目录

[401. Binary Watch 2](#_Toc21268075)

[402. Remove K Digits 4](#_Toc21268076)

[403. Frog Jump 6](#_Toc21268077)

[404. Sum of Left Leaves 9](#_Toc21268078)

[405. Convert a Number to Hexadecimal 11](#_Toc21268079)

[406. Queue Reconstruction by Height 13](#_Toc21268080)

[407. Trapping Rain Water II 16](#_Toc21268081)

[409. Longest Palindrome 19](#_Toc21268082)

[410. Split Array Largest Sum 20](#_Toc21268083)

[412. Fizz Buzz 21](#_Toc21268084)

[413. Arithmetic Slices 23](#_Toc21268085)

[414. Third Maximum Number 25](#_Toc21268086)

[415. Add Strings 27](#_Toc21268087)

[416. Partition Equal Subset Sum 28](#_Toc21268088)

[417. Pacific Atlantic Water Flow 30](#_Toc21268089)

[419. Battleships in a Board 32](#_Toc21268090)

[420. Strong Password Checker 34](#_Toc21268091)

[421. Maximum XOR of Two Numbers in an Array 35](#_Toc21268092)

[423. Reconstruct Original Digits from English 37](#_Toc21268093)

[424. Longest Repeating Character Replacement 39](#_Toc21268094)

[427. Construct Quad Tree 40](#_Toc21268095)

[429. N-ary Tree Level Order Traversal 43](#_Toc21268096)

[430. Flatten a Multilevel Doubly Linked List 45](#_Toc21268097)

[432. All O`one Data Structure 47](#_Toc21268098)

[433. Minimum Genetic Mutation 48](#_Toc21268099)

[434. Number of Segments in a String 51](#_Toc21268100)

[435. Non-overlapping Intervals 52](#_Toc21268101)

[436. Find Right Interval 54](#_Toc21268102)

[437. Path Sum III 56](#_Toc21268103)

[438. Find All Anagrams in a String 58](#_Toc21268104)

[440. K-th Smallest in Lexicographical Order 60](#_Toc21268105)

[441. Arranging Coins 61](#_Toc21268106)

[442. Find All Duplicates in an Array 63](#_Toc21268107)

[443. String Compression 65](#_Toc21268108)

[445. Add Two Numbers II 68](#_Toc21268109)

[446. Arithmetic Slices II - Subsequence 70](#_Toc21268110)

[447. Number of Boomerangs 72](#_Toc21268111)

[448. Find All Numbers Disappeared in an Array 74](#_Toc21268112)

[449. Serialize and Deserialize BST 77](#_Toc21268113)

[450. Delete Node in a BST 78](#_Toc21268114)

### 401. Binary Watch

Easy

A binary watch has 4 LEDs on the top which represent the **hours** (**0-11**), and the 6 LEDs on the bottom represent the **minutes** (**0-59**).

Each LED represents a zero or one, with the least significant bit on the right.



For example, the above binary watch reads "3:25".

Given a non-negative integer *n* which represents the number of LEDs that are currently on, return all possible times the watch could represent.

**Example:**

Input: n = 1  
Return: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]

**Note:**

* The order of output does not matter.
* The hour must not contain a leading zero, for example "01:00" is not valid, it should be "1:00".
* The minute must be consist of two digits and may contain a leading zero, for example "10:2" is not valid, it should be "10:02".

class Solution **{**

public**:**

vector**<**string**>** readBinaryWatch**(**int num**)** **{**

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

**for** **(**int h **=** 0**;** h **<** 12**;** h**++)**

**for** **(**int m **=** 0**;** m **<** 60**;** m**++)** **{**

**if** **(**bitset**<**10**>(**h **<<** 6 **|** m**).**count**()** **==** num**)**

res**.**push\_back**(**to\_string**(**h**)** **+** **(**m **<** 10 **?** ":0" **:** ":"**)**

**+** to\_string**(**m**));**

**}**

**return** res**;**

**}**

**};**

### 402. Remove K Digits

Medium

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

**Note:**

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

**Example 1:**

Input: num = "1432219", k = 3

Output: "1219"

Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

**Example 2:**

Input: num = "10200", k = 1

Output: "200"

Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.

**Example 3:**

Input: num = "10", k = 2

Output: "0"

Explanation: Remove all the digits from the number and it is left with nothing which is 0.

class Solution **{**

public**:**

string removeKdigits**(**string num**,** int k**)** **{**

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

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

**while** **(!**res**.**empty**()** **&&** res**.**back**()** **>** c **&&** k**)** **{**

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

k**--;**

**}**

**if** **(!**res**.**empty**()** **||** c **!=** '0'**)** res**.**push\_back**(**c**);**

**}**

**while** **(!**res**.**empty**()** **&&** k**--)** res**.**pop\_back**();**

**return** res**.**empty**()** **?** "0" **:** res**;**

**}**

**};**

### 403. Frog Jump

Hard

A frog is crossing a river. The river is divided into x units and at each unit there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of stones' positions (in units) in sorted ascending order, determine if the frog is able to cross the river by landing on the last stone. Initially, the frog is on the first stone and assume the first jump must be 1 unit.

If the frog's last jump was *k* units, then its next jump must be either *k* - 1, *k*, or *k* + 1 units. Note that the frog can only jump in the forward direction.

**Note:**

* The number of stones is ≥ 2 and is < 1,100.
* Each stone's position will be a non-negative integer < 231.
* The first stone's position is always 0.

**Example 1:**

**[0,1,3,5,6,8,12,17]**

There are a total of 8 stones.

The first stone at the 0th unit, second stone at the 1st unit,

third stone at the 3rd unit, and so on...

The last stone at the 17th unit.

**Return true**. The frog can jump to the last stone by jumping

1 unit to the 2nd stone, then 2 units to the 3rd stone, then

2 units to the 4th stone, then 3 units to the 6th stone,

4 units to the 7th stone, and 5 units to the 8th stone.

**Example 2:**

**[0,1,2,3,4,8,9,11]**

**Return false**. There is no way to jump to the last stone as

the gap between the 5th and 6th stone is too large.

class Solution **{**

public**:**

bool canCross**(**vector**<**int**>&** stones**)** **{**

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

unordered\_set**<**int**>** s**[**n**];**

s**[**0**].**insert**(**0**);**

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

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

**if** **(**s**[**j**].**empty**())** **continue;**

int d **=** stones**[**i**]** **-** stones**[**j**];**

**if** **(**d **>** n**)** **continue;**

**if** **(**s**[**j**].**count**(**d**-**1**)** **||** s**[**j**].**count**(**d**)** **||** s**[**j**].**count**(**d**+**1**)** **)** **{**

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

**}**

**}**

**}**

**return** **!**s**[**n**-**1**].**empty**();**

**}**

**};**

class Solution **{**

public**:**

bool canCross**(**vector**<**int**>&** stones**)** **{**

int N **=** stones**.**size**();**

vector**<**vector**<**bool**>>** dp**(**N**,** vector**<**bool**>(**N**+**1**,** **false));**

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

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

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

int d **=** stones**[**i**]** **-** stones**[**j**];**

**if** **(**d **<** 0 **||** d **>** N **||** **!**dp**[**j**][**d**])** **continue;**

dp**[**i**][**d**]** **=** **true;**

**if** **(**d **-** 1 **>=** 0**)** dp**[**i**][**d**-**1**]** **=** **true;**

**if** **(**d **+** 1 **<=** N**)** dp**[**i**][**d**+**1**]** **=** **true;**

**if** **(**i **==** N**-**1**)** **return** **true;**

**}**

**}**

**return** **false;**

**}**

**};**

### 404. Sum of Left Leaves

Easy

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

**Example:**

3

/ \

9 20

/ \

15 7

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

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution **{**

public**:**

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

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

dfs**(**root**,** res**,** **false);**

**return** res**;**

**}**

private**:**

void dfs**(**TreeNode**\*** root**,** int **&**res**,** bool type**)** **{**

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

**if** **(!**root**->**left **&&** **!**root**->**right **&&** type**)** **{**

res **+=** root**->**val**;**

**}**

dfs**(**root**->**left**,** res**,** **true);**

dfs**(**root**->**right**,** res**,** **false);**

**}**

**};**

### 405. Convert a Number to Hexadecimal

Easy

Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, [two’s complement](https://en.wikipedia.org/wiki/Two%27s_complement) method is used.

