目录

[551. Student Attendance Record I 2](#_Toc25654294)

[552. Student Attendance Record II 3](#_Toc25654295)

[553. Optimal Division 4](#_Toc25654296)

[554. Brick Wall 5](#_Toc25654297)

[556. Next Greater Element III 7](#_Toc25654298)

[558. Quad Tree Intersection 8](#_Toc25654299)

[559. Maximum Depth of N-ary Tree 11](#_Toc25654300)

[560. Subarray Sum Equals K 13](#_Toc25654301)

[561. Array Partition I 14](#_Toc25654302)

[563. Binary Tree Tilt 15](#_Toc25654303)

[564. Find the Closest Palindrome 16](#_Toc25654304)

[565. Array Nesting 17](#_Toc25654305)

[566. Reshape the Matrix 18](#_Toc25654306)

[567. Permutation in String 20](#_Toc25654307)

[572. Subtree of Another Tree 21](#_Toc25654308)

[575. Distribute Candies 23](#_Toc25654309)

[576. Out of Boundary Paths 24](#_Toc25654310)

[581. Shortest Unsorted Continuous Subarray 26](#_Toc25654311)

[583. Delete Operation for Two Strings 27](#_Toc25654312)

[587. Erect the Fence 28](#_Toc25654313)

[589. N-ary Tree Preorder Traversal 30](#_Toc25654314)

[590. N-ary Tree Postorder Traversal 32](#_Toc25654315)

[591. Tag Validator 34](#_Toc25654316)

[592. Fraction Addition and Subtraction 37](#_Toc25654317)

[593. Valid Square 38](#_Toc25654318)

[594. Longest Harmonious Subsequence 39](#_Toc25654319)

[595. (SOL) 40](#_Toc25654320)

[596. (SOL) 41](#_Toc25654321)

[598. Range Addition II 42](#_Toc25654322)

[599. Minimum Index Sum of Two Lists 44](#_Toc25654323)

[600. Non-negative Integers without Consecutive Ones 45](#_Toc25654324)

### 551. Student Attendance Record I

Easy

You are given a string representing an attendance record for a student. The record only contains the following three characters:

1. **'A'** : Absent.
2. **'L'** : Late.
3. **'P'** : Present.

A student could be rewarded if his attendance record doesn't contain **more than one 'A' (absent)** or **more than two continuous 'L' (late)**.

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

**Example 1:**

**Input:** "PPALLP"

**Output:** True

**Example 2:**

**Input:** "PPALLL"

**Output:** False

class Solution **{**

public**:**

bool checkRecord**(**string s**)** **{**

int L **=** 0**,** A **=** 0**,** L\_cnt **=** 0**;**

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

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

**case** 'A'**:** A**++;** L\_cnt **=** 0**;** **break;**

**case** 'L'**:** L **=** max**(**L**,** **++**L\_cnt**);** **break;**

**case** 'P'**:** L\_cnt **=** 0**;** **break;**

**}**

**}**

**return** L **<** 3 **&&** A **<** 2**;**

**}**

**};**

### 552. Student Attendance Record II

Hard

Given a positive integer **n**, return the number of all possible attendance records with length n, which will be regarded as rewardable. The answer may be very large, return it after mod 109 + 7.

A student attendance record is a string that only contains the following three characters:

1. **'A'** : Absent.
2. **'L'** : Late.
3. **'P'** : Present.

A record is regarded as rewardable if it doesn't contain **more than one 'A' (absent)** or **more than two continuous 'L' (late)**.

**Example 1:**

**Input:** n = 2

**Output:** 8

**Explanation:**

There are 8 records with length 2 will be regarded as rewardable:

"PP" , "AP", "PA", "LP", "PL", "AL", "LA", "LL"

Only "AA" won't be regarded as rewardable owing to more than one absent times.

**Note:** The value of **n** won't exceed 100,000.

### 553. Optimal Division

Medium

Given a list of **positive integers**, the adjacent integers will perform the float division. For example, [2,3,4] -> 2 / 3 / 4.

However, you can add any number of parenthesis at any position to change the priority of operations. You should find out how to add parenthesis to get the **maximum** result, and return the corresponding expression in string format. **Your expression should NOT contain redundant parenthesis.**

**Example:**

**Input:** [1000,100,10,2]

**Output:** "1000/(100/10/2)"

**Explanation:**

1000/(100/10/2) = 1000/((100/10)/2) = 200

However, the bold parenthesis in "1000/(**(**100/10**)**/2)" are redundant,   
since they don't influence the operation priority. So you should return "1000/(100/10/2)".

Other cases:

1000/(100/10)/2 = 50

1000/(100/(10/2)) = 50

1000/100/10/2 = 0.5

1000/100/(10/2) = 2

**Note:**

1. The length of the input array is [1, 10].
2. Elements in the given array will be in range [2, 1000].
3. There is only one optimal division for each test case.

### 554. Brick Wall

Medium

There is a brick wall in front of you. The wall is rectangular and has several rows of bricks. The bricks have the same height but different width. You want to draw a vertical line from the **top** to the **bottom** and cross the **least** bricks.

The brick wall is represented by a list of rows. Each row is a list of integers representing the width of each brick in this row from left to right.

If your line go through the edge of a brick, then the brick is not considered as crossed. You need to find out how to draw the line to cross the least bricks and return the number of crossed bricks.

