**int**>>&** intervals**)** **{**

auto cmp **=** **[](**const vector**<**int**>** **&**lhs**,** const vector**<**int**>** **&**rhs**)** **{**

**return** lhs**[**0**]** **<** rhs**[**0**];**

**};**

sort**(**intervals**.**begin**(),** intervals**.**end**(),** cmp**);**

**for** **(**int i **=** 1**;** i **<** intervals**.**size**();** i**++)** **{**

**if** **(**intervals**[**i**][**0**]** **<** intervals**[**i**-**1**][**1**])** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

**};**

### 253. Meeting Rooms II

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.

class Solution **{**

public**:**

int minMeetingRooms**(**vector**<**vector**<**int**>>&** intervals**)** **{**

**}**

**};**

class Solution **{**

public**:**

int minMeetingRooms**(**vector**<**vector**<**int**>>&** intervals**)** **{**

map**<**int**,** int**>** m**;**

**for** **(**auto **&**v **:** intervals**)** **{**

**++**m**[**v**[**0**]];**

**--**m**[**v**[**1**]];**

**}**

int rooms **=** 0**,** res **=** 0**;**

**for** **(**auto **&**it **:** m**)** **{**

res **=** max**(**res**,** rooms **+=** it**.**second**);**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

int minMeetingRooms**(**vector**<**vector**<**int**>>&** intervals**)** **{**

**if** **(**intervals**.**empty**())** **return** 0**;**

auto cmp **=** **[](**const vector**<**int**>** **&**lhs**,** const vector**<**int**>** **&**rhs**)** **{**

**return** lhs**[**0**]** **<** rhs**[**0**];**

**};**

sort**(**intervals**.**begin**(),** intervals**.**end**(),** cmp**);**

priority\_queue**<**int**,** vector**<**int**>,** greater**<**int**>>** q**;**

**for** **(**auto **&**v **:** intervals**)** **{**

**if** **(!**q**.**empty**()** **&&** q**.**top**()** **<=** v**[**0**])** q**.**pop**();**

q**.**push**(**v**[**1**]);**

**}**

**return** q**.**size**();**

**}**

**};**

### 254. Factor Combinations

Numbers can be regarded as product of its factors. For example,

8 = 2 x 2 x 2;

= 2 x 4.

Write a function that takes an integer *n* and return all possible combinations of its factors.

**Note:**

1. You may assume that *n* is always positive.
2. Factors should be greater than 1 and less than *n*.

**Example 1:**

**Input:** 1

**Output:** []

**Example 2:**

**Input:** 37

**Output:**[]

**Example 3:**

**Input:** 12

**Output:**

[

[2, 6],

[2, 2, 3],

[3, 4]

]

**Example 4:**

**Input:** 32

**Output:**

[

[2, 16],

[2, 2, 8],

[2, 2, 2, 4],

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

[2, 4, 4],

[4, 8]

]

class Solution **{**

public**:**

vector**<**vector**<**int**>>** getFactors**(**int n**)** **{**

vector**<**int**>** path**;**

dfs**(**2**,** n**,** path**);**

**return** res**;**

**}**

private**:**

vector**<**vector**<**int**>>** res**;**

void dfs**(**int cur**,** int n**,** vector**<**int**>** **&**path**)** **{**

int Sqrt **=** sqrt**(**n**)+**0.5**;**

**for** **(**int i **=** cur**;** i **<=** Sqrt**;** i**++)** **{**

**if** **(**n **%** i **==** 0**)** **{**

vector**<**int**>** temp **=** path**;**

temp**.**push\_back**(**i**);**

dfs**(**i**,** n**/**i**,** temp**);**

temp**.**push\_back**(**n **/** i**);**

res**.**push\_back**(**temp**);**

**}**

**}**

**}**

**};**

### 255. Verify Preorder Sequence in Binary Search Tree

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Consider the following binary search tree:

5

/ \

2 6

/ \

1 3

**Example 1:**

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

**Output:** false

**Example 2:**

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

**Output:** true

**Follow up:**  
Could you do it using only constant space complexity?

class Solution {

public:

bool verifyPreorder(vector<int>& preorder) {

stack<int> s;

int min = INT\_MIN;

for(int i = 0; i < preorder.size(); i++){

if (preorder[i] < min) return false;

while (!s.empty() && s.top() < preorder[i]) {

min = s.top();

s.pop();

}

s.push(preorder[i]);

}

return true;

}

};

class Solution {

public:

bool verifyPreorder(vector<int>& preorder) {

int min = INT\_MIN;

int i = -1;

for (auto p : preorder) {

if (p < min) return false;

while (i >= 0 && p > preorder[i]) {

min = preorder[i--];

}

preorder[++i] = p;

}

return true;

}

};

### 256. Paint House

There are a row of *n* houses, each house can be painted with one of the three colors: red, blue or green. 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 *3* cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

**Note:**  
All costs are positive integers.

**Example:**

**Input:** [[17,2,17],[16,16,5],[14,3,19]]

**Output:** 10

**Explanation:** Paint house 0 into blue, paint house 1 into green, paint house 2 into blue.

  Minimum cost: 2 + 5 + 3 = 10.

class Solution **{**

public**:**

int minCost**(**vector**<**vector**<**int**>>&** costs**)** **{**

**}**

**};**

class Solution **{**

public**:**

int minCost**(**vector**<**vector**<**int**>>&** costs**)** **{**

**if** **(**costs**.**empty**())** **return** 0**;**

int min1 **=** 0**,** min2 **=** 0**,** idx **=** **-**1**;**

**for** **(**auto **&**cost **:** costs**)** **{**

int m1 **=** INT\_MAX**,** m2 **=** m1**,** id **=** **-**1**;**

**for** **(**int j **=** 0**;** j **<** cost**.**size**();** j**++)** **{**

int temp **=** cost**[**j**]** **+** **(**j **==** idx **?** min2 **:** min1**);**

**if** **(**temp **<** m1**)** **{**

m2 **=** m1**;** m1 **=** temp**;** id **=** j**;**

**}** **else** **if** **(**temp **<** m2**)** **{**

m2 **=** temp**;**

**}**

**}**

min1 **=** m1**;** min2 **=** m2**;** idx **=** id**;**

**}**

**return** min1**;**

**}**

**};**

### 257. Binary Tree Paths

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

/\*\*

\* 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**<**string**>** binaryTreePaths**(**TreeNode**\*** root**)** **{**

**}**

**};**

/\*\*

\* 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**<**string**>** binaryTreePaths**(**TreeNode**\*** root**)** **{**

vector**<**string**>** res**;**

**if** **(**root **==** **nullptr)** **return** res**;**

dfs**(**root**,** ""**,** res**);**

**return** res**;**

**}**

private**:**

void dfs**(**TreeNode**\*** root**,** string path**,** vector**<**string**>** **&**res**)** **{**

path **+=** to\_string**(**root**->**val**);**

**if** **(**root**->**left **==** **nullptr** **&&** root**->**right **==** **nullptr)** **{**

res**.**push\_back**(**path**);**

**return;**

**}**

path **+=** "->"**;**

**if** **(**root**->**left**)** dfs**(**root**->**left**,** path**,** res**);**

**if** **(**root**->**right**)** dfs**(**root**->**right**,** path**,** res**);**

**}**

**};**

### 258. Add Digits

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

**Example:**

**Input:** 38

**Output:** 2

**Explanation:** The process is like: 3 + 8 = 11, 1 + 1 = 2.

  Since 2 has only one digit, return it.

