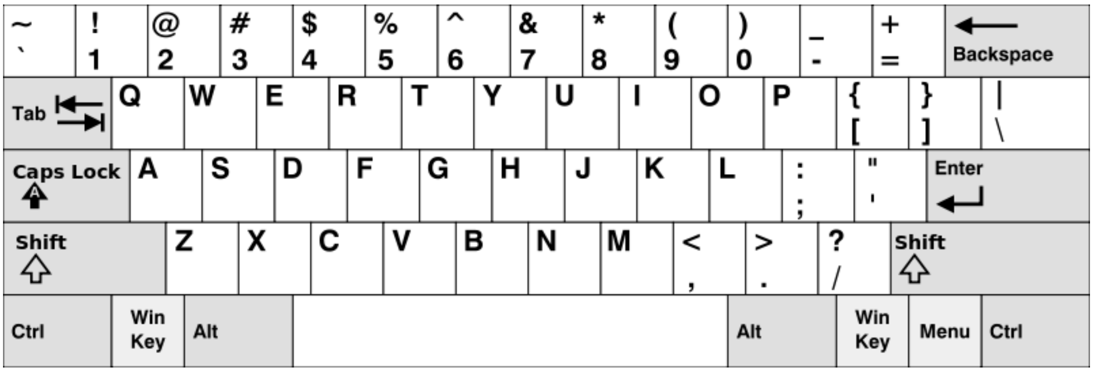
目录

500. Keyboard Row

Easy

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


~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl

Example:

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

Output: ["Alaska", "Dad"]

Note:

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

501. Find Mode in Binary Search Tree

Easy

Given a binary search tree (BST) with duplicates, find all the mode(s) (the most frequently occurred element) in the given BST.

Assume a BST is defined as follows:

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

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

```
   1
    \
     2
    /
   2
```

return `[2]`.

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

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

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> findMode(TreeNode* p) {
        int cnt = 0, maxcnt = 0, cur = 0;
        vector<int> res;
        stack<TreeNode*> stk;
        while (!stk.empty() || p) {
          while (p) {
              stk.push(p);
              p = p->left;
          }
          p = stk.top();
          if (p->val != cur) {
              cnt = 0;
              cur = p->val;
          }
          if (++cnt > maxcnt) {
              res.clear();
              res.push_back(cur);
              maxcnt = cnt;
          }
          else if (cnt == maxcnt) res.push_back(cur);
          stk.pop();
          p = p->right;
        }
        return res;
    }
};
```

502. IPO

Hard

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

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

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

Example 1:

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

Output: 4

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

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

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

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

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

Note:

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

503. Next Greater Element II

Medium

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

Example 1:

Input: [1,2,1]

Output: [2,-1,2]

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

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

```
class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        stack<int> stk;
        int n = nums.size();
        vector<int> res(n, -1);
        for(int k = 0; k < 2; k++) {
            for (int i = 0; i < n; i++) {
                while (!stk.empty() && nums[stk.top()] < nums[i]) {
                    res[stk.top()] = nums[i];
                    stk.pop();
                }
                stk.push(i);
            }
        }
        return res;
    }
};
```

504. Base 7

Easy

Given an integer, return its base 7 string representation.

Example 1:

Input: 100

Output: "202"

Example 2:

Input: -7

Output: "-10"

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

```
class Solution {
public:
    string convertToBase7(int num) {
        if (num == 0) return "0";
        string res;
        bool isNeg = num < 0;
        while (num != 0) {
            res += char('0'+abs(num % 7));
            num /= 7;
        }
        reverse(res.begin(), res.end());
        return isNeg ? '-' + res : res;
    }
};
```

506. Relative Ranks

Easy

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

Example 1:

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

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

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

Note:

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

```
class Solution {
public:
    vector<string> findRelativeRanks(vector<int>& nums) {
        int n = nums.size();
        map<int, int> mp;
        for (int i = 0; i < n; i++) mp[nums[i]] = i;
        vector<string> ans(n);
        int cnt = 1;
        for (map<int, int>::reverse_iterator it = mp.rbegin(); it !=
mp.rend(); it++, cnt++) {
            if (cnt == 1) ans[it->second] = "Gold Medal";
            else if (cnt == 2) ans[it->second] = "Silver Medal";
            else if (cnt == 3) ans[it->second] = "Bronze Medal";
            else ans[it->second] = to_string(cnt);
        }
        return ans;
    }
};
```

507. Perfect Number

Easy

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

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

Example:

Input: 28

Output: True

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

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

```cpp
class Solution {
public:
    bool checkPerfectNumber(int num) {
        if (num <= 1) return false;
        int sum = 0, Sqrt = sqrt(num);
        for (int i = 1; i <= Sqrt; i++) {
            if (num % i == 0) {
                if (i == 1 || num/i == i) sum += i;
                else sum += i + num/i;
                if (sum > num) break;
            }
        }
        return sum == num;
    }
};
```

508. Most Frequent Subtree Sum

Medium

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

Examples 1

Input:

```
  5

 /  \

2    -3
```

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

Examples 2

Input:

```
  5

 /  \

2    -5
```

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

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

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> findFrequentTreeSum(TreeNode* root) {
        dfs(root);
        vector<int> res;
        for (auto &i : m) if (i.second == MaxCnt) {
            res.push_back(i.first);
        }
        return res;
    }

private:
    unordered_map<int, int> m;
    int MaxCnt = 0;

    int dfs(TreeNode *root) {
        if (!root) return 0;
        int l = dfs(root->left), r = dfs(root->right);
        int ret = l + r + root->val;
        MaxCnt = max(MaxCnt, ++m[ret]);
        return ret;
    }
};
```

509. Fibonacci Number

Easy

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

```
F(0) = 0,    F(1) = 1

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

Given `N`, calculate `F(N)`.

Example 1:

```
Input: 2

Output: 1

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

Example 2:

```
Input: 3

Output: 2

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

Example 3:

```
Input: 4

Output: 3

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

Note:

0 ≤ `N` ≤ 30.

```
class Solution {
public:
    struct Matrix{
        int a[2][2] = {0};
        void init(){
            for (int i = 0; i < 2; i++)
               a[i][i] = 1;
        }
    };

    Matrix MUL(Matrix A, Matrix B){
        Matrix C;
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                for (int k = 0; k < 2; k++){
                    C.a[i][j] += A.a[i][k] * B.a[k][j];
                }
            }
        }
        return C;
    }


    int fib(int n) {
        Matrix res, x;
        x.a[0][0] = x.a[0][1] = x.a[1][0] = 1;
        x.a[1][1] = 0;
        res.init();
        while (n) {
            if (n&1) res = MUL(res, x);
            x = MUL(x, x);
            n >>= 1;
        }
        return res.a[1][0];
    }
};
```

513. Find Bottom Left Tree Value

Medium

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

Example 1:

```
Input:

    2
   / \
  1   3

Output:
1
```

Example 2:

```
Input:

        1
       / \
      2   3
     /   / \
    4   5   6
       /
      7

Output:
7
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*> que;
        que.push(root);
        TreeNode *t;
        while (!que.empty()) {
            t = que.front();
            que.pop();
            if (t->right) que.push(t->right);
            if (t->left) que.push(t->left);
        }
        return t->val;
    }
};
```