**You cannot draw a line just along one of the two vertical edges of the wall, in which case the line will obviously cross no bricks.**

**Example:**

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

[3,1,2],

[1,3,2],

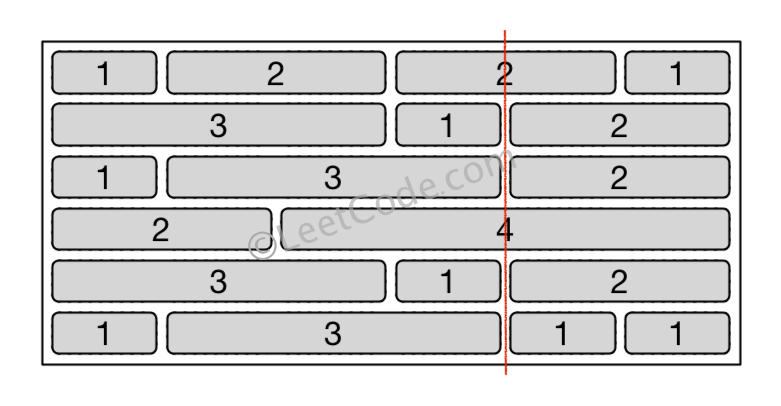
[2,4],

[3,1,2],

[1,3,1,1]]

**Output:** 2

**Explanation:**



**Note:**

1. The width sum of bricks in different rows are the same and won't exceed INT\_MAX.
2. The number of bricks in each row is in range [1,10,000]. The height of wall is in range [1,10,000]. Total number of bricks of the wall won't exceed 20,000.

class Solution **{**

public**:**

int leastBricks**(**vector**<**vector**<**int**>>** **&**wall**)** **{**

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

int res **=** 0**,** sz **=** wall**.**size**();**

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

**for** **(**int j **=** 0**,** sum **=** 0**;** j **<** wall**[**i**].**size**()** **-** 1**;** j**++)** **{**

res **=** max**(**res**,** **++**m**[**sum **+=** wall**[**i**][**j**]]);**

**}**

**}**

**return** sz **-** res**;**

**}**

**};**

### 556. Next Greater Element III

Medium

Given a positive **32-bit** integer **n**, you need to find the smallest **32-bit** integer which has exactly the same digits existing in the integer **n** and is greater in value than n. If no such positive **32-bit** integer exists, you need to return -1.

**Example 1:**

**Input:** 12

**Output:** 21

**Example 2:**

**Input:** 21

**Output:** -1

class Solution **{**

public**:**

int nextGreaterElement**(**int n**)** **{**

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

**if** **(!**next\_permutation**(**s**.**begin**(),** s**.**end**()))** **return** **-**1**;**

long long res **=** stoll**(**s**);**

**return** res **>** INT\_MAX **?** **-**1 **:** res**;**

**}**

**};**

### 558. Quad Tree Intersection

Easy

A quadtree is a tree data in which each internal node has exactly four children: topLeft, topRight, bottomLeft and bottomRight. Quad trees are often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions.

We want to store True/False information in our quad tree. The quad tree is used to represent a N \* N boolean 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.

For example, below are two quad trees A and B:

A:

+---------+----------+ T: true

| | | F: false

| T | T |

| | |

+-------+-----------+

| | |

| F | F |

| | |

+-------+-----------+

topLeft: T

topRight: T

bottomLeft: F

bottomRight: F

B:

+---------+----+----+

| | F | F |

| T +-----+---+

| | T | T |

+--------+-----+-----+

| | |

| T | F |

| | |

+---------+--------+

topLeft: T

topRight:

topLeft: F

topRight: F

bottomLeft: T

bottomRight: T

bottomLeft: T

bottomRight: F

Your task is to implement a function that will take two quadtrees and return a quadtree that represents the logical OR (or union) of the two trees.

A: B: C (A or B):

+-------+-------+ +-------+---+---+ +-------+-------+

| | | | | F | F | | | |

| T | T | | T +---+---+ | T | T |

| | | | | T | T | | | |

+-------+-------+ +-------+---+---+ +-------+-------+

| | | | | | | | |

| F | F | | T | F | | T | F |

| | | | | | | | |

+-------+-------+ +-------+-------+ +-------+-------+

**Note:**

1. Both A and B represent grids of size N \* N.
2. N is guaranteed to be a power of 2.
3. If you want to know more about the quad tree, you can refer to its [wiki](https://en.wikipedia.org/wiki/Quadtree).
4. The logic OR operation is defined as this: "A or B" is true if A is true, or if B is true, or if both A and B are true.

/\*

// Definition for a QuadTree node.

class Node {

public:

bool val;

bool isLeaf;

Node\* topLeft;

Node\* topRight;

Node\* bottomLeft;

Node\* bottomRight;

Node() {}

Node(bool \_val, bool \_isLeaf, Node\* \_topLeft, Node\* \_topRight, Node\* \_bottomLeft, Node\* \_bottomRight) {

val = \_val;

isLeaf = \_isLeaf;

topLeft = \_topLeft;

topRight = \_topRight;

bottomLeft = \_bottomLeft;

bottomRight = \_bottomRight;

}

};

\*/

class Solution **{**

public**:**

Node**\*** intersect**(**Node**\*** T1**,** Node**\*** T2**)** **{**

Node **\***T **=** **new** Node**(false,** **false,** **nullptr,** **nullptr,** **nullptr,** **nullptr);**

**if** **(**T1**->**isLeaf **&&** T2**->**isLeaf**)** **{**

T**->**isLeaf **=** **true;**

T**->**val **=** **(**T1**->**val **||** T2**->**val**);**

**}**

**else** **if** **(**T1**->**isLeaf **||** T2**->**isLeaf**)** **{**

**if** **(!**T1**->**isLeaf**)** swap**(**T1**,** T2**);**

**if** **(**T1**->**val**)** T**->**isLeaf **=** T**->**val **=** **true;**

**else** T **=** T2**;**

**}**

**else** **{**

T**->**topLeft **=** intersect**(**T1**->**topLeft**,** T2**->**topLeft**);**

T**->**topRight **=** intersect**(**T1**->**topRight**,** T2**->**topRight**);**

T**->**bottomLeft **=** intersect**(**T1**->**bottomLeft**,** T2**->**bottomLeft**);**

T**->**bottomRight **=**intersect**(**T1**->**bottomRight**,**T2**->**bottomRight**);**