**Follow up:**  
Could you do it without any loop/recursion in O(1) runtime?

class Solution **{**

public**:**

int addDigits**(**int num**)** **{**

**}**

**};**

class Solution **{**

public**:**

int addDigits**(**int num**)** **{**

**while** **(**num **>** 9**)** **{**

int t **=** 0**;**

**while** **(**num **>** 0**)** **{**

t **+=** num**%**10**;**

num **/=** 10**;**

**}**

num **=** t**;**

**}**

**return** num**;**

**}**

**};**

//////////？？？？？？？？？？？？？？？？////////

class Solution **{**

public**:**

int addDigits**(**int num**)** **{**

**return** 1 **+** **(**num **-** 1**)** **%** 9**;**

**}**

**};**

### 259. Sum Smaller

Given an array of *n* integers *nums* and a *target*, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

**Example:**

**Input:** *nums* = [-2,0,1,3], and *target* = 2

**Output:** 2

**Explanation:** Because there are two triplets which sums are less than 2:

  [-2,0,1]

[-2,0,3]

**Follow up:** Could you solve it in *O*(*n*2) runtime?

class Solution **{**

public**:**

int threeSumSmaller**(**vector**<**int**>&** nums**,** int target**)** **{**

**}**

**};**

class Solution **{**

public**:**

int threeSumSmaller**(**vector**<**int**>&** nums**,** int target**)** **{**

int res **=** 0**,** n **=** nums**.**size**();**

sort**(**nums**.**begin**(),** nums**.**end**());**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

int j **=** i**+**1**,** k **=** n**-**1**;**

**while** **(**j **<** k**)** **{**

int sum **=** nums**[**i**]** **+** nums**[**j**]** **+** nums**[**k**];**

**if** **(**sum **<** target**)** res **+=** k**-(**j**++);**

**else** k**--;**

**}**

**}**

**return** res**;**

**}**

**};**

### 260. Single Number III

Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

**Example:**

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

**Output:** [3,5]

**Note**:

1. The order of the result is not important. So in the above example, [5, 3] is also correct.
2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

class Solution **{**

public**:**

vector**<**int**>** singleNumber**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** singleNumber**(**vector**<**int**>&** nums**)** **{**

int x **=** 0**,** y **=** 0**,** z **=** 0**,** bit **=** 1**;**

**for** **(**auto **&**i **:** nums**)** x **^=** i**;**

**while** **((**bit **&** x**)** **==** 0**)** bit **<<=** 1**;**

**for** **(**auto **&**i **:** nums**)** **{**

**if** **(**bit **&** i**)** y **^=** i**;**

**else** z **^=** i**;**

**}**

**return** **{**y**,** z**};**

**}**

**};**

### 261. Graph Valid Tree

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 check whether these edges make up a valid tree.

**Example 1:**

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

**Output:** true

**Example 2:**

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

**Output:** false

**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.

class Solution **{**

public**:**

bool validTree**(**int n**,** vector**<**vector**<**int**>>&** edges**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool validTree**(**int n**,** vector**<**vector**<**int**>>&** edges**)** **{**

**if** **(**edges**.**size**()** **!=** n**-**1**)** **return** **false;**

fa**.**resize**(**n**,** **-**1**);**

**for** **(**auto a **:** edges**)** **{**

int x **=** find**(**a**[**0**]),** y **=** find**(**a**[**1**]);**

**if** **(**x **==** y**)** **return** **false;**

**else** fa**[**x**]** **=** y**;**

**}**

**return** **true;**

**}**

private**:**

vector**<**int**>** fa**;**

int find**(**int i**)** **{**

**return** **(**fa**[**i**]** **==** **-**1**)** **?** i **:** **(**fa**[**i**]** **=** find**(**fa**[**i**]));**

**}**

**};**

class Solution **{**

public**:**

bool validTree**(**int n**,** vector**<**vector**<**int**>>&** edges**)** **{**

vector**<**bool**>** v**(**n**,** **false);**

vector**<**vector**<**int**>>** g**(**n**);**

**for** **(**auto **&**edge **:** edges**)** **{**

g**[**edge**[**0**]].**push\_back**(**edge**[**1**]);**

g**[**edge**[**1**]].**push\_back**(**edge**[**0**]);**

**}**

**if** **(!**dfs**(**g**,** v**,** 0**,** **-**1**))** **return** **false;**

**for** **(**auto a **:** v**)** **{**

**if** **(!**a**)** **return** **false;**

**}**

**return** **true;**

**}**

bool dfs**(**vector**<**vector**<**int**>>** **&**g**,** vector**<**bool**>** **&**v**,** int cur**,** int pre**)** **{**

**if** **(**v**[**cur**])** **return** **false;**

v**[**cur**]** **=** **true;**

**for** **(**auto a **:** g**[**cur**])** **{**

**if** **(**a **!=** pre**)** **{**

**if** **(!**dfs**(**g**,** v**,** a**,** cur**))** **return** **false;**

**}**

**}**

**return** **true;**

**}**

**};**

### 262. Trips and Users（SQL）

The Trips table holds all taxi trips. Each trip has a unique Id, while Client\_Id and Driver\_Id are both foreign keys to the Users\_Id at the Users table. Status is an ENUM type of (‘completed’, ‘cancelled\_by\_driver’, ‘cancelled\_by\_client’).

+----+-----------+-----------+---------+--------------------+----------+

| Id | Client\_Id | Driver\_Id | City\_Id | Status |Request\_at|

+----+-----------+-----------+---------+--------------------+----------+

| 1 | 1 | 10 | 1 | completed |2013-10-01|

| 2 | 2 | 11 | 1 | cancelled\_by\_driver|2013-10-01|

| 3 | 3 | 12 | 6 | completed |2013-10-01|

| 4 | 4 | 13 | 6 | cancelled\_by\_client|2013-10-01|

| 5 | 1 | 10 | 1 | completed |2013-10-02|

| 6 | 2 | 11 | 6 | completed |2013-10-02|

| 7 | 3 | 12 | 6 | completed |2013-10-02|

| 8 | 2 | 12 | 12 | completed |2013-10-03|

| 9 | 3 | 10 | 12 | completed |2013-10-03|

| 10 | 4 | 13 | 12 | cancelled\_by\_driver|2013-10-03|

+----+-----------+-----------+---------+--------------------+----------+

The Users table holds all users. Each user has an unique Users\_Id, and Role is an ENUM type of (‘client’, ‘driver’, ‘partner’).

+----------+--------+--------+

| Users\_Id | Banned | Role |

+----------+--------+--------+

| 1 | No | client |

| 2 | Yes | client |

| 3 | No | client |

| 4 | No | client |

| 10 | No | driver |

| 11 | No | driver |

| 12 | No | driver |

| 13 | No | driver |

+----------+--------+--------+

Write a SQL query to find the cancellation rate of requests made by unbanned users between **Oct 1, 2013** and **Oct 3, 2013**. For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.

+------------+-------------------+

| Day | Cancellation Rate |

+------------+-------------------+

| 2013-10-01 | 0.33 |

| 2013-10-02 | 0.00 |

| 2013-10-03 | 0.50 |

+------------+-------------------+

**Credits:**  
Special thanks to [@cak1erlizhou](https://leetcode.com/discuss/user/cak1erlizhou) for contributing this question, writing the problem description and adding part of the test cases.

### 263. Ugly Number

Easy

Write a program to check whether a given number is an ugly number.

Ugly numbers are **positive numbers** whose prime factors only include 2, 3, 5.

**Example 1:**

**Input:** 6

**Output:** true

**Explanation:** 6 = 2 × 3

**Example 2:**

**Input:** 8

**Output:** true

**Explanation:** 8 = 2 × 2 × 2

**Example 3:**

**Input:** 14

**Output:** false

**Explanation:** 14 is not ugly since it includes another prime factor 7.

**Note:**

1. 1 is typically treated as an ugly number.
2. Input is within the 32-bit signed integer range: [−231,  231− 1].

class Solution **{**

public**:**

bool isUgly**(**int num**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isUgly**(**int num**)** **{**

**if (**num **<** 1**)** **return** **false;**

int a**[**3**]** **=** **{**2**,** 3**,** 5**};**

**for (**auto **&**i **:** a**)** **{**

**while** **(**num **%** i **==** 0**)** num **/=** i**;**

**}**

**return** num **==** 1**;**

**}**

**};**

### 264. Ugly Number II

Medium

Write a program to find the n-th ugly number.

Ugly numbers are **positive numbers** whose prime factors only include 2, 3, 5.