514. Freedom Trail

Hard

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

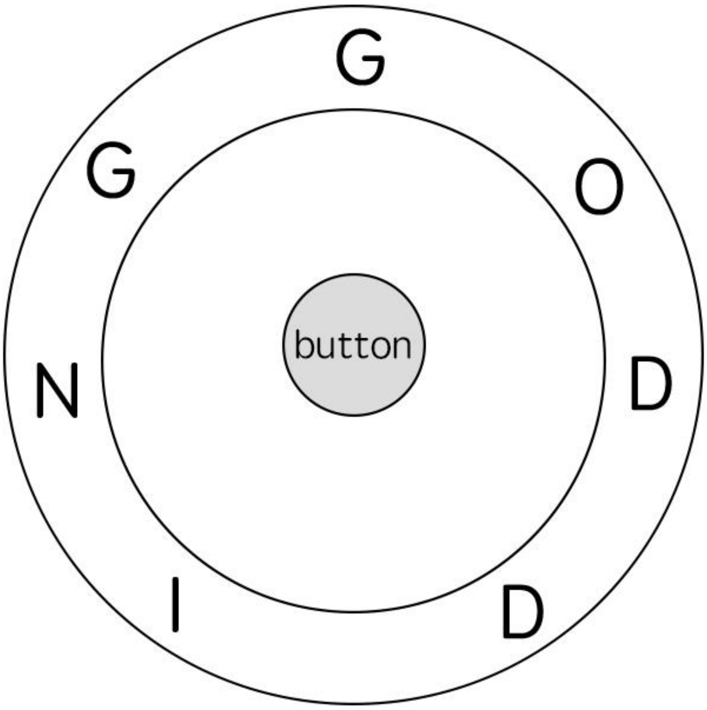
Given a string **ring**, which represents the code engraved on the outer ring and another string **key**, which represents the keyword needs to be spelled. You need to find the **minimum** number of steps in order to spell all the characters in the keyword.

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

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

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

Example:



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

Output: 4

Explanation:

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

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

Also, we need 1 more step for spelling.

So the final output is 4.

Note:

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

```
class Solution {
public:
    int findRotateSteps(string ring, string key) {
        unordered_map<char, vector<int>> mp;
        int n = ring.size();
        for (int i = 0; i < n; ++i) mp[ring[i]].push_back(i);
        vector<vector<pair<int, int>>> dp(2);
        dp[0].push_back({0, 0});
        int k = 0;
        for (auto c : key) {
            vector<pair<int, int>> &cur = dp[k^1], &pre = dp[k];
            cur.clear();
            for (auto pos : mp[c]) {
                int step = INT_MAX;
                for (auto &pii : pre) {
                    step = min(step, pii.second + dist(pii.first, pos, n));
                }
                cur.push_back({pos, step});
            }
            k ^= 1;
        }
        int res = INT_MAX;
        for (auto &pii : dp[k]) {
            res = min(res, pii.second);
        }
        return res + key.size();
    }
private:
    int dist(int i, int j, int n) {
        int t = (i-j+n) % n;
        return min(t, n-t);
    }
};
```

515. Find Largest Value in Each Tree Row

Medium

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

Example:

Input:

```
          1
         / \
        3   2
       / \   \
      5   3   9
```

Output: [1, 3, 9]

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> largestValues(TreeNode* root) {
        vector<int> res;
        if (!root) return res;
        queue<TreeNode*> que;
        que.push(root);
        while (!que.empty()) {
            int Max = que.front()->val, sz = que.size();
            while (sz--) {
                auto t = que.front();
                if (t->left) que.push(t->left);
                if (t->right) que.push(t->right);
                que.pop();
                Max = max(Max, t->val);

            }
            res.push_back(Max);
        }
        return res;
    }
};
```

516. Longest Palindromic Subsequence

Medium

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

Example 1:

Input:

```
"bbbab"
```

Output:

```
4
```

One possible longest palindromic subsequence is "bbbb".

Example 2:

Input:

```
"cbbd"
```

Output:

```
2
```

One possible longest palindromic subsequence is "bb".

```
class Solution {
public:
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<vector<int>> dp(n, vector<int>(n));
        for (int i = n - 1; i >= 0; --i) {
            dp[i][i] = 1;
            for (int j = i + 1; j < n; ++j) {
                if (s[i] == s[j]) {
                    dp[i][j] = dp[i + 1][j - 1] + 2;
                } else {
                    dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[0][n - 1];
    }
};
```

```
class Solution {
public:
    int longestPalindromeSubseq(string s) {
        int n = s.length();
        vector<vector<int>> g(n, vector<int>(n, 1));
        for (int len = 1; len < n; len++) {
            for (int i = 0, j = len; j < n; i++, j++) {
                g[i][j] = s[i] != s[j] ?  max(g[i+1][j], g[i][j-1])
                                 : (2 + (j-i > 1 ? g[i+1][j-1] : 0));
            }
        }
        return g[0][n-1];
    }
};
```

517. Super Washing Machines

Hard

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

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

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

Example1

Input: [1,0,5]

Output: 3

Explanation:

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

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

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

Example2

Input: [0,3,0]

Output: 2

Explanation:

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

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

Example3

Input: [0,2,0]

Output: -1

Explanation:

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

Note:

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

518. Coin Change 2

Medium

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

Example 1:

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

Output: 4

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

5=5

5=2+2+1

5=2+1+1+1

5=1+1+1+1+1

Example 2:

Input: amount = 3, coins = [2]

Output: 0

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

Example 3:

Input: amount = 10, coins = [10]

Output: 1

Note:

You can assume that

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

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

519. Random Flip Matrix

Medium

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

Note:

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

Example 1:

Input:

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

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

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

Example 2:

Input:

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

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

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

Explanation of Input Syntax:

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

```
class Solution {
public:
    Solution(int n_rows, int n_cols) : n(n_rows), m(n_cols) {
        reset();
    }

    vector<int> flip() {
        int used_id = rand() % (sz--);
        int unused_id = mp.count(used_id) ? mp[used_id] : used_id;
        mp[used_id] = mp.count(sz) ? mp[sz] : sz;
        return {unused_id / m, unused_id % m};

    }

    void reset() {
        mp.clear();
        sz = n * m;
    }

private:
    int n, m, sz;
    unordered_map<int, int> mp;  //<used, not used>
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(n_rows, n_cols);
 * vector<int> param_1 = obj->flip();
 * obj->reset();
 */
```

520. Detect Capital

Easy

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

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

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

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

Example 1:

Input: "USA"

Output: True

Example 2:

Input: "FlaG"

Output: False

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

```
class Solution {
public:
    bool detectCapitalUse(string a) {
        for(int i = 1; i < a.size(); i++) {
            if (isupper(a[1]) != isupper(a[i])
                  || islower(a[0]) && isupper(a[i]))
                return false;
        }
        return true;
    }
};
```

521. Longest Uncommon Subsequence I

Easy

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

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

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

Example 1:

Input: "aba", "cdc"

Output: 3

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

Note:

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