**}**

**if** **(!**T**->**isLeaf **&&** T**->**topLeft**->**val **&&** T**->**topRight**->**val

**&&** T**->**bottomLeft**->**val **&&** T**->**bottomRight**->**val**)** **{**

T**->**isLeaf **=** T**->**val **=** **true;**

**}**

**return** T**;**

**}**

**};**

### 559. Maximum Depth of N-ary Tree

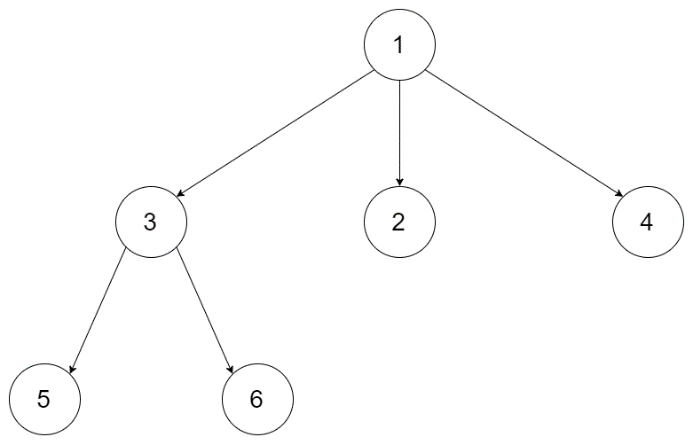
Easy

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

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

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

**Example 1:**



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

**Output:** 3

**Example 2:**



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

**Output:** 5

**Constraints:**

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

/\*

// 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**:**

int maxDepth**(**Node**\*** root**)** **{**

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

int depth **=** 0**;**

**for** **(**auto child **:** root**->**children**)** d

epth **=** max**(**depth**,** maxDepth**(**child**));**

**return** 1 **+** depth**;**

**}**

**};**

### 560. Subarray Sum Equals K

Medium

Given an array of integers and an integer **k**, you need to find the total number of continuous subarrays whose sum equals to **k**.

**Example 1:**

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

**Output:** 2

**Note:**

1. The length of the array is in range [1, 20,000].
2. The range of numbers in the array is [-1000, 1000] and the range of the integer **k** is [-1e7, 1e7].

### 561. Array Partition I

Easy

Given an array of **2n** integers, your task is to group these integers into **n** pairs of integer, say (a1, b1), (a2, b2), ..., (an, bn) which makes sum of min(ai, bi) for all i from 1 to n as large as possible.

**Example 1:**

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

**Output:** 4

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

**Note:**

1. **n** is a positive integer, which is in the range of [1, 10000].
2. All the integers in the array will be in the range of [-10000, 10000].

### 563. Binary Tree Tilt

Easy

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

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

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

**Example:**

**Input:**

1

/ \

2 3

**Output:** 1

**Explanation:**

Tilt of node 2 : 0

Tilt of node 3 : 0

Tilt of node 1 : |2-3| = 1

Tilt of binary tree : 0 + 0 + 1 = 1

**Note:**

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

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

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

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

**return** res**;**

**}**

private**:**

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

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

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

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

res **+=** abs**(**left**-**right**);**

**return** left **+** right **+** root**->**val**;**

**}**

**};**

### 564. Find the Closest Palindrome

Hard

Given an integer n, find the closest integer (not including itself), which is a palindrome.

The 'closest' is defined as absolute difference minimized between two integers.

**Example 1:**

**Input:** "123"

**Output:** "121"

**Note:**

1. The input **n** is a positive integer represented by string, whose length will not exceed 18.
2. If there is a tie, return the smaller one as answer.

### 565. Array Nesting

Medium

A zero-indexed array A of length N contains all integers from 0 to N-1. Find and return the longest length of set S, where S[i] = {A[i], A[A[i]], A[A[A[i]]], ... } subjected to the rule below.

Suppose the first element in S starts with the selection of element A[i] of index = i, the next element in S should be A[A[i]], and then A[A[A[i]]]… By that analogy, we stop adding right before a duplicate element occurs in S.

**Example 1:**

**Input:** A = [5,4,0,3,1,6,2]

**Output:** 4

**Explanation:**

A[0] = 5, A[1] = 4, A[2] = 0, A[3] = 3, A[4] = 1, A[5] = 6, A[6] = 2.

One of the longest S[K]:

S[0] = {A[0], A[5], A[6], A[2]} = {5, 6, 2, 0}

**Note:**

1. N is an integer within the range [1, 20,000].
2. The elements of A are all distinct.
3. Each element of A is an integer within the range [0, N-1].

class Solution **{**

public**:**

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

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

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **if** **(**nums**[**i**]** **>=** 0**)** **{**

int j **=** i**;**

**while** **(**nums**[**j**]** **>=** 0**)** **{**

int t **=** nums**[**j**];**

nums**[**j**]** **=** **-**1**;**

j **=** t**;**

cnt**++;**

**}**

res **=** max**(**res**,** cnt**);**

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

**}**

**return** res**;**

**}**

**};**

### 566. Reshape the Matrix

Easy

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

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

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

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

**Example 1:**

**Input:**

nums =

[[1,2],

[3,4]]

r = 1, c = 4

**Output:**

[[1,2,3,4]]

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

**Example 2:**

**Input:**

nums =

[[1,2],

[3,4]]

r = 2, c = 4

**Output:**

[[1,2],

[3,4]]

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

**Note:**

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

class Solution **{**

public**:**

vector**<**vector**<**int**>>** matrixReshape**(**vector**<**vector**<**int**>>&** nums**,** int r**,** int c**)** **{**

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

**if** **(**n**\***m **!=** r**\***c**)** **return** nums**;**

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

vector**<**int**>** temp**;**

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

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

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

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

**if** **(++**cnt **%** c **==** 0**)** **{**

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

temp**.**clear**();**

**}**

**}**

**}**

**return** res**;**

**}**

**};**

### 567. Permutation in String

Medium

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

**Example 1:**

**Input:** s1 = "ab" s2 = "eidbaooo"

**Output:** True

**Explanation:** s2 contains one permutation of s1 ("ba").

**Example 2:**

**Input:**s1= "ab" s2 = "eidboaoo"

**Output:** False

**Note:**

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

class Solution **{**

public**:**

bool checkInclusion**(**string s1**,** string s2**)** **{**

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

**for(**auto **&**c **:** s1**)** m1**[**c**]++;**

int cnt **=** 0**,** left **=** 0**;**

int len1 **=** s1**.**length**(),** len2 **=** s2**.**length**();**

**for(**int i **=** 0**;** i **<** len2**;** i**++)** **{**

char c **=** s2**[**i**];**

**if** **(**m1**.**count**(**c**))** **{**

**if** **(++**m2**[**c**]** **>** m1**[**c**])** **{**

**while** **(**s2**[**left**]** **!=** c**)** **{**

m2**[**s2**[**left**++]]--;**

cnt**--;**

**}**

m2**[**s2**[**left**++]]--;**

**}**

**else** **if** **(++**cnt **==** len1**)** **return** **true;**

**}**

**else** **{**

left **=** i**+**1**;**

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

m2**.**clear**();**

**}**

**}**

**return** **false;**

**}**

**};**

### 572. Subtree of Another Tree

Easy

Given two non-empty binary trees **s** and **t**, check whether tree **t** has exactly the same structure and node values with a subtree of **s**. A subtree of **s** is a tree consists of a node in **s** and all of this node's descendants. The tree **s** could also be considered as a subtree of itself.

