

## 105.从前序与中序遍历序列构造二叉树

105. 从前序与中序遍历序列构造二叉树

难度 中等 426 收藏 大众

根据一棵树的前序遍历与中序遍历构造二叉树。

注意：

你可以假设树中没有重复的元素。

例如，给出

```
前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]
```

返回如下的二叉树：

```
    3
   / \
  9  20
   / \
  15  7
```

通过次数 62,557 | 提交次数 96,079

```
1 class Solution {
2     private int[] mInorder;
3     private int[] mPreorder;
4     private int mPreorderIdx;
5     private HashMap<Integer, Integer> mMap;
6     public TreeNode buildTree(int[] preorder, int[] inorder) {
7         if (preorder == null || inorder == null) return null;
8         if (preorder.length != inorder.length) return null;
9         this.mInorder = inorder;
10        this.mPreorder = preorder;
11        this.mMap = new HashMap<>();
12        for (int i = 0; i < mInorder.length; i++) {
13            mMap.put(mInorder[i], i);
14        }
15        mPreorderIdx = 0;
16        return builder(0, mInorder.length - 1);
17    }
18    private TreeNode builder(int lid, int rid) {
19        if (lid > rid)
20            return null;
21        int pValue = mPreorder[mPreorderIdx++];
22        int index = mMap.get(pValue);
23        TreeNode node = new TreeNode(pValue);
24        node.left = builder(lid, index - 1);
25        node.right = builder(index + 1, rid);
26        return node;
27    }
}
```

## 106.从中序与后序遍历序列构造二叉树

106. 从中序与后序遍历序列构造二叉树

难度 中等 192 收藏 大众

根据一棵树的中序遍历与后序遍历构造二叉树。

注意：

你可以假设树中没有重复的元素。

例如，给出

```
中序遍历 inorder = [9,3,15,20,7]
后序遍历 postorder = [9,15,7,20,3]
```

返回如下的二叉树：

```
    3
   / \
  9  20
   / \
  15  7
```

```
1 class Solution {
2     private int mPostIdx;
3     private int[] mInorders, mPostorders;
4     private HashMap<Integer, Integer> mMap = new HashMap<>();
5     public TreeNode buildTree(int[] inorder, int[] postorder) {
6         if (inorder == null || postorder == null) return null;
7         if (inorder.length != postorder.length) return null;
8         this.mInorders = inorder;
9         this.mPostorders = postorder;
10        this.mPostIdx = postorder.length - 1;
11        for (int idx = 0; idx < mInorders.length; idx++) {
12            mMap.put(mInorders[idx], idx);
13        }
14        return builder(0, mInorders.length - 1);
15    }
16    private TreeNode builder(int inLid, int inRid) {
17        if (inLid > inRid) return null;
18        int postValue = mPostorders[mPostIdx];
19        TreeNode node = new TreeNode(postValue);
20        int index = mMap.get(postValue);
21        mPostIdx--;
22        node.right = builder(index + 1, inRid);
23        node.left = builder(inLid, index - 1);
24        return node;
25    }
}
```

## 156.上下翻转二叉树

156. 上下翻转二叉树

难度 中等 24 收藏 大众

给定一个二叉树，其中所有的右节点要么是具有兄弟节点（拥有相同父节点的左节点）的叶节点，要么为空，将此二叉树上下翻转并将它变成一棵树，原来的右节点将转换成左叶节点。返回新的根。

例子：

```
输入： [1,2,3,4,5]
```

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

```
1 class Solution {
2     public TreeNode upsideDownBinaryTree(TreeNode root) {
3         TreeNode prev = null;
4         TreeNode prevRight = null;
5         while (root != null) {
6             TreeNode left = root.left;
7             root.left = prevRight;
8             prevRight = root.right;
9             root.right = prev;
10            prev = root;
11            root = left;
12        }
13        return prev;
14    }
}
```

## 157.用 Read4 读取 N 个字符

157. 用 Read4 读取 N 个字符

难度 简单 14 收藏 大众

给你一个文件，并且该文件只能通过给定的 `read4` 方法来读取，请实现一个方法使其能够读取 n 个字符。

`read4` 方法：

API `read4` 可以从文件中读取 4 个连续的字符，并且将它们写入缓存数组 `buf` 中。

返回值为实际读取的字符个数。

```
1 public class Solution extends Reader4 {
2     public int read(char[] buf, int n) {
3         int sum = 0;
4         int cnt = 0;
5         char[] tmp = new char[4];
6         while ((cnt = read4(tmp)) > 0) {
7             for (int i = 0; i < cnt; i++) {
8                 buf[sum + i] = tmp[i];
9             }
10            sum += cnt;
11        }
12        return sum > n ? n : sum;
13    }
14 }
```

## 158. 用 Read4 读取 N 个字符 II

158. 用 Read4 读取 N 个字符 II

难度 困难 16 ❤️ ⚡ 🌐 💬 🔍

给你一个文件，并且该文件只能通过给定的 `read4` 方法来读取，请实现一个方法使其能够读取 `n` 个字符。注意：你的 `read` 方法可能会被调用多次。

`read4` 的定义：

参数类型: `char[] buf`  
返回类型: `int`

注意: `buf[]` 是目标缓存区不是源缓存区，`read4` 的返回结果将会复制到 `buf[]` 当中。

## 159. 至多包含两个不同字符的最长子串

159. 至多包含两个不同字符的最长子串

难度 中等 37 ❤️ ⚡ 🌐 💬 🔍

给定一个字符串 `s`，找出至多包含两个不同字符的最长子串 `t`。

示例 1：

输入: "ecea"  
输出: 3  
解释: `t` 是 "ece"，长度为3。

示例 2：

输入: "ccaaabbb"  
输出: 5  
解释: `t` 是 "aabbb"，长度为5。

```
1 public class Solution extends Reader4 {  
2     private int mIndex = 0, mCount = 0;  
3     private char[] mBuffer = new char[4];  
4     public int read(char[] buf, int n) {  
5         int ans = 0;  
6         while (ans < n) {  
7             if (mIndex == 0) mCount = read4(mBuffer);  
8             if (mCount == 0) break;  
9             while (ans < n && mIndex < mCount)  
10                 buf[ans++] = mBuffer[mIndex++];  
11             if (mIndex == mCount) mIndex = 0;  
12             if (mCount < 4) break;  
13         }  
14     }  
15     return ans;  
16 }  
17 }
```

## 161. 相隔为 1 的编辑距离

161. 相隔为 1 的编辑距离

难度 中等 21 ❤️ ⚡ 🌐 💬 🔍

给定两个字符串 `s` 和 `t`，判断他们的编辑距离是否为 1。

注意：

满足编辑距离等于 1 有三种可能的情形：

1. 往 `s` 中插入一个字符得到 `t`
2. 从 `s` 中删除一个字符得到 `t`
3. 在 `s` 中替换一个字符得到 `t`

示例 1：

输入: `s = "ab"`, `t = "acb"`  
输出: true  
解释: 可以将 'c' 插入字符串 `s` 来得到 `t`。

示例 2：

输入: `s = "cab"`, `t = "ad"`  
输出: false  
解释: 无法通过 1 步操作使 `s` 变为 `t`。

示例 3：

输入: `s = "1203"`, `t = "1213"`  
输出: true  
解释: 可以将字符串 `s` 中的 '0' 替换为 '1' 来得到 `t`。

```
1 class Solution {  
2     public int lengthOfLongestSubstringTwoDistinct(String s) {  
3         if (s == null || s.isEmpty()) return 0;  
4         int len = s.length();  
5         if (len < 3) return len;  
6         int lid = 0, rid = 0;  
7         char[] chs = s.toCharArray();  
8         HashMap<Character, Integer> map = new HashMap<>();  
9         int max = 2;  
10        while (rid < len) {  
11            if (map.size() < 3) map.put(chs[rid], rid++);  
12            if (map.size() == 3) {  
13                int delIdx = Collections.min(map.values());  
14                map.remove(chs[delIdx]);  
15                lid = delIdx + 1;  
16            }  
17            max = Math.max(max, rid - lid);  
18        }  
19    }  
20    return max;  
21 }
```

```
1 class Solution {  
2     public boolean isOneEditDistance(String s, String t) {  
3         int ns = s.length();  
4         int nt = t.length();  
5  
6         // Ensure that s is shorter than t.  
7         if (ns > nt)  
8             return isOneEditDistance(t, s);  
9  
10        // The strings are NOT one edit away distance  
11        // if the length diff is more than 1.  
12        if (nt - ns > 1)  
13            return false;  
14  
15        for (int i = 0; i < ns; i++)  
16            if (s.charAt(i) != t.charAt(i))  
17                // if strings have the same length  
18                if (ns == nt)  
19                    return s.substring(i + 1).equals(t.substring(i + 1));  
20                // if strings have different lengths  
21                else  
22                    return s.substring(i).equals(t.substring(i + 1));  
23  
24        // If there is no diffs on ns distance  
25        // the strings are one edit away only if  
26        // t has one more character.  
27        return (ns + 1 == nt);  
28    }  
29 }
```

### 163. 缺失的区间

163. 缺失的区间  
难度 中等 ⌂ 17 ❤️ ⌂ 🎯 ⌂ ⌂ ⌂

给定一个排序的整数数组 `nums`，其中元素的范围在 闭区间 `[lower, upper]` 当中，返回不包含在数组中的缺失区间。

示例：

输入: `nums = [0, 1, 3, 50, 75]`, `lower = 0` 和 `upper = 99`,  
输出: `["2", "4->49", "51->74", "76->99"]`

```
1 class Solution {
2     public List<String> findMissingRanges(int[] nums, int lower, int upper) {
3         List<String> res = new ArrayList<>();
4         long pre = (long)lower - 1; // prevent 'int' overflow
5         for (int i = 0; i < nums.length; i++) {
6             if (nums[i] - pre == 2) res.add(String.valueOf(pre + 1));
7             else if (nums[i] - pre > 2) res.add((pre + 1) + "->" + (nums[i] - 1));
8             pre = nums[i]; // 'int' to 'long'
9         }
10        if (upper - pre == 1) res.add(String.valueOf(pre + 1));
11        else if (upper - pre > 1) res.add((pre + 1) + "->" + upper);
12    }
13 }
14 }
```

### 170. 两数之和 III - 数据结构设计

170. 两数之和 III - 数据结构设计

难度 简单 ⌂ 16 ❤️ ⌂ 🎯 ⌂ ⌂ ⌂

设计并实现一个 `TwoSum` 的类，使该类需要支持 `add` 和 `find` 的操作。

`add` 操作 - 对内部数据结构增加一个数。

`find` 操作 - 寻找内部数据结构中是否存在一对整数，使得两数之和与给定的数相等。

示例 1：

```
add(1); add(3); add(5);
find(4) -> true
find(7) -> false
```

示例 2：

```
add(3); add(1); add(2);
find(3) -> true
find(6) -> false
```

```
1 class TwoSum {
2     private Set<Integer> all = new HashSet<>();
3     private Set<Integer> duplicate = new HashSet<>();
4     public TwoSum() {
5     }
6     public void add(int number) {
7         if (all.contains(number)) duplicate.add(number);
8         else all.add(number);
9     }
10    public boolean find(int value) {
11        int target;
12        for (int num: all) {
13            target = value - num;
14            if (target == num && duplicate.contains(target)) return true;
15            if (target != num && all.contains(target)) return true;
16        }
17    }
18 }
19 }
20 }
```

### 186. 翻转字符串里的单词 II

186. 翻转字符串里的单词 II

难度 中等 ⌂ 22 ❤️ ⌂ 🎯 ⌂ ⌂ ⌂

给定一个字符串，逐个翻转字符串中的每个单词。

示例：

输入: `["t", "h", "e", " ", "s", "h", "y", " ", "i", "s", "h", "b", "l", "u", "e"]`
输出: `["b", "l", "u", "e", " ", "i", "s", "h", "k", "y", "t", "h", "e"]`

注意：

- 单词的定义是不包含空格的一系列字符
- 输入字符串中不会包含前置或尾随的空格
- 单词与单词之间永远是以单个空格隔开的

进阶：使用  $O(1)$  额外空间复杂度的原地解法。

```
1 class Solution {
2     private void reverse(char[] s, int start, int end) {
3         while (start < end) {
4             char tmp = s[start];
5             s[start] = s[end]; s[end] = tmp; start++; end--;
6         }
7     }
8     // 两次翻转即可，第一次全局翻转，第二次翻转各个单词
9     public void reverseWords(char[] s) {
10        int len = s.length, start = 0;
11        reverse(s, 0, len - 1);
12        for (int i = 0; i < len; i++) {
13            if (s[i] == ' ') { // 翻转前面的单词
14                reverse(s, start, i-1);
15                start = i + 1;
16            }
17        }
18        reverse(s, start, len - 1); // 翻转最后一个单词
19    }
20 }
```



## 246. 中心对称数

### 246. 中心对称数

难度 简单 ✓ 10 ❤️ ⚡ 🎁 ✨

中心对称数是指一个数字在旋转了 180 度之后看起来依旧相同的数字（或者上下颠倒地看）。

请写一个函数来判断该数字是否是中心对称数，其输入将会以一个字符串的形式来表达数字。

示例 1：

输入： "69"  
输出： true

示例 2：

输入： "88"  
输出： true

示例 3：

输入： "962"  
输出： false

```
1 class Solution {
2     public boolean isStrobogrammatic(String num) {
3         Map<Character, Character> map = new HashMap<>();
4         map.put('6', '9');
5         map.put('9', '6');
6         map.put('1', '1');
7         map.put('0', '0');
8         map.put('8', '8');
9         for (int idx = 0; idx < num.length(); idx++) {
10            Character cher = map.get(num.charAt(idx));
11            if (cher == null) {
12                return false;
13            }
14            if (num.charAt(num.length() - 1 - idx) != cher.charValue()) {
15                return false;
16            }
17        }
18        return true;
19    }
20 }
```

## 247. 中心对称数 II

### 247. 中心对称数 II

难度 中等 ✓ 18 ❤️ ⚡ 🎁 ✨

中心对称数是指一个数字在旋转了 180 度之后看起来依旧相同的数字（或者上下颠倒地看）。

找到所有长度为 n 的中心对称数。

示例：

输入： n = 2  
输出： ["11", "69", "88", "96"]

通过次数 1,913 | 提交次数 3,963

在真实的面试中遇到过这道题？

是 否

贡献者

相关企业 i

相关标签

```
1 class Solution {
2     private final char[][] MAP = {{'0', '0'}, {'1', '1'},
3                                     {'8', '8'}, {'6', '9'}, {'9', '6'}};
4     public List<String> findStrobogrammatic(int n) {
5         List<String> res = new ArrayList<>();
6         if (n < 1) return res;
7         char[] chs = new char[n];
8         helper(chs, 0, chs.length - 1, res);
9         return res;
10    }
11    private void helper(char[] chs, int lo, int hi, List<String> res) {
12        if (lo > hi) {
13            if (chs.length == 1 || chs[0] != '0') {
14                res.add(String.valueOf(chs));
15            }
16            return;
17        }
18        for (char[] map : MAP) {
19            if (lo == hi && map[0] != map[1]) continue;
20            chs[lo] = map[0];
21            chs[hi] = map[1];
22            helper(chs, lo + 1, hi - 1, res);
23        }
24    }
25 }
```

## 248. 中心对称数 III

### 248. 中心对称数 III

难度 困难 ✓ 15 ❤️ ⚡ 🎁 ✨

中心对称数是指一个数字在旋转了 180 度之后看起来依旧相同的数字（或者上下颠倒地看）。

写一个函数来计算范围在 [low, high] 之间中心对称数的个数。

示例：

输入： low = "50", high = "100"  
输出： 3  
解释： 69, 88 和 96 是三个在该范围内的中心对称数

注意：

由于范围可能很大，所以 low 和 high 都用字符串表示。

通过次数 840 | 提交次数 1,953

在真实的面试中遇到过这道题？

是 否

贡献者

相关企业 i

相关标签

相似题目

```
1 class Solution {
2     private final char[] MAP = {{'0', '0'}, {'1', '1'},
3                                 {'8', '8'}, {'6', '9'}, {'9', '6'}};
4     public int count = 0;
5     public int strobogrammaticInRange(String low, String high) {
6         int lo = low.length();
7         int hi = high.length();
8         for (int n = lo; n <= hi; n++) {
9             char[] chs = new char[n];
10            getStrobogrammatic(chs, 0, chs.length - 1, low, high);
11        }
12        return count;
13    }
14    public void getStrobogrammatic(char[] chs, int lo, int hi, String low, String high) {
15        if (lo > hi) {
16            if (chs.length == 1 || chs[0] != '0') {
17                String str = String.valueOf(chs);
18                if (compare(str, low) && compare(high, str)) {
19                    count++;
20                }
21            }
22            return;
23        }
24        for (char[] map : MAP) {
25            if (lo == hi && map[0] != map[1]) continue;
26            chs[lo] = map[0];
27            chs[hi] = map[1];
28            getStrobogrammatic(chs, lo + 1, hi - 1, low, high);
29        }
30    }
31    public boolean compare(String s1, String s2) {
32        if (s1.length() == s2.length()) {
33            if (s1.compareTo(s2) >= 0) {
34                return true;
35            } else {
36                return false;
37            }
38        }
39        return true;
40    }
41 }
```

## 249. 移位字符串分组

### 249. 移位字符串分组

难度 中等 ⚡ 12 ❤ 1 文章 0 回复

给定一个字符串，对该字符串可以进行“移位”的操作，也就是将字符串中每个字母都变为其在字母表中后续的字母，比如：“abc” -> “bcd”。这样，我们可以持续进行“移位”操作，从而生成如下移位序列：

“abc” -> “bcd” -> ... -> “xyz”

给定一个包含仅小写字母字符串的列表，将该列表中所有满足“移位”操作规律的组合进行分组并返回。

示例：

```
输入: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"]
输出:
[
    ["abc","bcd","xyz"],
    ["az","ba"],
    ["acef"],
    ["a","z"]
]
```

通过次数 1,740 提交次数 2,908

```
1 class Solution {
2     public List<List<String>> groupStrings(String[] strings) {
3         Map<String, List<String>> map = new HashMap<>();
4         StringBuilder buffer = new StringBuilder();
5         for (String str : strings) {
6             // 计算差值
7             char[] chars = str.toCharArray();
8             for (int i = 1; i < chars.length; i++) {
9                 int diff = chars[i] - chars[i - 1];
10                diff = diff < 0 ? diff + 26 : diff;
11                buffer.append(diff).append(",");
12            }
13            // 如果只有一个元素，则将差值记为 "*"
14            String key = buffer.toString().equals("") ?
15                "*" : buffer.substring(0, buffer.length() - 1);
16            List<String> list = map.getOrDefault(key, new ArrayList<>());
17            list.add(str);
18            map.put(key, list);
19            buffer.delete(0, buffer.length());
20        }
21        List<List<String>> res = new ArrayList<>();
22        for (List<String> list : map.values()) {
23            res.add(list);
24        }
25    }
26    return res;
27 }
```

## 250. 统计同值子树

### 250. 统计同值子树

难度 中等 ⚡ 15 ❤ 1 文章 0 回复

给定一个二叉树，统计该二叉树数值相同的子树个数。

同值子树是指该子树的所有节点都拥有相同的数值。

示例：

输入：root = [5,1,5,5,5,null,5]



输出：4

```
1 class Solution {
2     int mCount = 0;
3
4     public int countUnivalSubtrees(TreeNode root) {
5         dfs(root, 0);
6         return mCount;
7     }
8
9     boolean dfs(TreeNode node, int val) {
10        if (node == null) {
11            return true;
12        }
13        if (!dfs(node.left, node.val)
14            | !dfs(node.right, node.val)) {
15            return false;
16        }
17        mCount++;
18        return node.val == val;
19    }
20 }
```

## 251. 展开二维向量

### 251. 展开二维向量

难度 中等 ⚡ 18 ❤ 1 文章 0 回复

请设计并实现一个能够展开二维向量的迭代器。该迭代器需要支持 `next` 和 `hasNext` 两种操作。、

示例：

```
Vector2D iterator = new Vector2D([[1,2],[3],[4]]);

iterator.next(); // 返回 1
iterator.next(); // 返回 2
iterator.next(); // 返回 3
iterator.hasNext(); // 返回 true
iterator.hasNext(); // 返回 true
iterator.next(); // 返回 4
iterator.hasNext(); // 返回 false
```

```
1 class Vector2D {
2     private Queue<Integer> mQueue = new LinkedList<>();
3
4     public Vector2D(int[][] v) {
5         for (int i = 0; i < v.length; i++) {
6             for (int j = 0; j < v[i].length; j++) {
7                 mQueue.offer(v[i][j]);
8             }
9         }
10    }
11
12    public int next() {
13        return mQueue.poll();
14    }
15
16    public boolean hasNext() {
17        return !mQueue.isEmpty();
18    }
19 }
```

## 252. 会议室

### 252. 会议室

难度 简单 23 喜欢 例题 文章 会议

给定一个会议时间安排的数组，每个会议时间都会包括开始和结束的时间  $[[s_1, e_1], [s_2, e_2], \dots]$  ( $s_i < e_i$ )，请你判断一个人是否能够参加这里面的全部会议。

示例 1：

输入:  $[[0, 30], [5, 10], [15, 20]]$

输出: false

```

1 class Solution {
2     public boolean canAttendMeetings(int[][] intervals) {
3         if (intervals == null || intervals.length == 0)
4             return true;
5         // 以会议的开始时间排序
6         Arrays.sort(intervals, (o1, o2) -> o1[0] - o2[0]);
7         for (int i = 0; i < intervals.length - 1; i++) {
8             // 当前会议的结束时间大于下一个会议的开始时间，则不能参加全部的会议。
9             if (intervals[i][1] > intervals[i + 1][0])
10                return false;
11         }
12     }
13 }

```

## 253. 会议室 II

### 253. 会议室 II

难度 中等 78 喜欢 例题 文章 会议

给定一个会议时间安排的数组，每个会议时间都会包括开始和结束的时间  $[[s_1, e_1], [s_2, e_2], \dots]$  ( $s_i < e_i$ )，为避免会议冲突，同时要考虑到充分利用会议室资源，请你计算至少需要多少间会议室，才能满足这些会议安排。

示例 1：

输入:  $[[0, 30], [5, 10], [15, 20]]$

输出: 2

示例 2：

```

1 class Solution {
2     public int minMeetingRooms(int[][] intervals) {
3         if (intervals == null || intervals.length == 0)
4             return 0;
5         Arrays.sort(intervals, (o1, o2) -> o1[0] == o2[0]
6                     ? o1[1] - o2[1] : o1[0] - o2[0]);
7         PriorityQueue<Integer> heap = new PriorityQueue<>();
8         heap.offer(intervals[0][1]);
9         for (int i = 1; i < intervals.length; i++) {
10            if (intervals[i][0] >= heap.peek())
11                heap.poll();
12            heap.offer(intervals[i][1]);
13        }
14     }
15 }

```

## 254. 因子的组合

### 254. 因子的组合

难度 中等 17 喜欢 例题 文章 会议

整数可以被看作是其因子的乘积。

例如：

$8 = 2 \times 2 \times 2;$   
 $= 2 \times 4.$

请实现一个函数，该函数接收一个整数  $n$  并返回该整数所有的因子组合。

注意：

1. 你可以假定  $n$  为永远为正数。
2. 因子必须大于 1 并且小于  $n$ 。

示例 1：

输入: 1  
输出: []

示例 2：

输入: 37  
输出: []

```

1 class Solution {
2     List<List<Integer>> mList = new ArrayList<>();
3
4     public List<List<Integer>> getFactors(int n) {
5         mList.clear();
6         dfs(new Stack<>(), n);
7         return mList;
8     }
9
10    public void dfs(Stack<Integer> stack, int val) {
11        if (val == 1)
12            return;
13        if (stack.size() != 0) {
14            List<Integer> list = new ArrayList<>(stack);
15            list.add(val);
16            mList.add(list);
17        }
18        for (int i = 2; i <= Math.sqrt(val); i++) {
19            if (val % i == 0 && (stack.isEmpty() || i >= stack.peek())) {
20                stack.add(i);
21                dfs(stack, val / i);
22                stack.pop();
23            }
24        }
25    }

```

## 255. 验证前序遍历序列二叉搜索树

### 255. 验证前序遍历序列二叉搜索树

难度 中等 28 喜欢 例题 文章 会议

给定一个整数数组，你需要验证它是否是一个二叉搜索树正确的先序遍历序列。

你可以假定该序列中的数都是不相同的。

参考以下这颗二叉搜索树：



示例 1：

输入: [5,2,6,1,3]  
输出: false

示例 2：

```

1 class Solution {
2     /*
3      * 先序遍历，如果递减，一定是左子树；  

4      * 如果出现非递减的值，意味着到了某个节点的右子树；  

5      * 利用栈来寻找该节点，最后一个比当前元素小的从栈弹出的元素即为该节点的父亲节点，  

6      * 而且当前元素父节点即为新的下限值；  

7      * 后续的元素一定是比当前的下限值要大的，否则return false.  

8      */
9     public boolean verifyPreorder(int[] preorder) {
10        Stack<Integer> stack = new Stack<>();
11        int min = Integer.MIN_VALUE;
12        for (int i = 0; i < preorder.length; i++) {
13            if (preorder[i] < min) {
14                return false;
15            }
16            while (!stack.isEmpty() && preorder[i] > stack.peek()) {
17                min = stack.pop();
18            }
19            stack.push(preorder[i]);
20        }
21        return true;
22    }
23 }

```

## 256. 粉刷房子

### 256. 粉刷房子

难度 简单 36 喜欢 代码

假如有一排房子，共  $n$  个，每个房子可以被粉刷成红色、蓝色或者绿色这三种颜色中的一种，你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。

当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也是不同的。每个房子粉刷成不同颜色的花费是以一个  $n \times 3$  的矩阵来表示的。

例如， $\text{costs}[0][0]$  表示第 0 号房子粉刷成红色的成本花费； $\text{costs}[1][2]$  表示第 1 号房子粉刷成绿色的花费，以此类推。请你计算出粉刷完所有房子最少的花费成本。

注意：

所有花费均为正整数。

示例：

输入： [[17, 2, 17], [16, 16, 5], [14, 3, 19]]

输出： 10

解释： 将 0 号房子粉刷成蓝色，1 号房子粉刷成绿色，2 号房子粉刷成蓝色。

最少花费：  $2 + 5 + 3 = 10$ 。

```
1 class Solution {
2     public int minCost(int[][] costs) {
3         int preR = 0;
4         int curR = 0;
5         int preB = 0;
6         int curB = 0;
7         int preG = 0;
8         int curG = 0;
9
10        for (int i = 0; i < costs.length; i++) {
11            curR = Math.min(preB, preG) + costs[i][0];
12            curB = Math.min(preR, preG) + costs[i][1];
13            curG = Math.min(preR, preB) + costs[i][2];
14            preR = curR;
15            preB = curB;
16            preG = curG;
17        }
18
19        return Math.min(Math.min(curR, curB), curG);
20    }
21 }
```

## 265. 粉刷房子 II

### 265. 粉刷房子 II

难度 困难 23 喜欢 代码

假如有一排房子，共  $n$  个，每个房子可以被粉刷成  $k$  种颜色中的一种，你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。

当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也是不同的。每个房子粉刷成不同颜色的花费是以一个  $n \times k$  的矩阵来表示的。

例如， $\text{costs}[0][0]$  表示第 0 号房子粉刷成 0 号颜色的成本花费； $\text{costs}[1][2]$  表示第 1 号房子粉刷成 2 号颜色的成本花费，以此类推。请你计算出粉刷完所有房子最少的花费成本。

注意：

所有花费均为正整数。

示例：

输入： [[1, 5, 3], [2, 9, 4]]

输出： 5

解释： 将 0 号房子粉刷成 0 号颜色，1 号房子粉刷成 2 号颜色。最少花费：  $1 + 4 = 5$ ；

或者将 0 号房子粉刷成 2 号颜色，1 号房子粉刷成 0 号颜色。

最少花费：  $3 + 2 = 5$ 。

进阶：  
您能否在  $O(nk)$  的时间复杂度下解决此问题？

```
1 class Solution {
2     public int minCostII(int[][] costs) {
3         if (costs == null || costs.length == 0) {
4             return 0;
5         }
6         int k = costs[0].length;
7         int minCosts = 0, secondMinCost = 0, minColorIndex = -1;
8         for (int[] cost : costs) {
9             int temMinCost = Integer.MAX_VALUE;
10            int temSecondMinCost = Integer.MAX_VALUE;
11            int temMinColorIndex = -1;
12            for (int j = 0; j < k; j++) {
13                int curMinCost = cost[j] + (j == minColorIndex ? secondMinCost : minCosts);
14                if (curMinCost < temMinCost) {
15                    temSecondMinCost = temMinCost;
16                    temMinCost = curMinCost;
17                    temMinColorIndex = j;
18                } else if (curMinCost < temSecondMinCost) {
19                    temSecondMinCost = curMinCost;
20                }
21            }
22            minCosts = temMinCost;
23            minColorIndex = temMinColorIndex;
24            secondMinCost = temSecondMinCost;
25        }
26        return minCosts;
27    }
28 }
```

## 259. 较小的三数之和

### 259. 较小的三数之和

难度 中等 25 喜欢 代码

给定一个长度为  $n$  的整数数组和一个目标值  $target$ ，寻找能够使条件  $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] < target$  成立的三元组  $i, j, k$  个数 ( $0 \leq i < j < k < n$ )。

示例：

输入：  $\text{nums} = [-2, 0, 1, 3]$ ,  $target = 2$

输出： 2

解释： 因为一共有两个三元组满足累加和小于 2：  
[-2, 0, 1]  
[-2, 0, 3]

进阶： 是否能在  $O(n^2)$  的时间复杂度内解决？

通过次数 2,445 提交次数 4,388

在真实的面试中遇到过这道题？

```
1 class Solution {
2     public int threeSumSmaller(int[] nums, int target) {
3         if (nums == null || nums.length < 3) {
4             return 0;
5         }
6         int ans = 0, len = nums.length;
7         Arrays.sort(nums);
8         for (int i = 0; i < len - 2; i++) {
9             int l = i + 1, r = len - 1;
10            while (l < r) {
11                int sum = nums[i] + nums[l] + nums[r];
12                if (sum < target) {
13                    ans += r - l;
14                    l++;
15                } else {
16                    r--;
17                }
18            }
19        }
20        return ans;
21    }
22 }
```

## 260. 只出现一次的数字 III

260. 只出现一次的数字 III

难度 中等 211 喜欢 例题 贡献

给定一个整数数组 `nums`，其中恰好有两个元素只出现一次，其余所有元素均出现两次。找出只出现一次的那两个元素。

示例：

输入： [1,2,1,3,2,5]  
输出： [3,5]

```
1 class Solution {
2     public int[] singleNumber(int[] nums) {
3         Map<Integer, Integer> hashmap = new HashMap<>();
4         for (int n : nums)
5             hashmap.put(n, hashmap.getOrDefault(n, 0) + 1);
6         int[] output = new int[2];
7         int idx = 0;
8         for (Map.Entry<Integer, Integer> item : hashmap.entrySet())
9             if (item.getValue() == 1) output[idx++] = item.getKey();
10    }
11 }
```

## 261. 以图判树

261. 以图判树

难度 中等 29 喜欢 例题 贡献

给定从 0 到  $n-1$  标号的  $n$  个结点，和一个无向边列表（每条边以结点对来表示），请编写一个函数用来判断这些边是否能够形成一个合法有效的树结构。

示例 1：

输入：  $n = 5$ , 边列表 `edges` = [[0,1], [0,2], [0,3], [1,4]]  
输出： true

示例 2：

输入：  $n = 5$ , 边列表 `edges` = [[0,1], [1,2], [2,3], [1,3], [1,4]]  
输出： false

注意：你可以假定边列表 `edges` 中不会出现重复的边。由于所有的边是无向边，边 [0,1] 和边 [1,0] 是相同的，因此不会同时出现在边列表 `edges` 中。

通过次数 2,590 提交次数 5,657

在真实的面试中遇到过这道题？

贡献者

相关企业 i

相关标签

相似题目

```
1 class Solution {
2     public boolean validTree(int n, int[][] edges) {
3         int[][] graph = new int[n][n];
4         for (int[] edge : edges) {
5             graph[edge[0]][edge[1]] = 1;
6             graph[edge[1]][edge[0]] = 1;
7         }
8         Queue<Integer> queue = new LinkedList<>();
9         queue.add(0);
10        boolean[] visited = new boolean[n];
11        while (!queue.isEmpty()) {
12            Integer node = queue.poll();
13            visited[node] = true;
14            for (int i = 0; i < n; i++) {
15                if (graph[node][i] == 1) {
16                    if (visited[i])
17                        return false;
18                    visited[i] = true;
19                    graph[node][i] = 0;
20                    graph[i][node] = 0;
21                    queue.add(i);
22                }
23            }
24        }
25        //判断是否为单连通分量
26        for (int i = 0; i < n; i++) {
27            if (!visited[i])
28                return false;
29        }
30    }
31 }
32 }
```

## 268. 缺失数字

268. 缺失数字

难度 简单 250 喜欢 例题 贡献

给定一个包含  $0, 1, 2, \dots, n$  中  $n$  个数的序列，找出  $0 \dots n$  中没有出现在序列中的那个数。

示例 1：

输入： [3,0,1]  
输出： 2

```
1 class Solution {
2     public int missingNumber(int[] nums) {
3         int sum = 0;
4         for (int i = 1; i <= nums.length; i++) {
5             sum += i;
6             sum -= nums[i - 1];
7         }
8         return sum;
9     }
10 }
```

## 266. 回文排列

### 266. 回文排列

难度 简单 12 感谢 0 贡献 0 回答 0

给定一个字符串，判断该字符串中是否可以通过重新排列组合，形成一个回文字符串。

示例 1：

输入： "code"  
输出： false

```
1 class Solution {  
2     public boolean canPermutePalindrome(String s) {  
3         HashMap<Character, Integer> map = new HashMap<>();  
4         for (int i = 0; i < s.length(); i++) {  
5             map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);  
6         }  
7         int count = 0;  
8         for (char key : map.keySet()) {  
9             count += map.get(key) % 2;  
10        }  
11    return count <= 1;  
12 }  
13 }
```

## 267. 回文排列 II

### 267. 回文排列 II

难度 中等 21 感谢 0 贡献 0 回答 0

给定一个字符串  $s$ ，返回其通过重新排列组合后所有可能的回文字符串，并去除重复的组合。

如不能形成任何回文排列时，则返回一个空列表。

示例 1：

输入： "aabb"  
输出： ["abba", "baab"]

示例 2：

输入： "abc"  
输出： []

通过次数 1,077 | 提交次数 2,670

在真实的面试中遇到过这道题？

是 否

贡献者

相关企业  $i$

相关标签

相似题目

显示提示1

```
1 class Solution {  
2     public List<String> generatePalindromes(String s) {  
3         int[] cnt = new int[256];  
4         for (char ch : s.toCharArray()) {  
5             cnt[ch]++;  
6         }  
7         List<String> res = new ArrayList<>();  
8         int id = -1;  
9         for (int i = 0; i < 256; i++) {  
10            if ((cnt[i] & 1) == 1) {  
11                if (id == -1) id = i;  
12                else return res;  
13            }  
14        }  
15        int len = s.length();  
16        char[] chars = new char[len];  
17        if (id != -1) {  
18            chars[len / 2] = (char) id;  
19            cnt[id]--;  
20        }  
21        dfs(res, chars, cnt, 0);  
22        return res;  
23    }  
24    private void dfs(List<String> res, char[] chars, int[] cnt, int id) {  
25        int n = chars.length;  
26        if (id == n / 2) {  
27            res.add(new String(chars));  
28            return;  
29        }  
30        for (int i = 0; i < 256; i++) {  
31            if (cnt[i] > 0) {  
32                chars[id] = (char) i;  
33                chars[n - 1 - id] = (char) i;  
34                cnt[i] -= 2;  
35                dfs(res, chars, cnt, id + 1);  
36                cnt[i] += 2;  
37            }  
38        }  
39    }  
40 }
```

## 269. 火星词典

### 269. 火星词典

难度 困难 46 感谢 0 贡献 0 回答 0

现有一种使用字母的新鲜语言，这门语言的字母顺序与英语顺序不同。

假设，您并不知道其中字母之间的先后顺序。但是，会收到词典中获得一个不为空的单词列表。因为是从词典中获得的，所以该单词列表内的单词已经按这门新语言的字母顺序进行了排序。