**Example:**

**Input:** n = 10

**Output:** 12

**Explanation:** 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.

**Note:**

1. 1 is typically treated as an ugly number.
2. n **does not exceed 1690**.

class Solution **{**

public**:**

int nthUglyNumber**(**int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

int nthUglyNumber**(**int n**)** **{**

vector **<**int**>** res **{**1**};**

int i **=** 0**,** j **=** 0**,** k **=** 0**;**

**while** **(--**n**)** **{**

res**.**push\_back**(**min**(**res**[**i**]\***2**,** min**(**res**[**j**]\***3**,** res**[**k**]\***5**)));**

**if** **(**res**.**back**()** **==** res**[**i**]** **\*** 2**)** **++**i**;**

**if** **(**res**.**back**()** **==** res**[**j**]** **\*** 3**)** **++**j**;**

**if** **(**res**.**back**()** **==** res**[**k**]** **\*** 5**)** **++**k**;**

**}**

**return** res**.**back**();**

**}**

**};**

class Solution **{**

public**:**

struct cmp**{**

bool **operator** **()** **(**int **&**lhs**,** int **&**rhs**)** const **{**

**return** lhs **>** rhs**;**

**}**

**};**

int nthUglyNumber**(**int n**)** **{**

int a**[**3**]** **=** **{**2**,** 3**,** 5**};**

priority\_queue**<**int**,** vector**<**int**>,** greater**<**int**>>** pq**;**

// priority\_queue<int, vector<int>, cmp> pq;

unordered\_set**<**int**>** My\_set**;**

pq**.**push**(**1**);**

**while** **(--**n**)** **{**

int t **=** pq**.**top**();**

pq**.**pop**();**

**for** **(**int i **:** a**)** **{**

**if (**INT\_MAX **/** i **<** t**)** **continue;**

i **\*=** t**;**

**if** **(**My\_set**.**count**(**i**))** **continue;**

My\_set**.**insert**(**i**);**

pq**.**push**(**i**);**

**}**

**}**

**return** pq**.**top**();**

**}**

**};**

### 265. Paint House II

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?

class Solution **{**

public**:**

int minCostII**(**vector**<**vector**<**int**>>&** costs**)** **{**

**}**

**};**

class Solution **{**

public**:**

int minCostII**(**vector**<**vector**<**int**>>&** costs**)** **{**

**if** **(**costs**.**empty**())** **return** 0**;**

int min1 **=** 0**,** min2 **=** 0**,** idx **=** **-**1**;**

**for** **(**auto **&**cost **:** costs**)** **{**

int m1 **=** INT\_MAX**,** m2 **=** m1**,** id **=** **-**1**;**

**for** **(**int j **=** 0**;** j **<** cost**.**size**();** j**++)** **{**

int temp **=** cost**[**j**]** **+** **(**j **==** idx **?** min2 **:** min1**);**

**if** **(**temp **<** m1**)** **{**

m2 **=** m1**;** m1 **=** temp**;** id **=** j**;**

**}** **else** **if** **(**temp **<** m2**)** **{**

m2 **=** temp**;**

**}**

**}**

min1 **=** m1**;** min2 **=** m2**;** idx **=** id**;**

**}**

**return** min1**;**

**}**

**};**

### 266. Palindrome Permutation

Given a string, determine if a permutation of the string could form a palindrome.

**Example 1:**

**Input:** "code"

**Output:** false

**Example 2:**

**Input:** "aab"

**Output:** true

**Example 3:**

**Input:** "carerac"

**Output:** true

class Solution **{**

public**:**

bool canPermutePalindrome**(**string s**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool canPermutePalindrome**(**string s**)** **{**

bitset**<**256**>** b**;**

**for** **(**auto **&**c **:** s**)** **{**

b**.**flip**(**c**);**

**}**

**return** b**.**count**()** **<** 2**;**

**}**

**};**

class Solution **{**

public**:**

bool canPermutePalindrome**(**string s**)** **{**

int n **=** s**.**length**();**

vector**<**bool**>** v**(**256**,** **false);**

**for** **(**auto **&**c **:** s**)** **{**

**if** **(**v**[**c**])** n **-=** 2**;**

v**[**c**]** **=** **!**v**[**c**];**

**}**

**return** n **<=** 1**;**

**}**

**};**

### 267. Palindrome Permutation II

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:** []

class Solution **{**

public**:**

vector**<**string**>** generatePalindromes**(**string s**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**string**>** generatePalindromes**(**string s**)** **{**

**for** **(**auto **&**c **:** s**)** m**[**c**]++;**

int cnt **=** 0**,** n **=** s**.**length**();**

char mid**;**

**for** **(**auto **&**i **:** m**)** **if** **(**i**.**second **%** 2**)** **{**

**if** **(++**cnt **>** 1**)** **return** res**;**

**else** mid **=** i**.**first**;**

**}**

**if** **(**cnt **==** 1**)** m**[**s**[**n**/**2**]** **=** mid**]--;**

dfs**(**0**,** n**,** s**);**

**return** res**;**

**}**

private**:**

unordered\_map**<**char**,** int**>** m**;**

vector**<**string**>** res**;**

void dfs**(**int i**,** int n**,** string **&**s**)** **{**

**if** **(**i **==** n**/**2**)** res**.**push\_back**(**s**);**

**else** **for** **(**auto **&**it **:** m**)** **if** **(**it**.**second **>=** 2**)** **{**

s**[**i**]** **=** s**[**n**-**1**-**i**]** **=** it**.**first**;**

it**.**second **-=** 2**;**

dfs**(**i**+**1**,** n**,** s**);**

it**.**second **+=** 2**;**

**}**

**}**

**};**

### 268. Missing Number

Easy

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?

class Solution **{**

public**:**

int missingNumber**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

int missingNumber**(**vector**<**int**>&** nums**)** **{**

int n **=** nums**.**size**(),** res **=** n**;**

**for (**int i **=** 0**;** i **<** n**;** i**++)** res **^=** i **^** nums**[**i**];**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

int missingNumber**(**vector**<**int**>&** nums**)** **{**

sort **(**nums**.**begin**(),** nums**.**end**());**

int left **=** 0**,** right **=** nums**.**size**()-**1**;**

**while** **(**left **<** right**)** **{**

int mid **=** left **+** **(**right**-**left**)/**2**;**

**if** **(**nums**[**mid**]** **==** mid**)** left **=** mid**+**1**;**

**else** right **=** mid**;**

**}**

**if** **(**nums**[**left**]** **==** left**)** **return** nums**.**size**();**

**else** **return** nums**[**left**]-**1**;**

**}**

**};**

### 269. Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where **words are sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

**Example 1:**

**Input:**

[

"wrt",

"wrf",

"er",

"ett",

"rftt"

]

**Output:** "wertf"

**Example 2:**

**Input:**

[

"z",

"x"

]

**Output:** "zx"

**Example 3:**

**Input:**

[

"z",

"x",

"z"

]

**Output:** ""

**Explanation:** The order is invalid, so return "".

**Note:**

1. You may assume all letters are in lowercase.
2. You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return any one of them is fine.

class Solution **{**

public**:**

string alienOrder**(**vector**<**string**>&** words**)** **{**

g**.**resize**(**256**);**

v**.**resize**(**256**,** 1**);**

f**(**0**,** 0**,** words**.**size**()-**1**,** words**);**

**for** **(**auto **&**s **:** words**)** **for** **(**auto **&**c **:** s**)** v**[**c**]** **=** 0**;**

string res**;**

**for** **(**int i **=** 0**;** i **<** 256**;** i**++)** **if** **(!**v**[**i**])** **{**

**if** **(!**topo**(**i**,** res**))** **return** ""**;**

**}**

**return** res**;**

**}**

private**:**

vector**<**vector**<**int**>>** g**;**

vector**<**int**>** v**;**

void f**(**int pos**,** int st**,** int ed**,** vector**<**string**>** **&**words**)** **{**