**Example 1:**  
Given tree s:

3

/ \

4 5

/ \

1 2

Given tree t:

4

/ \

1 2

Return **true**, because t has the same structure and node values with a subtree of s.

**Example 2:**  
Given tree s:

3

/ \

4 5

/ \

1 2

/

0

Given tree t:

4

/ \

1 2

Return **false**.

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

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

\* };

\*/

class Solution **{**

public**:**

bool isSubtree**(**TreeNode**\*** s**,** TreeNode**\*** t**)** **{**

string str **=** treeTostring**(**t**);**

string treestr **=** ""**;**

**return** findstr**(**s**,** treestr**,** str**);**

**}**

private**:**

string treeTostring**(**TreeNode **\***t**)** **{**

**if** **(!**t**)** **return** ""**;**

string l **=** treeTostring**(**t**->**left**);**

string r **=** treeTostring**(**t**->**right**);**

**return** l **+** to\_string**(**t**->**val**)** **+** r**;**

**}**

bool findstr**(**TreeNode **\***t**,** string **&**treestr**,** string **&**s**)** **{**

**if** **(!**t**)** **{**

treestr **=** ""**;**

**return** **false;**

**}**

string l**,** r**;**

**if** **(**findstr**(**t**->**left**,** l**,** s**)** **||** findstr**(**t**->**right**,** r**,** s**))**

**return** **true;**

treestr **=** l **+** to\_string**(**t**->**val**)** **+** r**;**

**return** treestr **==** s**;**

**}**

**};**

class Solution **{**

public**:**

bool isSubtree**(**TreeNode**\*** s**,** TreeNode**\*** t**)** **{**

**if(!**s**)** **return** **false;**

**return** isSameTree**(**s**,**t**)** **||** isSubtree**(**s**->**left**,**t**)**

**||** isSubtree**(**s**->**right**,**t**);**

**}**

private**:**

bool isSameTree**(**TreeNode**\*** p**,** TreeNode**\*** q**)** **{**

**if** **(!**p **&&** **!**q**)** **return** **true;**

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

**else** **return** isSameTree**(**p**->**left**,** q**->**left**)**

**&&** isSameTree**(**p**->**right**,** q**->**right**);**

**}**

**};**

### 575. Distribute Candies

Easy

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.

**Example 1:**

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

**Output:** 3

**Explanation:**

There are three different kinds of candies (1, 2 and 3), and two candies for each kind.

Optimal distribution: The sister has candies [1,2,3] and the brother has candies [1,2,3], too.

The sister has three different kinds of candies.

**Example 2:**

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

**Output:** 2

**Explanation:** For example, the sister has candies [2,3] and the brother has candies [1,1].

The sister has two different kinds of candies, the brother has only one kind of candies.

**Note:**

1. The length of the given array is in range [2, 10,000], and will be even.
2. The number in given array is in range [-100,000, 100,000].

class Solution **{**

public**:**

int distributeCandies**(**vector**<**int**>&** candies**)** **{**

unordered\_set**<**int**>** Myset**(**candies**.**begin**(),** candies**.**end**());**

**return** min**(**Myset**.**size**(),** candies**.**size**()/**2**);**

**}**

**};**

### 576. Out of Boundary Paths

Medium

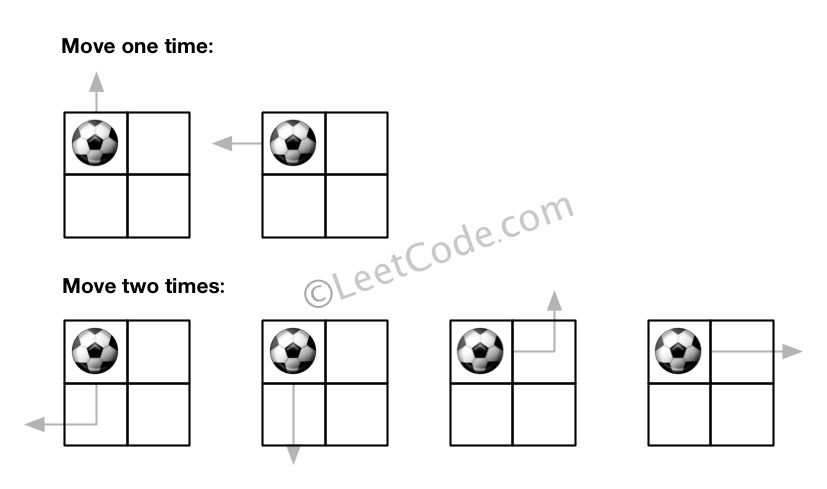
There is an **m** by **n** grid with a ball. Given the start coordinate **(i,j)** of the ball, you can move the ball to **adjacent** cell or cross the grid boundary in four directions (up, down, left, right). However, you can **at most** move **N** times. Find out the number of paths to move the ball out of grid boundary. The answer may be very large, return it after mod 109 + 7.