```
class Solution {
public:
    int findLUSlength(string a, string b) {
        if (a == b) return -1;
        else return max(a.size(), b.size());
    }
};
```

522. Longest Uncommon Subsequence II

Medium

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

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

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

Example 1:

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

Output: 3

Note:

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

```
class Solution {
public:
    int findLUSlength(vector<string>& strs) {
        int n = strs.size();
        unordered_set<string> s;
        sort(strs.begin(), strs.end(), [](string a, string b){
            if (a.size() == b.size()) return a > b;
            return a.size() > b.size();
        });
        for (int i = 0; i < n; ++i) {
            if (i == n-1 || strs[i] != strs[i+1]) {
                bool found = true;
                for (auto &a : s) {
                    int j = 0;
                    for (char c : a) {
                        if (c == strs[i][j]) ++j;
                        if (j == strs[i].size()) break;
                    }
                    if (j == strs[i].size()) {
                        found = false;
                        break;
                    }
                }
                if (found) return strs[i].size();
            }
            s.insert(strs[i]);
        }
        return -1;
    }
};
```

523. Continuous Subarray Sum

Medium

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

Example 1:

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

Output: True

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

Example 2:

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

Output: True

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

Note:

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

```
class Solution {
public:
    bool checkSubarraySum(vector<int>& nums, int k) {
        int n = nums.size(), sum = 0, pre = 0;
        unordered_map<int, int> modk;
        modk[0] = -1;
        for (int i = 0; i < n; ++i) {
            sum += nums[i];
            // 特例 k == 0
            int mod = k == 0 ? sum : sum % k;
            if (modk.count(mod)) {
                if (modk[mod] != i-1) return true;
            }
            else modk[mod] = i;
        }
        return false;
    }
};
```

524. Longest Word in Dictionary through Deleting

Medium

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

Example 1:

Input:

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

Output:

"apple"

Example 2:

Input:

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

Output:

"a"

Note:

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

```cpp
class Solution {
public:
    string findLongestWord(string s, vector<string>& d) {
        string res = "";
        for (auto &a : d) {
            if (a.length() < res.length()) continue;
            if (check(a, s) && (res.length() < a.length() || res > a)){
                res = a;
            }
        }
        return res;
    }

private:
    bool check(string &a, string &s) {
        auto i = a.begin(), j = s.begin();
        while (i != a.end() && j != s.end()) {
            if (*i == *j) i++;
            j++;
        }
        return i == a.end();
    }
};
```

525. Contiguous Array

Medium

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

Example 1:

Input: [0,1]

Output: 2

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

Example 2:

Input: [0,1,0]

Output: 2

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

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

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

526. Beautiful Arrangement

Medium

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

1. The number at the i_{th} position is divisible by **i**.
2. **i** is divisible by the number at the i_{th} position.

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

Example 1:

Input: 2

Output: 2

Explanation:

The first beautiful arrangement is [1, 2]:

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

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

The second beautiful arrangement is [2, 1]:

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

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

Note:

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

```
class Solution {
public:
    int countArrangement(int N) {
        vector<int> nums;
        for(int i = 0; i < N; i++) nums.push_back(i+1);
        return dfs(0, N, nums);
    }

private:
    int dfs(int i, int n, vector<int> &nums) {
        if (i == n) return 1;
        int res = 0;
        for (int j = i; j < n; ++j) {
            if (nums[j] % (i+1) == 0 || (i+1) % nums[j] == 0) {
                swap(nums[i],nums[j]);
                res += dfs(i+1, n, nums);
                swap(nums[i],nums[j]);
            }
        }
        return res;
    }
};
```

528. Random Pick with Weight

Medium

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

Note:

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

Example 1:

Input:

["Solution","pickIndex"]

[[[1]],[]]

Output: [null,0]

Example 2:

Input:

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

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

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

Explanation of Input Syntax:

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

```
class Solution {
public:
    vector<int> v;
    int sum = 0;
    //c++11 random integer generation
    default_random_engine e;
    uniform_int_distribution<int> uni;

    Solution(vector<int> w) {
        for (int x : w) {
            v.push_back(sum += x);
        }
        uni = uniform_int_distribution<int>{0, sum - 1};
    }

    int pickIndex() {
        int targ = uni(e);
        int low = 0, high = v.size();
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (targ >= v[mid]) low = mid + 1;
            else high = mid;
        }
        return low;
    }
};
```

529. Minesweeper

Medium

Let's play the minesweeper game (Wikipedia, online game)!

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

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

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

Example 1:

Input:

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

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

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

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



Click : [3,0]
```

Output:

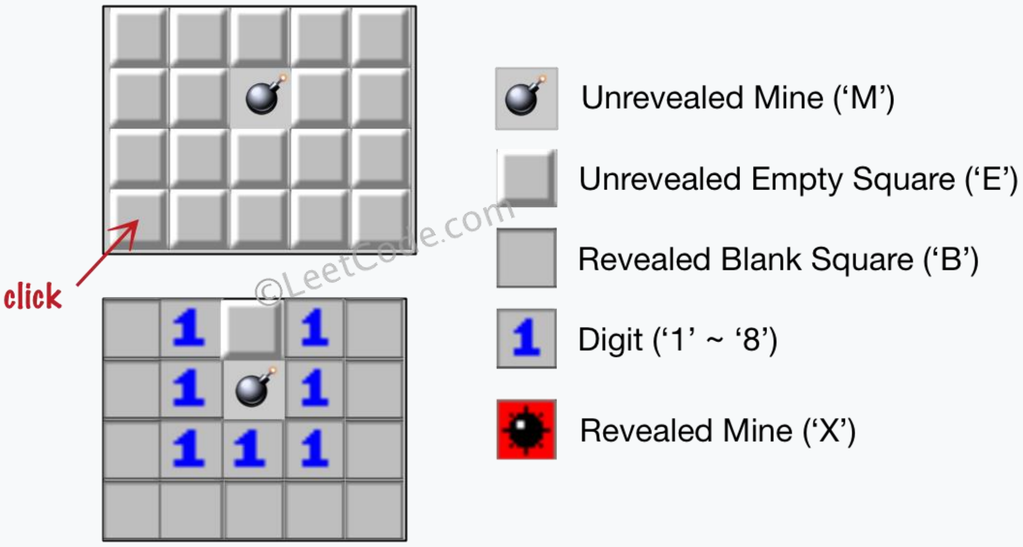
```
[['B', '1', 'E', '1', 'B'],
```

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

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

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

Explanation:



Example 2:

Input:

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

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

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

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

Click : [1,2]

Output:

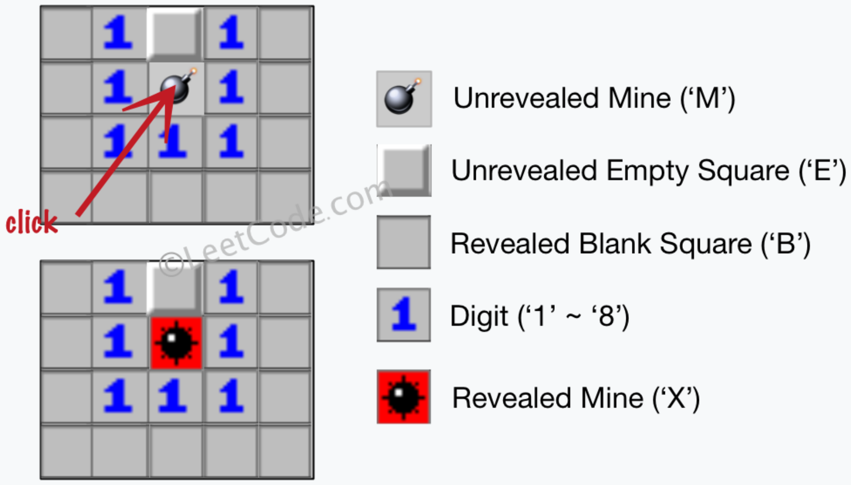
```
[['B', '1', 'E', '1', 'B'],

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

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

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

Explanation:



Note:

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

```
class Solution {
public:
    vector<vector<char>> updateBoard(vector<vector<char>>& board, vector<int>& click) {
        n = board.size(), m = board[0].size();
        if (board[click[0]][click[1]] == 'M') {
            board[click[0]][click[1]] = 'X';
        }
        else dfs(click[0], click[1], board);
        return board;
    }
private:
    int n, m;
    const vector<int> dx{0,0,-1,1,-1,1,-1,1};
    const vector<int> dy{-1,1,0,0,-1,1,1,-1};

    void dfs(int x, int y, vector<vector<char>>& board) {
        char cnt = '0';
        vector<pair<int, int>> book;
        for (int i = 0; i < 8; ++i) {
            int xx = x + dx[i], yy = y + dy[i];
            if (xx >= n || yy >= m || xx < 0 || yy < 0) continue;
            if (board[xx][yy] == 'M') ++cnt;
            if (board[xx][yy] == 'E')
                book.push_back({xx, yy});
        }
        board[x][y] = cnt == '0' ? 'B' : cnt;
        if (cnt == '0') {
            for (auto &pii : book) dfs(pii.first, pii.second, board);
        }
    }
};
```

530. Minimum Absolute Difference in BST

Easy

Given a binary search tree with non-negative values, find the minimum absolute difference between values of any two nodes.

Example:

Input:

```
   1
    \
     3
    /
   2
```

Output:

1

Explanation:

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

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

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int getMinimumDifference(TreeNode* p) {
        int pre, res = INT_MAX;
        bool isFirst = true;
        stack<TreeNode*> stk;
        while (p || !stk.empty()) {
            while (p) {
                stk.push(p);
                p = p->left;
            }
            p = stk.top();
            stk.pop();
            if (isFirst) {
                pre = p->val;
                isFirst = false;
            }
            else {
                res = min(res, p->val - pre);
                pre = p->val;
            }
            p = p->right;
        }
        return res;
    }
};
```

532. K-diff Pairs in an Array

Easy

Given an array of integers and an integer **k**, you need to find the number of **unique** k-diff pairs in the array. Here a **k-diff** pair is defined as an integer pair (i, j), where **i** and **j** are both numbers in the array and their absolute difference is **k**.

Example 1:

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

Output: 2

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

Example 2:

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

Output: 4

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

Example 3:

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

Output: 1

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

Note:

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

```
class Solution {
public:
    int findPairs(vector<int>& nums, int k) {
        if (k < 0) return 0;
        unordered_map<int, int> mp;
        for (auto &i : nums) ++mp[i];
        int res = 0;
        for (auto &i : mp) {
            if (!k && i.second > 1 || k && mp.count(i.first+k))
                ++res;
        }
        return res;
    }
};
```

535. Encode and Decode TinyURL

Medium

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

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

537. Complex Number Multiplication

Medium

Given two strings representing two complex numbers.

You need to return a string representing their multiplication. Note i^2 = -1 according to the definition.

Example 1:

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

Output: "0+2i"

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

Example 2:

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

Output: "0+-2i"

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

Note:

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

```
class Solution {
public:
    string complexNumberMultiply(string a, string b) {
        int a1, b1, a2, b2;
        sscanf(a.c_str(), "%d+%di", &a1, &b1);
        sscanf(b.c_str(), "%d+%di", &a2, &b2);
        return to_string(a1*a2-b1*b2)+"+"+to_string(a1*b2+a2*b1)+ "i";
    }
};
```

538. Convert BST to Greater Tree

Easy

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

Example:

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

              5

            /   \

           2     13



Output: The root of a Greater Tree like this:

             18

            /   \

          20     13
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *    int val;
 *    TreeNode *left;
 *    TreeNode *right;
 *    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* convertBST(TreeNode* root) {
        if (!root) return nullptr;
        stack<TreeNode*> stk;
        TreeNode *p = root;
        int sum = 0;
        while (!stk.empty() || p) {
            while (p) {
                stk.push(p);
                p = p->right;
            }
            p = stk.top();
            stk.pop();
            // visit
            sum = p->val = p->val + sum;
            p = p->left;
        }
        return root;
    }
};
```

539. Minimum Time Difference

Medium

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

Example 1:

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

Output: 1

Note:

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

```
class Solution {
public:
    int findMinDifference(vector<string>& timePoints) {
        vector<bool> bucket(24*60, false);
        for (auto &s : timePoints) {
            int time = stoi(s.substr(0, 2))*60 + stoi(s.substr(3, 2));
            if (bucket[time]) return 0;
            bucket[time] = true;
        }
        int pre = -1, first, res = INT_MAX;
        for (int i = 0; i < 24*60; i++) if (bucket[i]) {
            if (pre == -1) first = pre = i;
            else {
                res = min(res, i-pre);
                pre = i;
            }
        }
        return min(res, first+24*60-pre);
    }
};
```

540. Single Element in a Sorted Array

Medium

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

Example 1:

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

Output: 2

Example 2:

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

Output: 10

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

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

541. Reverse String II

Easy

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

Example:

Input: s = "abcdefg", k = 2

Output: "bacdfeg"

Restrictions:

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

```cpp
class Solution {
public:
    string reverseStr(string s, int k) {
        auto l = s.begin();
        while (l != s.end()) {
            auto r = (s.end() - l > k) ? l + k : s.end();
            reverse(l, r);
            l = (s.end() - r > k) ? r + k : s.end();
        }
        return s;
    }
};
```

542. 01 Matrix

Medium

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

The distance between two adjacent cells is 1.

Example 1:

```
Input:

[[0,0,0],

  [0,1,0],

  [0,0,0]]



Output:

[[0,0,0],

 [0,1,0],

 [0,0,0]]
```

Example 2:

```
Input:

[[0,0,0],

  [0,1,0],

  [1,1,1]]



Output:

[[0,0,0],

  [0,1,0],

  [1,2,1]]
```

Note:

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

```
class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
        int n = matrix.size(), m = matrix[0].size();
        const int MAX = 10000+10;
        vector<vector<int>> res(n, vector<int>(m, MAX));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (!matrix[i][j]) res[i][j] = 0;
                else {
                    int up = i ? res[i-1][j] : MAX;
                    int left = j ? res[i][j-1] : MAX;
                    res[i][j] = min(up, left)+1;
                }
            }
        }

        for (int i = n-1; i >= 0; i--) {
            for (int j = m-1; j >= 0; j--) if (matrix[i][j]) {
                int down = i != n-1 ? res[i+1][j] : MAX;
                int right = j != m-1 ? res[i][j+1] : MAX;
                res[i][j] = min(res[i][j], min(down, right)+1);
            }
        }
        return res;
    }
};
```

543. Diameter of Binary Tree

Easy

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

Example:

Given a binary tree

```
          1

         / \

        2   3

       / \

      4   5
```

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

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

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        int res = 0;
        f(root, res);
        return res;
    }

private:
    int f(TreeNode *root, int &res) {
        if (!root) return 0;
        int L = f(root->left, res);
        int R = f(root->right, res);
        res = max(res, L+R);
        return max(L, R) + 1;
    }
};
```

546. Remove Boxes

Hard

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

Example 1:
Input:

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

Output:

```
23
```

Explanation:

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

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

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

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

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

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

547. Friend Circles

Medium

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

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

Example 1:

Input:

[[1,1,0],

 [1,1,0],

 [0,0,1]]

Output: 2

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

Example 2:

Input:

[[1,1,0],

 [1,1,1],

 [0,1,1]]

Output: 1

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

Note:

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

```
class Solution {
public:
    int findCircleNum(vector<vector<int>>& M) {
        int n = M.size(), res = n;
        vector<int> fa(n);
        for (int i = 0; i < n; i++) fa[i] = i;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) if (M[i][j]) {
                int fa0 = find(fa, i);
                int fa1 = find(fa, j);
                if (fa0 != fa1) {
                    fa[fa0] = fa1;
                    res--;
                }
            }
        }
        return res;
    }

private:
    int find(vector<int> &fa, int x) {
        return x == fa[x] ? x : (fa[x] = find(fa, fa[x]));
    }
};
```

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 10^9 + 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.

Too slow

```
class Solution {
public:
    int checkRecord(int n) {
        if (n <= 0) return 0;
        vector<long long> pre(6, 0);
        pre[0] = 1;
        while (n--) {
            vector<long long> cur(6, 0);

            cur[0] = pre[0] + pre[1] + pre[2];
            cur[3] = cur[0] + pre[3] + pre[4] + pre[5];

            for (int k = 0; k < 6; k++) {
                if (k % 3 == 0) continue;
                cur[k] += pre[k-1];
            }

            for (auto &i : cur) i %= 1000000007;
            pre = cur;
        }
        return  (accumulate(pre.begin(),  pre.end(),  (long  long)0))  %
1000000007;
    }
};
```

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.

```
class Solution {
public:
    string optimalDivision(vector<int>& nums) {
        string res;
        if (nums.empty()) return res;
        res = to_string(nums[0]);
        if (nums.size() == 1) return res;
        if (nums.size() == 2) return res + "/" + to_string(nums[1]);
        res += "/(" + to_string(nums[1]);
        for (int i = 2; i < nums.size();++i)
            res += "/" + to_string(nums[i]);
        return res + ")";
    }
};
```

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],

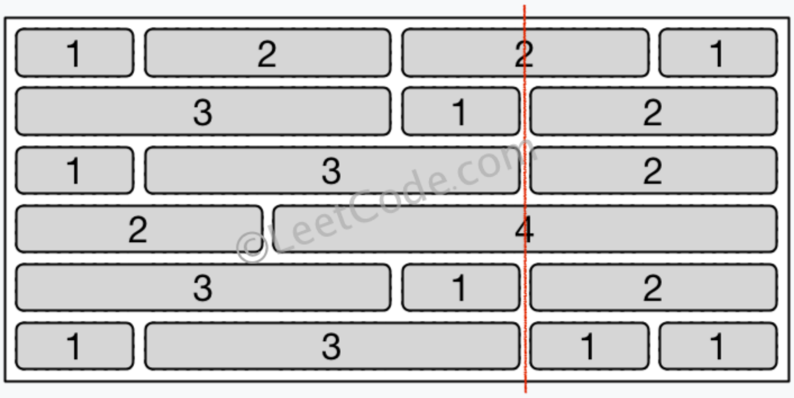
[2,4],

[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;
    }
};
```

557. Reverse Words in a String III

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1:
Input: "Let's take LeetCode contest"
Output: "s'teL ekat edoCteeL tsetnoc"

Note:
In the string, each word is separated by single space and there will not be any extra space in the string.

```
class Solution {
public:
    string reverseWords(string s) {
        string::iterator p = s.begin(), q;
        while (p != s.end()) {
            q = p;
            while (q != s.end() && *q != ' ') q++;
            reverse(p, q);
            if (q == s.end()) break;
            p = ++q;
        }
        return s;
    }
};
```

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.
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

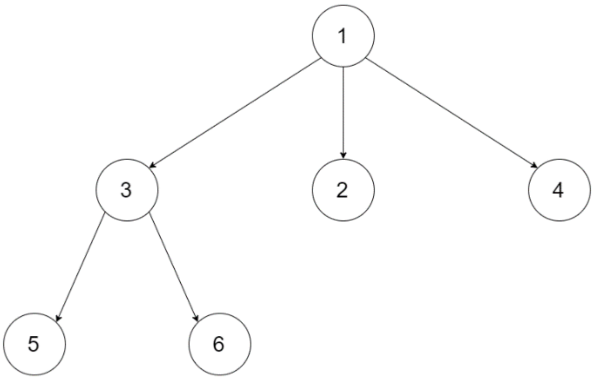
Easy

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).

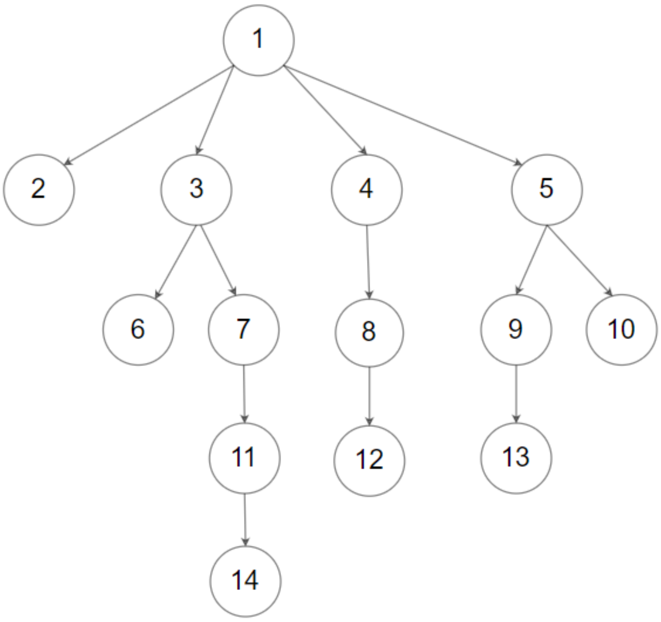
Example 1:



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].

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        unordered_map<int, int> mp{{0, 1}};
        int sum = 0, res = 0, sz = nums.size();
        for (int i = 0; i < sz; i++) {
            sum += nums[i];
            res += (mp.count(sum-k) ? mp[sum-k] : 0);
            ++mp[sum];
        }
        return res;

    }
};
```

