目录

[500. Keyboard Row 2](#_Toc25841327)

[501. Find Mode in Binary Search Tree 3](#_Toc25841328)

[502. IPO 4](#_Toc25841329)

[503. Next Greater Element II 5](#_Toc25841330)

[504. Base 7 6](#_Toc25841331)

[506. Relative Ranks 7](#_Toc25841332)

[507. Perfect Number 9](#_Toc25841333)

[508. Most Frequent Subtree Sum 10](#_Toc25841334)

[509. Fibonacci Number 12](#_Toc25841335)

[513. Find Bottom Left Tree Value 14](#_Toc25841336)

[514. Freedom Trail 16](#_Toc25841337)

[515. Find Largest Value in Each Tree Row 18](#_Toc25841338)

[516. Longest Palindromic Subsequence 20](#_Toc25841339)

[517. Super Washing Machines 23](#_Toc25841340)

[518. Coin Change 2 25](#_Toc25841341)

[519. Random Flip Matrix 27](#_Toc25841342)

[520. Detect Capital 29](#_Toc25841343)

[521. Longest Uncommon Subsequence I 31](#_Toc25841344)

[522. Longest Uncommon Subsequence II 32](#_Toc25841345)

[523. Continuous Subarray Sum 34](#_Toc25841346)

[524. Longest Word in Dictionary through Deleting 36](#_Toc25841347)

[525. Contiguous Array 38](#_Toc25841348)

[526. Beautiful Arrangement 39](#_Toc25841349)

[528. Random Pick with Weight 41](#_Toc25841350)

[529. Minesweeper 42](#_Toc25841351)

[530. Minimum Absolute Difference in BST 45](#_Toc25841352)

[532. K-diff Pairs in an Array 47](#_Toc25841353)

[535. Encode and Decode TinyURL 49](#_Toc25841354)

[537. Complex Number Multiplication 50](#_Toc25841355)

[538. Convert BST to Greater Tree 51](#_Toc25841356)

[539. Minimum Time Difference 53](#_Toc25841357)

[540. Single Element in a Sorted Array 55](#_Toc25841358)

[541. Reverse String II 57](#_Toc25841359)

[542. 01 Matrix 58](#_Toc25841360)

[543. Diameter of Binary Tree 60](#_Toc25841361)

[546. Remove Boxes 62](#_Toc25841362)

[547. Friend Circles 63](#_Toc25841363)

### 500. Keyboard Row

Easy

Given a List of words, return the words that can be typed using letters of **alphabet** on only one row's of American keyboard like the image below.



**Example:**

**Input:** ["Hello", "Alaska", "Dad", "Peace"]

**Output:** ["Alaska", "Dad"]

**Note:**

1. You may use one character in the keyboard more than once.
2. You may assume the input string will only contain letters of alphabet.

### 501. Find Mode in Binary Search Tree

Easy

Given a binary search tree (BST) with duplicates, find all the [mode(s)](https://en.wikipedia.org/wiki/Mode_(statistics)) (the most frequently occurred element) in the given BST.

Assume a BST is defined as follows:

* The left subtree of a node contains only nodes with keys **less than or equal to** the node's key.
* The right subtree of a node contains only nodes with keys **greater than or equal to** the node's key.
* Both the left and right subtrees must also be binary search trees.

For example:  
Given BST [1,null,2,2],

1

\

2

/

2

return [2].

**Note:** If a tree has more than one mode, you can return them in any order.

**Follow up:** Could you do that without using any extra space? (Assume that the implicit stack space incurred due to recursion does not count).

### 502. IPO

Hard

Suppose LeetCode will start its IPO soon. In order to sell a good price of its shares to Venture Capital, LeetCode would like to work on some projects to increase its capital before the IPO. Since it has limited resources, it can only finish at most **k** distinct projects before the IPO. Help LeetCode design the best way to maximize its total capital after finishing at most **k** distinct projects.

You are given several projects. For each project **i**, it has a pure profit **Pi** and a minimum capital of **Ci** is needed to start the corresponding project. Initially, you have **W** capital. When you finish a project, you will obtain its pure profit and the profit will be added to your total capital.

To sum up, pick a list of at most **k** distinct projects from given projects to maximize your final capital, and output your final maximized capital.

**Example 1:**

**Input:** k=2, W=0, Profits=[1,2,3], Capital=[0,1,1].

**Output:** 4

**Explanation:** Since your initial capital is 0, you can only start the project indexed 0.

After finishing it you will obtain profit 1 and your capital becomes 1.

With capital 1, you can either start the project indexed 1 or the project indexed 2.

Since you can choose at most 2 projects, you need to finish the project indexed 2 to get the maximum capital.

Therefore, output the final maximized capital, which is 0 + 1 + 3 = 4.

**Note:**

1. You may assume all numbers in the input are non-negative integers.
2. The length of Profits array and Capital array will not exceed 50,000.
3. The answer is guaranteed to fit in a 32-bit signed integer.

### 503. Next Greater Element II

Medium

Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element. The Next Greater Number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, output -1 for this number.

**Example 1:**

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

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

**Explanation:** The first 1's next greater number is 2;   
The number 2 can't find next greater number;   
The second 1's next greater number needs to search circularly, which is also 2.

**Note:** The length of given array won't exceed 10000.

class Solution **{**

public**:**

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

stack**<**int**>** stk**;**

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

vector**<**int**>** res**(**n**,** **-**1**);**

**for(**int k **=** 0**;** k **<** 2**;** k**++)** **{**

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

**while** **(!**stk**.**empty**()** **&&** nums**[**stk**.**top**()]** **<** nums**[**i**])** **{**

res**[**stk**.**top**()]** **=** nums**[**i**];**

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

**}**

stk**.**push**(**i**);**

**}**

**}**

**return** res**;**

**}**

**};**

### 504. Base 7

Easy

Given an integer, return its base 7 string representation.

**Example 1:**

**Input:** 100

**Output:** "202"

**Example 2:**

**Input:** -7

**Output:** "-10"

**Note:** The input will be in range of [-1e7, 1e7].

class Solution **{**

public**:**

string convertToBase7**(**int num**)** **{**

**if** **(**num **==** 0**)** **return** "0"**;**

string res**;**

bool isNeg **=** num **<** 0**;**

**while** **(**num **!=** 0**)** **{**

res **+=** char**(**'0'**+**abs**(**num **%** 7**));**

num **/=** 7**;**

**}**

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

**return** isNeg **?** '-' **+** res **:** res**;**

**}**

**};**

### 506. Relative Ranks

Easy

Given scores of **N** athletes, find their relative ranks and the people with the top three highest scores, who will be awarded medals: "Gold Medal", "Silver Medal" and "Bronze Medal".