**if** **(**st **>=** ed **||** words**[**st**].**length**()** **<** pos**)** **return;**

char pre **=** words**[**st**][**pos**];**

int new\_st **=** st**;**

**for** **(**int i **=** st**;** i **<=** ed**;** i**++)** **{**

string **&**s **=** words**[**i**];**

**if** **(**s**[**pos**]** **!=** pre**)** **{**

g**[**s**[**pos**]].**push\_back**(**pre**);**

pre **=** s**[**pos**];**

f**(**pos**+**1**,** new\_st**,** i**-**1**,** words**);**

new\_st **=** i**;**

**}**

**}**

f**(**pos**+**1**,** new\_st**,** ed**,** words**);**

**}**

bool topo**(**int i**,** string **&**res**)** **{**

v**[**i**]** **=** **-**1**;**

**for** **(**auto **&**j **:** g**[**i**])** **if** **(**v**[**j**]** **!=** 1**)** **{**

**if** **(**v**[**j**]** **==** **-**1 **||** **!**topo**(**j**,** res**))** **return** **false;**

**}**

v**[**i**]** **=** 1**;**

res **+=** char**(**i**);**

**return** **true;**

**}**

**};**

### 270. Closest Binary Search Tree Value

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

/\*\*

\* 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**)** **{**

**}**

**};**

/\*\*

\* 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**)** **{**

dfs**(**root**,** target**);**

**return** res**;**

**}**

private**:**

int res**;**

double sub **=** numeric\_limits**<**double**>::**max**();**

void dfs**(**TreeNode**\*** root**,** double target**)** **{**

**if** **(!**root**)** **return;**

**if** **(**sub **>** fabs**(**target**-**root**->**val**))** **{**

sub **=** fabs**(**target**-**root**->**val**);**

res **=** root**->**val**;**

**}**

**if** **(**target **>** root**->**val**)** dfs**(**root**->**right**,** target**);**

**else** **if** **(**target **<** root**->**val**)** dfs**(**root**->**left**,** target**);**

**}**

**};**

### 271. Encode and Decode Strings

Design an algorithm to encode **a list of strings** to **a string**. The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

string encode(vector<string> strs) {

// ... your code

return encoded\_string;

}

Machine 2 (receiver) has the function:

vector<string> decode(string s) {

//... your code

return strs;

}

So Machine 1 does:

string encoded\_string = encode(strs);

and Machine 2 does:

vector<string> strs2 = decode(encoded\_string);

strs2 in Machine 2 should be the same as strs in Machine 1.

Implement the encode and decode methods.

**Note:**

* The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
* Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.
* Do not rely on any library method such as eval or serialize methods. You should implement your own encode/decode algorithm.

class Codec {

public:

// Encodes a list of strings to a single string.

string encode(vector<string>& strs) {

string res;

for (auto s : strs) {

string t(to\_string(s.length()));

res += string(3 - t.length(), '0') + t + s;

}

return res;

}

// Decodes a single string to a list of strings.

vector<string> decode(string s) {

vector<string> res;

int i = 0, n = s.length();

while (i < n) {

int k = stoi(s.substr(i, 3));

res.push\_back(s.substr(i+3, k));

i += 3 + k;

}

return res;

}

};

### 272. Closest Binary Search Tree Value II

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

**Note:**

* Given target value is a floating point.
* You may assume *k* is always valid, that is: *k* ≤ total nodes.
* You are guaranteed to have only one unique set of *k* values in the BST that are closest to the target.

**Example:**

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

4

/ \

2 5

/ \

1 3

**Output:** [4,3]

**Follow up:**  
Assume that the BST is balanced, could you solve it in less than *O*(*n*) runtime (where *n* = total nodes)?

class Solution **{**

public**:**

vector**<**int**>** closestKValues**(**TreeNode**\*** root**,** double target**,** int k**)** **{**

vector**<**int**>** res**;**

stack**<**TreeNode**\*>** s**;**

TreeNode **\***p **=** root**;**

**while** **(**p **||** **!**s**.**empty**())** **{**

**while** **(**p**)** **{**

s**.**push**(**p**);**

p **=** p**->**left**;**

**}**

p **=** s**.**top**();** s**.**pop**();**

**if** **(**res**.**size**()** **<** k**)** res**.**push\_back**(**p**->**val**);**

**else** **if** **(**abs**(**p**->**val **-** target**)** **<** abs**(**res**[**0**]** **-** target**))** **{**

res**.**erase**(**res**.**begin**());**

res**.**push\_back**(**p**->**val**);**

**}** **else** **break;**

p **=** p**->**right**;**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

struct node **{**

double sub**;**

int value**;**

node**(**double s**,** int v**):**sub**(**s**),** value**(**v**){}**

bool **operator** **<** **(**const node **&**rhs**)** const **{**

**return** sub **<** rhs**.**sub**;**

**};**

**};**

vector**<**int**>** closestKValues**(**TreeNode**\*** root**,** double target**,** int k**)** **{**

dfs**(**root**,** target**,** k**);**

vector**<**int**>** res**;**

**while** **(!**q**.**empty**())** **{**

res**.**push\_back**(**q**.**top**().**value**);**

q**.**pop**();**

**}**

**return** res**;**

**}**

private**:**

priority\_queue**<**node**>** q**;**

void dfs**(**TreeNode**\*** root**,** double target**,** int k**)** **{**

**if** **(!**root**)** **return;**

double sub **=** q**.**size**()** **<** k **?** numeric\_limits**<**double**>::**max**()**

**:** q**.**top**().**sub**;**

double a **=** fabs**(**target**-**root**->**val**);**

**if** **(**sub **>** a**)** **{**

q**.**push**({**a**,** root**->**val**});**

**if** **(**q**.**size**()** **>** k**)** q**.**pop**();**

**}**

**if** **(**q**.**size**()** **<** k **||** a **!=** q**.**top**().**sub **||** target **<** root**->**val**)**

dfs**(**root**->**left**,** target**,** k**);**

**if** **(**q**.**size**()** **<** k **||** a **!=** q**.**top**().**sub **||** target **>** root**->**val**)**

dfs**(**root**->**right**,** target**,** k**);**

**}**

**};**

### 273. Integer to English Words

Hard

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 231 - 1.

**Example 1:**

**Input:** 123

**Output:** "One Hundred Twenty Three"

**Example 2:**

**Input:** 12345

**Output:** "Twelve Thousand Three Hundred Forty Five"

**Example 3:**

**Input:** 1234567

**Output:** "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

**Example 4:**

**Input:** 1234567891

**Output:** "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand Eight Hundred Ninety One"

class Solution **{**

public**:**

string numberToWords**(**int num**)** **{**

**if** **(**num **==** 0**)** **return** "Zero"**;**

string s**[**3**]** **=** **{**"Billion"**,** "Million"**,** "Thousand"**},** res**;**

**for** **(**int i **=** 0**;** i **<=** 3**;** i**++)** **{**

int k **=** pow**(**10**,**9**-**3**\***i**),** t **=** num **/** k**;**

**if** **(**t **>** 0**)** **{**

read**(**t**,** res**);**

**if** **(**i **!=** 3**)** res **+=** s**[**i**]** **+** ' '**;**

**}**

num **%=** k**;**

**}**

res**.**erase**(**prev**(**res**.**end**()));**

**return** res**;**

**}**

private**:**

void read**(**int n**,** string **&**res**)** **{**

string a**[**10**]** **=** **{**""**,** ""**,** "Twenty"**,** "Thirty"**,** "Forty"**,** "Fifty"**,**

"Sixty"**,** "Seventy"**,** "Eighty"**,** "Ninety"**};**

string b**[**10**]** **=** **{**""**,** "One"**,** "Two"**,** "Three"**,** "Four"**,** "Five"**,**

"Six"**,** "Seven"**,** "Eight"**,** "Nine"**};**

string c**[**10**]** **=** **{**"Ten"**,** "Eleven"**,** "Twelve"**,** "Thirteen"**,** "Fourteen"**,**

"Fifteen"**,** "Sixteen"**,** "Seventeen"**,** "Eighteen"**,** "Nineteen"**};**

**if** **(**n**/**100 **>** 0**)** res **+=** b**[**n**/**100**]** **+** ' ' **+** "Hundred "**;**

n **%=** 100**;**

**if** **(**n**/**10 **==** 1**)** res **+=** c**[**n**-**10**]** **+** ' '**;**

**else** **{**

**if** **(**n**/**10**)** res **+=** a**[**n**/**10**]** **+** ' '**;**

**if** **(**n**%**10**)** res **+=** b**[**n**%**10**]** **+** ' '**;**

**}**

**}**

**};**

### 274. H-Index(●ˇ∀ˇ●)

Medium

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](https://en.wikipedia.org/wiki/H-index): "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.

class Solution **{**

public**:**

int hIndex**(**vector**<**int**>&** citations**)** **{**

**}**

**};**

class Solution **{**

public**:**

int hIndex**(**vector**<**int**> &**citations**)** **{**

const int n **=** citations**.**size**();**

vector**<**int**>** buckets**(**n**+**1**,** 0**);**

**for (**int c **:** citations**)** **{**

**if (**c **>=** n**)** buckets**[**n**]++;**

**else** buckets**[**c**]++;**

**}**

int cnt **=** 0**;**

**for** **(**int i **=** n**;** i **>=** 0**;** i**--)** **{**

cnt **+=** buckets**[**i**];**

**if (**cnt **>=** i**)** **return** i**;**

**}**

**return** 0**;**

**}**

**};**

### 275. H-Index II

Medium

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

According to the [definition of h-index on Wikipedia](https://en.wikipedia.org/wiki/H-index): "A scientist has index *h* if *h* of his/her *N* papers have **at least** *h* citations each, and the other *N − h* papers have **no more than** *h*citations each."

