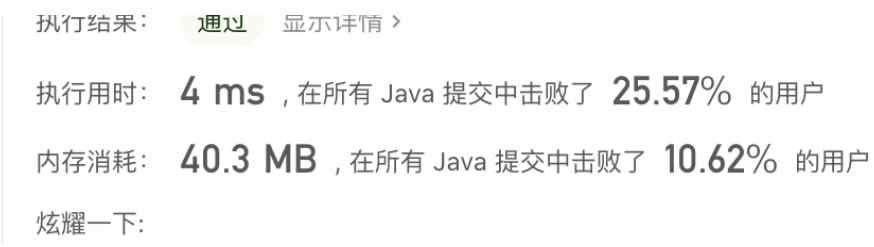
501 - 600

501Find Mode in Binary Search Tree




```java
/*
    简单易懂的暴力方法
*/
HashMap<Integer, Integer> map = new HashMap<>();
    int maxFreq = 0;
    public int[] findMode(TreeNode root) {
        inorder(root);
        List<Integer> list = new ArrayList<>();
        for(Integer num : map.keySet()){
            if(map.get(num) == maxFreq)
                list.add(num);
        }

        int[] res = new int[list.size()];
        for(int i = 0; i < list.size(); i++){
            res[i] = list.get(i);
        }

        return res;
    }

    public void inorder(TreeNode root){
        if(root == null) return;

        inorder(root.left);
        map.put(root.val, map.getOrDefault(root.val, 0) + 1);
        maxFreq = Math.max(map.get(root.val), maxFreq);
        inorder(root.right);
    }
```

```java
/*
    对中序遍历进行改造
*/
    int maxFreq = 0;
    int count = 0;
    List<Integer> list = new ArrayList<>();
    TreeNode pre = null;
    public int[] findMode(TreeNode root) {
        inorder(root);

        int[] res = new int[list.size()];
        for(int i = 0; i < list.size(); i++){
            res[i] = list.get(i);
        }

        return res;
    }

    public void inorder(TreeNode root){
        if(root == null) return;

        inorder(root.left);

        if(pre == null || pre.val == root.val){
            count++;
        }else{
            count = 1;
        }

        pre = root;

        if(count > maxFreq){
            list.clear();
            maxFreq = count;
            list.add(root.val);
        }else if(count == maxFreq){
            list.add(root.val);
        }

        inorder(root.right);
    }


```

502 IPO

502. IPO

难度 困难 72

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
```

```
/*
  source : https://www.youtube.com/watch?v=VFXfhB8vS94&t=570s

  写的真好，思路是：
  使用两个优先队列，第一个存储我们的pair 键值对， Profit + Capital
  第二个是我们的consideration set
    它用来，贪心，做出当前最优选择
```

```cpp

  整体循环 K 次， 拿到每次的结果

  同时注意。 Capital 只是一个threashold， 并不会扣除我们的 profit
  profit 是我们的净利润
*/

typedef pair<int, int> ii;
class cmp{
public:
  bool operator()(ii a, ii b){
    return a.first > b.first;
  }
};

class Solution{
  public:
    int findMaximizedCapital(int k, int W, vector<int>& Profits, vector<int>&
Capital){
      int n = Profits.size();
      vector<ii> projects;

      for(int i = 0; i < n; i++)
        projects.push_back({Capital[i], Profits[i]});

      priority_queue<ii, vector<ii>, cmp> projects_pq{cmp{}, projects};
      priority_queue<int> feasible;

      while(k--){
        while(not projects_pq.empty() and projects_pq.top().first <= W){
          feasible.push(projects_pq.top().second);
          projects_pq.pop();
        }

        if(feasible.empty())
          return W;

        W += feasible.top();
        feasible.pop();
      }

      return W;
    }
};
```

```java
/*
    Java 版本

*/
class Solution {
    public int findMaximizedCapital(int k, int W, int[] Profits, int[] Capital) {
        //  cap, prof, small heap
        PriorityQueue<Node>    pq       = new PriorityQueue<>((x, y) -> x.key - y.key);

        //max heap
        PriorityQueue<Integer> considerationSet = new PriorityQueue<>((x, y) -> (y - x));

        for (int i = 0; i < Profits.length; i++) {
            pq.add(new Node(Capital[i], Profits[i]));
        }

        while(k > 0){
            while(!pq.isEmpty() && pq.peek().key <= W){
                Node node = pq.poll();
                considerationSet.add(node.val);
            }

            if(considerationSet.isEmpty())
                return W;
            W += considerationSet.poll();
            k--;
        }

        return W;
    }


}

class Node{
    public int key;
    public int val;

    public Node(int key, int val) {
        this.key = key;
        this.val = val;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        Node node = (Node) o;
        return key == node.key && val == node.val;
    }

    @Override
    public int hashCode() {
        return toString().hashCode();
    }

    @Override
    public String toString() {
        return key + "@" + val;
    }
}
```

504 Base 7

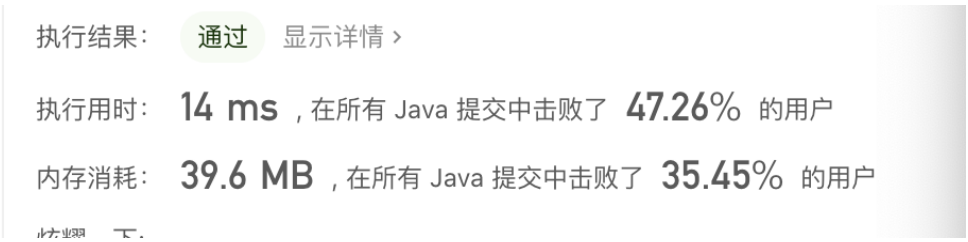
执行用时： 1 ms , 在所有 Java 提交中击败了 90.70% 的用户

内存消耗： 35.9 MB , 在所有 Java 提交中击败了 37.14% 的用户

```
    public String convertToBase7(int num) {
        if(num == 0)        return "0";
        StringBuilder sb = new StringBuilder();
        int sign = num < 0 ? -1 : 1;
        num = num < 0 ? -num : num;
        while(num > 0){
            int quo       = num / 7;
            int remainder = num % 7;
            sb.append(remainder);
            num /= 7;
        }

        sb.reverse();
        if(sign == -1)
            sb.insert(0, '-');
```

```
        return sb.toString();
    }
```

506 Relative Rank




```java

    public String[] findRelativeRanks(int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<>();

        int[] newNums = Arrays.copyOf(nums, nums.length);
        Arrays.sort(newNums);

        for(int i = 0; i < newNums.length; i++){
            map.put(newNums[i], newNums.length - i);
        }

        String[] res = new String[nums.length];
        for(int i = 0; i < res.length; i++){
            if(newNums.length >= 1 && nums[i] == newNums[newNums.length - 1])
                res[i] = "Gold Medal";
            else if(newNums.length >= 2 && nums[i] == newNums[newNums.length - 2])
                res[i] = "Silver Medal";
            else if(newNums.length >= 3 && nums[i] == newNums[newNums.length - 3])
                res[i] = "Bronze Medal";
            else
                res[i] = map.get(nums[i]) + "";
        }

        return res;
    }
```

507 Perfect Number

507. Perfect Number

难度 简单 81

A perfect number is a **positive integer** that is equal to the sum of its **positive divisors**, excluding the number itself. A **divisor** of an integer `x` is an integer that can divide `x` evenly.

Given an integer `n`, return `true` *if* `n` *is a perfect number, otherwise return* `false`.

Example 1:

```
Input: num = 28
Output: true
Explanation: 28 = 1 + 2 + 4 + 7 + 14
1, 2, 4, 7, and 14 are all divisors of 28.
```

Example 2:

```
Input: num = 6
Output: true
```

Example 3:

```
Input: num = 496
Output: true
```

Example 4:

```
Input: num = 8128
```

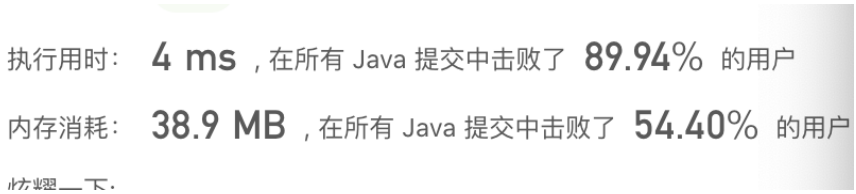
```java
/*
    brute force,  肯定超时
    66/99
*/
public boolean checkPerfectNumber(int num) {
        int sum = 0;

        for(int i = 1 ;i < num; i++){
            if(num % i == 0)
                sum += i;
```

```
        }

        return sum == num;
    }
```

```
/*
    打表
*/
public boolean checkPerfectNumber(int num) {

        switch(num) {
            case 6: case 28: case 496: case 8_128: case 33_550_336: return true;
            default: return false;
        }

    }
```

```
/*
  正经解法
*/
public boolean checkPerfectNumber(int num) {
        /*
            1,  2,  4,  7
        */
        if(num == 1)
            return false;
        int sum = 1;
        for(int i = 2; i <= Math.sqrt(num); i++){
            if(num % i == 0){
                sum += i;
                if(i != num / i)
                    sum += num / i;
            }
        }

        return sum == num;
    }
```

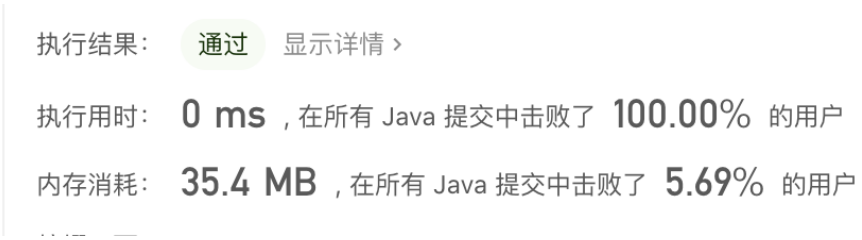
508 Most Frequent Subtree




```java
HashMap<Integer, Integer> map = new HashMap<>();
int maxFreq = 0;
public int[] findFrequentTreeSum(TreeNode root) {
    postOrder(root);

    List<Integer> ans = new ArrayList<>();
    for(Integer num : map.keySet()){
        if(map.get(num) == maxFreq)
            ans.add(num);
    }

    int[] res = new int[ans.size()];
    for(int i = 0; i < ans.size(); i++)
        res[i] = ans.get(i);

    return res;
}

public int postOrder(TreeNode root){
    if(root == null)
        return 0;

    int leftSum  = postOrder(root.left);
    int rightSum = postOrder(root.right);

    int val = root.val + leftSum + rightSum;
    map.put(val, map.getOrDefault(val, 0) + 1);
    maxFreq = Math.max(maxFreq, map.get(val));

    return val;
}
```

509 Fibonacci Number

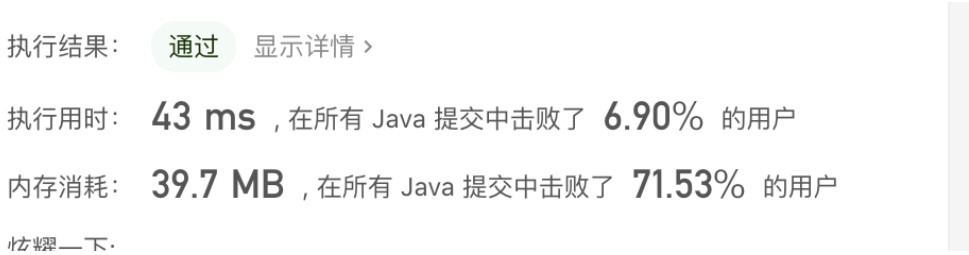
```go
func fib(n int) int {
  m := make(map[int]int)
  m[0] = 0
  m[1] = 1
  return dfs(n, &m)
}

func dfs(n int, m *map[int]int) int{
  if _, ok := (*m)[n]; ok{
    return (*m)[n]
  }

  res := 0
  res += dfs(n - 1, m) + dfs(n - 2, m)

  (*m)[n] = res
  return res
}
```




```java
    public int fib(int n) {
        if(n <= 1)
            return n;

        int a = 0;
        int b = 1;
        /*
            F(2) = a + b = 1;
                a = b  为1;
            F(3) = a + b = 2;
                b = 2; a = 1;
            F(4) = a + b = 3;
                b = 3 a = 2
        */

```

```
        for(int i = 2; i <= n; i++){
            int temp = a + b;
            a = b;
            b = temp;
        }

        return b;
    }
```

510 Inorder Successor In BST II




```
/*
    暴力解法
*/
List<Node> list = new ArrayList<>();
    public Node inorderSuccessor(Node node) {
        Node ancestor = node.parent;
        while(ancestor != null &&  ancestor.parent != null)
            ancestor = ancestor.parent;

        if(ancestor == null){
            inorder(node);
        }else{
            inorder(ancestor);
        }

        for(int i = 0; i < list.size(); i++)
            if(node.val == list.get(i).val && i < list.size() - 1)
                return list.get(i + 1);

        return null;
    }

    public void inorder(Node node){
        if(node == null)
            return;
```

```
        inorder(node.left);

        list.add(node);

        inorder(node.right);
    }
```

```
/*
    优化解法
*/
public Node inorderSuccessor(Node node) {
        if(node == null)
            return null;

        if(node.right == null){
            while(node.parent != null && node.parent.right == node)
                node = node.parent;

            return node.parent;
        }else{ // node.right != null
            node = node.right;
            while(node.left != null)
                node = node.left;
            return node;
        }
    }
```

513 Find Bottom Left Tree Value

执行结果： 通过 显示详情 >

执行用时： 2 ms , 在所有 Java 提交中击败了 68.84% 的用户

内存消耗： 37.9 MB , 在所有 Java 提交中击败了 93.05% 的用户

```
   public int findBottomLeftValue(TreeNode root) {
        Deque<TreeNode> queue = new ArrayDeque<>();
```

```
        queue.add(root);

        while(!queue.isEmpty()){
            int size = queue.size();
            boolean lastLevel = true;
            int target = 0;
            for(int i = 0; i < size; i++){
                TreeNode cur = queue.pollFirst();
                if(i == 0)
                    target = cur.val;
                if(cur.left != null){
                    queue.addLast(cur.left);
                    lastLevel = false;
                }
                if(cur.right != null){
                    queue.addLast(cur.right);
                    lastLevel = false;
                }
            }

            if(lastLevel)
                return target;
        }

        return -1;
    }
```

514 Freedom Trail

514. Freedom Trail

难度 困难 184

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.

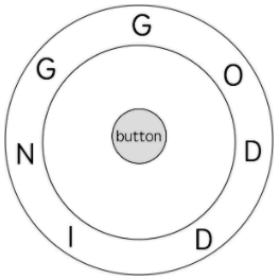
Given a string **ring**, which represents the code engraved on the outer ring and another string **key**, which represents the keyword needs to be spelled. You need to find the **minimum** number of steps in order to spell all the characters in the keyword.

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:



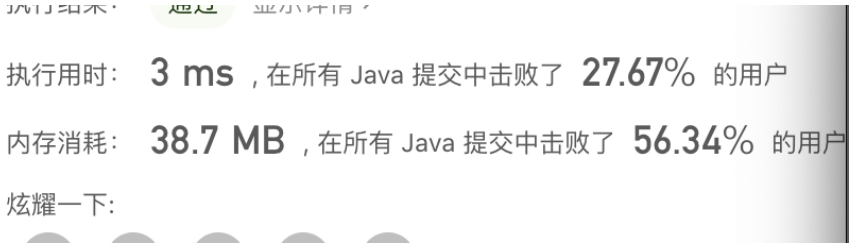

```
/*
  source:  https://www.youtube.com/watch?v=1ErNzKfRrX8

  算法分为三步
    1 记录下每个char 出现的位置

    2 对于每一个cur char,  通过上一个potential candidate  prevPos 的旋转差值
*/
int findRotateSteps(string ring, string key){
        unordered_map<char, vector<int>> letter2pos;

        for(int i = 0; i < ring.size(); i++)
            letter2pos[ring[i]].push_back(i);
```

```
14
15          int n = key.size(), m = ring.size();
16      /*
17          dp[i][j] 代表第 i 轮下 旋转到 该char 下从 j的位置，最小步数
18
19              i 代表的是第几轮了，以key 为标准
20              j 代表的是对于该轮次下， 该char的对应位置
21
22      */
23          vector<vector<int>> dp = vector<vector<int>>(n, vector<int>(m, INT_MAX / 2));
24
25
26          for(int i = 0; i < n; i++){
27              if(i == 0){
28                  for(int curPos : letter2pos[key[i]])
29                      //              或者顺时针，或者逆时针旋转
30                      dp[i][curPos] = min(curPos, abs(m - curPos));
31              }else{
32                  for(int curPos : letter2pos[key[i]])
33                      for(int prePos : letter2pos[key[i - 1]])
34                          dp[i][curPos] = min(dp[i][curPos], dp[i - 1][prePos] + min(abs(curPos - prePos), m - abs(curPos - prePos)));
35              }
36          }
37
38          int res = INT_MAX;
39          for(int pos : letter2pos[key[n - 1]])
40              res = min(res, dp[n - 1][pos]);
41
42          return res + n;
43      }
```

```
1       public int findRotateSteps(String ring, String key) {
2           Map<Character, List<Integer>> map = new HashMap<>();
3
4           for(int i = 0; i < ring.length(); i++){
5               char ch = ring.charAt(i);
6               map.putIfAbsent(ch, new ArrayList<>());
7               map.get(ch).add(i);
8           }
9           int m = ring.length();
10          int n = key.length();
11
12          int[][] dp = new int[n][m];
13          for(int i = 0; i < n; i++)
14              Arrays.fill(dp[i], Integer.MAX_VALUE);
```

```
        for(int i = 0; i < n; i++){
            if(i == 0){
                for(Integer curPos : map.get(key.charAt(i)))
                    dp[i][curPos] = Math.min(Math.abs(m - curPos), curPos);
            }else{
                for(Integer curPos : map.get(key.charAt(i)))
                    for(Integer prevPos : map.get(key.charAt(i - 1))){
                        dp[i][curPos] = Math.min(dp[i][curPos], dp[i - 1][prevPos]
+ Math.min(Math.abs(curPos - prevPos), m - Math.abs(curPos - prevPos)));
                    }
            }
        }

        int res = Integer.MAX_VALUE;
        for(Integer pos : map.get(key.charAt(n - 1)))
            res = Math.min(res, dp[n - 1][pos]);
        return res + n;
    }
```

515 Find Largest Value in Each Tree Row




```
class Solution {
    public List<Integer> largestValues(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root == null)
            return res;

        Deque<TreeNode> queue = new ArrayDeque<>();
        queue.add(root);

        while(!queue.isEmpty()){
            int size = queue.size();
```

```
            int max = Integer.MIN_VALUE;
            for(int i = 0; i < size; i++){
                TreeNode cur = queue.pollFirst();
                max = Math.max(cur.val, max);

                if(cur.left != null)
                    queue.addLast(cur.left);
                if(cur.right != null)
                    queue.addLast(cur.right);
            }

            res.add(max);
        }

        return res;
    }
}
```

516 Longest Palindromic Subsequence 不错的DP题目

516. Longest Palindromic Subsequence

难度 中等 384

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".