**Example 1:**

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

**Output:** ["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"]

**Explanation:** The first three athletes got the top three highest scores, so they got "Gold Medal", "Silver Medal" and "Bronze Medal".   
For the left two athletes, you just need to output their relative ranks according to their scores.

**Note:**

1. N is a positive integer and won't exceed 10,000.
2. All the scores of athletes are guaranteed to be unique.

class Solution **{**

public**:**

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

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

map**<**int**,** int**>** mp**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** mp**[**nums**[**i**]]** **=** i**;**

vector**<**string**>** ans**(**n**);**

int cnt **=** 1**;**

**for** **(**map**<**int**,** int**>::**reverse\_iterator it **=** mp**.**rbegin**();** it **!=** mp**.**rend**();** it**++,** cnt**++)** **{**

**if** **(**cnt **==** 1**)** ans**[**it**->**second**]** **=** "Gold Medal"**;**

**else** **if** **(**cnt **==** 2**)** ans**[**it**->**second**]** **=** "Silver Medal"**;**

**else** **if** **(**cnt **==** 3**)** ans**[**it**->**second**]** **=** "Bronze Medal"**;**

**else** ans**[**it**->**second**]** **=** to\_string**(**cnt**);**

**}**

**return** ans**;**

**}**

**};**

### 507. Perfect Number

Easy

We define the Perfect Number is a **positive** integer that is equal to the sum of all its **positive** divisors except itself.

Now, given an **integer** n, write a function that returns true when it is a perfect number and false when it is not.

**Example:**

**Input:** 28

**Output:** True

**Explanation:** 28 = 1 + 2 + 4 + 7 + 14

**Note:** The input number **n** will not exceed 100,000,000. (1e8)

class Solution **{**

public**:**

bool checkPerfectNumber**(**int num**)** **{**

**if** **(**num **<=** 1**)** **return** **false;**

int sum **=** 0**,** Sqrt **=** sqrt**(**num**);**

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

**if** **(**num **%** i **==** 0**)** **{**

**if** **(**i **==** 1 **||** num**/**i **==** i**)** sum **+=** i**;**

**else** sum **+=** i **+** num**/**i**;**

**if** **(**sum **>** num**)** **break;**

**}**

**}**

**return** sum **==** num**;**

**}**

**};**

### 508. Most Frequent Subtree Sum

Medium

Given the root of a tree, you are asked to find the most frequent subtree sum. The subtree sum of a node is defined as the sum of all the node values formed by the subtree rooted at that node (including the node itself). So what is the most frequent subtree sum value? If there is a tie, return all the values with the highest frequency in any order.

**Examples 1**  
Input:

5

/ \

2 -3

return [2, -3, 4], since all the values happen only once, return all of them in any order.

**Examples 2**  
Input:

5

/ \

2 -5

return [2], since 2 happens twice, however -5 only occur once.

**Note:** You may assume the sum of values in any subtree is in the range of 32-bit signed integer.

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution **{**

public**:**

vector**<**int**>** findFrequentTreeSum**(**TreeNode**\*** root**)** **{**

dfs**(**root**);**

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

**for** **(**auto **&**i **:** m**)** **if** **(**i**.**second **==** MaxCnt**)** **{**

res**.**push\_back**(**i**.**first**);**

**}**

**return** res**;**

**}**

private**:**

unordered\_map**<**int**,** int**>** m**;**

int MaxCnt **=** 0**;**

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

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

int l **=** dfs**(**root**->**left**),** r **=** dfs**(**root**->**right**);**

int ret **=** l **+** r **+** root**->**val**;**

MaxCnt **=** max**(**MaxCnt**,** **++**m**[**ret**]);**

**return** ret**;**

**}**

**};**

### 509. Fibonacci Number

Easy

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

F(0) = 0,   F(1) = 1

F(N) = F(N - 1) + F(N - 2), for N > 1.

Given N, calculate F(N).

**Example 1:**

**Input:** 2

**Output:** 1

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

**Example 2:**

**Input:** 3

**Output:** 2

**Explanation:** F(3) = F(2) + F(1) = 1 + 1 = 2.

**Example 3:**

**Input:** 4

**Output:** 3

**Explanation:** F(4) = F(3) + F(2) = 2 + 1 = 3.

**Note:**

0 ≤ N ≤ 30.

class Solution **{**

public**:**

struct Matrix**{**

int a**[**2**][**2**]** **=** **{**0**};**

void init**(){**

**for** **(**int i **=** 0**;** i **<** 2**;** i**++)**

a**[**i**][**i**]** **=** 1**;**

**}**

**};**

Matrix MUL**(**Matrix A**,** Matrix B**){**

Matrix C**;**

**for** **(**int i **=** 0**;** i **<** 2**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** 2**;** j**++)** **{**

**for** **(**int k **=** 0**;** k **<** 2**;** k**++){**

C**.**a**[**i**][**j**]** **+=** A**.**a**[**i**][**k**]** **\*** B**.**a**[**k**][**j**];**

**}**

**}**

**}**

**return** C**;**

**}**

int fib**(**int n**)** **{**

Matrix res**,** x**;**

x**.**a**[**0**][**0**]** **=** x**.**a**[**0**][**1**]** **=** x**.**a**[**1**][**0**]** **=** 1**;**

x**.**a**[**1**][**1**]** **=** 0**;**

res**.**init**();**

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

**if** **(**n**&**1**)** res **=** MUL**(**res**,** x**);**

x **=** MUL**(**x**,** x**);**

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

**}**

**return** res**.**a**[**1**][**0**];**

**}**

**};**

### 513. Find Bottom Left Tree Value

Medium

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

**Example 1:**

Input:

2

/ \

1 3

Output:

1

**Example 2:**

Input:

1

/ \

2 3

/ / \

4 5 6

/

7

Output:

7

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution **{**

public**:**

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

queue**<**TreeNode**\*>** que**;**

que**.**push**(**root**);**

TreeNode **\***t**;**

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

t **=** que**.**front**();**

que**.**pop**();**

**if** **(**t**->**right**)** que**.**push**(**t**->**right**);**

**if** **(**t**->**left**)** que**.**push**(**t**->**left**);**

**}**

**return** t**->**val**;**

**}**

**};**

### 514. Freedom Trail

Hard

In the video game Fallout 4, the quest "Road to Freedom" requires players to reach a metal dial called the "Freedom Trail Ring", and use the dial to spell a specific keyword in order to open the door.