**Example:**

**Input:** citations = [0,1,3,5,6]

**Output:** 3

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

received 0, 1, 3, 5, 6 citations respectively.

  Since the researcher has 3 papers with **at least** 3 citations each and the remaining

  two with **no more than** 3 citations each, her h-index is 3.

**Note:**

If there are several possible values for *h*, the maximum one is taken as the h-index.

**Follow up:**

* This is a follow up problem to [H-Index](https://leetcode.com/problems/h-index/description/), where citations is now guaranteed to be sorted in ascending order.
* Could you solve it in logarithmic time complexity?

class Solution **{**

public**:**

int hIndex**(**vector**<**int**>&** citations**)** **{**

**}**

**};**

class Solution **{**

public**:**

int hIndex**(**vector**<**int**>&** citations**)** **{**

int n **=** citations**.**size**();**

**if** **(!**n /\*|| citations.back() < 1\*/**)** **return** 0**;**

int l **=** 0**,** r **=** n**;**

**while** **(**l **<** r**)** **{**

int mid **=** l **+** **(**r**-**l**)/**2**;**

**if** **(**citations**[**mid**]** **<** n**-**mid**)** l **=** mid**+**1**;**

**else** r **=** mid**;**

**}**

**return** n**-**l**;**

**}**

**};**

### 276. Paint Fence

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

class Solution **{**

public**:**

int numWays**(**int n**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

int numWays**(**int n**,** int k**)** **{**

**if** **(**n **==** 0**)** **return** 0**;**

int f\_0 **=** 0**,** f\_1 **=** k**;**

// f\_0 = 前两次相同颜色\*(k-1) + 一次颜色相同\*(k-1);

// f\_1 = 一次颜色相同

**for** **(**int i **=** 1**;** i **<** n**;** i**++)** **{**

int temp **=** **(**f\_0 **+** f\_1**)** **\*** **(**k**-**1**);**

f\_0 **=** f\_1**;**

f\_1 **=** temp**;**

**}**

**return** f\_0 **+** f\_1**;**

**}**

**};**

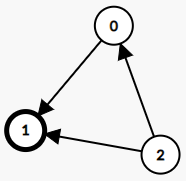
### 277. Find the Celebrity

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n). There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

**Example 1:**



**Input:** graph = [

  [1,1,0],

  [0,1,0],

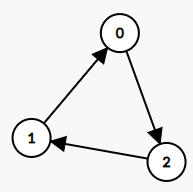
  [1,1,1]

]

**Output:** 1

**Explanation:** There are three persons labeled with 0, 1 and 2. graph[i][j] = 1 means person i knows person j, otherwise graph[i][j] = 0 means person i does not know person j. The celebrity is the person labeled as 1 because both 0 and 2 know him but 1 does not know anybody.

**Example 2:**



**Input:** graph = [

  [1,0,1],

  [1,1,0],

  [0,1,1]

]

**Output:** -1

**Explanation:** There is no celebrity.

**Note:**

1. The directed graph is represented as an adjacency matrix, which is an n x n matrix where a[i][j] = 1 means person i knows person j while a[i][j] = 0 means the contrary.
2. Remember that you won't have direct access to the adjacency matrix.

// Forward declaration of the knows API.

bool knows(int a, int b);

class Solution {

private:

public:

int findCelebrity(int n) {

int left = 0, right = n - 1;

while (left < right) {

if (knows(left, right)) ++left;

else --right;

}

for (int i = 0; i < n; i ++) {

if (i != left && (!knows(i, left) || knows(left, i)))

return -1;

}

return left;

}

};

### 278. First Bad Version

Easy

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.

// Forward declaration of isBadVersion API.

bool isBadVersion**(**int version**);**

class Solution **{**

public**:**

int firstBadVersion**(**int n**)** **{**

**}**

**};**

bool isBadVersion**(**int version**);**

class Solution **{**

public**:**

int firstBadVersion**(**int n**)** **{**

int l **=** 1**,** r **=** n**;**

**while** **(**l **<** r**)** **{**

int mid **=** l**+(**r**-**l**)/**2**;**

**if** **(**isBadVersion**(**mid**))** r **=** mid**;**

**else** l **=** mid**+**1**;**

**}**

**return** l**;**

**}**

**};**

### 279. Perfect Squares(●ˇ∀ˇ●)

Medium

Given a positive integer *n*, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to *n*.

**Example 1:**

**Input:** *n* = 12

**Output:** 3

**Explanation:** 12 = 4 + 4 + 4.

**Example 2:**

**Input:** *n* = 13

**Output:** 2

**Explanation:** 13 = 4 + 9.

class Solution **{**

public**:**

int numSquares**(**int n**)** **{**

vector**<**int**>** dp**(**n**+**1**);**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** **{**

dp**[**i**]** **=** i**;**

**for** **(**int j **=** 1**;** j**\***j **<=** i**;** j**++)** **{**

dp**[**i**]** **=** min**(**dp**[**i**],** dp**[**i**-**j**\***j**]+**1**);**

**}**

**}**

**return** dp**[**n**];**

**}**

**};**

### 280. Wiggle Sort

Given an unsorted array nums, reorder it **in-place** such that nums[0] <= nums[1] >= nums[2] <= nums[3]....

**Example:**

**Input:** nums = [3,5,2,1,6,4]

**Output:** One possible answer is [3,5,1,6,2,4]

class Solution **{**

public**:**

void wiggleSort**(**vector**<**int**>** **&**nums**)** **{**

int n **=** nums**.**size**();**

**if** **(**n **<=** 1**)** **return;**

**for** **(**int i **=** 1**;** i **<** n**;** **++**i**)** **{**

**if** **(**i**%**2 **&&** nums**[**i**]** **<** nums**[**i**-**1**]** **||** **!(**i**%**2**)** **&&** nums**[**i**]** **>** nums**[**i**-**1**])** **{**

swap**(**nums**[**i**],** nums**[**i**-**1**]);**

**}**

**}**

**}**

**};**

### 281. Zigzag Iterator

Given two 1d vectors, implement an iterator to return their elements alternately.

**Example:**

**Input:**

v1 = [1,2]

v2 = [3,4,5,6]

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

**Explanation:** By calling *next* repeatedly until *hasNext* returns false,

  the order of elements returned by *next* should be: [1,3,2,4,5,6].

**Follow up**: What if you are given k 1d vectors? How well can your code be extended to such cases?

**Clarification for the follow up question:**  
The "Zigzag" order is not clearly defined and is ambiguous for k > 2 cases. If "Zigzag" does not look right to you, replace "Zigzag" with "Cyclic". For example:

**Input:**

[1,2,3]

[4,5,6,7]

[8,9]

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

class ZigzagIterator **{**

public**:**

ZigzagIterator**(**vector**<**int**>&** v1**,** vector**<**int**>&** v2**)** **{**

**if** **(!**v1**.**empty**())** q**.**push**(**make\_pair**(**v1**.**begin**(),** v1**.**end**()));**

**if** **(!**v2**.**empty**())** q**.**push**(**make\_pair**(**v2**.**begin**(),** v2**.**end**()));**

**}**

int next**()** **{**

auto it **=** q**.**front**().**first**,** end **=** q**.**front**().**second**;**

q**.**pop**();**

**if** **(**it **+** 1 **!=** end**)** q**.**push**(**make\_pair**(**it **+** 1**,** end**));**

**return** **\***it**;**

**}**

bool hasNext**()** **{**

**return** **!**q**.**empty**();**

**}**

private**:**

queue**<**pair**<**vector**<**int**>::**iterator**,** vector**<**int**>::**iterator**>>** q**;**

**};**

### 282. Expression Add Operators

Hard

Given a string that contains only digits 0-9 and a target value, return all possibilities to add **binary** operators (not unary) +, -, or \* between the digits so they evaluate to the target value.