您需要根据这个输入的列表，还原出此语言中已知的字母顺序。

示例 1：

输入：  
[  
 "wrt",  
 "wrf",  
 "er",  
 "ett",  
 "rftt"  
]  
输出： "wertf"

示例 2：

输入：  
[  
 "z",  
 "x"  
]  
输出： "zx"

示例 3：

输入：  
[  
 "z",  
 "x",  
 "y"  
]

```
1 class Solution {  
2     public String alienOrder(String[] words) {  
3         Map<Character, Set<Character>> map = new HashMap<>();  
4         boolean flag = false;  
5         for (int i = 0; i < words.length - 1; i++) {  
6             flag = false;  
7             for (int j = 0; j < words[i].length() && j < words[i + 1].length(); j++) {  
8                 if (words[i].charAt(j) == words[i + 1].charAt(j)) {  
9                     continue;  
10                } else {  
11                    Set<Character> set = map.getOrDefault(words[i].charAt(j), new HashSet<>());  
12                    set.add(words[i + 1].charAt(j));  
13                    map.put(words[i].charAt(j), set);  
14                    flag = true;  
15                    break;  
16                }  
17            }  
18            if (!flag && (words[i].length() > words[i + 1].length())) return new String();  
19        }  
20        int[] indegree = new int[26]; Arrays.fill(indegree, -1);  
21        for (int i = 0; i < words.length; i++) {  
22            for (int j = 0; j < words[i].length(); j++) {  
23                indegree[words[i].charAt(j) - 'a'] = 0;  
24            }  
25            for (Character key : map.keySet()) {  
26                for (Character value : map.get(key)) {  
27                    indegree[value - 'a']++;  
28                }  
29            }  
30            int count = 0;  
31            Queue<Character> queue = new LinkedList<>();  
32            for (int i = 0; i < 26; i++) {  
33                if (indegree[i] != -1) count++;  
34                if (indegree[i] == 0) queue.add((char) ('a' + i));  
35            }  
36            StringBuffer buffer = new StringBuffer();  
37            while (!queue.isEmpty()) {  
38                Character node = queue.poll();  
39                buffer.append(node);  
40                if (map.containsKey(node)) {  
41                    for (Character cher : map.get(node)) {  
42                        indegree[cher - 'a']--;  
43                        if (indegree[cher - 'a'] == 0) queue.add(cher);  
44                    }  
45                }  
46            }  
47            String ans = buffer.toString();  
48            if (ans.length() != count) ans = new String();  
49            return ans;  
50        }  
51    }  
52 }
```

## 270. 最接近的二叉搜索树值

270. 最接近的二叉搜索树值

难度 简单 23 喜欢 10 收藏 10

给定一个不为空的二叉搜索树和一个目标值 target，请在该二叉搜索树中找到最接近目标值 target 的数值。

注意：

- 给定的目标值 target 是一个浮点数
- 题目保证在该二叉搜索树中只会存在一个最接近目标值的数

```
1 class Solution {
2     public int closestValue(TreeNode root, double target) {
3         int min = root.val;
4         while (root != null) {
5             int val = root.val;
6             if (Math.abs(val - target) < Math.abs(min - target))
7                 min = val;
8             root = target < root.val ? root.left : root.right;
9         }
10    return min;
11 }
12 }
```

## 272. 最接近的二叉搜索树值 II

272. 最接近的二叉搜索树值 II

难度 困难 26 喜欢 10 收藏 10

给定一个不为空的二叉搜索树和一个目标值 target，请在该二叉搜索树中找到最接近目标值 target 的 k 个值。

注意：

- 给定的目标值 target 是一个浮点数
- 你可以默认 k 值永远是有效的，即  $k \leq$  总结点数
- 题目保证该二叉搜索树中只会存在一种 k 个值集合最接近目标值

示例：

输入：root = [4,2,5,1,3]，目标值 = 3.714286，且 k = 2  
4  
/ \  
2 5  
/ \  
1 3  
输出：[4,3]

拓展：

假设该二叉搜索树是平衡的，请问您是否能在小于  $O(n)$  ( $n$  为总结点数) 的时间复杂度内解决该问题呢？

```
1 class Solution {
2     public List<Integer> closestKValues(TreeNode root, double target, int k) {
3         List<Integer> ans = new ArrayList<>(k);
4         if (root == null)
5             return ans;
6         inorder(root, target, k, ans);
7         return ans;
8     }
9
10    private void inorder(TreeNode node, double target, int k, List<Integer> list) {
11        if (node == null)
12            return;
13        inorder(node.left, target, k, list);
14        if (list.size() < k) {
15            list.add(node.val);
16        } else if (Math.abs(list.get(0) - target) - Math.abs(node.val - target) > 1e-9d) {
17            list.remove(0);
18            list.add(node.val);
19        } else {
20            return;
21        }
22        inorder(node.right, target, k, list);
23    }
24 }
25
26 }
```

## 271. 字符串的编码与解码

271. 字符串的编码与解码

难度 中等 19 喜欢 10 收藏 10

请你设计一个算法，可以将一个字符串列表 编码成为一个字符串。这个编码后的字符串是可以通过网络进行高效传送的，并且可以在接收端被解码回原来的字符串列表。

1 号机（发送方）有如下函数：

```
string encode(vector<string> strs) {
    // ... your code
    return encoded_string;
}
```

2 号机（接收方）有如下函数：

```
vector<string> decode(string s) {
    //... your code
    return strs;
}
```

1 号机（发送方）执行：

```
string encoded_string = encode(strs);
```

2 号机（接收方）执行：

```
vector<string> strs2 = decode(encoded_string);
```

此时，2 号机（接收方）的 strs2 需要和 1 号机（发送方）的 strs 相同。

请你来实现这个 encode 和 decode 方法。

```
1 public class Codec {
2     private String intToString(int len) {
3         char[] bytes = new char[4];
4         for (int i = 0; i < 4; i++) {
5             bytes[i] = (char) (len >> ((3 - i) * 8) & 0xff);
6         }
7         return new String(bytes);
8     }
9
10    public String encode(List<String> strs) {
11        StringBuilder builder = new StringBuilder();
12        for (String str : strs) {
13            int len = str.length();
14            builder.append(intToString(len));
15            builder.append(str);
16        }
17        return builder.toString();
18    }
19
20    private int stringToInt(String str) {
21        char[] bytes = str.toCharArray();
22        int ret = 0;
23        for (int i = 0; i < 4; i++) {
24            ret = (ret << 8) | bytes[i];
25        }
26        return ret;
27    }
28
29    public List<String> decode(String s) {
30        List<String> ans = new ArrayList<>();
31        int i = 0;
32        while (i < s.length()) {
33            int len = stringToInt(s.substring(i, i + 4));
34            i += 4;
35            ans.add(s.substring(i, i + len));
36            i += len;
37        }
38        return ans;
39    }
40 }
```

## 276. 栅栏涂色

### 276. 栅栏涂色

难度 简单 ⚡ 32 ❤️ ⚡ 🌐 🗂️

有  $k$  种颜色的涂料和一个包含  $n$  个栅栏柱的栅栏，每个栅栏柱可以用其中一种颜色进行上色。

你需要给所有栅栏柱上色，并且保证其中相邻的栅栏柱 最多连续两个颜色相同。然后，返回所有有效涂色的方案数。

注意：  
 $n$  和  $k$  均为非负的整数。

示例：

输入：  $n = 3, k = 2$   
输出： 6

解析：用  $c_1$  表示颜色 1,  $c_2$  表示颜色 2, 所有可能的涂色方案有：

	柱 1	柱 2	柱 3
1	$c_1$	$c_1$	$c_2$
2	$c_1$	$c_2$	$c_1$
3	$c_1$	$c_2$	$c_2$
4	$c_2$	$c_1$	$c_1$
5	$c_2$	$c_1$	$c_2$
6	$c_2$	$c_2$	$c_1$

```
1 class Solution {
2     /**
3      * 动态规划
4      * dp[i] 用来表示 i 个栅栏柱的涂色的方案数，有两种情况：
5      * 若 i 与 i-1 的颜色相同，则表明 i-1 与 i-2 的颜色不同，则 i 的数目为 dp[i-2] * (k-1)
6      * 若 i 与 i-1 的颜色不同，则 i 的数目为 dp[i-1] * (k-1)
7      * 故递推公式为：
8      * dp[i] = dp[i-2] * (k-1) + dp[i-1] * (k-1)
9     */
10    public int numWays(int n, int k) {
11        if (n == 0) {
12            return 0;
13        }
14        if (n < 2) {
15            return k;
16        }
17        if (n == 2) {
18            return k * k;
19        }
20        int[] dp = new int[n];
21        dp[0] = k;
22        dp[1] = k * k;
23        for (int i = 2; i < n; i++) {
24            dp[i] = dp[i - 2] * (k - 1) + dp[i - 1] * (k - 1);
25        }
26        return dp[n - 1];
27    }
28}
```

## 277. 搜索名人

### 277. 搜索名人

难度 中等 ⚡ 34 ❤️ ⚡ 🌐 🗂️

假设你是一个专业的狗仔，参加了一个  $n$  人派对，其中每个人被从 0 到  $n - 1$  标号。在这个派对人群当中可能存在一位“名人”。所谓“名人”的定义是：其他所有  $n - 1$  个人都认识他/她，而他/她并不认识其他任何人。

现在你想要确认这个“名人”是谁，或者确定这里没有“名人”。而你唯一能做的就是问诸如“A 你好呀，请问你认不认识 B 呀？”的问题，以确定 A 是否认识 B。你需要在（渐近意义上）尽可能少的问题内来确定这位“名人”是谁（或者确定这里没有“名人”）。

在本题中，你可以使用辅助函数 `bool knows(a, b)` 获取到 A 是否认识 B。请你来实现一个函数 `int findCelebrity(n)`。

派对最多只会有一个“名人”参加。若“名人”存在，请返回他/她的编号；若“名人”不存在，请返回 -1。

```
1 public class Solution extends Relation {
2     public int findCelebrity(int n) {
3         int cand = 0;
4         for (int i = 1; i < n; i++) {
5             if (knows(cand, i)) {
6                 cand = i;
7             }
8         }
9         for (int i = 0; i < n; i++) {
10            if (cand == i) {
11                continue;
12            }
13            if (knows(cand, i) || !knows(i, cand)) {
14                return -1;
15            }
16        }
17        return cand;
18    }
19}
```

## 279. 完全平方数

### 279. 完全平方数

难度 中等 ⚡ 409 ❤️ ⚡ 🌐 🗂️

给定正整数  $n$ ，找到若干个完全平方数（比如  $1, 4, 9, 16, \dots$ ）使得它们的和等于  $n$ 。你需要让组成和的完全平方数的个数最少。

示例 1：

输入：  $n = 12$   
输出： 3  
解释：  $12 = 4 + 4 + 4.$

示例 2：

输入：  $n = 13$   
输出： 2  
解释：  $13 = 4 + 9.$

通过次数 56,153 提交次数 100,535

在真实的面试中遇到过这道题？

贡献者

```
1 class Solution {
2     public int numSquares(int num) {
3         if (num <= 1) {
4             return num;
5         }
6         ArrayList<Integer> squares = new ArrayList<>();
7         for (int i = 1; i * i <= num; i++) {
8             squares.add(i * i);
9         }
10        HashSet<Integer> queue = new HashSet<>();
11        queue.add(num);
12        int level = 0;
13        while (queue.size() > 0) {
14            level += 1;
15            HashSet<Integer> nextQueue = new HashSet<>();
16            for (Integer remain : queue) {
17                for (Integer square : squares) {
18                    if (remain.intValue() < square.intValue()) {
19                        break;
20                    } else if (remain.intValue() == square.intValue()) {
21                        return level;
22                    } else {
23                        nextQueue.add(remain.intValue() - square.intValue());
24                    }
25                }
26            }
27            queue = nextQueue;
28        }
29        return level;
30    }
31}
```

## 280. 摆動排序

### 280. 摆動排序

难度 中等 ⚡ 17 ❤️ 🔍 文章 ⌂

给你一个无序的数组 `nums`，将该数字 原地 重排后使得 `nums[0] <= nums[1] >= nums[2] <= nums[3]...`。

示例：

输入： `nums = [3,5,2,1,6,4]`  
输出： 一个可能的解答是 `[3,5,1,6,2,4]`

通过次数 2,240 提交次数 3,293

## 281. 锯齿迭代器

### 281. 锯齿迭代器

难度 中等 ⚡ 11 ❤️ 🔍 文章 ⌂

给出两个一维的向量，请你实现一个迭代器，交替返回它们中间的元素。

示例：

输入：  
`v1 = [1,2]`  
`v2 = [3,4,5,6]`  
  
输出： `[1,3,2,4,5,6]`  
  
解析：通过连续调用 `next` 函数直到 `hasNext` 函数返回 `false`，  
`next` 函数返回值的次序应依次为： `[1,3,2,4,5,6]`。

```

1 class Solution {
2     public void wiggleSort(int[] nums) {
3         int len = nums.length;
4         for (int i = 1; i < len; i += 2) {
5             if (nums[i] < nums[i - 1])
6                 swap(nums, i, i - 1);
7             if (i < len - 1 && nums[i] < nums[i + 1])
8                 swap(nums, i, i + 1);
9         }
10    }
11    private void swap(int[] nums, int idx1, int idx2) {
12        nums[idx1] ^= nums[idx2];
13        nums[idx2] ^= nums[idx1];
14        nums[idx1] ^= nums[idx2];
15    }
}

```

## 285. 二叉搜索树中的顺序后继

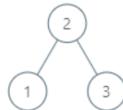
### 285. 二叉搜索树中的顺序后继

难度 中等 ⚡ 26 ❤️ 🔍 文章 ⌂

给你一个二叉搜索树和其中的某一个结点，请你找出该结点在树中顺序后继的节点。

结点 `p` 的后继是值比 `p.val` 大的结点中键值最小的结点。

示例 1：



输入： `root = [2,1,3], p = 1`  
输出： 2

```

1 public class ZigzagIterator {
2     private int mIndex;
3     private List<Integer> mList1, mList2;
4     public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
5         mIndex = 0; mList1 = v1; mList2 = v2;
6     }
7     public int next() {
8         if (mList1 == null || mList1.isEmpty())
9             return mList2.remove(0);
10        if (mList2 == null || mList2.isEmpty())
11            return mList1.remove(0);
12        if (mIndex++ % 2 == 0)
13            return mList1.remove(0);
14        else
15            return mList2.remove(0);
16    }
17    public boolean hasNext() {
18        return (mList1 != null && mList1.size() > 0)
19               || (mList2 != null && mList2.size() > 0);
20    }
21 }

```

## 510. 二叉搜索树中的顺序后继 II

### 510. 二叉搜索树中的中序后继 II

难度 中等 ⚡ 15 ❤️ 🔍 文章 ⌂

给定一棵二叉搜索树和其中的一个节点 `node`，找到该节点在树中的中序后继。

如果节点没有中序后继，请返回 `null`。

一个结点 `node` 的中序后继是键值比 `node.val` 大所有的结点中键值最小的那个。

你可以直接访问结点，但无法直接访问树。每个节点都会有其父节点的引用。节点定义如下：

```

class Node {
    public int val;
    public Node left;
    public Node right;
    public Node parent;
}

```

```

1 class Solution {
2     public TreeNode inorderSuccessor(TreeNode root, TreeNode node) {
3         if (root == null || node == null)
4             return null;
5         TreeNode succ = null;
6         TreeNode curr = root;
7         while (curr != null) {
8             if (curr.val <= node.val) {
9                 curr = curr.right;
10            } else {
11                succ = curr;
12                curr = curr.left;
13            }
14        }
15        return succ;
16    }
}

```

```

1 class Solution {
2     public Node inorderSuccessor(Node node) {
3         if (node == null)
4             return null;
5         if (node.right != null) {
6             node = node.right;
7             while (node.left != null)
8                 node = node.left;
9             return node;
10        }
11        while (node.parent != null && node.parent.val < node.val) {
12            node = node.parent;
13        }
14        return node.parent;
15    }
}

```

## 130. 被围绕的区域 (DFS)

130. 被围绕的区域

难度 中等 击 204 喜欢 48 举报

给定一个二维的矩阵，包含 'X' 和 'O' (字母 O)。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例：

```
X X X X  
X O O X  
X X O X  
X O X X
```

运行你的函数后，矩阵变为：

```
X X X X  
X X X X  
X X X X  
X O X X
```

解释：

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

通过次数 32,331 | 提交次数 80,951

在真实的面试中遇到过这道题？

## 130. 被围绕的区域 (BFS)

130. 被围绕的区域

难度 中等 击 204 喜欢 48 举报

给定一个二维的矩阵，包含 'X' 和 'O' (字母 O)。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例：

```
X X X X  
X O O X  
X X O X  
X O X X
```

运行你的函数后，矩阵变为：

```
X X X X  
X X X X  
X X X X  
X O X X
```

解释：

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

通过次数 32,331 | 提交次数 80,951

在真实的面试中遇到过这道题？

贡献者

```
1 class Solution { // DFS  
2     public void solve(char[][] board) {  
3         if (board == null || board.length == 0 || board[0].length == 0)  
4             return;  
5         int row = board.length, col = board[0].length;  
6         for (int rid = 0; rid < row; rid++) {  
7             if (board[rid][0] == 'O')  
8                 dfs(board, rid, 0);  
9             if (board[rid][col - 1] == 'O')  
10                dfs(board, rid, col - 1);  
11        }  
12        for (int cid = 0; cid < col; cid++) {  
13            if (board[0][cid] == 'O')  
14                dfs(board, 0, cid);  
15            if (board[row - 1][cid] == 'O')  
16                dfs(board, row - 1, cid);  
17        }  
18        for (int rid = 0; rid < row; rid++)  
19            for (int cid = 0; cid < col; cid++)  
20                if (board[rid][cid] == 'I')  
21                    board[rid][cid] = 'O';  
22                else  
23                    board[rid][cid] = 'X';  
24    }  
25    public void dfs(char[][] board, int rid, int cid) {  
26        if (board[rid][cid] == 'I')  
27            return;  
28        board[rid][cid] = 'I';  
29        int row = board.length, col = board[0].length;  
30        if (rid > 0 && board[rid - 1][cid] == 'O')  
31            dfs(board, rid - 1, cid);  
32        if (rid < row - 1 && board[rid + 1][cid] == 'O')  
33            dfs(board, rid + 1, cid);  
34        if (cid > 0 && board[rid][cid - 1] == 'O')  
35            dfs(board, rid, cid - 1);  
36        if (cid < col - 1 && board[rid][cid + 1] == 'O')  
37            dfs(board, rid, cid + 1);  
38    }  
39}
```

```
1 class Solution { // BFS  
2     public void solve(char[][] board) {  
3         if (board == null || board.length == 0 || board[0].length == 0)  
4             return;  
5         int row = board.length, col = board[0].length;  
6         Queue<int[]> queue = new LinkedList<int[]>;  
7         for (int cid = 0; cid < col; cid++) {  
8             if (board[0][cid] == 'O') {  
9                 board[0][cid] = 'I'; queue.offer(new int[]{0, cid});  
10            }  
11            if (board[row - 1][cid] == 'O') {  
12                board[row - 1][cid] = 'I'; queue.offer(new int[]{row - 1, cid});  
13            }  
14        }  
15        for (int rid = 1; rid < row - 1; rid++) {  
16            if (board[rid][0] == 'O') {  
17                board[rid][0] = 'I'; queue.offer(new int[]{rid, 0});  
18            }  
19            if (board[rid][col - 1] == 'O') {  
20                board[rid][col - 1] = 'I'; queue.offer(new int[]{rid, col - 1});  
21            }  
22        }  
23        final int[] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};  
24        while (!queue.isEmpty()) {  
25            int[] node = queue.poll();  
26            for (int[] dir : dirs) {  
27                int[] rIdx = node[0] + dir[0], cIdx = node[1] + dir[1];  
28                if (rIdx < 0 || rIdx >= row || cIdx < 0 || cIdx >= col || board[rIdx][cIdx] != 'O')  
29                    continue;  
30                board[rIdx][cIdx] = 'I';  
31                queue.add(new int[]{rIdx, cIdx});  
32            }  
33        }  
34        for (int rid = 0; rid < row; rid++)  
35            for (int cid = 0; cid < col; cid++)  
36                if (board[rid][cid] == 'O')  
37                    board[rid][cid] = 'X';  
38                else if (board[rid][cid] == 'I')  
39                    board[rid][cid] = 'O';  
40                else {  
41                    continue;  
42                }  
43        }  
44    }
```

## 286. 墙与门

### 286. 墙与门

难度 中等 击 43 喜欢 14 收藏 14

你被给定一个  $m \times n$  的二维网格，网格中有以下三种可能的初始化值：

1. -1 表示墙或是障碍物
2. 0 表示一扇门
3. INF 无限表示一个空的房间。然后，我们用  $2^{31} - 1 = 2147483647$  代表 INF。你可以认为通往门的距离总是小于 2147483647 的。

你要给每个空房间位上填上该房间到 最近 门的距离，如果无法到达门，则填 INF 即可。

示例：

给定二维网格：

```
INF -1 0 INF
INF INF INF -1
INF -1 INF -1
0 -1 INF INF
```

运行完你的函数后，该网格应该变成：

```
3 -1 0 1
2 2 1 -1
1 -1 2 -1
0 -1 3 4
```

通过次数 4,270 | 提交次数 9,656

在真实的面试中遇到过这道题？

是

否

```
1 class Solution {
2     public void wallsAndGates(int[][] rooms) {
3         if (rooms == null || rooms.length == 0 || rooms[0].length == 0)
4             return;
5         int rLen = rooms.length, cLen = rooms[0].length;
6         Queue<Integer> queue = new LinkedList<>();
7         for (int rid = 0; rid < rLen; rid++)
8             for (int cid = 0; cid < cLen; cid++)
9                 if (rooms[rid][cid] == 0)
10                     queue.offer(rid * cLen + cid);
11         bfs(queue, rooms, rLen, cLen);
12     }
13     public void bfs(Queue<Integer> queue, int[][] rooms, int m, int n) {
14         int size = queue.size();
15         for (int k = 0; k < size; k++) {
16             int node = queue.poll();
17             int rid = node / n;
18             int cid = node % n;
19             int val = rooms[rid][cid];
20             if (cid + 1 < n && rooms[rid][cid + 1] == Integer.MAX_VALUE) {
21                 rooms[rid][cid + 1] = val + 1;
22                 queue.offer(rid * n + cid + 1);
23             }
24             if (cid - 1 >= 0 && rooms[rid][cid - 1] == Integer.MAX_VALUE) {
25                 rooms[rid][cid - 1] = val + 1;
26                 queue.offer(rid * n + cid - 1);
27             }
28             if (rid - 1 >= 0 && rooms[rid - 1][cid] == Integer.MAX_VALUE) {
29                 rooms[rid - 1][cid] = val + 1;
30                 queue.offer((rid - 1) * n + cid);
31             }
32             if (rid + 1 < m && rooms[rid + 1][cid] == Integer.MAX_VALUE) {
33                 rooms[rid + 1][cid] = val + 1;
34                 queue.offer((rid + 1) * n + cid);
35             }
36         }
37         if (queue.size() > 0)
38             bfs(queue, rooms, m, n);
39     }
40 }
```

## 296. 最佳的碰头地点

### 296. 最佳的碰头地点

难度 困难 击 20 喜欢 14 收藏 14

有一队人（两人或以上）想要在一个地方碰面，他们希望能够最小化他们的总行走距离。

给你一个 2D 网格，其中各个格子内的值要么是 0，要么是 1。

1 表示某个人的家所处的位置。这里，我们将使用 曼哈顿距离 来计算，其中  $distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|$ 。

示例：

输入：

```
1 - 0 - 0 - 0 - 1
|   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |
0 - 0 - 1 - 0 - 0
```

输出：6

```
1 class Solution {
2     public int minTotalDistance(int[][] grid) {
3         if (grid == null || grid.length == 0 || grid[0].length == 0)
4             return Integer.MAX_VALUE;
5         List<Integer> rows = new ArrayList<>();
6         List<Integer> cols = new ArrayList<>();
7         for (int rid = 0; rid < grid.length; rid++)
8             for (int cid = 0; cid < grid[0].length; cid++)
9                 if (grid[rid][cid] == 1)
10                     rows.add(rid);
11         for (int cid = 0; cid < grid[0].length; cid++)
12             for (int rid = 0; rid < grid.length; rid++)
13                 if (grid[rid][cid] == 1)
14                     cols.add(cid);
15         int rid = rows.get(rows.size() / 2);
16         int cid = cols.get(cols.size() / 2);
17         return calculator(rows, rid) + calculator(cols, cid);
18     }
19     private int calculator(List<Integer> list, int idx) {
20         int res = 0;
21         for (int val: list)
22             res += Math.abs(val - idx);
23         return res;
24     }
25 }
```

## 317. 离建筑物最近的距离

317. 离建筑物最近的距离

难度 困难 击 19 喜欢 0 贡献 0

你是个房地产开发商，想要选择一片空地建一栋大楼。你想把这栋大楼够造在一个距离周边设施都比较方便的地方，通过调研，你希望从它出发能在最短的距离和内抵达周边全部的建筑物。请你计算出这个最佳的选址到周边全部建筑物的最短距离。

注意：

你只能通过向上、下、左、右四个方向上移动。

给你一个由 0、1 和 2 组成的二维网格，其中：

- 0 代表你可以自由通过和选择建造的空地
- 1 代表你无非通行的建筑物
- 2 代表你无非通行的障碍物

示例：

输入： [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

1 - 0 - 2 - 0 - 1  
| | | | |  
0 - 0 - 0 - 0 - 0  
| | | | |  
0 - 0 - 1 - 0 - 0

输出： 7

解析：

```
1 class Solution {
2     private static final int[] DIRS = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};
3     public int shortestDistance(int[][] grid) {
4         if (grid == null || grid.length == 0 || grid[0].length == 0)
5             return 0;
6         int rLen = grid.length, cLen = grid[0].length;
7         int[] total = new int[rLen][cLen];
8         int ans = Integer.MAX_VALUE;
9         int mark = 0; // 标记地
10        for (int rid = 0; rid < rLen; rid++)
11            for (int cid = 0; cid < cLen; cid++)
12                if (grid[rid][cid] == 1) { // 从每一个建筑物开始进行广度优先搜索
13                    ans = bfs(grid, rLen, cLen, rid, cid, mark, total);
14                    mark--; // mark减1, 表示所有空地又被遍历一次
15                }
16        return ans;
17    }
18    private int bfs(int[][] grid, int rLen, int cLen, int rid, int cid, int mark, int[] total) {
19        int res = Integer.MAX_VALUE;
20        Queue<int[]> queue = new LinkedList<int[]>;
21        queue.offer(new int[]{rid, cid, 0});
22        while (!queue.isEmpty()) {
23            int[] node = queue.poll();
24            int dist = node[2];
25            for (int[] dir : DIRS) {
26                int row = node[0] + dir[0], col = node[1] + dir[1];
27                if (row < 0 || row >= rLen || col < 0 || col >= cLen || grid[row][col] != mark)
28                    continue;
29                int temp = dist + 1; // 在搜索的同时计算每一个空格到这个建筑物的距离
30                total[row][col] += temp; // 在搜索的同时将每一个空格到每一个建筑物的距离进行累加
31                res = Math.min(res, total[row][col]); // 取空格到所有建筑物的最小距离
32                queue.add(new int[]{row, col, temp});
33                grid[row][col]--; // 与mark标记相对应
34            }
35        }
36        return res == Integer.MAX_VALUE ? -1 : res;
37    }
}
```

## 489. 扫地机器人

489. 扫地机器人

难度 困难 击 28 喜欢 0 贡献 0

房间（用格栅表示）中有一个扫地机器人。格栅中的每一个格子有空和障碍物两种可能。

扫地机器人提供4个API，可以向前进，向左转或者向右转。每次转弯90度。

当扫地机器人试图进入障碍物格子时，它的碰撞传感器会探测出障碍物，使它停留在原地。

请利用提供的4个API编写让机器人清理整个房间的算法。

```
interface Robot {
    // 若下一个方格为空，则返回true，并移动至该方格
    // 若下一个方格为障碍物，则返回false，并停留在原地
    boolean move();

    // 在调用turnLeft/turnRight后机器人会停留在原位置
    // 每次转弯90度
    void turnLeft();
    void turnRight();

    // 清理所在方格
    void clean();
}
```

```
1 class Solution {
2     public void cleanRoom(Robot robot) {
3         Map<Integer, Set<Integer>> map = new HashMap<Integer, Set<Integer>>();
4         cleanRoom(robot, 0, 0, 2, map);
5     }
6
7     private void cleanRoom(Robot robot, int rid, int cid, int dir, Map<Integer, Set<Integer>> map) {
8         final int[] dirs = {{1, 0}, {0, -1}, {-1, 0}, {0, 1}};
9         robot.clean();
10        map.putIfAbsent(rid, new HashSet<Integer>());
11        map.get(rid).add(cid);
12        for (int i = 0; i < 4; i++) {
13            int rIdx = rid + dirs[dir][0];
14            int cIdx = cid + dirs[dir][1];
15            if (!map.getOrDefault(rIdx, new HashSet<Integer>()).contains(cIdx) && robot.move()) {
16                cleanRoom(robot, rIdx, cIdx, dir, map);
17                backTrack(robot);
18            }
19            robot.turnRight();
20            dir = (dir + 1) % 4;
21        }
22
23        private void backTrack(Robot robot) {
24            robot.turnRight();
25            robot.turnRight();
26            robot.move();
27            robot.turnRight();
28            robot.turnRight();
29        }
30    }
31 }
```

## 994. 腐烂的橘子

994. 腐烂的橘子

难度 中等 击 206 喜欢 0 贡献 0

在给定的网格中，每个单元格可以有以下三个值之一：

- 值 0 代表空单元格；
- 值 1 代表新鲜橘子；
- 值 2 代表腐烂的橘子。

每分钟，任何与腐烂的橘子（在 4 个正方向上）相邻的新鲜橘子都会腐烂。

返回直到单元格中没有新鲜橘子为止所必须经过的最小分钟数。如果不可能，返回 -1。

示例 1：



输入： [[2,1,1],[1,1,0],[0,1,1]]  
输出： 4

示例 2：

输入： [[2,1,1],[0,1,1],[1,0,1]]  
输出： -1  
解释： 左下角的橘子（第 2 行，第 0 列）永远不会腐烂，因为腐烂只会发生在 4 个正向上。

示例 3：

```
1 class Solution { // BFS
2     public int orangesRotting(int[][] grid) {
3         if (grid == null || grid.length == 0 || grid[0].length == 0)
4             return -1;
5         int row = grid.length, col = grid[0].length;
6         Queue<int[]> queue = new LinkedList<int[]>;
7         int count = 0; // count表示新鲜橘子的数量
8         for (int rid = 0; rid < row; rid++) {
9             for (int cid = 0; cid < col; cid++) {
10                if (grid[rid][cid] == 1) {
11                    count++;
12                } else if (grid[rid][cid] == 2) {
13                    queue.offer(new int[]{rid, cid});
14                }
15            }
16        }
17        int round = 0; // round表示腐烂的分钟数
18        while (count > 0 && !queue.isEmpty()) {
19            round++;
20            int size = queue.size();
21            for (int i = 0; i < size; i++) {
22                int[] node = queue.poll();
23                int rid = node[0], cid = node[1];
24                if (rid - 1 >= 0 && grid[rid - 1][cid] == 1) {
25                    grid[rid - 1][cid] = 2; count--; queue.offer(new int[]{rid - 1, cid});
26                }
27                if (rid + 1 < row && grid[rid + 1][cid] == 1) {
28                    grid[rid + 1][cid] = 2; count--; queue.offer(new int[]{rid + 1, cid});
29                }
30                if (cid - 1 >= 0 && grid[rid][cid - 1] == 1) {
31                    grid[rid][cid - 1] = 2; count--; queue.offer(new int[]{rid, cid - 1});
32                }
33                if (cid + 1 < col && grid[rid][cid + 1] == 1) {
34                    grid[rid][cid + 1] = 2; count--; queue.offer(new int[]{rid, cid + 1});
35                }
36            }
37        }
38        return count > 0 ? -1 : round;
39    }
40 }
```

## 288. 单词的唯一缩写

288. 单词的唯一缩写

难度 中等 ⚡ 3 ❤ 1 🌟 2 ✎ 1

一个单词的缩写需要遵循 <起始字母><中间字母数><结尾字母> 这样的格式。

以下是一些单词缩写的范例：

a) it --> it (没有缩写)

1

b) d|o|g --> dg

1 1 1  
1---5---0---5--8

c) i|nternationalizatio|n --> i18n

1  
1---5---0

d) l|ocalizatio|n --> l10n

假设你有一个字典和一个单词，请你判断该单词的缩写在这本字典中是否唯一。若单词的缩写在字典中没有任何 其他 单词与其缩写相同，则被称为单词的唯一缩写。

示例：

```
1 public class ValidWordAbbr {  
2     Map<String, List<String>> mMap;  
3  
4     public ValidWordAbbr(String[] dictionary) {  
5         mMap = new HashMap<>();  
6         for (String word : dictionary) {  
7             String abbr = getAbbr(word);  
8             List<String> list = mMap.getOrDefault(abbr, new ArrayList<>());  
9             if (!list.contains(word)) {  
10                 list.add(word);  
11                 mMap.put(abbr, list);  
12             }  
13         }  
14     }  
15  
16     public boolean isUnique(String word) {  
17         String abbr = getAbbr(word);  
18         List<String> list = mMap.getOrDefault(abbr, new ArrayList<>());  
19         int size = list.size();  
20         if (size == 0)  
21             return true;  
22         else if (size == 1)  
23             return word.equals(list.get(0));  
24         else  
25             return false;  
26     }  
27  
28     private String getAbbr(String word) {  
29         int length = word.length();  
30         if (length <= 2)  
31             return word;  
32         return String.valueOf(word.charAt(0)) + (length - 2) + word.charAt(length - 1);  
33     }  
34 }
```

## 291. 单词规律 II

291. 单词规律 II

难度 困难 ⚡ 15 ❤ 1 🌟 2 ✎ 1

给你一种规律 pattern 和一个字符串 str，请你判断 str 是否遵循其相同的规律。

这里我们指的是 完全遵循，例如 pattern 里的每个字母和字符串 str 中每个非空 单词 之间，存在着双向连接的对应规律。

示例1：

输入：pattern = "abab", str = "redblueredblue"  
输出：true

示例2：

输入：pattern = "aaaa", str = "asdasdasdasd"  
输出：true

示例3：

输入：pattern = "aabb", str = "xyzabcxzyabc"  
输出：false

提示：

- 你可以默认 pattern 和 str 都只会包含小写字母。

通过次数 700 | 提交次数 1,373

在真实的面试中遇到过这道题？

贡献者

```
1 class Solution {  
2     public boolean wordPatternMatch(String pattern, String str) {  
3         Map<Character, String> map = new HashMap<>();  
4         return dfs(pattern, 0, str, 0, map);  
5     }  
6  
7     public boolean dfs(String pattern, int pIdx, String str, int sIdx, Map<Character, String> map) {  
8         int pLen = pattern.length(), sLen = str.length();  
9         if (pIdx == pLen && sIdx == sLen) {  
10             return true;  
11         }  
12         if (pIdx == pLen || sIdx == sLen) {  
13             return false;  
14         }  
15         char key = pattern.charAt(pIdx);  
16         if (map.containsKey(key)) {  
17             String val = map.get(key);  
18             if (sLen - sIdx < val.length()) {  
19                 return false;  
20             } else {  
21                 String sub = str.substring(sIdx, sIdx + val.length());  
22                 if (sub.equals(val)) {  
23                     return dfs(pattern, pIdx + 1, str, sIdx + val.length(), map);  
24                 } else {  
25                     return false;  
26                 }  
27             }  
28         } else {  
29             for (int i = 1; i <= sLen - sIdx - (pLen - pIdx - 1); i++) {  
30                 String sub = str.substring(sIdx, sIdx + i);  
31                 if (map.containsValue(sub)) {  
32                     continue;  
33                 }  
34                 map.put(key, sub);  
35                 if (dfs(pattern, pIdx + 1, str, sIdx + i, map)) {  
36                     return true;  
37                 }  
38                 map.remove(key);  
39             }  
40         }  
41     }  
42     return false;  
43 }
```