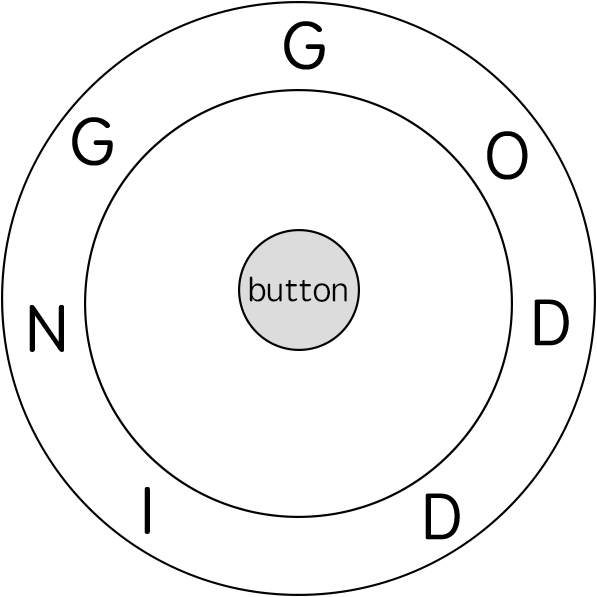
Given a string **ring**, which represents the code engraved on the outer ring and another string **key**, which represents the keyword needs to be spelled. You need to find the **minimum** number of steps in order to spell all the characters in the keyword.

Initially, the first character of the **ring** is aligned at 12:00 direction. You need to spell all the characters in the string **key** one by one by rotating the ring clockwise or anticlockwise to make each character of the string **key** aligned at 12:00 direction and then by pressing the center button.

At the stage of rotating the ring to spell the key character **key[i]**:

1. You can rotate the **ring** clockwise or anticlockwise **one place**, which counts as 1 step. The final purpose of the rotation is to align one of the string **ring's** characters at the 12:00 direction, where this character must equal to the character **key[i]**.
2. If the character **key[i]** has been aligned at the 12:00 direction, you need to press the center button to spell, which also counts as 1 step. After the pressing, you could begin to spell the next character in the key (next stage), otherwise, you've finished all the spelling.

**Example:**



**Input:** ring = "godding", key = "gd"

**Output:** 4

**Explanation:**

For the first key character 'g', since it is already in place, we just need 1 step to spell this character.

For the second key character 'd', we need to rotate the ring "godding" anticlockwise by two steps to make it become "ddinggo".

Also, we need 1 more step for spelling.

So the final output is 4.

**Note:**

1. Length of both ring and **key** will be in range 1 to 100.
2. There are only lowercase letters in both strings and might be some duplcate characters in both strings.
3. It's guaranteed that string **key** could always be spelled by rotating the string **ring**.

### 515. Find Largest Value in Each Tree Row

Medium

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

**Example:**

**Input:**

1

/ \

3 2

/ \ \

5 3 9

**Output:** [1, 3, 9]

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution **{**

public**:**

vector**<**int**>** largestValues**(**TreeNode**\*** root**)** **{**

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

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

queue**<**TreeNode**\*>** que**;**

que**.**push**(**root**);**

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

int Max **=** que**.**front**()->**val**,** sz **=** que**.**size**();**

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

auto t **=** que**.**front**();**

**if** **(**t**->**left**)** que**.**push**(**t**->**left**);**

**if** **(**t**->**right**)** que**.**push**(**t**->**right**);**

que**.**pop**();**

Max **=** max**(**Max**,** t**->**val**);**

**}**

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

**}**

**return** res**;**

**}**

**};**

### 516. Longest Palindromic Subsequence

Medium

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

**Example 1:**  
Input:

"bbbab"

Output:

4

One possible longest palindromic subsequence is "bbbb".

**Example 2:**  
Input:

"cbbd"

Output:

2

One possible longest palindromic subsequence is "bb".

class Solution **{**

public**:**

int longestPalindromeSubseq**(**string s**)** **{**

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

vector**<**vector**<**int**>>** dp**(**n**,** vector**<**int**>(**n**));**

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

dp**[**i**][**i**]** **=** 1**;**

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

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

dp**[**i**][**j**]** **=** dp**[**i **+** 1**][**j **-** 1**]** **+** 2**;**

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

dp**[**i**][**j**]** **=** max**(**dp**[**i **+** 1**][**j**],** dp**[**i**][**j **-** 1**]);**

**}**

**}**

**}**

**return** dp**[**0**][**n **-** 1**];**

**}**

**};**

class Solution **{**

public**:**

int longestPalindromeSubseq**(**string s**)** **{**

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

vector**<**vector**<**int**>>** g**(**n**,** vector**<**int**>(**n**,** 1**));**

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

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

g**[**i**][**j**]** **=** s**[**i**]** **!=** s**[**j**]** **?** max**(**g**[**i**+**1**][**j**],** g**[**i**][**j**-**1**])**

**:** **(**2 **+** **(**j**-**i **>** 1 **?** g**[**i**+**1**][**j**-**1**]** **:** 0**));**

**}**

**}**

**return** g**[**0**][**n**-**1**];**

**}**

**};**

### 517. Super Washing Machines

Hard

You have **n** super washing machines on a line. Initially, each washing machine has some dresses or is empty.

For each **move**, you could choose **any m** (1 ≤ m ≤ n) washing machines, and pass **one dress** of each washing machine to one of its adjacent washing machines **at the same time** .

Given an integer array representing the number of dresses in each washing machine from left to right on the line, you should find the **minimum number of moves** to make all the washing machines have the same number of dresses. If it is not possible to do it, return -1.

**Example1**

**Input:** [1,0,5]

**Output:** 3

**Explanation:**

1st move: 1 0 <-- 5 => 1 1 4

2nd move: 1 <-- 1 <-- 4 => 2 1 3

3rd move: 2 1 <-- 3 => 2 2 2

**Example2**

**Input:** [0,3,0]

**Output:** 2

**Explanation:**

1st move: 0 <-- 3 0 => 1 2 0

2nd move: 1 2 --> 0 => 1 1 1

**Example3**

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

**Output:** -1

**Explanation:**

It's impossible to make all the three washing machines have the same number of dresses.