**Note:**

1. All letters in hexadecimal (a-f) must be in lowercase.
2. The hexadecimal string must not contain extra leading 0s. If the number is zero, it is represented by a single zero character '0'; otherwise, the first character in the hexadecimal string will not be the zero character.
3. The given number is guaranteed to fit within the range of a 32-bit signed integer.
4. You **must not use *any* method provided by the library** which converts/formats the number to hex directly.

**Example 1:**

Input:

26

Output:

"1a"

**Example 2:**

Input:

-1

Output:

"ffffffff"

class Solution **{**

public**:**

string toHex**(**int num**)** **{**

const string HEX **=** "0123456789abcdef"**;**

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

string res**;**

unsigned int n **=** num**;**

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

res **=** HEX**[(**n **&** 0xf**)]** **+** res**;**

n **>>=** 4**;**

**}**

**return** res**;**

**}**

**};**

### 406. Queue Reconstruction by Height

Medium

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

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

**Example**

Input:

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

Output:

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

class Solution **{**

public**:**

vector**<**vector**<**int**>>** reconstructQueue**(**vector**<**vector**<**int**>>&** people**)** **{**

sort**(**people**.**begin**(),** people**.**end**(),** **[](**const vector**<**int**>** **&**lhs**,** const vector**<**int**>** **&**rhs**)** **{**

**if** **(**lhs**[**0**]** **==** rhs**[**0**])** **return** lhs**[**1**]** **>=** rhs**[**1**];**

**else** **return** lhs**[**0**]** **<** rhs**[**0**];**

**});**

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

vector**<**int**>** indices**;**

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

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

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

**}**

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

int idx **=** indices**[**people**[**i**][**1**]];**

res**[**indices**[**people**[**i**][**1**]]]** **=** people**[**i**];**

indices**.**erase**(**indices**.**begin**()** **+** people**[**i**][**1**]);**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

vector**<**vector**<**int**>>** reconstructQueue**(**vector**<**vector**<**int**>>&** people**)** **{**

sort**(**people**.**begin**(),** people**.**end**(),** **[](**const vector**<**int**>** **&**lhs**,** const vector**<**int**>** **&**rhs**)** **{**

**if** **(**lhs**[**0**]** **==** rhs**[**0**])** **return** lhs**[**1**]** **<** rhs**[**1**];**

**else** **return** lhs**[**0**]** **>** rhs**[**0**];**

**});**

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

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

res**.**insert**(**res**.**begin**()+**v**[**1**],** v**);**

**}**

**return** res**;**

**}**

**};**

### 407. Trapping Rain Water II

Hard

Given an m x n matrix of positive integers representing the height of each unit cell in a 2D elevation map, compute the volume of water it is able to trap after raining.

**Note:**

Both *m* and *n* are less than 110. The height of each unit cell is greater than 0 and is less than 20,000.

**Example:**

Given the following 3x6 height map:

[

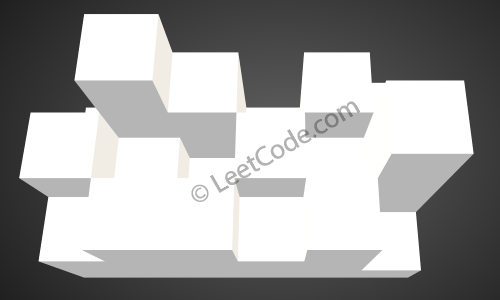
[1,4,3,1,3,2],

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

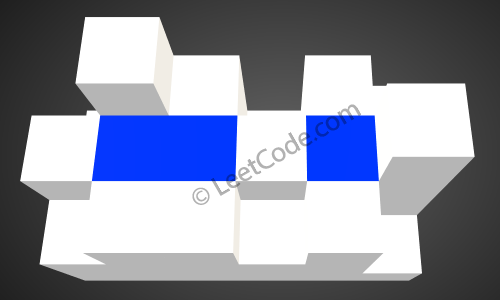
[2,3,3,2,3,1]

]

Return 4.



The above image represents the elevation map [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]] before the rain.



After the rain, water is trapped between the blocks. The total volume of water trapped is 4.

class Solution **{**

public**:**

int trapRainWater**(**vector**<**vector**<**int**>>&** heightMap**)** **{**

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

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

priority\_queue**<**node**>** pq**;**

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

**if** **(**i **==** 0 **||** i **==** n**-**1**)** **{**

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

pq**.**push**({**i**,** j**,** heightMap**[**i**][**j**]});**

heightMap**[**i**][**j**]** **=** **-**1**;**

**}**

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

pq**.**push**({**i**,** 0**,** heightMap**[**i**][**0**]});**

pq**.**push**({**i**,** m**-**1**,** heightMap**[**i**][**m**-**1**]});**

heightMap**[**i**][**0**]** **=** heightMap**[**i**][**m**-**1**]** **=** **-**1**;**

**}**

**}**

int res **=** 0**,** MIN\_height **=** INT\_MIN**;**

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

node t **=** pq**.**top**();**

pq**.**pop**();**

MIN\_height **=** max**(**MIN\_height**,** t**.**value**);**

**for** **(**int k **=** 0**;** k **<** 4**;** k**++)** **{**

int xx **=** t**.**x**+**dx**[**k**],** yy **=** t**.**y**+**dy**[**k**];**

**if** **(**inside**(**xx**,** yy**)** **&&** heightMap**[**xx**][**yy**]** **>=** 0**)** **{**

int **&**h **=** heightMap**[**xx**][**yy**];**

**if** **(**h **<** MIN\_height**)** res **+=** MIN\_height**-**h**;**

pq**.**push**({**xx**,** yy**,** h**});**

h **=** **-**1**;**

**}**

**}**

**}**

**return** res**;**

**}**

private**:**

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

const int dx**[**4**]** **=** **{**0**,**0**,**1**,-**1**},** dy**[**4**]** **=** **{-**1**,**1**,**0**,**0**};**

struct node**{**

int x**,** y**,** value**;**

node**(**int x**,** int y**,** int v**):**x**(**x**),** y**(**y**),** value**(**v**){}**

bool **operator** **<** **(**const node **&**rhs**)** const **{**

**return** value **>** rhs**.**value**;**

**}**

**};**

bool inside**(**int x**,** int y**)** **{**

**return** x **>=** 0 **&&** y **>=** 0 **&&** x **<** n **&&** y **<** m**;**

**}**

**};**

### 409. Longest Palindrome

Easy

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

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

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

**Example:**

Input:

"abccccdd"

Output:

7

Explanation:

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

class Solution **{**

public**:**

int longestPalindrome**(**string s**)** **{**

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

**for** **(**auto c **:** s**)** m**[**c**]++;**

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

**for** **(**auto pci **:** m**)** **{**

res **+=** pci**.**second**/**2**\***2**;**

**if** **(**res **%** 2 **==** 0 **&&** pci**.**second **%** 2 **==** 1**)** res**++;**

**}**

**return** res**;**

**}**

**};**

### 410. Split Array Largest Sum

Hard

Given an array which consists of non-negative integers and an integer *m*, you can split the array into *m* non-empty continuous subarrays. Write an algorithm to minimize the largest sum among these *m* subarrays.

**Note:**  
If *n* is the length of array, assume the following constraints are satisfied:

* 1 ≤ *n* ≤ 1000
* 1 ≤ *m* ≤ min(50, *n*)

**Examples:**

Input:

**nums** = [7,2,5,10,8]

**m** = 2

Output:

18

Explanation:

There are four ways to split **nums** into two subarrays.