**Example 1:**

**Input:** m = 2, n = 2, N = 2, i = 0, j = 0

**Output:** 6

**Explanation:**

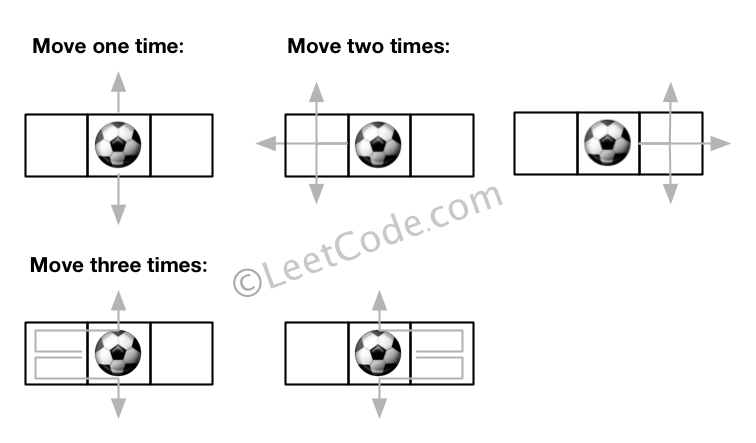


**Example 2:**

**Input:** m = 1, n = 3, N = 3, i = 0, j = 1

**Output:** 12

**Explanation:**



**Note:**

1. Once you move the ball out of boundary, you cannot move it back.
2. The length and height of the grid is in range [1,50].
3. N is in range [0,50].

class Solution **{**

public**:**

int findPaths**(**int m**,** int n**,** int N**,** int i**,** int j**)** **{**

**return** dfs**(**i**,** j**,** N**-**1**,** m**,** n**);**

**}**

private**:**

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

const int MOD **=** 1000000007**;**

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

int dfs**(**int x**,** int y**,** int z**,** int n**,** int m**)** **{**

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

auto t **=** make\_tuple**(**x**,** y**,** z**);**

**if** **(**mp**.**count**(**t**))** **return** mp**[**t**];**

long ret **=** 0**;**

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

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

**if** **(**xx **<** 0 **||** yy **<** 0 **||** xx **>=** n **||** yy **>=** m**)** ret**++;**

**else** ret **+=** dfs**(**xx**,** yy**,** z**-**1**,** n**,** m**);**

**}**

**return** mp**[**t**]** **=** **(**ret **%=** MOD**);**

**}**

**};**

### 581. Shortest Unsorted Continuous Subarray

Easy

Given an integer array, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order, too.

You need to find the **shortest** such subarray and output its length.

**Example 1:**

**Input:** [2, 6, 4, 8, 10, 9, 15]

**Output:** 5

**Explanation:** You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.

**Note:**

1. Then length of the input array is in range [1, 10,000].
2. The input array may contain duplicates, so ascending order here means **<=**.

### 583. Delete Operation for Two Strings

Medium

Given two words *word1* and *word2*, find the minimum number of steps required to make *word1* and *word2* the same, where in each step you can delete one character in either string.

**Example 1:**

**Input:** "sea", "eat"

**Output:** 2

**Explanation:** You need one step to make "sea" to "ea" and another step to make "eat" to "ea".

**Note:**

1. The length of given words won't exceed 500.
2. Characters in given words can only be lower-case letters.

class Solution **{**

public**:**

int minDistance**(**string s**,** string t**)** **{**

int n **=** s**.**size**(),** m **=** t**.**size**();**

vector**<**vector**<**int**>>** dp**(**2**,** vector**<**int**>** **(**m**+**1**,** n**+**m**));**

int p **=** 0**;**

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

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

**if** **(!**i **||** **!**j**)** dp**[**p**][**j**]** **=** i **+** j**;**

**else** **{**

dp**[**p**][**j**]** **=** dp**[**p**^**1**][**j**-**1**]** **+(**s**[**i**-**1**]** **==** t**[**j**-**1**]** **?** 0 **:** 2**);**

dp**[**p**][**j**]** **=** min**(**dp**[**p**][**j**],**

1 **+** min**(**dp**[**p**^**1**][**j**],** dp**[**p**][**j**-**1**]));**

**}**

**}**

p **^=** 1**;**

**}**

**return** dp**[**p**^**1**][**m**];**

**}**

**};**

### 587. Erect the Fence

Hard

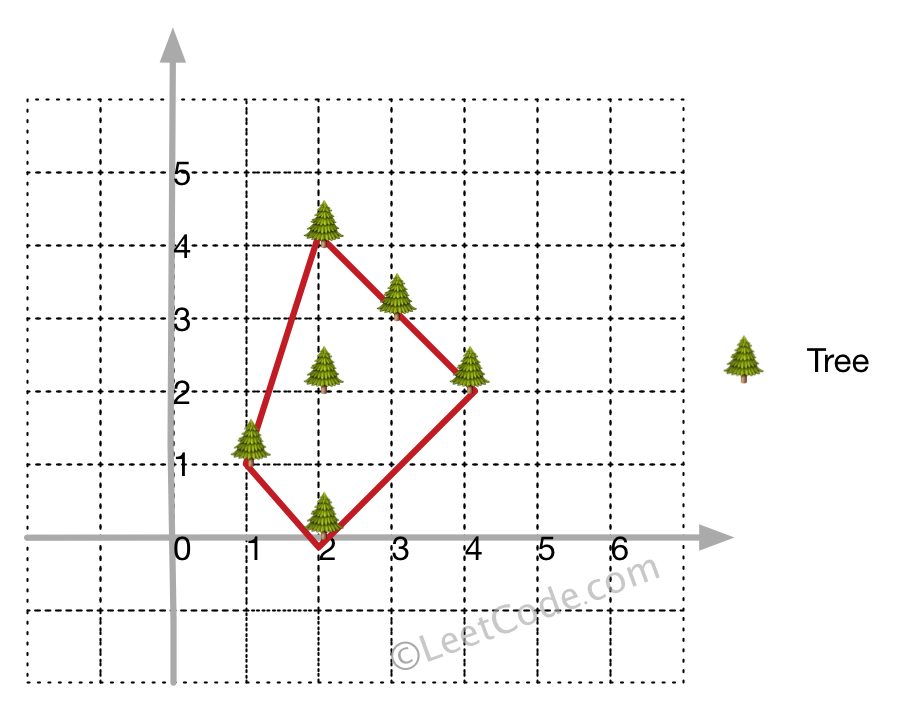
There are some trees, where each tree is represented by (x,y) coordinate in a two-dimensional garden. Your job is to fence the entire garden using the **minimum length** of rope as it is expensive. The garden is well fenced only if all the trees are enclosed. Your task is to help find the coordinates of trees which are exactly located on the fence perimeter.