## 293. 翻转游戏

293. 翻转游戏

难度 简单 ⚡ 12 ❤ 1 🌟 2 ✎ 1

你和朋友玩一个叫做「翻转游戏」的游戏，游戏规则：给定一个只有 + 和 - 的字符串。你和朋友轮流将 连续 的两个 “++” 反转成 “--”。当一方无法进行有效的翻转时便意味着游戏结束，则另一方获胜。

请你写出一个函数，来计算出每个有效操作后，字符串所有的可能状态。

示例：

输入：s = "++++"  
输出：  
[  
 "--++",  
 "+--+",
 "++-+"
]

```
1 class Solution {  
2     public List<String> generatePossibleNextMoves(String s) {  
3         List<String> ans = new ArrayList<>();  
4         if (s == null || s.isEmpty()) {  
5             return ans;  
6         }  
7         char[] ches = s.toCharArray();  
8         for (int i = 1; i < ches.length; i++) {  
9             if (ches[i - 1] == '+' && ches[i] == '+') {  
10                 ches[i - 1] = '-';  
11                 ches[i] = '-';  
12                 ans.add(new String(ches));  
13                 ches[i - 1] = '+';  
14                 ches[i] = '+';  
15             }  
16         }  
17     }  
18     return ans;  
19 }
```

## 294. 翻转游戏 II

294. 翻转游戏 II

难度 中等 击 21 喜欢 6 收藏 0 提交

你和朋友玩一个叫做「翻转游戏」的游戏，游戏规则：给定一个只含有 + 和 - 的字符串。你和朋友轮流将连续的两个 “++” 反转成 “--”。当一方无法进行有效的翻转时便意味着游戏结束，则另一方获胜。

请你写出一个函数来判定起始玩家是否存在必胜的方案。

示例：

输入: s = "++++"  
输出: true  
解析: 起始玩家可将中间的 "++" 翻转变为 "+--+”从而得胜。

延伸：  
请推导你算法的时间复杂度。

通过次数 1.331 | 提交次数 2.325

在真实的面试中遇到过这道题？

是 否

```
1 v class Solution {  
2 v     public boolean canWin(String s) {  
3 v         Set<String> winners = new HashSet<>();  
4 v         Set<String> failers = new HashSet<>();  
5 v         return dfs(s, winners, failers);  
6 v     }  
7 v  
8 v     private boolean dfs(String s, Set<String> winners, Set<String> failers) {  
9 v         if (failers.contains(s)) {  
10 v             return false;  
11 v         }  
12 v         if (winners.contains(s)) {  
13 v             return true;  
14 v         }  
15 v         for (int i = 1; i < s.length(); i++) {  
16 v             if (s.charAt(i) == '+' && s.charAt(i - 1) == '+') {  
17 v                 String tmp = s.substring(0, i - 1) + "--" + s.substring(i + 1);  
18 v                 if (!dfs(tmp, winners, failers)) {  
19 v                     winners.add(s);  
20 v                     return true;  
21 v                 }  
22 v             }  
23 v         }  
24 v         failers.add(s);  
25 v     }  
26 v }  
27 }
```

## 298. 二叉树最长连续序列

298. 二叉树最长连续序列

难度 中等 击 13 喜欢 6 收藏 0 提交

给你一棵指定的二叉树，请你计算它最长连续序列路径的长度。

该路径，可以是从某个初始结点到树中任意结点，通过「父 - 子」关系连接而产生的任意路径。

这个最长连续的路径，必须从父结点到子结点，反过来是不可以的。

示例 1：

```
1 v class Solution {  
2 v     public int longestConsecutive(TreeNode root) {  
3 v         return dfs(root, null, 0);  
4 v     }  
5 v     private int dfs(TreeNode node, TreeNode prev, int len) {  
6 v         if (node == null) {  
7 v             return 0;  
8 v         }  
9 v         len = (prev != null && (prev.val + 1 == node.val)) ? len + 1 : 1;  
10 v        int lLen = dfs(node.left, node, len);  
11 v        int rLen = dfs(node.right, node, len);  
12 v        return Math.max(len, Math.max(lLen, rLen));  
13 v    }  
14 v }
```

## 302. 包含全部黑色像素的最小矩形

302. 包含全部黑色像素的最小矩形

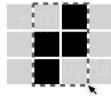
难度 困难 击 7 喜欢 6 收藏 0 提交

图片在计算机处理中往往是使用二维矩阵来表示的。

假设，这里我们用的是一张黑白的图片，那么 0 代表白色像素，1 代表黑色像素。

其中黑色的像素他们相互连接，也就是说，图片中只会有一片连在一快儿的黑色像素（像素点是水平或竖直方向连接的）。

那么，给出某一个黑色像素点 (x, y) 的位置，你是否可以找出包含全部黑色像素的最小矩形（与坐标轴对齐）的面积呢？



示例：

输入：  
[  
 "0010",  
 "0110",  
 "0100"  
]  
和 x = 0, y = 2  
输出: 6

```
1 v class Solution {  
2 v     public int minArea(char[][] image, int x, int y) {  
3 v         if (image == null) {  
4 v             return 0;  
5 v         }  
6 v         if (image.length == 0 || image[0].length == 0) {  
7 v             return 0;  
8 v         }  
9 v         int[] dirs = new int[4];  
10 v        dirs[0] = dirs[1] = x;  
11 v        dirs[2] = dirs[3] = y;  
12 v        dfs(image, x, y, dirs);  
13 v        return (dirs[3] - dirs[2]) * (dirs[1] - dirs[0]);  
14 v    }  
15 v  
16 v    private void dfs(char[][] image, int x, int y, int[] dirs) {  
17 v        if (x < 0 || x >= image.length || y < 0 || y >= image[0].length) {  
18 v            return;  
19 v        }  
20 v        if (image[x][y] == '0') {  
21 v            return;  
22 v        }  
23 v        image[x][y] = '0';  
24 v        dirs[0] = Math.min(dirs[0], x);  
25 v        dirs[1] = Math.max(dirs[1], x + 1);  
26 v        dirs[2] = Math.min(dirs[2], y);  
27 v        dirs[3] = Math.max(dirs[3], y + 1);  
28 v        dfs(image, x + 1, y, dirs);  
29 v        dfs(image, x - 1, y, dirs);  
30 v        dfs(image, x, y - 1, dirs);  
31 v        dfs(image, x, y + 1, dirs);  
32 v    }  
33 }
```

## 200. 岛屿数量

### 200. 岛屿数量

难度 中等 通过率 550 / 1000  
给你一个由 '1' (陆地) 和 '0' (水) 组成的二维网格, 请你计算网格中岛屿的数量。

岛屿总是被水包围, 并且每座岛屿只能由水平方向或竖直方向上相邻的陆地连接形成。

此外, 你可以假设该网格的四条边均被水包围。

#### 示例 1:

输入:

11110

11010

11000

00000

输出: 1

#### 示例 2:

输入:

11000

11000

00100

00011

输出: 3

解释: 每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。

```
1 class Solution {
2     private static final char NOT_VISITED = '1';
3     private static final char HAD_VISITED = '2';
4 }
5 public int numIslands(char[][] grid) {
6     int num = 0;
7     for (int rid = 0; rid < grid.length; rid++) {
8         for (int cid = 0; cid < grid[rid].length; cid++) {
9             if (grid[rid][cid] == NOT_VISITED) {
10                 num += 1;
11                 dfs(grid, rid, cid);
12             }
13         }
14     }
15     return num;
16 }
17
18 private void dfs(char[][] grid, int rid, int cid) {
19     grid[rid][cid] = HAD_VISITED;
20     if (rid - 1 >= 0 && grid[rid - 1][cid] == NOT_VISITED) {
21         dfs(grid, rid - 1, cid);
22     }
23     if (rid + 1 < grid.length && grid[rid + 1][cid] == NOT_VISITED) {
24         dfs(grid, rid + 1, cid);
25     }
26     if (cid - 1 >= 0 && grid[rid][cid - 1] == NOT_VISITED) {
27         dfs(grid, rid, cid - 1);
28     }
29     if (cid + 1 < grid[rid].length && grid[rid][cid + 1] == NOT_VISITED) {
30         dfs(grid, rid, cid + 1);
31     }
32 }
33 }
```

## 305. 岛屿数量 II

### 305. 岛屿数量 II

难度 困难 通过率 18 / 1000

假设你设计一个游戏, 用一个  $m$  行  $n$  列的 2D 网格来存储你的游戏地图。

起始的时候, 每个格子的地形都被默认标记为「水」。我们可以通过使用 `addLand` 进行操作, 将位置  $(row, col)$  的「水」变成「陆地」。

你将会被给定一个列表, 来记录所有需要被操作的位置, 然后你需要返回计算出来 每次 `addLand` 操作后岛屿的数量。

注意: 一个岛的定义是被「水」包围的「陆地」, 通过水平方向或者垂直方向上相邻的陆地连接而成。你可以假设地图网格的四边均被无边无际的「水」所包围。

请仔细阅读下方示例与解析, 更加深入了解岛屿的判定。

#### 示例:

输入:  $m = 3$ ,  $n = 3$ , positions = [[0,0], [0,1],  
[1,2], [2,1]]  
输出: [1,1,2,3]

#### 解析:

起初, 二维网格 `grid` 被全部注入「水」。`0` 代表「水」, `1` 代表「陆地」)

0 0 0  
0 0 0  
0 0 0

操作 #1: `addLand(0, 0)` 将 `grid[0][0]` 的水变为陆地。

1 0 0  
0 0 0 Number of islands = 1  
0 0 0

操作 #2: `addLand(0, 1)` 将 `grid[0][1]` 的水变为陆地。

```
1 class Solution {
2     public List<Integer> numIslands2(int m, int n, int[][] positions) {
3         List<Integer> ans = new ArrayList<>();
4         int count = 0;
5         int islandIdx = 0;
6         Map<Integer, Integer> map = new HashMap<>();
7         for (int[] pos : positions) {
8             int rid = pos[0], cid = pos[1];
9             if (map.containsKey(rid * n + cid)) {
10                 ans.add(count);
11                 continue;
12             }
13             Set<Integer> set = new HashSet<>();
14             if (rid - 1 >= 0 && map.containsKey((rid - 1) * n + cid)) {
15                 set.add(map.get((rid - 1) * n + cid));
16             }
17             if (rid + 1 < m && map.containsKey((rid + 1) * n + cid)) {
18                 set.add(map.get((rid + 1) * n + cid));
19             }
20             if (cid - 1 >= 0 && map.containsKey(rid * n + cid - 1)) {
21                 set.add(map.get(rid * n + cid - 1));
22             }
23             if (cid + 1 < n && map.containsKey(rid * n + cid + 1)) {
24                 set.add(map.get(rid * n + cid + 1));
25             }
26             if (set.isEmpty()) {
27                 ++count;
28                 map.put(rid * n + cid, islandIdx++);
29             } else if (set.size() == 1) {
30                 map.put(rid * n + cid, set.iterator().next());
31             } else {
32                 int next = set.iterator().next();
33                 for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
34                     int key = entry.getKey();
35                     int val = entry.getValue();
36                     if (set.contains(val)) {
37                         map.put(key, next);
38                     }
39                 }
40                 map.put(rid * n + cid, next);
41                 count -= (set.size() - 1);
42             }
43         }
44         ans.add(count);
45     }
46     return ans;
47 }
```

## 463. 岛屿的周长

### 463. 岛屿的周长

难度 简单 通过数 181 收藏 复制

给定一个包含 0 和 1 的二维网格地图，其中 1 表示陆地 0 表示水域。

网格中的格子水平和垂直方向相连（对角线方向不相连）。整个网格被水完全包围，但其中恰好有一个岛屿（或者说，一个或多个表示陆地的格子相连组成的岛屿）。

岛屿中没有“湖”（“湖”指水域在岛屿内部且不和岛屿周围的水相连）。格子是边长为 1 的正方形。网格为长方形，且宽度和高度均不超过 100。计算这个岛屿的周长。

示例：

输入：  
[[0,1,0,0],  
 [1,1,1,0],  
 [0,1,0,0],  
 [1,1,0,0]]

输出：16

```
1 class Solution {  
2     public int islandPerimeter(int[][] grid) {  
3         for (int r = 0; r < grid.length; r++) {  
4             for (int c = 0; c < grid[0].length; c++) {  
5                 if (grid[r][c] == 1) {  
6                     return dfs(grid, r, c);  
7                 }  
8             }  
9         }  
10     return 0;  
11 }  
12  
13 private int dfs(int[][] grid, int r, int c) {  
14     if (r < 0 || r >= grid.length || c < 0 || c >= grid[0].length) {  
15         return 1;  
16     }  
17     if (grid[r][c] == 0) {  
18         return 1;  
19     }  
20     if (grid[r][c] != 1) {  
21         return 0;  
22     }  
23     grid[r][c] = 2;  
24     return dfs(grid, r - 1, c) + dfs(grid, r + 1, c)  
25         + dfs(grid, r, c - 1) + dfs(grid, r, c + 1);  
26 }  
27 }
```

## 694. 不同岛屿的数量

### 694. 不同岛屿的数量

难度 中等 通过数 21 收藏 复制

给定一个非空二维数组表示的网格。一个岛屿由四连通（上、下、左、右四个方向）的 1 组成，你可以认为网格的四周被海水包围。

请你计算这个网格中共有多少个形状不同的岛屿。两个岛屿被认为是相同的，当且仅当一个岛屿可以通过平移变换（不可以旋转、翻转）和另一个岛屿重合。

样例 1：

11000  
11000  
00011  
00011

给定上图，返回结果 1。

样例 2：

11011  
10000  
00001  
11011

```
1 class Solution {  
2     public int numDistinctIslands(int[][] grid) {  
3         if (grid == null || grid.length == 0 || grid[0].length == 0) {  
4             return 0;  
5         }  
6         Set<List<Integer>> set = new HashSet<List<Integer>>();  
7         for (int rid = 0; rid < grid.length; rid++) {  
8             for (int cid = 0; cid < grid[rid].length; cid++) {  
9                 if (grid[rid][cid] == 1) {  
10                     List<Integer> list = new ArrayList<Integer>();  
11                     dfs(grid, list, rid, cid, 0);  
12                     if (!list.isEmpty()) {  
13                         set.add(list);  
14                     }  
15                 }  
16             }  
17         }  
18         return set.size();  
19     }  
20     private void dfs(int[][] grid, List<Integer> result, int rid, int cid, int x, int y) {  
21         if (rid < 0 || rid >= grid.length || cid < 0  
22             || cid >= grid[0].length || grid[rid][cid] == 0) {  
23             return;  
24         }  
25         final int[] dirs = new int[]{-1, 0, 1, 0, -1, 0, 0, 1};  
26         grid[rid][cid] = 0;  
27         result.add(rid - x);  
28         result.add(cid - y);  
29         for (int k = 0; k < dirs.length; k++) {  
30             dfs(grid, result, rid + dirs[k][0], cid + dirs[k][1], x, y);  
31         }  
32     }  
33 }
```

## 695. 岛屿的最大面积

### 695. 岛屿的最大面积

难度 中等 通过数 264 收藏 复制

给定一个包含了一些 0 和 1 的非空二维数组 grid。

一个岛屿是由一些相邻的 1（代表土地）构成的组合，这里的「相邻」要求两个 1 必须在水平或者竖直方向上相邻。你可以假设 grid 的四个边缘都被 0（代表水）包围着。

找到给定的二维数组中最大的岛屿面积。（如果没有岛屿，则返回面积为 0。）

示例 1：

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

对于上面这个给定矩阵应返回 6。注意答案不应该是 11，因为岛屿只能包含水平或垂直的四个方向的 1。

示例 2：

[[0,0,0,0,0,0,0,0]]

对于上面这个给定的矩阵，返回 0。

```
1 class Solution {  
2     private static final int ISLAND_NOT_VISITED = 1;  
3     private static final int ISLAND_HAS_VISITED = 2;  
4  
5     public int maxAreaOfIsland(int[][] grid) {  
6         int ans = 0;  
7         for (int rid = 0; rid < grid.length; rid++) {  
8             for (int cid = 0; cid < grid[rid].length; cid++) {  
9                 int area = area(grid, rid, cid);  
10                ans = Math.max(ans, area);  
11            }  
12        }  
13        return ans;  
14    }  
15  
16    private int area(int[][] grid, int rid, int cid) {  
17        if (!isValid(grid, rid, cid)) {  
18            return 0;  
19        }  
20        if (grid[rid][cid] != ISLAND_NOT_VISITED) {  
21            return 0;  
22        }  
23        grid[rid][cid] = ISLAND_HAS_VISITED;  
24        return area(grid, rid - 1, cid)  
25            + area(grid, rid + 1, cid)  
26            + area(grid, rid, cid - 1)  
27            + area(grid, rid, cid + 1)  
28            + 1;  
29    }  
30  
31    private boolean isValid(int[][] grid, int rid, int cid) {  
32        return (0 <= rid && rid < grid.length)  
33            && (0 <= cid && cid < grid[0].length);  
34    }  
35 }
```

## 711. 不同岛屿的数量 II

711. 不同岛屿的数量 II

难度 困难 ⌂ 13 ❤️ 🔍 🎁

给定一个非空01二维数组表示的网格，一个岛屿由四连通（上、下、左、右四个方向）的 1 组成，你可以认为网格的四周被海水包围。

请你计算这个网格中共有多少个形状不同的岛屿。如果两个岛屿的形状相同，或者通过旋转（顺时针旋转 90°, 180°, 270°），翻转（左右翻转、上下翻转）后形状相同，则就认为这两个岛屿是相同的。

样例 1：

```
11000  
10000  
00001  
00011
```

给定上图，返回结果 1。

注意：

```
11  
1
```

和

```
1  
11
```

是相同的岛屿。因为我们通过 180° 旋转第一个岛屿，两个岛屿的形状相同。

样例 2：

```
11100  
10001  
01001  
01110
```

```
1 * class Solution {  
2 *     public int numDistinctIslands2(int[][] grid) {  
3 *         if (grid == null || grid.length == 0 || grid[0].length == 0)  
4 *             return 0;  
5 *         Set<String> set = new HashSet<>();  
6 *         int rows = grid.length, cols = grid[0].length;  
7 *         boolean[][] visited = new boolean[rows][cols];  
8 *         for (int rid = 0; rid < rows; rid++)  
9 *             for (int cid = 0; cid < cols; cid++)  
10 *                 if (grid[rid][cid] == 1 && !visited[rid][cid]) {  
11 *                     List<Integer> list = new ArrayList<>();  
12 *                     dfs(grid, rows, cols, rid, cid, list, visited);  
13 *                     if (!list.isEmpty())  
14 *                         set.add(serializer(grid, rows, cols, list));  
15 *                 }  
16 *         return set.size();  
17 *     }  
18 *     private String serializer(int[][] grid, int rows, int cols, List<Integer> list) {  
19 *         String res = null;  
20 *         int size = list.size();  
21 *         int[] rids = new int[size], cids = new int[size], outs = new int[size];  
22 *         int[] dirs = {{1, 1}, {-1, 1}, {1, -1}, {-1, -1}};  
23 *         for (int i = 0; i < size; i++) {  
24 *             for (int k = 0; k < size; k++) {  
25 *                 int v = list.get(k), x = v / cols, y = v % cols;  
26 *                 rids[k] = (x < 4 ? x : y) * dirs[i % 4][0];  
27 *                 cids[k] = (x < 4 ? y : x) * dirs[i % 4][1];  
28 *             }  
29 *             int rid = rids[0], cid = cids[0];  
30 *             for (int k = 1; k < size; k++) {  
31 *                 rid = Math.min(rid, rids[k]); cid = Math.min(cid, cids[k]);  
32 *             }  
33 *             outs[k] = (rids[k] - rid) * cols + (cids[k] - cid);  
34 *             Arrays.sort(outs); String tmp = Arrays.toString(outs);  
35 *             if (res == null || res.compareTo(tmp) < 0)  
36 *                 res = tmp;  
37 *         }  
38 *         return res;  
39 *     }  
40 *     private void dfs(int[][] grid, int rows, int cols, int rid, int cid, List<Integer> list, boolean[][] visited) {  
41 *         if (rid < 0 || rid >= rows || cid < 0 || cid >= cols || grid[rid][cid] == 0 || visited[rid][cid])  
42 *             return;  
43 *         visited[rid][cid] = true; list.add(rid * cols + cid);  
44 *         dfs(grid, rows, cols, rid - 1, cid, list, visited);  
45 *         dfs(grid, rows, cols, rid + 1, cid, list, visited);  
46 *         dfs(grid, rows, cols, rid, cid + 1, list, visited);  
47 *         dfs(grid, rows, cols, rid, cid - 1, list, visited);  
48 *     }  
49 * }
```

## 827. 最大人工岛

827. 最大人工岛

难度 困难 ⌂ 37 ❤️ 🔍 🎁

在二维地图上，0 代表海洋，1 代表陆地，我们最多只能将一格 0 海洋变成 1 变成陆地。

进行填海之后，地图上最大的岛屿面积是多少？（上、下、左、右四个方向相连的 1 可形成岛屿）

示例 1：

```
输入: [[1, 0], [0, 1]]  
输出: 3  
解释: 将一格0变成1，最终连通两个小岛得到面积为 3 的岛屿。
```

示例 2：

```
输入: [[1, 1], [1, 0]]  
输出: 4  
解释: 将一格0变成1，岛屿的面积扩大为 4。
```

示例 3：

```
输入: [[1, 1], [1, 1]]  
输出: 4  
解释: 没有0可以让我们变成1，面积依然为 4。
```

说明：

- $1 \leq \text{grid.length} = \text{grid[0].length} \leq 50$
- $0 \leq \text{grid}[i][j] \leq 1$

通过次数 1,770 | 提交次数 4,472

在真实的面试中遇到过这道题？

是 否

贡献者

相关企业 i

```
1 * class Solution {  
2 *     public int largestIsland(int[][] grid) {  
3 *         int rows = grid.length, cols = grid[0].length;  
4 *         int idx = 2;  
5 *         int[] area = new int[rows * cols + 2];  
6 *         for (int rid = 0; rid < rows; ++rid)  
7 *             for (int cid = 0; cid < cols; ++cid)  
8 *                 if (grid[rid][cid] == 1)  
9 *                     area[idx] = dfs(grid, rows, cols, rid, cid, idx++);  
10 *            int ans = 0;  
11 *            for (int x : area)  
12 *                ans = Math.max(ans, x);  
13 *            for (int rid = 0; rid < rows; ++rid)  
14 *                for (int cid = 0; cid < cols; ++cid)  
15 *                    if (grid[rid][cid] == 0) {  
16 *                        Set<Integer> seen = new HashSet<>();  
17 *                        for (Integer move : neighbors(rows, cols, rid, cid))  
18 *                            if (grid[move / rows][move % rows] > 1)  
19 *                                seen.add(grid[move / rows][move % rows]);  
20 *                        int bns = 1;  
21 *                        for (int i : seen)  
22 *                            bns += area[i];  
23 *                        ans = Math.max(ans, bns);  
24 *                    }  
25 *            return ans;  
26 *        }  
27 *        public int dfs(int[][] grid, int rows, int cols, int rid, int cid, int idx) {  
28 *            int ans = 1;  
29 *            grid[rid][cid] = idx;  
30 *            for (Integer move : neighbors(rows, cols, rid, cid))  
31 *                if (grid[move / rows][move % rows] == 1) {  
32 *                    grid[move / rows][move % rows] = idx;  
33 *                    ans += dfs(grid, rows, cols, move / rows, move % cols, idx);  
34 *                }  
35 *            return ans;  
36 *        }  
37 *        public List<Integer> neighbors(int rows, int cols, int rid, int cid) {  
38 *            final int[] rDirs = new int[]{-1, 0, 1, 0};  
39 *            final int[] cDirs = new int[]{0, -1, 0, 1};  
40 *            List<Integer> ans = new ArrayList<>();  
41 *            for (int k = 0; k < 4; ++k) {  
42 *                int nRid = rid + rDirs[k];  
43 *                int nCid = cid + cDirs[k];  
44 *                if (nRid >= 0 && nRid < rows && nCid >= 0 && nCid < cols)  
45 *                    ans.add(nRid * rows + nCid);  
46 *            }  
47 *            return ans;  
48 *        }  
49 *    }
```

## 1254. 统计封闭岛屿的数目

1254. 统计封闭岛屿的数目

难度 中等 通过率 23% 热度 4.4K 提交 4.4K

有一个二维矩阵 `grid`，每个位置要么是陆地（记号为 `0`）要么是水域（记号为 `1`）。

我们从一块陆地出发，每次可以往上下左右 4 个方向相邻区域走，能走到的所有陆地区域，我们将称其为一座「岛屿」。

如果一座岛屿 完全 由水域包围，即陆地边缘上下左右所有相邻区域都是水域，那么我们将其称为「封闭岛屿」。

请返回封闭岛屿的数目。

示例 1:

A 9x9 grid representing land (0) and water (1). A 3x3 subgrid in the center is highlighted in light blue, representing a closed island. The rest of the grid contains 0s (water).

1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	0
1	0	1	0	1	1	1	0
1	0	0	0	0	1	0	1
1	1	1	1	1	1	1	0

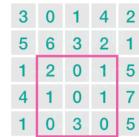
```
1 class Solution {
2     public int closedIsland(int[][] grid) {
3         if (grid == null || grid.length == 0 || grid[0].length == 0) {
4             return 0;
5         }
6         int res = 0;
7         for (int rid = 0; rid < grid.length; rid++) {
8             for (int cid = 0; cid < grid[0].length; cid++) {
9                 if (grid[rid][cid] == 0) {
10                     res += dfs(grid, rid, cid);
11                 }
12             }
13         }
14         return res;
15     }
16
17     private int dfs(int[][] grid, int rid, int cid) {
18         if (rid < 0 || rid >= grid.length || cid < 0 || cid >= grid[0].length) {
19             return 0;
20         }
21         int res = 1;
22         final int[] dirs = new int[]{1, 0}, {0, 1}, {-1, 0}, {0, -1};
23         if (grid[rid][cid] == 0) {
24             grid[rid][cid] = 1;
25             for (int[] dir : dirs) {
26                 res = Math.min(res, dfs(grid, rid + dir[0], cid + dir[1]));
27             }
28         }
29     }
30     return res;
31 }
```

## 308. 二维区域和检索 - 可变

308. 二维区域和检索 - 可变

难度 困难 通过率 13% 热度 4.4K 提交 4.4K

给你一个 2D 矩阵 `matrix`，请计算出从左上角 (`row1, col1`) 到右下角 (`row2, col2`) 组成的矩形中所有元素的和。



A 5x5 matrix with elements ranging from 1 to 5. A submatrix from (2, 1) to (4, 3) is highlighted with a pink border.

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

上述粉色矩形框内的，该矩形由左上角 (`row1, col1`) = (2, 1) 和右下角 (`row2, col2`) = (4, 3) 确定。其中，所包括的元素总和 `sum = 8`。

示例：

```
给定 matrix = [
 [3, 0, 1, 4, 2],
 [5, 6, 3, 2, 1],
 [1, 2, 0, 1, 5],
 [4, 1, 0, 1, 7],
 [1, 0, 3, 0, 5]
]

sumRegion(2, 1, 4, 3) -> 8
update(3, 2, 2)
sumRegion(2, 1, 4, 3) -> 10
```

注意:

1. 矩阵 `matrix` 的值只能通过 `update` 函数来进行修改
2. 你可以默认 `update` 函数和 `sumRegion` 函数的调用次数是均匀分布的
3. 你可以默认 `row1 ≤ row2, col1 ≤ col2`

通过次数 756 | 提交次数 1,308

```
1 class NumMatrix {
2     private Bit mBit;
3     private int[][] mMMatrix;
4     public NumMatrix(int[][] matrix) {
5         if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
6             return;
7         int rows = matrix.length, cols = matrix[0].length;
8         this.mMatrix = matrix; this.mBit = new Bit(rows, cols);
9         for (int rid = 0; rid < rows; rid++)
10             for (int cid = 0; cid < cols; cid++)
11                 mBit.update(rid, cid, matrix[rid][cid]);
12     }
13     public void update(int row, int col, int val) {
14         mBit.update(row, col, val - mMMatrix[row][col]);
15         mMMatrix[row][col] = val;
16     }
17     public int sumRegion(int row1, int col1, int row2, int col2) {
18         return mBit.query(row1, col1, row2, col2);
19     }
20     class Bit {
21         private int mRow, mCol; private int[][] mSums;
22         public Bit(int m, int n) {
23             this.mRow = m; this.mCol = n;
24             this.mSums = new int[m + 1][n + 1];
25         }
26         public void update(int row, int col, int delta) {
27             if (mRow == 0 || mCol == 0)
28                 return;
29             for (int i = row + 1; i <= mRow; i += lowBit(i))
30                 for (int j = col + 1; j <= mCol; j += lowBit(j))
31                     mSums[i][j] += delta;
32         }
33         public int query(int row1, int col1, int row2, int col2) {
34             if (mRow == 0 || mCol == 0)
35                 return 0;
36             return sum(row1, col1) - sum(row1, col2 + 1) + sum(row2 + 1, col1) - sum(row2 + 1, col2 + 1);
37         }
38         public int sum(int row, int col) {
39             int ans = 0;
40             for (int rid = row; rid > 0; rid -= lowBit(rid))
41                 for (int cid = col; cid > 0; cid -= lowBit(cid))
42                     ans += mSums[rid][cid];
43             return ans;
44         }
45         private int lowBit(int x) {
46             return x & (-x);
47         }
48     }
49 }
```

### 311. 稀疏矩阵的乘法

311. 稀疏矩阵的乘法

难度 中等 收藏 13 提交 1A 单元测试

给你两个 稀疏矩阵 A 和 B，请你返回 AB 的结果。你可以默认 A 的列数等于 B 的行数。

请仔细阅读下面的示例。

示例：

输入：

```
A = [
    [ 1, 0, 0 ],
    [-1, 0, 3 ]
]
```

```
B = [
    [ 7, 0, 0 ],
    [ 0, 0, 0 ],
    [ 0, 0, 1 ]
]
```

输出：

```
AB = | 1 0 0 | x | 7 0 0 | = | -7 0 3 |
      | -1 0 3 |     | 0 0 0 |           | 0 0 1 |
```

通过次数 981 | 提交次数 1,339

在真实的面试中遇到过这道题？

贡献者

```
1 class Solution {
2     public int[][] multiply(int[][] A, int[][] B) {
3         int m = A.length;
4         int n = B[0].length;
5         int[] grid = new int[m][n];
6         Map<Integer, Map<Integer, Integer>> mapA = transA(A);
7         Map<Integer, Map<Integer, Integer>> mapB = transB(B);
8         for (Map.Entry<Integer, Map<Integer, Integer>> entry1 : mapA.entrySet()) {
9             Map<Integer, Integer> map1 = entry1.getValue();
10            Set<Integer> keys1 = map1.keySet();
11            int key1 = entry1.getKey();
12            for (Map.Entry<Integer, Map<Integer, Integer>> entry2 : mapB.entrySet()) {
13                Map<Integer, Integer> map2 = entry2.getValue();
14                Set<Integer> keys2 = new HashSet<>(map2.keySet());
15                int key2 = entry2.getKey();
16                keys2.retainAll(keys1);
17                for (Integer idx : keys2) {
18                    grid[key1][key2] += map1.get(idx) * map2.get(idx);
19                }
20            }
21        }
22        return grid;
23    }
24    public Map<Integer, Map<Integer, Integer>> transA(int[][] A) {
25        Map<Integer, Map<Integer, Integer>> mA = new HashMap<>();
26        for (int rid = 0; rid < A.length; rid++) {
27            Map<Integer, Integer> rowA = new HashMap<>();
28            for (int cid = 0; cid < A[0].length; cid++) {
29                if (A[rid][cid] != 0)
30                    rowA.put(cid, A[rid][cid]);
31                if (rowA.size() > 0) mA.put(rid, rowA);
32            }
33        }
34        return mA;
35    }
36    public Map<Integer, Map<Integer, Integer>> transB(int[][] B) {
37        Map<Integer, Map<Integer, Integer>> mB = new HashMap<>();
38        for (int rid = 0; rid < B[0].length; rid++) {
39            Map<Integer, Integer> colB = new HashMap<>();
40            for (int cid = 0; cid < B.length; cid++) {
41                if (B[cid][rid] != 0)
42                    colB.put(cid, B[cid][rid]);
43                if (colB.size() > 0) mB.put(rid, colB);
44            }
45        }
46        return mB;
}
```

### 314. 二叉树的垂直遍历

314. 二叉树的垂直遍历

难度 中等 收藏 22 提交 1A 单元测试

给定一个二叉树，返回其结点 垂直方向（从上到下，逐列）遍历的值。

如果两个结点在同一行和列，那么顺序则为 从左到右。

示例 1：

输入： [3,9,20,null,null,15,7]

```
3
/\ 
/ \
9   20
  /\
 / \
15   7
```

输出：

```
[
  [9],
  [3,15],
  [20],
  [7]
]
```

示例 2：

输入： [3,9,8,4,0,1,7]

```
3
/\ 
/ \
9   8
/\   /\
/ \ / \
4   0 1   7
```

```
1 class Solution {
2     public List<List<Integer>> verticalOrder(TreeNode root) {
3         List<List<Integer>> ans = new ArrayList<>();
4         if (root == null)
5             return ans;
6         Map<Integer, List<Integer>> map = new HashMap<>();
7         Queue<Node> queue = new LinkedList<>();
8         queue.offer(new Node(root, 0));
9         while (!queue.isEmpty()) {
10            int size = queue.size();
11            for (int i = 0; i < size; i++) {
12                Node node = queue.poll();
13                if (!map.containsKey(node.col)) {
14                    List<Integer> list = new ArrayList<>();
15                    list.add(node.root.val);
16                    map.put(node.col, list);
17                } else {
18                    List<Integer> list = map.get(node.col);
19                    list.add(node.root.val);
20                }
21                if (node.root.left != null)
22                    queue.offer(new Node(node.root.left, node.col - 1));
23                if (node.root.right != null)
24                    queue.offer(new Node(node.root.right, node.col + 1));
25            }
26            int[] keys = new int[map.size()];
27            int idx = 0;
28            for (int key : map.keySet())
29                keys[idx++] = key;
30            Arrays.sort(keys);
31            for (int i = 0; i < keys.length; i++)
32                ans.add(map.get(keys[i]));
33        }
34        return ans;
35    }
36
37    class Node {
38        int col;
39        TreeNode root;
40
41        Node(TreeNode root, int col) {
42            this.col = col;
43            this.root = root;
44        }
45    }
46 }
```

## 320. 列举单词的全部缩写

320. 列举单词的全部缩写

难度 中等 ⌂ 14 ❤ 14 🎁 14 🏆 14

请你写出一个能够举单词全部缩写的函数。

注意：输出的顺序并不重要。

示例：

输入： "word"  
输出：  
["word", "l0rd", "w1rd", "wo1d", "w0r1", "2rd",  
"w2d", "wo2", "10ld", "1or1", "w1r1", "1o2", "2r1",  
"3d", "w3", "4"]

```
1 class Solution {
2     public List<String> generateAbbreviations(String word) {
3         List<String> ans = new ArrayList<>();
4         helper(word, 0, ans);
5         ans.add(word);
6         return ans;
7     }
8     private void helper(String word, int idx, List<String> list) {
9         for (int i = idx; i < word.length(); i++) {
10            for (int x = 1; x <= word.length() - i; x++) {
11                String sub = word.substring(0, i) + x + word.substring(i + x);
12                list.add(sub);
13                String srt = "" + x;
14                helper(sub, i + srt.length() + 1, list);
15            }
16        }
17    }
18 }
```

## 323. 无向图中连通分量的数目

323. 无向图中连通分量的数目

难度 中等 ⌂ 23 ❤ 14 🎁 14 🏆 14

给定编号从 0 到 n-1 的 n 个节点和一个无向边列表（每条边都是一对节点），请编写一个函数来计算无向图中连通分量的数目。

示例 1：

输入： n = 5 和 edges = [[0, 1], [1, 2], [3, 4]]  
  
0        3  
|        |  
1 --- 2    4  
  
输出： 2

```
1 class Solution {
2     public int countComponents(int n, int[][] edges) {
3         int ans = 0;
4         boolean[] visited = new boolean[n];
5         Map<Integer, List<Integer>> graph = new HashMap<>;
6         for (int i = 0; i < n; i++) {
7             graph.put(i, new ArrayList<>());
8         }
9         for (int[] edge: edges) {
10            graph.get(edge[0]).add(edge[1]);
11            graph.get(edge[1]).add(edge[0]);
12        }
13        for (int i = 0; i < n; i++) {
14            if (visited[i]) {
15                continue;
16            }
17            dfs(i, graph, visited);
18            ans += 1;
19        }
20        return ans;
21    }
22
23    private void dfs(int vertex, Map<Integer, List<Integer>> graph, boolean[] visited) {
24        List<Integer> vertexes = graph.get(vertex);
25        visited[vertex] = true;
26        for (int v: vertexes) {
27            if (visited[v]) {
28                continue;
29            }
30            dfs(v, graph, visited);
31        }
32    }
33 }
```

## 547. 朋友圈

547. 朋友圈

难度 中等 ⌂ 228 ❤ 14 🎁 14 🏆 14

班上有 N 名学生。其中有些人是朋友，有些则不是。他们的友谊具有是传递性。如果已知 A 是 B 的朋友，B 是 C 的朋友，那么我们可以认为 A 也是 C 的朋友。所谓的朋友圈，是指所有朋友的集合。

给定一个  $N \times N$  的矩阵 M，表示班级中学生之间的朋友关系。如果  $M[i][j] = 1$ ，表示已知第 i 个和 j 个学生互为朋友关系，否则为不知道。你必须输出所有学生中的已知的朋友圈总数。

示例 1：

输入：  
[[1,1,0],  
 [1,1,0],  
 [0,0,1]]  
输出： 2  
说明：已知学生0和学生1互为朋友，他们在同一个朋友圈。  
第2个学生自己在一个朋友圈。所以返回2。

```
1 class Solution { // 转化为求连通分量
2     public int findCircleNum(int[][] M) {
3         if (M == null || M.length == 0 || M[0].length == 0)
4             return 0;
5         int count = 0;
6         boolean[] visited = new boolean[M.length];
7         for (int vtx = 0; vtx < M.length; vtx++) {
8             if (visited[vtx])
9                 continue;
10            dfs(M, visited, vtx);
11            count++;
12        }
13        return count;
14    }
15    public void dfs(int[][] grid, boolean[] visited, int vtx) {
16        for (int idx = 0; idx < grid.length; idx++) {
17            if (grid[vtx][idx] == 0 || visited[idx])
18                continue;
19            visited[idx] = true;
20            dfs(grid, visited, idx);
21        }
22    }
23 }
```

## 325. 和等于 k 的最长子数组长度

### 325. 和等于 k 的最长子数组长度

难度 中等 ⚡ 31 ❤️ ⚡ 🌟 💡

给定一个数组  $nums$  和一个目标值  $k$ , 找到和等于  $k$  的最长子数组长度。如果不存在任意一个符合要求的子数组, 则返回 0。

注意:

$nums$  数组的总和是一定在 32 位有符号整数范围之内的。

示例 1:

输入:  $nums = [1, -1, 5, -2, 3]$ ,  $k = 3$   
输出: 4  
解释: 子数组  $[1, -1, 5, -2]$  和等于 3, 且长度最长。

示例 2:

输入:  $nums = [-2, -1, 2, 1]$ ,  $k = 1$   
输出: 2  
解释: 子数组  $[-2, 1]$  和等于 1, 且长度最长。

## 333. 最大 BST 子树

### 333. 最大 BST 子树

难度 中等 ⚡ 27 ❤️ ⚡ 🌟 💡

给定一个二叉树, 找到其中最大的二叉搜索树 (BST) 子树, 其中最大指的是子树节点数最多的。

注意:

子树必须包含其所有后代。

示例:

输入:  $[10, 5, 15, 1, 8, \text{null}, 7]$   
  
10  
  /\  
  5  15  
  /\  \  
  1  8  7  
  
输出: 3  
解释: 高亮部分为最大的 BST 子树。  
返回值 3 在这个样例中为子树大小。

进阶:

你能想出用  $O(n)$  的时间复杂度解决这个问题吗?