The best way is to split it into **[7,2,5]** and **[10,8]**,

where the largest sum among the two subarrays is only 18.

### 412. Fizz Buzz

Easy

Write a program that outputs the string representation of numbers from 1 to *n*.

But for multiples of three it should output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.

**Example:**

n = 15,

Return:

[

"1",

"2",

"Fizz",

"4",

"Buzz",

"Fizz",

"7",

"8",

"Fizz",

"Buzz",

"11",

"Fizz",

"13",

"14",

"FizzBuzz"

]

class Solution **{**

public**:**

vector**<**string**>** fizzBuzz**(**int n**)** **{**

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

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

**if** **(**i **%** 3 **!=** 0 **&&** i **%** 5 **!=** 0**)** **{**

res**.**push\_back**(**to\_string**(**i**));**

**}** **else** **if** **(**i **%** 3 **!=** 0**)** **{**

res**.**push\_back**(**"Buzz"**);**

**}** **else** **if** **(**i **%** 5 **!=** 0**)** **{**

res**.**push\_back**(**"Fizz"**);**

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

res**.**push\_back**(**"FizzBuzz"**);**

**}**

**}**

**return** res**;**

**}**

**};**

### 413. Arithmetic Slices

Medium

A sequence of number is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequence:

1, 3, 5, 7, 9

7, 7, 7, 7

3, -1, -5, -9

The following sequence is not arithmetic.

1, 1, 2, 5, 7

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 <= P < Q < N.

A slice (P, Q) of array A is called arithmetic if the sequence:  
A[P], A[p + 1], ..., A[Q - 1], A[Q] is arithmetic. In particular, this means that P + 1 < Q.

The function should return the number of arithmetic slices in the array A.

**Example:**

A = [1, 2, 3, 4]

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.

class Solution **{**

public**:**

int numberOfArithmeticSlices**(**vector**<**int**>&** A**)** **{**

**if** **(**A**.**size**()** **<** 3**)** **return** 0**;**

int cnt **=** 0**,** res **=** 0**;**

**for(**int i **=** 2**;** i **<** A**.**size**();** i**++)** **{**

**if** **(**A**[**i**-**1**]** **-** A**[**i**-**2**]** **==** A**[**i**]** **-** A**[**i**-**1**])** res **+=** **++**cnt**;**

**else** cnt **=** 0**;**

**}**

**return** res**;**

**}**

**};**

### 414. Third Maximum Number

Easy

Given a **non-empty** array of integers, return the **third** maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).

**Example 1:**

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

**Output:** 1

**Explanation:** The third maximum is 1.

**Example 2:**

**Input:** [1, 2]

**Output:** 2

**Explanation:** The third maximum does not exist, so the maximum (2) is returned instead.

**Example 3:**

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

**Output:** 1

**Explanation:** Note that the third maximum here means the third maximum distinct number.

Both numbers with value 2 are both considered as second maximum.

class Solution **{**

public**:**

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

long long a**,** b**,** c**;**

a **=** b **=** c **=** LLONG\_MIN**;**

**for** **(**auto num **:** nums**)** **{**

**if** **(**num **<=** c **||** num **==** b **||** num **==** a**)** **continue;**

c **=** num**;**

**if** **(**c **>** b**)** swap**(**b**,** c**);**

**if** **(**b **>** a**)** swap**(**a**,** b**);**

**}**

**return** c **==** LLONG\_MIN **?** a **:** c**;**

**}**

**};**

class Solution **{**

public**:**

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

set**<**int**>** top3**;**

**for** **(**int num **:** nums**)** **{**

top3**.**insert**(**num**);**

**if** **(**top3**.**size**()** **>** 3**)**

top3**.**erase**(**top3**.**begin**());**

**}**

**return** top3**.**size**()** **==** 3 **?** **\***top3**.**begin**()** **:** **\***top3**.**rbegin**();**

**}**

**};**

### 415. Add Strings

Easy

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

**Note:**

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

class Solution **{**

public**:**

string addStrings**(**string num1**,** string num2**)** **{**

reverse**(**num1**.**begin**(),** num1**.**end**());**

reverse**(**num2**.**begin**(),** num2**.**end**());**

string res**;**

int carry **=** 0**,** n **=** num1**.**length**(),** m **=** num2**.**length**();**

int i **=** 0**,** j **=** 0**;**

**while** **(**i **<** n **||** j **<** m**)** **{**

carry **+=** i **<** n **?** num1**[**i**++]-**'0' **:** 0**;**

carry **+=** j **<** m **?** num2**[**j**++]-**'0' **:** 0**;**

res **+=** carry **%** 10 **+** '0'**;**

carry **/=** 10**;**

**}**

**if** **(**carry**)** res **+=** '1'**;**

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

**return** res**;**

**}**

**};**

### 416. Partition Equal Subset Sum

Medium

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

**Note:**

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

**Example 1:**

Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].

**Example 2:**

Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.

class Solution **{**

public**:**

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

int sum **=** accumulate**(**nums**.**begin**(),** nums**.**end**(),** 0**);**

**if** **(**sum **%** 2 **!=** 0**)** **return** **false;**

int V **=** sum**/**2**,** n **=** nums**.**size**();**

vector**<**bool**>** dp**(**V**+**1**,** **false);**

dp**[**0**]** **=** **true;**

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

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

**if** **(**j **<** nums**[**i**])** **continue;**

dp**[**j**]** **=** dp**[**j**]** **||** dp**[**j**-**nums**[**i**]];**

**}**

**if** **(**dp**[**V**])** **return** **true;**

**}**

**return** **false;**

**}**

**};**

class Solution **{**

public**:**

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

bitset**<**100**\***200**/**2**+**1**>** bits**(**1**);**

int sum **=** accumulate**(**nums**.**begin**(),** nums**.**end**(),** 0**);**

**if** **(**sum **%** 2 **!=** 0**)** **return** **false;**

**for** **(**auto n **:** nums**)** bits **|=** bits **<<** n**;**

**return** bits**[**sum **>>** 1**];**

**}**

**};**

### 417. Pacific Atlantic Water Flow

Medium

Given an m x n matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

**Note:**

1. The order of returned grid coordinates does not matter.
2. Both *m* and *n* are less than 150.

**Example:**

Given the following 5x5 matrix:

Pacific ~ ~ ~ ~ ~

~ 1 2 2 3 (5) \*

~ 3 2 3 (4) (4) \*

~ 2 4 (5) 3 1 \*

~ (6) (7) 1 4 5 \*

~ (5) 1 1 2 4 \*

\* \* \* \* \* Atlantic

Return:

[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (positions with parentheses in above matrix).

class Solution **{**

public**:**

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

**if** **(**matrix**.**empty**())** **return** **{};**

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

buffer**.**resize**(**n**,** vector**<**int**>** **(**m**,** 0**));**

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

**for** **(**int j **=** 0**;** j **<** m**;** **++**j**)** dfs**(**0**,** j**,** matrix**);**

**for** **(**int i **=** 1**;** i **<** n**;** **++**i**)** dfs**(**i**,** 0**,** matrix**);**

visit **=** vector**<**vector**<**bool**>>(**n**,** vector**<**bool**>** **(**m**,** **false));**

**for** **(**int j **=** 0**;** j **<** m**;** **++**j**)** dfs**(**n**-**1**,** j**,** matrix**);**

**for** **(**int i **=** 0**;** i **<** n**-**1**;** **++**i**)** dfs**(**i**,** m**-**1**,** matrix**);**

**return** res**;**

**}**

private**:**

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

vector**<**vector**<**int**>>** res**,** buffer**;**

vector**<**vector**<**bool**>>** visit**;**

const vector**<**int**>** dx**{**0**,**0**,**1**,-**1**};**

const vector**<**int**>** dy**{-**1**,**1**,**0**,**0**};**

void dfs**(**int i**,** int j**,** vector**<**vector**<**int**>>&** matrix**)** **{**