**Example 1:**

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

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

**Explanation:**

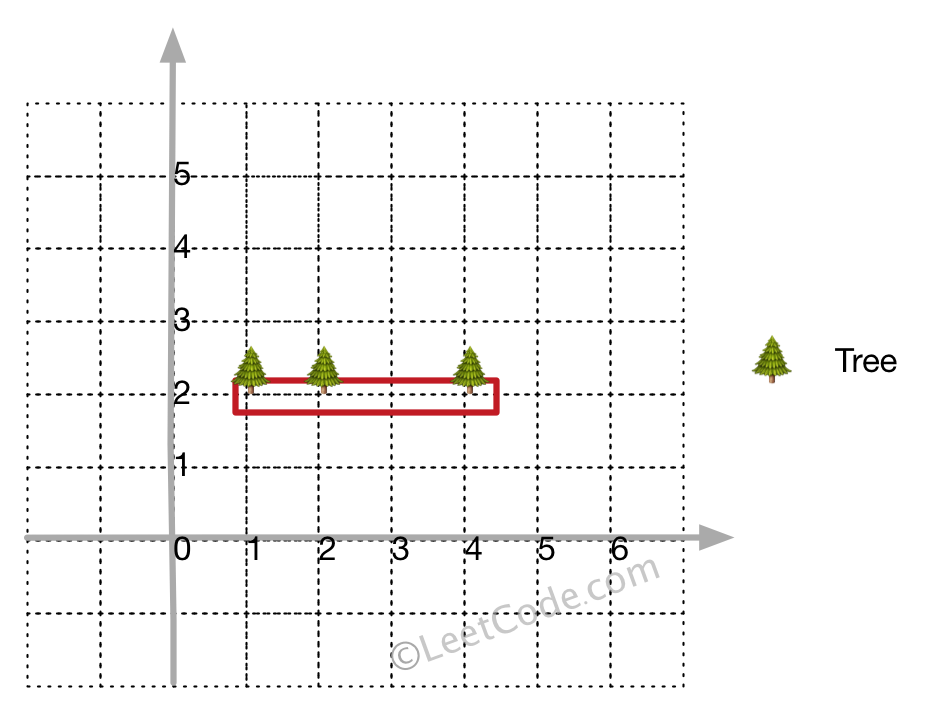


**Example 2:**

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

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

**Explanation:**



Even you only have trees in a line, you need to use rope to enclose them.

**Note:**

1. All trees should be enclosed together. You cannot cut the rope to enclose trees that will separate them in more than one group.
2. All input integers will range from 0 to 100.
3. The garden has at least one tree.
4. All coordinates are distinct.
5. Input points have **NO** order. No order required for output.
6. input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

class Solution **{**

public**:**

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

**if** **(**points**.**size**()** **<** 4**)** **return** points**;**

set**<**vector**<**int**>>** myset**;**

int left\_most **=** 0**,** n **=** points**.**size**();**

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

**if** **(**points**[**i**][**0**]** **<** points**[**left\_most**][**0**])** **{**

left\_most **=** i**;**

**}**

**}**

int p **=** left\_most**;**

**do** **{**

int q **=** **(**p**+**1**)** **%** n**;** //只要不是p点皆可， 如(p+2) % n

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

**if** **(**orientation**(**points**[**p**],** points**[**i**],** points**[**q**])** **<** 0**)** **{**

q **=** i**;**

**}**

**}**

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

**if** **(**i **!=** p **&&** i **!=** q

**&&** orientation**(**points**[**p**],** points**[**i**],** points**[**q**])** **==** 0 **&&** inBetween**(**points**[**p**],** points**[**i**],** points**[**q**]))** **{**

myset**.**insert**(**points**[**i**]);**

**}**

**}**

myset**.**insert**(**points**[**q**]);**

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

**}** **while** **(**p **!=** left\_most**);**

**return** vector**<**vector**<**int**>>** **(**myset**.**begin**(),** myset**.**end**());**

**}**

private**:**

int orientation**(**vector**<**int**>** **&**p**,** vector**<**int**>** **&**q**,** vector**<**int**>** **&**r**)** **{**

**return** **(**q**[**1**]** **-** p**[**1**])** **\*** **(**r**[**0**]** **-** q**[**0**])** **-** **(**q**[**0**]** **-** p**[**0**])** **\*** **(**r**[**1**]** **-** q**[**1**]);**

**}**

bool inBetween**(**vector**<**int**>** **&**p**,** vector**<**int**>** **&**i**,** vector**<**int**>** **&**q**)** **{**

bool a **=** i**[**0**]** **>=** p**[**0**]** **&&** i**[**0**]** **<=** q**[**0**]** **||** i**[**0**]** **<=** p**[**0**]**

**&&** i**[**0**]** **>=** q**[**0**];**

bool b **=** i**[**1**]** **>=** p**[**1**]** **&&** i**[**1**]** **<=** q**[**1**]** **||** i**[**1**]** **<=** p**[**1**]**

**&&** i**[**1**]** **>=** q**[**1**];**

**return** a **&&** b**;**

**}**

**};**

### 589. N-ary Tree Preorder Traversal

Easy

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

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

**Follow up:**

Recursive solution is trivial, could you do it iteratively?

**Example 1:**



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

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

**Example 2:**



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

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

**Constraints:**

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

/\*

// 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**<**int**>** preorder**(**Node**\*** root**)** **{**

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

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

stack**<**Node**\*>** stk**;**

stk**.**push**(**root**);**

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

auto t **=** stk**.**top**();**

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

res**.**push\_back**(**t**->**val**);**

**for** **(**int i **=** t**->**children**.**size**()-**1**;** i **>=** 0**;** **--**i**)** **{**

stk**.**push**(**t**->**children**[**i**]);**

**}**

**}**

**return** res**;**

**}**

**};**

### 590. N-ary Tree Postorder Traversal

Easy

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

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

**Follow up:**

Recursive solution is trivial, could you do it iteratively?