## 339. 嵌套列表权重和

### 339. 嵌套列表权重和

难度 简单 ⚡ 14 ❤️ ⚡ 🌟 💡

给定一个嵌套的整数列表, 请返回该列表按深度加权后所有整数的总和。

每个元素要么是整数, 要么是列表。同时, 列表中元素同样也可以是整数或者是另一个列表。

示例 1:

输入:  $[[1, 1], 2, [1, 1]]$   
输出: 10  
解释: 因为列表中有四个深度为 2 的 1, 和一个深度为 1 的 2。

## 340. 至多包含 K 个不同字符的最长子串

### 340. 至多包含 K 个不同字符的最长子串

难度 困难 ⚡ 27 ❤️ ⚡ 🌟 💡

给定一个字符串  $s$ , 找出至多包含  $k$  个不同字符的最长子串  $T$ 。

示例 1:

输入:  $s = "eceba"$ ,  $k = 2$   
输出: 3  
解释: 则  $T$  为 "e", 所以长度为 3。

示例 2:

输入:  $s = "aa"$ ,  $k = 1$   
输出: 2  
解释: 则  $T$  为 "aa", 所以长度为 2。

```
1 class Solution {  
2     public int maxSubArrayLen(int[] nums, int k) {  
3         if (nums == null || nums.length == 0) {  
4             return 0;  
5         }  
6         int res = 0;  
7         int sum = 0;  
8         Map<Integer, Integer> map = new HashMap<>();  
9         for (int idx = 0; idx < nums.length; idx++) {  
10             sum += nums[idx];  
11             if (sum == k) {  
12                 res = idx + 1;  
13             } else if (map.containsKey(sum - k)) {  
14                 res = Math.max(res, idx - map.get(sum - k));  
15             }  
16             if (map.containsKey(sum)) {  
17                 continue;  
18             }  
19             map.put(sum, idx);  
20         }  
21         return res;  
22     }  
23 }
```

```
1 class Solution {  
2     public int largestBSTSubtree(TreeNode root) {  
3         if (root == null)  
4             return 0;  
5         int max = binarySearchTreeNodeCounter(root);  
6         int lMax = largestBSTSubtree(root.left);  
7         int rMax = largestBSTSubtree(root.right);  
8         return Math.max(max, Math.max(lMax, rMax));  
9     }  
10    private int binarySearchTreeNodeCounter(TreeNode node) {  
11        if (node == null)  
12            return 0;  
13        List<Integer> list = new ArrayList<>();  
14        inorder(node, list);  
15        int size = list.size();  
16        for (int i = 1; i < size; i++)  
17            if (list.get(i - 1) >= list.get(i)) {  
18                return -1;  
19            }  
20        return size;  
21    }  
22    private void inorder(TreeNode node, List<Integer> list) {  
23        if (node == null)  
24            return;  
25        inorder(node.left, list);  
26        list.add(node.val);  
27        inorder(node.right, list);  
28    }  
29 }
```

```
1 class Solution { // 递归  
2     public int depthSum(List<NestedInteger> nestedList) {  
3         return depthSum(nestedList, 1);  
4     }  
5     public int depthSum(List<NestedInteger> list, int depth) {  
6         int sum = 0;  
7         for (NestedInteger ni : list) {  
8             if (ni.isInteger()) {  
9                 sum += ni.getInteger() * depth;  
10            } else {  
11                sum += depthSum(ni.getList(), depth + 1);  
12            }  
13        }  
14        return sum;  
15    }  
16 }
```

```
1 class Solution {  
2     public int lengthOfLongestSubstringKDistinct(String s, int k) {  
3         if (s == null || s.isEmpty() || k == 0) {  
4             return 0;  
5         }  
6         int max = 0, lid = 0, rid = 0, len = s.length();  
7         HashMap<Character, Integer> map = new HashMap<>();  
8         while (rid < len) {  
9             map.put(s.charAt(rid), rid++);  
10            if (map.size() > k) {  
11                // 从map中删除下标最小的键值对  
12                int idx = Collections.min(map.values());  
13                map.remove(s.charAt(idx));  
14                lid = idx + 1;  
15            }  
16            max = Math.max(max, rid - lid);  
17        }  
18        return max;  
19    }  
20 }
```

## 346. 数据流中的移动平均值

### 346. 数据流中的移动平均值

难度 简单 击 15 喜欢 1 文章 举报

给定一个整数数据流和一个窗口大小，根据该滑动窗口的大小，计算其所有整数的移动平均值。

示例：

```
MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3
```

通过次数 4,192 | 提交次数 6,035

```
1 class MovingAverage {
2     private int mSize;
3     private int mIndex;
4     private List<Integer> mList;
5     public MovingAverage(int size) {
6         this.mIndex = 0;
7         this.mSize = size;
8         this.mList = new ArrayList<>(size);
9     }
10    public double next(int val) {
11        int idx = ++mIndex;
12        if (mIndex > mSize)
13            idx = mSize;
14        if (mList.size() == mSize)
15            mList.remove(0);
16        mList.add(val);
17        double sum = 0.0;
18        for (int i : mList)
19            sum += i;
20        return sum / idx;
21    }
22 }
```

## 348. 判定井字棋胜负

### 348. 判定井字棋胜负

难度 中等 击 19 喜欢 1 文章 举报

请在  $n \times n$  的棋盘上，实现一个判定井字棋（Tic-Tac-Toe）胜负的神器，判断每一次玩家落子后，是否有胜出的玩家。

在这个井字棋游戏中，会有 2 名玩家，他们将轮流在棋盘上放置自己的棋子。

在实现这个判定器的过程中，你可以假设以下这些规则一定成立：

1. 每一步棋都是在棋盘内的，并且只能被放置在一个空的格子里；
2. 一旦游戏中有一名玩家胜出的话，游戏将不能再继续；
3. 一个玩家如果在同一行、同一列或者同一斜对角线上都放置了自己的棋子，那么他便获得胜利。

示例：

给定棋盘边长  $n = 3$ ，玩家 1 的棋子符号是 "X"，玩家 2 的棋子符号是 "O"。

```
TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> 函数返回 0 (此时, 暂时没有玩家赢得这场对决)
|X| |
| | | // 玩家 1 在 (0, 0) 落子。
| | |

toe.move(0, 2, 2); -> 函数返回 0 (暂时没有玩家赢得本场比赛)
|X| |0|
| | | // 玩家 2 在 (0, 2) 落子。
| | |
```

```
1 class TicTacToe {
2     private final int[] mGrid;
3     public TicTacToe(int n) {
4         mGrid = new int[n][n];
5     }
6     public int move(int row, int col, int player) {
7         mGrid[row][col] = player;
8         int num = mGrid.length;
9         for (int idx = 0; idx < num; idx++) {
10             if (mGrid[idx][col] != player)
11                 break;
12             if (idx == num - 1) {
13                 return player;
14             }
15         }
16         for (int idx = 0; idx < num; idx++) {
17             if (mGrid[row][idx] != player)
18                 break;
19             if (idx == num - 1) {
20                 return player;
21             }
22         }
23         if (row == col)
24             for (int idx = 0; idx < num; idx++) {
25                 if (mGrid[idx][idx] != player)
26                     break;
27                 if (idx == num - 1)
28                     return player;
29             }
30         if (row + col == num - 1)
31             for (int idx = 0; idx < num; idx++) {
32                 if (mGrid[idx][num - idx - 1] != player)
33                     break;
34                 if (idx == num - 1)
35                     return player;
36             }
37         }
38     }
39     return 0;
40 }
```

## 351. 安卓系统手势解锁

351. 安卓系统手势解锁

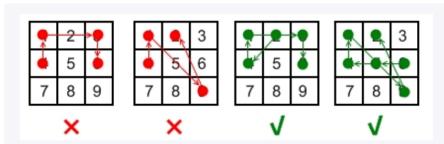
难度 中等 通过 28 收藏 10 举报

我们都知道安卓有个手势解锁的界面，是一个  $3 \times 3$  的点所绘制出来的网格。

给你两个整数，分别为  $m$  和  $n$ ，其中  $1 \leq m \leq n \leq 9$ ，那么请你统计一下有多少种解锁手势，是至少需要经过  $m$  个点，但是最多经过不超过  $n$  个点的。

先来了解下什么是一个有效的安卓解锁手势：

1. 每一个解锁手势必须至少经过  $m$  个点、最多经过  $n$  个点。
2. 解锁手势里不能设置经过重复的点。
3. 假如手势中有两个点是顺序经过的，那么这两个点的手势轨迹之间是绝对不能跨过任何未被经过的点。
4. 经过点的顺序不同则表示为不同的解锁手势。



```
1 v class Solution {  
2 v     public int numberOfPatterns(int m, int n) {  
3 v         int[] ans = new int[1];  
4 v         for (int idx = 0; idx < 9; ++idx) {  
5 v             for (int i = m; i <= n; ++i) {  
6 v                 boolean[] visited = new boolean[9];  
7 v                 dfs(idx, -1, i, visited, ans);  
8 v             }  
9 v         }  
10 v     return ans[0];  
11 v }  
12 v private void dfs(int idx, int pre, int step, boolean[] visited, int[] ans) {  
13 v     if (--step == 0) {  
14 v         ++ans[0];  
15 v     } else {  
16 v         visited[idx] = true;  
17 v         for (int i = 0; i < 9; ++i)  
18 v             if (isValid(i, idx, visited))  
19 v                 dfs(i, idx, step, visited, ans);  
20 v             visited[idx] = false;  
21 v     }  
22 v }  
23 v private boolean isValid(int idx, int pre, boolean[] visited) {  
24 v     if (visited[idx]) {  
25 v         return false;  
26 v     }  
27 v     if (idx + pre != 8 && (idx + pre) % 2 == 1  
28 v         || (idx % 3 != pre % 3 && idx / 3 != pre / 3))  
29 v         return true;  
30 v     return visited[(idx + pre) / 2];  
31 v }  
32 v }
```

## 353. 贪吃蛇

353. 贪吃蛇

难度 中等 通过 14 收藏 10 举报

请你设计一个贪吃蛇游戏。该游戏将会在一个屏幕尺寸 = 宽度  $\times$  高度 的屏幕上运行。如果你不熟悉这个游戏，可以[点击这里](#)在线试玩。

起初时，蛇在左上角的  $(0, 0)$  位置，身体长度为 1 个单位。

你将被给出一个  $(行, 列)$  形式的食物位置序列。当蛇吃到食物时，身子的长度会增加 1 个单位，得分也会  $+1$ 。

食物不会同时出现，会按列表的顺序逐一显示在屏幕上。比方讲，第一个食物被蛇吃掉后，第二个食物才会出现。

当一个食物在屏幕上出现时，它被保证不能出现在被蛇身体占据的格子里。

对于每个 `move()` 操作，你需要返回当前得分或 -1（表示蛇与自己身体或墙相撞，意味游戏结束）。

示例：

给定  $width = 3$ ,  $height = 2$ , 食物序列为  $food = [[1,2], [0,1]]$ 。

`Snake snake = new Snake(width, height, food);`

初始时，蛇的位置在  $(0, 0)$  且第一个食物在  $(1, 2)$ 。

|S| | |  
| | |F|

`snake.move("R");` -> 函数返回 0

| |S| |  
| | |F|

`snake.move("D");` -> 函数返回 0

| | | |  
| |S|F|

`snake.move("R");` -> 函数返回 1 (蛇吃掉了第一个食物，同时第二个食物出现在位置  $(0,1)$ )

```
1 v class SnakeGame {  
2 v     private int mWidth, mHeight, mFoodId, mScore;  
3 v     private final int[] mFood;  
4 v     private final Set<Integer> mSeens;  
5 v     private final Deque<Integer> mSnack;  
6 v     public SnakeGame(int width, int height, int[][] food) {  
7 v         this.mWidth = width;  
8 v         this.mHeight = height;  
9 v         this.mFood = food;  
10 v        this.mFoodId = 0;  
11 v        this.mScore = 0;  
12 v        this.mSeens = new HashSet<>();  
13 v        this.mSnack = new ArrayDeque<>();  
14 v        mSeens.add(0);  
15 v        mSnack.addLast(0);  
16 v    }  
17 v    public int move(String direction) {  
18 v        Integer head = mSnack.peekLast();  
19 v        int row = head / mWidth, col = head % mWidth;  
20 v        if (direction.equals("U")) {  
21 v            row--;  
22 v        } else if (direction.equals("L")) {  
23 v            col--;  
24 v        } else if (direction.equals("R")) {  
25 v            col++;  
26 v        } else {  
27 v            row++;  
28 v        }  
29 v        // 是否超出边界  
30 v        if (row < 0 || row >= mHeight || col < 0 || col >= mWidth)  
31 v            return -1;  
32 v        // 吃到食物加头  
33 v        if (mFoodId < mFood.length && row == mFood[mFoodId][0] && col == mFood[mFoodId][1]) {  
34 v            mSeens.add(row * mWidth + col);  
35 v            mSnack.addLast(row * mWidth + col);  
36 v            mFoodId++;  
37 v            return ++mScore;  
38 v        }  
39 v        mSeens.remove(mSnack.pollFirst()); // 去尾  
40 v        if (mSeens.contains(row * mWidth + col)) { // 检查是否与自身相撞  
41 v            return -1;  
42 v        } else { // 加头  
43 v            mSeens.add(row * mWidth + col);  
44 v            mSnack.addLast(row * mWidth + col);  
45 v            return mScore;  
46 v        }  
47 v    }  
48 v }
```

## 356. 直线镜像

356. 直线镜像

难度 中等 通过 7 收藏 10 举报

在一个二维平面空间中，给你  $n$  个点的坐标。问，是否能找出一条平行于  $y$  轴的直线，让这些点关于这条直线成镜像排布？

示例 1：

输入：  $[[1,1], [-1,1]]$   
输出： true

示例 2：

输入：  $[[1,1], [-1,-1]]$   
输出： false

拓展：  
你能找到比  $O(n^2)$  更优的解法吗？

```
1 v class Solution {  
2 v     public boolean isReflected(int[][] points) {  
3 v         if (points == null || points.length == 0 || points[0].length == 0)  
4 v             return false;  
5 v         Map<Integer, Set<Integer>> map = new HashMap<>();  
6 v         int lid = Integer.MAX_VALUE, rid = Integer.MIN_VALUE;  
7 v         for (int point : points) {  
8 v             lid = Math.min(lid, point[0]);  
9 v             rid = Math.max(rid, point[0]);  
10 v            Set<Integer> set = map.getOrDefault(point[1], new HashSet<>());  
11 v            set.add(point[0]);  
12 v            map.put(point[1], set);  
13 v        }  
14 v        int mid = lid + rid; // 找到对称点  
15 v        for (Set<Integer> set : map.values()) {  
16 v            for (int val : set) {  
17 v                if (!set.contains(mid - val))  
18 v                    return false;  
19 v            }  
20 v        }  
21 v        return true;  
22 v    }  
23 v }
```

## 358. K 距离间隔重排字符串

358. K 距离间隔重排字符串

难度 困难 ⚡ 19 ❤️ ⚡ 🌟 ⚡

给你一个非空的字符串  $s$  和一个整数  $k$ ，你要将这个字符串中的字母进行重新排列，使得重排后的字符串中相同字母的位置间隔至少为  $k$ 。

所有输入的字符串都由小写字母组成，如果找不到距离至少为  $k$  的重排结果，请返回一个空字符串 `""`。

示例 1：

输入:  $s = "aabbcc"$ ,  $k = 3$

输出: `"abcabc"`

解释: 相同的字母在新的字符串中间隔至少 3 个单位距离。

示例 2：

输入:  $s = "aaabbc"$ ,  $k = 3$

输出: `""`

解释: 没有办法找到可能的重排结果。

示例 3：

输入:  $s = "aaadbccc"$ ,  $k = 2$

输出: `"abacabcd"`

解释: 相同的字母在新的字符串中间隔至少 2 个单位距离。

```
1 class Solution {
2     public String rearrangeString(String s, int k) {
3         if(k <= 1)
4             return s;
5         int[] count = new int[26];
6         for(char c: s.toCharArray())
7             ++count[c - 'a'];
8         Queue<Integer> queue = new PriorityQueue<>(
9             (o1, o2) -> count[o2] - count[o1] == 0 ? o1 - o2 : count[o2] - count[o1]);
10        for(int i = 0; i < 26; ++i)
11            if(count[i] != 0)
12                queue.offer(i);
13        StringBuilder buffer = new StringBuilder();
14        for(int i = 0; i < s.length(); i += k) {
15            List<Integer> list = new ArrayList<>();
16            for(int j = 0; j < k && i + j < s.length(); ++j) {
17                if(queue.isEmpty())
18                    return "";
19                int c = queue.poll();
20                list.add(c);
21                --count[c];
22                buffer.append((char)('a' + c));
23            }
24            for(int c: list)
25                if(count[c] != 0)
26                    queue.offer(c);
27        }
28        return buffer.toString();
29    }
30 }
31
32 }
```

## 359. 日志速率限制器

359. 日志速率限制器

难度 简单 ⚡ 7 ❤️ ⚡ 🌟 ⚡

请你设计一个日志系统，可以流式接收日志以及它的时间戳。

该日志会被打印出来，需要满足一个条件：当且仅当日志内容 在过去的 10 秒钟内没有被打印过。

给你一条日志的内容和它的时间戳（粒度为秒级），如果这条日志在给定的时间戳应该被打印出来，则返回 `true`，否则请返回 `false`。

要注意的是，可能会有多条日志在同一时间被系统接收。

```
1 class Logger {
2     private Map<String, Integer> mMap;
3     public Logger() {
4         mMap = new HashMap<>();
5     }
6     public boolean shouldPrintMessage(int timestamp, String message) {
7         if (mMap.containsKey(message) && timestamp - mMap.get(message) < 10) {
8             return false;
9         }
10        mMap.put(message, timestamp);
11        return true;
12    }
13 }
```

## 360. 有序转化数组

360. 有序转化数组

难度 中等 ⚡ 10 ❤️ ⚡ 🌟 ⚡

给你一个已经 排好序 的整数数组  $nums$  和整数  $a$ 、 $b$ 、 $c$ ，对于数组中的每一个数  $x$ ，计算函数值  $f(x) = ax^2 + bx + c$ ，请将函数值产生的数组返回。

要注意，返回的这个数组必须按照 升序排列，并且我们所期望的解法时间复杂度为  $O(n)$ 。

示例 1：

输入:  $nums = [-4, -2, 2, 4]$ ,  $a = 1$ ,  $b = 3$ ,  $c = 5$   
输出: `[3, 9, 15, 33]`

示例 2：

输入:  $nums = [-4, -2, 2, 4]$ ,  $a = -1$ ,  $b = 3$ ,  $c = 5$   
输出: `[-23, -5, 1, 7]`

通过次数 1,028 提交次数 1,712

在真实的面试中遇到过这道题?

贡献者

相关企业 

相关标签

相似题目

显示提示1

```
1 class Solution {
2     public int[] sortTransformedArray(int[] nums, int a, int b, int c) {
3         int[] ans = new int[nums.length];
4         if (a == 0) {
5             if (b == 0)
6                 Arrays.fill(ans, c);
7             for (int idx = 0; idx < nums.length; idx++) {
8                 if (b > 0)
9                     ans[idx] = b * nums[idx] + c;
10                else
11                    ans[nums.length - 1 - idx] = b * nums[idx] + c;
12            }
13            return ans;
14        }
15        int idx = 0;
16        double mid = -b * 1.0 / a / 2;
17        int lid = 0, rid = nums.length - 1;
18        if (a > 0) {
19            idx = nums.length - 1;
20            while (lid <= rid) {
21                if (Math.abs(mid - nums[lid]) > Math.abs(mid - nums[rid]))
22                    ans[idx] = fun(nums[lid++], a, b, c); // 最大值
23                else
24                    ans[idx] = fun(nums[rid--], a, b, c);
25                idx--;
26            }
27        } else {
28            while (lid <= rid) {
29                if (Math.abs(mid - nums[lid]) > Math.abs(mid - nums[rid]))
30                    ans[idx++] = fun(nums[lid++], a, b, c); // 最小值
31                else
32                    ans[idx++] = fun(nums[rid--], a, b, c);
33            }
34        }
35        return ans;
36    }
37    private int fun(int x, int a, int b, int c) {
38        return a * x * x + b * x + c;
39    }
40 }
```

## 361. 轰炸敌人

### 361. 轰炸敌人

难度 中等 13 喜欢 20 收藏 14 回归

想象一下炸弹人游戏，在你面前有一个二维的网格来表示地图，网格中的格子分别被以下三种符号占据：

- 'W' 表示一堵墙
- 'E' 表示一个敌人
- '0' (数字 0) 表示一个空位



请你计算一个炸弹最多能炸多少敌人。

由于炸弹的威力不足以穿透墙体，炸弹只能炸到同一行和同一列没被墙体挡住的敌人。

注意：你只能把炸弹放在一个空的格子里

示例：

```
输入: [["0","E","0","0"],["E","0","W","E"],  
["0","E","0","0"]]  
输出: 3  
解释: 对于如下网格
```

```
0 E 0 0  
E 0 W E  
0 E 0 0
```

假如在位置 (1,1) 放置炸弹的话，可以炸到 3 个敌人

通过次数 1,305 | 提交次数 2,490

在真实的面试中遇到过这道题？

```
1 class Solution {  
2     public int maxKilledEnemies(char[][] grid) {  
3         if (grid == null || grid.length == 0 || grid[0].length == 0) return 0;  
4         int max = 0, rNum = grid.length, cNum = grid[0].length;  
5         int[] rows = new int[rNum], cols = new int[cNum];  
6         for (int i = 0; i < rNum; i++) rows[i] = -1;  
7         for (int i = 0; i < cNum; i++) cols[i] = -1;  
8         for (int rid = 0; rid < rNum; rid++) {  
9             for (int cid = 0; cid < cNum; cid++) {  
10                 if (grid[rid][cid] == 'W') {  
11                     rows[rid] = -1; cols[cid] = -1; continue;  
12                 }  
13                 if (grid[rid][cid] == 'E') continue;  
14                 if (rows[rid] == -1) {  
15                     rows[rid] = 0;  
16                     for (int idx = cid - 1; idx >= 0; idx--) {  
17                         if (grid[rid][idx] == 'E')  
18                             rows[rid]++;  
19                         else if (grid[rid][idx] == 'W')  
20                             break;  
21                     }  
22                 }  
23                 for (int idx = cid + 1; idx < cNum; idx++) {  
24                     if (grid[rid][idx] == 'E')  
25                         rows[rid]++;  
26                     else if (grid[rid][idx] == 'W')  
27                         break;  
28                 }  
29             }  
30             if (cols[cid] == -1) {  
31                 cols[cid] = 0;  
32                 for (int idx = rid - 1; idx >= 0; idx--) {  
33                     if (grid[idx][cid] == 'E')  
34                         cols[cid]++;  
35                     else if (grid[idx][cid] == 'W')  
36                         break;  
37                 }  
38                 for (int idx = rid + 1; idx < rNum; idx++) {  
39                     if (grid[idx][cid] == 'E')  
40                         cols[cid]++;  
41                     else if (grid[idx][cid] == 'W')  
42                         break;  
43                 }  
44             }  
45             max = Math.max(max, rows[rid] + cols[cid]);  
46         }  
47     }  
48     return max;  
49 }
```

## 362. 敲击计数器

### 362. 敲击计数器

难度 中等 18 喜欢 20 收藏 14 回归

设计一个敲击计数器，使它可以统计在过去5分钟内被敲击次数。

每个函数会接收一个时间戳参数（以秒为单位），你可以假设最早的时间戳从1开始，且都是按照时间顺序对系统进行调用（即时间戳是单调递增）。

在同一时刻有可能会有多次敲击。

示例：

```
HitCounter counter = new HitCounter();  
  
// 在时刻 1 敲击一次。  
counter.hit(1);
```

```
1 class HitCounter {  
2     private List<Integer> mList;  
3  
4     public HitCounter() {  
5         this.mList = new ArrayList<>();  
6     }  
7  
8     public void hit(int timestamp) {  
9         mList.add(timestamp);  
10    }  
11  
12    public int getHits(int timestamp) {  
13        while (mList.size() > 0 && timestamp - mList.get(0) >= 300) {  
14            mList.remove(0);  
15        }  
16        return mList.size();  
17    }  
18}
```

## 364. 加权嵌套序列和 II

### 364. 加权嵌套序列和 II

难度 中等 13 喜欢 20 收藏 14 回归

给一个嵌套整数序列，请你返回每个数字在序列中的加权和，它们的权重由它们的深度决定。

序列中的每一个元素要么是一个整数，要么是一个序列（这个序列中的每个元素也同样是整数或序列）。

与 前一个问题 不同的是，前一题的权重按照从根到叶逐一增加，而本题的权重从叶到根逐一增加。

也就是说，在本题中，叶子的权重为1，而根拥有最大的权重。

示例 1：

```
输入: [[1,1],2,[1,1]]  
输出: 8  
解释: 四个 1 在深度为 1 的位置，一个 2 在深度为 2 的位置。
```

```
1 class Solution {  
2     public int depthSumInverse(List<NestedInteger> nestedList) {  
3         int prevSum = 0, currSum = 0;  
4         Queue<NestedInteger> queue = new LinkedList<>();  
5         for (NestedInteger ni : nestedList)  
6             queue.offer(ni);  
7         while (!queue.isEmpty()) {  
8             int size = queue.size(), level = 0;  
9             for (int i = 0; i < size; i++) {  
10                 NestedInteger current = queue.poll();  
11                 if (current.isInteger()) {  
12                     level += current.getInteger();  
13                 } else {  
14                     for (NestedInteger ni : current.getList())  
15                         queue.offer(ni);  
16                 }  
17             }  
18             prevSum += level;  
19             currSum += prevSum;  
20         }  
21         return currSum;  
22     }  
23 }
```

## 366. 寻找二叉树的叶子结点

### 366. 寻找二叉树的叶子节点

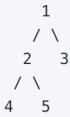
难度 中等 通过率 28% 收藏 复制

给你一棵二叉树，请按以下要求的顺序收集它的全部节点：

1. 依次从左到右，每次收集并删除所有的叶子节点
2. 重复如上过程直到整棵树为空

示例：

输入： [1,2,3,4,5]



输出： [[4,5,3],[2],[1]]

```
1 class Solution {  
2     public List<List<Integer>> findLeaves(TreeNode root) {  
3         List<List<Integer>> ans = new ArrayList<>();  
4         if (root == null) {  
5             return ans;  
6         }  
7         while (root != null) {  
8             List<Integer> list = new ArrayList<>();  
9             root = dfs(root, list);  
10            ans.add(list);  
11        }  
12        return ans;  
13    }  
14    private TreeNode dfs(TreeNode node, List<Integer> list) {  
15        if (node == null) {  
16            return null;  
17        }  
18        if (node.left == null && node.right == null) {  
19            list.add(node.val);  
20            node = null;  
21            return null;  
22        }  
23        node.left = dfs(node.left, list);  
24        node.right = dfs(node.right, list);  
25        return node;  
26    }  
27}
```

## 369. 给单链表加一

### 369. 给单链表加一

难度 中等 通过率 27% 收藏 复制

用一个 **非空** 单链表来表示一个非负整数，然后将这个整数加一。

你可以假设这个整数除了 0 本身，没有任何前导的 0。

这个整数的各个数位按照 **高位在链表头部、低位在链表尾部** 的顺序排列。

示例：

输入： [1,2,3]

输出： [1,2,4]

通过次数 1,644 | 提交次数 2,697

在真实的面试中遇到过这道题？

```
1 class Solution {  
2     public ListNode plusOne(ListNode head) { // 双指针  
3         ListNode slow = new ListNode(0);  
4         ListNode fast = head;  
5         slow.next = head;  
6         // 遍历链表  
7         while (fast != null) {  
8             if (fast.val != 9) {  
9                 slow = fast;  
10            }  
11            fast = fast.next;  
12        }  
13        // 末位加1  
14        slow.val += 1;  
15        ListNode curr = slow.next;  
16        while (curr != null) {  
17            curr.val = 0;  
18            curr = curr.next;  
19        }  
20        return slow.next == head ? slow : head;  
21    }  
22}
```

## 370. 区间加法

### 370. 区间加法

难度 中等 通过率 16% 收藏 复制

假设你有一个长度为  $n$  的数组，初始情况下所有的数字均为 0，你将会被给出  $k$  个更新的操作。

其中，每个操作会被表示为一个三元组：[startIndex, endIndex, inc]，你需要将子数组  $A[\text{startIndex} \dots \text{endIndex}]$ （包括 startIndex 和 endIndex）增加 inc。

请你返回  $k$  次操作后的数组。

示例：

输入： length = 5, updates = [[1,3,2],[2,4,3],  
[0,2,-2]]  
输出： [-2,0,3,5,3]

```
1 class Solution {  
2     public int[] getModifiedArray(int length, int[][] updates) {  
3         if (length == 0 || updates == null) {  
4             return new int[0];  
5         }  
6         int[] ans = new int[length];  
7         if (updates.length == 0 || updates[0].length == 0) {  
8             return ans;  
9         }  
10        for (int[] update : updates) {  
11            if (update[0] < 0 || update[1] < 0 || update[0] >= length  
12                || update[1] >= length || update[0] > update[1]) {  
13                return new int[0];  
14            }  
15            for (int idx = update[0]; idx <= update[1]; idx++) {  
16                ans[idx] += update[2];  
17            }  
18        }  
19        return ans;  
20    }  
21}
```

## 379. 电话目录管理系统

### 379. 电话目录管理系统

难度 中等 | 11 | 0 | 0 | 0 | 0 | 0

设计一个电话目录管理系统，让它支持以下功能：

1. `get`：分配给用户一个未被使用的电话号码，获取失败请返回 -1
2. `check`：检查指定的电话号码是否被使用
3. `release`：释放掉一个电话号码，使其能够重新被分配

示例：

```
// 初始化电话目录，包括 3 个电话号码: 0, 1 和 2。
PhoneDirectory directory = new PhoneDirectory(3);

// 可以返回任意未分配的号码，这里我们假设它返回 0。
directory.get();

// 假设，函数返回 1。
```

## 408. 有效单词缩写

### 408. 有效单词缩写

难度 简单 | 12 | 0 | 0 | 0 | 0 | 0

给一个非空字符串 `s` 和一个单词缩写 `abbr`，判断这个缩写是否可以是给定单词的缩写。

字符串 "word" 的所有有效缩写：

```
["word", "l0rd", "w1rd", "wo1d", "wor1", "2rd",
 "w2d", "wo2", "l0ld", "1or1", "w1r1", "102",
 "2r1", "3d", "w3", "4"]
```

注意单词 "word" 的所有有效缩写仅包含以上这些。任何其他的字符串都不是 "word" 的有效缩写。

注意：

假设字符串 `s` 仅包含小写字母且 `abbr` 只包含小写字母和数字。

## 411. 最短特异单词缩写

### 411. 最短特异单词缩写

难度 困难 | 8 | 0 | 0 | 0 | 0 | 0

字符串 "word" 包含以下这些缩写形式：

```
["word", "l0rd", "w1rd", "wo1d", "wor1", "2rd",
 "w2d", "wo2", "l0ld", "1or1", "w1r1", "102",
 "2r1", "3d", "w3", "4"]
```

给一个目标字符串和一个字符串字典，为目标字符串找一个最短长度的缩写字符串，同时这个缩写字符串不是字典中其他字符串的缩写形式。

缩写形式中每一个数字或者字母都被视为长度为 1。比方说，缩写形式 "a32bc" 的长度为 4 而不是 5。

注意：

- 如果像第二个示例一样有多个有效答案，你可以返回它们中的任意一个。
- 假设目标字符串的长度为 `m`，字典中的字符串数目为 `n`。你可以假设  $m \leq 21$ ,  $n \leq 1000$ , 且  $\log_2(n) + m \leq 20$ 。

示例：

```
"apple", ["blade"] -> "a4" (因为 "5" 或者 "4e" 同时也是 "blade" 的缩写形式，所以它们是无效的缩写)

"apple", ["plain", "amber", "blade"] -> "1p3" (其他有效的缩写形式还包括 "ap3", "a3e", "2p2", "3le", "3l1")。
```

通过次数 384 | 提交次数 829

在真实的面试中遇到过这道题？

举报

```
1 class PhoneDirectory {
2     private TreeSet<Integer> mTreeSet;
3     public PhoneDirectory(int maxNumbers) {
4         this.mTreeSet = new TreeSet<>();
5         for (int i = 0; i < maxNumbers; i++) {
6             mTreeSet.add(i);
7         }
8     }
9     public int get() {
10        if (mTreeSet.isEmpty()) {
11            return -1;
12        }
13        return mTreeSet.pollFirst();
14    }
15    public boolean check(int number) {
16        return mTreeSet.contains(number);
17    }
18    public void release(int number) {
19        mTreeSet.add(number);
20    }
21 }
```

```
1 class Solution {
2     public boolean validWordAbbreviation(String word, String abbr) {
3         int abbrLen = abbr.length(), wordLen = word.length();
4         int len = 0, num = 0;
5         for (int i = 0; i < abbrLen; ++i) {
6             if (abbr.charAt(i) >= 'a' && abbr.charAt(i) <= 'z') {
7                 len += num + 1;
8                 num = 0; // WARNING
9                 if (len > wordLen || abbr.charAt(i) != word.charAt(len - 1))
10                     return false;
11             } else {
12                 if (num == 0 && abbr.charAt(i) == '0')
13                     return false;
14                 num = num * 10 + abbr.charAt(i) - '0';
15             }
16         }
17         return len + num == wordLen;
18     }
19 }
```

```
1 class Solution {
2     public String minAbbreviation(String target, String[] dictionary) {
3         int tLen = target.length(), bound = 1 << tLen;
4         List<Integer> list = new ArrayList<>();
5         for (String str : dictionary) {
6             int clen = str.length();
7             if (clen != tLen) continue;
8             int bits = 0, curr = bound >> 1;
9             for (int i = 0; i < clen; i++) {
10                 if (str.charAt(i) != target.charAt(i)) bits |= curr;
11                 curr >>= 1;
12             }
13             list.add(bits);
14         }
15         if (list.isEmpty()) return String.valueOf(target.length());
16         int minSize = tLen, minMask = bound - 1;
17         for (int mask = 1; mask < bound; mask++) {
18             int size = callen(mask, tLen);
19             if (size >= minSize) continue;
20             boolean collision = false;
21             for (int i : list)
22                 if ((i & mask) == 0) { collision = true; break; }
23             if (collision) continue;
24             minSize = size; minMask = mask;
25         }
26         StringBuilder buffer = new StringBuilder();
27         int sht = bound >> 1, cnt = 0;
28         for (int i = 0; i < tLen; i++) {
29             if ((minMask & sht) != 0) {
30                 if (cnt != 0) buffer.append(cnt);
31                 buffer.append(target.charAt(i)); cnt = 0;
32             } else ++cnt;
33             sht >>= 1;
34         }
35         if (cnt != 0) buffer.append(cnt);
36         return buffer.toString();
37     }
38     private int callen(int mask, int n) {
39         int cnt = 0, res = 0, tmp = n;
40         while (tmp - !res) {
41             if ((mask & 1) != 0) { ++res; cnt = 0; } else ++cnt;
42             mask >>= 1;
43         }
44         if (cnt != 0) ++res;
45         return res;
46     }
47 }
```

## 418. 屏幕可显示句子的数量

### 418. 屏幕可显示句子的数量

难度 中等 18 热度 2A 举报

给你一个  $rows \times cols$  的屏幕和一个用 非空 的单词列表组成的句子，请你计算给出定句子可以在屏幕上完整显示的次数。

注意：

1. 一个单词不能拆分成两行。
2. 单词在句子中的顺序必须保持不变。
3. 在一行中 的两个连续单词必须用一个空格符分隔。
4. 句子中的单词总量不会超过 100。
5. 每个单词的长度大于 0 且不会超过 10。
6.  $1 \leq rows, cols \leq 20,000$ .