**if** **(**visit**[**i**][**j**])** **return;**

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

**if** **(++**buffer**[**i**][**j**]** **==** 2**){**

res**.**push\_back**({**i**,** j**});**

**}**

**for** **(**int k **=** 0**;** k **<** 4**;** **++**k**)** **{**

int x **=** i **+** dx**[**k**],** y **=** j **+** dy**[**k**];**

**if** **(**x **<** 0 **||** y **<** 0 **||** x **>=** n **||** y **>=** m**)** **continue;**

**if** **(!**visit**[**x**][**y**]** **&&** matrix**[**x**][**y**]** **>=** matrix**[**i**][**j**])** **{**

dfs**(**x**,** y**,** matrix**);**

**}**

**}**

**}**

**};**

### 419. Battleships in a Board

Medium

Given an 2D board, count how many battleships are in it. The battleships are represented with 'X's, empty slots are represented with '.'s. You may assume the following rules:

* You receive a valid board, made of only battleships or empty slots.
* Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape 1xN (1 row, N columns) or Nx1 (N rows, 1 column), where N can be of any size.
* At least one horizontal or vertical cell separates between two battleships - there are no adjacent battleships.

**Example:**

X..X

...X

...X

In the above board there are 2 battleships.

**Invalid Example:**

...X

XXXX

...X

This is an invalid board that you will not receive - as battleships will always have a cell separating between them.

**Follow up:**  
Could you do it in **one-pass**, using only **O(1) extra memory** and **without modifying** the value of the board?

class Solution **{**

public**:**

int countBattleships**(**vector**<**vector**<**char**>>&** board**)** **{**

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

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

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

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

**if** **(**board**[**i**][**j**]** **==** 'X'**)** **{**

res**++;**

**if** **(**j **!=** 0 **&&** board**[**i**][**j**-**1**]** **==** 'X'**)** res**--;**

**if** **(**i **!=** 0 **&&** board**[**i**-**1**][**j**]** **==** 'X'**)** res**--;**

**}**

**}**

**}**

**return** res**;**

**}**

**};**

### 420. Strong Password Checker

Hard

A password is considered strong if below conditions are all met:

1. It has at least 6 characters and at most 20 characters.
2. It must contain at least one lowercase letter, at least one uppercase letter, and at least one digit.
3. It must NOT contain three repeating characters in a row ("...aaa..." is weak, but "...aa...a..." is strong, assuming other conditions are met).

Write a function strongPasswordChecker(s), that takes a string s as input, and return the **MINIMUM** change required to make s a strong password. If s is already strong, return 0.

Insertion, deletion or replace of any one character are all considered as one change.

### 421. Maximum XOR of Two Numbers in an Array

Medium

Given a **non-empty** array of numbers, a0, a1, a2, … , an-1, where 0 ≤ ai < 231.

Find the maximum result of ai XOR aj, where 0 ≤ *i*, *j* < *n*.

Could you do this in O(*n*) runtime?

**Example:**

**Input:** [3, 10, 5, 25, 2, 8]

**Output:** 28

**Explanation:** The maximum result is **5** ^ **25** = 28.

class Solution **{**

public**:**

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

TrieNode **\***Trie **=** **new** TrieNode**(-**1**);**

**for** **(**auto **&**i **:** nums**)** Trie**->**insert**(**i**);**

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

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

res **=** max**(**res**,** Trie**->**search**(**i**));**

**}**

**return** res**;**

**}**

class TrieNode**{**

public**:**

TrieNode**(**int v**)** **:** left**(nullptr),** right**(nullptr),** val**(**v**)** **{}**

void insert**(**int i**)** **{**

TrieNode **\***p **=** **this;**

**for** **(**int k **=** 31**;** k **>=** 0**;** **--**k**)** **{**

**if** **(**i **&** **(**1 **<<** k**))** **{**

**if** **(!**p**->**left**)** p**->**left **=** **new** TrieNode**(**1**);**

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

**}**

**else** **{**

**if** **(!**p**->**right**)** p**->**right **=** **new** TrieNode**(**0**);**

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

**}**

**}**

**}**

int search**(**int i**)** **{**

TrieNode **\***p **=** **this;**

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

**for(**int k **=** 31**;** k **>=** 0**;** k**--){**

int tmp **=** i **&** **(**1 **<<** k**);**

**if** **(**p**->**left **&&** p**->**right**){**

**if** **(!**tmp**)** p **=** p**->**left**;**

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

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

p **=** p**->**left **?** p**->**left **:** p**->**right**;**

**}**

sum **+=** tmp **^** **(**p**->**val **<<** k**);**

**}**

**return** sum**;**

**}**

private**:**

int val**;**

TrieNode **\***left**,** **\***right**;**

**};**

**};**

### 423. Reconstruct Original Digits from English

Medium

Given a **non-empty** string containing an out-of-order English representation of digits 0-9, output the digits in ascending order.

**Note:**

1. Input contains only lowercase English letters.
2. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as "abc" or "zerone" are not permitted.
3. Input length is less than 50,000.

**Example 1:**

Input: "owoztneoer"

Output: "012"

**Example 2:**

Input: "fviefuro"

Output: "45"

class Solution **{**

public**:**

string originalDigits**(**string s**)** **{**

string res**;**

chars**.**resize**(**26**,** 0**);**

**for** **(**auto **&**i **:** s**)** chars**[**i**-**'a'**]++;**

**for** **(**auto **&**pci **:** vect**)** **{**

f**(**pci**.**first**,** pci**.**second**,** res**);**

**}**

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

**return** res**;**

**}**

private**:**

vector**<**int**>** chars**;**

vector**<**string**>** nums**{**"zero"**,** "one"**,** "two"**,** "three"**,** "four"**,**

"five"**,** "six"**,** "seven"**,** "eight"**,** "nine"**};**

vector**<**pair**<**char**,** int**>>** vect**{{**'z'**,** 0**},** **{**'w'**,** 2**},** **{**'u'**,** 4**},**

**{**'x'**,** 6**},** **{**'r'**,** 3**},** **{**'f'**,** 5**},**

**{**'s'**,** 7**},** **{**'h'**,** 8**},** **{**'o'**,** 1**},** **{**'i'**,** 9**}};**

void f**(**char c**,** int number**,** string **&**res**)** **{**

int cnt **=** chars**[**c**-**'a'**];**

**for** **(**auto **&**i **:** nums**[**number**])** chars**[**i**-**'a'**]** **-=** cnt**;**

res **+=** string**(**cnt**,** '0'**+**number**);**

**}**

**};**

### 424. Longest Repeating Character Replacement

Medium

Given a string s that consists of only uppercase English letters, you can perform at most k operations on that string.

In one operation, you can choose **any** character of the string and change it to any other uppercase English character.

Find the length of the longest sub-string containing all repeating letters you can get after performing the above operations.

**Note:**  
Both the string's length and *k* will not exceed 104.

**Example 1:**

**Input:**

s = "ABAB", k = 2

**Output:**

4

**Explanation:**

Replace the two 'A's with two 'B's or vice versa.

**Example 2:**

**Input:**

s = "AABABBA", k = 1

**Output:**

4

**Explanation:**

Replace the one 'A' in the middle with 'B' and form "AABBBBA".

The substring "BBBB" has the longest repeating letters, which is 4.