**Note:**

1. The range of n is [1, 10000].
2. The range of dresses number in a super washing machine is [0, 1e5].

### 518. Coin Change 2

Medium

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

**Example 1:**

**Input:** amount = 5, coins = [1, 2, 5]

**Output:** 4

**Explanation:** there are four ways to make up the amount:

5=5

5=2+2+1

5=2+1+1+1

5=1+1+1+1+1

**Example 2:**

**Input:** amount = 3, coins = [2]

**Output:** 0

**Explanation:** the amount of 3 cannot be made up just with coins of 2.

**Example 3:**

**Input:** amount = 10, coins = [10]

**Output:** 1

**Note:**

You can assume that

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

class Solution **{**

public**:**

int change**(**int amount**,** vector**<**int**>&** coins**)** **{**

vector**<**int**>** dp**(**amount**+**1**,** 0**);**

dp**[**0**]** **=** 1**;**

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

**for** **(**int s **=** 0**;** s **<=** amount**;** **++**s**)** **{**

**if** **(**s**-**i **>=** 0**)** dp**[**s**]** **+=** dp**[**s**-**i**];**

**}**

**}**

**return** dp**[**amount**];**

**}**

**};**

### 519. Random Flip Matrix

Medium

You are given the number of rows n\_rows and number of columns n\_cols of a 2D binary matrix where all values are initially 0. Write a function flip which chooses a 0 value [uniformly at random](https://en.wikipedia.org/wiki/Discrete_uniform_distribution), changes it to 1, and then returns the position [row.id, col.id] of that value. Also, write a function reset which sets all values back to 0. **Try to minimize the number of calls to system's Math.random()** and optimize the time and space complexity.

Note:

1. 1 <= n\_rows, n\_cols <= 10000
2. 0 <= row.id < n\_rows and 0 <= col.id < n\_cols
3. flip will not be called when the matrix has no 0 values left.
4. the total number of calls to flip and reset will not exceed 1000.

**Example 1:**

**Input:**

["Solution","flip","flip","flip","flip"]

[[2,3],[],[],[],[]]

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

**Example 2:**

**Input:**

["Solution","flip","flip","reset","flip"]

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

**Output:** [null,[0,0],[0,1],null,[0,0]]

**Explanation of Input Syntax:**

The input is two lists: the subroutines called and their arguments. Solution's constructor has two arguments, n\_rows and n\_cols. flip and reset have no arguments. Arguments are always wrapped with a list, even if there aren't any.

class Solution **{**

public**:**

Solution**(**int n\_rows**,** int n\_cols**)** **:** n**(**n\_rows**),** m**(**n\_cols**)** **{**

reset**();**

**}**

vector**<**int**>** flip**()** **{**

int used\_id **=** rand**()** **%** **(**sz**--);**

int unused\_id **=** mp**.**count**(**used\_id**)** **?** mp**[**used\_id**]** **:** used\_id**;**

mp**[**used\_id**]** **=** mp**.**count**(**sz**)** **?** mp**[**sz**]** **:** sz**;**

**return** **{**unused\_id **/** m**,** unused\_id **%** m**};**

**}**

void reset**()** **{**

mp**.**clear**();**

sz **=** n **\*** m**;**

**}**

private**:**

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

unordered\_map**<**int**,** int**>** mp**;** //<used, not used>

**};**

/\*\*

\* Your Solution object will be instantiated and called as such:

\* Solution\* obj = new Solution(n\_rows, n\_cols);

\* vector<int> param\_1 = obj->flip();

\* obj->reset();

\*/

### 520. Detect Capital

Easy

Given a word, you need to judge whether the usage of capitals in it is right or not.

We define the usage of capitals in a word to be right when one of the following cases holds:

1. All letters in this word are capitals, like "USA".
2. All letters in this word are not capitals, like "leetcode".
3. Only the first letter in this word is capital, like "Google".

Otherwise, we define that this word doesn't use capitals in a right way.

**Example 1:**

**Input:** "USA"

**Output:** True

**Example 2:**

**Input:** "FlaG"

**Output:** False

**Note:** The input will be a non-empty word consisting of uppercase and lowercase latin letters.

class Solution **{**

public**:**

bool detectCapitalUse**(**string a**)** **{**

**for(**int i **=** 1**;** i **<** a**.**size**();** i**++)** **{**

**if** **(**isupper**(**a**[**1**])** **!=** isupper**(**a**[**i**])** **||** islower**(**a**[**0**])** **&&** isupper**(**a**[**i**]))**

**return** **false;**

**}**

**return** **true;**

**}**

**};**

### 521. Longest Uncommon Subsequence I

Easy

Given a group of two strings, you need to find the longest uncommon subsequence of this group of two strings. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be **any** subsequence of the other strings.

A **subsequence** is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be two strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

**Example 1:**

**Input:** "aba", "cdc"

**Output:** 3

**Explanation:** The longest uncommon subsequence is "aba" (or "cdc"),   
because "aba" is a subsequence of "aba",   
but not a subsequence of any other strings in the group of two strings.

**Note:**

1. Both strings' lengths will not exceed 100.
2. Only letters from a ~ z will appear in input strings.

class Solution **{**

public**:**

int findLUSlength**(**string a**,** string b**)** **{**

**if** **(**a **==** b**)** **return** **-**1**;**

**else** **return** max**(**a**.**size**(),** b**.**size**());**

**}**

**};**

### 522. Longest Uncommon Subsequence II

Medium

Given a list of strings, you need to find the longest uncommon subsequence among them. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be **any** subsequence of the other strings.

A **subsequence** is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be a list of strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

**Example 1:**

**Input:** "aba", "cdc", "eae"

**Output:** 3

**Note:**

1. All the given strings' lengths will not exceed 10.
2. The length of the given list will be in the range of [2, 50].

class Solution **{**

public**:**

int findLUSlength**(**vector**<**string**>&** strs**)** **{**

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

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

sort**(**strs**.**begin**(),** strs**.**end**(),** **[](**string a**,** string b**){**

**if** **(**a**.**size**()** **==** b**.**size**())** **return** a **>** b**;**

**return** a**.**size**()** **>** b**.**size**();**

**});**

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

**if** **(**i **==** n**-**1 **||** strs**[**i**]** **!=** strs**[**i**+**1**])** **{**

bool found **=** **true;**

**for** **(**auto **&**a **:** s**)** **{**

int j **=** 0**;**

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

**if** **(**c **==** strs**[**i**][**j**])** **++**j**;**

**if** **(**j **==** strs**[**i**].**size**())** **break;**

**}**

**if** **(**j **==** strs**[**i**].**size**())** **{**

found **=** **false;**

**break;**

**}**

**}**

**if** **(**found**)** **return** strs**[**i**].**size**();**

**}**

s**.**insert**(**strs**[**i**]);**

**}**

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

**}**

**};**

### 523. Continuous Subarray Sum

Medium

Given a list of **non-negative** numbers and a target **integer** k, write a function to check if the array has a continuous subarray of size at least 2 that sums up to a multiple of **k**, that is, sums up to n\*k where n is also an **integer**.