示例 1：

```
输入:  
rows = 2, cols = 8, 句子 sentence = ["hello",  
"world"]  
  
输出:  
1
```

```
1 class Solution {  
2     public int wordsTyping(String[] sentence, int rows, int cols) {  
3         final int len = sentence.length;  
4         final int[] idxs = new int[len + 2];  
5         final int[] nums = new int[len + 2];  
6         int idx = 0, cnt = 0;  
7         for (int i = 1; i <= Math.min(rows, len + 1); i++) {  
8             int tmp = cols;  
9             while (tmp >= sentence[idx].length()) {  
10                 tmp -= (sentence[idx].length() + 1);  
11                 if (++idx == len) {  
12                     cnt++;  
13                     idx = 0;  
14                 }  
15             }  
16             for (int x = 1; x < i; x++) {  
17                 if (idxs[x] == idx) {  
18                     int repeat = (rows - x) / (i - x) * (cnt - nums[x]);  
19                     int other = x + (rows - x) % (i - x);  
20                     return repeat + nums[other];  
21                 }  
22             }  
23             idxs[i] = idx;  
24             nums[i] = cnt;  
25         }  
26     }  
27     return cnt;  
28 }
```

## 422. 有效的单词方块

### 422. 有效的单词方块

难度 简单 11 热度 2A 举报

给你一个单词序列，判断其是否形成了一个有效的单词方块。

有效的单词方块是指此由单词序列组成的名字方块的第 k 行 和 第 k 列 ( $0 \leq k < \max(\text{行数}, \text{列数})$ ) 所显示的字符串完全相同。

注意：

1. 给定的单词数大于等于 1 且不超过 500。
2. 单词长度大于等于 1 且不超过 500。
3. 每个单词只包含小写英文字母 `a-z`。

示例 1：

```
输入:  
[  
    "abcd",  
    "bnrt",  
    "cmy",  
    "dtye"  
]  
  
输出:  
true
```

```
1 class Solution {  
2     public boolean validWordSquare(List<String> words) {  
3         if (words == null || words.size() == 0) {  
4             return false;  
5         }  
6         int size = words.size();  
7         for (String word : words) { // WARNING  
8             size = Math.max(size, word.length());  
9         }  
10        char[][] grid = new char[size][size];  
11        int idx = 0;  
12        for (String word : words) {  
13            if (word == null || word.isEmpty()) {  
14                return false;  
15            }  
16            int len = word.length();  
17            for (int cid = 0; cid < len; cid++) {  
18                grid[idx][cid] = word.charAt(cid);  
19            }  
20            ++idx;  
21        }  
22        for (int rid = 0; rid < size; rid++) {  
23            for (int cid = 0; cid < size; cid++) {  
24                if (grid[rid][cid] != grid[cid][rid]) {  
25                    return false;  
26                }  
27            }  
28        }  
29        return true;  
30    }  
31 }
```

## 425. 单词方块

### 425. 单词方块

难度 困难 ⌂ 19 ❤ 1 文章

给定一个单词集合（没有重复），找出其中所有的 单词方块。

一个单词序列形成了一个有效的单词方块的意思是指从第 k 行和第 k 列 ( $0 \leq k < \max(\text{行数}, \text{列数})$ ) 来看都是相同的字符串。

例如，单词序列 `["ball", "area", "lead", "lady"]` 形成了一个单词方块，因为每个单词从水平方向看和从竖直方向看都是相同的。

```
b a l l  
a r e a  
l e a d  
l a d y
```

注意：

1. 单词个数大于等于 1 且不超过 500。
2. 所有的单词长度都相同。
3. 单词长度大于等于 1 且不超过 5。
4. 每个单词只包含小写英文字母 `a-z`。

示例 1：

```
输入：  
["area", "lead", "wall", "lady", "ball"]  
  
输出：  
[  
  [ "wall",  
    "area",  
    "lead",  
    "lady"  
  ],  
  [ "ball",  
    "area",  
    "lead",  
    "lady"  
  ]  
]
```

```
1 v class Solution {  
2 v   public List<List<String>> wordSquares(String[] words) {  
3 v     List<List<String>> ans = new ArrayList<>();  
4 v     if (words == null || words.length == 0 || words[0].length() == 0) {  
5 v       return ans;  
6 v     }  
7 v     Node root = new Node();  
8 v     for (String word : words) addWord(word, root);  
9 v     for (String word : words) {  
10 v       String[] strings = new String[word.length()];  
11 v       strings[0] = word;  
12 v       dfs(strings, 1, root, ans);  
13 v     }  
14 v   }  
15 v  
16 v   private void addWord(String word, Node root) {  
17 v     Node node = root;  
18 v     for (char ch : word.toCharArray()) {  
19 v       if (node.childs[ch - 'a'] == null) node.childs[ch - 'a'] = new Node();  
20 v       node = node.childs[ch - 'a'];  
21 v       node.words.add(word);  
22 v     }  
23 v   }  
24 v  
25 v   private void dfs(String[] words, int idx, Node root, List<List<String>> list) {  
26 v     int len = words.length;  
27 v     if (idx == len) {  
28 v       list.add(Arrays.asList(Arrays.copyOf(words, len)));  
29 v       return;  
30 v     }  
31 v     Node node = root;  
32 v     for (int i = 0; i < idx; i++) {  
33 v       char c = words[i].charAt(i);  
34 v       if (node.childs[c - 'a'] == null) return;  
35 v       node = node.childs[c - 'a'];  
36 v     }  
37 v     for (String next : node.words) {  
38 v       words[idx] = next;  
39 v       dfs(words, idx + 1, root, list);  
40 v     }  
41 v   }  
42 v  
43 v   private class Node {  
44 v     Node[] childs = new Node[26];  
45 v     List<String> words = new ArrayList<>();  
}
```

## 426. 将二叉搜索树转化为排序的双向链表

### 426. 将二叉搜索树转化为排序的双向链表

难度 中等 ⌂ 33 ❤ 1 文章

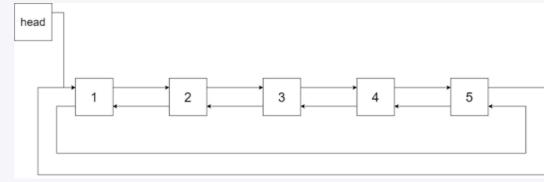
将一个 二叉搜索树 就地转化为一个 已排序的双向循环链表。

对于双向循环列表，你可以将左右孩子指针作为双向循环链表的前驱和后继指针，第一个节点的前驱是最后一个节点，最后一个节点的后继是第一个节点。

特别地，我们希望可以 就地 完成转换操作。当转化完成以后，树中节点的左指针需要指向前驱，树中节点的右指针需要指向后继。还需要返回链表中最小元素的指针。

示例 1：

输入：root = [4,2,5,1,3]



输出：[1,2,3,4,5]

```
1 v class Solution {  
2 v   private Node mHead;  
3 v   private Node mTail;  
4 v   public Node treeToDoublyList(Node root) {  
5 v     if (root == null) {  
6 v       return null;  
7 v     }  
8 v     inorder(root);  
9 v     mTail.right = mHead;  
10 v    mHead.left = mTail;  
11 v    return mHead;  
12 v  }  
13 v  public void inorder(Node node) {  
14 v    if (node == null) {  
15 v      return;  
16 v    }  
17 v    inorder(node.left);  
18 v    if (mTail != null) {  
19 v      mTail.right = node;  
20 v      node.left = mTail;  
21 v    } else {  
22 v      mHead = node;  
23 v    }  
24 v    mTail = node;  
25 v    inorder(node.right);  
26 v  }
```

## 428. 序列化和反序列化 N 叉树

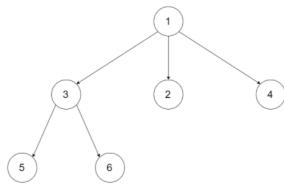
### 428. 序列化和反序列化 N 叉树

难度 困难 17 热度 20 文章 4

序列化是指将一个数据结构转化为位序列的过程，因此可以将其存储在文件中或内存缓冲区中，以便稍后在相同或不同的计算机环境中恢复结构。

设计一个序列化和反序列化 N 叉树的算法。一个 N 叉树是指每个节点都有不超过 N 个孩子节点的有根树。序列化 / 反序列化算法的算法实现没有限制。你只需要保证 N 叉树可以被序列化为一个字符串并且该字符串可以被反序列化成原树结构即可。

例如，你需要序列化下面的 3-叉 树。



为 [1 [3[5 6] 2 4]]。你不需要以这种形式完成，你可以自己创造和实现不同的方法。

注意：

1. N 的范围在 [1, 1000]
2. 不要使用类成员 / 全局变量 / 静态变量来存储状态。你的序列化和反序列化算法应是无状态的。

通过次数 654 | 提交次数 1.079

在真实的面试中遇到过这道题？

贡献者

相关企业

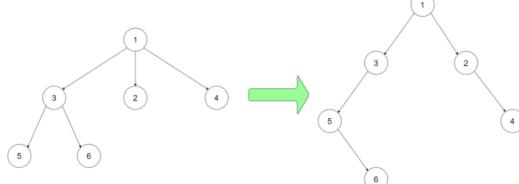
## 431. 将 N 叉树编码为二叉树

### 431. 将 N 叉树编码为二叉树

难度 困难 11 热度 20 文章 4

设计一个算法，可以将 N 叉树编码为二叉树，并能将该二叉树解码为原 N 叉树。一个 N 叉树是指每个节点都有不超过 N 个孩子节点的有根树。类似地，一个二叉树是指每个节点都有不超过 2 个孩子节点的有根树。你的编码 / 解码的算法的实现没有限制，你只需要保证一个 N 叉树可以编码为二叉树且该二叉树可以解码回原始 N 叉树即可。

例如，你可以将下面的 3-叉 树以该种方式编码：



注意，上面的方法仅仅是一个例子，可能可行也可能不可行。你没有必要遵循这种形式转化，你可以自己创造和实现不同的方法。

```
1 class Codec {  
2     public String serialize(Node root) {  
3         StringBuilder buffer = new StringBuilder();  
4         if (root == null) return buffer.toString();  
5         encode(root, buffer);  
6         return buffer.toString();  
7     }  
8  
9     private void encode(Node node, StringBuilder buffer) {  
10        if (node == null) {  
11            return;  
12        }  
13        buffer.append(node.val).append(" ");  
14        boolean hasChildren = !node.children.isEmpty();  
15        if (hasChildren) buffer.append("[");  
16        for (int i = 0; i < node.children.size(); i++) {  
17            Node children = node.children.get(i);  
18            encode(children, buffer);  
19            if (i == node.children.size() - 1)  
20                buffer.deleteCharAt(buffer.length() - 1);  
21        }  
22        if (hasChildren) buffer.append("]");  
23    }  
24  
25    public Node deserialize(String data) {  
26        if (data.isEmpty()) return null;  
27        String[] strings = data.split(" ");  
28        Stack<Node> stack = new Stack<Node>();  
29        Node root = null, curr = null;  
30        for (String string : strings) {  
31            if (string.equals("[")) {  
32                stack.push(curr);  
33            } else if (string.equals("]")) {  
34                stack.pop();  
35            } else {  
36                Node node = new Node(Integer.valueOf(string));  
37                node.children = new LinkedList<Node>();  
38                if (root == null) {  
39                    root = node;  
40                } else {  
41                    Node parent = stack.peek();  
42                    parent.children.add(node);  
43                }  
44                curr = node;  
45            }  
46        }  
47        return root;  
48    }  
49}
```

```
1 class Codec {  
2     public TreeNode encode(Node root) {  
3         if (root == null) return null;  
4         TreeNode node = new TreeNode(root.val);  
5         List<Node> children = root.children;  
6         if (children == null || children.isEmpty())  
7             return node;  
8         TreeNode head = encode(children.get(0));  
9         node.left = head;  
10        final int size = children.size();  
11        for (int idx = 1; idx < size; idx++) {  
12            head.right = encode(children.get(idx));  
13            head = head.right;  
14        }  
15        return node;  
16    }  
17  
18    public Node decode(TreeNode root) {  
19        if (root == null) return null;  
20        Node node = new Node(root.val, new ArrayList<>());  
21        TreeNode left = root.left;  
22        while (left != null) {  
23            node.children.add(decode(left));  
24            left = left.right;  
25        }  
26        return node;  
27    }  
28  
29}
```

## 439. 三元表达式解析器

### 439. 三元表达式解析器

难度 中等 通过 17 提交 答案 退出

给定一个以字符串表示的任意嵌套的三元表达式，计算表达式的值。你可以假定给定的表达式始终都是有效的并且只包含数字 0-9，?，:，T 和 F（T 和 F 分别表示真和假）。

注意：

1. 给定的字符串长度  $\leq 10000$ 。
2. 所包含的数字都只有一位数。
3. 条件表达式从右至左结合（和大多数程序设计语言类似）。
4. 条件是 T 和 F 其一，即条件永远不会是数字。
5. 表达式的结果是数字 0-9，T 或者 F。

示例 1：

```
1 class Solution {
2     public String parseTernary(String expression) {
3         int level = 0;
4         int len = expression.length();
5         for (int j = 1; j < len; j++) {
6             if (expression.charAt(j) == '?') {
7                 level++;
8             }
9             if (expression.charAt(j) == ':') {
10                level--;
11            }
12            if (level == 0) {
13                return (expression.charAt(0) == 'T')
14                    ? parseTernary(expression.substring(2, j))
15                    : parseTernary(expression.substring(j + 1, len));
16            }
17        }
18    }
19    return expression;
20 }
```

## 444. 序列重建

### 444. 序列重建

难度 中等 通过 12 提交 答案 退出

验证原始的序列 org 是否可以从序列集 seqs 中唯一地重建。序列 org 是 1 到 n 整数的排列，其中  $1 \leq n \leq 10^4$ 。重建是指在序列集 seqs 中构建最短的公共超序列。（即使得所有 seqs 中的序列都是该最短序列的子序列）。确定是否只可以从 seqs 重建唯一的序列，且该序列就是 org。

示例 1：

输入：  
org: [1,2,3], seqs: [[1,2],[1,3]]  
  
输出：  
false  
  
解释：  
[1,2,3] 不是可以被重建的唯一的序列，因为 [1,3,2] 也是一个合法的序列。

示例 2：

输入：  
org: [1,2,3], seqs: [[1,2]]  
  
输出：  
false  
  
解释：  
可以重建的序列只有 [1,2]。

```
1 class Solution {
2     public boolean sequenceReconstruction(int[] org, List<List<Integer>> seqs) {
3         if (org == null || org.length == 0 || seqs == null || seqs.isEmpty()) {
4             return false;
5         }
6         int len = org.length;
7         int[] idxs = new int[len + 1];
8         boolean[] visited = new boolean[len];
9         for (int idx = 0; idx < len; idx++) {
10            if (idxs[org[idx]] != 0) {
11                return false;
12            } else {
13                idxs[org[idx]] = idx;
14            }
15        }
16        boolean reconstructed = false;
17        for (List<Integer> seq : seqs) {
18            int last = -1;
19            for (Integer i : seq) {
20                if (i > len || i < 1) {
21                    return false;
22                }
23                if (last != -1) {
24                    if (idxs[i] - last == 1) {
25                        visited[last] = true;
26                    } else if (idxs[i] <= last) {
27                        return false;
28                    }
29                }
30                last = idxs[i];
31                if (idxs[i] == len - 1) {
32                    reconstructed = true;
33                }
34            }
35            for (int i = 0; i < len - 1; i++) {
36                if (!visited[i]) {
37                    return false;
38                }
39            }
40        }
41        return reconstructed;
42    }
43 }
```

## 469. 凸多边形

### 469. 凸多边形

难度 中等 通过 9 提交 答案 退出

给定一个按顺序连接的多边形的顶点，判断该多边形是否为凸多边形。（凸多边形的定义）

注：

1. 顶点个数至少为 3 个且不超过 10,000。
2. 坐标范围为 -10,000 到 10,000。
3. 你可以假定给定的点形成的多边形均为简单多边形（简单多边形的定义）。换句话说，保证每个顶点处恰好是两条边的汇合点，并且这些边互不相交。

示例 1：

[[0,0],[0,1],[1,1],[1,0]]  
输出： True

```
1 public class Solution {
2     public boolean isConvex(List<List<Integer>> points) {
3         int size = points.size();
4         long curr = 0;
5         long prev = 0;
6         for (int i = 0; i < size; i++) {
7             curr = crossProduct(points.get((i + 1) % size).get(0) - points.get(i).get(0),
8                                 points.get((i + 1) % size).get(1) - points.get(i).get(1),
9                                 points.get((i + 2) % size).get(0) - points.get(i).get(0),
10                                points.get((i + 2) % size).get(1) - points.get(i).get(1));
11            if (curr != 0) {
12                if (curr * prev < 0) {
13                    return false;
14                }
15                prev = curr;
16            }
17        }
18        return true;
19    }
20
21    private long crossProduct(long x1, long y1, long x2, long y2) {
22        return x1 * y2 - x2 * y1;
23    }
24 }
```

## 465. 最优账单平衡

### 465. 最优账单平衡

难度 困难 ⚡ 18 ❤️ 🔍 🌐 🗃

一群朋友在度假期间会相互借钱。比如说，小爱同学支付了小新同学的午餐共计 10 美元。如果小明同学支付了小爱同学的出租车钱共计 5 美元。我们可以用一个三元组  $(x, y, z)$  表示一次交易，表示  $x$  借给  $y$  共计  $z$  美元。用 0, 1, 2 表示小爱同学、小新同学和小明同学（0, 1, 2 为人的标号），上述交易可以表示为  $[[0, 1, 10], [2, 0, 5]]$ 。

给定一群人之间的交易信息列表，计算能够还清所有债务的最小次数。

注意：

1. 一次交易会以三元组  $(x, y, z)$  表示，并有  $x \neq y$  且  $z > 0$ 。
2. 人的标号可能不是按顺序的，例如标号可能为 0, 1, 2 也可能为 0, 2, 6。

示例 1：

输入：  
[[0,1,10], [2,0,5]]

输出：

2

解释：

人 #0 给人 #1 共计 10 美元。  
人 #2 给人 #0 共计 5 美元。

需要两次交易。一种方式是人 #1 分别给人 #0 和人 #2 各 5 美元。

```
1 class Solution {
2     public int minTransfers(int[][] transactions) {
3         Map<Integer, Integer> map = new HashMap<>();
4         for (int[] tran : transactions) {
5             map.put(tran[0], map.getOrDefault(tran[0], 0) + tran[2]);
6             map.put(tran[1], map.getOrDefault(tran[1], 0) - tran[2]);
7         }
8         List<Integer> list = new ArrayList<>();
9         for (int val : map.values()) {
10            if (val != 0) {
11                list.add(val);
12            }
13        }
14        return dfs(0, list);
15    }
16
17    private int dfs(int idx, List<Integer> list) {
18        if (idx == list.size()) {
19            return 0;
20        }
21        int curr = list.get(idx);
22        if (curr == 0) {
23            return dfs(idx + 1, list);
24        }
25        int min = Integer.MAX_VALUE;
26        for (int i = idx + 1; i < list.size(); i++) {
27            int next = list.get(i);
28            if (curr * next < 0) {
29                list.set(i, curr + next);
30                min = Math.min(min, 1 + dfs(idx + 1, list));
31                list.set(i, next);
32            }
33            if (curr + next == 0) {
34                break;
35            }
36        }
37        return min;
38    }
39}
```

## 471. 编码最短长度的字符串

### 471. 编码最短长度的字符串

难度 困难 ⚡ 15 ❤️ 🔍 🌐 🗃

给定一个非空字符串，将其编码为具有最短长度的字符串。

编码规则是： $k[\text{encoded\_string}]$ ，其中在方括号  $\text{encoded\_string}$  中的内容重复  $k$  次。

注：

1.  $k$  为正整数且编码后的字符串不能为空或有额外的空格。
2. 你可以假定输入的字符串只包含小写的英文字母。字符串长度不超过 160。
3. 如果编码的过程不能使字符串缩短，则不要对其进行编码。如果有多种编码方式，返回任意一种即可。

示例 1：

输入：“aaa”  
输出：“aaa”  
解释：无法将其编码为更短的字符串，因此不进行编码。

```
1 class Solution {
2     public String encode(String s) {
3         int len = s.length();
4         String[][] dp = new String[len][len + 1];
5         for (int idx = 1; idx <= len; idx++) {
6             for (int i = 0; i + idx <= len; i++) {
7                 int j = i + idx;
8                 String sub = s.substring(i, j);
9                 if (sub.length() < 5) {
10                     dp[i][j] = sub;
11                 } else {
12                     int pos = (sub + sub).indexOf(sub, 1);
13                     if (pos < sub.length()) {
14                         dp[i][j] = sub.length() / pos + "[" + dp[i][i + pos] + "]";
15                     } else {
16                         dp[i][j] = sub;
17                         for (int k = 1; k < sub.length(); k++) {
18                             int tmp = dp[i][i + k].length() + dp[i + k][j].length();
19                             if (dp[i][j].length() > tmp) {
20                                 dp[i][j] = dp[i][i + k] + dp[k + i][j];
21                             }
22                         }
23                     }
24                 }
25             }
26         }
27         return dp[0][s.length()];
28     }
29 }
```

## 484. 寻找排列

### 484. 寻找排列

难度 中等 14 喜欢 20 收藏 14

现在给定一个只由字符 'D' 和 'I' 组成的 **秘密签名**。'D' 表示两个数字间的递减关系，'I' 表示两个数字间的递增关系。并且 **秘密签名** 是由一个特定的整数数组生成的，该数组唯一地包含 1 到 n 中所有不同的数字（**秘密签名**的长度加 1 等于 n）。例如，**秘密签名** "DI" 可以由数组 [2,1,3] 或 [3,1,2] 生成，但是不能由数组 [3,2,4] 或 [2,1,3,4] 生成，因为它们都不是合法的能代表 "DI" **秘密签名** 的特定串。

现在你的任务是找到具有最小字典序的 [1, 2, ..., n] 的排列，使其能代表输入的 **秘密签名**。

示例 1：

```
输入: "I"
输出: [1,2]
解释: [1,2] 是唯一合法的可以生成秘密签名 "I" 的特定串，数字 1 和 2 构成递增关系。
```

示例 2：

```
输入: "DI"
输出: [2,1,3]
解释: [2,1,3] 和 [3,1,2] 可以生成秘密签名 "DI"，但是由于我们要找字典序最小的排列，因此你需要输出 [2,1,3]。
```

注：

- 输出字符串只会包含字符 'D' 和 'I'。
- 输入字符串的长度是一个正整数且不会超过 10,000。

通过次数 611 | 提交次数 942

## 485. 最大连续1的个数

### 485. 最大连续1的个数

难度 简单 93 喜欢 20 收藏 14

给定一个二进制数组，计算其中最大连续1的个数。

示例 1：

```
输入: [1,1,0,1,1,1]
输出: 3
解释: 开头的两位和最后的三位都是连续1，所以最大连续1的个数是3。
```

注意：

- 输入的数组只包含 0 和 1。

## 487. 最大连续1的个数 II

### 487. 最大连续1的个数 II

难度 中等 18 喜欢 20 收藏 14

给定一个二进制数组，你可以最多将 1 个 0 翻转为 1，找出其中最大连续 1 的个数。

示例 1：

```
输入: [1,0,1,1,0]
输出: 4
解释: 翻转第一个 0 可以得到最长的连续 1。
当翻转以后，最大连续 1 的个数为 4。
```

```
1 class Solution {
2     public int[] findPermutation(String s) {
3         if (s == null || s.isEmpty()) {
4             return new int[0];
5         }
6         int[] ans = new int[s.length() + 1];
7         dsf(ans, s, 0);
8         return ans;
9     }
10
11     private void dsf(int[] ret, String str, int idx) {
12         int len = str.length();
13         if (idx >= len) {
14             if (ret[len] == 0) {
15                 ret[len] = len + 1;
16             }
17             return;
18         }
19         int id = idx;
20         int dCnt = 0;
21         while (id < len && str.charAt(id) == 'D') {
22             dCnt++;
23             id++;
24         }
25         int iCnt = 0;
26         while (id < len && str.charAt(id) == 'I') {
27             iCnt++;
28             id++;
29         }
30         int digit = idx + 1 + dCnt;
31         for (int i = 0; i <= dCnt; i++) {
32             ret[idx] = digit;
33             digit--;
34             idx++;
35         }
36         digit = idx + 1;
37         for (int i = 0; i < iCnt - 1; i++) {
38             ret[idx] = digit;
39             digit++;
40             idx++;
41         }
42         dsf(ret, str, idx);
43     }
44 }
```

```
1 class Solution {
2     public int findMaxConsecutiveOnes(int[] nums) {
3         if (nums == null || nums.length == 0) {
4             throw new IllegalArgumentException("nums is empty");
5         }
6         int max = 0;
7         int cnt = 0;
8         for (int num : nums) {
9             if (num == 1) {
10                 ++cnt;
11             } else {
12                 max = Math.max(max, cnt);
13                 cnt = 0;
14             }
15         }
16         return Math.max(max, cnt);
17     }
18 }
```

```
1 class Solution { // 滑动窗口
2     public int findMaxConsecutiveOnes(int[] nums) {
3         int ans = 0, cnt = 0;
4         int lid = 0, rid = 0;
5         while (rid < nums.length) {
6             if (nums[rid] == 0) {
7                 cnt++;
8                 while (cnt > 1)
9                     cnt -= nums[lid++];
10            }
11            ans = Math.max(ans, rid - lid + 1);
12            ++rid;
13        }
14        return ans;
15    }
16 }
```

## 1004. 最大连续1的个数 III

### 1004. 最大连续1的个数 III

难度 中等 通过率 62% 热度 2A 单元测试

给定一个由若干 0 和 1 组成的数组 A，我们最多可以将 K 个值从 0 变成 1。

返回仅包含 1 的最长（连续）子数组的长度。

示例 1：

输入：A = [1,1,1,0,0,0,1,1,1,1,0], K = 2  
输出：6

```
1 class Solution { // 滑动窗口
2     public int longestOnes(int[] nums, int K) {
3         int ans = 0, cnt = 0;
4         int lid = 0, rid = 0;
5         while (rid < nums.length) {
6             if (nums[rid] == 0) {
7                 cnt++;
8                 while (cnt > K)
9                     cnt -= nums[lid++] == 0 ? 1 : 0;
10            }
11            ans = Math.max(ans, rid - lid + 1);
12            ++rid;
13        }
14        return ans;
15    }
16 }
```

## 490. 迷宫

### 490. 迷宫

难度 中等 通过率 30% 热度 2A 单元测试

由空地和墙组成的迷宫中有一个球。球可以向上下左右四个方向滚动，但在遇到墙壁前不会停止滚动。当球停下时，可以选择下一个方向。

给定球的起始位置，目的地和迷宫，判断球能否在目的地停下。

迷宫由一个0和1的二维数组表示。1表示墙壁，0表示空地。你可以假定迷宫的边缘都是墙壁。起始位置和目的地的坐标通过行号和列号给出。

示例 1：

输入 1：迷宫由以下二维数组表示  
0 0 1 0 0  
0 0 0 0 0  
0 0 0 1 0  
1 0 0 1 1  
0 0 0 0 0  
  
输入 2：起始位置坐标 (rowStart, colStart) = (0, 4)  
输入 3：目的地坐标 (rowDest, colDest) = (4, 4)  
  
输出：true

```
1 public class Solution {
2     public boolean hasPath(int[][] maze, int[] start, int[] destination) {
3         // TODO: 参数检查
4         final int[][] dirs = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
5         boolean[][] visited = new boolean[maze.length][maze[0].length];
6         Queue<int[]> queue = new LinkedList<int[]>;
7         queue.offer(start);
8         visited[start[0]][start[1]] = true;
9         while (!queue.isEmpty()) {
10             int[] node = queue.poll();
11             if (node[0] == destination[0] && node[1] == destination[1])
12                 return true;
13             for (int[] dir : dirs) {
14                 int rid = node[0] + dir[0];
15                 int cid = node[1] + dir[1];
16                 while (rid >= 0 && rid < maze.length
17                         && cid >= 0 && cid < maze[0].length
18                         && maze[rid][cid] == 0) {
19                     rid += dir[0];
20                     cid += dir[1];
21                 }
22                 if (visited[rid - dir[0]][cid - dir[1]]) {
23                     continue;
24                 }
25                 queue.offer(new int[]{rid - dir[0], cid - dir[1]});
26                 visited[rid - dir[0]][cid - dir[1]] = true;
27             }
28         }
29         return false;
30     }
31 }
```

## 499. 迷宫 III

### 499. 迷宫 III

难度 困难 通过率 17% 热度 2A 单元测试

由空地和墙组成的迷宫中有一个球。球可以向上 (u) 下 (d) 左 (l) 右 (r) 四个方向滚动。但在遇到墙壁前不会停止滚动。当球停下时，可以选择下一个方向。迷宫中还有一个洞。当球运动经过洞时，就会掉进洞里。

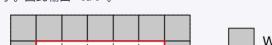
给定球的起始位置，目的地和迷宫，找出让球以最短距离掉进洞里的路径。路径的定义是球从起始位置（不包括）到目的地（包括）经过的空地个数。通过 'u', 'd', 'l' 和 'r' 输出球的移动方向。由于可能有多条最短路径，请输出字典序最小的路径。如果球无法进入洞，输出"impossible"。

迷宫由一个0和1的二维数组表示。1表示墙壁，0表示空地。你可以假定迷宫的边缘都是墙壁。起始位置和目的地的坐标通过行号和列号给出。

示例1：

输入 1：迷宫由以下二维数组表示  
0 0 0 0 0  
1 1 0 0 1  
0 0 0 0 0  
0 1 0 0 1  
0 1 0 0 0  
  
输入 2：球的初始位置 (rowBall, colBall) = (4, 3)  
输入 3：洞的位置 (rowHole, colHole) = (0, 1)  
  
输出："lul"

解析：有两条让球进洞的最短路径。  
第一条路径是 左 -> 上 -> 左，记为 "lul".  
第二条路径是 上 -> 左，记为 'ul'.  
两条路径都具有最短距离6，但'l' < 'u'，故第一条路径字典序更小。因此输出"lul"。



```
1 class Solution {
2     public String findShortestWay(int[][] maze, int[] ball, int[] hole) {
3         int rows = maze.length, cols = maze[0].length;
4         String[][] ways = new String[rows][cols];
5         for (int i = 0; i < rows; i++) {
6             for (int j = 0; j < cols; j++) {
7                 ways[i][j] = "";
8             }
9         }
10        final int[] rDirs = new int[]{-1, 1, 0, 0};
11        final int[] cDirs = new int[]{0, 0, 1, -1};
12        final int[][] steps = new int[rows][cols];
13        final String[] dicts = new String[]{"u", "d", "l", "r"};
14        Queue<int[]> queue = new LinkedList<int[]>;
15        queue.offer(ball);
16        while (!queue.isEmpty()) {
17            int[] node = queue.poll();
18            for (int dir = 0; dir < 4; dir++) {
19                int rid = node[0], cid = node[1];
20                while (rid >= 0 && rid < rows && cid >= 0 && cid < cols && maze[rid][cid] == 0) {
21                    if (rid == hole[0] && cid == hole[1]) {
22                        rid += rDirs[dir];
23                        cid += cDirs[dir];
24                        break;
25                    }
26                    rid += rDirs[dir];
27                    cid += cDirs[dir];
28                }
29                rid -= rDirs[dir];
30                cid -= cDirs[dir];
31                int step = steps[node[0]][node[1]] + Math.abs(rid - node[0]) + Math.abs(cid - node[1]);
32                if (((ways[node[0]][node[1]] + dicts[dir]).compareTo(ways[rid][cid]) < 0)
33                    && steps[rid][cid] == step || steps[rid][cid] > step) || steps[rid][cid] == 0)
34                    && (rid != node[0] || cid != node[1])) {
35                    steps[rid][cid] = step;
36                    ways[rid][cid] = ways[node[0]][node[1]] + dicts[dir];
37                    if (!(rid == hole[0] && cid == hole[1])) {
38                        queue.offer(new int[]{rid, cid});
39                    }
40                }
41            }
42        }
43        return ways[hole[0]][hole[1]].equals("") ? "impossible" : ways[hole[0]][hole[1]];
44    }
45 }
```

## 505. 迷宫 II

### 505. 迷宫 II

难度 中等 通过数 26 喜欢数 6 难度数 4

由空地和墙组成的迷宫中有一个球。球可以向上下左右四个方向滚动，但在遇到墙壁前不会停止滚动。当球停下时，可以选择下一个方向。

给定球的起始位置，目的地和迷宫，找出让球停在目的地的最短距离。距离的定义是球从起始位置（不包括）到目的地（包括）经过的空地个数。如果球无法停在目的地，返回 -1。

迷宫由一个0和1的二维数组表示。1表示墙壁，0表示空地。你可以假定迷宫的边缘都是墙壁。起始位置和目的地的坐标通过行号和列号给出。

示例 1：

输入 1：迷宫由以下二维数组表示

```
0 0 1 0 0  
0 0 0 0 0  
0 0 1 0 0  
1 0 0 1 1  
0 0 0 0 0
```

输入 2：起始位置坐标 (rowStart, colStart) = (0, 4)