### 427. Construct Quad Tree

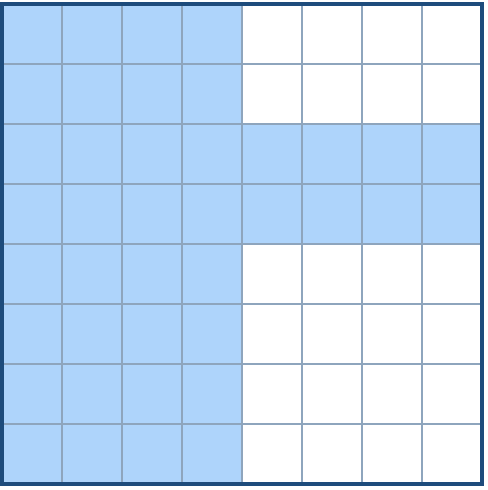
Medium

We want to use quad trees to store an N x N boolean grid. Each cell in the grid can only be true or false. The root node represents the whole grid. For each node, it will be subdivided into four children nodes **until the values in the region it represents are all the same**.

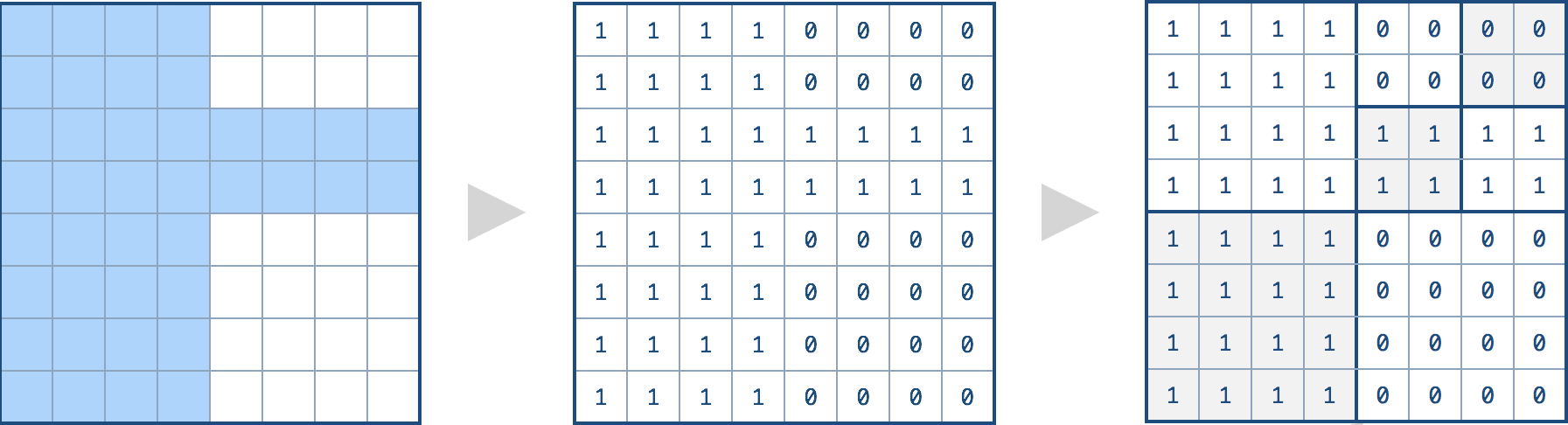
Each node has another two boolean attributes : isLeaf and val. isLeaf is true if and only if the node is a leaf node. The val attribute for a leaf node contains the value of the region it represents.

Your task is to use a quad tree to represent a given grid. The following example may help you understand the problem better:

Given the 8 x 8 grid below, we want to construct the corresponding quad tree:

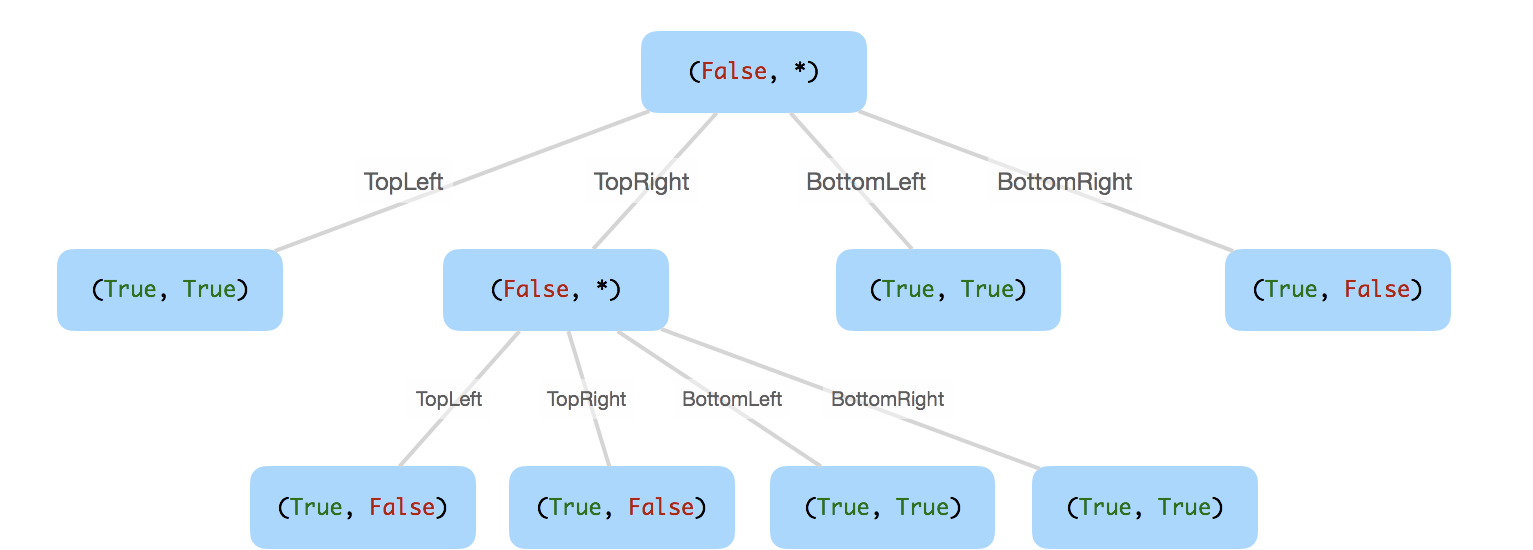


It can be divided according to the definition above:



The corresponding quad tree should be as following, where each node is represented as a (isLeaf, val) pair.

For the non-leaf nodes, val can be arbitrary, so it is represented as \*.



**Note:**

1. N is less than 1000 and guaranteened to be a power of 2.
2. If you want to know more about the quad tree, you can refer to its [wiki](https://en.wikipedia.org/wiki/Quadtree).

class Solution **{**

public**:**

Node**\*** construct**(**vector**<**vector**<**int**>>&** grid**)** **{**

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

**return** dfs**(**0**,** 0**,** n**,** n**,** grid**);**

**}**

private**:**

Node**\*** dfs**(**int x1**,** int y1**,** int x2**,** int y2**,** vector**<**vector**<**int**>>&** grid**)** **{**

int x\_mid **=** x1 **+** **(**x2**-**x1**)/**2**;**

int y\_mid **=** y1 **+** **(**y2**-**y1**)/**2**;**

int val **=** grid**[**x1**][**y1**];**

**for** **(**int r **=** x1**;** r **<** x2**;** r**++)** **{**

**for** **(**int c **=** y1**;** c **<** y2**;** c**++)** **{**

**if** **(**grid**[**r**][**c**]** **!=** val**)** **{**

**return** **new** Node**(**

**true,** **false,**

dfs**(**x1**,** y1**,** x\_mid**,** y\_mid**,** grid**),**

dfs**(**x1**,** y\_mid**,** x\_mid**,** y2**,** grid**),**

dfs**(**x\_mid**,** y1**,** x2**,** y\_mid**,** grid**),**

dfs**(**x\_mid**,** y\_mid**,** x2**,** y2**,** grid**)**

**);**

**}**

**}**

**}**

**return** **new** Node**(**val **!=** 0**,** **true,** **nullptr,** **nullptr,** **nullptr,** **nullptr);**

**}**

**};**

### 429. N-ary Tree Level Order Traversal

Easy

Given an n-ary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example, given a 3-ary tree:



We should return its level order traversal:

[

[1],

[3,2,4],

[5,6]

]

**Note:**

1. The depth of the tree is at most 1000.
2. The total number of nodes is at most 5000.

/\*

// Definition for a Node.

class Node {

public:

int val;

vector<Node\*> children;

Node() {}

Node(int \_val, vector<Node\*> \_children) {

val = \_val;

children = \_children;

}

};

\*/

class Solution **{**

public**:**

vector**<**vector**<**int**>>** levelOrder**(**Node**\*** root**)** **{**

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

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

queue**<**Node**\*>** q**;**

q**.**push**(**root**);**

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

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

vector**<**int**>** vect**;**

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

vect**.**push\_back**(**q**.**front**()->**val**);**

auto **&**child **=** q**.**front**()->**children**;**

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

**for** **(**auto **&**i **:** child**)** q**.**push**(**i**);**

**}**

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

**}**

**return** res**;**

**}**

**};**

### 430. Flatten a Multilevel Doubly Linked List

Medium

49185FavoriteShare

You are given a doubly linked list which in addition to the next and previous pointers, it could have a child pointer, which may or may not point to a separate doubly linked list. These child lists may have one or more children of their own, and so on, to produce a multilevel data structure, as shown in the example below.

Flatten the list so that all the nodes appear in a single-level, doubly linked list. You are given the head of the first level of the list.

**Example:**

**Input:**

1---2---3---4---5---6--NULL

|

7---8---9---10--NULL

|

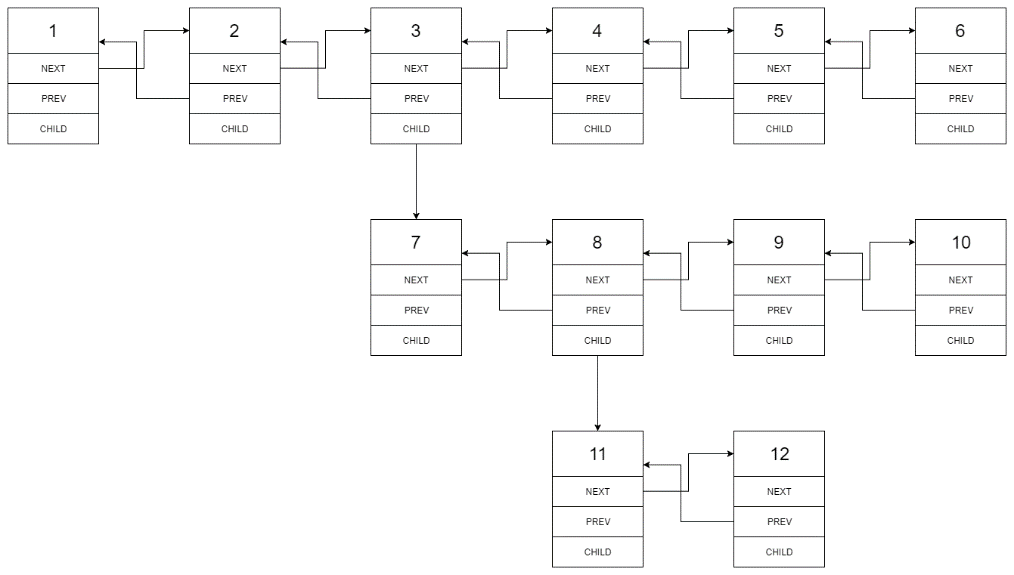
11--12--NULL

**Output:**

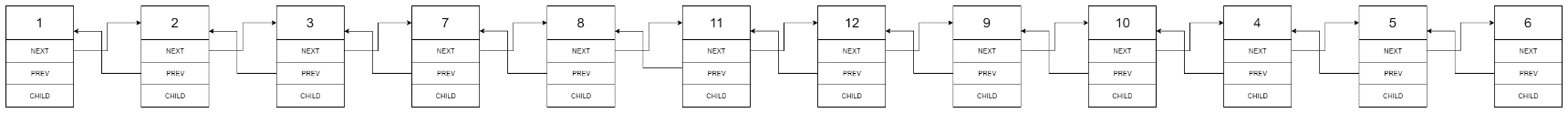
1-2-3-7-8-11-12-9-10-4-5-6-NULL

**Explanation for the above example:**

Given the following multilevel doubly linked list:



We should return the following flattened doubly linked list:



### 432. All O`one Data Structure

Hard

Implement a data structure supporting the following operations:

1. Inc(Key) - Inserts a new key with value 1. Or increments an existing key by 1. Key is guaranteed to be a **non-empty** string.
2. Dec(Key) - If Key's value is 1, remove it from the data structure. Otherwise decrements an existing key by 1. If the key does not exist, this function does nothing. Key is guaranteed to be a **non-empty** string.
3. GetMaxKey() - Returns one of the keys with maximal value. If no element exists, return an empty string "".
4. GetMinKey() - Returns one of the keys with minimal value. If no element exists, return an empty string "".

Challenge: Perform all these in O(1) time complexity.

### 433. Minimum Genetic Mutation

Medium

A gene string can be represented by an 8-character long string, with choices from "A", "C", "G", "T".

Suppose we need to investigate about a mutation (mutation from "start" to "end"), where ONE mutation is defined as ONE single character changed in the gene string.

For example, "AACCGGTT" -> "AACCGGTA" is 1 mutation.

Also, there is a given gene "bank", which records all the valid gene mutations. A gene must be in the bank to make it a valid gene string.

Now, given 3 things - start, end, bank, your task is to determine what is the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

**Note:**

1. Starting point is assumed to be valid, so it might not be included in the bank.
2. If multiple mutations are needed, all mutations during in the sequence must be valid.
3. You may assume start and end string is not the same.

**Example 1:**

start: "AACCGGTT"

end: "AACCGGTA"

bank: ["AACCGGTA"]

return: 1

**Example 2:**

start: "AACCGGTT"

end: "AAACGGTA"

bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]

return: 2

**Example 3:**

start: "AAAAACCC"

end: "AACCCCCC"

bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]

return: 3

class Solution **{**

public**:**

int minMutation**(**string start**,** string end**,** vector**<**string**>** **&**bank**)** **{**

vector**<**char**>** gene**{**'A'**,** 'T'**,** 'C'**,** 'G'**};**

queue**<**string**>** q**;**

q**.**push**(**start**);**

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

unordered\_set**<**string**>** My\_set**(**bank**.**begin**(),** bank**.**end**());**

**if** **(!**My\_set**.**count**(**end**))** **return** **-**1**;**

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

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

res**++;**

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

string s **=** q**.**front**();**

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

**for** **(**int i **=** 0**;** i **<** 8**;** i**++)** **{**

int c **=** s**[**i**];**

**for** **(**int j **=** 0**;** j **<** 4**;** j**++)** **{**

**if** **(**gene**[**j**]** **==** c**)** **continue;**

s**[**i**]** **=** gene**[**j**];**

auto it **=** My\_set**.**find**(**s**);**

**if** **(**it **!=** My\_set**.**end**())** **{**

**if** **(**s **==** end**)** **return** res**;**

My\_set**.**erase**(**it**);**

q**.**push**(**s**);**

**}**

**}**

s**[**i**]** **=** c**;**

**}**

**}**

**}**

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

**}**

**};**

### 434. Number of Segments in a String

Easy

Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any **non-printable** characters.