**Example 1:**

**Input:** [23, 2, 4, 6, 7], k=6

**Output:** True

**Explanation:** Because [2, 4] is a continuous subarray of size 2 and sums up to 6.

**Example 2:**

**Input:** [23, 2, 6, 4, 7], k=6

**Output:** True

**Explanation:** Because [23, 2, 6, 4, 7] is an continuous subarray of size 5 and sums up to 42.

**Note:**

1. The length of the array won't exceed 10,000.
2. You may assume the sum of all the numbers is in the range of a signed 32-bit integer.

class Solution **{**

public**:**

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

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

unordered\_map**<**int**,** int**>** modk**;**

modk**[**0**]** **=** **-**1**;**

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

sum **+=** nums**[**i**];**

// 特例 k == 0

int mod **=** k **==** 0 **?** sum **:** sum **%** k**;**

**if** **(**modk**.**count**(**mod**))** **{**

**if** **(**modk**[**mod**]** **!=** i**-**1**)** **return** **true;**

**}**

**else** modk**[**mod**]** **=** i**;**

**}**

**return** **false;**

**}**

**};**

### 524. Longest Word in Dictionary through Deleting

Medium

Given a string and a string dictionary, find the longest string in the dictionary that can be formed by deleting some characters of the given string. If there are more than one possible results, return the longest word with the smallest lexicographical order. If there is no possible result, return the empty string.

**Example 1:**

**Input:**

s = "abpcplea", d = ["ale","apple","monkey","plea"]

**Output:**

"apple"

**Example 2:**

**Input:**

s = "abpcplea", d = ["a","b","c"]

**Output:**

"a"

**Note:**

1. All the strings in the input will only contain lower-case letters.
2. The size of the dictionary won't exceed 1,000.
3. The length of all the strings in the input won't exceed 1,000.

class Solution **{**

public**:**

string findLongestWord**(**string s**,** vector**<**string**>&** d**)** **{**

string res **=** ""**;**

**for** **(**auto **&**a **:** d**)** **{**

**if** **(**a**.**length**()** **<** res**.**length**())** **continue;**

**if** **(**check**(**a**,** s**)** **&&** **(**res**.**length**()** **<** a**.**length**()** **||** res **>** a**))** **{**

res **=** a**;**

**}**

**}**

**return** res**;**

**}**

private**:**

bool check**(**string **&**a**,** string **&**s**)** **{**

auto i **=** a**.**begin**(),** j **=** s**.**begin**();**

**while** **(**i **!=** a**.**end**()** **&&** j **!=** s**.**end**())** **{**

**if** **(\***i **==** **\***j**)** i**++;**

j**++;**

**}**

**return** i **==** a**.**end**();**

**}**

**};**

### 525. Contiguous Array

Medium

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

**Example 1:**

**Input:** [0,1]

**Output:** 2

**Explanation:** [0, 1] is the longest contiguous subarray with equal number of 0 and 1.

**Example 2:**

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

**Output:** 2

**Explanation:** [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

**Note:** The length of the given binary array will not exceed 50,000.

class Solution **{**

public**:**

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

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

unordered\_map**<**int**,** int**>** mp**{{**0**,** **-**1**}};**

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

nums**[**i**]** **==** 0 **?** **++**sum **:** **--**sum**;**

**if** **(**mp**.**count**(-**sum**))** res **=** max**(**res**,** i**-**mp**[-**sum**]);**

**else** mp**[-**sum**]** **=** i**;**

**}**

**return** res**;**

**}**

**};**

### 526. Beautiful Arrangement

Medium

Suppose you have **N** integers from 1 to N. We define a beautiful arrangement as an array that is constructed by these **N** numbers successfully if one of the following is true for the ith position (1 <= i <= N) in this array:

1. The number at the ith position is divisible by **i**.
2. **i** is divisible by the number at the ith position.

Now given N, how many beautiful arrangements can you construct?

**Example 1:**

**Input:** 2

**Output:** 2

**Explanation:**

The first beautiful arrangement is [1, 2]:

Number at the 1st position (i=1) is 1, and 1 is divisible by i (i=1).

Number at the 2nd position (i=2) is 2, and 2 is divisible by i (i=2).

The second beautiful arrangement is [2, 1]:

Number at the 1st position (i=1) is 2, and 2 is divisible by i (i=1).

Number at the 2nd position (i=2) is 1, and i (i=2) is divisible by 1.

**Note:**

1. **N** is a positive integer and will not exceed 15.

### 528. Random Pick with Weight

Medium

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

Note:

1. 1 <= w.length <= 10000
2. 1 <= w[i] <= 10^5
3. pickIndex will be called at most 10000 times.

**Example 1:**

**Input:**

["Solution","pickIndex"]

[[[1]],[]]

**Output:** [null,0]

**Example 2:**

**Input:**

["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"]

[[[1,3]],[],[],[],[],[]]

**Output:** [null,0,1,1,1,0]

**Explanation of Input Syntax:**

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

### 529. Minesweeper

Medium

Let's play the minesweeper game ([Wikipedia](https://en.wikipedia.org/wiki/Minesweeper_(video_game)), [online game](http://minesweeperonline.com/))!

You are given a 2D char matrix representing the game board. **'M'** represents an **unrevealed** mine, **'E'** represents an **unrevealed** empty square, **'B'** represents a **revealed** blank square that has no adjacent (above, below, left, right, and all 4 diagonals) mines, **digit** ('1' to '8') represents how many mines are adjacent to this **revealed** square, and finally **'X'** represents a **revealed** mine.