输入 3：目的地坐标 (rowDest, colDest) = (4, 4)

```
1 public class Solution {  
2     public int shortestDistance(int[][] maze, int[] start, int[] dest) {  
3         // TODO: 参数检查  
4         final int[][] dirs = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};  
5         final int[][] dists = new int[maze.length][maze[0].length];  
6         for (int[] row : dists)  
7             Arrays.fill(row, Integer.MAX_VALUE);  
8         dists[start[0]][start[1]] = 0;  
9         Queue<int[]> queue = new LinkedList<>();  
10        queue.add(start);  
11        while (!queue.isEmpty()) {  
12            int[] node = queue.remove();  
13            for (int[] dir : dirs) {  
14                int rid = node[0] + dir[0];  
15                int cid = node[1] + dir[1];  
16                int cnt = 0;  
17                while (rid >= 0 && rid < maze.length  
18                    && cid >= 0 && cid < maze[0].length  
19                    && maze[rid][cid] == 0) {  
20                    rid += dir[0];  
21                    cid += dir[1];  
22                    cnt++;  
23                }  
24                if (dists[node[0]][node[1]] + cnt < dists[rid - dir[0]][cid - dir[1]]) {  
25                    dists[rid - dir[0]][cid - dir[1]] = dists[node[0]][node[1]] + cnt;  
26                    queue.add(new int[]{rid - dir[0], cid - dir[1]});  
27                }  
28            }  
29        }  
30        return dists[dest[0]][dest[1]] == Integer.MAX_VALUE ? -1 : dists[dest[0]][dest[1]];  
31    }  
32}
```

## 510. 二叉搜索树中的中序后继 II

### 510. 二叉搜索树中的中序后继 II

难度 中等 通过数 15 喜欢数 6 难度数 4

给定一棵二叉搜索树和其中的一个节点 `node`，找到该节点在树中的中序后继。

如果节点没有中序后继，请返回 `null`。

一个结点 `node` 的中序后继是键值比 `node.val` 大所有的结点中键值最小的那个。

你可以直接访问结点，但无法直接访问树。每个节点都会有其父节点的引用。节点定义如下：

```
class Node {  
    public int val;  
    public Node left;  
    public Node right;  
    public Node parent;  
}
```

```
1 class Solution {  
2     public Node inorderSuccessor(Node node) {  
3         if (node == null) {  
4             return null;  
5         }  
6         if (node.right != null) {  
7             node = node.right;  
8             while (node.left != null) {  
9                 node = node.left;  
10            }  
11            return node;  
12        }  
13        while (node.parent != null && node.parent.val < node.val) {  
14            node = node.parent;  
15        }  
16        return node.parent;  
17    }  
18}
```

## 527. 单词缩写

### 527. 单词缩写

难度 困难 通过数 9 喜欢数 6 难度数 4

给定一个由n个不重复非空字符串组成的数组，你需要按照以下规则为每个单词生成最小的缩写。

1. 初始缩写由起始字母+省略字母的数量+结尾字母组成。
2. 若存在冲突，亦即多于一个单词有同样的缩写，则使用更长的前缀代替首字母，直到从单词到缩写的映射唯一。换而言之，最终的缩写必须只能映射到一个单词。
3. 若缩写并不比原单词更短，则保留原样。

示例：

```
输入: ["like", "god", "internal", "me", "internet",  
"interval", "intension", "face", "intrusion"]  
输出:  
["l2e", "god", "internal", "me", "i6t", "interval", "inte4n"]
```

注意：

1. n和每个单词的长度均不超过 400。
2. 每个单词的长度大于 1。
3. 单词只由英文小写字母组成。
4. 返回的答案需要和原数组保持同一顺序。

通过次数 893 | 提交次数 1,755

在真实的面试中遇到过这道题？

是 否

贡献者



相关企业 i



相关标签



```
1 class Solution {  
2     public List<String> wordsAbbreviation(List<String> dict) {  
3         // TODO: 参数检查  
4         int size = dict.size(); String[] ans = new String[size];  
5         Map<String, List<IndexedWord>> map = new HashMap<>();  
6         for (int idx = 0; idx < size; ++idx) {  
7             List<IndexedWord> list = map.computeIfAbsent(  
8                 abbrev(dict.get(idx)), () -> new ArrayList<>());  
9             list.add(new IndexedWord(dict.get(idx), idx));  
10        }  
11        map.forEach((key, value) -> {  
12            Collections.sort(value, Comparator.comparing(a -> a.str));  
13            int[] prefixes = new int[value.size()];  
14            for (int idx = 1; idx < value.size(); ++idx) {  
15                int len = commonPrefix(value.get(idx - 1).str, value.get(idx).str);  
16                prefixes[idx] = len;  
17                prefixes[idx - 1] = Math.max(prefixes[idx - 1], len);  
18            }  
19            for (int idx = 0; idx < value.size(); ++idx)  
20                ans[value.get(idx).index] = abbrev(value.get(idx).str, prefixes[idx]);  
21        });  
22        return Arrays.asList(ans);  
23    }  
24    private String abbrev(String word, int idx) {  
25        int len = word.length();  
26        if (len - idx <= 3) return word;  
27        return word.substring(0, idx + 1) + (len - idx - 2) + word.charAt(len - 1);  
28    }  
29    private int commonPrefix(String word1, String word2) {  
30        int idx = 0;  
31        while (idx < word1.length() && idx < word2.length()  
32            && word1.charAt(idx) == word2.charAt(idx)) {  
33            idx++;  
34        }  
35        return idx;  
36    }  
37    public class IndexedWord {  
38        String str;  
39        int index;  
40        IndexedWord(String str, int index) {  
41            this.str = str;  
42            this.index = index;  
43        }  
44    }  
45}
```

## 531. 孤独的像素 I

### 531. 孤独像素 I

难度 中等 击 7 喜欢 20 复制

给定一幅黑白像素组成的图像，计算黑色孤独像素的数量。

图像由一个由'B'和'W'组成二维字符数组表示，'B'和'W'分别代表黑色像素和白色像素。

黑色孤独像素指的是在同一行和同一列不存在其他黑色像素的黑色像素。

示例:

输入:  
[['W', 'W', 'B'],  
 ['W', 'B', 'W'],  
 ['B', 'W', 'W']]

输出: 3

解析: 全部三个'B'都是黑色孤独像素。

注意:

1. 输入二维数组行和列的范围是 [1,500]。

通过次数 1,213 提交次数 1,848

在真实的面试中遇到过这道题?

贡献者

## 533. 孤独的像素 II

### 533. 孤独像素 II

难度 中等 击 6 喜欢 20 复制

给定一幅由黑色像素和白色像素组成的图像，与一个正整数N，找到位于某行 R 和某列 C 中且符合下列规则的黑色像素的数量:

1. 行R 和列C都恰好包括N个黑色像素。
2. 列C中所有黑色像素所在的行必须和行R完全相同。

图像由一个由'B'和'W'组成二维字符数组表示，'B'和'W'分别代表黑色像素和白色像素。

示例:

输入:  
[['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

输出: 6

解析: 所有粗体的'B'都是我们所求的像素(第1列和第3列的所有'B')。

0 1 2 3 4 5 列号  
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]]  
行号

以R = 0行和C = 1列的'B'为例:

规则 1, R = 0行和C = 1列都恰好有N = 3个黑色像素。  
规则 2, 在C = 1列的黑色像素分别位于0, 1和2行。它们都和R = 0行完全相同。

```
1 class Solution {
2     public int findLonelyPixel(char[][] picture) {
3         int rowsCnt = 0, colsCnt = 0;
4         // 遍历每一行，统计符合要求的行数
5         int rows = picture.length, cols = picture[0].length;
6         for (int rid = 0; rid < rows; rid++) {
7             boolean blankExisted = false;
8             for (int cid = 0; cid < cols; cid++) {
9                 if (picture[rid][cid] == 'B') {
10                     if (blankExisted) {
11                         blankExisted = false;
12                         break;
13                     }
14                     blankExisted = true;
15                 }
16             }
17             if (blankExisted) {
18                 rowsCnt++;
19             }
20         }
21         // 遍历每一列，统计符合要求的列数
22         for (int cid = 0; cid < cols; cid++) {
23             boolean blankExisted = false;
24             for (int rid = 0; rid < rows; rid++) {
25                 if (picture[rid][cid] == 'B') {
26                     if (blankExisted) {
27                         blankExisted = false;
28                         break;
29                     }
30                     blankExisted = true;
31                 }
32             }
33             if (blankExisted) {
34                 colsCnt++;
35             }
36         }
37         // 返回符合要求的行数、列数的较小的那个值
38         return Math.min(colsCnt, rowsCnt);
39     }
40 }
```

```
1 class Solution {
2     public int findBlackPixel(char[][] picture, int N) {
3         if (N == 0) return 0;
4         int[][] rGrid = new int[picture.length][N + 1];
5         int[][] cGrid = new int[picture[0].length][N + 1];
6         for (int rid = 0; rid < picture.length; rid++) {
7             for (int cid = 0; cid < picture[0].length; cid++) {
8                 if (picture[rid][cid] == 'B') {
9                     if (++rGrid[rid][0] <= N) rGrid[rid][rGrid[rid][0]] = cid;
10                    if (++cGrid[cid][0] <= N) cGrid[cid][cGrid[cid][0]] = rid;
11                }
12            }
13        }
14        int ans = 0;
15        for (int rid = 0; rid < rGrid.length; rid++) {
16            if (rGrid[rid][0] == N) {
17                boolean same = true;
18                int colIndex = -1;
19                for (int cid = 1; cid <= N; cid++) {
20                    if (cGrid[rGrid[rid][0]][cid] == N) {
21                        colIndex = cid;
22                        break;
23                    }
24                }
25                if (colIndex == -1) continue;
26                for (int i = 1; i < N && same; i++) {
27                    if (cGrid[cGrid[rGrid[rid][0]][cid]][i] != rGrid[rid][i]) {
28                        same = false;
29                    }
30                }
31            }
32            if (same) {
33                for (int cid = 1; cid <= N; cid++) {
34                    if (cGrid[cGrid[rGrid[rid][0]][cid]][cid] == N) {
35                        ans += N;
36                    }
37                }
38                rGrid[cGrid[rGrid[rid][0]][cid]][0] = 0;
39            }
40        }
41    }
42    return ans;
43 }
44 }
45 }
```

## 532. 数组中的 K-diff 数对

### 532. 数组中的K-diff数对

难度 简单 击 77 喜欢 20 收藏 4

给定一个整数数组和一个整数  $k$ ，你需要在数组里找到不同的  $k$ -diff 数对。这里将  $k$ -diff 数对定义为一个整数对  $(i, j)$ ，其中  $i$  和  $j$  都是数组中的数字，且两数之差的绝对值是  $k$ 。

示例 1：

```
输入: [3, 1, 4, 1, 5], k = 2
输出: 2
解释: 数组中有两个 2-diff 数对, (1, 3) 和 (3, 5)。
尽管数组中有两个1, 但我们只应返回不同的数对的数量。
```

```
1 class Solution {
2     public int findPairs(int[] nums, int k) {
3         if (k < 0) {
4             return 0;
5         }
6         Set<Integer> set = new HashSet<>();
7         Set<Integer> diff = new HashSet<>();
8         for (int num : nums) {
9             if (set.contains(num - k)) {
10                diff.add(num - k);
11            }
12            if (set.contains(num + k)) {
13                diff.add(num);
14            }
15            set.add(num);
16        }
17     }
18 }
```

## 536. 从字符串生成二叉树

### 536. 从字符串生成二叉树

难度 中等 击 18 喜欢 20 收藏 4

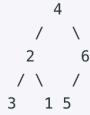
你需要从一个包括括号和整数的字符串构建一棵二叉树。

输入的字符串代表一棵二叉树。它包括整数和随后的0, 1或2对括号。整数代表根的值，一对括号内表示同样结构的子树。

若存在左子结点，则从左子结点开始构建。

示例：

```
输入: "4(2(3)(1))(6(5))"
输出: 返回代表下列二叉树的根节点:
```



注意：

1. 输入字符串中只包含 '(', ')'，'-' 和 '0' ~ '9'
2. 空树由 "" 而非 "()" 表示。

通过次数 1,339 | 提交次数 2,574

在真实的面试中遇到过这道题？

```
1 class Solution {
2     public TreeNode str2tree(String s) {
3         if (s == null || s.isEmpty()) return null;
4         TreeNode root = new TreeNode(0);
5         dfs(s, root, 0);
6         return root;
7     }
8     private int dfs(String s, TreeNode root, int idx) {
9         int id;
10    boolean isNegative = false;
11    for (id = idx; id < s.length(); id++) {
12        char ch = s.charAt(id);
13        if (ch == '(') {
14            TreeNode node = new TreeNode(0);
15            if (root.left == null) root.left = node;
16            else root.right = node;
17            id = dfs(s, node, id + 1);
18        } else if (ch == ')') {
19            break;
20        } else if (ch == '-') {
21            isNegative = true;
22        } else {
23            if (isNegative) {
24                root.val = -(ch - '0');
25                isNegative = false;
26            } else {
27                if (root.val < 0) {
28                    root.val = root.val * 10 - (ch - '0');
29                } else {
30                    root.val = root.val * 10 + ch - '0';
31                }
32            }
33        }
34    }
35    return id;
36 }
37 }
```

## 544. 输出比赛匹配对

### 544. 输出比赛匹配对

难度 中等 击 13 喜欢 20 收藏 4

在 NBA 季后赛中，我们总是安排较强的队伍对战较弱的队伍，例如用排名第 1 的队伍和第  $n$  的队伍对决，这是一个可以让比赛更加有趣的好策略。现在，给你  $n$  支队伍，你需要以字符串格式输出它们的最终比赛配对。

$n$  支队伍按从 1 到  $n$  的正整数格式给出，分别代表它们的初始排名（排名 1 最强，排名  $n$  最弱）。我们用括号  $(' ', ' )'$  和逗号  $(', )'$  来表示匹配对——括号  $(' ', ' )'$  表示匹配，逗号  $(', )'$  来用于分割。在每一轮的匹配过程中，你都需要遵循将强队与弱队配对的原则。

示例 1：

```
输入: 2
输出: (1,2)
解析:
初始地，我们有队1和队2两支队伍，按照1, 2排列。
因此用'('，')' 和 ',' 来将队1和队2进行配对，得到最终答案。
```

```
1 class Solution {
2     private static final int INIT = 1;
3     private static final int HAVF = 2;
4     private static final int INIT_SUM = 2 * INIT + 1;
5
6     public String findContestMatch(int n) {
7         StringBuilder buffer = new StringBuilder();
8         dsf(INIT, INIT_SUM, n, buffer);
9         return buffer.toString();
10    }
11
12    void dsf(int i, int sum, int n, StringBuilder buffer) {
13        if (n == INIT) {
14            buffer.append(i);
15            return;
16        }
17        buffer.append('(');
18        dsf(i, HAVF * sum - 1, n / HAVF, buffer);
19        buffer.append(',');
20        dsf(sum - i, HAVF * sum - 1, n / HAVF, buffer);
21        buffer.append(')');
22    }
23
24 }
25 }
```

## 545. 二叉树的边界

545. 二叉树的边界

难度 中等 ⚡ 15 ❤️ 🌟 💬 🔍

给定一棵二叉树，以逆时针顺序从根开始返回其边界。边界按顺序包括左边界、叶子结点和右边界而不包括重复的结点。（结点的值可能重复）

左边界 的定义是从根到最左侧结点的路径。右边界的定义是从根到最右侧结点的路径。若根没有左子树或右子树，则根本身就是左边界或右边界。注意该定义只对输入的二叉树有效，而对子树无效。

最左侧结点的定义是：在左子树存在时总是优先访问，如果不存在左子树则访问右子树。重复以上操作，首先抵达的结点就是最左侧结点。

最右侧结点的定义方式相同，只是将左替换为右。

示例 1

输入：  
1  
  \  
  2  
  / \  
  3  4  
  
输出：  
[1, 3, 4, 2]

```
1 class Solution {  
2     public List<Integer> boundaryOfBinaryTree(TreeNode root) {  
3         List<Integer> ans = new ArrayList<>();  
4         dfs(root, true, true, ans);  
5         return ans;  
6     }  
7  
8     private void dfs(TreeNode node, boolean isLeftBound, boolean isRightBound, List<Integer> list) {  
9         if (node == null) {  
10             return;  
11         }  
12         if (isLeftBound) {  
13             list.add(node.val);  
14         } else if (node.left == null && node.right == null) {  
15             list.add(node.val);  
16             return;  
17         }  
18         dfs(node.left, isLeftBound, !isLeftBound && isRightBound && node.right == null, list);  
19         dfs(node.right, !isRightBound && isLeftBound && node.left == null, isRightBound, list);  
20         if (isLeftBound || isRightBound) {  
21             return;  
22         }  
23         list.add(node.val);  
24     }  
25 }  
26  
27  
28  
29 }
```

## 548. 将数组分割成相等的子数组

548. 将数组分割成相等的子数组

难度 中等 ⚡ 12 ❤️ 🌟 💬 🔍

给定一个有 n 个整数的数组，你需要找到满足以下条件的三元组 (i, j, k)：

1.  $0 < i + 1 < j, j + 1 < k < n - 1$
2. 子数组  $(0, i - 1), (i + 1, j - 1), (j + 1, k - 1), (k + 1, n - 1)$  的和应该相等。

这里我们定义子数组  $(L, R)$  表示原数组从索引为 L 的元素开始至索引为 R 的元素。

示例：

输入： [1,2,1,2,1,2,1]  
输出： True  
解释：  
i = 1, j = 3, k = 5。  
 $\text{sum}(0, i - 1) = \text{sum}(0, 0) = 1$   
 $\text{sum}(i + 1, j - 1) = \text{sum}(2, 2) = 1$   
 $\text{sum}(j + 1, k - 1) = \text{sum}(4, 4) = 1$   
 $\text{sum}(k + 1, n - 1) = \text{sum}(6, 6) = 1$

注意：

1.  $1 \leq n \leq 2000$ 。
2. 给定数组中的元素会在  $[-1,000,000, 1,000,000]$  范围内。

```
1 class Solution {  
2     public boolean splitArray(int[] nums) {  
3         if (nums == null || nums.length < 7) {  
4             return false;  
5         }  
6         int max = Integer.MIN_VALUE;  
7         int min = Integer.MAX_VALUE;  
8         int[] sums = new int[nums.length + 1];  
9         for (int i = 1; i <= nums.length; i++) {  
10             sums[i] = sums[i - 1] + nums[i - 1];  
11             max = Math.max(nums[i - 1], max);  
12             min = Math.min(nums[i - 1], min);  
13         }  
14         int len = nums.length;  
15         int diff = max - min;  
16         for (int i = 3; i < nums.length - 3; i++) {  
17             int abs = Math.abs(sums[len] - sums[i + 1] - sums[i]);  
18             if (abs > diff)  
19                 continue;  
20             for (int x = 1; x < nums.length - 5; x++) {  
21                 int sum = sums[x];  
22                 int dif = sums[i] - sums[x + 1];  
23                 if (sum != dif)  
24                     continue;  
25                 for (int z = nums.length - 1; z >= i + 2; z--) {  
26                     int dif1 = sums[z] - sums[i + 1];  
27                     int dif2 = sums[len] - sums[z + 1];  
28                     if (dif == dif1 && dif1 == dif2) {  
29                         return true;  
30                     }  
31                 }  
32             }  
33         }  
34     }  
35 }
```

## 549. 二叉树最长的连续序列

549. 二叉树中最长的连续序列

难度 中等 ⚡ 18 ❤️ 🌟 💬 🔍

给定一个二叉树，你需要找出二叉树中最长的连续序列路径的长度。

请注意，该路径可以是递增的或者递减的。例如，[1,2,3,4] 和 [4,3,2,1] 都被认为是合法的，而路径 [1,2,4,3] 则不合法。另一方面，路径可以是子-父-子顺序，并不一定是父-子顺序。

示例 1：

输入：  
1  
  \  
  2  3  
  
输出： 2  
解释： 最长的连续路径是 [1, 2] 或者 [2, 1]。

示例 2：

输入：  
2  
  \  
  1  3  
  
输出： 3  
解释： 最长的连续路径是 [1, 2, 3] 或者 [3, 2, 1]。

```
1 class Solution {  
2     public int longestConsecutive(TreeNode root) {  
3         int[] max = new int[1];  
4         dfs(root, max);  
5         return max[0];  
6     }  
7  
8     private int[] dfs(TreeNode node, int[] max) {  
9         if (node == null) {  
10             return new int[]{0, 0};  
11         }  
12         int incLen = 1;  
13         int decLen = 1;  
14         if (node.left != null) {  
15             int[] leftLens = dfs(node.left, max);  
16             if (node.left.val == node.val + 1) {  
17                 incLen = leftLens[0] + 1;  
18             } else if (node.left.val == node.val - 1) {  
19                 decLen = leftLens[1] + 1;  
20             }  
21         }  
22         if (node.right != null) {  
23             int[] rightLens = dfs(node.right, max);  
24             if (node.right.val == node.val + 1) {  
25                 incLen = Math.max(incLen, rightLens[0] + 1);  
26             } else if (node.right.val == node.val - 1) {  
27                 decLen = Math.max(decLen, rightLens[1] + 1);  
28             }  
29         }  
30         max[0] = Math.max(max[0], incLen + decLen - 1);  
31         return new int[]{incLen, decLen};  
32     }  
33 }
```

## 555. 分割连接字符串

555. 分割连接字符串

难度 中等 ⚡ 12 ❤️ 🔍 🌐

给定一个字符串列表，你可以将这些字符串连接成一个循环字符串，对于每个字符串，你可以选择是否翻转它。在所有可能的循环字符串中，你需要分割循环字符串（这将使循环字符串变成一个常规的字符串），然后找到字典序最大的字符串。

具体来说，要找到字典序最大的字符串，你需要经历两个阶段：

1. 将所有字符串连接成一个循环字符串，你可以选择是否翻转某些字符串，并按照给定的顺序连接它们。
2. 在循环字符串的某个位置分割它，这将使循环字符串从分割点变成一个常规的字符串。

你的工作是在所有可能的常规字符串中找到字典序最大的一个。

示例：

```
输入: "abc", "xyz"
输出: "zyxcba"
解释: 你可以得到循环字符串 "-abcyxz-", "-abczyx-", "-cbaxyz-", "-cbazyx-",
其中 '-' 代表循环状态。
答案字符串来自第四个循环字符串。
你可以从中间字符 'a' 分割开然后得到 "zyxcba"。
```

注意：

1. 输入字符串只包含小写字母。
2. 所有字符串的总长度不会超过 1,000。

```
1+ class Solution {
2+     public String splitLoopedString(String[] strsArr) {
3+         String ans = new String();
4+         String loop = new String();
5+         int size = strsArr.length;
6+         for (int i = 0; i < size; i++) {
7+             String str = strsArr[i];
8+             String rvs = (new StringBuffer(str)).reverse().toString();
9+             if (str.compareTo(rvs) < 0) {
10+                 loop += rvs;
11+             } else {
12+                 loop += str;
13+             }
14+         }
15+         int sumLen = 0;
16+         for (int i = 0; i < size; i++) {
17+             int len = strsArr[i].length();
18+             String rLoop = loop.substring(0, sumLen) +
19+                 (new StringBuffer(loop.substring(sumLen, sumLen + len))).reverse().toString() +
20+                 loop.substring(sumLen + len);
21+             for (int j = 0; j < len; j++) {
22+                 String splitLoop;
23+                 splitLoop = loop.substring(sumLen + j) + loop.substring(0, sumLen + j);
24+                 if (ans.compareTo(splitLoop) < 0) {
25+                     ans = splitLoop;
26+                 }
27+                 splitLoop = rLoop.substring(sumLen + j) + rLoop.substring(0, sumLen + j);
28+                 if (ans.compareTo(splitLoop) < 0) {
29+                     ans = splitLoop;
30+                 }
31+             }
32+             sumLen += len;
33+         }
34+         return ans;
35+     }
36+ }
```

## 562. 矩阵中最长的连续 1 线段

562. 矩阵中最长的连续 1 线段

难度 中等 ⚡ 16 ❤️ 🔍 🌐

给定一个01矩阵  $M$ ，找到矩阵中最长的连续1线段。这条线段可以是水平的、垂直的、对角线的或者反对角线的。

示例：

```
输入:
[[0,1,1,0],
 [0,1,1,0],
 [0,0,0,1]]
输出: 3
```

提示：给定矩阵中的元素数量不会超过 10,000。

通过次数 1,702 | 提交次数 4,205

在真实的面试中遇到过这道题？

```
1+ class Solution {
2+     public int longestLine(int[][] M) {
3+         if (M == null || M.length == 0 || M[0].length == 0) return 0;
4+         if (ans == 0; int[] dp = new int[M[0].length][4];
5+         for (int rid = 0; rid < M.length; rid++) {
6+             int prev = 0;
7+             for (int cid = 0; cid < M[0].length; cid++) {
8+                 if (M[rid][cid] == 1) {
9+                     if (M[rid][cid - 1] == 1) {
10+                         dp[cid][0] = cid - 1[0] + 1 : 1;
11+                         dp[cid][1] = rid - 1[0] ? dp[cid][1] + 1 : 1;
12+                         int curr = dp[cid][2];
13+                         dp[cid][2] = (rid > 0 && cid > 0) ? prev + 1 : 1;
14+                         prev = curr;
15+                         dp[cid][3] = (rid > 0 && cid < M[0].length - 1) ? dp[cid + 1][3] + 1 : 1;
16+                         int max = Math.max(Math.max(dp[cid][0], dp[cid][1]), Math.max(dp[cid][2], dp[cid][3]));
17+                         ans = Math.max(ans, max);
18+                     } else {
19+                         prev = dp[cid][2];
20+                         dp[cid][0] = dp[cid][1] = dp[cid][2] = dp[cid][3] = 0;
21+                     }
22+                 }
23+             }
24+         }
25+         return ans;
26+     }
27+ }
```

## 568. 最大休假天数

568. 最大休假天数

难度 困难 ⚡ 10 ❤️ 🔍 🌐

力扣想让一个最优秀的员工在  $N$  个城市间旅行来收集算法问题。但只工作不玩耍，聪明的孩子也会变傻，所以您可以在某些特定的城市和星期休假。您的工作就是安排旅行使得最大化您可以休假的天数，但是您需要遵守一些规则和限制。

规则和限制：

1. 您只能在  $N$  个城市之间旅行，用 0 到  $N-1$  的索引表示。一开始，您在索引 0 的城市，並且那天是星期一。
2. 这些城市通过航班相连。这些航班用  $N \times N$  矩阵  $\text{flights}$ （不一定是对称的）表示。 $\text{flights}[i][j]$  代表城市*i*到城市*j*的航空状态。如果没有城市*i*到城市的航班， $\text{flights}[i][j] = 0$ ；否则， $\text{flights}[i][j] = 1$ 。同时，对于所有的*i*， $\text{flights}[i][i] = 0$ 。
3. 您总共有  $K$  周（每周7天）的时间旅行。您每天最多只能乘坐一次航班，并且只能在每周的星期一上午乘坐航班。由于飞行时间很短，我们不考虑飞行时间的影响。
4. 对于每个城市，不同的星期您休假天数是不同的，给定一个  $N \times K$  矩阵  $\text{days}$  代表这种限制， $\text{days}[i][j]$  代表您在第*j*个星期在城市*i*能休假的最长天数。

给定  $\text{flights}$  矩阵和  $\text{days}$  矩阵，您需要输出  $K$  周内可以休假的最长天数。

```
1+ class Solution {
2+     public int maxVacationDays(int[][] flights, int[][] days) {
3+         int fLen = flights.length;
4+         int dLen = days[0].length;
5+         int[] dp = new int[fLen];
6+         for (int i = dLen - 1; i >= 0; i--) {
7+             int[] tmp = new int[fLen];
8+             for (int x = 0; x < fLen; x++) {
9+                 tmp[x] = days[x][i];
10+                int max = dp[x];
11+                for (int n = 0; n < fLen; n++) {
12+                    if (flights[x][n] == 1) {
13+                        max = Math.max(max, dp[n]);
14+                    }
15+                }
16+                tmp[x] += max;
17+            }
18+            dp = tmp;
19+        }
20+        int max = dp[0];
21+        for (int i = 0; i < fLen; i++) {
22+            if (flights[0][i] == 1) {
23+                max = Math.max(max, dp[i]);
24+            }
25+        }
26+        return max;
27+    }
28+ }
```



## 604. 迭代压缩字符串

604. 迭代压缩字符串

难度 简单 ⚡ 13 ❤ 14 🌟 15 🔍 16

对于一个压缩字符串，设计一个数据结构，它支持如下两种操作：`next()` 和 `hasNext()`。

给定的压缩字符串格式为：每个字母后面紧跟一个正整数，这个整数表示该字母在解压后的字符串里连续出现的次数。

`next()` - 如果压缩字符串仍然有字母未被解压，则返回下一个字母，否则返回一个空格。  
`hasNext()` - 判断是否还有字母仍然没被解压。

注意：

请记得将你的类在 `StringIterator` 中 初始化，因为静态变量或类变量在多组测试数据中不会被自动清空。更多细节请访问 [这里](#)。

示例：

```
StringIterator iterator = new StringIterator("L1e2t1C1o1d1e1");
iterator.next(); // 返回 'L'
iterator.next(); // 返回 'e'
iterator.next(); // 返回 'e'
```

```
1 class StringIterator {
2     private String mString;
3     private int mCurrentIdx = 0;
4     private int mCurrentNum = 0;
5     private char mCurrentCh = ' ';
6
7     public StringIterator(String s) {
8         this.mString = s;
9     }
10
11    public char next() {
12        if (!hasNext())
13            return ' ';
14        if (mCurrentNum == 0) {
15            mCurrentCh = mString.charAt(mCurrentIdx++);
16            while (mCurrentIdx < mString.length()
17                  && Character.isDigit(mString.charAt(mCurrentIdx)))
18                mCurrentNum = mCurrentNum * 10 + mString.charAt(mCurrentIdx++) - '0';
19        }
20        mCurrentNum--;
21        return mCurrentCh;
22    }
23
24    public boolean hasNext() {
25        return mCurrentIdx != mString.length() || mCurrentNum != 0;
26    }
27 }
```

## 616. 个字符串添加加粗标签

616. 给字符串添加加粗标签

难度 中等 ⚡ 24 ❤ 14 🌟 15 🔍 16

给一个字符串 `s` 和一个字符串列表 `dict`，你需要将在字符串列表中出现过的 `s` 的子串添加加粗闭合标签 `<b>` 和 `</b>`。如果两个子串有重叠部分，你需要把它们一起用一个闭合标签包围起来。同理，如果两个子字符串连续被加粗，那么你也需要把它们合起来用一个加粗标签包围。

样例 1：

```
输入:
s = "abcxyz123"
dict = ["abc", "123"]
输出:
<b>abc</b>xyz<b>123</b>"
```

样例 2：

```
输入:
s = "aaabbcc"
dict = ["aaa", "aab", "bc"]
输出:
<b>aaabb</b>c"
```

```
1 class Solution {
2     public String addBoldTag(String s, String[] dict) {
3         boolean[] bold = new boolean[s.length()];
4         for (String word : dict) {
5             int idx = -1;
6             while ((idx = s.indexOf(word, idx + 1)) != -1) {
7                 for (int j = idx; j < idx + word.length(); j++) {
8                     bold[j] = true;
9                 }
10            }
11        }
12        boolean marked = false;
13        StringBuilder buffer = new StringBuilder();
14        for (int idx = 0; idx < s.length(); idx++) {
15            if (!marked && !bold[idx]) {
16                buffer.append("<b>");
17                marked = true;
18            } else if (marked && !bold[idx]) {
19                buffer.append("</b>");
20                marked = false;
21            }
22            buffer.append(s.charAt(idx));
23        }
24        if (marked) {
25            buffer.append("</b>");
26        }
27    }
28 }
29 }
```

## 624. 数组列表中的最大距离

624. 数组列表中的最大距离

难度 简单 ⚡ 17 ❤ 14 🌟 15 🔍 16

给定 `m` 个数组，每个数组都已经按照升序排好序了。现在你需要从两个不同的数组中选择两个整数（每个数组选一个）并且计算它们的距离。两个整数 `a` 和 `b` 之间的距离定义为它们差的绝对值  $|a-b|$ 。你的任务就是去找最大距离。

示例 1：

```
输入:
[[1,2,3],
 [4,5],
 [1,2,3]]
输出: 4
解释:
一种得到答案 4 的方法是从第一个数组或者第三个数组中选择 1，同时从第二个数组中选择 5。
```

```
1 class Solution {
2     public int maxDistance(List<List<Integer>> arrays) {
3         int ans = 0;
4         int size = arrays.size();
5         List<Integer> list = arrays.get(0);
6         int min = list.get(0);
7         int max = list.get(list.size() - 1);
8         for (int i = 1; i < size; ++i) {
9             list = arrays.get(i);
10            int len = list.size();
11            int curMin = list.get(0);
12            int curMax = list.get(len - 1);
13            ans = max(curMax - min, ans, max - curMin);
14            max = Math.max(curMax, max);
15            min = Math.min(curMin, min);
16        }
17        return ans;
18    }
19
20    private int max(int a, int b, int c) {
21        return Math.max(Math.max(a, b), c);
22    }
23 }
```

## 625. 最小因式分解

625. 最小因式分解

难度 中等 ⚡ 12 ❤️ ⚡ 🌟 🌟 🌟 🌟 🌟

给定一个正整数 `a`，找出最小的正整数 `b` 使得 `b` 的所有数位相乘恰好等于 `a`。

如果不存在这样的结果或者结果不是 32 位有符号整数，返回 0。

样例 1

输入：

48      输出: 68

```
1 public class Solution {
2     public int smallestFactorization(int a) {
3         if (a < 10)
4             return a;
5         int tmp = a;
6         long ans = 0, mul = 1;
7         for (int num = 9; num > 1; num--) {
8             while (tmp % num == 0) {
9                 tmp /= num;
10                ans = mul * num + ans;
11                mul *= 10;
12            }
13        }
14        return tmp < 2 && ans <= Integer.MAX_VALUE ? (int) ans : 0;
15    }
16 }
```

## 632. 设计 Excel 求和公式

631. 设计 Excel 求和公式

难度 困难 ⚡ 10 ❤️ ⚡ 🌟 🌟 🌟 🌟 🌟

你的任务是实现 Excel 的求和功能，具体的操作如下：

`Excel(int H, char W)`：这是一个构造函数，输入表明了 Excel 的高度和宽度。H 是一个正整数，范围从 1 到 26，代表高度。W 是一个字符，范围从 'A' 到 'Z'，宽度等于从 'A' 到 W 的字母个数。Excel 表格是一个高度 \* 宽度的二维整数数组，数组中元素初始化为 0。第一行下标从 1 开始，第一列下标从 'A' 开始。

`void Set(int row, char column, int val)`：设置 `C(row, column)` 中的值为 `val`。

`int Get(int row, char column)`：返回 `C(row, column)` 中的值。

`int Sum(int row, char column, List of Strings : numbers)`：这个函数会将计算的结果放入 `C(row, column)` 中，计算的结果等于在 `numbers` 中代表的所有元素之和，这个函数同时也会将这个结果返回。求和公式会一直计算更新结果直到这个公式被其他的值或者公式覆盖。

`numbers` 是若干字符串的集合，每个字符串代表单个位置或一个区间。  
如果这个字符串表示单个位置，它的格式如下：`ColRow`，例如 "F7" 表示位置 (7, F)。  
如果这个字符串表示一个区间，它的格式如下：`ColRow1:ColRow2`。区间就是左上角为 `ColRow1` 右下角为 `ColRow2` 的长方形。