```java
/*
    暴力回溯，  时间复杂度 O(2 ^ n)

    通过 61 / 83
*/
    int res = 1;
    public int longestPalindromeSubseq(String s) {
        if(s == null || s.length() == 0)
            return 0;

        dfs(s, new StringBuilder(), 0);
        return res;
    }

    public void dfs(String s, StringBuilder sb, int start){
        if(isPalindromic(sb.toString()))
            res = Math.max(res, sb.length());
```

```
        if(start == s.length())
            return;


        dfs(s, new StringBuilder(sb), start + 1);
        sb.append(s.charAt(start));
        dfs(s, new StringBuilder(sb), start + 1);
    }

    public boolean isPalindromic(String sb){
        int left = 0, right = sb.length() - 1;

        while(left <= right){
            if(sb.charAt(left) != sb.charAt(right))
                return false;
            left++;
            right--;
        }

        return true;
    }
```

```
/*
   采用 dp
   dp 定义： 在子串 s[i...j] 中，最长回文子序列的长度为dp[i][j]

   状态转移需要归纳思维，如何从已知推出未知
   从而定义容易归纳，容易发现状态转移方程
*/

    public int longestPalindromeSubseq(String s) {
        int len = s.length();
        /*
            dp[i][j] means
                s[i...j] 中最长的串子长度
        */
        if(s == null || s.length() == 0)
            return 0;

        int res = 1;
        int[][] dp = new int[len][len];
        for(int i = 0; i < len; i++)
            dp[i][i] = 1;
        for(int i = len - 1; i >= 0; i--){
            for(int j = i + 1; j < len; j++){
                if(s.charAt(i) == s.charAt(j))
                    dp[i][j] = (j - i == 2 ? 3 : dp[i + 1][j - 1] + 2);
```

```
                else
                    dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);

                res = Math.max(res, dp[i][j]);
            }
        }

        return res;
    }
```

517 Super Washing Machines

517. Super Washing Machines

难度 困难 57

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

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

Given an integer array representing the number of dresses in eac
machine from left to right on the line, you should find the **minimu
of moves** to make all the washing machines have the same numb
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
```

```java
/*
  source:
    https://www.youtube.com/watch?v=QNAGlcqu0nA&t=4s
    https://leetcode-cn.com/problems/super-washing-machines/solution/csi-lu-by-beny-g/

    本质思路类似于水流的流入与流出
    拿到每个点，最大的净输出即可

    因此定义两个数组， left / right

    有如下关系式
    left[i] + right[i] = machiens[i] - k;
    right[i] = -left[i + 1]

    定义 输出为 负， 输入为正
*/

    public int findMinMoves(int[] machines) {
        int sum = 0;
        int avg = 0;
        int len = machines.length;
        for(int num : machines)
            sum += num;
        if(sum % len != 0)
            return -1;
        avg = sum / len;

        int[] left = new int[len];
        int[] right = new int[len];

        for(int i = 0; i < len; i++){
            if(i == 0){
                left[i] = 0;
                right[i] = machines[i] - avg;
            }else if(i == len - 1){
                left[i] = machines[i] - avg - right[i];
                right[i] = 0;
            }else{
                left[i] = -right[i - 1];
                right[i] = machines[i] - avg - left[i];
            }
        }

        int res = 0;
        for(int i = 0; i < len; i++){
            int temp = 0;
            if(left[i] > 0)
                temp += left[i];
```

```
            if(right[i] > 0)
                temp += right[i];
            res = Math.max(res, temp);
        }

        return res;
    }
```

519 Random Flip Matrix 未完成

519. Random Flip Matrix

难度 中等 33

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:

```
1
```

520 Detect Capital

执行用时： 1 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 36.8 MB , 在所有 Java 提交中击败了 56.53% 的用户

```
1
```

```java
    public boolean detectCapitalUse(String word) {
        if(word == null || word.length() == 0)
            return true;

        boolean firstUpper = upperCase(word.charAt(0));

        if(firstUpper){
            if(word.length() >= 2){
                if(upperCase(word.charAt(1))){
                    for(int i = 2; i < word.length(); i++)
                        if(!upperCase(word.charAt(i)))
                            return false;

                    return true;
                }else{
                    for(int i = 2; i < word.length(); i++)
                        if(upperCase(word.charAt(i)))
                            return false;

                    return true;
                }
            }else{
                return true;
            }

        }else{
            for(int i = 1; i < word.length(); i++){
                char ch = word.charAt(i);
                if(upperCase(ch))
                    return false;
            }

            return true;
        }
    }

    public boolean upperCase(char ch){
        return 'A' <= ch && ch <= 'Z';
    }
```

521 Longest Uncommon Subsequence I脑经急转弯题目

521. Longest Uncommon Subsequence I

难度 简单 89

Given two strings `a` and `b`, find the length of the **longest uncommon subsequence** between them.

A **subsequence** of a string `s` is a string that can be obtained after deleting any number of characters from `s`. For example, `"abc"` is a subsequence of `"aebdc"` because you can delete the underlined characters in `"aebdc"` to get `"abc"`. Other subsequences of `"aebdc"` include `"aebdc"`, `"aeb"`, and `""` (empty string).

An **uncommon subsequence** between two strings is a string that is a **subsequence of one but not the other**.

Return *the length of the **longest uncommon subsequence** between* `a` *and* `b`. If the longest uncommon subsequence doesn't exist, return `-1`.

Example 1:

```
Input: a = "aba", b = "cdc"
Output: 3
Explanation: One longest uncommon subsequence is
"aba" because "aba" is a subsequence of "aba" but
not "cdc".
Note that "cdc" is also a longest uncommon
subsequence.
```

Example 2:

```java
public int findLUSlength(String a, String b) {
    if(a.equals(b))
        return -1;

    return a.length() > b.length() ? a.length() : b.length();
}
```

522 Longest Uncommon Susbequence II

522. Longest Uncommon Subsequence II

难度 中等 56

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].

```java
/*
  ref: https://leetcode.com/problems/longest-uncommon-subsequence-ii/discuss/99443/Java(15ms)-Sort-%2B-check-subsequence
*/
public int findLUSlength(String[] strs) {
        Arrays.sort(strs, (o1, o2) -> o2.length() - o1.length());
        Set<String> duplicates = getDup(strs);

        for(int i = 0; i < strs.length; i++){
            if(!duplicates.contains(strs[i])){
                if(i == 0)
                    return strs[0].length();
                for(int j = 0; j < i; j++){
                    if(isSubsequence(strs[j], strs[i]))
                        break;
                    if(j == i - 1)
                        return strs[i].length();
                }
            }
```

```
        }

        return -1;
    }
/*
    判断 b 是否 是 a的一个子序列，方法不错
*/
    public boolean isSubsequence(String a, String b){
        int i = 0, j = 0;
        while(i < a.length() && j < b.length()){
            if(a.charAt(i) == b.charAt(j))
                j++;
            i++;
        }

        return j == b.length();
    }

    private Set<String> getDup(String[] strs){
        Set<String> set = new HashSet<>();
        Set<String> dup = new HashSet<>();

        for(String s : strs){
            if(set.contains(s))
                dup.add(s);
            set.add(s);
        }

        return dup;
    }
}

```

523 Continuous Subarray Sum

执行结果： 通过 显示详情 >

执行用时： 25 ms ，在所有 Java 提交中击败了 18.66% 的用户

内存消耗： 39 MB ，在所有 Java 提交中击败了 85.79% 的用户

```java
    public boolean checkSubarraySum(int[] nums, int k) {

        for(int i = 1; i < nums.length; i++)
            nums[i] += nums[i - 1];

        for(int i = 0; i < nums.length - 1; i++){
            for(int j = i + 1; j <= nums.length; j++){
                int target = 0;
                if(j == nums.length)
                    target = nums[j - 1];
                else
                    target = nums[j] - nums[i];

                boolean val = findSuitableKVal(target, k);
                if(val && j - i > 1)
                    return true;
            }
        }

        return false;
    }

    public boolean findSuitableKVal(int target, int k){
        if(k == 0)
            return target == 0;

        return target % k == 0;
    }
```

```java
//简化写法
public class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {
        int[] sum = new int[nums.length];
        sum[0] = nums[0];
        for (int i = 1; i < nums.length; i++)
            sum[i] = sum[i - 1] + nums[i];
        for (int start = 0; start < nums.length - 1; start++) {
            for (int end = start + 1; end < nums.length; end++) {
                int summ = sum[end] - sum[start] + nums[start];
                if (summ == k || (k != 0 && summ % k == 0))
                    return true;
```

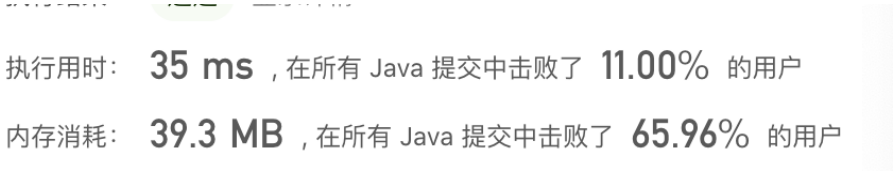
```
            }
        }
        return false;
    }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/continuous-subarray-sum/solution/lian-xu-de-zi-shu-zu-he-by-leetcode/
```

空间换时间

```
/*
  这种方法的思路是
   假设 sum[:i]的和为 n * k + 余数
   假设 sum[;j]的和为 m * k + 余数

   只要两个的余数是一致的，那么sum[j] - sum[i] 就可以被 k 整除

   最后记得一个特例[0,0] 0
   需要map提前处理
*/
public class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {
        int sum = 0;
        HashMap<Integer, Integer> map = new HashMap < > ();
        map.put(0, -1);
        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            if (k != 0)
                sum = sum % k;
            if (map.containsKey(sum)) {
                if (i - map.get(sum) > 1)
                    return true;
            } else
                map.put(sum, i);
        }

        return false;
    }
}
```

```
作者：LeetCode
链接：https://leetcode-cn.com/problems/continuous-subarray-sum/solution/lian-xu-de-zi-shu-zu-he-by-leetcode/
```

524 Longest Word In Dictionary Through Deleting




```java
/*
  思路很简单，就是将其排序，从大到小
  然后判断每一个string 是否是 s的子串，将所有的candidates 加入到List中

  最后sort 它们
*/
  public String findLongestWord(String s, List<String> d) {
        TreeMap<Integer, List<String>> map = new TreeMap<>((o1, o2) -> o2 - o1);
        for(String str : d){
            map.putIfAbsent(str.length(), new ArrayList<>());
            map.get(str.length()).add(str);
        }


        Iterator<Integer> it = map.keySet().iterator();
        List<String> candidate = new ArrayList<>();
        while(it.hasNext()){
            Integer len = it.next();
            List<String> sameLenCollection = map.get(len);
            for(String str : sameLenCollection){
                if(isSubsequence(s, str))
                    candidate.add(str);
            }

            if(candidate.size() >= 1)
                break;
        }

        if(candidate.size() == 0)
            return "";
        Collections.sort(candidate);
```

```
        return candidate.get(0);
    }

    /**
     * to justify whether B is a subsequence of A
     * @param a the relatively long string
     * @param b the shorter one
     * @return true or not
     */
    private boolean isSubsequence(String a, String b){
        int i = 0, j = 0;
        while(i < a.length() && j < b.length()){
            if(a.charAt(i) == b.charAt(j))
                j++;
            i++;
        }

        return j == b.length();
    }
```

525 Contiguous Array

525. Contiguous Array

难度 中等 230

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.
```

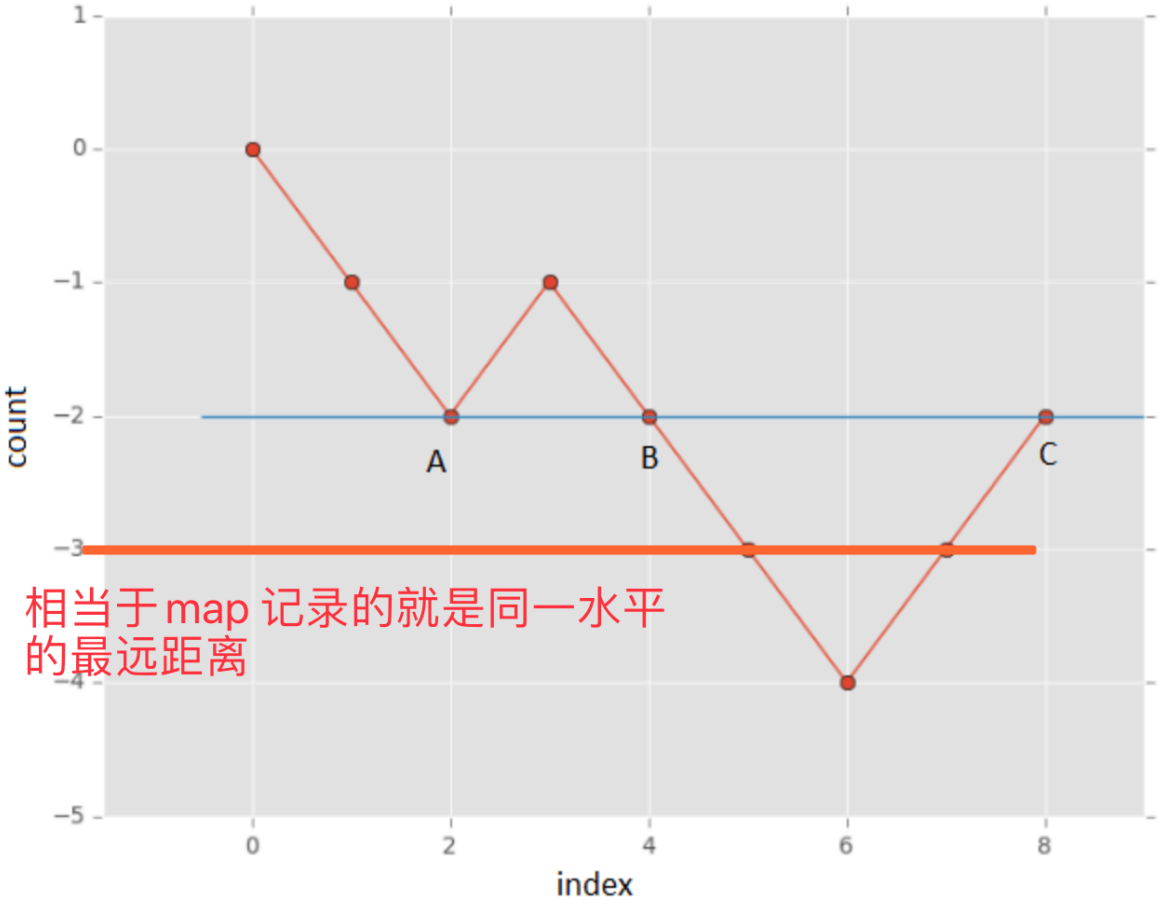
Example 2:

```
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.

```java
/*
    暴力解法，时间复杂度 O(N^2)
*/
    public int findMaxLength(int[] nums) {
        /*
            0 1 0 1 0 0 1 1 0 0 1 1
        */
            //  index,   #0 and #1
        HashMap<Integer, String> map = new HashMap<>();
        map.put(-1, 0 + "@" + 0);
        int zero = 0;
        int one  = 0;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] == 0)
                zero++;
            else
                one++;

            map.put(i, zero + "@" + one);
        }

        int res = 0;
        for(int i = -1; i < nums.length; i++){
            for(int j = i + 1; j < nums.length; j++){
                if(i == -1){
                    String[] tmp = map.get(j).split("@");
                    int z = Integer.parseInt(tmp[0]);
                    int o = Integer.parseInt(tmp[1]);

                    if(z == o)
                        res = Math.max(res, z + o);
                }else{
                    String[] tmp1 = map.get(i).split("@");
                    String[] tmp2 = map.get(j).split("@");

                    int z = Integer.parseInt(tmp2[0]) - Integer.parseInt(tmp1[0]);
                    int o = Integer.parseInt(tmp2[1]) - Integer.parseInt(tmp1[1]);
                    if(z == o)
                        res = Math.max(res, z + o);
                }
            }
        }

        return res;
    }
```

贼几把类似 523




```java
public class Solution {

    public int findMaxLength(int[] nums) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, -1);
        int maxlen = 0, count = 0;
        for (int i = 0; i < nums.length; i++) {
            count = count + (nums[i] == 1 ? 1 : -1);
            if (map.containsKey(count)) {
                maxlen = Math.max(maxlen, i - map.get(count));
            } else {
                map.put(count, i);
            }
        }
        return maxlen;
    }
}


作者：LeetCode
```

```
链接：https://leetcode-cn.com/problems/contiguous-array/solution/lian-xu-shu-zu-by-leetcode/
```

526 Beautiful Arrangement

526. Beautiful Arrangement

难度 中等 120

Suppose you have `n` integers labeled `1` through `n`. A permutation of those `n` integers `perm` (**1-indexed**) is considered a **beautiful arrangement** if for every `i` (`1 <= i <= n`), **either** of the following is true:

- `perm[i]` is divisible by `i` .
- `i` is divisible by `perm[i]` .