**Example 1:**

**Input:** *num* = "123", *target* = 6

**Output:** ["1+2+3", "1\*2\*3"]

**Example 2:**

**Input:** *num* = "232", *target* = 8

**Output:** ["2\*3+2", "2+3\*2"]

**Example 3:**

**Input:** *num* = "105", *target* = 5

**Output:** ["1\*0+5","10-5"]

**Example 4:**

**Input:** *num* = "00", *target* = 0

**Output:** ["0+0", "0-0", "0\*0"]

**Example 5:**

**Input:** *num* = "3456237490", *target* = 9191

**Output:** []

class Solution **{**

public**:**

vector**<**string**>** addOperators**(**string num**,** int target**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**string**>** addOperators**(**string num**,** int target**)** **{**

**if** **(**num**.**empty**())** **return** **{};**

string expr**(**num**.**size**()\***2**,** '\0'**);**

**this->**num **=** num**;**

**this->**target **=** target**;**

dfs**(**0**,** expr**,** 0**,** 0**,** 0**);**

**return** res**;**

**}**

private**:**

string num**;**

int target**;**

vector**<**string**>** res**;**

void dfs**(**int pos**,** string **&**expr**,** int len**,** long prev**,** long cur**)** **{**

**if** **(**pos **==** num**.**size**())** **{**

**if** **(**cur **==** target**)** res**.**push\_back**(**expr**.**substr**(**0**,** len**));**

**return;**

**}**

long n **=** 0**;**

int start **=** pos**;**

int l **=** len**;**

**if** **(**start **!=** 0**)** **++**len**;** // leave the place for operator

**while** **(**pos **<** num**.**size**())** **{**

n **=** 10**\***n **+** num**[**pos**]-**'0'**;**

**if** **(**num**[**start**]** **==** '0' **&&** pos **>** start**)** **break;**

expr**[**len**++]** **=** num**[**pos**++];**

**if** **(**start **==** 0**)** dfs**(**pos**,** expr**,** len**,** n**,** n**);**

**else** **{**

expr**[**l**]** **=** '+'**;;**

dfs**(**pos**,** expr**,** len**,** n**,** cur**+**n**);**

expr**[**l**]** **=** '-'**;**

dfs**(**pos**,** expr**,** len**,** **-**n**,** cur**-**n**);**

expr**[**l**]** **=** '\*'**;**

dfs**(**pos**,** expr**,** len**,** prev**\***n**,** cur**-**prev**+**prev**\***n**);**

**}**

**}**

**}**

**};**

### 283. Move Zeroes

Easy

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

**Example:**

**Input:** [0,1,0,3,12]

**Output:** [1,3,12,0,0]

**Note**:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

class Solution **{**

public**:**

void moveZeroes**(**vector**<**int**>&** nums**)** **{**

int cnt **=** 0**,** n **=** nums**.**size**();**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**if** **(**nums**[**i**]** **!=** 0**)** **{**

nums**[**cnt**++]** **=** nums**[**i**];**

**}**

**}**

**while** **(**cnt **<** n**)** nums**[**cnt**++]** **=** 0**;**

**}**

**};**

### 284. Peeking Iterator

Medium

Given an Iterator class interface with methods: next() and hasNext(), design and implement a PeekingIterator that support the peek() operation -- it essentially peek() at the element that will be returned by the next call to next().

**Example:**

Assume that the iterator is initialized to the beginning of the list: **[1,2,3]**.

Call **next()** gets you **1**, the first element in the list.

Now you call **peek()** and it returns **2**, the next element. Calling **next()** after that ***still*** return **2**.

You call **next()** the final time and it returns **3**, the last element.

Calling **hasNext()** after that should return **false**.

**Follow up**: How would you extend your design to be generic and work with all types, not just integer?

class Iterator **{**

struct Data**;**

Data**\*** data**;**

public**:**

Iterator**(**const vector**<**int**>&** nums**);**

Iterator**(**const Iterator**&** iter**);**

virtual **~**Iterator**();**

int next**();**

bool hasNext**()** const**;**

**};**

class PeekingIterator **:** public Iterator **{**

public**:**

PeekingIterator**(**const vector**<**int**>&** nums**):**Iterator**(**nums**),** isPeek**(false)** **{}**

int peek**()** **{**

//return Iterator(\*this).next();

**if** **(**isPeek**)** **return** next\_element**;**

isPeek **=** **true;**

**return** next\_element **=** Iterator**::**next**();**

**}**

int next**()** **{**

**if** **(**isPeek**)** **{**

isPeek **=** **false;**

**return** next\_element**;**

**}**

**else** **return** Iterator**::**next**();**

**}**

bool hasNext**()** const **{**

**if** **(**isPeek**)** **return** **true;**

**else** **return** Iterator**::**hasNext**();**

**}**

private**:**

bool isPeek**;**

int next\_element**;**

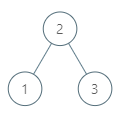
**};**

### 285. Inorder Successor in BST

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:**

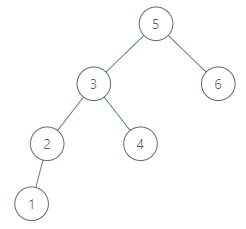


**Input:** root = [2,1,3], p = 1

**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:**

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

/\*\*

\* 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) {

if (!root) return nullptr;

if (root->val > p->val) {

auto q = inorderSuccessor(root->left, p);

return q ? q : root;

}

else {

return inorderSuccessor(root->right, p);

}

}

};

### 286. Walls and Gates

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

1. -1 - A wall or an obstacle.
2. 0 - A gate.
3. 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

class Solution **{**

public**:**

void wallsAndGates**(**vector**<**vector**<**int**>>&** rooms**)** **{**

**if** **(**rooms**.**empty**())** **return;**

n **=** rooms**.**size**(),** m **=** rooms**[**0**].**size**();**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** j**++)** **{**

**if** **(**rooms**[**i**][**j**]** **==** 0**)** dfs**(**i**,** j**,** 0**,** rooms**);**

**}**

**}**

**}**

private**:**

int n**,** m**;**

vector**<**int**>** dx**{**0**,**0**,-**1**,**1**};**

vector**<**int**>** dy**{-**1**,**1**,**0**,**0**};**

void dfs**(**int x**,** int y**,** int cnt**,** vector**<**vector**<**int**>>** **&**rooms**)** **{**

**for** **(**int k **=** 0**;** k **<** 4**;** k**++)** **{**

int xx **=** x **+** dx**[**k**],** yy **=** y **+** dy**[**k**];**

**if** **(**xx **>=** 0 **&&** yy **>=** 0 **&&** xx **<** n **&&** yy **<** m **&&** rooms**[**xx**][**yy**]** **>** 0**)** **{**

**if** **(**rooms**[**xx**][**yy**]** **>** cnt**+**1**)** **{**

dfs**(**xx**,** yy**,** rooms**[**xx**][**yy**]** **=** cnt**+**1**,** rooms**);**

**}**

**}**

**}**

**}**

**};**

### 287. Find the Duplicate Number(●ˇ∀ˇ●)

Medium

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:**

1. You **must not** modify the array (assume the array is read only).
2. You must use only constant, *O*(1) extra space.
3. Your runtime complexity should be less than *O*(*n*2).
4. There is only one duplicate number in the array, but it could be repeated more than once.

class Solution **{**

public**:**

int findDuplicate**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

int findDuplicate**(**vector**<**int**>&** nums**)** **{**

int slow **=** 0**,** fast **=** 0**;**

**while** **(**1**)** **{**

slow **=** nums**[**slow**];**

fast **=** nums**[**nums**[**fast**]];**

**if** **(**slow **==** fast**)** **break;**

**}**

fast **=** 0**;**

**while** **(**1**)** **{**

slow **=** nums**[**slow**];**

fast **=** nums**[**fast**];**

**if** **(**slow **==** fast**)** **return** slow**;**

**}**

**}**

**};**

### 288. Unique Word Abbreviation

An abbreviation of a word follows the form <first letter><number><last letter>. Below are some examples of word abbreviations:

a) it --> it (no abbreviation)

1

↓

b) d|o|g --> d1g

1 1 1

1---5----0----5--8

↓ ↓ ↓ ↓ ↓

c) i|nternationalizatio|n --> i18n

1

1---5----0

  ↓ ↓ ↓

d) l|ocalizatio|n --> l10n

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no *other* word from the dictionary has the same abbreviation.