**Example:**

**Input:** "Hello, my name is John"

**Output:** 5

class Solution **{**

public**:**

int countSegments**(**string s**)** **{**

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

istringstream ss**(**s**);**

**while** **(**ss **>>** s**)** res**++;**

**return** res**;**

**}**

**};**

### 435. Non-overlapping Intervals

Medium

Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

**Example 1:**

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

**Output:** 1

**Explanation:** [1,3] can be removed and the rest of intervals are non-overlapping.

**Example 2:**

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

**Output:** 2

**Explanation:** You need to remove two [1,2] to make the rest of intervals non-overlapping.

**Example 3:**

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

**Output:** 0

**Explanation:** You don't need to remove any of the intervals since they're already non-overlapping.

**Note:**

1. You may assume the interval's end point is always bigger than its start point.
2. Intervals like [1,2] and [2,3] have borders "touching" but they don't overlap each other.

class Solution **{**

public**:**

int eraseOverlapIntervals**(**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**);**

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

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

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

auto **&**v **=** intervals**[**i**];**

**if** **(**stk**.**empty**()** **||** stk**.**top**()[**1**]** **<=** v**[**0**])** **{**

stk**.**push**(**v**);**

i**++;**

**}**

**else** **if** **(**stk**.**top**()[**1**]** **>** v**[**1**])** stk**.**pop**();**

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

**}**

**return** n **-** stk**.**size**();**

**}**

**};**

### 436. Find Right Interval

Medium

Given a set of intervals, for each of the interval i, check if there exists an interval j whose start point is bigger than or equal to the end point of the interval i, which can be called that j is on the "right" of i.

For any interval i, you need to store the minimum interval j's index, which means that the interval j has the minimum start point to build the "right" relationship for interval i. If the interval j doesn't exist, store -1 for the interval i. Finally, you need output the stored value of each interval as an array.

1. You may assume the interval's end point is always bigger than its start point.
2. You may assume none of these intervals have the same start point.

**Example 1:**

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

**Output:** [-1]

**Explanation:** There is only one interval in the collection, so it outputs -1.

**Example 2:**

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

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

**Explanation:** There is no satisfied "right" interval for [3,4].

For [2,3], the interval [3,4] has minimum-"right" start point;

For [1,2], the interval [2,3] has minimum-"right" start point.

**Example 3:**

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

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

**Explanation:** There is no satisfied "right" interval for [1,4] and [3,4].

For [2,3], the interval [3,4] has minimum-"right" start point.

### 437. Path Sum III

Easy

You are given a binary tree in which each node contains an integer value.

Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

**Example:**

root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

10

/ \

**5** **-3**

**/** **\** **\**

**3** **2** **11**

/ \ **\**

3 -2 **1**

Return 3. The paths that sum to 8 are:

1. 5 -> 3

2. 5 -> 2 -> 1

3. -3 -> 11

class Solution **{**

public**:**

int pathSum**(**TreeNode**\*** root**,** int sum**)** **{**

**return** dfs**(**root**,** sum**,** 0**);**

**}**

private**:**

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

int dfs**(**TreeNode **\***root**,** int sum**,** int pre**)** **{**

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

int cur **=** root**->**val **+** pre**;**

int res **=** **(**cur **==** sum**)** **+** **(**mp**.**count**(**cur **-** sum**)** **?**

mp**[**cur **-** sum**]** **:** 0**);**

mp**[**cur**]++;**

res **+=** dfs**(**root**->**left**,** sum**,** cur**)** **+** dfs**(**root**->**right**,** sum**,** cur**);**

mp**[**cur**]--;**

**return** res**;**

**}**

**};**

### 438. Find All Anagrams in a String

Medium

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

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

The order of output does not matter.

**Example 1:**

**Input:**

s: "cbaebabacd" p: "abc"

**Output:**

[0, 6]

**Explanation:**

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

**Example 2:**

**Input:**

s: "abab" p: "ab"

**Output:**

[0, 1, 2]

**Explanation:**

The substring with start index = 0 is "ab", which is an anagram of "ab".

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

The substring with start index = 2 is "ab", which is an anagram of "ab".

class Solution **{**

public**:**

vector**<**int**>** findAnagrams**(**string s**,** string p**)** **{**

int n **=** s**.**length**(),** m **=** p**.**length**();**

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

unordered\_map**<**char**,** int**>** m1**,** m2**;**

**for** **(**auto **&**c **:** p**)** m1**[**c**]++;**

int cnt **=** 0**,** i **=** 0**,** j **=** 0**;**

**while(**i **<=** j**)** **{**

**while** **(**j **<** n **&&** cnt **<** m**)** **{**

**if** **(**m1**.**count**(**s**[**j**])** **&&** m1**[**s**[**j**]]** **>=** **++**m2**[**s**[**j**]])** **{**

cnt**++;**

**}**

j**++;**

**}**

**if** **(**cnt **==** m **&&** j**-**i **==** m**)** res**.**push\_back**(**i**);**

**if** **(**m1**.**count**(**s**[**i**])** **&&** m1**[**s**[**i**]]** **>=** m2**[**s**[**i**]]--)** **{**

cnt**--;**

**}**

i**++;**

**}**

**return** res**;**

**}**

**};**

### 440. K-th Smallest in Lexicographical Order

Hard

Given integers n and k, find the lexicographically k-th smallest integer in the range from 1 to n.

Note: 1 ≤ k ≤ n ≤ 109.

**Example:**

**Input:**

n: 13 k: 2

**Output:**

10

**Explanation:**

The lexicographical order is [1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9], so the second smallest number is 10.

### 441. Arranging Coins

Easy

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

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

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

**Example 1:**

n = 5

The coins can form the following rows:

¤

¤ ¤

¤ ¤

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

**Example 2:**

n = 8

The coins can form the following rows:

¤

¤ ¤

¤ ¤ ¤

¤ ¤

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

class Solution **{**

public**:**

int arrangeCoins**(**int n**)** **{**

**return** floor**(-**0.5**+**sqrt**((**double**)**2**\***n**+**0.25**));**

**}**

**};**

### 442. Find All Duplicates in an Array

Medium

Given an array of integers, 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear **twice** and others appear **once**.

Find all the elements that appear **twice** in this array.

Could you do it without extra space and in O(*n*) runtime?

**Example:**

**Input:**

[4,3,2,7,8,2,3,1]

**Output:**

[2,3]

class Solution **{**

public**:**

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

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

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

**while** **(**i **!=** nums**[**i**]-**1 **&&** nums**[**i**]** **!=** nums**[**nums**[**i**]-**1**])** **{**

swap**(**nums**[**i**],** nums**[**nums**[**i**]-**1**]);**

**}**

**}**

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

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

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

res**.**push\_back**(**nums**[**i**]);**

**}**

**}**

**return** res**;**

**}**

**};**

### 443. String Compression

Easy

Given an array of characters, compress it [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm).

The length after compression must always be smaller than or equal to the original array.

Every element of the array should be a **character** (not int) of length 1.

After you are done **modifying the input array** [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), return the new length of the array.

**Follow up:**  
Could you solve it using only O(1) extra space?

**Example 1:**

**Input:**

["a","a","b","b","c","c","c"]

**Output:**

Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

**Explanation:**

"aa" is replaced by "a2". "bb" is replaced by "b2". "ccc" is replaced by "c3".

**Example 2:**

**Input:**

["a"]

**Output:**

Return 1, and the first 1 characters of the input array should be: ["a"]

**Explanation:**

Nothing is replaced.

**Example 3:**

**Input:**

["a","b","b","b","b","b","b","b","b","b","b","b","b"]

**Output:**

Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].

**Explanation:**

Since the character "a" does not repeat, it is not compressed. "bbbbbbbbbbbb" is replaced by "b12".

Notice each digit has it's own entry in the array.