Given an integer `n` , return *the **number** of the **beautiful arrangements** that you can construct*.

Example 1:

```
Input: n = 2
Output: 2
Explanation:
The first beautiful arrangement is [1,2]:
    - perm[1] = 1 is divisible by i = 1
    - perm[2] = 2 is divisible by i = 2
The second beautiful arrangement is [2,1]:
    - perm[1] = 2 is divisible by i = 1
    - i = 2 is divisible by perm[2] = 1
```

Example 2:

```
Input: n = 1
Output: 1
```

```java
    int count = 0;
    public int countArrangement(int n) {
        backtrack(n, 0, new HashSet<>(), new StringBuilder());
        return count;
    }


    private void backtrack(int n, int start, HashSet<Integer> visited,
StringBuilder path) {
        if(start == n) {
            count++;
            return;
        }

        for(int i = 1; i <= n; i++){
            if(visited.contains(i))
                continue;
            //start + 1;
            boolean flag = false;
            if(i > start + 1){
                if(i % (start + 1) == 0)
                    flag = true;
            }else{
                if((start + 1) % i == 0)
                    flag = true;
            }

            if(flag == false)
                continue;

            int len = path.length();
//            path.append(i).append();
            visited.add(i);
            backtrack(n, start + 1, visited, path);
            visited.remove(i);
//            path.setLength(len);
        }
    }
```

527 Word Abbreviation

执行结果： 通过 查看详情

执行用时： 38 ms ，在所有 Java 提交中击败了 66.67% 的用户

内存消耗： 40 MB ，在所有 Java 提交中击败了 53.85% 的用户

炫耀一下:

```java
/*
    采用全局 lookup 的映射，实现对dict中每个String的更新
*/
HashMap<String, String>  lookup = new HashMap<>();
    public List<String> wordsAbbreviation(List<String> dict) {
        wordsAbbreviation(dict, 0);
        List<String> res = new ArrayList<>();
        for(int i = 0; i < dict.size(); i++){
            res.add(lookup.get(dict.get(i)));
        }
        return res;
    }

    public void wordsAbbreviation(List<String> dict, int start){

        //  key 为缩写， String 为 value
        HashMap<String, List<String>> group = new HashMap<>();

      /*
          将他们分组，
      */
        List<String> counterpart = new ArrayList<>();
        for (int i = 0; i < dict.size(); i++) {
            String s    = dict.get(i);
            String abbr = generateAbbr(s, 0);
            group.putIfAbsent(abbr, new ArrayList<>());
            group.get(abbr).add(s);

            lookup.put(s, abbr);
        }

        for(String str : group.keySet()){
            List<String> strs = group.get(str);
            if(strs.size() == 1)
                continue;

            HashMap<String, List<String>> reused = new HashMap<>();
            for(String string : strs){
                String abbr = generateAbbr(string, start);
                reused.putIfAbsent(abbr, new ArrayList<>());
                reused.get(abbr).add(string);
                lookup.put(string, abbr);
            }
```

```
44          //如果都没有重复，直接break
45                if(reused.size() == strs.size())
46                    break;
47                else{
48                  //将没有重复的直接去除，只递归有重复的
49                    for(String toCheck : reused.keySet()){
50                        if(reused.get(toCheck).size() != 1)
51                            wordsAbbreviation(reused.get(toCheck), start + 1);
52                    }
53
54                }
55            }
56
57        }
58
59    //负责产生所需要的缩写
60        private String generateAbbr(String s, int start) {
61            int left = start + 1, right = s.length() - 2;
62            int len = right - left + 1;
63            if((len + "").length() >= len)
64                return s;
65
66            return s.substring(0, left) + len + s.substring(left + len);
67        }
68
69    /*
70      li1e
71    */
```

528 Random Pick with Weight

528. Random Pick with Weight

难度 中等 80

You are given an array of positive integers `w` where `w[i]` des
weight of `i th` index (0-indexed).

We need to call the function `pickIndex()` which **randomly** re
integer in the range `[0, w.length - 1]`. `pickIndex()` sh
integer proportional to its weight in the `w` array. For example, f
`3]`, the probability of picking the index `0` is `1 / (1 + 3) =`
25%) while the probability of picking the index `1` is `3 / (1 +`
(i.e 75%).

More formally, the probability of picking index `i` is `w[i] / su`

Example 1:

```
Input
["Solution","pickIndex"]
[[[1]],[]]
Output
[null,0]

Explanation
Solution solution = new Solution([1]);
solution.pickIndex(); // return 0. Since there
one single element on the array the only option
return the first element.
```

Example 2:

```
1
2
```

529 Mine Sweeper

```
1    int[][] direction = {{1,0},{0,1},{-1,0},{0,-1}};
2    int[][] check = {{1,0},{0,1},{-1,0},{0,-1},{1,1},{1,-1},{-1,-1},{-1,1}};
```

```
    char[][] board;
    public char[][] updateBoard(char[][] board, int[] click) {
        if(board[click[0]][click[1]] == 'M'){
            board[click[0]][click[1]] = 'X';
            return board;
        }
        this.board = board;
        dfs(click[0], click[1]);
        return board;
    }

    private void dfs(int r, int c){
        if(isInRange(r,c) && board[r][c] == 'E'){
            int numOfMines = numMinesAround(r, c);

            if(numOfMines != 0){
                board[r][c] = ((char)('0' + numOfMines));
                return;
            }else{
                board[r][c] = 'B';


            for(int k = 0; k < check.length; k++){
                int newX = r + check[k][0];
                int newY = c + check[k][1];

                if(isInRange(newX, newY) && board[newX][newY] == 'E')
                    dfs(newX, newY);
            }

        }
    }

    private int numMinesAround(int r,int c){
        int res = 0;
        for(int k = 0; k < check.length; k++){
            int newX = r + check[k][0];
            int newY = c + check[k][1];

            if(isInRange(newX, newY) && board[newX][newY] == 'M')
                res++;
        }

        return res;
    }

    private boolean isInRange(int r, int c){
        return r >= 0 && c >= 0 && r < board.length && c < board[0].length;
    }
```

```
52
```

530 Minimum Absolute Difference

> 执行用时： 1 ms ，在所有 Java 提交中击败了 71.93% 的用户
>
> 内存消耗： 38.1 MB ，在所有 Java 提交中击败了 66.62% 的用户

```java
    TreeNode pre = null;
    long res = Integer.MAX_VALUE;
    public int getMinimumDifference(TreeNode root) {
        inorder(root);

        return (int)res;
    }

    public void inorder(TreeNode root){
        if(root == null)
            return;

        inorder(root.left);

        if(pre != null)
            res = Math.min(res, Math.abs((long)(root.val) - (long)(pre.val)));

        pre = root;

        inorder(root.right);
    }
```

531 Lonely Pixel I

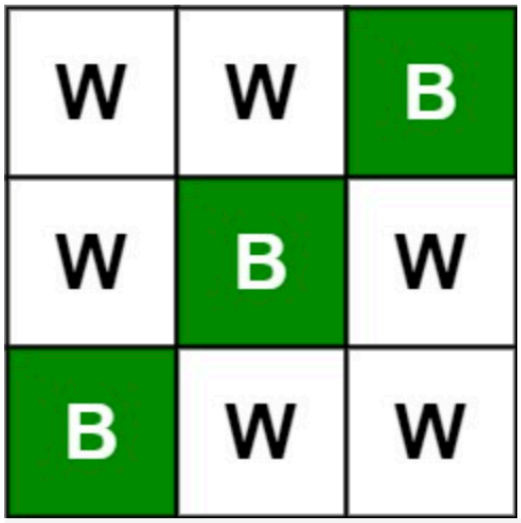
531. Lonely Pixel I

难度 中等 20

Given an `m x n` `picture` consisting of black `'B'` and white `'W'` pixels, return *the number of **black** lonely pixels*.

A black lonely pixel is a character `'B'` that located at a specific position where the same row and same column don't have **any other** black pixels.

Example 1:




```
Input: picture = [["W","W","B"],["W","B","W"],
["B","W","W"]]
Output: 3
Explanation: All the three 'B's are black lonely pixels.
```

执行用时： 2 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 43.4 MB ，在所有 Java 提交中击败了 5.51% 的用户

```java

public int findLonelyPixel(char[][] picture) {
    int res = 0;
    int row = picture.length;
    int col = picture[0].length;
    int[] countRow = new int[row];
    int[] countCol = new int[col];

    for(int i = 0; i < row; i++)
        for(int j = 0; j < col; j++){
            if(picture[i][j] == 'B'){
                countRow[i]++;
```

```
                    countCol[j]++;
                }
            }

        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(picture[i][j] == 'B' && countCol[j] == 1 && countRow[i] == 1)
                    res++;
            }
    }

        return res;
    }
```

532 K-diff Pairs in an Array

532. K-diff Pairs in an Array

难度 中等 116

Given an array of integers `nums` and an integer `k`, return *the number of **unique** k-diff pairs in the array*.

A **k-diff** pair is an integer pair `(nums[i], nums[j])`, where the following are true:

- `0 <= i, j < nums.length`
- `i != j`
- `|nums[i] - nums[j]| == k`

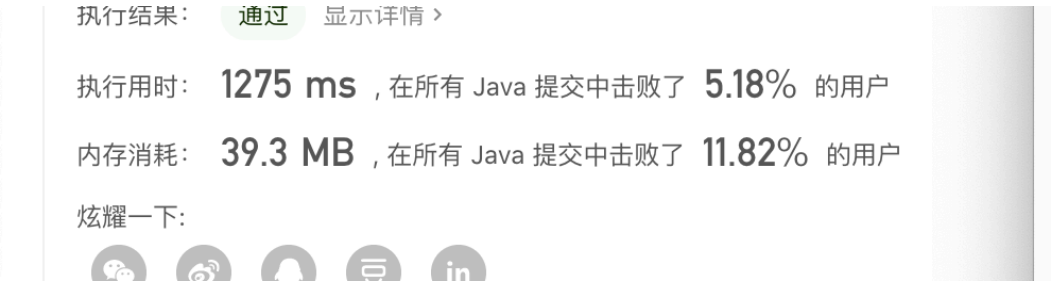
Notice that `|val|` denotes the absolute value of `val`.

Example 1:

```
Input: nums = [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: nums = [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).
```




```java
/*
    O(N2)的时间复杂度
*/
  public int findPairs(int[] nums, int k) {
        HashSet<String> set = new HashSet<>();
        int count = 0;
        for(int i = 0; i < nums.length; i++){
            for(int j = 0; j < nums.length; j++){
```

```
                if(i == j)
                    continue;
                if(Math.abs(nums[i] - nums[j]) == k){
                    if(nums[i] > nums[j])
                        set.add(nums[i] + "@" + nums[j]);
                    else
                        set.add(nums[j] + "@" + nums[i]);
                }
            }
        }
//         System.out.println(set);

        return set.size();

    }
```

执行用时： 4 ms ，在所有 Java 提交中击败了 99.79% 的用户

内存消耗： 38.3 MB ，在所有 Java 提交中击败了 80.71% 的用户

```
/*
    O(nlogn) 时间复杂度
*/
public int findPairs(int[] nums, int k) {
        int count = 0;
        int left = 0, right = 1;
        Arrays.sort(nums);

        while(right < nums.length){
            if(left == right) {
                right++;
                continue;
            }
            int diff = nums[right] - nums[left];
            if(diff == k){
                count++;
                while(right < nums.length - 1 && nums[right] == nums[right + 1])
                    right++;
                right++;
                while(left < nums.length - 1  && nums[left] == nums[left + 1])
                    left++;
                left++;
            }
            else if(diff < k)
                right++;
            else
```

```
                left++;

            }

            return count;
        }

```

533 Lonely Pixel II

533. Lonely Pixel II

难度 中等 14

Given a picture consisting of black and white pixels, and a positive integer N, find the number of black pixels located at some specific row **R** and column **C** that align with all the following rules:

1. Row R and column C both contain exactly N black pixels.
2. For all rows that have a black pixel at column C, they should be exactly the same as row R

The picture is represented by a 2D char array consisting of 'B' and 'W', which means black and white pixels respectively.

Example:

```
Input:
[['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'W', 'B', 'W', 'B', 'W']]

N = 3
Output: 6
Explanation: All the bold 'B' are the black pixels we
need (all 'B's at column 1 and 3).
        0    1    2    3    4    5         column index
0    [['W', 'B', 'W', 'B', 'B', 'W'],
1     ['W', 'B', 'W', 'B', 'B', 'W'],
2     ['W', 'B', 'W', 'B', 'B', 'W'],
3     ['W', 'W', 'B', 'W', 'B', 'W']]
row index
```

执行用时： 9 ms ，在所有 Java 提交中击败了 30.36% 的用户

内存消耗： 40.6 MB ，在所有 Java 提交中击败了 5.36% 的用户

```java
public int findBlackPixel(char[][] picture, int N) {
    int row = picture.length;
    int col = row == 0 ? 0 : picture[0].length;
    if(col == 0)
        return 0;

    List<String> rowString = new ArrayList<>();
    int[] numBlackRow = new int[row];
    int[] numBlackCol = new int[col];

    for(int i = 0; i < row; i++){
        StringBuilder sb = new StringBuilder();

        for(int j = 0; j < col; j++){
            if(picture[i][j] == 'B'){
                numBlackRow[i]++;
                numBlackCol[j]++;
            }

            sb.append(picture[i][j]);
        }

        rowString.add(sb.toString());
    }

    int count = 0;
    HashSet<Integer> colVisited = new HashSet<>();
    for(int j = 0; j < col; j++){
        if(numBlackCol[j] != N)
            continue;
        for(int i = 0; i < row; i++){
            if(numBlackRow[i] != numBlackCol[j] || colVisited.contains(j))
                continue;

            HashSet<String> visited = new HashSet<>();
            for(int k = 0; k < row; k++){
                if(picture[k][j] == 'B')
                    visited.add(rowString.get(k));
            }

            if(visited.size() == 1){
                count += numBlackCol[j];
                colVisited.add(j);
            }
```

```
            }
        }

        return count;
    }
```

535 TinyURL

535. Encode and Decode TinyURL

难度 中等 111

> 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.

> 执行用时： **7 ms** , 在所有 Java 提交中击败了 **34.17%** 的用户
>
> 内存消耗： **38.4 MB** , 在所有 Java 提交中击败了 **85.71%** 的用户

```java
/*
    当然这种方法存在 弊端
    1. 如果被 asked 编码same URL 多次，会产生不同的entries， 这样会浪费空间
    2. 容易被别人猜出来我们用了多少次
    3. 采用 6 位数字只能产生 a million codes
       使用 混合6位数字 + alpha可以产生 (10 + 26 * 2)^6  = 56,800,235,584
*/
public class Codec {
    HashMap<Integer, String> map = new HashMap<>();
    int count = 0;
    final String BASE_HOST = "http://tinyurl.com/";
    // Encodes a URL to a shortened URL.
    public String encode(String longUrl) {
        String temp = BASE_HOST + count;
```

```
        map.put(count, longUrl);
        count++;
        return temp;
    }

    // Decodes a shortened URL to its original URL.
    public String decode(String shortUrl) {
        int pos = shortUrl.indexOf("http://tinyurl.com/");
        int num = Integer.parseInt(shortUrl.substring(19));
        return map.get(num);
    }
}

// Your Codec object will be instantiated and called as such:
// Codec codec = new Codec();
// codec.decode(codec.encode(url));
```

```
/*
    正经方法
*/
//      index    longUrl
    HashMap<String, String> index    = new HashMap<>();
    //      longUrl   index
    HashMap<String, String> revIndex = new HashMap<>();
    final String charSet = "abcdefghijklmnopqrstuvwxyz0123456789";
    final String BASE_HOST = "http://tinyurl.com/4e9iAk";
    // Encodes a URL to a shortened URL.
    public String encode(String longUrl) {
        if(revIndex.containsKey(longUrl))
            return BASE_HOST + revIndex.get(longUrl);

        StringBuilder sb = new StringBuilder();
        while(true){
            for(int i = 0; i < 6; i++){
                int pos = (int)(Math.random() * charSet.length());
                sb.append(charSet.charAt(pos));
            }

            if(!index.containsKey(sb.toString()))
                break;
        }
        String key = sb.toString();
        index.put(key, longUrl);
        revIndex.put(longUrl, key);
        return BASE_HOST + key;
    }

```

```
31      // Decodes a shortened URL to its original URL.
32      public String decode(String shortUrl) {
33          return index.get(shortUrl.substring(shortUrl.indexOf(BASE_HOST) +
BASE_HOST.length()));
34
35      }
```

536 Construct Binary Tree From String

536. Construct Binary Tree from String

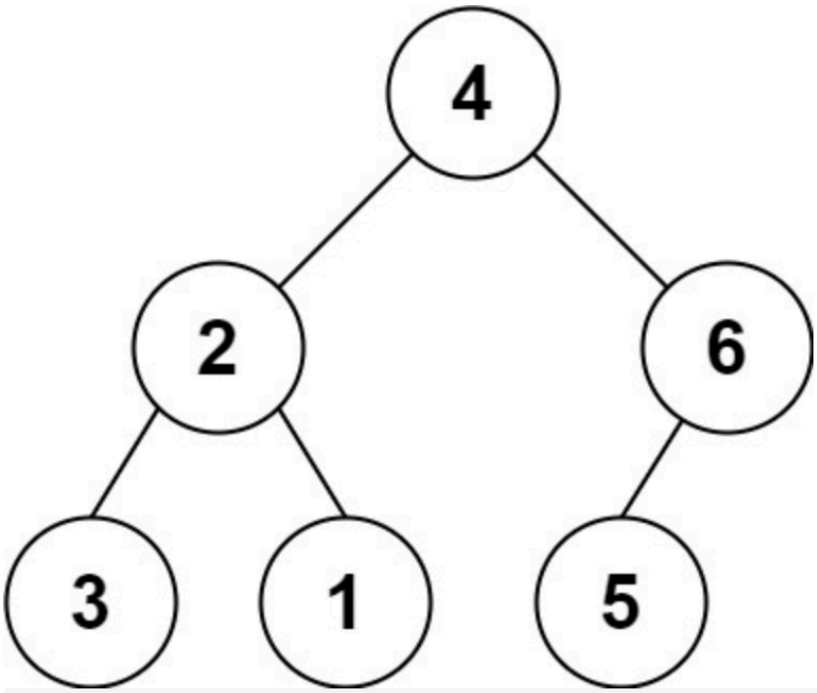
难度 中等 54

You need to construct a binary tree from a string consisting of parenthesis and integers.