**Example:**

Given dictionary = [ "deer", "door", "cake", "card" ]

isUnique("dear") -> false

isUnique("cart") -> true

isUnique("cane") -> false

isUnique("make") -> true

class ValidWordAbbr **{**

public**:**

ValidWordAbbr**(**vector**<**string**>&** dictionary**)** **{**

**for(**auto **&**s **:** dictionary**)** **{**

string a **=** s**[**0**]** **+** to\_string**(**s**.**length**()-**2**)** **+** s**.**back**();**

**if** **(**m**.**count**(**a**)** **&&** m**[**a**]** **!=** s**)** m**[**a**]** **=** ""**;**

**else** m**[**a**]** **=** s**;**

**}**

**}**

bool isUnique**(**string word**)** **{**

string s **=** word**[**0**]** **+** to\_string**(**word**.**length**()-**2**)** **+** word**.**back**();**

**return** **!**m**.**count**(**s**)** **||** m**[**s**]** **==** word**;**

**}**

private**:**

unordered\_map**<**string**,** string**>** m**;**

**};**

### 289. Game of Life

Medium

According to the [Wikipedia's article](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life): "The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a *board* with *m* by *n* cells, each cell has an initial state *live* (1) or *dead* (0). Each cell interacts with its [eight neighbors](https://en.wikipedia.org/wiki/Moore_neighborhood) (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population..
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state. The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously.

**Example:**

**Input:**

[ [0,1,0],

  [0,0,1],

  [1,1,1],

  [0,0,0]]

**Output:**

[ [0,0,0],

  [1,0,1],

  [0,1,1],

  [0,1,0]]

**Follow up**:

1. Could you solve it in-place? Remember that the board needs to be updated at the same time: You cannot update some cells first and then use their updated values to update other cells.
2. In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array. How would you address these problems?

// 2 live -> dead

// -1 dead-> live

// 1 live -> live

// 0 dead -> dead

class Solution **{**

public**:**

int n**,** m**;**

const int dx**[**8**]** **=** **{**1**,**1**,**1**,**0**,**0**,-**1**,-**1**,-**1**};**

const int dy**[**8**]** **=** **{**1**,**0**,-**1**,**1**,-**1**,**1**,**0**,-**1**};**

bool inside**(**int x**,** int y**){**

**return** **(**x **>=** 0 **&&** x **<** n **&&** y **>=** 0 **&&** y **<** m**);**

**}**

void gameOfLife**(**vector**<**vector**<**int**>>&** board**)** **{**

n **=** board**.**size**(),** m **=** board**[**0**].**size**();**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** j**++)** **{**

int cnt **=** 0**;**

**for** **(**int k **=** 0**;** k **<** 8**;** k**++){**

int x **=** i**+**dx**[**k**],** y **=** j**+**dy**[**k**];**

**if (**inside**(**x**,** y**)** **&&** board**[**x**][**y**]** **>** 0**)** cnt**++;**

**}**

**if** **(**board**[**i**][**j**]** **>** 0**)** board**[**i**][**j**] = (**cnt **<** 2 **||** cnt **>** 3**)** **?** 2 **:**1**;**

**else** board**[**i**][**j**]** **=** **(**cnt **==** 3**)** **?** **-**1 **:** 0**;**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** j**++){**

board**[**i**][**j**]** **=** **(**board**[**i**][**j**]** **==** 2 **||** board**[**i**][**j**]** **==** 0**)** **?** 0 **:** 1**;**

**}**

**}**

**}**

**};**

### 290. Word Pattern

Easy

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 separated by a single space.

class Solution **{**

public**:**

bool wordPattern**(**string pattern**,** string str**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool wordPattern**(**string pattern**,** string str**)** **{**

stringstream ss**(**str**),** sss**(**pattern**);**

int cnt **=** 0**;**

char c**;**

unordered\_map**<**char**,** int**>** m1**;**

unordered\_map**<**string**,** int**>** m2**;**

**while** **(**ss **>>** str**)** **{**

**if (!(**sss **>>** c**))** **return** **false;**

**if (**m2**[**str**]** **!=** m1**[**c**])** **return** **false;**

m1**[**c**]** **=** m2**[**str**]** **=** **++**cnt**;**

**}**

**return** !(sss **>>** c)**;**

**}**

**};**

### 291. Word Pattern II

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** substring in str.

**Example 1:**

**Input:** pattern = "abab", str = "redblueredblue"

**Output:** true

**Example 2:**

**Input:** pattern = pattern = "aaaa", str = "asdasdasdasd"

**Output:** true

**Example 3:**

**Input:** pattern = "aabb", str = "xyzabcxzyabc"

**Output:** false

**Notes:**  
You may assume both pattern and str contains only lowercase letters.

class Solution {

public:

bool wordPatternMatch(string pattern, string str) {

v.resize(26, "");

return dfs(0, pattern.size(), 0, str.size(), pattern, str);

}

private:

vector<string> v;

unordered\_set<string> myset;

bool dfs(int i, int n, int j, int m, string &pattern, string &str) {

if (i == n || j == m) {

return i == n && j == m;

}

int c = pattern[i] - 'a';

if (v[c] != "") {

int len = v[c].size();

if (m-j < len || str.substr(j, len) != v[c]) return false;

return dfs(i+1, n, j+len, m, pattern, str);

}

for (int k = j+1; k <= m; ++k) {

v[c] = str.substr(j, k-j);

if (myset.count(v[c])) continue;

myset.insert(v[c]);

if (dfs(i+1, n, k, m, pattern, str)) return true;

myset.erase(v[c]);

}

v[c] = "";

return false;

}

};

### 292. Nim Game

Easy

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

**Example:**

**Input:** 4

**Output:** false

**Explanation:** If there are 4 stones in the heap, then you will never win the game;

  No matter 1, 2, or 3 stones you remove, the last stone will always be

  removed by your friend.

class Solution **{**

public**:**

bool canWinNim**(**int n**)** **{**

**return** n**%**4 **!=** 0**;**

**}**

**};**

### 293. Flip Game

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two **consecutive** "++" into "--". The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to compute all possible states of the string after one valid move.

**Example:**

**Input:** s = "++++"

**Output:**

[

"--++",

"+--+",

"++--"

]

**Note:** If there is no valid move, return an empty list [].

class Solution **{**

public**:**

vector**<**string**>** generatePossibleNextMoves**(**string s**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**string**>** generatePossibleNextMoves**(**string s**)** **{**

vector**<**string**>** res**;**

**for** **(**int i **=** s**.**length**()-**1**;** i **>** 0**;** i**--)** **{**

**if** **(**s**[**i**]** **==** '+' **&&** s**[**i**-**1**]** **==** '+'**)** **{**

string t **=** s**;**

t**[**i**]** **=** t**[**i**-**1**]** **=** '-'**;**

res**.**push\_back**(**t**);**

**}**

**}**

**return** res**;**

**}**

**};**

### 294. Flip Game II

You are playing the following Flip Game with your friend: Given a string that contains only these two characters: + and -, you and your friend take turns to flip two **consecutive** "++" into "--". The game ends when a person can no longer make a move and therefore the other person will be the winner.

Write a function to determine if the starting player can guarantee a win.

**Example:**

**Input:** s = "++++"

**Output:** true

**Explanation:** The starting player can guarantee a win by flipping the middle "++" to become "+--+".

**Follow up:**  
Derive your algorithm's runtime complexity.

class Solution **{**

public**:**

bool canWin**(**string s**)** **{**

**for** **(**int i **=** s**.**length**()-**1**;** i **>** 0**;** i**--)** **{**

**if** **(**s**[**i**]** **==** '+' **&&** s**[**i**-**1**]** **==** '+'**)** **{**

string t **=** s**.**substr**(**0**,** i**-**1**)** **+** "--" **+** s**.**substr**(**i**+**1**);**

**if** **(!**canWin**(**t**))** **return** **true;**

**}**

**}**

**return** **false;**

**}**

**};**

### 295. Find Median from Data Stream

Hard

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,

[2,3,4], the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Design a data structure that supports the following two operations:

* void addNum(int num) - Add a integer number from the data stream to the data structure.
* double findMedian() - Return the median of all elements so far.