**Note:**

1. All characters have an ASCII value in [35, 126].
2. 1 <= len(chars) <= 1000.

class Solution **{**

public**:**

int compress**(**vector**<**char**>** **&**chars**)** **{**

chars**.**push\_back**(**'&'**);**

char pre **=** chars**[**0**];**

int cnt **=** 1**,** res **=** 0**,** j **=** 0**,** n **=** chars**.**size**();**

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

**if** **(**chars**[**i**]** **==** pre**)** cnt**++;**

**else** **{**

string s **=** to\_string**(**cnt**);**

chars**[**j**++]** **=** pre**;**

res **+=** 1 **+** **(**cnt **==** 1 **?** 0 **:** s**.**length**());**

**if** **(**cnt **!=** 1**)** **{**

**for** **(**auto **&**c **:** s**)** chars**[**j**++]** **=** c**;**

**}**

pre **=** chars**[**i**];**

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

**}**

**}**

**return** res**;**

**}**

**};**

### 445. Add Two Numbers II

Medium

You are given two **non-empty** linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Follow up:**  
What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

**Example:**

**Input:** (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

**Output:** 7 -> 8 -> 0 -> 7

class Solution **{**

public**:**

ListNode**\*** addTwoNumbers**(**ListNode**\*** l1**,** ListNode**\*** l2**)** **{**

int n1 **=** count\_len**(**l1**),** n2 **=** count\_len**(**l2**),** diff **=** abs**(**n1 **-** n2**);**

**if** **(**n1 **<** n2**)** swap**(**l1**,** l2**);**

ListNode **\***dummy **=** **new** ListNode**(**0**),** **\***cur **=** dummy**,** **\***right **=** cur**;**

**while** **(**diff**--** **>** 0**)** **{**

cur**->**next **=** **new** ListNode**(**l1**->**val**);**

**if** **(**l1**->**val **!=** 9**)** right **=** cur**->**next**;**

cur **=** cur**->**next**;**

l1 **=** l1**->**next**;**

**}**

**while** **(**l1**)** **{**

int val **=** l1**->**val **+** l2**->**val**;**

**if** **(**val **>** 9**)** **{**

val **%=** 10**;**

**++**right**->**val**;**

**while** **(**right**->**next**)** **{**

right**->**next**->**val **=** 0**;**

right **=** right**->**next**;**

**}**

right **=** cur**;**

**}**

cur**->**next **=** **new** ListNode**(**val**);**

**if** **(**val **!=** 9**)** right **=** cur**->**next**;**

cur **=** cur**->**next**;**

l1 **=** l1**->**next**;**

l2 **=** l2**->**next**;**

**}**

**return** **(**dummy**->**val **==** 1**)** **?** dummy **:** dummy**->**next**;**

**}**

private**:**

int count\_len**(**ListNode **\***l**)** **{**

int len **=** 0**;**

**while** **(**l**)** **{**

**++**len**;**

l **=** l**->**next**;**

**}**

**return** len**;**

**}**

**};**

### 446. Arithmetic Slices II - Subsequence

Hard

A sequence of numbers is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequences:

1, 3, 5, 7, 9

7, 7, 7, 7

3, -1, -5, -9

The following sequence is not arithmetic.

1, 1, 2, 5, 7

A zero-indexed array A consisting of N numbers is given. A **subsequence** slice of that array is any sequence of integers (P0, P1, ..., Pk) such that 0 ≤ P0 < P1 < ... < Pk < N.

A **subsequence** slice (P0, P1, ..., Pk) of array A is called arithmetic if the sequence A[P0], A[P1], ..., A[Pk-1], A[Pk] is arithmetic. In particular, this means that k ≥ 2.

The function should return the number of arithmetic subsequence slices in the array A.

The input contains N integers. Every integer is in the range of -231 and 231-1 and 0 ≤ N ≤ 1000. The output is guaranteed to be less than 231-1.

**Example:**

**Input:** [2, 4, 6, 8, 10]

**Output:** 7

**Explanation:**

All arithmetic subsequence slices are:

[2,4,6]

[4,6,8]

[6,8,10]

[2,4,6,8]

[4,6,8,10]

[2,4,6,8,10]

[2,6,10]

class Solution **{**

public**:**

int numberOfArithmeticSlices**(**vector**<**int**>&** A**)** **{**

**if** **(**A**.**size**()** **<** 3**)** **return** 0**;**

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

vector**<**unordered\_map**<**long long**,** int**>>** dp**(**n**);**

/\*unordered\_set<int> s(A.begin(), A.end());\*/

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

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

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

auto delta **=** **(**long long**)**A**[**i**]** **-** **(**long long**)**A**[**j**];**

int tmp **=** dp**[**j**].**count**(**delta**)** **?** dp**[**j**][**delta**]** **:** 0**;**

res **+=** tmp**;**

/\*if (s.count(A[i]+delta))\*/

dp**[**i**][**delta**]** **+=** 1**+**tmp**;**

**}**

**}**

**return** res**;**

**}**

**};**

### 447. Number of Boomerangs

Easy

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

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

**Example:**

**Input:**

[[0,0],[1,0],[2,0]]

**Output:**

2

**Explanation:**

The two boomerangs are **[[1,0],[0,0],[2,0]]** and **[[1,0],[2,0],[0,0]]**

class Solution **{**

public**:**

int numberOfBoomerangs**(**vector**<**vector**<**int**>>&** points**)** **{**

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

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

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

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

int a **=** points**[**i**][**0**]** **-** points**[**j**][**0**];**

int b **=** points**[**i**][**1**]** **-** points**[**j**][**1**];**

**++**m**[**a**\***a**+**b**\***b**];**

**}**

**for** **(**auto **&**pii **:** m**)** **{**

res **+=** pii**.**second **\*** **(**pii**.**second **-** 1**);**

**}**

**}**

**return** res**;**

**}**

**};**

### 448. Find All Numbers Disappeared in an Array

Easy

Given an array of integers where 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear twice and others appear once.

Find all the elements of [1, *n*] inclusive that do not appear in this array.

Could you do it without extra space and in O(*n*) runtime? You may assume the returned list does not count as extra space.

**Example:**

**Input:**

[4,3,2,7,8,2,3,1]

**Output:**

[5,6]

class Solution **{**

public**:**

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

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

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

**while** **(**i **!=** nums**[**i**]-**1 **&&** nums**[**i**]** **!=** nums**[**nums**[**i**]-**1**])** **{**

swap**(**nums**[**i**],** nums**[**nums**[**i**]-**1**]);**

**}**

**}**

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

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

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

res**.**push\_back**(**i**+**1**);**

**}**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

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

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

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

int m **=** abs**(**nums**[**i**])-**1**;** // index start from 0

nums**[**m**]** **=** nums**[**m**]** **>** 0 **?** **-**nums**[**m**]** **:** nums**[**m**];**

**}**

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

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

**if** **(**nums**[**i**]** **>** 0**)** res**.**push\_back**(**i**+**1**);**

**}**

**return** res**;**

**}**

**};**

### 449. Serialize and Deserialize BST

Medium

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 search tree**. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary search tree can be serialized to a string and this string can be deserialized to the original tree structure.

**The encoded string should be as compact as possible.**

**Note:** Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

### 450. Delete Node in a BST

Medium

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

**Note:** Time complexity should be O(height of tree).

**Example:**

root = [5,3,6,2,4,null,7]

key = 3

5

/ \

3 6

/ \ \

2 4 7

Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the following BST.

5

/ \

4 6

/ \

2 7

Another valid answer is [5,2,6,null,4,null,7].

5

/ \

2 6

\ \

4 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**:**

TreeNode **\***deleteNode**(**TreeNode**\*** root**,** int key**)** **{**

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

**if** **(**root**->**val **==** key**)** **{**

TreeNode **\***t **=** root**->**right**;**

**if** **(**t **==** **nullptr)** **{**

**return** root**->**left**;**

**}**

**while** **(**t**->**left**)** t **=** t**->**left**;**

root**->**val **=** t**->**val**;**

root**->**right **=** deleteNode**(**root**->**right**,** t**->**val**);**

**}**

**else** **if** **(**root**->**val **>** key **&&** root**->**left**)**

root**->**left **=** deleteNode**(**root**->**left**,** key**);**

**else** **if** **(**root**->**val **<** key **&&** root**->**right**)**

root**->**right **=** deleteNode**(**root**->**right**,** key**);**

**return** root**;**

**}**

**};**