The whole input represents a binary tree. It contains an integer followed by zero, one or two pairs of parenthesis. The integer represents the root's value and a pair of parenthesis contains a child binary tree with the same structure.

You always start to construct the **left** child node of the parent first if it exists.

Example 1:



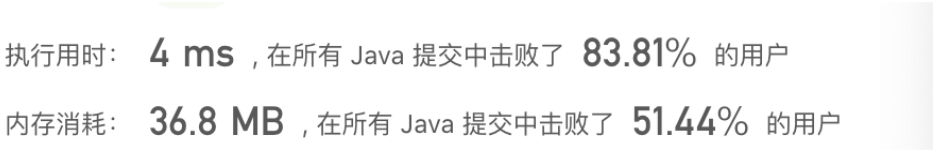
执行用时： 10 ms ，在所有 Java 提交中击败了 39.23% 的用户

内存消耗： 39.1 MB ，在所有 Java 提交中击败了 35.15% 的用户

```java
public TreeNode str2tree(String s) {
    if(s == null || s.length() == 0)
        return null;

    if(s.charAt(0) == '(' && s.charAt(s.length() - 1) == ')')
        s = s.substring(1, s.length() - 1);

    int leftParenthesis = 0;
    int index = 0;
    TreeNode root = null;
```

```
11
12          while(index < s.length()){
13              if(s.charAt(index) == '('){
14                  root = new TreeNode(Integer.parseInt(s.substring(0, index)));
15                  break;
16              }
17              index++;
18          }
19
20          if(index == s.length()){
21              root = new TreeNode(Integer.parseInt(s.substring(0, index)));
22              return root;
23          }
24
25          int leftBound = index;
26          while(index < s.length()){
27              if(s.charAt(index) == '(')
28                  leftParenthesis++;
29              else if(s.charAt(index) == ')')
30                  leftParenthesis--;
31
32
33              if(leftParenthesis == 0)
34                  break;
35              index++;
36          }
37
38          TreeNode leftNode  = str2tree(s.substring(leftBound + 1, index));
39          TreeNode rightNode = str2tree(s.substring(index + 1, s.length()));
40
41          root.left  = leftNode;
42          root.right =  rightNode;
43
44          return root;
45      }
```

537 Complex Number Multiplication



```java
    public String complexNumberMultiply(String a, String b) {
        int pos1 = a.indexOf('+');
        int pos2 = b.indexOf('+');

        int af   = Integer.parseInt(a.substring(0, pos1));
        int bf   = Integer.parseInt(b.substring(0, pos2));
        int al   = Integer.parseInt(a.substring(pos1 + 1, a.length() - 1));
        int bl   = Integer.parseInt(b.substring(pos2 + 1, b.length() - 1));

        int resf = af * bf - (al * bl);
        int refl = af * bl + al * bf;
        return resf + "+" + refl + "i";
    }
```

538 Convert BST to Greater Tree

执行用时： 1 ms , 在所有 Java 提交中击败了 84.79% 的用户

内存消耗： 38.9 MB , 在所有 Java 提交中击败了 22.52% 的用户

炫耀一下：

```java
/*
    推荐遍历顺序
    右 根 左
*/
class Solution {
    int sum = 0;
    public TreeNode convertBST(TreeNode root) {
        guoguoOrder(root);
        return root;
    }

    public void guoguoOrder(TreeNode root){
        if(root == null)
            return;

        guoguoOrder(root.right);

        int target = sum;
        sum      += root.val;
        root.val += target;

```

```
        guoguoOrder(root.left);
    }
}
```

539 Minimum Time Difference

> 执行用时： 18 ms ，在所有 Java 提交中击败了 16.73% 的用户
>
> 内存消耗： 40.2 MB ，在所有 Java 提交中击败了 21.56% 的用户

```
class Solution {
    public int findMinDifference(List<String> timePoints) {
        /*
            1.find the suitable way to sort

            2. check it one by one
        */
        Node beginNode = new Node(0,0);
        Node endNode   = new Node(24, 0);
        List<Node> list = new ArrayList<>();
        for(String str : timePoints){
            String[] strs = str.split(":");
            int hour = Integer.parseInt(strs[0]);
            int mins = Integer.parseInt(strs[1]);

            list.add(new Node(hour, mins));
        }

        list.sort(new Comparator<Node>() {
            @Override
            public int compare(Node o1, Node o2) {
                return o1.hour == o2.hour ? (o1.mins - o2.mins) : o1.hour - o2.hour;
            }
        });

        int minDiff = Integer.MAX_VALUE;
        for(int i = 0; i < list.size() - 1; i++){
            int curDiff = getDiff(list.get(i), list.get(i + 1));

            minDiff = Math.min(curDiff, minDiff);
        }

```

```
        minDiff = Math.min(minDiff, getDiff(beginNode, list.get(0)) +
getDiff(list.get(list.size() - 1), endNode));

        return minDiff;
    }

    //assume first is eariler then end
    private int getDiff(Node first, Node end) {
        int curDiffHour = end.hour - first.hour;
        int curDiffMIn  = 0;
        if(end.mins < first.mins){
            curDiffHour--;
            curDiffMIn = end.mins + 60  - first.mins;
        }else {
            curDiffMIn = end.mins - first.mins;
        }

        return curDiffHour * 60 + curDiffMIn;
    }

    public static void main(String[] args) {
        List<String> list = List.of("23:59", "00:00");

        (new Solution()).findMinDifference(list);
    }
}

class Node{
    public int hour;
    public int mins;

    public Node(int hour, int mins){
        this.hour = hour;
        this.mins = mins;
    }
}

```

540 Single Element in a Sortede Array

执行用时： 0 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 38.6 MB ，在所有 Java 提交中击败了 56.34% 的用户

```java
public int singleNonDuplicate(int[] nums) {
    int num = nums[0];
    for(int i = 1; i < nums.length; i++)
        num ^= nums[i];

    return num;
}
```

541 Reverse String II

541. Reverse String II

难度 简单 113

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 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]

执行用时： 3 ms ，在所有 Java 提交中击败了 17.14% 的用户

内存消耗： 38.8 MB ，在所有 Java 提交中击败了 10.43% 的用户

```java
    public String reverseStr(String s, int k) {
        char[] chars = s.toCharArray();

        int left = 0, right = 0;

        while(left < s.length()){
            right += k;
            int rightBound = Math.min(right - 1, s.length() - 1);
            reverse(chars, left, rightBound);

            left = right + k;
            right = left;
        }

        StringBuilder sb = new StringBuilder();
        for(char ch : chars)
            sb.append(ch);
        return sb.toString();
    }

    private void reverse(char[] chars, int left, int right){
        if(right < left)
            return;
        while(left < right){
            char ch      = chars[left];
            chars[left]  = chars[right];
            chars[right] = ch;
            left++;
            right--;
        }
    }
}
```

542 01 Matrix

执行用时： 191 ms ，在所有 Java 提交中击败了 5.34% 的用户

内存消耗： 46.9 MB ，在所有 Java 提交中击败了 5.07% 的用户

```java
/*
    实际上就是多点 BFS
*/
int[][] direction = {{1, 0}, {0, 1}, {-1, 0},{0, -1}};
    int row = 0;
    int col = 0;
    public int[][] updateMatrix(int[][] matrix) {
        this.row = matrix.length;
        this.col = row == 0 ? 0 :matrix[0].length;
        int[][] res = new int[row][col];

        HashSet<String> visited = new HashSet<>();
        Deque<String> queue = new ArrayDeque<>();
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(matrix[i][j] == 0)
                    queue.add(i + "@" + j);
            }
        }

        int count = 0;
        while(!queue.isEmpty() && visited.size() != row * col){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                String curStr = queue.pollFirst();
                if(visited.contains(curStr))
                    continue;

                String[] temp = curStr.split("@");
                int curX      = Integer.parseInt(temp[0]);
                int curY      = Integer.parseInt(temp[1]);

                visited.add(curStr);
                res[curX][curY] = count;
                for(int k = 0; k < 4; k++){
                    int newX = curX + direction[k][0];
                    int newY = curY + direction[k][1];
                    String newStr = newX + "@" + newY;
                    if(isInRange(newX, newY) && !visited.contains(newStr)){
                        queue.add(newStr);
                    }
                }
            }

            count++;
        }

        return res;
    }
```

```
50
51      private boolean isInRange(int x, int y){
52          return x >= 0 && y >= 0 && x < row && y < col;
53      }
```

543 Diameter of Binary Tree

执行用时： 13 ms , 在所有 Java 提交中击败了 10.29% 的用户

内存消耗： 38.8 MB , 在所有 Java 提交中击败了 5.08% 的用户

```
1       int max = 0;
2       public int diameterOfBinaryTree(TreeNode root) {
3           if(root == null)
4               return 0;
5
6           max = Math.max(max, maxDepth(root.left) + maxDepth(root.right));
7           diameterOfBinaryTree(root.left);
8           diameterOfBinaryTree(root.right);
9
10          return max;
11      }
12
13      public int maxDepth(TreeNode root){
14          if(root == null)
15              return 0;
16
17          return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
18      }
```

544 Output Contest Matches

544. Output Contest Matches

难度 中等 30

During the NBA playoffs, we always arrange the rather strong team to play with the rather weak team, like make the rank 1 team play with the rank n_{th} team, which is a good strategy to make the contest more interesting. Now, you're given **n** teams, you need to output their **final** contest matches in the form of a string.

The **n** teams are given in the form of positive integers from 1 to n, which represents their initial rank. (Rank 1 is the strongest team and Rank n is the weakest team.) We'll use parentheses('(', ')') and commas(',') to represent the contest team pairing - parentheses('(' , ')') for pairing and commas(',') for partition. During the pairing process in each round, you always need to follow the strategy of making the rather strong one pair with the rather weak one.

Example 1:

```
Input: 2
Output: (1,2)
Explanation:
Initially, we have the team 1 and the team 2, placed
like: 1,2.
Then we pair the team (1,2) together with '(', ')' and
',', which is the final answer.
```

Example 2:

执行用时： 4 ms , 在所有 Java 提交中击败了 89.06% 的用户

内存消耗： 37.8 MB , 在所有 Java 提交中击败了 89.06% 的用户

```java
    public String findContestMatch(int n) {
        List<String> res = null;

        int round = (int)Math.pow(n, 1 / 2) + 1;
        for(int i = 0; ; i++){
            List<String> temp = new ArrayList<>();
            if(i == 0){

                for(int j = 1; j <= n / 2; j++){
                    StringBuilder sb = new StringBuilder();
                    sb.append("(").append(j).append(",").append(n - j + 1).append(")");
                    temp.add(sb.toString());
                }

                res = temp;
                if(res.size() == 1)
```

```
                    break;
            }else{
                int left = 0, right = res.size() - 1;

                while(left < right){
                    StringBuilder sb = new StringBuilder();
                    sb.append("
(").append(res.get(left)).append(",").append(res.get(right)).append(")");
                    left++;
                    right--;
                    temp.add(sb.toString());
                }

                res = temp;
                if(res.size() == 1)
                    break;
            }
        }

        return res.get(0);
    }
```

545 Boundary of Binary Tree

4 天前	通过	6 ms	38.8 MB	J	4 5 6

```
    List<Integer> leftBoundary = new ArrayList<>();
    List<Integer> leaves       = new ArrayList<>();
    List<Integer> rightBoundary = new ArrayList<>();
    public List<Integer> boundaryOfBinaryTree(TreeNode root) {
        if(root == null)
            return new ArrayList<>();
        if(root.left == null && root.right == null){
            List<Integer> res = new ArrayList<>();
            res.add(root.val);
            return res;
        }
        getLeftBoundary(root);
        System.out.println(leftBoundary);
        getRightBoundary(root);
        System.out.println(rightBoundary);
        getLeaves(root);
```

```
        System.out.println(leaves);

        List<Integer> res = new ArrayList<>();
        res.add(root.val);
        res.addAll(leftBoundary);
        res.addAll(leaves);
        Collections.reverse(rightBoundary);
        res.addAll(rightBoundary);

        return res;
    }

    private void getLeaves(TreeNode root) {
        if(root == null)
            return;

        if(root.left == null && root.right == null) {
            leaves.add(root.val);
            return;
        }

        getLeaves(root.left);
        getLeaves(root.right);
    }


    private void getRightBoundary(TreeNode root) {
        if(root.right == null)
            return;
        TreeNode head = root;

        while(root != null){
            while(root.right != null){
                if(head != root)
                    rightBoundary.add(root.val);
                root = root.right;
            }


            if(root.left != null){
                if(root != head)
                    rightBoundary.add(root.val);
                root = root.left;
            }else
                break;
        }
    }

    private void getLeftBoundary(TreeNode root) {
```

```
66          if(root.left == null)
67              return;
68          TreeNode head = root;
69
70          while(root != null){
71              while(root.left != null){
72
73                  if(head != root)
74                      leftBoundary.add(root.val);
75                  root = root.left;
76              }
77
78              if(root.right != null){
79                  if(root != head)
80                      leftBoundary.add(root.val);
81                  root = root.right;
82              }else
83                  break;
84          }
85      }
86
```

546 Remove Boxes

546. Remove Boxes

难度 困难 274

You are 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 (i.e., composed of `k` boxes, `k >= 1`), remove them and get `k * k` points.

Return *the maximum points you can get*.

Example 1:

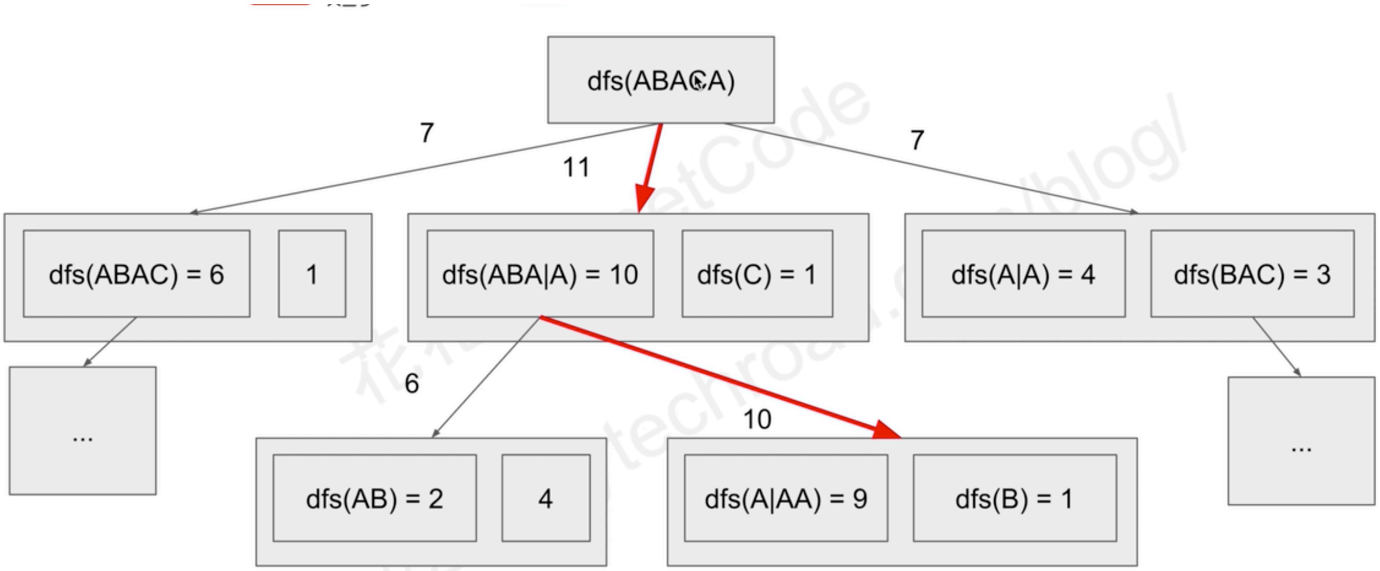
```
Input: boxes = [1,3,2,2,2,3,4,3,1]
Output: 23
Explanation:
[1, 3, 2, 2, 2, 3, 4, 3, 1]
----> [1, 3, 3, 4, 3, 1] (3*3=9 points)
----> [1, 3, 3, 3, 1] (1*1=1 points)
----> [1, 1] (3*3=9 points)
----> [] (2*2=4 points)
```