561. Array Partition I

Easy

Given an array of **2n** integers, your task is to group these integers into **n** pairs of integer, say (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) which makes sum of min(a_i, b_i) 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].

```
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int res = 0;
        for (int i = 0; i < nums.size(); i += 2) res += nums[i];
        return res;
    }
};
```

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;
    }
};
```

```cpp
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

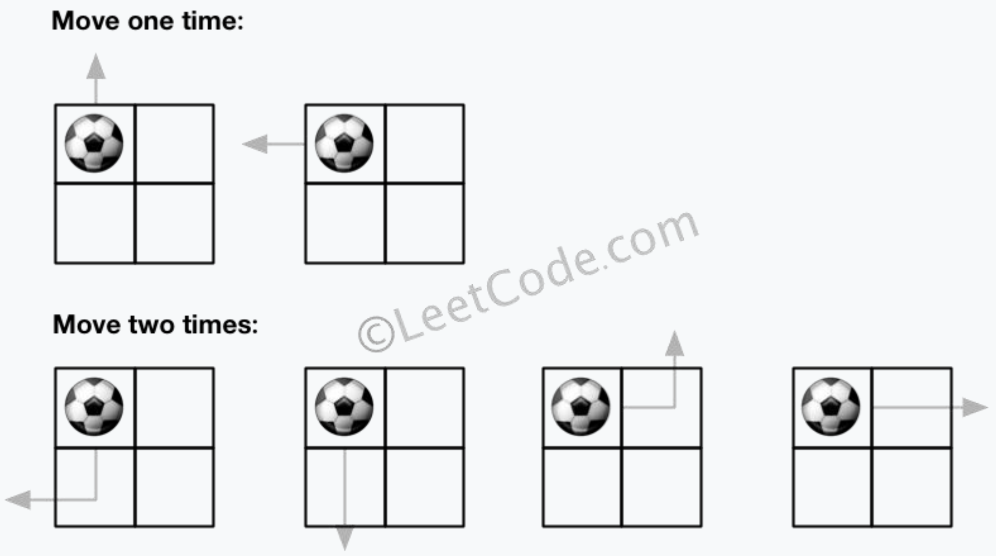
There is an **m** by **n** grid with a ball. Given the start coordinate **(i,j)** of the ball, you can move the ball to **adjacent** cell or cross the grid boundary in four directions (up, down, left, right). However, you can **at most** move **N** times. Find out the number of paths to move the ball out of grid boundary. The answer may be very large, return it after mod $10^9 + 7$.

Example 1:

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

Output: 6

Explanation:

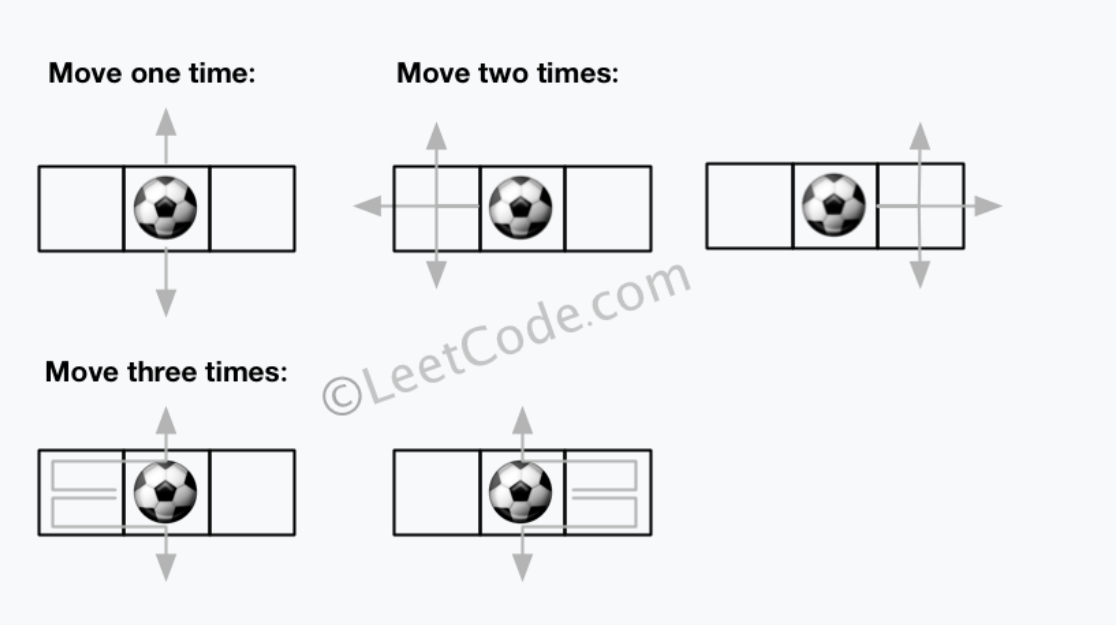


Example 2:

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 **<=**.

```
/**
 *          /-------------\
 * nums:  [2, 6, 4, 8, 10, 9, 15]
 * minr:   2  4  4  8   9  9  15
 *        <---------------------
 * maxl:   2  6  6  8  10 10  15
 *        --------------------->
 */
class Solution {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        int n = nums.size();
        vector<int> maxlhs(n);   // max number from left to cur
        vector<int> minrhs(n);   // min number from right to cur
        for (int i = n-1, minr = INT_MAX; i >= 0; --i)
            minrhs[i] = minr = min(minr, nums[i]);
        for (int i = 0, maxl = INT_MIN; i < n;  ++i)
            maxlhs[i] = maxl = max(maxl, nums[i]);

        int i = 0, j = n-1;
        while (i < n && nums[i] == minrhs[i]) ++i;
        while (j > i && nums[j] == maxlhs[j]) --j;

        return j + 1 - i;
    }
};
```