Now given the next click position (row and column indices) among all the **unrevealed** squares ('M' or 'E'), return the board after revealing this position according to the following rules:

1. If a mine ('M') is revealed, then the game is over - change it to **'X'**.
2. If an empty square ('E') with **no adjacent mines** is revealed, then change it to revealed blank ('B') and all of its adjacent **unrevealed** squares should be revealed recursively.
3. If an empty square ('E') with **at least one adjacent mine** is revealed, then change it to a digit ('1' to '8') representing the number of adjacent mines.
4. Return the board when no more squares will be revealed.

**Example 1:**

**Input:**

[['E', 'E', 'E', 'E', 'E'],

['E', 'E', 'M', 'E', 'E'],

['E', 'E', 'E', 'E', 'E'],

['E', 'E', 'E', 'E', 'E']]

Click : [3,0]

**Output:**

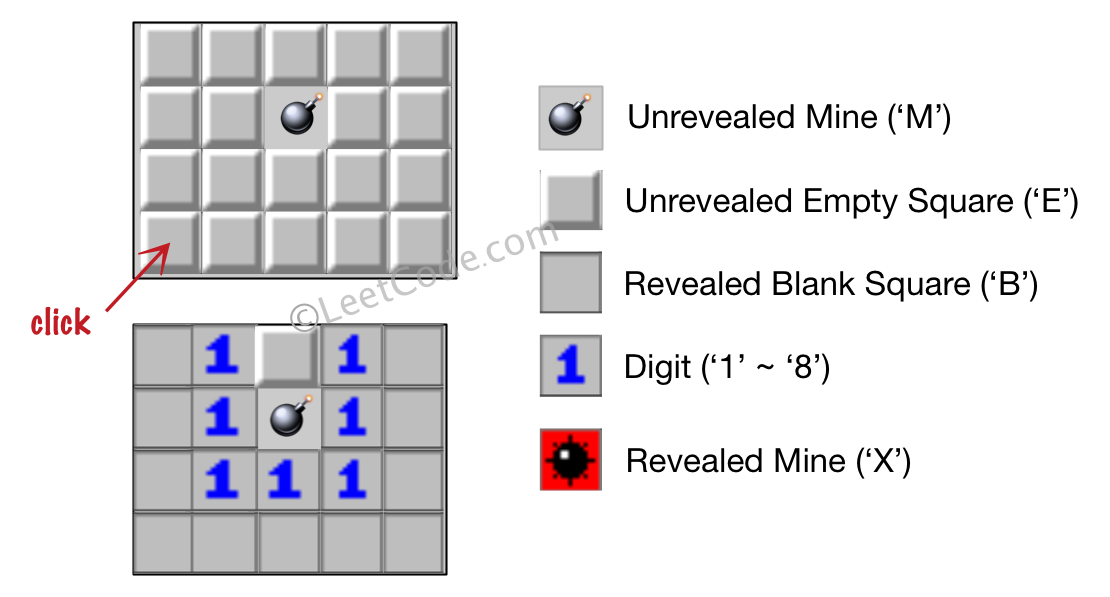
[['B', '1', 'E', '1', 'B'],

['B', '1', 'M', '1', 'B'],

['B', '1', '1', '1', 'B'],

['B', 'B', 'B', 'B', 'B']]

**Explanation:**



**Example 2:**

**Input:**

[['B', '1', 'E', '1', 'B'],

['B', '1', 'M', '1', 'B'],

['B', '1', '1', '1', 'B'],

['B', 'B', 'B', 'B', 'B']]

Click : [1,2]

**Output:**

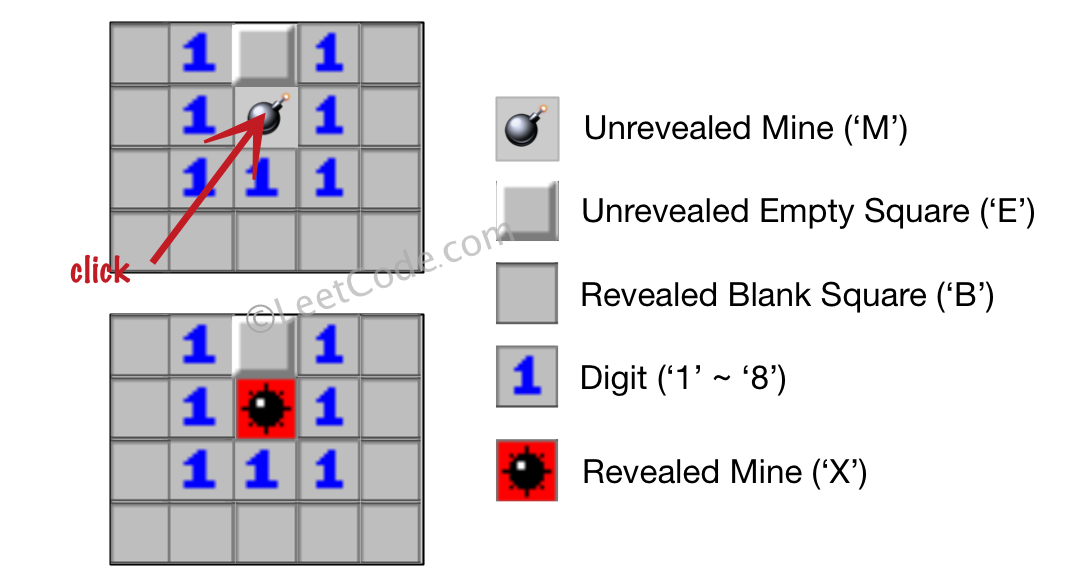
[['B', '1', 'E', '1', 'B'],

['B', '1', 'X', '1', 'B'],

['B', '1', '1', '1', 'B'],

['B', 'B', 'B', 'B', 'B']]

**Explanation:**



**Note:**

1. The range of the input matrix's height and width is [1,50].
2. The click position will only be an unrevealed square ('M' or 'E'), which also means the input board contains at least one clickable square.
3. The input board won't be a stage when game is over (some mines have been revealed).
4. For simplicity, not mentioned rules should be ignored in this problem. For example, you **don't** need to reveal all the unrevealed mines when the game is over, consider any cases that you will win the game or flag any squares.

### 530. Minimum Absolute Difference in BST

Easy

Given a binary search tree with non-negative values, find the minimum [absolute difference](https://en.wikipedia.org/wiki/Absolute_difference) between values of any two nodes.

**Example:**

**Input:**

1

\

3

/

2

**Output:**

1

**Explanation:**

The minimum absolute difference is 1, which is the difference between 2 and 1 (or between 2 and 3).