**Example 1:**



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

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

**Example 2:**



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

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

**Constraints:**

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

### 591. Tag Validator

Hard

Given a string representing a code snippet, you need to implement a tag validator to parse the code and return whether it is valid. A code snippet is valid if all the following rules hold:

1. The code must be wrapped in a **valid closed tag**. Otherwise, the code is invalid.
2. A **closed tag** (not necessarily valid) has exactly the following format : <TAG\_NAME>TAG\_CONTENT</TAG\_NAME>. Among them, <TAG\_NAME> is the start tag, and </TAG\_NAME> is the end tag. The TAG\_NAME in start and end tags should be the same. A closed tag is **valid** if and only if the TAG\_NAME and TAG\_CONTENT are valid.
3. A **valid** TAG\_NAME only contain **upper-case letters**, and has length in range [1,9]. Otherwise, the TAG\_NAME is **invalid**.
4. A **valid** TAG\_CONTENT may contain other **valid closed tags**, **cdata** and any characters (see note1) **EXCEPT** unmatched <, unmatched start and end tag, and unmatched or closed tags with invalid TAG\_NAME. Otherwise, the TAG\_CONTENT is **invalid**.
5. A start tag is unmatched if no end tag exists with the same TAG\_NAME, and vice versa. However, you also need to consider the issue of unbalanced when tags are nested.
6. A < is unmatched if you cannot find a subsequent >. And when you find a < or </, all the subsequent characters until the next > should be parsed as TAG\_NAME (not necessarily valid).
7. The cdata has the following format : <![CDATA[CDATA\_CONTENT]]>. The range of CDATA\_CONTENT is defined as the characters between <![CDATA[ and the **first subsequent** ]]>.
8. CDATA\_CONTENT may contain **any characters**. The function of cdata is to forbid the validator to parse CDATA\_CONTENT, so even it has some characters that can be parsed as tag (no matter valid or invalid), you should treat it as **regular characters**.

**Valid Code Examples:**

**Input:** "<DIV>This is the first line <![CDATA[<div>]]></DIV>"

**Output:** True

**Explanation:**

The code is wrapped in a closed tag : <DIV> and </DIV>.

The TAG\_NAME is valid, the TAG\_CONTENT consists of some characters and cdata.

Although CDATA\_CONTENT has unmatched start tag with invalid TAG\_NAME, it should be considered as plain text, not parsed as tag.

So TAG\_CONTENT is valid, and then the code is valid. Thus return true.

**Input:** "<DIV>>> ![cdata[]] <![CDATA[<div>]>]]>]]>>]</DIV>"

**Output:** True

**Explanation:**

We first separate the code into : start\_tag|tag\_content|end\_tag.

start\_tag -> **"<DIV>"**

end\_tag -> **"</DIV>"**

tag\_content could also be separated into : text1|cdata|text2.

text1 -> **">> ![cdata[]] "**

cdata -> **"<![CDATA[<div>]>]]>"**, where the CDATA\_CONTENT is **"<div>]>"**

text2 -> **"]]>>]"**

The reason why start\_tag is NOT **"<DIV>>>"** is because of the rule 6.

The reason why cdata is NOT **"<![CDATA[<div>]>]]>]]>"** is because of the rule 7.

**Invalid Code Examples:**

**Input:** "<A> <B> </A> </B>"

**Output:** False

**Explanation:** Unbalanced. If "<A>" is closed, then "<B>" must be unmatched, and vice versa.

**Input:** "<DIV> div tag is not closed <DIV>"

**Output:** False

**Input:** "<DIV> unmatched < </DIV>"

**Output:** False

**Input:** "<DIV> closed tags with invalid tag name <b>123</b> </DIV>"

**Output:** False

**Input:** "<DIV> unmatched tags with invalid tag name </1234567890> and <CDATA[[]]> </DIV>"

**Output:** False

**Input:** "<DIV> unmatched start tag <B> and unmatched end tag </C> </DIV>"

**Output:** False

**Note:**

1. For simplicity, you could assume the input code (including the **any characters** mentioned above) only contain letters, digits, '<','>','/','!','[',']' and ' '.

### 592. Fraction Addition and Subtraction

Medium

Given a string representing an expression of fraction addition and subtraction, you need to return the calculation result in string format. The final result should be [irreducible fraction](https://en.wikipedia.org/wiki/Irreducible_fraction). If your final result is an integer, say 2, you need to change it to the format of fraction that has denominator 1. So in this case, 2 should be converted to 2/1.

**Example 1:**

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

**Output:** "0/1"

**Example 2:**

**Input:**"-1/2+1/2+1/3"

**Output:** "1/3"

**Example 3:**

**Input:**"1/3-1/2"

**Output:** "-1/6"

**Example 4:**

**Input:**"5/3+1/3"

**Output:** "2/1"

**Note:**

1. The input string only contains '0' to '9', '/', '+' and '-'. So does the output.
2. Each fraction (input and output) has format ±numerator/denominator. If the first input fraction or the output is positive, then '+' will be omitted.
3. The input only contains valid **irreducible fractions**, where the **numerator** and **denominator** of each fraction will always be in the range [1,10]. If the denominator is 1, it means this fraction is actually an integer in a fraction format defined above.
4. The number of given fractions will be in the range [1,10].
5. The numerator and denominator of the **final result** are guaranteed to be valid and in the range of 32-bit int.

class Solution **{**

public**:**

string fractionAddition**(**string expression**)** **{**

istringstream in**(**expression**);**

int A **=** 0**,** B **=** 1**,** a**,** b**;**

char \_**;**

**while** **(**in **>>** a **>>** \_ **>>** b**)** **{**

A **=** A **\*** b **+** a **\*** B**;**

B **\*=** b**;**

int g **=** abs**(**\_\_gcd**(**A**,** B**));**

A **/=** g**;**

B **/=** g**;**

**}**

**return** to\_string**(**A**)** **+** '/' **+** to\_string**(**B**);**

**}**

**};**

### 593. Valid Square

Medium

Given the coordinates of four points in 2D space, return whether the four points could construct a square.

The coordinate (x,y) of a point is represented by an integer array with two integers.