Example 2:

```
Input: boxes = [1,1,1]
Output: 9
```

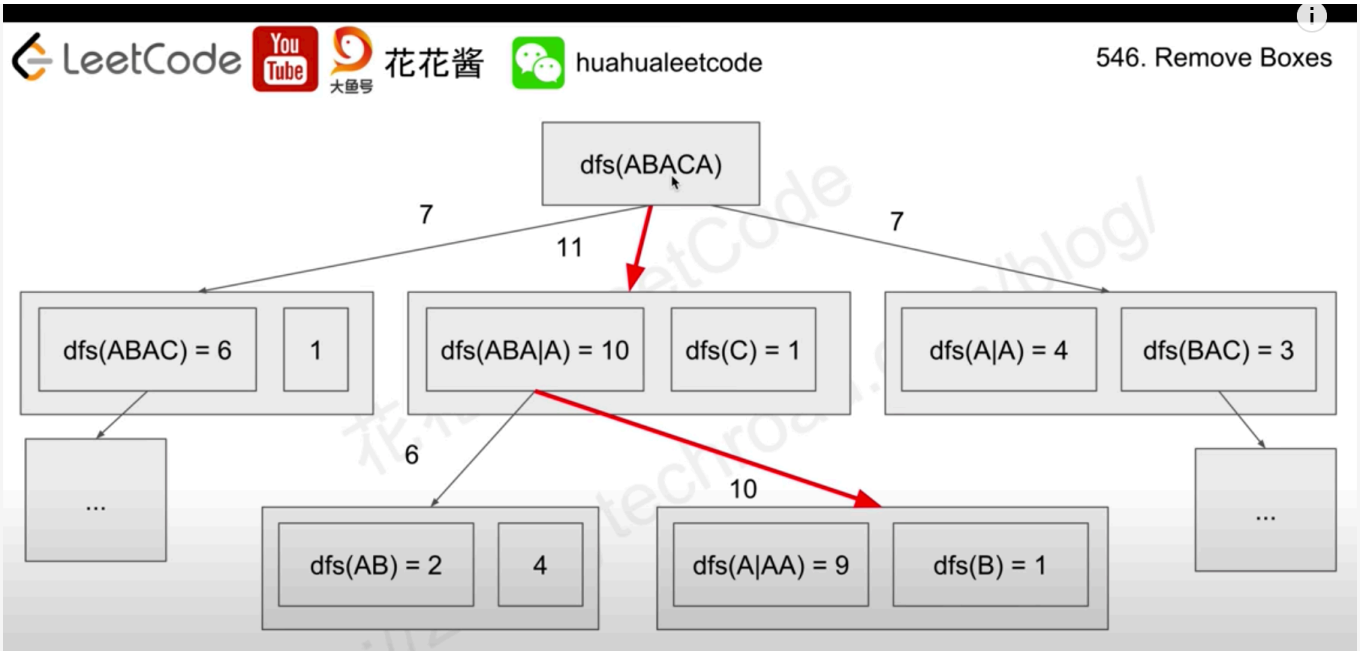
Example 3:

```
Input: boxes = [1]
```




```
/*
  类似
  312 Burst Ballons
  488 Zuma Game

  dp[i][j][k] = dp[i][j - 1][0] + score
                or
                dp[i][p][k + 1] + dp[p + 1][j - 1][0];

source: https://www.bilibili.com/video/BV11W411Z7jG?
from=search&seid=17616780077873944864

  采用记忆化回溯，  比 dp 稍微好一点

  刚开始的思路 可能是
  dp[i][j] -> max score obtained from array[i..j]
    然后通过取分割点 k    i <= k < j 进行求解
    dp[i][j] = Math.max(dp[i][k] + dp[k + 1][j])
    但是无法计算出这种情况
      比如 A B A
      dp(ABA) = dp(A) + dp(BA) = 3;
              = dp(AB) + dp(A) = 3;

    无法考察到 dp(ABA) = dp(AA) + dp(B)


  因此提升维度，
  dp[i][j][k] = max score of subarry : arr[i] ~ ar[j] if there are k boxes
  that have the same color as arr[j] following arre[j]
    也就是 k 的作用，记录 j 索引之后，还有几个和j的颜色相同的box

  比如
```

```
  “ABACA”, dp[0][0][2] = dfs("A|AA")  = 9,  // B C 假设被移除

  状态转移方程
  dp[i][j][k] = dp[i][j - 1][0] + (k - 1) ^ 2  // case 1 假设将j 及以后的相同颜色合并消除
              = dp[i][p][k + 1] + dp[p + 1][j - 1][0] // case 2
                //寻找分割点，使得 p 处的颜色 与 j 相同
                //将j 追加到 p 后面，然后分割两个子串
*/
class Solution{
public:
  int removeBoxes(vector<int>& boxes){
    const int n = boxes.size();
    m_ = vector<vector<vector<int>>>(n, vector<vector<int>>(n, vector<int<(n)));

    return dfs(boxes, 0, n - 1, 0);
  }

private :
  vector<vector<vector<int>>> m_;
  int dfs(const vector<int>& boxes, int l, int r, int k){
    if(l > r)
        return 0;
    if(m_[l][r][k] > 0)
        return m_[l][r][k];

    m_[l][r][k] = dfs(boxes, l, r - 1, 0) + (k + 1) * (k + 1);
    for(int i = l; i < r; i++)
        if(boxes[i] == boxes[r])
            m_[l][r][k] = max(m_[l][r][k], dfs(boxes, l, i, k + 1) + dfs(boxes, i + 1, r - 1, 0));

    return m_[l][r][k];
  }
};
```




```java
//    java version
    int[][][] dp;
    public int removeBoxes(int[] boxes) {
        int len = boxes.length;
        dp = new int[len][len][len];

        return dfs(boxes, 0, len - 1, 0);
    }

    private int dfs(int[] boxes, int l, int r, int k){
        if(l > r)
            return 0;
        if(dp[l][r][k] > 0)
            return dp[l][r][k];

        dp[l][r][k]  = dfs(boxes, l, r - 1, 0) + (k + 1) * (k + 1);
        for(int i = l; i < r; i++)
            if(boxes[i] == boxes[r])
                dp[l][r][k] = Math.max(dp[l][r][k], dfs(boxes, l, i, k + 1) +
dfs(boxes, i + 1, r - 1, 0));

        return dp[l][r][k];
    }s
```

547 Number of Provinces UF classic Q

执行用时： 4 ms ，在所有 Java 提交中击败了 22.55% 的用户

内存消耗： 39.5 MB ，在所有 Java 提交中击败了 33.25% 的用户

```java
    public int findCircleNum(int[][] isConnected) {
        WeightedUnionFind wuf = new WeightedUnionFind(isConnected.length);

        for(int i = 0; i < isConnected.length; i++){
            for(int j = 0; j < isConnected[i].length; j++){
                if(isConnected[i][j] == 1) {
                    wuf.union(i, j);
                }
            }
        }

        return wuf.getCount();
    }
}

class WeightedUnionFind{
    private int[] id;
    private int[] sz;
    private int count;

    public WeightedUnionFind(int count){
        this.id    = new int[count];
        this.sz    = new int[count];
        this.count = count;

        for(int i = 0; i < count; i++){
            this.id[i] = i;
            this.sz[i] = 1;
        }
    }

    public boolean isConnected(int p, int q){
        return find(p) == find(q);
    }

    private int find(int p) {
        while(p != id[p]){
            id[p] = id[id[p]];
            p = id[p];
        }

        return p;
    }

```

```
45      public void union(int p, int q){
46          int pRoot = find(p);
47          int qRoot = find(q);
48
49          if(pRoot == qRoot)
50              return;
51
52          if(sz[pRoot] > sz[qRoot]){
53              sz[pRoot] += sz[qRoot];
54              id[qRoot] = id[pRoot];
55          }else{
56              sz[qRoot] += sz[pRoot];
57              id[pRoot] = id[qRoot];
58          }
59
60          count--;
61      }
62
63      public int getCount(){
64          return count;
65      }
66  }
```

548 Split Array With Equal Sum

548. Split Array with Equal Sum

难度 中等 32

Given an array with n integers, you need to find if there are triplets (i, j, k) which satisfies following conditions:

1. 0 < i, i + 1 < j, j + 1 < k < n - 1
2. Sum of subarrays (0, i - 1), (i + 1, j - 1), (j + 1, k - 1) and (k + 1, n - 1) should be equal.

where we define that subarray (L, R) represents a slice of the original array starting from the element indexed L to the element indexed R.

Example:

```
Input: [1,2,1,2,1,2,1]
Output: True
Explanation:
i = 1, j = 3, k = 5.
sum(0, i - 1) = sum(0, 0) = 1
sum(i + 1, j - 1) = sum(2, 2) = 1
sum(j + 1, k - 1) = sum(4, 4) = 1
sum(k + 1, n - 1) = sum(6, 6) = 1
```

Note:

1. 1 <= n <= 2000.
2. Elements in the given array will be in range [-1,000,000, 1,000,000].

通过次数 1,283 | 提交次数 3,776

```java
/*
    source: https://www.youtube.com/watch?v=GeT_IfUbC0s

    本质上也是求前缀和

*/
public boolean splitArray(int[] nums) {
        int len = nums.length;
        int[] prefix = new int[len];
        for(int i = 0; i < len; i++){
            if(i == 0)
                prefix[i] = nums[i];
            else
                prefix[i] += prefix[i - 1] + nums[i];
        }

  /*
      对每一个j， 可以并行求解 i 和 k

```

```
        防止出现这种案例
          i       j
        4000000004x

        这样的话，  i的位置会反复横跳

        时间复杂度是 O(n2)
    */
          for(int j = 3; j <= len - 3;j++){
              HashSet<Integer> set = new HashSet<>();

              for(int i = 1; i < j - 1; i++){
                  int sum_l1 = prefix[i - 1];
                  int sum_l2 = prefix[j - 1]  - prefix[i];
                  if(sum_l1 == sum_l2)
                      set.add(sum_l1);
              }

              //寻找符合题目条件的点
              for(int k = j + 2; k < len - 1; k++){
                  int sum_r1 = prefix[k - 1] - prefix[j];
                  int sum_r2 = prefix[len - 1] - prefix[k];
                  if(sum_r1 == sum_r2 && set.contains(sum_r1))
                      return true;
              }
          }

          return false;
      }
```

549 Binary Tree Longest Consecutive Sequence II

549. Binary Tree Longest Consecutive Sequence II

难度 中等 65

Given a binary tree, you need to find the length of Longest Consecutive
in Binary Tree.

Especially, this path can be either increasing or decreasing. For examp
[1,2,3,4] and [4,3,2,1] are both considered valid, but the path [1,2,4,3
valid. On the other hand, the path can be in the child-Parent-child orde
where not necessarily be parent-child order.

Example 1:

```
Input:
        1
       / \
      2   3
Output: 2
Explanation: The longest consecutive path is [1, 2] c
[2, 1].
```

Example 2:

```
Input:
        2
       / \
      1   3
Output: 3
Explanation: The longest consecutive path is [1, 2, 3
```

```java
/*
  类似题目 128
*/
  int res = 0;
  public int longestConsecutive(TreeNode root) {
      postorder(root);
      return res;
  }

  //arr[0] 递增 arr[1] 递减
  private int[] postorder(TreeNode root){
      int[] arr = new int[2];
      if(root == null)
          return arr;
      arr[0] = 1;
      arr[1] = 1;
```

```
        int[] left = postorder(root.left);
        int[] right = postorder(root.right);

        if(root.left != null){
            if(root.left.val - 1 == root.val)
                arr[1] = left[1] + 1;
            else if(root.left.val + 1 == root.val)
                arr[0] = left[0] + 1;
        }

        if(root.right != null){
            if(root.right.val - 1 == root.val)
                arr[1] = Math.max(arr[1], right[1] + 1);
            if(root.right.val + 1 == root.val)
                arr[0] = Math.max(arr[0], right[0] + 1);
        }

        res = Math.max(res, arr[0] + arr[1] - 1);
        return arr;
    }

作者：mmmmmJCY
链接：https://leetcode-cn.com/problems/binary-tree-longest-consecutive-sequence-ii/solution/java-di-gui-by-zxy0917-12/
```

551 Student Attendance Record I

执行用时： 1 ms ，在所有 Java 提交中击败了 40.74% 的用户

内存消耗： 36.5 MB ，在所有 Java 提交中击败了 75.63% 的用户

```
    public boolean checkRecord(String s) {
        int countA = 0;
        int left = 0, right = 0;
        while(right < s.length()){
            while(right < s.length() && s.charAt(right) != 'L'){
                if(s.charAt(right) == 'A')
                    countA++;
                right++;
            }

            if(right == s.length())
                break;
```

```
13
14              left = right;
15              while(right < s.length() && s.charAt(right) == 'L')
16                  right++;
17
18              if(right - left >= 3)
19                  return false;
20
21              left = right;
22          }
23
24          return countA <= 1;
25      }
```

552 Student Attendance Record II

552. Student Attendance Record II

难度 困难 121

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.

```java
/*
  超时： 通过14 / 68
  时间复杂度 O（3 ^N)


  解法就是把551的拿过来，用个回溯不断尝试
*/
long count = 0;
    List<Character> list = new ArrayList<>();
    public int checkRecord(int n) {
        list.add('A');
        list.add('L');
        list.add('P');
        backtrack(new StringBuilder(), n);
        return (int) (count % (1000000000 + 7));
    }

    private void backtrack(StringBuilder path, int n) {
        if(path.length() == n){
```

```
            if(checkRecord(path.toString()))
                count++;

            return;
        }

        for(int i = 0; i < list.size(); i++){
            path.append(list.get(i));

            backtrack(path, n);

            path.setLength(path.length() - 1);
        }
    }

    public boolean checkRecord(String s) {
        int countA = 0;
        int left = 0, right = 0;
        while(right < s.length()){
            while(right < s.length() && s.charAt(right) != 'L'){
                if(s.charAt(right) == 'A')
                    countA++;
                right++;
            }

            if(right == s.length())
                break;

            left = right;
            while(right < s.length() && s.charAt(right) == 'L')
                right++;

            if(right - left >= 3)
                return false;

            left = right;
        }

        return countA <= 1;
    }

```

```
/*
    采用O(N) 时间复杂度

    source : youtube.com/watch?v=06YtJdBG0rk

```

```
    定义 dp[i] = total number of rewardable student records with i len
      注意这里的dp 不包含 A

    分为几种情况
    首先计算 -> Without A
      1) end with p,  如果当前i结尾的是P
        那无求所谓
        dp[i] = dp[i - 1]
      2) ends with L 如果当前 i 结尾的是 L
        dp[i] = dp[i - 1] - dp[i - 4]
        ...PLLL

    然后再计算含有 A的情况
    Assume A 的index是 i
    [....i - 1] A [i + 1.....]
    dp[i - 1] * dp[n - i]

    i + 1, N
    n - (i + 1) + 1
*/
public class Solution {
    long M = 1000000007;
    public int checkRecord(int n) {
        long[] f = new long[n <= 5 ? 6 : n + 1];
        f[0] = 1;
        f[1] = 2;
        f[2] = 4;
        f[3] = 7;
        for (int i = 4; i <= n; i++)
            f[i] = ((2 * f[i - 1]) % M + (M - f[i - 4])) % M;
        long sum = f[n];
        for (int i = 1; i <= n; i++) {
            sum += (f[i - 1] * f[n - i]) % M;
        }
        return (int)(sum % M);
    }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/student-attendance-record-ii/solution/xue-sheng-chu-qin-ji-lu-ii-by-leetcode/
来源：力扣（LeetCode）

```

553 Optimal Division 很棒的 DP 题目

553. Optimal Division

难度 中等 61

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:

```java
/*
  O(n) 的时间复杂度，利用的是数学方法
  比如  a / b / c / d


*/
class Solution {
    public String optimalDivision(int[] nums) {
        if(nums.length == 1)
            return nums[0] + "";
```

```
        else if(nums.length == 2){
            return nums[0] + "/" + nums[1];
        }
        StringBuilder res = new StringBuilder();
        res.append(nums[0]).append("/(");

        for(int i = 1; i < nums.length; i++){
            res.append(nums[i]);
            if(i != nums.length - 1)
                res.append("/");
        }

        res.append(")");
        return res.toString();
    }
}
```

```
    public String optimalDivision(int[] nums) {
        int len = nums.length;

        /*
            0 means max
            1 means min
        */
        double[][][] dp = new double[len][len][2];
        String[][][] res = new String[len][len][2];

        for(int i = 0; i < len; i++){
            dp[i][i][0] = nums[i];
            dp[i][i][1] = nums[i];
            res[i][i][0] = String.valueOf(nums[i]);
            res[i][i][1] = String.valueOf(nums[i]);
        }

        for(int i = len - 2; i >= 0; i--){
            for(int j = i + 1; j < len; j++){
                StringBuilder maxString = new StringBuilder();
                StringBuilder minString = new StringBuilder();
                double maxNum           = Double.MIN_VALUE;
                double minNum           = Double.MAX_VALUE;

                for(int k = i; k < j; k++){
                    if(dp[i][k][0] / dp[k + 1][j][1] > maxNum){
                        maxString = new StringBuilder();
                        if(k + 1 == j){
```

```
                            maxString.append(res[i][k][0]).append("/").append(res[k + 1][j][0]);
                        }else{
                            maxString.append(res[i][k][0]).append("/(").append(res[k + 1][j][1]).append(")");
                        }

                        maxNum = dp[i][k][0] / dp[k + 1][j][1];
                    }

                    if(dp[i][k][1] / dp[k + 1][j][0] < minNum){
                        minString = new StringBuilder();
                        if(k + 1 == j){
                            minString.append(res[i][k][1]).append("/").append(res[k + 1][j][0]);
                        }else{
                            minString.append(res[i][j][1]).append("/(").append(res[k + 1][j][1]).append(")");
                        }

                        minNum = dp[i][k][1] / dp[k + 1][j][0];
                    }
                }

                dp[i][j][0] = maxNum;
                dp[i][j][1] = minNum;
                res[i][j][0] = maxString.toString();
                res[i][j][1] = minString.toString();
            }
        }


        return res[0][len - 1][0];
    }

```

554 Brick Wall

554. Brick Wall

难度 中等 130

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:
```

执行用时： 20 ms ，在所有 Java 提交中击败了 13.92% 的用户

内存消耗： 41 MB ，在所有 Java 提交中击败了 93.08% 的用户

```java
public int leastBricks(List<List<Integer>> wall) {
        //position, number
    HashMap<Integer, Integer> map = new HashMap<>();
    int sum = 0;
    for(int i = 0; i < wall.size(); i++){
        int pos = 0;

        for(int j = 0; j < wall.get(i).size(); j++){
            pos += wall.get(i).get(j);
```

```
                map.put(pos, map.getOrDefault(pos, 0) + 1);

                if(i == 0)
                    sum += wall.get(i).get(j);
            }
        }

        int res = Integer.MAX_VALUE;
        int len = wall.size();
        for(Integer pos : map.keySet()){
            if(sum == pos)
                continue;
            res = Math.min(res, len - map.get(pos));
        }

        return res == Integer.MAX_VALUE ? len : res;
    }
```

555 Split Concatenated Strings

555. Split Concatenated Strings

难度 中等 20

Given a list of strings, you could concatenate these strings together into a loop, where for each string you could choose to reverse it or not. Among all the possible loops, you need to find the lexicographically biggest string after cutting the loop, which will make the looped string into a regular one.

Specifically, to find the lexicographically biggest string, you need to experience two phases:

1. Concatenate all the strings into a loop, where you can reverse some strings or not and connect them in the same order as given.
2. Cut and make one breakpoint in any place of the loop, which will make the looped string into a regular one starting from the character at the cutpoint.

And your job is to find the lexicographically biggest one among all the possible regular strings.

Example:

```
Input: "abc", "xyz"
Output: "zyxcba"
Explanation: You can get the looped string "-abcxyz-", "-abczyx-", "-cbaxyz-", "-cbazyx-",
where '-' represents the looped status.
The answer string came from the fourth looped one,
where you could cut from the middle character 'a'
and get "zyxcba".
```

```java
class Solution {
    public String splitLoopedString(String[] strs) {
        int len = strs.length;
        if(len<1) return "";

        //transform to max lexi order
        for(int i =0; i<strs.length; i++){
            String rev = new StringBuilder(strs[i]).reverse().toString();
            if(strs[i].compareTo(rev)<0)
              strs[i] = rev;
        }

        String ans = "";
        char max = 'a';
```

```
        //try each str be the cut string, denote i
        for(int i =0; i<strs.length; i++){
            String rev = new StringBuilder(strs[i]).reverse().toString();

            //try normal and reverse order for the target str i
            for(String s : new String[]{strs[i],rev}){

                //pick the index of the cut position in this str, denote j
                for(int j = 0 ; j < strs[i].length(); j++){

                    // always start from the max one
                    if(s.charAt(j)<max)
                      continue;
                    else
                      max = s.charAt(j);

                    StringBuilder trys = new StringBuilder(s.substring(j));
                    for(int k = i+1; k < strs.length;k++){
                        trys.append(strs[k]);
                    }
                    for(int k = 0; k<i; k ++){
                        trys.append(strs[k]);
                    }
                    trys.append(s.substring(0,j));
                    if(trys.toString().compareTo(ans)>0)
                      ans = trys.toString();
                }

            }

        }
        return ans;
    }
}
```

556 Next Greater Element III

556. Next Greater Element III

难度 中等 120

Given a positive integer `n`, find *the smallest integer which has exactly th same digits existing in the integer* `n` *and is greater in value than* `n`. If n such positive integer exists, return `-1`.

Note that the returned integer should fit in **32-bit integer**, if there is a v answer but it does not fit in **32-bit integer**, return `-1`.

Example 1:

```
Input: n = 12
Output: 21
```

Example 2:

```
Input: n = 21
Output: -1
```

Constraints:

- `1 <= n <= 2^31 - 1`

通过次数 8,627 | 提交次数 27,564

执行用时： 5 ms ，在所有 Java 提交中击败了 41.97% 的用户

内存消耗： 35.6 MB ，在所有 Java 提交中击败了 9.12% 的用户

```java
public int nextGreaterElement(int n) {
    if((n + "").length() == 1)
        return -1;

    char[] chars = (n + "").toCharArray();

    int len = chars.length;
    int index = len - 1;

    while(index >= 1 && chars[index - 1] >= chars[index])
        index--;

```

```
        if(index == 0)
            return -1;

        int right        = index + 1;
        int exchNumIndex = index;
        while(right < len && chars[right] > chars[index - 1]){
            exchNumIndex = right;
            right++;
        }

        if(index == len - 1){
            exch(chars, index, index - 1);
            return parseInt(chars);
        }else{
            exch(chars, exchNumIndex, index - 1);
            Arrays.sort(chars, index, len);
        }

        int res = parseInt(chars);
        return res;
    }

    private int parseInt(char[] chars){
        int res   = 0;
        int index = 0;
        while(index < chars.length){
            int num = chars[index] - '0';
            if(res > Integer.MAX_VALUE / 10 || (res == Integer.MAX_VALUE / 10 &&
num > Integer.MAX_VALUE % 10))
                return -1;

            res *= 10;
            res += num;
            index++;
        }

        return res;
    }

    private void exch(char[] chars, int i, int j){
        char ch  = chars[i];
        chars[i] = chars[j];
        chars[j] = ch;
    }

    private void reverse(char[] chars){
        int left = 0, right = chars.length - 1;
        while(left < right){
            char temp = chars[left];
```

```
            chars[left] = chars[right];
            chars[right] = temp;

            left++;
            right--;
        }
    }
```

557 Reverse Words In a String III

执行用时： 5 ms ，在所有 Java 提交中击败了 84.70% 的用户

内存消耗： 39.1 MB ，在所有 Java 提交中击败了 46.59% 的用户

```

public String reverseWords(String s) {
        String[] temp = s.split(" ");

        StringBuilder res = new StringBuilder();
        for(int i = 0; i < temp.length; i++){
            res.append(new StringBuilder(temp[i]).reverse());
            if(i != temp.length - 1)
                res.append(" ");
        }

        return res.toString();
    }
```

558 Logical OR of Two Binary Grids Represented as Quad-Trees

558. Logical OR of Two Binary Grids Represented as Quad-Trees

难度 中等 21

A Binary Matrix is a matrix in which all the elements are either **0** or **1**.

Given `quadTree1` and `quadTree2`. `quadTree1` represents a `n * n` binary matrix and `quadTree2` represents another `n * n` binary matrix.

Return *a Quad-Tree* representing the `n * n` binary matrix which is the result of **logical bitwise OR** of the two binary matrixes represented by `quadTree1` and `quadTree2`.

Notice that you can assign the value of a node to **True** or **False** when `isLeaf` is **False**, and both are **accepted** in the answer.

A Quad-Tree is a tree data structure in which each internal node has exactly four children. Besides, each node has two attributes:

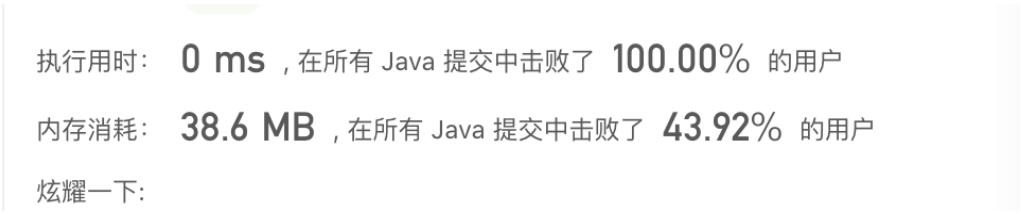
- `val`: True if the node represents a grid of 1's or False if the node represents a grid of 0's.
- `isLeaf`: True if the node is leaf node on the tree or False if the node has the four children.

```
class Node {
    public boolean val;
    public boolean isLeaf;
    public Node topLeft;
    public Node topRight;
    public Node bottomLeft;
    public Node bottomRight;
}
```

```
1  class Solution {
2      public Node intersect(Node quadTree1, Node quadTree2) {
3          if(quadTree1 == null || quadTree2 == null)
4              return quadTree1 == null ? quadTree2 : quadTree1;
5
6          /*
7              1. deal with special cases
8
```

```
            2. general cases
        */

        if(quadTree1.isLeaf || quadTree2.isLeaf){
            if((quadTree1.isLeaf && quadTree2.isLeaf) || (quadTree1.isLeaf && quadTree1.val) || (quadTree2.isLeaf && quadTree2.val)  )
            return new Node(quadTree1.val | quadTree2.val, true, null, null, null, null);
        }


        Node res   = new Node();
        res.isLeaf = false;
        res.topLeft     = intersect(quadTree1.topLeft, quadTree2.topLeft);
        res.topRight    = intersect(quadTree1.topRight, quadTree2.topRight);
        res.bottomLeft  = intersect(quadTree1.bottomLeft, quadTree2.bottomLeft);
        res.bottomRight = intersect(quadTree1.bottomRight, quadTree2.bottomRight);

        int isLeaf = sumAll(res.topLeft.isLeaf, res.topRight.isLeaf, res.bottomLeft.isLeaf, res.bottomRight.isLeaf);
        int val    = sumAll(res.topLeft.val, res.topRight.val, res.bottomLeft.val, res.bottomRight.val);

        if(isLeaf == 4 && (val == 4 || val == 0)){
            res.isLeaf = true;
            res.val    = val == 4 ? true : false;
            res.topLeft = res.topRight = res.bottomRight = res.bottomLeft = null;
            return res;
        }

        return res;
    }

    private int sumAll(boolean a, boolean b, boolean c, boolean d){
        int i1 = a ? 1 : 0;
        int i2 = b ? 1 : 0;
        int i3 = c ? 1 : 0;
        int i4 = d ? 1 : 0;

        return i1 + i2 + i3 + i4;
    }
}
```

559 Maximum Depth of N Ary Tree




```java
    public int maxDepth(Node root) {
        if(root == null)
            return 0;

        int maxDepth = 0;
        for(Node node : root.children)
            maxDepth = Math.max(maxDepth, maxDepth(node));

        return 1 + maxDepth;
    }
```

560 Subarray Sum Equals K

560. Subarray Sum Equals K

难度 中等 772

Given an array of integers `nums` and an integer `k`, return *the total number of continuous subarrays whose sum equals to* `k`.

Example 1:

```
Input: nums = [1,1,1], k = 2
Output: 2
```

Example 2:

```
Input: nums = [1,2,3], k = 3
Output: 2
```

Constraints:

- `1 <= nums.length <= 2 * 10^4`
- `-1000 <= nums[i] <= 1000`
- `-10^7 <= k <= 10^7`

执行用时： 2234 ms ，在所有 Java 提交中击败了 5.02% 的用户

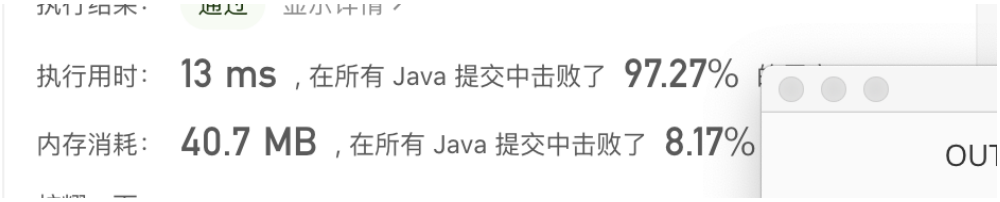
内存消耗： 40.8 MB ，在所有 Java 提交中击败了 71.50% 的用户

```java
/*
    前缀和的典型应用
*/
public int subarraySum(int[] nums, int k) {
        int len = nums.length;
        int[] prefixSum = new int[len];

        for(int i = 0; i < len; i++){
            if(i == 0)
                prefixSum[i] = nums[i];
            else
                prefixSum[i] = prefixSum[i - 1] + nums[i];
        }

        int count = 0;
        for(int i = -1; i < len; i++){
```

```
            for(int j = i + 1; j < len; j++){
                int subarraySum;

                if(i == -1)
                    subarraySum = prefixSum[j];
                else
                    subarraySum = prefixSum[j] - prefixSum[i];

                if(subarraySum == k)
                    count++;
            }
        }

        return count;
    }
```

```
/*
    精彩解法，  空间换时间
      O(n) 时间复杂度
*/

public int subarraySum(int[] nums, int k) {
        HashMap<Integer, Integer> map = new HashMap<>();

        int sum = 0;

  // 边界情况
        map.put(0, 1);
        int count = 0;
        for(int i = 0; i < nums.length; i++){
            sum += nums[i];

            if(map.containsKey(sum - k))
                count += map.get(sum - k);

            map.put(sum, map.getOrDefault(sum, 0) + 1);
        }

        return count;
    }
```

561 Array partition I




```java
  //秒了
public int arrayPairSum(int[] nums) {
        Arrays.sort(nums);

        int sum = 0;
        for(int i = 0; i < nums.length; i += 2)
            sum += nums[i];

        return sum;
    }
```

562 Longest Line of Consecutive One in Matrix DP 不错的应用

562. Longest Line of Consecutive One in Matrix

难度 中等 36

Given a 01 matrix **M**, find the longest line of consecutive one in the matrix. The line could be horizontal, vertical, diagonal or anti-diagonal.
Example:

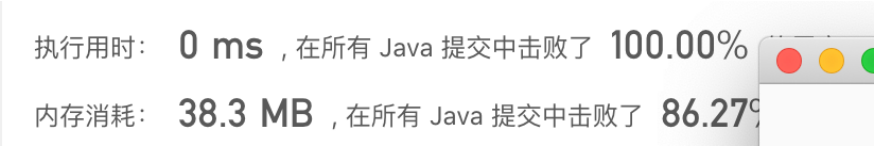
```
Input:
[[0,1,1,0],
 [0,1,1,0],
 [0,0,0,1]]
Output: 3
```

Hint: The number of elements in the given matrix will not exceed 10,000.

```java
public int longestLine(int[][] M) {
    int row = M.length;
    int col = row == 0 ? 0 : M[0].length;

    //0 horizontal, 1 vertical, 2 main diagonal, 3 anti-diagonal
    int[][][] dp = new int[row][col][4];

    int res = 0;
    for(int i = 0; i < row; i++){
        for(int j = 0; j < col; j++){
            if(M[i][j] == 0)
                continue;

            for(int k = 0; k < 4; k++)
                dp[i][j][k] = 1;

            //horizontal
            if(j > 0)
                dp[i][j][0] += dp[i][j - 1][0];
            //vertical
            if(i > 0)
                dp[i][j][1] += dp[i - 1][j][1];
            //main diagonal
            if(i > 0 && j > 0)
                dp[i][j][2] += dp[i - 1][j - 1][2];
            //anti-diagonal
            if(i > 0 && j < col - 1)
                dp[i][j][3] += dp[i - 1][j + 1][3];

            for(int k = 0; k < 4; k++)
```

```
                    res = Math.max(res, dp[i][j][k]);
                }
            }

            return res;
        }

作者：jyj407
链接：https://leetcode-cn.com/problems/longest-line-of-consecutive-one-in-matrix/solution/zhong-gui-zhong-ju-dong-tai-gui-hua-jie-tdskl/
来源：力扣（LeetCode）
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

563 Binary Tre Tilt




```java
    int res = 0;
    public int findTilt(TreeNode root) {
        postOrder(root);
        return res;
    }

    /* return the sum of the root val*/
    private int postOrder(TreeNode root){
        if(root == null)
            return 0;

        int leftSum  = postOrder(root.left);
        int rightSum = postOrder(root.right);

        int total = leftSum + rightSum + root.val;
        root.val = Math.abs(leftSum - rightSum);

        res += root.val;

        return total;
    }
```

565 Array Nesting 并查集的典型应用

565. Array Nesting

难度 中等 135

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].

```java
/*
    通过 853 / 856 个案例
*/
public int arrayNesting(int[] nums) {
        /*          //index ,       set of index
            HashMap<Integer, HashSet<Integer>> map =
        */

        HashMap<Integer, HashSet<Integer>> map = new HashMap<>();
```

```
        int len = nums.length;

        if(len == 0)
            return 0;

        int res = 1;
        for(int i = 0; i < len; i++){
            int index = nums[i];

            if(index >= len)
                continue;

            if(map.containsKey(index)){
                map.put(i, map.get(index));

                map.get(i).add(index);

                res = Math.max(map.get(i).size(), res);
            }else{
                map.put(i, new HashSet<Integer>());

                int j = i;
                HashSet<Integer> visited = new HashSet<>();
                while(j < len && !visited.contains(j)){
                    visited.add(j);
                    map.get(i).add(j);

                    j = nums[j];
                }
            }
        }

        return res;
    }
```

```
class Solution {
    public int arrayNesting(int[] nums) {
        int len = nums.length;
        if(len == 0)
            return 0;

        WeightedUnionFind wuf = new WeightedUnionFind(len);

        for(int i = 0; i < len; i++){
            wuf.union(i, nums[i]);
        }

```

```
        int res = 1;
        for(int i = 0; i < nums.length; i++)
            res = Math.max(wuf.sz[i], res);

        return res;
    }

}

class WeightedUnionFind{
    public int[] id;
    public  int[] sz;
    private int count;

    public WeightedUnionFind(int N){
        this.count = N;
        this.id    = new int[N];
        this.sz    = new int[count];
        for(int i = 0; i < count; i++){
            this.sz[i] = 1;
            this.id[i] = i;
        }
    }


    public boolean connected(int p, int q){
        return find(p) == find(q);
    }

    public int getCount(){
        return count;
    }

    private int find(int p){
        while(p != id[p]){
            id[p] = id[id[p]];
            p = id[p];
        }

        return p;
    }

    public void union(int p, int q){
        int pRoot = find(p);
        int qRoot = find(q);

        if(pRoot == qRoot)
            return;

```

```
        if(sz[pRoot] > sz[qRoot]){
            sz[pRoot] += sz[qRoot];
            id[qRoot] = id[pRoot];

        }else{
            sz[qRoot] += sz[pRoot];
            id[pRoot] = id[qRoot];
        }
    }

}

```

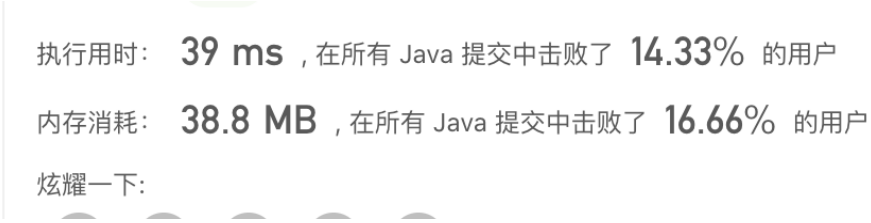
566 Reshape the Matrix

执行用时： 1 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 39.4 MB ，在所有 Java 提交中击败了 88.46% 的用户

```
    public int[][] matrixReshape(int[][] nums, int r, int c) {
        int row= nums.length;
        int col = nums[0].length;

        if(row * col != r * c)
            return nums;

        int index = 0;
        int[][] res = new int[r][c];
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                res[index / c][index % c] = nums[i][j];
                index++;
            }
        }

        return res;
    }
```