**Example:**

addNum(1)

addNum(2)

findMedian() -> 1.5

addNum(3)

findMedian() -> 2

**Follow up:**

1. If all integer numbers from the stream are between 0 and 100, how would you optimize it?
2. If 99% of all integer numbers from the stream are between 0 and 100, how would you optimize it?

class MedianFinder **{**

priority\_queue**<**long**>** small**,** large**;**

public**:**

void addNum**(**int num**)** **{**

small**.**push**(**num**);**

large**.**push**(-**small**.**top**());**

small**.**pop**();**

**if** **(**small**.**size**()** **<** large**.**size**())** **{**

small**.**push**(-**large**.**top**());**

large**.**pop**();**

**}**

**}**

double findMedian**()** **{**

**if** **(**small**.**size**()** **>** large**.**size**())** **return** small**.**top**();**

**else** **return** **(**small**.**top**()** **-** large**.**top**())** **/** 2.0**;**

**}**

**};**

### 296. Best Meeting Point

A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. The distance is calculated using [Manhattan Distance](http://en.wikipedia.org/wiki/Taxicab_geometry), where distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|.

**Example:**

**Input:**

1 - 0 - 0 - 0 - 1

| | | | |

0 - 0 - 0 - 0 - 0

| | | | |

0 - 0 - 1 - 0 - 0

**Output: 6**

**Explanation:** Given three people living at (0,0), (0,4), and (2,2):

  The point (0,2) is an ideal meeting point, as the total travel distance

  of 2+2+2=6 is minimal. So return 6.

class Solution **{**

public**:**

int minTotalDistance**(**vector**<**vector**<**int**>>&** grid**)** **{**

vector**<**int**>** rows**,** cols**;**

int n **=** grid**.**size**(),** m **=** grid**[**0**].**size**();**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** **++**j**)** **{**

**if** **(**grid**[**i**][**j**]** **==** 1**)** **{**

rows**.**push\_back**(**i**);**

cols**.**push\_back**(**j**);**

**}**

**}**

**}**

sort**(**cols**.**begin**(),** cols**.**end**());**

//row 已经排序好了

int res **=** 0**,** i **=** 0**,** j **=** rows**.**size**()** **-** 1**;**

**while** **(**i **<** j**)** res **+=** rows**[**j**]** **-** rows**[**i**]** **+** cols**[**j**--]** **-** cols**[**i**++];**

**return** res**;**

**}**

**};**

### 297. Serialize and Deserialize Binary Tree(●ˇ∀ˇ●)

Hard

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

**Example:**

You may serialize the following tree:

1

/ \

2 3

/ \

4 5

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

**Clarification:** The above format is the same as [how LeetCode serializes a binary tree](https://leetcode.com/faq/#binary-tree). You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

**Note:**Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Codec **{**

public**:**

string serialize**(**TreeNode **\***root**)** **{**

ostringstream out**;**

serialize**(**root**,** out**);**

**return** out**.**str**();**

**}**

TreeNode **\***deserialize**(**string data**)** **{**

istringstream in**(**data**);**

**return** deserialize**(**in**);**

**}**

private**:**

void serialize**(**TreeNode **\***root**,** ostringstream **&**out**)** **{**

**if** **(!**root**)** **{**

out **<<** "# "**;**

**return;**

**}**

out **<<** root**->**val **<<** ' '**;**

serialize**(**root**->**left**,** out**);**

serialize**(**root**->**right**,** out**);**

**}**

TreeNode**\*** deserialize**(**istringstream **&**in**)** **{**

string val**;**

in **>>** val**;**

**if** **(**val **==** "#"**)** **return** **nullptr;**

TreeNode**\*** root **=** **new** TreeNode**(**stoi**(**val**));**

root**->**left **=** deserialize**(**in**);**

root**->**right **=** deserialize**(**in**);**

**return** root**;**

**}**

**};**

### 298. Binary Tree Longest Consecutive Sequence

Given a binary tree, find the length of the longest consecutive sequence path.

The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

**Example 1:**

**Input:**

1

\

3

/ \

2 4

\

5

**Output:** 3

**Explanation:** Longest consecutive sequence path is 3-4-5, so return 3.

**Example 2:**

**Input:**

2

\

3

/

2

/

1

**Output: 2**

**Explanation:** Longest consecutive sequence path is 2-3, not 3-2-1, so return 2.

class Solution **{**

public**:**

int longestConsecutive**(**TreeNode**\*** root**)** **{**

**}**

**};**

class Solution **{**

public**:**

int longestConsecutive**(**TreeNode**\*** root**)** **{**

**if** **(!**root**)** **return** 0**;**

dfs**(**root**,** root**->**val**,** 0**);**

**return** res**;**

**}**

private**:**

int res **=** 0**;**

void dfs**(**TreeNode **\***root**,** int pre**,** int len**)** **{**

**if** **(!**root**)** **return;**

res **=** max**(**res**,** len **=** **(**root**->**val **==** pre **+** 1**)** **?** len**+**1 **:** 1**);**

dfs**(**root**->**left**,** root**->**val**,** len**);**

dfs**(**root**->**right**,** root**->**val**,** len**);**

**}**

**};**

### 299. Bulls and Cows

Medium

You are playing the following [Bulls and Cows](https://en.wikipedia.org/wiki/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.

class Solution **{**

public**:**

string getHint**(**string secret**,** string guess**)** **{**

**}**

**};**

class Solution **{**

public**:**

string getHint**(**string secret**,** string guess**)** **{**

int m**[**256**]** **=** **{**0**};**

int bulls **=** 0**,** sum **=** 0**,** sz **=** guess**.**size**();**

**for** **(**auto **&**c **:** secret**)** m**[**c**]++;**

**for** **(**int i **=** 0**;** i **<** sz**;** **++**i**)** **{**

**if** **(**secret**[**i**]** **==** guess**[**i**])** bulls**++;**

**if** **(**m**[**guess**[**i**]])** **{**

sum**++;**

m**[**guess**[**i**]]--;**

**}**

**}**

**return** to\_string**(**bulls**)** **+** "A" **+** to\_string**(**sum**-**bulls**)** **+** "B"**;**

**}**

**};**

### 300. Longest Increasing Subsequence(●ˇ∀ˇ●)

Medium

Given an unsorted array of integers, find the length of longest increasing subsequence.

**Example:**

**Input:** [10,9,2,5,3,7,101,18]

**Output:** 4

**Explanation:** The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

**Note:**

* There may be more than one LIS combination, it is only necessary for you to return the length.
* Your algorithm should run in O(*n2*) complexity.

**Follow up:** Could you improve it to O(*n* log *n*) time complexity?

class Solution **{**

public**:**

int lengthOfLIS**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

///////////////////////////////////////////////// O(*n^2*)/////////////////////////////////////////////////

class Solution **{**

public**:**

int lengthOfLIS**(**vector**<**int**>&** nums**)** **{**

**if** **(**nums**.**empty**())** **return** 0**;**

int n **=** nums**.**size**(),** res **=** 1**;**

vector**<**int**>** f**(**n**,** 1**);**

**for** **(**int i **=** 1**;** i **<** n**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** i**;** j**++)** **{**

**if** **(**nums**[**i**]** **>** nums**[**j**])** **{**

f**[**i**]** **=** max**(**f**[**j**]+**1**,** f**[**i**]);**

res **=** max**(**res**,** f**[**i**]);**

**}**

**}**

**}**

**return** res**;**

**}**

**};**

////////////////////////////////////////////////////////O(*n* log *n*)///////////////////////////////////////////////////

class Solution **{**

public**:**

int lengthOfLIS**(**vector**<**int**>&** nums**)** **{**

vector**<**int**>** res**;**

**for** **(**auto **&**i **:** nums**)** **{**

auto it **=** lower\_bound**(**res**.**begin**(),** res**.**end**(),** i**);**

**if** **(**it **==** res**.**end**())** res**.**push\_back**(**i**);**

**else** **\***it **=** i**;**

**}**

**return** res**.**size**();**

**}**

**};**