**Example:**

**Input:** p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,1]

**Output:** True

Note:

1. All the input integers are in the range [-10000, 10000].
2. A valid square has four equal sides with positive length and four equal angles (90-degree angles).
3. Input points have no order.

class Solution **{**

public**:**

bool validSquare**(**vector**<**int**>&** p1**,** vector**<**int**>&** p2**,** vector**<**int**>&** p3**,** vector**<**int**>&** p4**)** **{**

vector**<**vector**<**int**>>** p**{**p1**,** p2**,** p3**,** p4**};**

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

**return** lhs**[**0**]** **==** rhs**[**0**]** **?** lhs**[**1**]** **<** rhs**[**1**]**

**:** lhs**[**0**]** **<** rhs**[**0**];**

**});**

int len0 **=** dist**(**p**[**0**],** p**[**1**]),** len1 **=** dist**(**p**[**1**],** p**[**3**]);**

**int** len2 **=** dist**(**p**[**3**],** p**[**2**]),** len3 **=** dist**(**p**[**2**],** p**[**0**]);**

**return** len0 **==** len1 **&&** len1 **==** len2 **&&** len2 **==** len3

**&&** len0 **!=** 0 **&&** dist**(**p**[**0**],** p**[**3**])** **==** dist**(**p**[**1**],** p**[**2**]);**

**}**

private**:**

double dist**(**vector**<**int**>** **&**p1**,** vector**<**int**>** **&**p2**)** **{**

**return** **(**p2**[**1**]** **-** p1**[**1**])** **\*** **(**p2**[**1**]** **-** p1**[**1**])** **+** **(**p2**[**0**]** **-** p1**[**0**])** **\*** **(**p2**[**0**]** **-** p1**[**0**]);**

**}**

**};**

### 594. Longest Harmonious Subsequence

Easy

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

Now, given an integer array, you need to find the length of its longest harmonious subsequence among all its possible [subsequences](https://en.wikipedia.org/wiki/Subsequence).

**Example 1:**

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

**Output:** 5

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

**Note:** The length of the input array will not exceed 20,000.

class Solution **{**

public**:**

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

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

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

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

**for** **(**auto **&**i **:** mp**)** **if** **(**mp**.**count**(**i**.**first**-**1**))** **{**

res **=** max**(**res**,** i**.**second **+** mp**[**i**.**first**-**1**]);**

**}**

**return** res**;**

**}**

**};**

### 595. (SOL)

### 596. (SOL)

### 598. Range Addition II

Easy

Given an m \* n matrix **M** initialized with all **0**'s and several update operations.

Operations are represented by a 2D array, and each operation is represented by an array with two **positive** integers **a** and **b**, which means **M[i][j]** should be **added by one** for all **0 <= i < a** and **0 <= j < b**.

You need to count and return the number of maximum integers in the matrix after performing all the operations.

**Example 1:**

**Input:**

m = 3, n = 3

operations = [[2,2],[3,3]]

**Output:** 4

**Explanation:**

Initially, M =

[[0, 0, 0],

[0, 0, 0],

[0, 0, 0]]

After performing [2,2], M =

[[1, 1, 0],

[1, 1, 0],

[0, 0, 0]]

After performing [3,3], M =

[[2, 2, 1],

[2, 2, 1],

[1, 1, 1]]

So the maximum integer in M is 2, and there are four of it in M. So return 4.

**Note:**

1. The range of m and n is [1,40000].
2. The range of a is [1,m], and the range of b is [1,n].
3. The range of operations size won't exceed 10,000.

class Solution **{**

public**:**

int maxCount**(**int m**,** int n**,** vector**<**vector**<**int**>>&** ops**)** **{**

**for** **(**auto **&**v **:** ops**)** **{**

m **=** min**(**m**,** v**[**0**]);**

n **=** min**(**n**,** v**[**1**]);**

**}**

**return** n**\***m**;**

**}**

**};**

### 599. Minimum Index Sum of Two Lists

Easy

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their **common interest** with the **least list index sum**. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

**Example 1:**

**Input:**

["Shogun", "Tapioca Express", "Burger King", "KFC"]

["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]

**Output:** ["Shogun"]

**Explanation:** The only restaurant they both like is "Shogun".

**Example 2:**

**Input:**

["Shogun", "Tapioca Express", "Burger King", "KFC"]

["KFC", "Shogun", "Burger King"]

**Output:** ["Shogun"]

**Explanation:** The restaurant they both like and have the least index sum is "Shogun" with index sum 1 (0+1).

**Note:**

1. The length of both lists will be in the range of [1, 1000].
2. The length of strings in both lists will be in the range of [1, 30].
3. The index is starting from 0 to the list length minus 1.
4. No duplicates in both lists.

class Solution **{**

public**:**

vector**<**string**>** findRestaurant**(**vector**<**string**>&** list1**,** vector**<**string**>&** list2**)** **{**

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

**for** **(**int i **=** 0**;** i **<** list1**.**size**();** **++**i**)** mp**[**list1**[**i**]]** **=** i**;**

int sum **=** INT\_MAX**;**

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

**for** **(**int i **=** 0**;** i **<** list2**.**size**();** **++**i**)** **{**

**if** **(**mp**.**count**(**list2**[**i**]))** **{**

int cnt **=** i **+** mp**[**list2**[**i**]];**

**if** **(**cnt **<** sum**)** **{**

res**.**clear**();**

res**.**push\_back**(**list2**[**i**]);**

sum **=** cnt**;**

**}**

**else** **if** **(**cnt **==** sum**)** res**.**push\_back**(**list2**[**i**]);**

**}**

**}**

**return** res**;**

**}**

**};**

### 600. Non-negative Integers without Consecutive Ones

Hard

Given a positive integer n, find the number of **non-negative** integers less than or equal to n, whose binary representations do NOT contain **consecutive ones**.

**Example 1:**

**Input:** 5

**Output:** 5

**Explanation:**

Here are the non-negative integers <= 5 with their corresponding binary representations:

0 : 0

1 : 1

2 : 10

3 : 11

4 : 100

5 : 101

Among them, only integer 3 disobeys the rule (two consecutive ones) and the other 5 satisfy the rule.

**Note:** 1 <= n <= 109