567 Permutation in String




```java
public boolean checkInclusion(String s1, String s2) {
    if(s1.length() > s2.length())
        return false;
    Map<Character, Integer> map = new HashMap<>();
    for(char ch : s1.toCharArray())
        map.put(ch, map.getOrDefault(ch, 0) + 1);

    Map<Character, Integer> map2 = new HashMap<>();
    for(int i = 0; i < s1.length(); i++)
        map2.put(s2.charAt(i), map2.getOrDefault(s2.charAt(i), 0) + 1);

    int index = s1.length();
    if(map2.equals(map))
        return true;

    while(index < s2.length()){
        char oldChar = s2.charAt(index - s1.length());
        char newChar = s2.charAt(index);

        map2.put(oldChar, map2.get(oldChar) - 1);
        if(map2.get(oldChar) == 0)
            map2.remove(oldChar);

        map2.put(newChar, map2.getOrDefault(newChar, 0) + 1);

        if(map2.equals(map))
            return true;

        index++;
    }

    return false;
}
```

```java
    public boolean checkInclusion(String s1, String s2) {
        if(s1.length() > s2.length())
            return false;
        HashMap<Character, Integer> mapS1 = new HashMap<>();
        HashMap<Character, Integer> mapS2 = new HashMap<>();

        for(char ch : s1.toCharArray())
            mapS1.put(ch, mapS1.getOrDefault(ch, 0) + 1);


        int index = 0;
        for(int i = 0; i < s1.length(); i++, index++)
            mapS2.put(s2.charAt(i), mapS2.getOrDefault(s2.charAt(i), 0) + 1) ;


        while(index < s2.length()){
            if(mapS1.equals(mapS2))
                return true;

            char before = s2.charAt(index - s1.length());
            char after  = s2.charAt(index);

            mapS2.put(before, mapS2.get(before) - 1);
            if(mapS2.get(before) == 0)
                mapS2.remove(before);
            mapS2.put(after, mapS2.getOrDefault(after, 0) + 1);

            index++;
        }


        return mapS1.equals(mapS2);
    }
```

572 Subtree of Another Tree 回看

572. Subtree of Another Tree

难度 简单 445

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
    / \
```

```java
public boolean isSubtree(TreeNode s, TreeNode t) {
    if(t == null)
        return true;
    if(s == null)
        return false;

    return isSubtree(s.left, t) || isSubtree(s.right, t) || isSameTree(s, t);
}

private boolean isSameTree(TreeNode s, TreeNode t){
    if(s == null || t == null){
        if(s == null && t == null)
            return true;
        return false;
    }
```

```
16
17          if(s.val != t.val)
18              return false;
19
20          return isSameTree(s.left, t.left) && isSameTree(s.right, t.right);
21      }
```

573 Squirrel Simulation 不错的题目

573. Squirrel Simulation

难度 中等 20

There's a tree, a squirrel, and several nuts. Positions are represented by the cells in a 2D grid. Your goal is to find the **minimal** distance for the squirrel to collect all the nuts and put them under the tree one by one. The squirrel can only take at most **one nut** at one time and can move in four directions - up, down, left and right, to the adjacent cell. The distance is represented by the number of moves.

Example 1:

```
Input:
Height : 5
Width : 7
Tree position : [2,2]
Squirrel : [4,4]
Nuts : [[3,0], [2,5]]
Output: 12
Explanation:
```

Note:

1. All given positions won't overlap.
2. The squirrel can take at most one nut at one time.
3. The given positions of nuts have no order.
4. Height and width are positive integers. 3 <= height * width <= 10,000.
5. The given positions contain at least one nut, only one tree and one squirrel.

```java
    public int minDistance(int height, int width, int[] tree, int[] squirrel, int[][] nuts) {

        int sum = 0;
        for(int[] nut : nuts){
            int diffX = Math.abs(nut[0] - tree[0]);
            int diffY = Math.abs(nut[1] - tree[1]);

            sum += diffX + diffY;
        }

        if(squirrel[0] == tree[0] && squirrel[1] == tree[1])
            return sum * 2;

        sum *= 2;
        int res = Integer.MAX_VALUE;
        for(int i = 0; i < nuts.length; i++){
            int diffXS2N = Math.abs(squirrel[0] - nuts[i][0]);
            int diffYS2N = Math.abs(squirrel[1] - nuts[i][1]);

            int diffXN2T = Math.abs(nuts[i][0] - tree[0]);
            int diffYN2T = Math.abs(nuts[i][1] - tree[1]);

            res = Math.min(res, sum + diffXS2N + diffYS2N - diffXN2T - diffYN2T);
        }

        return res;
    }
```

575 Distribute Candies

575. Distribute Candies

难度 简单 98

Alice has `n` candies, where the `i^th` candy is of type `candyType[i]`. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat `n / 2` of the candies she has (`n` is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array `candyType` of length `n`, return *the* ***maximum*** *number of different types of candies she can eat if she only eats* `n / 2` *of them.*

Example 1:

```
Input: candyType = [1,1,2,2,3,3]
Output: 3
Explanation: Alice can only eat 6 / 2 = 3 candies.
Since there are only 3 types, she can eat one of
each type.
```

Example 2:

```
Input: candyType = [1,1,2,3]
Output: 2
Explanation: Alice can only eat 4 / 2 = 2 candies.
Whether she eats types [1,2], [1,3], or [2,3], she
still can only eat 2 different types.
```

执行用时： 43 ms , 在所有 Java 提交中击败了 27.83% 的用户

内存消耗： 40.8 MB , 在所有 Java 提交中击败了 14.17% 的用户

```java
public int distributeCandies(int[] candyType) {
    int count = 0;
    int left = 0, right = 0;
    int len = candyType.length;

    Arrays.sort(candyType);

    while(right < len){
```

```
            count++;
            while(right < len && candyType[right] == candyType[left])
                right++;

            if(right == len)
                break;

            left = right;
        }

        return count > len / 2 ? len / 2 : count;
    }
```

```
//简单优化
public int distributeCandies(int[] candyType) {
        int count = 0;
        int len = candyType.length;

        HashSet<Integer> set = new HashSet<>();
        for(int num : candyType){
            set.add(num);

            if(set.size() >= len / 2)
                return len / 2;
        }


        return set.size() > len / 2 ? len / 2 : set.size();
    }
```

576 Out of Boundary Paths

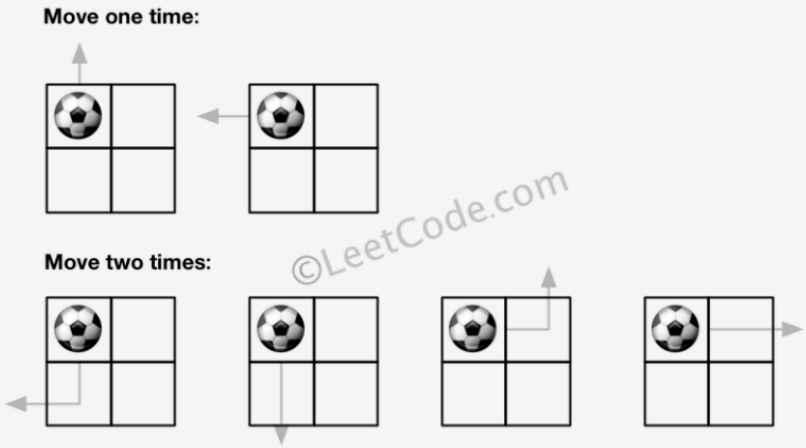
576. Out of Boundary Paths

难度 中等 109

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
```

```
https://www.youtube.com/watch?v=92zh6XvqEgc
类似题目
  62 Unique Paths
  63 Unique Paths II
  688 Knight Probability

/*
想要求 从 i, j 到移出边界的步数
实际上，等价于 从边界到 i, j 步数的总和
  dp[N][i][j] means : the number of paths start from out of boundary to (i, j) by moving n steps
```

```
  dp[*][x][y] = 1 if(x, y is out of bound)
  dp[s][i][j] = dp[s - 1][i + 1][j]
  + dp[s - 1][i - 1][j]
  + dp[s - 1][i][j + 1]
  + dp[s - 1][i][j - 1]

  ANS: dp[N][i][j]
*/
  public int findPaths(int m, int n, int N, int i, int j) {
    if(N == 0)
      return 0;

    int mod = 1000000007;

    int[][] dir = {{1, 0},{0, 1},{-1, 0},{0, -1}};
    int[][][] dp = new int[N + 1][m][n];

    for(int s = 1; s <= N; s++){
      for(int y = 0; y < m; y++){
        for(int x = 0; x < n; x++){
          for(int d = 0; d < 4; d++){
            int newX = x + dir[d][0];
            int newY = y + dir[d][1];

            if(newX < 0 || newY < 0 || newX >= n || newY >= m)
              dp[s][y][x] += 1;
            else
              dp[s][y][x] = (dp[s][y][x] + dp[s - 1][newY][newX]) % mod;
          }
        }
      }
    }

    return dp[N][i][j];
}
```

581 Shorest Unsorted Continuous Subarray

执行结果： 通过 显示详情 >

执行用时： 8 ms , 在所有 Java 提交中击败了 40.70% 的用户

内存消耗： 40.1 MB , 在所有 Java 提交中击败了 17.20% 的用户

```java
/*
    brute force
*/
public int findUnsortedSubarray(int[] nums) {
      int len = nums.length;
      int[] newNums = Arrays.copyOf(nums,len);
      Arrays.sort(newNums);

      int left = 0, right = len - 1;

      for(int i = 0;  i < len; i++, left++)
          if(nums[i] != newNums[left])
              break;


      for(int i = len - 1; i >= 0; i--, right--)
          if(nums[i] != newNums[right])
              break;

      if(left >=  right)
          return 0;
      return right - left + 1;
  }
```

```java
/*
  本质上采用两个指针
  source : https://leetcode-cn.com/problems/shortest-unsorted-continuous-subarray/solution/shi-jian-chao-guo-100de-javajie-fa-by-zackqf/

  每次更新边界的值
*/
public int findUnsortedSubarray(int[] nums) {
      int len = nums.length;

      int max = nums[0];
      int min = nums[len - 1];
      int l = 0, r = len - 1;

      for(int i = 0;  i < len; i++){
          if(max <= nums[i])
```

```
                max = nums[i];
            else
                l = i;
        }

        for(int j = len - 1; j >= 0; j--){
            if(min >= nums[j])
                min = nums[j];
            else
                r = j;
        }

        if(l == 0 && r == len - 1)
            return 0;
        return l - r + 1;
    }
```

582 Kill Process

582. Kill Process

难度 中等 52

Given **n** processes, each process has a unique **PID (process id)** and its **PPID (parent process id)**.

Each process only has one parent process, but may have one or more children processes. This is just like a tree structure. Only one process has PPID that is 0, which means this process has no parent process. All the PIDs will be distinct positive integers.

We use two list of integers to represent a list of processes, where the first list contains PID for each process and the second list contains the corresponding PPID.

Now given the two lists, and a PID representing a process you want to kill, return a list of PIDs of processes that will be killed in the end. You should assume that when a process is killed, all its children processes will be killed. No order is required for the final answer.

Example 1:

```
Input:
pid =  [1, 3, 10, 5]
ppid = [3, 0, 5, 3]
kill = 5
Output: [5,10]
Explanation:
           3
         /   \
        1     5
             /
            10
Kill 5 will also kill 10.
```

执行用时： 40 ms , 在所有 Java 提交中击败了 27.14% 的用户

内存消耗： 47.8 MB , 在所有 Java 提交中击败了 32.14% 的用户

炫耀一下：

```java
class Solution {
    /*
        1. construct the tree
            1) get the root node

            2) go add them
        2. find the node we want to kill
        and it and its child into the result set
        return
    */
```

```
11
12      List<Integer> res;
13      public List<Integer> killProcess(List<Integer> pid, List<Integer> ppid, int
    kill) {
14
15          res = new ArrayList<>();
16
17          //quick find
18          HashMap<Integer, Node> map = new HashMap<>();
19
20          //constructing the tree
21          Node root = null;
22          for(int i = 0; i < pid.size(); i++){
23              Node node = new Node(pid.get(i));
24              map.put(pid.get(i), node);
25
26              if(ppid.get(i) == 0)
27                  root = node;
28          }
29
30          for(int i = 0; i < ppid.size(); i++){
31              Node parent = map.get(ppid.get(i));
32              if(parent == null)
33                  continue;
34              parent.children.add(map.get(pid.get(i)));
35          }
36
37          //go find it.
38          preorder(root, kill);
39          return res;
40      }
41
42
43      /*
44       * this funciton is used to find the proceess ID that wants to kill
45       * using preorder traversal method
46       */
47      private boolean preorder(Node root, int kill){
48          if(root == null)
49              return false;
50          if(root.id == kill){
51              addId(root);
52              res.add(root.id);
53              return true;
54          }
55
56         for(Node n : root.children){
57            if(preorder(n, kill) == true)
58                return true;
```

```
        }

        return false;
    }

    private void addId(Node root){
        if(root == null)
            return;

        for(Node node : root.children){
            addId(node);
            res.add(node.id);
        }

    }
}

class Node{
    public int id;
    public List<Node> children;

    public Node(int id){
        this.id = id;
        this.children = new ArrayList<>();
    }
}
```

583 Delete Operation For Two Strings

583. Delete Operation for Two Strings

难度 中等 174

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".
```

```java
    public int minDistance(String word1, String word2) {
        int len1 = word1.length();
        int len2 = word2.length();

        int[][] dp = new int[len1 + 1][len2 + 1];

        for(int i = 1; i < len1 + 1; i++)
            dp[i][0] = dp[i - 1][0] + 1;

        for(int j = 1; j < len2 + 1; j++)
            dp[0][j] = dp[0][j - 1] + 1;

        for(int i = 1; i < len1 + 1; i++){
            for(int j = 1; j < len2 + 1; j++){
                if(word1.charAt(i - 1) == word2.charAt(j - 1))
                    dp[i][j] = dp[i - 1][j - 1];
                else
                    dp[i][j] = Math.min(dp[i - 1][j - 1] + 2, Math.min(dp[i - 1][j], dp[i][j - 1]) + 1);
            }

        }

        return dp[len1][len2];
    }
```

584 Find Customer Referee

1 分钟前	通过	313 ms	0 B	My

```mysql
# Write your MySQL query statement below
SELECT name
FROM customer
WHERE referee_id != 2 OR referee_id is NULL;
```

586 Customer Placing the Largest Number of Orders

```mysql
SELECT customer_number
FROM orders
GROUP BY customer_number
ORDER BY COUNT(*) DESC
LIMIT 1;

```

587 Erect the Rence

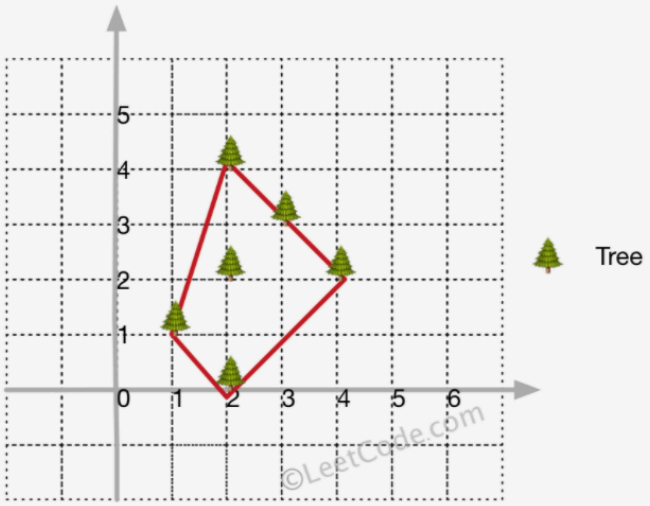
587. Erect the Fence

难度 困难 62

There are some trees, where each tree is represented by (x,y) coordinate in a two-dimensional garden. Your job is to fence the er 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 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:




```
/*
    一开始的思路
      比如拿到 leftmost, rightmost, upmost, downmost

      then 遍历每一个点，看看是谁谁在外面
*/
```

```
/*
    source: https://github.com/Nideesh1/Algo/blob/master/leetcode/L_587.java

    总结下来四步
```

```
      1. 找到最左边的

      2. 每次找到 couter-clock-wise 作为下一个添加点

      3. 循环遍历，直到拿到答案
*/
class Solution {
    public int[][] outerTrees(int[][] points) {
        int n = points.length;

        if(n < 4 || onSmaeLine(points))
            return points;
        int l = 0;
        for(int i = 0; i < n; i++){
            if(points[l][0] > points[i][0])
                l = i;
        }

        List<int[]> list = new ArrayList<>();
        int p = l;

        do{
            list.add(points[p]);

            int q = (p + 1) % n;

            for(int i = 0;  i < n; i++){
                if(i != p && i != q && orientation(points[p],points[q],points[i])
== 1){
                    q = i;
                }
            }

            for(int i = 0; i < n; i++){
                if(i != p && i != q && onSeg(points[p],points[q],points[i]))
                    list.add(points[i]);
            }

            p = q;


        }while(p != l);

        int[][] res = new int[list.size()][2];
        for(int i = 0; i < list.size(); i++){
            res[i][0] = list.get(i)[0];
            res[i][1] = list.get(i)[1];
        }

```