**Note:** There are at least two nodes in this BST.

/\*\*

\* 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 getMinimumDifference**(**TreeNode**\*** p**)** **{**

int pre**,** res **=** INT\_MAX**;**

bool isFirst **=** **true;**

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

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

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

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

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

**}**

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

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

**if** **(**isFirst**)** **{**

pre **=** p**->**val**;**

isFirst **=** **false;**

**}**

**else** **{**

res **=** min**(**res**,** p**->**val **-** pre**);**

pre **=** p**->**val**;**

**}**

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

**}**

**return** res**;**

**}**

**};**

### 532. K-diff Pairs in an Array

Easy

Given an array of integers and an integer **k**, you need to find the number of **unique** k-diff pairs in the array. Here a **k-diff** pair is defined as an integer pair (i, j), where **i** and **j** are both numbers in the array and their [absolute difference](https://en.wikipedia.org/wiki/Absolute_difference) is **k**.

**Example 1:**

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

**Output:** 2

**Explanation:** There are two 2-diff pairs in the array, (1, 3) and (3, 5).  
Although we have two 1s in the input, we should only return the number of **unique** pairs.

**Example 2:**

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

**Output:** 4

**Explanation:** There are four 1-diff pairs in the array, (1, 2), (2, 3), (3, 4) and (4, 5).

**Example 3:**

**Input:** [1, 3, 1, 5, 4], k = 0

**Output:** 1

**Explanation:** There is one 0-diff pair in the array, (1, 1).

**Note:**

1. The pairs (i, j) and (j, i) count as the same pair.
2. The length of the array won't exceed 10,000.
3. All the integers in the given input belong to the range: [-1e7, 1e7].

class Solution **{**

public**:**

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

**if** **(**k **<** 0**)** **return** 0**;**

unordered\_map**<**int**,** int**>** mp**;**

**for** **(**auto **&**i **:** nums**)** **++**mp**[**i**];**

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

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

**if** **(!**k **&&** i**.**second **>** 1 **||** k **&&** mp**.**count**(**i**.**first**+**k**))**

**++**res**;**

**}**

**return** res**;**

**}**

**};**

### 535. Encode and Decode TinyURL

Medium

Note: This is a companion problem to the [System Design](https://leetcode.com/discuss/interview-question/system-design/) problem: [Design TinyURL](https://leetcode.com/discuss/interview-question/124658/Design-a-URL-Shortener-(-TinyURL-)-System/).

TinyURL is a URL shortening service where you enter a URL such as https://leetcode.com/problems/design-tinyurl and it returns a short URL such as http://tinyurl.com/4e9iAk.

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

### 537. Complex Number Multiplication

Medium

Given two strings representing two [complex numbers](https://en.wikipedia.org/wiki/Complex_number).

You need to return a string representing their multiplication. Note i2 = -1 according to the definition.

**Example 1:**

**Input:** "1+1i", "1+1i"

**Output:** "0+2i"

**Explanation:** (1 + i) \* (1 + i) = 1 + i2 + 2 \* i = 2i, and you need convert it to the form of 0+2i.

**Example 2:**

**Input:** "1+-1i", "1+-1i"

**Output:** "0+-2i"

**Explanation:** (1 - i) \* (1 - i) = 1 + i2 - 2 \* i = -2i, and you need convert it to the form of 0+-2i.

**Note:**

1. The input strings will not have extra blank.
2. The input strings will be given in the form of **a+bi**, where the integer **a** and **b** will both belong to the range of [-100, 100]. And **the output should be also in this form**.

class Solution **{**

public**:**

string complexNumberMultiply**(**string a**,** string b**)** **{**

int a1**,** b1**,** a2**,** b2**;**

sscanf**(**a**.**c\_str**(),** "%d+%di"**,** **&**a1**,** **&**b1**);**

sscanf**(**b**.**c\_str**(),** "%d+%di"**,** **&**a2**,** **&**b2**);**

**return** to\_string**(**a1**\***a2**-**b1**\***b2**)+**"+"**+**to\_string**(**a1**\***b2**+**a2**\***b1**)+** "i"**;**

**}**

**};**

### 538. Convert BST to Greater Tree

Easy

Given a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST.

**Example:**

**Input:** The root of a Binary Search Tree like this:

5

/ \

2 13

**Output:** The root of a Greater Tree like this:

18

/ \

20 13

/\*\*

\* 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**\*** convertBST**(**TreeNode**\*** root**)** **{**

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

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

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

int sum **=** 0**;**

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

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

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

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

**}**

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

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

// visit

sum **=** p**->**val **=** p**->**val **+** sum**;**

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

**}**

**return** root**;**

**}**

**};**

### 539. Minimum Time Difference

Medium

Given a list of 24-hour clock time points in "Hour:Minutes" format, find the minimum **minutes** difference between any two time points in the list.

**Example 1:**

**Input:** ["23:59","00:00"]

**Output:** 1

**Note:**

1. The number of time points in the given list is at least 2 and won't exceed 20000.
2. The input time is legal and ranges from 00:00 to 23:59.

class Solution **{**

public**:**

int findMinDifference**(**vector**<**string**>&** timePoints**)** **{**

vector**<**bool**>** bucket**(**24**\***60**,** **false);**

**for** **(**auto **&**s **:** timePoints**)** **{**

int time **=** stoi**(**s**.**substr**(**0**,** 2**))\***60 **+** stoi**(**s**.**substr**(**3**,** 2**));**

**if** **(**bucket**[**time**])** **return** 0**;**

bucket**[**time**]** **=** **true;**

**}**

int pre **=** **-**1**,** first**,** res **=** INT\_MAX**;**

**for** **(**int i **=** 0**;** i **<** 24**\***60**;** i**++)** **if** **(**bucket**[**i**])** **{**

**if** **(**pre **==** **-**1**)** first **=** pre **=** i**;**

**else** **{**

res **=** min**(**res**,** i**-**pre**);**

pre **=** i**;**

**}**

**}**

**return** min**(**res**,** first**+**24**\***60**-**pre**);**

**}**

**};**

### 540. Single Element in a Sorted Array

Medium

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

**Example 1:**

**Input:** [1,1,2,3,3,4,4,8,8]

**Output:** 2

**Example 2:**

**Input:** [3,3,7,7,10,11,11]

**Output:** 10

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

class Solution **{**

public**:**

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

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

**while** **(**left **+**1 **<** right**)** **{**

int mid **=** left **+** **(**right**-**left**)/**2**;**

**if** **(**mid **%** 2**)** **{**

**if** **(**nums**[**mid**]** **!=** nums**[**mid**-**1**])** right **=** mid**;**

**else** left **=** mid **+** 1**;**

**}**

**else** **{**

**if** **(**nums**[**mid**]** **!=** nums**[**mid**-**1**])** left **=** mid**;**

**else** right **=** mid **-** 1**;**

**}**

**}**

**return** nums**[**left**];**

**}**

**};**

### 541. Reverse String II

Easy

Given a string and an integer k, you need to reverse the first k characters for every 2k characters counting from the start of the string. If there are less than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original.