样例 1：

```
Excel(3,"C");
// 构造一个 3x3 的二维数组，初始化全是 0。
// A B C
// 1 0 0 0
```

```
1 class Excel {
2     private int mWidth, mHeight;
3     private int[][] mDatas;
4     private String[][] mFunctions;
5     public Excel(int H, char W) {
6         this.mHeight = H;
7         this.mWidth = (W - 'A') + 1;
8         this.mData = new int[mHeight][mWidth];
9         this.mFunctions = new String[mHeight][mWidth];
10    }
11    public void set(int r, char c, int v) {
12        mDatas[r - 1][c - 'A'] = v;
13        mFunctions[r - 1][c - 'A'] = "";
14    }
15    public int get(int r, char c) {
16        String function = mFunctions[r - 1][c - 'A'];
17        if (function == null || function.isEmpty())
18            return mDatas[r - 1][c - 'A'];
19        int ans = 0;
20        String[] attributes = function.split(",");
21        for (String attribute : attributes) {
22            int idx = attribute.indexOf(":");
23            if (idx == -1) {
24                int nextRow = Integer.parseInt(attribute.substring(1));
25                ans += get(nextRow, attribute.charAt(0));
26            } else {
27                String bAtt = attribute.substring(0, idx), eAtt = attribute.substring(idx + 1);
28                char bChar = bAtt.charAt(0), eChar = eAtt.charAt(0);
29                int bRow = Integer.parseInt(bAtt.substring(1)), eRow = Integer.parseInt(eAtt.substring(1));
30                for (int i = bRow - 1; i <= eRow - 1; i++)
31                    for (int x = bChar - 'A'; x <= eChar - 'A'; x++)
32                        ans += get(i + 1, (char) ('A' + x));
33            }
34        }
35        return ans;
36    }
37    public int sum(int r, char c, String[] strs) {
38        StringBuilder buffer = new StringBuilder();
39        for (int idx = 0; idx < strs.length; idx++) {
40            if (idx > 0) buffer.append(',');
41            buffer.append(strs[idx]);
42        }
43        mFunctions[r - 1][c - 'A'] = buffer.toString();
44        return get(r, c);
45    }
46 }
```

## 634. 寻找数组的错位排列

634. 寻找数组的错位排列

难度 中等 ⚡ 10 ❤️ ⚡ 🌟 🌟 🌟 🌟 🌟

在组合数学中，如果一个排列中所有元素都不在原先的位置上，那么这个排列就被称为错位排列。

给定一个从 1 到 `n` 升序排列的数组，你可以计算出总共有多少个不同的错位排列吗？

由于答案可能非常大，你只需要将答案对  $10^9 + 7$  取余输出即可。

```
1 class Solution {
2     public int findDerangement(int n) {
3         if (n == 1 || n == 2) return n - 1;
4         final int mod = 1000000007;
5         long prev = 0, curr = 1;
6         for (int i = 3; i <= n; i++) {
7             long tmp = ((i - 1) * (prev + curr)) % mod;
8             prev = curr;
9             curr = tmp;
10        }
11        return (int) curr;
12    }
13 }
```

## 635. 设计日志存储系统

635. 设计日志存储系统

难度 中等 ⚡ 18 ❤ 14 🌟 14 🔍 14

你将获得多条日志，每条日志都有唯一的 id 和 timestamp，timestamp 是形如 Year:Month:Day:Hour:Minute:Second 的字符串，例如 2017:01:01:23:59:59，所有值域都是零填充的十进制数。

设计一个日志存储系统实现如下功能：

void Put(int id, string timestamp)：给定日志的 id 和 timestamp，将这个日志存入你的存储系统中。

```
int[] Retrieve(String start, String end, String granularity)：返回在给定时间区间内的所有日志的 id。start、end 和 timestamp 的格式相同，granularity 表示考虑的时间级。比如，start = "2017:01:01:23:59:59", end = "2017:01:02:23:59:59", granularity = "Day" 代表区间 2017 年 1 月 1 日到 2017 年 1 月 2 日。
```

样例 1：

```
put(1, "2017:01:01:23:59:59");
put(2, "2017:01:01:22:59:59");
put(3, "2016:01:01:00:00:00");
retrieve("2016:01:01:01:01", "2017:01:01:23:00:00");
// 返回值 [1,2,3]，返回从 2016 年到 2017 年所有的日志。
```

## 642. 设计搜索自动补全系统

642. 设计搜索自动补全系统

难度 困难 ⚡ 24 ❤ 14 🌟 14 🔍 14

为搜索引擎设计一个搜索自动补全系统。用户会输入一条语句（最少包含一个字母，以特殊字符 '#' 结尾）。除 '#' 以外用户输入的每个字符，返回历史中热度前三并以当前输入部分为前缀的句子。下面是详细规则：

1. 一条句子的热度定义为历史上用户输入这个句子的总次数。
2. 返回前三的句子需要按照热度从高到低排序（第一个是最热门的）。如果有多个热度相同的句子，请按照 ASCII 码的顺序输出（ASCII 码越小排名越前）。
3. 如果满足条件的句子个数少于 3，将它们全部输出。
4. 如果输入了特殊字符，意味着句子结束了，请返回一个空集合。

你的工作是实现以下功能：

构造函数：

```
AutocompleteSystem(String[] sentences, int[] times); 这是构造函数，输入的是历史数据。Sentences 是之前输入过的所有句子，Times 是每条句子输入的次数，你的系统需要记录这些历史信息。
```

现在，用户输入一条新的句子，下面的函数会提供用户输入的下一个字符：

```
List<String> input(char c); 其中 c 是用户输入的下一个字符。字符只会是小写英文字母（'a' 到 'z'），空格（' '）和特殊字符（'#'）。输出历史热度前三的具有相同前缀的句子。
```

样例：

操作： AutocompleteSystem(["i love you", "island", "ironman", "i love leetcode"], [5,3,2,1])

系统记录下所有的句子和出现的次数：

```
"i love you" : 5 次
"island" : 3 次
"ironman" : 2 次
"i love leetcode" : 2 次
```

现在，用户开始新的键入：

```
输入： input('i')
输出： ["i love you", "island", "i love leetcode"]
解释：
```

```
1 class LogSystem {
2     private Map<Integer, String> mMap;
3     private Map<String, Integer> mDic;
4
5     public LogSystem() {
6         this.mMap = new HashMap<>();
7         this.mDic = new HashMap<>();
8         this.mDic.put("Year", 4);
9         this.mDic.put("Month", 7);
10        this.mDic.put("Day", 10);
11        this.mDic.put("Hour", 13);
12        this.mDic.put("Minute", 16);
13        this.mDic.put("Second", 19);
14    }
15
16    public void put(int id, String timestamp) {
17        mMap.put(id, timestamp);
18    }
19
20    public List<Integer> retrieve(String s, String e, String gra) {
21        List<Integer> ans = new ArrayList<>();
22        Integer len = mDic.get(gra);
23        s = s.substring(0, len);
24        e = e.substring(0, len);
25        for (Integer key : mMap.keySet()) {
26            String time = mMap.get(key).substring(0, len);
27            if (time.compareTo(s) >= 0 && time.compareTo(e) <= 0) {
28                ans.add(key);
29            }
30        }
31        return ans;
32    }
33 }
```

```
1 class AutocompleteSystem {
2     private class TrieNode implements Comparable<TrieNode> {
3         int times; String word;
4         TrieNode[] children; List<TrieNode> hotNodes;
5         public TrieNode() {
6             this.times = 0; this.word = null;
7             this.children = new TrieNode[128];
8             this.hotNodes = new ArrayList<>();
9         }
10        public int compareTo(TrieNode o) {
11            if (this.times == o.times) return this.word.compareTo(o.word);
12            return o.times - this.times;
13        }
14        public void update(TrieNode node) {
15            if (!hotNodes.contains(node)) hotNodes.add(node);
16            Collections.sort(hotNodes);
17            if (hotNodes.size() > 3) hotNodes.remove(hotNodes.size() - 1);
18        }
19    }
20    TrieNode mRoot; TrieNode mCurr; StringBuilder mBuffer;
21    public AutocompleteSystem(String[] sentences, int[] times) {
22        mRoot = new TrieNode(); mCurr = mRoot; mBuffer = new StringBuilder();
23        for (int i = 0; i < times.length; i++) add(sentences[i], times[i]);
24    }
25    public void add(String sentence, int t) {
26        TrieNode root = mRoot; List<TrieNode> visited = new ArrayList<>();
27        for (char ch : sentence.toCharArray()) {
28            if (root.children[ch] == null) root.children[ch] = new TrieNode();
29            root = root.children[ch];
30            visited.add(root);
31        }
32        root.word = sentence; root.times += t;
33        for (TrieNode node : visited) node.update(root);
34    }
35    public List<String> input(char c) {
36        List<String> ans = new ArrayList<>();
37        if (c == '#') {
38            add(mBuffer.toString(), 1); mBuffer = new StringBuilder();
39            mCurr = mRoot; return ans;
40        }
41        mBuffer.append(c);
42        if (mCurr != null) mCurr = mCurr.children[c];
43        if (mCurr == null) return ans;
44        for (TrieNode node : mCurr.hotNodes) ans.add(node.word);
45    }
46    }
47 }
```

## 644. 最大平均子段和 II

### 644. 最大平均子段和 II

难度 困难 21 21 21 21 21 21

给定一个包含  $n$  个整数的数组，找到最大平均值的连续子序列，且长度大于等于  $k$ 。并输出这个最大平均值。

样例 1：

```
输入: [1,12,-5,-6,50,3], k = 4
输出: 12.75
解释:
当长度为 5 的时候, 最大平均值是 10.8,
当长度为 6 的时候, 最大平均值是 9.166667。
所以返回值是 12.75。
```

注释：

- 1  $\leq k \leq n \leq 10,000$ 。
- 数组中的元素范围是  $[-10,000, 10,000]$ 。
- 答案的计算误差小于  $10^{-5}$ 。

通过次数 373 | 提交次数 1,131

在真实的面试中遇到过这道题？

贡献者

```
1 class Solution {
2     public double findMaxAverage(int[] A, int K) {
3         double lid, rid, mid;
4         lid = rid = A[0];
5         for (int i = 0; i < A.length; i++) {
6             lid = Math.min(A[i], lid);
7             rid = Math.max(A[i], rid);
8         }
9         while (lid + 1e-5 < rid) {
10            mid = (lid + rid) / 2;
11            if (isValid(A, K, mid)) {
12                lid = mid;
13            } else {
14                rid = mid;
15            }
16        }
17        return lid;
18    }
19
20    private boolean isValid(int[] A, int K, double avg) {
21        double lSum = 0, rSum = 0, minSum = 0;
22        for (int i = 0; i < K; i++) {
23            rSum += A[i] - avg;
24        }
25        for (int i = K; i <= A.length; i++) {
26            if (rSum - minSum >= 0) {
27                return true;
28            }
29            if (i < A.length) {
30                rSum += A[i] - avg;
31                lSum += A[i - K] - avg;
32                minSum = Math.min(minSum, lSum);
33            }
34        }
35    }
36    return false;
37 }
```

## 651. 4键键盘

### 651. 4键键盘

难度 中等 20 20 20 20 20 20

假设你有一个特殊的键盘包含下面的按键：

- Key 1: (A) : 在屏幕上打印一个 'A'。
- Key 2: (Ctrl-A) : 选中整个屏幕。
- Key 3: (Ctrl-C) : 复制选中区域到缓冲区。
- Key 4: (Ctrl-V) : 将缓冲区内容输出到上次输入的结束位置，并显示在屏幕上。

现在，你只可以按键  $N$  次（使用上述四种按键），请问屏幕上最多可以显示几个 'A' 呢？

## 660. 移除 9

### 660. 移除 9

难度 困难 4 4 4 4 4 4

从 1 开始，移除所有包含数字 9 的所有整数，例如 9, 19, 29, .....

这样就获得了一个新的整数数列：1, 2, 3, 4, 5, 6, 7, 8, 10, 11, .....

给定正整数  $n$ ，请你返回新数列中第  $n$  个数字是多少。1 是新数列中的第一个数字。

```
1 class Solution {
2     public int maxAC(int N) {
3         if (N <= 5) {
4             return N;
5         }
6         int[] dp = new int[N];
7         dp[0] = 1;
8         dp[1] = 2;
9         dp[2] = 3;
10        dp[3] = 4;
11        dp[4] = 5;
12        for (int i = 5; i < N; ++i) {
13            dp[i] = Math.max(3 * dp[i - 4], 4 * dp[i - 5]);
14        }
15        return dp[N - 1];
16    }
17 }
```

```
1 class Solution {
2     public int newInteger(int n) {
3         // 移除9之后的剩余的数是9进制
4         return Integer.parseInt(Integer.toString(n, 9));
5     }
6 }
7
8
9
10 }
```

## 656. 金币路径

### 656. 金币路径

难度 困难 ⚡ 12 ❤️ ⌂ 🎮 ⌂

给定一个数组 A (下标从 1 开始) 包含 N 个整数: A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>N</sub> 和一个整数 B。你可以从数组 A 中的任何一个位置 (下标为 i) 跳到下标 i+1, i+2, ..., i+B 的任意一个可以跳到的位置上。如果你在下标为 i 的位置上, 你需要支付 A<sub>i</sub> 个金币。如果 A<sub>i</sub> 是 -1, 意味着下标为 i 的位置是不可以跳到的。

现在, 你希望花费最少的金币从数组 A 的 1 位置跳到 N 位置, 你需要输出花费最少的路径, 依次输出所有经过的下标 (从 1 到 N)。

如果有多种花费最少的方案, 输出字典顺序最小的路径。

如果无法到达 N 位置, 请返回一个空数组。

样例 1:

输入: [1,2,4,-1,2], 2  
输出: [1,3,5]

样例 2:

输入: [1,2,4,-1,2], 1  
输出: []

## 663. 均匀树划分

### 663. 均匀树划分

难度 中等 ⚡ 13 ❤️ ⌂ 🎮 ⌂

给定一棵有 n 个结点的二叉树, 你的任务是检查是否可以通过去掉树上的一条边将树分成两棵, 且这两棵树结点之和相等。

样例 1:

输入:  
5  
/\  
10 10  
/ \  
2 3  
  
输出: True  
解释:  
5  
/  
/  
10

```
1 class Solution {  
2     public List<Integer> cheapestJump(int[] A, int B) {  
3         List<Integer> ans = new ArrayList<>();  
4         int length = A.length;  
5         int[] dp = new int[length];  
6         Arrays.fill(dp, Integer.MAX_VALUE);  
7         if (A[length - 1] != -1)  
8             dp[length - 1] = A[length - 1];  
9         int[] next = new int[length];  
10        Arrays.fill(next, length);  
11        for (int i = length - 2; i >= 0; i--) {  
12            if (A[i] == -1)  
13                continue;  
14            int lid = i + 1;  
15            int rid = Math.min(i + B, length - 1);  
16            for (int x = lid; x <= rid; x++) {  
17                if (dp[x] == Integer.MAX_VALUE)  
18                    continue;  
19                if (dp[x] + A[i] < dp[i]) {  
20                    dp[i] = dp[x] + A[i];  
21                    next[i] = x;  
22                }  
23            }  
24        }  
25        if (dp[0] == Integer.MAX_VALUE)  
26            return ans;  
27        int idx = 0;  
28        while (idx < length) {  
29            ans.add(idx + 1);  
30            idx = next[idx];  
31        }  
32        return ans;  
33    }  
34}
```

```
1 class Solution {  
2     public boolean checkEqualTree(TreeNode root) {  
3         int sum = sum(root);  
4         if ((sum & 1) == 1) return false;  
5         int half = sum >> 1;  
6         boolean[] equal = new boolean[1];  
7         checkEqualTree(root.left, half, equal);  
8         checkEqualTree(root.right, half, equal);  
9         return equal[0];  
10    }  
11    private int sum(TreeNode node) {  
12        if (node == null) return 0;  
13        return node.val + sum(node.left) + sum(node.right);  
14    }  
15    private int checkEqualTree(TreeNode node, int target, boolean[] equal) {  
16        if (node == null || equal[0]) return 0;  
17        int sum = node.val;  
18        sum += checkEqualTree(node.left, target, equal);  
19        sum += checkEqualTree(node.right, target, equal);  
20        if (sum == target) equal[0] = true;  
21        return sum;  
22    }  
23}  
24
```

## 666. 路径和 IV

### 666. 路径和 IV

难度 中等 ⚡ 9 ❤️ ⌂ 🎮 ⌂

对于一棵深度小于 5 的树, 可以用一组三位十进制整数来表示。

对于每个整数:

1. 百位上的数字表示这个节点的深度 D, 1 <= D <= 4。
2. 十位上的数字表示这个节点在当前层所在的位置 P, 1 <= P <= B。  
B 位置编号与一棵满二叉树的位置编号相同。
3. 个位上的数字表示这个节点的权值 V, 0 <= V <= 9。

给定一个包含三位整数的升序 数组, 表示一棵深度小于 5 的二叉树, 请你返回从根到所有叶子结点的路径之和。

样例 1:

输入: [113, 215, 221]  
输出: 12  
解释:  
这棵树形状如下:  
3  
/\  
5 1  
  
路径和 = (3 + 5) + (3 + 1) = 12.

```
1 class Solution {  
2     public int pathSum(int[] nums) {  
3         if (nums == null) {  
4             return 0;  
5         }  
6         int[] ans = new int[1];  
7         Map<Integer, Integer> map = new HashMap<>();  
8         for (int num : nums) {  
9             map.put(num / 10, num % 10);  
10        }  
11        dfs(nums[0] / 10, 0, ans, map);  
12        return ans[0];  
13    }  
14    private void dfs(int node, int sum, int[] ans, Map<Integer, Integer> map) {  
15        if (!map.containsKey(node)) {  
16            return;  
17        }  
18        sum += map.get(node);  
19        int depth = node / 10;  
20        int index = node % 10;  
21        int left = (depth + 1) * 10 + 2 * index - 1;  
22        int right = left + 1;  
23        if (!map.containsKey(left) && !map.containsKey(right)) {  
24            ans[0] += sum;  
25        } else {  
26            dfs(left, sum, ans, map);  
27            dfs(right, sum, ans, map);  
28        }  
29    }  
30}
```

## 681. 最近时刻

### 681. 最近时刻

难度 中等 ⚡ 18 ❤️ 🔍 🎮

给定一个形如 "HH:MM" 表示的时刻，利用当前出现过的数字构造下一个距离当前时间最近的时刻。每个出现数字都可以被无限次使用。

你可以认为给定的字符串一定是合法的。例如，“01:34”和“12:09”是合法的，“1:34”和“12:9”是不合法的。

样例 1：

输入：“19:34”  
输出：“19:39”  
解释：利用数字 1, 9, 3, 4 构造出来的最近时刻是 19:39，是 5 分钟之后。结果不是 19:33 因为这个时刻是 23 小时 59 分钟之后。

样例 2：

输入：“23:59”  
输出：“22:22”  
解释：利用数字 2, 3, 5, 9 构造出来的最近时刻是 22:22。答案一定是第二天的某一时刻，所以选择可构造的最小时刻。

通过次数 954 | 提交次数 1.973

在真实的面试中遇到过这道题？

是  否

贡献者

相关企业 *i*

相关标签

```
1 class Solution {
2     public String nextClosestTime(String time) {
3         int min = Integer.MAX_VALUE;
4         int[] nums = new int[4];
5         for (int i = 0, x = 0; i < time.length(); i++) {
6             if (i == 2)
7                 continue;
8             nums[x] = time.charAt(i) - '0';
9             min = Math.min(min, nums[x]);
10            x++;
11        }
12        boolean found = false;
13        for (int rid = nums.length - 1; rid >= 0; rid--) {
14            int minNum = Integer.MAX_VALUE;
15            for (int lid = 0; lid < nums.length; lid++) {
16                if (nums[rid] < nums[lid])
17                    minNum = Math.min(minNum, nums[lid]);
18            }
19            if (minNum != Integer.MAX_VALUE) {
20                int tmp = nums[rid];
21                nums[rid] = minNum;
22                if (isValid(nums)) {
23                    for (int idx = rid + 1; idx < nums.length; idx++)
24                        nums[idx] = min;
25                    found = true;
26                    break;
27                } else {
28                    nums[rid] = tmp;
29                }
30            }
31        }
32        if (!found) {
33            for (int idx = 0; idx < nums.length; idx++) {
34                nums[idx] = min;
35            }
36        }
37        StringBuilder buffer = new StringBuilder();
38        for (int idx = 0; idx < nums.length; idx++) {
39            buffer.append(nums[idx]);
40            if (idx == 1) buffer.append(":");
41        }
42        return buffer.toString();
43    }
44    boolean isValid(int[] nums) {
45        if (nums[0] > 2 || (nums[0] == 2 && nums[1] > 3) || nums[2] > 5)
46            return false;
47        return true;
48    }
}
```

## 683. K 个空花盆

### 683. K 个空花盆

难度 困难 ⚡ 19 ❤️ 🔍 🎮

花园里有  $N$  个花盆，每个花盆里都有一朵花。这  $N$  朵花会在  $N$  天内依次开放，每天有且仅有花朵会开放并且会一直盛开下去。

给定一个数组  $\text{flowers}$  包含从 1 到  $N$  的数字，每个数字表示在那一天开放的花所在的花盆编号。

例如， $\text{flowers}[i] = x$  表示在第  $i$  天盛开的花在第  $x$  个花盆中， $i$  和  $x$  都在 1 到  $N$  的范围内。

给你一个整数  $k$ ，请你输出在哪一天恰好有两朵盛开的花，他们中间间隔了  $k$  朵花并且都没有开放。

如果不存在，输出 -1。

样例 1：

输入：  
 $\text{flowers}$ : [1,3,2]  
 $k$ : 1  
输出：2  
解释：在第二天，第一朵和第三朵花都盛开了。

```
1 class Solution {
2     public int kEmptySlots(int[] flowers, int k) {
3         int ans = Integer.MAX_VALUE;
4         int[] days = new int[flowers.length];
5         for (int i = 0; i < flowers.length; i++) {
6             days[flowers[i] - 1] = i + 1;
7         }
8         int lid = 0, rid = k + 1;
9         while (rid < days.length) {
10            boolean found = true;
11            for (int idx = lid + 1; idx < rid; ++idx) {
12                if (days[idx] < days[lid] || days[idx] < days[rid]) {
13                    lid = idx;
14                    rid = idx + k + 1;
15                    found = false;
16                    break;
17                }
18            }
19            if (!found) {
20                continue;
21            }
22            ans = Math.min(ans, Math.max(days[lid], days[rid]));
23            lid = rid;
24            rid = lid + k + 1;
25        }
26        return ans < Integer.MAX_VALUE ? ans : -1;
27    }
}
```

## 702. 搜索长度未知的有序数组

### 702. 搜索长度未知的有序数组

难度 中等 ⚡ 7 ❤️ 🔍 🎮

给定一个升序整数数组，写一个函数搜索  $\text{nums}$  中数字  $\text{target}$ 。如果  $\text{target}$  存在，返回它的下标，否则返回 -1。注意，这个数组的大小是未知的。你只能通过 `ArrayReader` 接口访问这个数组，`ArrayReader.get(k)` 返回数组中第  $k$  个元素（下标从 0 开始）。

你可以认为数组中所有的整数都小于 10000。如果你访问数组越界，`ArrayReader.get` 会返回 2147483647。

```
1 class Solution {
2     public int search(ArrayReader reader, int target) {
3         int lid = 0; int rid = 20000 - 1; // [-9999, 9999]
4         while (lid <= rid) {
5             int mid = lid + (rid - lid) / 2;
6             int val = reader.get(mid);
7             if (val < target) {
8                 lid = mid + 1;
9             } else if (val > target) {
10                rid = mid - 1;
11            } else {
12                return mid;
13            }
14        }
15        return -1;
16    }
}
```

## 708. 循环有序列表的插入

708. 循环有序列表的插入

难度 中等 收藏 12 热门 分享 复制

给定循环升序列表中的一个点，写一个函数向这个列表中插入一个新元素，使这个列表仍然是循环升序的。给定的可以是这个列表中任意一个顶点的指针，并不一定是这个列表中最小元素的指针。

如果有多个满足条件的插入位置，你可以选择任意一个位置插入新的值，插入后整个列表仍然保持有序。

如果列表为空（给定的节点是 `null`），你需要创建一个循环有序列表并返回这个点。否则，请返回原先给定的节点。

下面的例子可以帮你更好的理解这个问题：

在上图中，有一个包含三个元素的循环有序列表，你获得值为 3 的节点的指针，我们需要向表中插入元素 2。

新插入的节点应该在 1 和 3 之间，插入之后，整个列表如上图所示，最后返回节点 3。

```
1 v class Solution {
2 v     public Node insert(Node head, int insertVal) {
3 v         if (head == null) {
4 v             Node result = new Node(insertVal);
5 v             result.next = result;
6 v             return result;
7 v         }
8 v         Node node = head;
9 v         while (node.next != head) {
10 v             if (node.val <= insertVal && insertVal <= node.next.val) {
11 v                 // 插入非最值
12 v                 break;
13 v             } else if (node.val <= insertVal
14 v                         && node.next.val < insertVal && node.val > node.next.val) {
15 v                 // 插入一个最大值
16 v                 break;
17 v             } else if (node.val > insertVal
18 v                         && node.next.val >= insertVal && node.val > node.next.val) {
19 v                 // 插入一个最小值
20 v                 break;
21 v             } else {
22 v                 node = node.next;
23 v             }
24 v         }
25 v         Node inserted = new Node(insertVal);
26 v         inserted.next = node.next;
27 v         node.next = inserted;
28 v         return head;
29 v     }
}
```

## 716. 最大栈

716. 最大栈

难度 简单 收藏 18 热门 分享 复制

设计一个最大栈，支持 `push`、`pop`、`top`、`peekMax` 和 `popMax` 操作。

1. `push(x)` -- 将元素 `x` 压入栈中。
2. `pop()` -- 移除栈顶元素并返回这个值。
3. `top()` -- 返回栈顶元素。
4. `peekMax()` -- 返回栈中最大元素。
5. `popMax()` -- 返回栈中最大的元素，并将其删除。如果有多个最大元素，只要删除最靠近栈顶的那个。

样例 1：

```
MaxStack stack = new MaxStack();
stack.push(5);
stack.push(1);
stack.push(5);
stack.top(); -> 5
stack.popMax(); -> 5
stack.top(); -> 1
stack.peekMax(); -> 5
stack.pop(); -> 1
stack.top(); -> 5
```

```
1 v class MaxStack {
2 v     private ArrayDeque<Integer> mStack;
3 v     private ArrayDeque<Integer> mMaxStack;
4 v     public MaxStack() {
5 v         mStack = new ArrayDeque<>();
6 v         mMaxStack = new ArrayDeque<>();
7 v     }
8 v     public void push(int x) {
9 v         int max = mMaxStack.isEmpty() ? x : mMaxStack.peek();
10 v        mStack.push(x);
11 v        mMaxStack.push(x > max ? x : max);
12 v    }
13 v    public int pop() {
14 v        mMaxStack.pop();
15 v        return mStack.pop();
16 v    }
17 v    public int top() {
18 v        return mStack.peek();
19 v    }
20 v    public int peekMax() {
21 v        return mMaxStack.peek();
22 v    }
23 v    public int popMax() {
24 v        int max = peekMax();
25 v        ArrayDeque<Integer> stack = new ArrayDeque<>();
26 v        while (top() != max)
27 v            stack.push(pop());
28 v        pop();
29 v        while (!stack.isEmpty())
30 v            push(stack.pop());
31 v        return max;
32 v    }
33 }
```

## 723. 粉碎糖果

### 723. 粉碎糖果

难度 中等 24  
这个问题是实现一个简单的消除算法。  
给定一个二维整数数组 board 代表糖果所在的方格，不同的正整数 board[i][j] 代表不同种类的糖果，如果 board[i][j] = 0 表示 (i, j) 这个位置是空的。给定的方格是玩家移动后的游戏状态，现在需要你根据以下规则粉碎糖果，使得整个方格处于稳定状态并最终输出。

- 如果有三个及以上水平或者垂直相连的同种糖果，同一时间将它们粉碎，即把这些位置变成空的。
- 在同时粉碎掉这些糖果之后，如果有一个空的位置上方还有糖果，那么上方的糖果就会下落直到碰到下方的糖果或者底部，这些糖果都是同时下落，也不会有新的糖果从顶部出现并落下来。
- 通过前两步的操作，可能又会出现可以粉碎的糖果，请继续重复前面的操作。
- 当不存在可以粉碎的糖果，也就是状态稳定之后，请输出最终的状态。

你需要模拟上述规则使整个方格达到稳定状态，并输出。

样例：

输入：  
board =  
[[110, 5, 112, 113, 114], [210, 211, 5, 213, 214],  
[310, 311, 3, 313, 314], [410, 411, 412, 5, 414],  
[5, 1, 512, 3, 3], [610, 4, 1, 613, 614], [710, 1, 2, 713, 714],  
[810, 1, 2, 1, 1], [1, 1, 2, 2, 2], [4, 1, 4, 4, 1014]]  
  
输出：  
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0],  
[110, 0, 0, 0, 114], [210, 0, 0, 0, 214], [310, 0, 0, 113, 314],  
[410, 0, 0, 213, 414], [610, 211, 112, 313, 614],  
[710, 311, 412, 613, 714], [810, 411, 512, 713, 1014]]

```
1 class Solution {  
2     public int[][] candyCrush(int[][] board) {  
3         int rows = board.length;  
4         int cols = board[0].length;  
5         boolean stable = true;  
6         for (int rid = 0; rid < rows; rid++) { // 横向粉碎  
7             for (int cid = 0; cid < cols - 2; cid++) {  
8                 int abs = Math.abs(board[rid][cid]);  
9                 if (abs != 0 && abs == Math.abs(board[rid][cid + 1]))  
10                && abs == Math.abs(board[rid][cid + 2])) {  
11                     board[rid][cid] = board[rid][cid + 1] = board[rid][cid + 2] = -abs;  
12                     stable = false;  
13                 }  
14             }  
15         }  
16         for (int rid = 0; rid < rows - 2; rid++) { // 纵向粉碎  
17             for (int cid = 0; cid < cols; cid++) {  
18                 int abs = Math.abs(board[rid][cid]);  
19                 if (abs != 0 && abs == Math.abs(board[rid + 1][cid]))  
20                    && abs == Math.abs(board[rid + 2][cid])) {  
21                     board[rid][cid] = board[rid + 1][cid] = board[rid + 2][cid] = -abs;  
22                     stable = false;  
23                 }  
24             }  
25         }  
26         for (int cid = 0; cid < cols; cid++) { // 纵向下落  
27             int idx = rows - 1;  
28             for (int rid = rows - 1; rid >= 0; rid--) {  
29                 if (board[rid][cid] > 0) {  
30                     board[idx--][cid] = board[rid][cid];  
31                 }  
32             }  
33             while (idx >= 0) {  
34                 board[idx--][cid] = 0;  
35             }  
36         }  
37     }  
38     return stable ? board : candyCrush(board);  
39 }  
40 }
```

## 727. 最小窗口子序列

### 727. 最小窗口子序列

难度 困难 24  
给定字符串 s 和 t，找出 s 中最短的（连续）子串 w，使得 t 是 w 的子序列。

如果 s 中没有窗口可以包含 t 中的所有字符，返回空字符串 ""。如果有不止一个最短长度的窗口，返回开始位置最靠左的那个。

示例 1：

输入：  
S = "abcdebbde"，T = "bde"  
输出："bcde"  
解释：  
"bcde" 是答案，因为它在相同长度的字符串 "bdde" 出现之前。  
"deb" 不是一个更短的答案，因为在窗口中必须按顺序出现 T 中的元素。

注：

- 所有输入的字符串都只包含小写字母。All the strings in the input will only contain lowercase letters.
- s 长度的范围为 [1, 20000]。
- t 长度的范围为 [1, 100]。

```
1 class Solution {  
2     public String minWindow(String S, String T) {  
3         char[] sChars = S.toCharArray();  
4         char[] tChars = T.toCharArray();  
5         int sLen = sChars.length, tLen = tChars.length;  
6         int sIdx = 0, tIdx = 0, lid = 0, rid = sLen + 1;  
7         while (sIdx < sLen) {  
8             if (sChars[sIdx] == tChars[tIdx])  
9                 tIdx++;  
10            if (tIdx == tLen) {  
11                int tmp = sIdx;  
12                tIdx--;  
13                while (tIdx >= 0) {  
14                    if (sChars[sIdx] == tChars[tIdx])  
15                        tIdx--;  
16                    sIdx++;  
17                }  
18                if (tmp - sIdx + 1 < rid - lid + 1) {  
19                    lid = sIdx;  
20                    rid = tmp;  
21                }  
22            }  
23            tIdx = 0;  
24        }  
25        sIdx++;  
26    }  
27    return rid == sLen + 1 ? "" : S.substring(lid, rid + 1);  
28 }  
29 }
```

## 734. 句子相似性

734. 句子相似性  
难度 简单 12  
给定两个句子 words1, words2 (每个用字符串表示)，和一个相似单词对的列表 pairs，判断是否两个句子是相似的。

例如，当相似单词对是 pairs = [{"great", "fine"}, {"acting", "drama"}, {"skills", "talent"}] 的时候，“great acting skills” 和 “fine drama talent” 是相似的。

注意相似关系是不具有传递性的。例如，如果 “great” 和 “fine” 是相似的，“fine” 和 “good” 是相似的，但是 “great” 和 “good” 未必是相似的。

但是，相似关系是具有对称性的。例如，“great” 和 “fine” 是相似的，相当于 “fine” 和 “great” 是相似的。

而且，一个单词总是与其自身相似。例如，句子 words1 = ["great"], words2 = ["great"]，pairs = [] 是相似的，尽管没有输入特定的相似单词对。

最后，句子只会在具有相同单词个数的前提下才会相似。所以一个句子 words1 = ["great"] 永远不可能和句子 words2 = ["doubleplus", "good"] 相似。

```
1 class Solution {  
2     public boolean areSentencesSimilar(String[] words1, String[] words2, List<List<String>> pairs) {  
3         if (words1.length != words2.length) return false;  
4         if (words1.length == 0) return true;  
5         Map<String, Set<String>> map = new HashMap<>();  
6         for (List<String> pair : pairs) {  
7             Set<String> set = map.getOrDefault(pair.get(0), new HashSet<>());  
8             set.add(pair.get(1));  
9             map.put(pair.get(0), set);  
10        }  
11        for (int idx = 0; idx < words1.length; idx++) {  
12            if (words1[idx].equals(words2[idx]))  
13                continue;  
14            else if (map.containsKey(words1[idx]) && map.get(words1[idx]).contains(words2[idx]))  
15                continue;  
16            else if (map.containsKey(words2[idx]) && map.get(words2[idx]).contains(words1[idx]))  
17                continue;  
18            else  
19                return false;  
20        }  
21        return true;  
22    }  
23 }
```

## 737. 句子相似性 II

### 737. 句子相似性 II

难度 中等 ⚡ 10 ❤️ 💬 📄

给定两个句子 words1, words2 (每个用字符串数组表示), 和一个相似单词对的列表 pairs , 判断是否两个句子是相似的。

例如, 当相似单词对是 pairs = [["great", "fine"], ["acting", "drama"], ["skills", "talent"]] 的时候, words1 = ["great", "acting", "skills"] 和 words2 = ["fine", "drama", "talent"] 是相似的。

注意相似关系是具有传递性的。例如, 如果 "great" 和 "fine" 是相似的, "fine" 和 "good" 是相似的, 则 "great" 和 "good" 是相似的。

而且, 相似关系是具有对称性的。例如, "great" 和 "fine" 是相似的相当于 "fine" 和 "great" 是相似的。

并且, 一个单词总是与其自身相似。例如, 句子 words1 = ["great"], words2 = ["great"], pairs = [] 是相似的, 尽管没有输入特定的相似单词对。

最后, 句子只会在具有相同单词个数的前提下才会相似。所以一个句子 words1 = ["great"] 永远不可能和句子 words2 = ["doubleplus", "good"] 相似。

注:

- words1 和 words2 的长度不会超过 1000 。
- pairs 的长度不会超过 2000 。
- 每个 pairs[i] 的长度为 2 。
- 每个 words[i] 和 pairs[i][j] 的长度范围为 [1, 20] 。

通过次数 1,174 | 提交次数 2,704

在真实的面试中遇到过这道题?  是  否

```
1 class Solution {  
2     public boolean areSentencesSimilarTwo(  
3         String[] words1, String[] words2, List<List<String>> pairs) {  
4         if (words1.length != words2.length)  
5             return false;  
6         Map<String, List<String>> graph = new HashMap<>();  
7         for (List<String> pair : pairs) {  
8             for (String p : pair)  
9                 if (!graph.containsKey(p))  
10                     graph.put(p, new ArrayList<>());  
11             graph.get(pair.get(0)).add(pair.get(1));  
12             graph.get(pair.get(1)).add(pair.get(0));  
13         }  
14         for (int idx = 0; idx < words1.length; ++idx) {  
15             String word1 = words1[idx], word2 = words2[idx];  
16             Stack<String> stack = new Stack<>();  
17             Set<String> seen = new HashSet<>();  
18             stack.push(word1);  
19             seen.add(word1);  
20             search: {  
21                 while (!stack.isEmpty()) {  
22                     String word = stack.pop();  
23                     if (word.equals(word2))  
24                         break search;  
25                     if (graph.containsKey(word))  
26                         for (String nei : graph.get(word))  
27                             if (!seen.contains(nei)) {  
28                                 stack.push(nei);  
29                                 seen.add(nei);  
30                             }  
31             }  
32             if (stack.isEmpty())  
33                 return false;  
34         }  
35     }  
36     return true;  
37 }
```