```
        return res;
    }


    //is point r on the same segment forming by p and q?
    private boolean onSeg(int[] p, int[] q, int[] r){
        return r[0] >= Math.min(p[0], q[0]) && r[0] <= Math.max(p[0], q[0])
                && r[1] >= Math.min(p[1], q[1]) && r[1] <= Math.max(p[1], q[1])
                && orientation(p, q, r) == 0;
    }

    private boolean onSmaeLine(int[][] points){
        for(int i = 0; i < points.length - 2; i++){
            if(orientation(points[i], points[i + 1], points[i + 2]) != 0)
                return false;
        }

        return true;
    }

    //clock or couter-clock?
    //on couter-clock, 1
    //on clock 2
    private int orientation(int[] p, int[] q, int[] r) {
        int val = (r[0] - p[0]) * (q[1] - r[1]) - (r[1] - p[1]) * (q[0] - r[0]) ;

        if (val == 0)
            return 0;
        return val > 0 ? 1 : 2;
    }
}

```

589 N ary tree preorder traversal

执行结果： 通过 显示详情 >

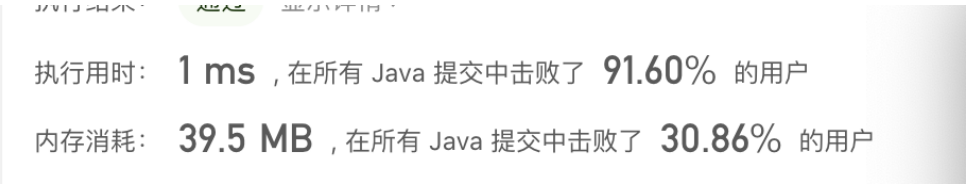
执行用时： 1 ms , 在所有 Java 提交中击败了 91.87% 的用户

内存消耗： 39.5 MB , 在所有 Java 提交中击败了 20.07% 的用户

```
    List<Integer> res = new ArrayList<>();
    public List<Integer> preorder(Node root) {
        myPreorder(root);
        return res;
```

```
    }

    private void myPreorder(Node root){
        if(root == null)
            return;

        res.add(root.val);
        for(Node n : root.children)
            myPreorder(n);
    }
```

590 N Ary Tree Postorder Traversal




```
    public List<Integer> postorder(Node root) {
        myPreorder(root);
        return res;
    }

    private void myPreorder(Node root){
        if(root == null)
            return;


        for(Node n : root.children)
            myPreorder(n);

        res.add(root.val);
    }
```

591 Tag Validator

591. Tag Validator

难度 困难 31

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

```
1
```

592 Fraction Addition and Subtraction

592. Fraction Addition and Subtraction

难度 中等 40

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:

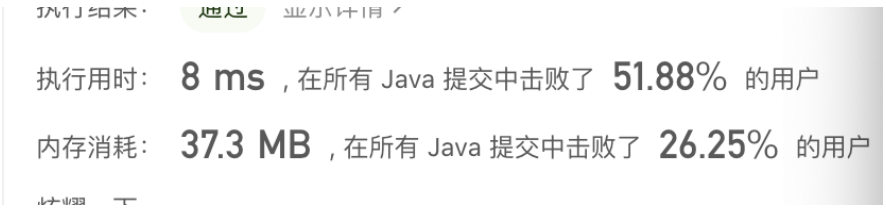
```
Input:"1/3-1/2"
Output: "-1/6"
```

Example 4:

```
Input:"5/3+1/3"
Output: "2/1"
```

Note:

1. The input string only contains `'0'` to `'9'`, `'/'`, `'+'` and




```
/*
    1. split using '+'

    2. for each one,
    using hashMap to store the denominator

    3. convert with those who can be convert

```

```
    4. leave the part that cannot merge,
        for every one ,just take a common and return back
*/
class Solution {
    public String fractionAddition(String expression) {
        /*
            1. split using '+'

            2. for each one,
            using hashMap to store the denominator

            3. convert with those who can be convert

            4. leave the part that cannot merge,
                for every one ,just take a common and return back
        */

        int integerPart = 0;

//       to classify it into differnet category
        HashMap<Integer, Integer> map = getClassifiedString(expression);

        HashSet<String> remains = new HashSet<>();
        String begin = "0/1";

        for(Integer down : map.keySet()){
            int down1 = Integer.parseInt(begin.substring(begin.indexOf('/') + 1));
            int down2 = down;

            int commonDown = down1 * down2;
            int up1   = Integer.parseInt(begin.substring(0, begin.indexOf('/'))) *
down2;
            int up2   = map.get(down) * down1;

            begin = (up1 + up2) + "/" + (commonDown);

            int upRes = Integer.parseInt(begin.substring(0, begin.indexOf('/')));
            int downRes = Integer.parseInt(begin.substring(begin.indexOf('/') +
1));



            begin = upRes + "/" + downRes;
        }

        int up   = Integer.parseInt(begin.substring(0, begin.indexOf('/')));
        int down = Integer.parseInt(begin.substring(begin.indexOf('/') + 1));

        while(true){
            boolean reduceable = false;
```

```
            for(int i = 2;  i <= 9; i++){
                if(up % i == 0 && down % i == 0){
                    up /= i;
                    down /= i;
                    reduceable = true;
                }
            }

            if(reduceable == false)
                break;
        }

        return up + "/" + down;
    }

    //              de      num
    public HashMap<Integer, Integer> getClassifiedString(String expression){
        int left = 0, right = 0;
        int len = expression.length();

        HashMap<Integer, Integer> map = new HashMap<>();
        while(right < len){
            while(right < len && expression.charAt(right) != '/')
                right++;

            int temp = right;
            while(right < len && (expression.charAt(right) != '+' &&
expression.charAt(right) != '-'))
                right++;

            int down =  Integer.parseInt(expression.substring(temp + 1, right));


            int up = Integer.parseInt(expression.substring(left, temp));
            if(map.containsKey(down)){
                int oldValue = map.get(down);
                oldValue += up;
                map.put(down, oldValue);
            }else
                map.put(down, up);

            left = right;
        }

        return map;
    }
}
```

593 Valid Square

593. Valid Square

难度 中等 59

Given the coordinates of four points in 2D space `p1`, `p2`, `p3` and `p4`, return `true` *if the four points construct a square*.

The coordinate of a point p_i is represented as $[x_i, y_i]$. The input is **not** given in any order.

A **valid square** has four equal sides with positive length and four equal angles (90-degree angles).

Example 1:

```
Input: p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,1]
Output: true
```

Example 2:

```
Input: p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,12]
Output: false
```

Example 3:

```
Input: p1 = [1,0], p2 = [-1,0], p3 = [0,1], p4 = [0,-1]
Output: true
```

执行结果: 通过 显示详情 >

执行用时: 11 ms , 在所有 Java 提交中击败了 5.73% 的用户

内存消耗: 38.6 MB , 在所有 Java 提交中击败了 5.21% 的用户

炫耀一下:

```java
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {

        HashSet<String> set = new HashSet<>();
        int[][] points = new int[4][2];
        points[0][0] = p1[0];       set.add(p1[0] + "@" + p1[1]);
        points[0][1] = p1[1];
        points[1][0] = p2[0];       set.add(p2[0] + "@" + p2[1]);
        points[1][1] = p2[1];
        points[2][0] = p3[0];       set.add(p3[0] + "@" + p3[1]);
        points[2][1] = p3[1];
        points[3][0] = p4[0];       set.add(p4[0] + "@" + p4[1]);
        points[3][1] = p4[1];
        if(set.size() != 4)
            return false;

        for(int i = 0; i < points.length; i++){
            for(int j = 0; j < points.length; j++){
                if(j == i)
                    continue;
                for(int k = 0; k < points.length; k++){
                    if(k == j || k == i)
                        continue;

                    if(isVertical(points[i], points[j], points[k])){
                        for(int m = 0; m < points.length; m++){
                            if(m == i || m == k || m == j)
                                continue;

                            if(isVertical(points[i], points[m], points[k]) &&
                              isVertical(points[m], points[i], points[j]))
                                if(distance(points[i], points[j]) ==
distance(points[i], points[m]))
                                    return true;
                        }
                    }
                }
            }
        }

        return false;
    }

    private int distance(int[] p1, int[] p2){

        return (p1[0] - p2[0]) * (p1[0] - p2[0]) + (p1[1] - p2[1] ) *(p1[1] -
p2[1]);
    }
    /*
        to see if line p1p2 and p2p3 can be vertical
```

```
    */
    private boolean isVertical(int[] p1, int[] p2, int[] p3){
        return (p2[1] - p1[1]) * (p3[1] - p2[1]) +
            (p2[0] - p1[0]) * (p3[0] - p2[0]) == 0;
    }
```

594 Longest Harmonious Subsequence

594. Longest Harmonious Subsequence

难度 简单 153

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

Given an integer array `nums`, return *the length of its longest harmonious subsequence among all its possible subsequences*.

A **subsequence** of array is a sequence that can be derived from t array by deleting some or no elements without changing the orde the remaining elements.

Example 1:

```
Input: nums = [1,3,2,2,5,2,3,7]
Output: 5
Explanation: The longest harmonious subsequence i
[3,2,2,2,3].
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: 2
```

Example 3:

```
Input: nums = [1,1,1,1]
Output: 0
```

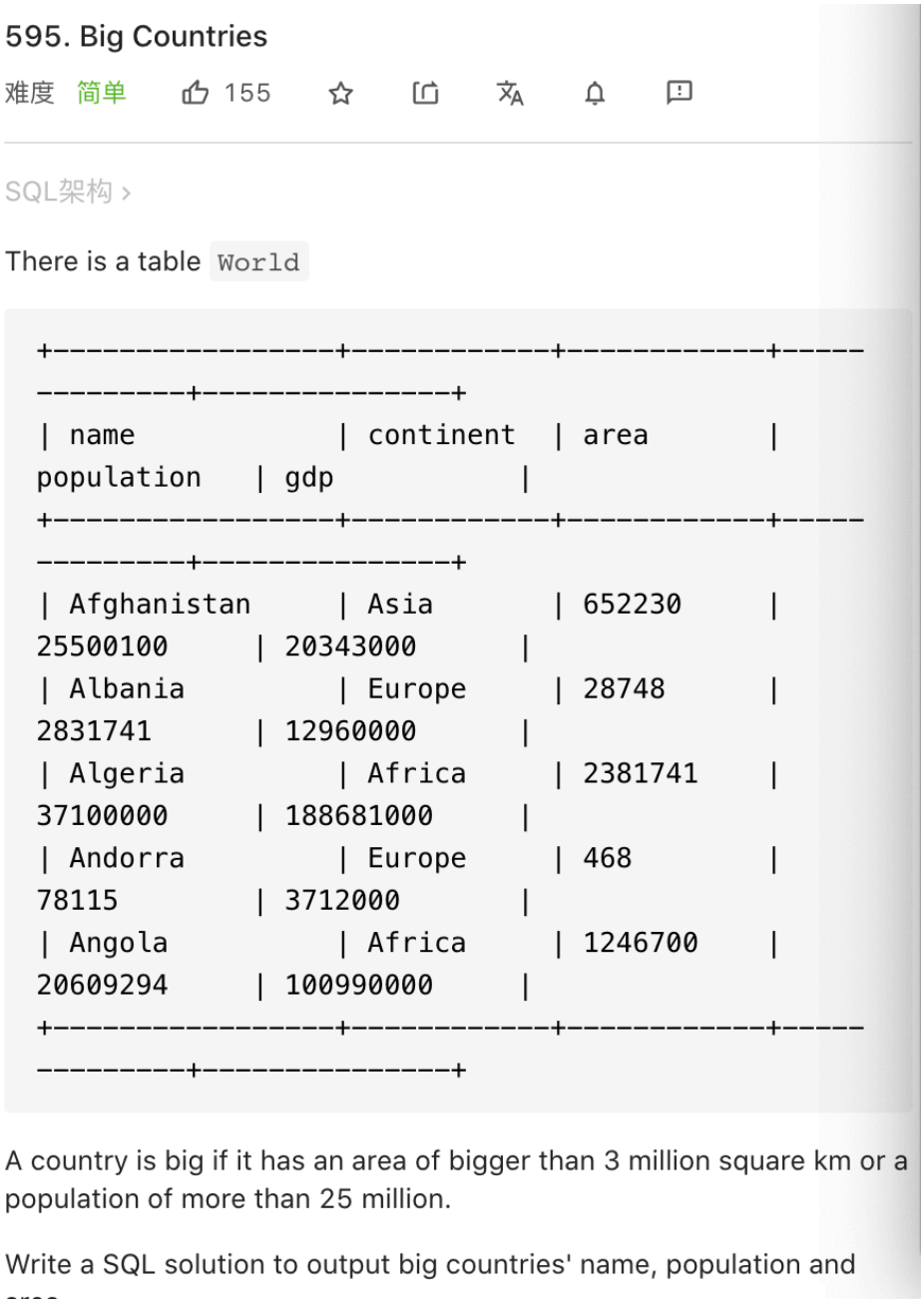
执行用时： 21 ms , 在所有 Java 提交中击败了 52.22% 的用户

内存消耗： 39.4 MB , 在所有 Java 提交中击败了 67.69% 的用户

```java
public int findLHS(int[] nums) {
    HashMap<Integer, Integer> map = new HashMap<>();

    for(int num : nums){
        map.put(num, map.getOrDefault(num, 0) + 1);
    }

    int res = 0;

    for(Integer num : map.keySet()){
```

```
            if(map.containsKey(num - 1))
                res = Math.max(res, map.get(num - 1) + map.get(num));

            if(map.containsKey(num + 1))
                res = Math.max(res, map.get(num + 1) + map.get(num));
        }

        return res;
    }
```

595 Big Countries



595. Big Countries

难度 简单 155

SQL架构 >

There is a table `World`

```
+-----------------+------------+------------+-----
---------+---------------+
| name            | continent  | area       |
population   | gdp           |
+-----------------+------------+------------+-----
---------+---------------+
| Afghanistan     | Asia       | 652230     |
25500100     | 20343000      |
| Albania         | Europe     | 28748      |
2831741      | 12960000      |
| Algeria         | Africa     | 2381741    |
37100000     | 188681000     |
| Andorra         | Europe     | 468        |
78115        | 3712000       |
| Angola          | Africa     | 1246700    |
20609294     | 100990000     |
+-----------------+------------+------------+-----
---------+---------------+
```

A country is big if it has an area of bigger than 3 million square km or a population of more than 25 million.

Write a SQL solution to output big countries' name, population and

```mysql
# Write your MySQL query statement below
SELECT name, population, area
FROM World
WHERE area >= 3000000 OR population >= 25000000;
```

597 Friend Requests I

597. Friend Requests I: Overall Acceptance Rate

难度 简单 39

SQL架构 >

Table: `FriendRequest`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| sender_id      | int     |
| send_to_id     | int     |
| request_date   | date    |
+----------------+---------+
There is no primary key for this table, it may
contain duplicates.
This table contains the ID of the user who sent
the request, the ID of the user who received the
request, and the date of the request.
```

Table: `RequestAccepted`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| requester_id   | int     |
| accepter_id    | int     |
| accept_date    | date    |
+----------------+---------+
There is no primary key for this table, it may
```

```
1
```

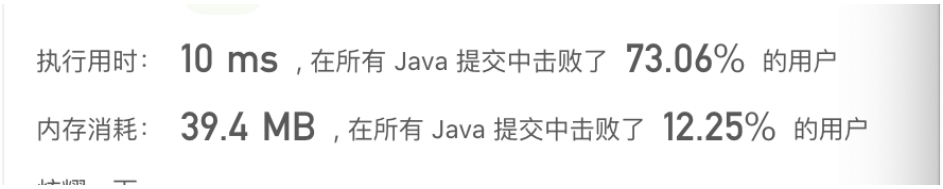
598 Range Addition II

执行用时： 12 ms ，在所有 Java 提交中击败了 73.27% 的用户

内存消耗： 38.8 MB ，在所有 Java 提交中击败了 5.29% 的用户

```java
public int maxCount(int m, int n, int[][] ops) {
    if(ops.length == 0)
        return m * n;

    int[] row = new int[m];
    int[] col = new int[n];


    for(int[] op : ops){
        for(int i = 0; i < op[0]; i++)
            row[i]++;
        for(int j = 0; j < op[1]; j++)
            col[j]++;
    }

    int rowIndex = 0, colIndex = 0;
    while(rowIndex < m && row[rowIndex] == row[0])
        rowIndex++;
    while(colIndex < n && col[colIndex] == col[0])
        colIndex++;

    return rowIndex * colIndex;

}
```

599




```java
public String[] findRestaurant(String[] list1, String[] list2) {
    int len1 = list1.length;
    int len2 = list2.length;

    //      content, index
    HashMap<String, Integer> mapL2 = new HashMap<>();
    for(int i = 0; i < len2; i++)
        mapL2.put(list2[i], i);

    int minIndexSum = len1 + len2;

    int index1 = 0;
```

```
        HashSet<String> res = new HashSet<>();

        /**
         * go traverse the list1
         *   whenever we find one that exists in list2
         *   get the index for them and compare with the minIndexSum
         *
         *   If it's smaller, clear the original result set and add new String
         */
        while(index1 < len1){
            String curRest = list1[index1];

            if(!mapL2.containsKey(curRest)){
                index1++;
                continue;
            }

            int index2 = mapL2.get(curRest);
            if(index1 + index2 > minIndexSum){
                index1++;
                continue;
            }else if(index1 + index2 == minIndexSum){
                index1++;
                res.add(curRest);
                continue;
            }else{
                res.clear();
                minIndexSum = index1 + index2;
                index1++;

                res.add(curRest);
            }
        }

        //form the answer
        String[] ans = new String[res.size()];
        int index = 0;
        for(String str : res)
            ans[index++] = str;

        return ans;

    }
```

600 Non-negative Integers without Consecutive Ones

600. Non-negative Integers without Consecutive Ones

难度 困难 87

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

通过次数 3,073 | 提交次数 9,212

```java
/*
    source: https://leetcode.com/problems/non-negative-integers-without-consecutive-ones/discuss/103754/C%2B%2B-Non-DP-O(32)-Fibonacci-solution
*/
    public int findIntegers(int num) {
        int[] dp = new int[32];
        dp[0] = 1;
        dp[1] = 2;

        for(int i = 2; i < 32; i++)
            dp[i] = dp[i - 1] + dp[i - 2];

```

```
        String numStr = getBinary(num);

        int res = 0;
        for(int i = 0; i < numStr.length(); i++){
            if(numStr.charAt(i) == '0')
                continue;

            res += dp[numStr.length() - i - 1];
            if(i != 0 && numStr.charAt(i - 1) == '1')
                return res;
        }

        return res + 1;
    }

    private String getBinary(int num){
        StringBuilder sb = new StringBuilder();

        while(num > 0){
            sb.insert(0, num & 1);
            num >>= 1;
        }

        return sb.toString();
    }
```