**Example:**

**Input:** s = "abcdefg", k = 2

**Output:** "bacdfeg"

**Restrictions:**

1. The string consists of lower English letters only.
2. Length of the given string and k will in the range [1, 10000]

class Solution **{**

public**:**

string reverseStr**(**string s**,** int k**)** **{**

auto l **=** s**.**begin**();**

**while** **(**l **!=** s**.**end**())** **{**

auto r **=** **(**s**.**end**()** **-** l **>** k**)** **?** l **+** k **:** s**.**end**();**

reverse**(**l**,** r**);**

l **=** **(**s**.**end**()** **-** r **>** k**)** **?** r **+** k **:** s**.**end**();**

**}**

**return** s**;**

**}**

**};**

### 542. 01 Matrix

Medium

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

The distance between two adjacent cells is 1.

**Example 1:**

**Input:**

[[0,0,0],

[0,1,0],

[0,0,0]]

**Output:**

[[0,0,0],

 [0,1,0],

 [0,0,0]]

**Example 2:**

**Input:**

[[0,0,0],

[0,1,0],

[1,1,1]]

**Output:**

[[0,0,0],

[0,1,0],

[1,2,1]]

**Note:**

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

class Solution **{**

public**:**

vector**<**vector**<**int**>>** updateMatrix**(**vector**<**vector**<**int**>>&** matrix**)** **{**

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

const int MAX **=** 10000**+**10**;**

vector**<**vector**<**int**>>** res**(**n**,** vector**<**int**>(**m**,** MAX**));**

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

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

**if** **(!**matrix**[**i**][**j**])** res**[**i**][**j**]** **=** 0**;**

**else** **{**

int up **=** i **?** res**[**i**-**1**][**j**]** **:** MAX**;**

int left **=** j **?** res**[**i**][**j**-**1**]** **:** MAX**;**

res**[**i**][**j**]** **=** min**(**up**,** left**)+**1**;**

**}**

**}**

**}**

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

**for** **(**int j **=** m**-**1**;** j **>=** 0**;** j**--)** **if** **(**matrix**[**i**][**j**])** **{**

int down **=** i **!=** n**-**1 **?** res**[**i**+**1**][**j**]** **:** MAX**;**

int right **=** j **!=** m**-**1 **?** res**[**i**][**j**+**1**]** **:** MAX**;**

res**[**i**][**j**]** **=** min**(**res**[**i**][**j**],** min**(**down**,** right**)+**1**);**

**}**

**}**

**return** res**;**

**}**

**};**

### 543. Diameter of Binary Tree

Easy

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest** path between any two nodes in a tree. This path may or may not pass through the root.

**Example:**  
Given a binary tree

1

/ \

2 3

/ \

4 5

Return **3**, which is the length of the path [4,2,1,3] or [5,2,1,3].

**Note:** The length of path between two nodes is represented by the number of edges between them.

/\*\*

\* 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 diameterOfBinaryTree**(**TreeNode**\*** root**)** **{**

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

f**(**root**,** res**);**

**return** res**;**

**}**

private**:**

int f**(**TreeNode **\***root**,** int **&**res**)** **{**

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

int L **=** f**(**root**->**left**,** res**);**

int R **=** f**(**root**->**right**,** res**);**

res **=** max**(**res**,** L**+**R**);**

**return** max**(**L**,** R**)** **+** 1**;**

**}**

**};**

### 546. Remove Boxes

Hard

Given several boxes with different colors represented by different positive numbers.   
You may experience several rounds to remove boxes until there is no box left. Each time you can choose some continuous boxes with the same color (composed of k boxes, k >= 1), remove them and get k\*k points.  
Find the maximum points you can get.

**Example 1:**  
Input:

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

Output:

23

Explanation:

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

----> [1, 3, 3, 4, 3, 1] (3\*3=9 points)

----> [1, 3, 3, 3, 1] (1\*1=1 points)

----> [1, 1] (3\*3=9 points)

----> [] (2\*2=4 points)

**Note:** The number of boxes n would not exceed 100.

### 547. Friend Circles

Medium

There are **N** students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a **direct** friend of B, and B is a **direct** friend of C, then A is an **indirect** friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a **N\*N** matrix **M** representing the friend relationship between students in the class. If M[i][j] = 1, then the ith and jth students are **direct** friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

**Example 1:**

**Input:**

[[1,1,0],

[1,1,0],

[0,0,1]]

**Output:** 2

**Explanation:**The 0th and 1st students are direct friends, so they are in a friend circle.   
The 2nd student himself is in a friend circle. So return 2.

**Example 2:**

**Input:**

[[1,1,0],

[1,1,1],

[0,1,1]]

**Output:** 1

**Explanation:**The 0th and 1st students are direct friends, the 1st and 2nd students are direct friends,   
so the 0th and 2nd students are indirect friends. All of them are in the same friend circle, so return 1.

**Note:**

1. N is in range [1,200].
2. M[i][i] = 1 for all students.
3. If M[i][j] = 1, then M[j][i] = 1.

class Solution **{**

public**:**

int findCircleNum**(**vector**<**vector**<**int**>>&** M**)** **{**

int n **=** M**.**size**(),** res **=** n**;**

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

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** fa**[**i**]** **=** i**;**

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

**for** **(**int j **=** 0**;** j **<** i**;** j**++)** **if** **(**M**[**i**][**j**])** **{**

int fa0 **=** find**(**fa**,** i**);**

int fa1 **=** find**(**fa**,** j**);**

**if** **(**fa0 **!=** fa1**)** **{**

fa**[**fa0**]** **=** fa1**;**

res**--;**

**}**

**}**

**}**

**return** res**;**

**}**

private**:**

int find**(**vector**<**int**>** **&**fa**,** int x**)** **{**

**return** x **==** fa**[**x**]** **?** x **:** **(**fa**[**x**]** **=** find**(**fa**,** fa**[**x**]));**

**}**

**};**