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

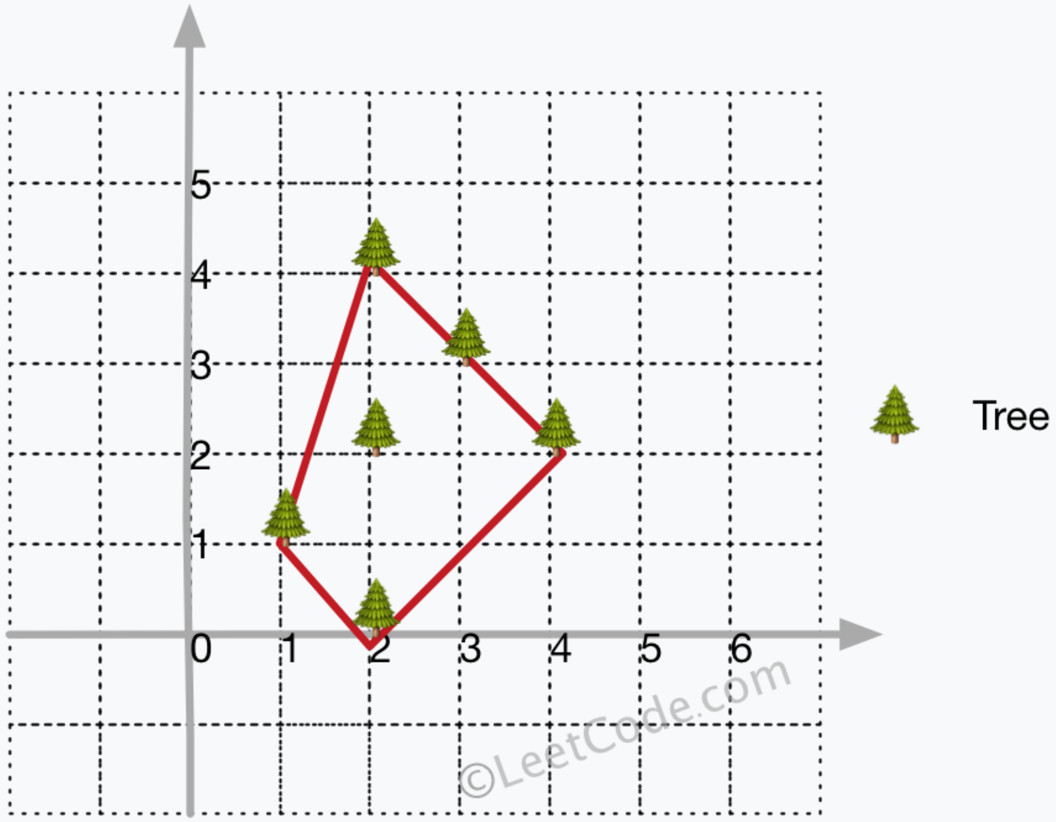
There are some trees, where each tree is represented by (x,y) coordinate in a two-dimensional garden. Your job is to fence the entire garden using the **minimum length** of rope as it is expensive. The garden is well fenced only if all the trees are enclosed. Your task is to help find the coordinates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

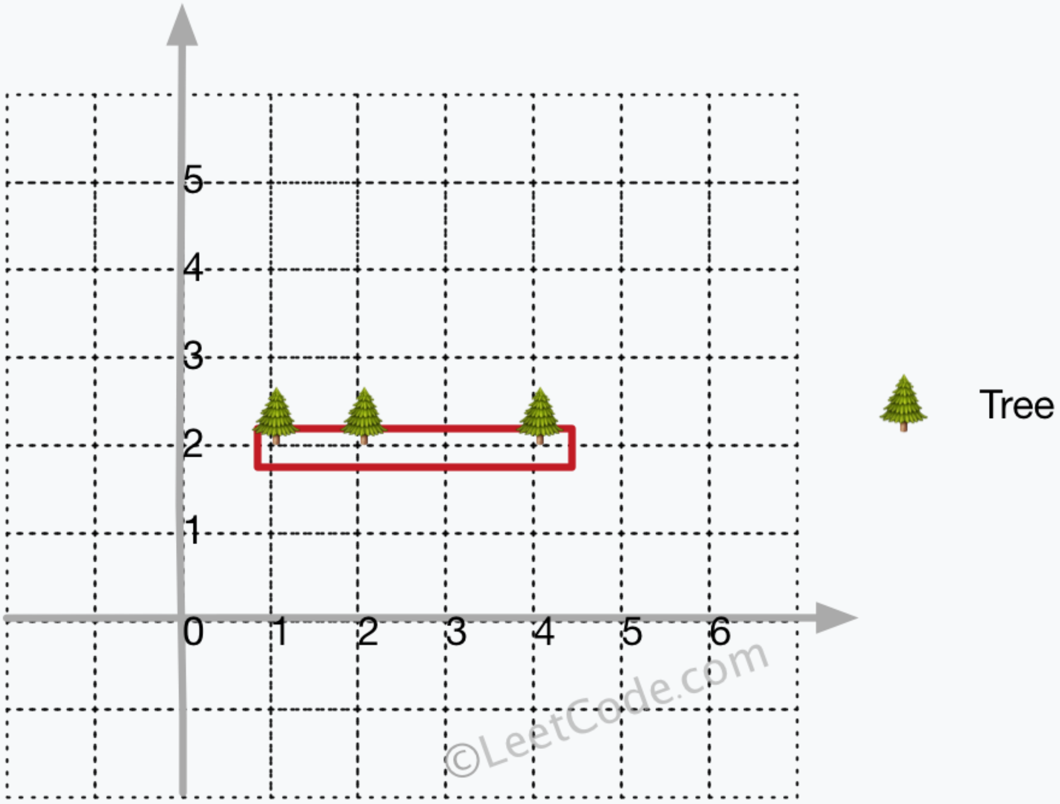


Example 2:

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]
                this && i[1] >= q[1];
        return a && b;
    }
};
```

589. N-ary Tree Preorder Traversal

Easy

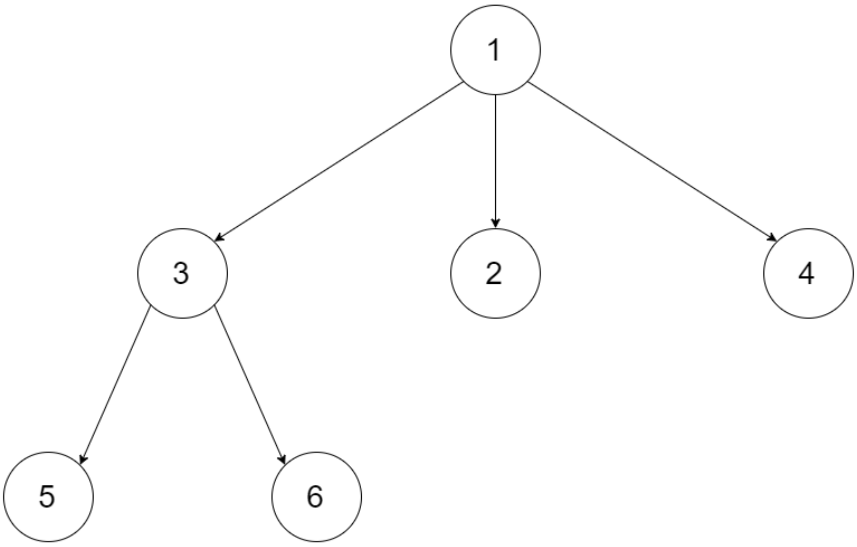
Given an n-ary tree, return the *preorder* traversal of its nodes' values.

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?

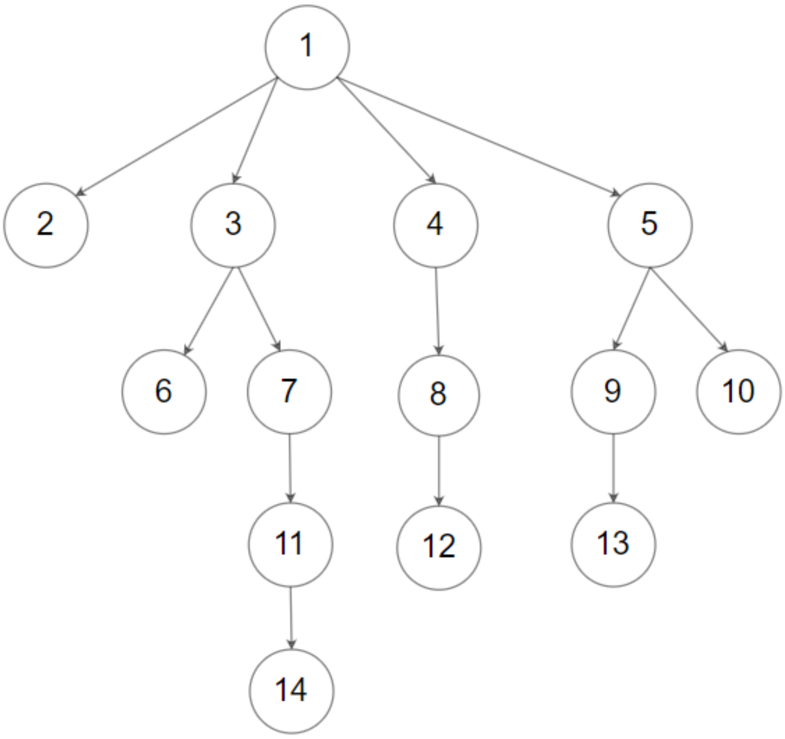
Example 1:



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]`

```cpp
/*
// 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

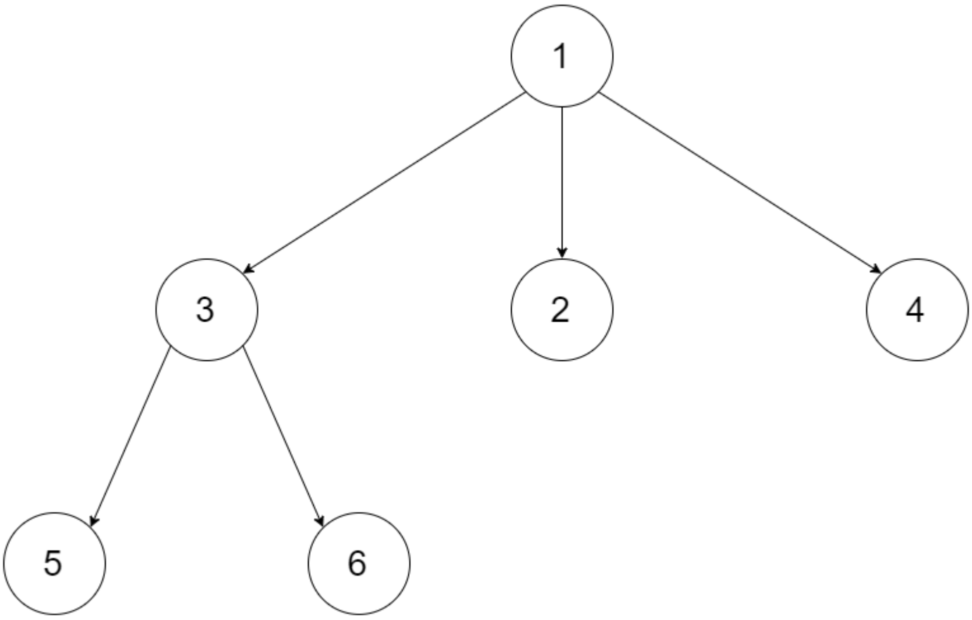
Given an n-ary tree, return the *postorder* traversal of its nodes' values.

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?

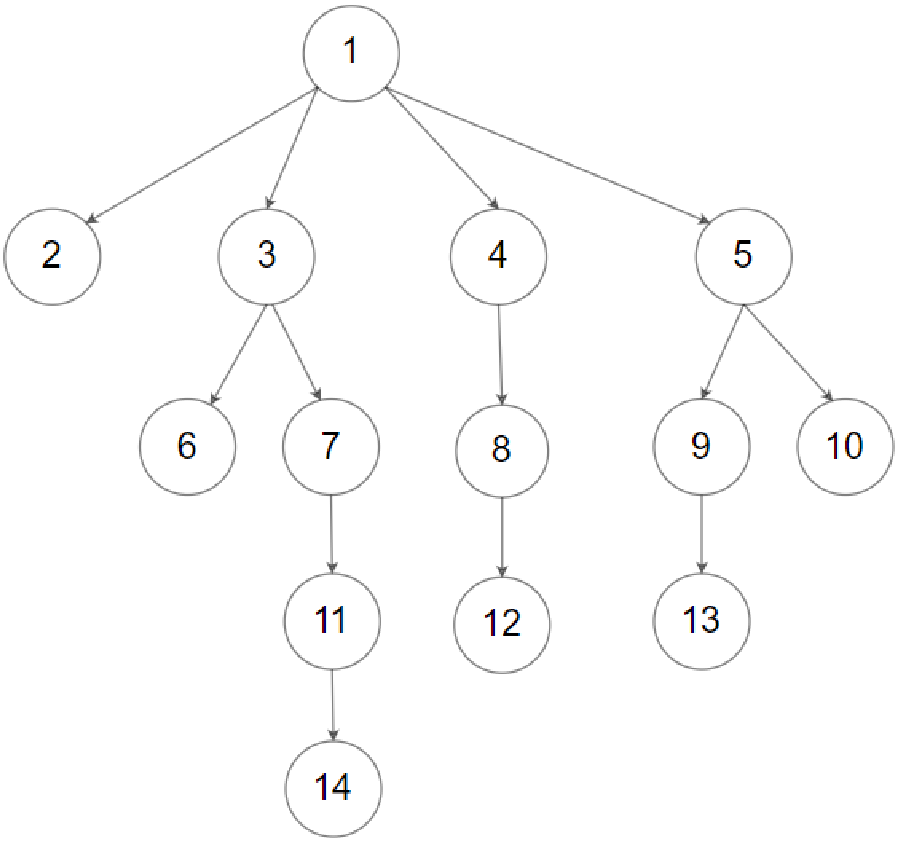
Example 1:



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]`

```
class Solution {
public:
    vector<int> postorder(Node* p) {
        if (!p) return {};
        stack<pair<Node *, size_t>> stk;
        vector<int> res;
        stk.emplace(p, 0);
        while (!stk.empty()) {
            auto &[p, index] = stk.top();
            if (index == p->children.size()) {
                res.push_back(p->val);
                stk.pop();
            } else {
                stk.emplace(p->children[index++], 0);
            }
        }
        return res;
    }
};
```

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. 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.

```cpp
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.

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 <= 10^9