## 737. 句子相似性 III

### 737. 句子相似性 III

难度 中等 ⚡ 10 ❤️ 💬 📄

给定两个句子 words1, words2 (每个用字符串数组表示), 和一个相似单词对的列表 pairs , 判断是否两个句子是相似的。

例如, 当相似单词对是 pairs = [["great", "fine"], ["acting", "drama"], ["skills", "talent"]] 的时候, words1 = ["great", "acting", "skills"] 和 words2 = ["fine", "drama", "talent"] 是相似的。

注意相似关系是具有传递性的。例如, 如果 "great" 和 "fine" 是相似的, "fine" 和 "good" 是相似的, 则 "great" 和 "good" 是相似的。

而且, 相似关系是具有对称性的。例如, "great" 和 "fine" 是相似的相当于 "fine" 和 "great" 是相似的。

并且, 一个单词总是与其自身相似。例如, 句子 words1 = ["great"], words2 = ["great"], pairs = [] 是相似的, 尽管没有输入特定的相似单词对。

最后, 句子只会在具有相同单词个数的前提下才会相似。所以一个句子 words1 = ["great"] 永远不可能和句子 words2 = ["doubleplus", "good"] 相似。

注:

- words1 和 words2 的长度不会超过 1000 。
- pairs 的长度不会超过 2000 。
- 每个 pairs[i] 的长度为 2 。
- 每个 words[i] 和 pairs[i][j] 的长度范围为 [1, 20] 。

```
1 class Solution { // 并查集, 效率高  
2     public boolean areSentencesSimilarTwo(String[] words1, String[] words2,  
3                                         List<List<String>> pairs) {  
4         if (words1.length != words2.length) {  
5             return false;  
6         }  
7         Map<String, String> map = new HashMap<>();  
8         for (List<String> pair : pairs) {  
9             union(pair.get(0), pair.get(1), map);  
10        }  
11        for (int i = 0; i < words1.length; i++) {  
12            if (!find(words1[i], map).equals(find(words2[i], map))) {  
13                return false;  
14            }  
15        }  
16        return true;  
17    }  
18  
19    public void union(String word1, String word2, Map<String, String> map) {  
20        String str1 = find(word1, map);  
21        String str2 = find(word2, map);  
22        if (!str1.equals(str2)) {  
23            map.put(str1, str2);  
24        }  
25    }  
26  
27    public String find(String word, Map<String, String> map) {  
28        while (map.containsKey(word) && map.get(word) != word) {  
29            word = map.get(word);  
30        }  
31        return word;  
32    }  
33 }
```

## 742. 二叉树最近的叶节点

### 742. 二叉树最近的叶节点

难度 中等 16 收藏 复制 举报

给定一个每个结点的值互不相同的二叉树，和一个目标值  $k$ ，找出树中与目标值  $k$  最近的叶结点。

这里，与叶结点 **最近** 表示在二叉树中到达该叶节点需要行进的边数与到达其它叶结点相比最少。而且，当一个结点没有孩子结点时称其为叶结点。

在下面的例子中，输入的树以逐行的平铺形式表示。实际上的有根树 `root` 将以 `TreeNode` 对象的形式给出。

示例 1：

输入：  
`root = [1, 3, 2], k = 1`

二叉树图示：  
1  
/ \  
3 2

输出：2 (或 3)

解释：2 和 3 都是距离目标 1 最近的叶节点。

示例 2：

输入：  
`root = [1], k = 1`

输出：1

解释：最近的叶节点是根结点自身。

```
1 public class Solution {
2     public int findClosestLeaf(TreeNode root, int k) {
3         if (root.left == null && root.right == null)
4             return k;
5         final int len = 1001;
6         List<Integer>[] graph = new List[len];
7         for (int i = 0; i < graph.length; i++)
8             graph[i] = new ArrayList<>();
9         Queue<TreeNode> nodeQueue = new LinkedList<>();
10        nodeQueue.offer(root);
11        boolean[] leaf = new boolean[len];
12        while (!nodeQueue.isEmpty()) {
13            TreeNode node = nodeQueue.poll();
14            if (node.left == null && node.right == null)
15                leaf[node.val] = true;
16            if (node.left != null) {
17                graph[node.val].add(node.left.val);
18                graph[node.left.val].add(node.val);
19                nodeQueue.add(node.left);
20            }
21            if (node.right != null) {
22                graph[node.val].add(node.right.val);
23                graph[node.right.val].add(node.val);
24                nodeQueue.add(node.right);
25            }
26        }
27        Queue<Integer> valueQueue = new LinkedList<>();
28        valueQueue.offer(k);
29        boolean[] visited = new boolean[len];
30        while (!valueQueue.isEmpty()) {
31            int size = valueQueue.size();
32            for (int i = 0; i < size; i++) {
33                int val = valueQueue.poll();
34                visited[val] = true;
35                if (leaf[val])
36                    return val;
37                for (int idx = 0; idx < graph[val].size(); idx++)
38                    if (!visited[graph[val].get(idx)])
39                        valueQueue.add(graph[val].get(idx));
40            }
41        }
42        return -1;
43    }
44}
```

## 750. 角矩形的数量

### 750. 角矩形的数量

难度 中等 15 收藏 复制 举报

给定一个只包含 0 和 1 的网格，找出其中角矩形的数量。

一个「角矩形」是由四个不同的在网格上的 1 形成的轴对称的矩形。注意只有角的位置才需要为 1。并且，4 个 1 需要是不同的。

示例 1：

输入：  
`grid = [[1, 0, 0, 1, 0], [0, 0, 1, 0, 1], [0, 0, 0, 1, 0], [1, 0, 1, 0, 1]]`

输出：1

解释：只有一个角矩形，角的位置为 `grid[1][2]`, `grid[1][4]`, `grid[3][2]`, `grid[3][4]`。

```
1 class Solution {
2     public int countCornerRectangles(int[][] grid) {
3         if (grid.length < 2 || grid[0].length < 2)
4             return 0;
5         int ans = 0;
6         int[][] map = new int[grid[0].length][grid[0].length];
7         for (int[] row : grid) {
8             for (int rid = 0; rid < row.length; rid++) {
9                 if (row[rid] == 1) {
10                     for (int cid = rid + 1; cid < row.length; cid++) {
11                         if (row[cid] == 1) {
12                             ans += map[rid][cid]++;
13                         }
14                     }
15                 }
16             }
17         }
18         return ans;
19     }
20 }
```

## 751. IP 到 CIDR

### 751. IP 到 CIDR

难度 简单 ⚡ 8 ❤ 8 🌐 8 ⚡ 8

给定一个起始 IP 地址 `ip` 和一个我们需要包含的 IP 的数量 `n`，返回用列表（最小可能的长度）表示的 CIDR 块的范围。

CIDR 块是包含 IP 的字符串，后接斜杠和固定长度。例如：  
“123.45.67.89/20”。固定长度 “20” 表示在特定的范围内中公共前缀的长度。

示例 1：

```
输入: ip = "255.0.0.7", n = 10
输出:
["255.0.0.7/32", "255.0.0.8/29", "255.0.0.16/32"]
解释:
转换为二进制时, 初始IP地址如下所示 (为清晰起见添加了空格) :
255.0.0.7 -> 11111111 00000000 00000000 00000111
地址 "255.0.0.7/32" 表示与给定地址有相同的 32 位前缀的所有地址,
在这里只有这一个地址。

地址 "255.0.0.8/29" 表示与给定地址有相同的 29 位前缀的所有地址:
255.0.0.8 -> 11111111 00000000 00000000 00001000
有相同的 29 位前缀的地址如下:
11111111 00000000 00000000 00001000
11111111 00000000 00000000 00001001
11111111 00000000 00000000 00001010
11111111 00000000 00000000 00001011
11111111 00000000 00000000 00001100
11111111 00000000 00000000 00001101
11111111 00000000 00000000 00001110
11111111 00000000 00000000 00001111

地址 "255.0.0.16/32" 表示与给定地址有相同的 32 位前缀的所有地址,
```

```
class Solution {
    public List<String> ipToCIDR(String ip, int n) {
        int curr = toInt(ip);
        List<String> ans = new ArrayList<>();
        while (n > 0) {
            int maxBits = Integer.numberOfTrailingZeros(curr);
            int bitVal = 1;
            int count = 0;
            while (bitVal < n && count < maxBits) {
                bitVal <<= 1;
                ++count;
            }
            if (bitVal > n) {
                bitVal >>= 1;
                --count;
            }
            ans.add(toString(curr, 32 - count));
            n -= bitVal;
            curr += (bitVal);
        }
        return ans;
    }

    private String toString(int number, int range) {
        final int WORD_SIZE = 8;
        StringBuilder buffer = new StringBuilder();
        for (int i = 3; i >= 0; --i) {
            buffer.append(((number >> (i * WORD_SIZE)) & 255));
            buffer.append(".");
        }
        buffer.setLength(buffer.length() - 1);
        buffer.append("/").append(range);
        return buffer.toString();
    }

    private int toInt(String ip) {
        String[] sep = ip.split("\\.");
        int sum = 0;
        for (int i = 0; i < sep.length; ++i) {
            sum *= 256;
            sum += Integer.parseInt(sep[i]);
        }
        return sum;
    }
}
```

## 755. 倒水

### 755. 倒水

难度 中等 ⚡ 8 ❤ 8 🌐 8 ⚡ 8

给出一个地形高度图，`heights[i]` 表示该索引处的高度。每个索引的宽度为 1。在 `v` 个单位的水落在索引 `k` 处以后，每个索引位置有多少水？

水最先会在索引 `k` 处下降并且落在该索引位置的最高地形或水面之上。然后按如下方式流动：

- 如果液滴最终可以通过向左流动而下降，则向左流动。
  - 否则，如果液滴最终可以通过向右流动而下降，则向右流动。
  - 否则，在当前位置上升。
- 这里，“最终下降”的意思是液滴如果按此方向移动的话，最终可以下降到一个较低的水平。而且，“水平”的意思是当前列的地形的高度加上水的高度。

我们可以假定在数组两侧的边界外有无限高的地形。而且，不能有部分水在多于 1 个的网格块上均匀分布 - 每个单位的水必须要位于一个块中。

```
class Solution {
    public int[] pourWater(int[] heights, int V, int K) {
        while (V-- > 0) {
            int min = heights[K];
            int minIdx = K;
            for (int idx = K - 1; idx >= 0 && heights[idx] <= min; idx--) {
                if (heights[idx] < min) {
                    min = heights[(minIdx = idx)];
                }
            }
            if (minIdx == K) {
                for (int idx = K + 1; idx < heights.length
                     && heights[idx] <= min; idx++) {
                    if (heights[idx] < min) {
                        min = heights[(minIdx = idx)];
                    }
                }
            }
            heights[minIdx]++;
        }
        return heights;
    }
}
```

## 758. 字符串中的加粗单词

### 758. 字符串中的加粗单词

难度 简单 ⚡ 12 ❤ 12 🌐 12 ⚡ 12

给定一个关键词集合 `words` 和一个字符串 `s`，将所有 `s` 中出现的关键词加粗。所有在标签 `<b>` 和 `</b>` 中的字母都会加粗。

返回的字符串需要使用尽可能少的标签，当然标签应形成有效的组合。

例如，给定 `words = ["ab", "bc"]` 和 `s = "aabca"`，需要返回 `"a<b>abc</b>d"`。注意返回 `"a<b>a<b>b</b>c</b>d"` 会使用更多的标签，因此是错误的。

注：

- `words` 长度的范围为 `[0, 50]`。
- `words[i]` 长度的范围为 `[1, 10]`。
- `s` 长度的范围为 `[0, 500]`。
- 所有 `words[i]` 和 `s` 中的字符都为小写字母。

通过次数 1,650 提交次数 3,753

```
class Solution {
    public String boldWords(String[] words, String S) {
        boolean[] bolds = new boolean[S.length()];
        for (String word : words) checkBold(word, S, bolds);
        StringBuffer buffer = new StringBuffer();
        if (bolds[0]) buffer.append("<b>");
        for (int idx = 0; idx < bolds.length - 1; idx++) {
            buffer.append(S.charAt(idx));
            if (!bolds[idx] && bolds[idx + 1]) buffer.append("<b>");
            if (bolds[idx] && !bolds[idx + 1]) buffer.append("</b>");
        }
        buffer.append(S.charAt(S.length() - 1));
        if (bolds[bolds.length - 1]) buffer.append("</b>");
        return buffer.toString();
    }

    private void checkBold(String word, String str, boolean[] bolds) {
        int idx = 0;
        while (true) {
            idx = str.indexOf(word, idx);
            if (idx == -1) break;
            for (int i = idx; i < idx + word.length(); i++)
                bolds[i] = true;
            idx++;
        }
    }
}
```

## 759. 员工空闲时间

759. 员工空闲时间

难度 困难 12 次 | 收藏 | 分享 | 复制 | 回答

给定员工的 schedule 列表，表示每个员工的工作时间。

每个员工都有一个非重叠的时间段 Intervals 列表，这些时间段已经排序。

返回表示所有员工的共同、正数长度的空闲时间 的有限时间段的列表，同样需要排序。

示例 1：

```
输入: schedule = [[[1,2],[5,6]],[[1,3]],[[4,10]]]
输出: [[3,4]]
解释:
共有 3 个员工，并且所有共同的
```

```
1 class Solution {
2     public List<Interval> employeeFreeTime(List<List<Interval>> schedule) {
3         List<Interval> ans = new ArrayList();
4         PriorityQueue<Interval> queue;
5         queue = new PriorityQueue<(a, b) -> a.start - b.start>();
6         schedule.forEach(e -> queue.addAll(e));
7         Interval node = queue.poll();
8         while (!queue.isEmpty()) {
9             if (node.end < queue.peek().start) {
10                 ans.add(new Interval(node.end, queue.peek().start));
11                 node = queue.poll();
12             } else {
13                 node = node.end < queue.peek().end ? queue.peek() : node;
14                 queue.poll();
15             }
16         }
17     return ans;
18 }
```

## 760. 找出变位映射

760. 找出变位映射

难度 简单 15 次 | 收藏 | 分享 | 复制 | 回答

给定两个列表 A 和 B，并且 B 是 A 的变位（即 B 是由 A 中的元素随机排列后组成的新列表）。

我们希望找出一个从 A 到 B 的索引映射 P。一个映射 P[i] = j 指的是列表 A 中的第 i 个元素出现于列表 B 中的第 j 个元素上。

列表 A 和 B 可能出现重复元素。如果有多于一种答案，输出任意一种。

例如，给定

```
A = [12, 28, 46, 32, 50]
B = [50, 12, 32, 46, 28]    返回 [1, 4, 3, 2, 0]
```

```
1 class Solution {
2     public int[] anagramMappings(int[] A, int[] B) {
3         int[] ans = new int[A.length];
4         for (int idx = 0; idx < A.length; idx++) {
5             ans[idx] = findIndex(B, A[idx]);
6         }
7         return ans;
8     }
9
10 private int findIndex(int[] nums, int num) {
11     for (int idx = 0; idx < nums.length; idx++) {
12         if (nums[idx] == num) {
13             return idx;
14         }
15     }
16     return 0;
17 }
18 }
```

## 772. 基本计算器 III (与 224. 基本计算器/227. 基本计算器 II 通用)

772. 基本计算器 III

难度 困难 17 次 | 收藏 | 分享 | 复制 | 回答

实现一个基本的计算器来计算简单的表达式字符串。

表达式字符串可以包含左括号 ( 和右括号 )，加号 + 和减号 -，非负整数和空格。

表达式字符串只包含非负整数，+，-，\*，/ 操作符，左括号 (，右括号 ) 和空格。整数除法需要向下截断。

你可以假定给定的字符串总是有效的。所有的中间结果的范围为 [-2147483648, 2147483647]。

一些例子：

```
"1 + 1" = 2
"6-4 / 2 " = 4
"2*(5+5*2)/3+(6/2+8)" = 21
"(2+6* 3+5- (3*14/7+2)*5)+3"=-12
```

注：不要 使用内置库函数 eval。

通过次数 1,022 | 提交次数 2,932

在真实的面试中遇到过这道题？

贡献者



相关企业 i



相关标签



相似题目



```
1 class Solution {
2     public int calculate(String s) {
3         return calculator(s, new int[]{0});
4     }
5
6     public int calculator(String s, int[] idxes) {
7         char sign = '+'; int num = 0; int len = s.length();
8         Stack<Integer> stack = new Stack();
9         for (int idx = idxes[0]; idx < len; idx++) {
10            char ch = s.charAt(idx);
11            if (Character.isDigit(ch)) {
12                num = num * 10 + (ch - '0');
13            }
14            if (ch == '(') {
15                idxes[0] = idx + 1;
16                num = calculator(s, idxes);
17                idx = idxes[0] - 1;
18            }
19            if (!Character.isDigit(ch) && ch != ' ') {
20                int prev;
21                switch (sign) {
22                    case '+':
23                        stack.push(num);
24                        break;
25                    case '-':
26                        stack.push(num * -1);
27                        break;
28                    case '*':
29                        prev = stack.pop();
30                        stack.push(prev * num);
31                        break;
32                    case '/':
33                        prev = stack.pop();
34                        stack.push(prev / num);
35                        break;
36                }
37                sign = ch;
38                num = 0;
39            }
40            if (ch == ')') {
41                idxes[0] = idx + 1;
42                break;
43            }
44        }
45        int ans = 0;
46        while (!stack.isEmpty())
47            ans += stack.pop();
48    }
49 }
```

## 774. 最小化去加油站的最大距离

### 774. 最小化去加油站的最大距离

难度 困难 ⚡ 9 ❤ 1 ⚡ 🎮 1 ⚡ 🔍 1

假设我们在一条水平数轴上，列表 `stations` 来表示各个加油站的位置，加油站分别在 `stations[0]`, `stations[1]`, ..., `stations[N-1]` 的位置上，其中 `N = stations.length`。

现在我们希望增加 `K` 个新的加油站，使得相邻两个加油站的距离 `D` 尽可能的最小，请你返回 `D` 可能的最小值。

示例：

输入: `stations = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, `K = 9`  
输出: `0.500000`

注：

- `stations.length` 是在范围 `[10, 2000]` 内的整数
- `stations[i]` 是在范围 `[0, 10^8]` 内的整数
- `K` 是在范围 `[1, 10^6]` 内的整数
- 在 `10^-6` 以内的正确值会被视为正确的答案

```
1 class Solution {
2     public double minmaxGasDist(int[] stations, int K) {
3         int len = stations.length;
4         double[] gaps = new double[len - 1];
5         for (int idx = 0; idx < len - 1; idx++)
6             gaps[idx] = stations[idx + 1] - stations[idx];
7         double lo = 0, hi = Integer.MAX_VALUE, eps = 1e-7;
8         while (Math.abs(hi - lo) > eps) {
9             double mid = (lo + hi) / 2;
10            if (check(gaps, mid, K)) {
11                hi = mid;
12            } else {
13                lo = mid;
14            }
15        }
16        return lo;
17    }
18    boolean check(double[] gaps, double mid, int target) {
19        int cnt = 0;
20        for (int idx = 0; idx < gaps.length; idx++)
21            cnt += (int)(gaps[idx] / mid);
22        return cnt <= target;
23    }
24 }
```

## 776. 拆分二叉搜索树

### 776. 拆分二叉搜索树

难度 中等 ⚡ 31 ❤ 1 ⚡ 🎮 1 ⚡ 🔍 1

给你一棵二叉搜索树 (BST)、它的根结点 `root` 以及目标值 `v`。

请将该树按要求拆分为两个子树：其中一个子树结点的值都必须小于等于给定的目标值 `v`；另一个子树结点的值都必须大于目标值 `v`；树中并非一定要存在值为 `v` 的结点。

除此之外，树中大部分结构都需要保留，也就是说原始树中父节点 P 的任意子节点 C，假如拆分后它们仍在同一个子树中，那么结点 P 应仍为 C 的结点。

你需要返回拆分后两个子树的根结点 `TreeNode`，顺序随意。

```
1 class Solution {
2     public TreeNode[] splitBST(TreeNode root, int V) {
3         if (root == null)
4             return new TreeNode[]{null, null};
5         if (root.val <= V) {
6             TreeNode[] nodes = splitBST(root.right, V);
7             root.right = nodes[0];
8             nodes[0] = root;
9             return nodes;
10        } else {
11            TreeNode[] nodes = splitBST(root.left, V);
12            root.left = nodes[1];
13            nodes[1] = root;
14            return nodes;
15        }
16    }
17 }
```

## 800. 相似 RGB 颜色

### 800. 相似 RGB 颜色

难度 简单 ⚡ 5 ❤ 1 ⚡ 🎮 1 ⚡ 🔍 1

RGB 颜色用十六进制来表示的话，每个大写字母都代表了某个从 0 到 f 的 16 进制数。

RGB 颜色 "#AABBCC" 可以简写成 "#ABC"。例如，"#15c" 其实是 "#115cc" 的简写。

现在，假如我们分别定义两个颜色 "#ABCDEF" 和 "#UVWXYZ"，则它们的相似度可以通过这个表达式  $-(AB - UV)^2 - (CD - WX)^2 - (EF - YZ)^2$  来计算。

那么给定颜色 "#ABCDEF"，请你返回一个与 #ABCDEF 最相似的 7 个字符代表的颜色，并且它是可以被简写形式表达的。（比如，可以表示成类似 "#XYZ" 的形式）

示例 1：  
输入: color = "#09f166"  
输出: "#11ee66"  
解释:  
因为相似度计算得出  $-(0x09 - 0x11)^2 - (0xf1 - 0xee)^2 - (0x66 - 0x66)^2 = -64 - 9 - 0 = -73$   
这已经是所有可以简写的颜色中最相似的了

```
1 class Solution {
2     public String similarRGB(String color) {
3         StringBuilder buffer = new StringBuilder();
4         for (int i = color.length(); i > 1; i = i - 2) {
5             String str = color.substring(i - 2, i);
6             int val = Integer.valueOf(str.substring(0, 1), 16);
7             int num = Integer.valueOf(str, 16);
8             int[] nums = new int[3];
9             nums[0] = val > 0 ? (val - 1) * 16 + (val - 1) : 0;
10            nums[1] = val > 0 ? val * 16 + val : 0;
11            nums[2] = val < 15 ? (val + 1) * 16 + (val + 1) : 16 * 15 + 16;
12            int min = num, index = 0;
13            for (int x = 0; x <= 2; x++) {
14                int abs = Math.abs(nums[x] - num);
15                if (abs < min) {
16                    min = abs;
17                    index = x;
18                }
19            }
20            String tmp = Integer.toHexString(nums[index]);
21            if (nums[index] < 16) {
22                tmp = "0" + tmp;
23            }
24            buffer.append(tmp);
25        }
26        return buffer.append("#").reverse().toString();
27    }
28 }
```

## 1055. 形成字符串的最短路径

### 1055. 形成字符串的最短路径

难度 中等 17 次提交 通过 等级分布

对于任何字符串，我们可以通过删除其中一些字符（也可能不删除）来构造该字符串的子序列。

给定源字符串 `source` 和目标字符串 `target`，找出源字符串中能通过串联形成目标字符串的子序列的最小数量。如果无法通过串联源字符串中的子序列来构造目标字符串，则返回 `-1`。

示例 1：

```
输入: source = "abc", target = "abcbc"
输出: 2
解释: 目标字符串 "abcbc" 可以由 "abc" 和 "bc" 形成，它们都是源字符串 "abc" 的子序列。
```

示例 2：

```
输入: source = "abc", target = "acdbc"
输出: -1
解释: 由于目标字符串中包含字符 "d"，所以无法由源字符串的子序列构建目标字符串。
```

```
1 class Solution {
2     public int shortestWay(String source, String target) {
3         char[] sChars = source.toCharArray();
4         char[] tChars = target.toCharArray();
5         int[] count = new int[26];
6         for (char ch: sChars) {
7             count[ch - 'a']++;
8         }
9         for (char ch: tChars) {
10            if (count[ch - 'a'] == 0) {
11                return -1;
12            }
13        }
14        int ans = 1;
15        int sIdx = 0, tIdx = 0;
16        while (tIdx < tChars.length) {
17            if (sChars[sIdx] == tChars[tIdx]) {
18                tIdx++;
19            }
20            sIdx++;
21            if (tIdx == tChars.length) {
22                break;
23            }
24            if (sIdx == sChars.length) {
25                ans++;
26                sIdx = 0;
27            }
28        }
29        return ans;
30    }
31 }
```

## 1063. 有效子数组的数目

### 1063. 有效子数组的数目

难度 困难 12 次提交 通过 等级分布

给定一个整数数组 `A`，返回满足下面条件的 非空、连续 子数组的数目：

子数组中，最左侧的元素不大于其他元素。

示例 1：

```
输入: [1,4,2,5,3]
输出: 11
```

```
1 class Solution {
2     public int validSubarrays(int[] nums) {
3         int ans = 0;
4         Stack<Integer> stack = new Stack<>();
5         for (int idx = nums.length - 1; idx >= 0; idx--) {
6             while (!stack.isEmpty() && nums[stack.peek()] >= nums[idx]) {
7                 stack.pop();
8             }
9             ans += stack.isEmpty() ? nums.length - idx : stack.peek() - idx;
10        }
11    }
12    return ans;
13 }
14 }
```

## 1056. 易混淆数

### 1056. 易混淆数

难度 简单 5 次提交 通过 等级分布

给定一个数字 `N`，当它满足以下条件的时候返回 `true`：

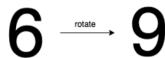
原数字旋转  $180^\circ$  以后可以得到新的数字。

如 0, 1, 6, 8, 9 旋转  $180^\circ$  以后，得到了新的数字 0, 1, 9, 8, 6。

2, 3, 4, 5, 7 旋转  $180^\circ$  后，得到的不是数字。

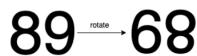
易混淆数 (confusing number) 在旋转 $180^\circ$ 以后，可以得到和原来不同的数，且新数字的每一位都是有效的。

示例 1：



```
输入: 6
输出: true
解释:
把 6 旋转 180° 以后得到 9, 9 是有效数字且 9!=6 。
```

示例 2：



```
1 class Solution {
2     private static final HashSet<Integer> VALID_NUMS = new HashSet<>();
3     static {
4         VALID_NUMS.add(0);
5         VALID_NUMS.add(1);
6         VALID_NUMS.add(6);
7         VALID_NUMS.add(8);
8         VALID_NUMS.add(9);
9     }
10    public boolean confusingNumber(int N) {
11        if (N < 0) {
12            return false;
13        }
14        int n = N;
15        int ret = 0;
16        while (n != 0) {
17            int tmp = n % 10;
18            if (!VALID_NUMS.contains(tmp)) {
19                return false;
20            }
21            if (tmp == 6) {
22                tmp = 9;
23            } else if (tmp == 9) {
24                tmp = 6;
25            }
26            ret = ret * 10 + tmp;
27            n /= 10;
28        }
29        if (ret == N) {
30            return false;
31        }
32        return true;
33    }
34 }
35 }
```

## 1088. 易混淆数 II

### 1088. 易混淆数 II

难度 困难 ⚡ 8 ❤️ ⚡ 🌐 ⚡ 🔍

本题我们会将数字旋转  $180^\circ$  来生成一个新的数字。

比如 0、1、6、8、9 旋转  $180^\circ$  以后，我们得到的新数字分别为 0、1、9、8、6。

2、3、4、5、7 旋转  $180^\circ$  后，是无法得到任何数字的。

易混淆数（Confusing Number）指的是一个数字在整体旋转  $180^\circ$  以后，能够得到一个和原来不同的数，且新数字的每一位都应该是有效的。（请注意，旋转后得到的新数字可能大于原数字）

给出正整数  $N$ ，请你返回 1 到  $N$  之间易混淆数字的数量。

示例 1：

```
输入: 20
输出: 6
解释:
易混淆数为 [6,9,10,16,18,19]。
6 转换为 9
9 转换为 6
10 转换为 01 也就是 1
16 转换为 91
18 转换为 81
19 转换为 61
```

示例 2：

```
输入: 100
输出: 19
解释:
易混淆数为
[6,9,10,16,18,19,60,61,66,68,80,81,86,89,90,91,98,99,
```

## 1064. 不动点

### 1064. 不动点

难度 简单 ⚡ 9 ❤️ ⚡ 🌐 ⚡ 🔍

给定已经按升序排列、由不同整数组成的数组  $A$ ，返回满足  $A[i] == i$  的最小索引  $i$ 。如果不存在这样的  $i$ ，返回 -1。

## 1058. 最小化舍入误差以满足目标

### 1058. 最小化舍入误差以满足目标

难度 中等 ⚡ 10 ❤️ ⚡ 🌐 ⚡ 🔍

给定一系列价格  $[p_1, p_2, \dots, p_n]$  和一个目标  $target$ ，将每个价格  $p_i$  舍入为  $\text{Round}_i(p_i)$  以便得舍入数组  $[\text{Round}_1(p_1), \text{Round}_2(p_2), \dots, \text{Round}_n(p_n)]$  之和达到给定的目标值  $target$ 。每次舍入操作  $\text{Round}_i(p_i)$  可以是向下舍  $\text{Floor}(p_i)$  也可以是向上入  $\text{Ceil}(p_i)$ 。

如果舍入数组之和无论如何都无法达到目标值  $target$ ，就返回 -1。否则，以保留到小数点后三位的字符串格式返回最小的舍入误差，其定义为  $\sum |\text{Round}_i(p_i) - (p_i)|$  ( $i$  从 1 到  $n$ )。

示例 1：

```
输入: prices = ["0.700","2.800","4.900"], target = 8
输出: "1.000"
解释:
使用 Floor, Ceil 和 Ceil 操作得到 (0.7 - 0) + (3 - 2.8) + (5 - 4.9) = 0.7 + 0.2 + 0.1 = 1.0。
```

示例 2：

```
输入: prices = ["1.500","2.500","3.500"], target = 10
输出: "-1"
```

```
1 class Solution {
2     private final int[] digits = new int[]{0, 1, 6, 8, 9};
3     public int confusingNumberII(int N) {
4         int ans = 0;
5         for (int idx = 1; idx <= 4; idx++) {
6             long num = digits[idx];
7             ans += dfs(N, num);
8         }
9         return ans;
10    }
11    private int dfs(int max, long num) {
12        if (num > max) {
13            return 0;
14        }
15        int res = 0;
16        if (isValid(num)) {
17            res++;
18        }
19        for (int digit : digits) {
20            num *= 10;
21            num += digit;
22            res += dfs(max, num);
23            num /= 10;
24        }
25        return res;
26    }
27    private boolean isValid(long num) {
28        long tmp = num;
29        long sum = 0;
30        while (tmp > 0) {
31            sum *= 10;
32            int digit = (int) (tmp % 10);
33            if (digit == 6) {
34                sum += 9;
35            } else if (digit == 9) {
36                sum += 6;
37            } else {
38                sum += digit;
39            }
40            tmp /= 10;
41        }
42        return num != sum;
43    }
44}
```

```
1 class Solution {
2     public int fixedPoint(int[] A) {
3         for (int idx = 0; idx < A.length; idx++) {
4             if (A[idx] == idx) return idx;
5         }
6         return -1;
7     }
8 }
```

```
1 class Solution {
2     public String minimizeError(String[] prices, int target) {
3         double[] decimals = new double[prices.length];
4         int len = 0;
5         for (String price : prices) {
6             double value = Double.parseDouble(price);
7             int intPart = (int) value;
8             double decPart = value - intPart;
9             if (decPart < 1e-6) {
10                 target = target - intPart;
11                 continue;
12             }
13             decimals[len++] = decPart;
14             target = target - intPart;
15         }
16         if (target < 0 || target > len) {
17             return "-1";
18         }
19         Arrays.sort(decimals, 0, len);
20         double decSum = 0;
21         int low = len - target;
22         for (int idx = 0; idx < len; idx++) {
23             if (idx < low) {
24                 decSum = decSum + decimals[idx];
25             } else {
26                 decSum = decSum + 1.0 - decimals[idx];
27             }
28         }
29         return String.format("%.3f", decSum);
30     }
31 }
32 }
```

## 1057. 校园分配自行车

### 1057. 校园自行车分配

难度 中等 ⚡ 24 ❤️ 💬 📄

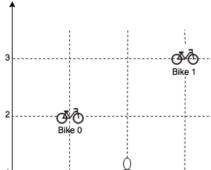
在由 2D 网格表示的校园里有  $n$  位工人 (worker) 和  $m$  辆自行车 (bike)， $n \leq m$ 。所有工人和自行车的位置都用网格上的 2D 坐标表示。

我们需要为每位工人分配一辆自行车。在所有可用的自行车和工人中，我们选取彼此之间曼哈顿距离最短的工人自行车对 (worker, bike)，并将其中的自行车分配给工人。如果有多个 (worker, bike) 对之间的曼哈顿距离相同，那么我们选择工人索引最小的那个。类似地，如果有多种不同的分配方法，则选择自行车索引最小的一对。不断重复这一过程，直到所有工人都分配到自行车为止。

给定两点  $p_1$  和  $p_2$  之间的曼哈顿距离为  $\text{Manhattan}(p_1, p_2) = |p_1.x - p_2.x| + |p_1.y - p_2.y|$ 。

返回长度为  $n$  的向量  $\text{ans}$ ，其中  $\text{a}[i]$  是第  $i$  位工人分配到的自行车的索引 (从 0 开始)。

示例 1：



```

1 v class Solution {
2 v     public int[] assignBikes(int[][] workers, int[][] bikes) {
3 v         int[] ans = new int[workers.length];
4 v         Arrays.fill(ans, -1);
5 v         boolean[] bikeAssigned = new boolean[bikes.length];
6 v         TreeMap<Integer, ArrayList<int[]>> map = new TreeMap<O>;
7 v         for (int wIdx = 0; wIdx < workers.length; wIdx++) {
8 v             for (int bIdx = 0; bIdx < bikes.length; bIdx++) {
9 v                 int dist = Math.abs(workers[wIdx][0] - bikes[bIdx][0])
10 v                     + Math.abs(workers[wIdx][1] - bikes[bIdx][1]);
11 v                 ArrayList<int[]> list = map.getOrDefault(dist, new ArrayList<O>());
12 v                 list.add(new int[]{wIdx, bIdx});
13 v                 map.put(dist, list);
14 v             }
15 v         }
16 v         int cnt = 0;
17 v         for (Map.Entry<Integer, ArrayList<int[]>> entry : map.entrySet()) {
18 v             ArrayList<int[]> list = entry.getValue();
19 v             int len = list.size();
20 v             for (int idx = 0; idx < len; idx++) {
21 v                 int wIdx = list.get(idx)[0];
22 v                 int bIdx = list.get(idx)[1];
23 v                 if (ans[wIdx] >= 0 || bikeAssigned[bIdx]) {
24 v                     continue;
25 v                 }
26 v                 ans[wIdx] = bIdx;
27 v                 bikeAssigned[bIdx] = true;
28 v                 if (++cnt == workers.length) {
29 v                     break;
30 v                 }
31 v             }
32 v         }
33 v     }
34 v     return ans;
35 v }
```

## 1066. 校园分配自行车 II

### 1066. 校园自行车分配 II

难度 中等 ⚡ 27 ❤️ 💬 📄

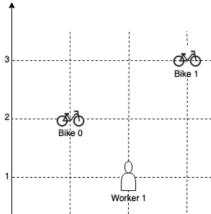
在由 2D 网格表示的校园里有  $n$  位工人 (worker) 和  $m$  辆自行车 (bike)， $n \leq m$ 。所有工人和自行车的位置都用网格上的 2D 坐标表示。

我们为每一位工人分配一辆专属自行车，使每个工人与其分配到的自行车之间的曼哈顿距离最小化。

$p_1$  和  $p_2$  之间的曼哈顿距离为  $\text{Manhattan}(p_1, p_2) = |p_1.x - p_2.x| + |p_1.y - p_2.y|$ 。

返回每个工人与分配到的自行车之间的曼哈顿距离的最小可能总和。

示例 1：



```

1 v class Solution {
2 v     public int assignBikes(int[][] workers, int[][] bikes) {
3 v         int wLen = workers.length;
4 v         int bLen = bikes.length;
5 v         int[][] dp = new int[wLen][1 << bLen];
6 v         for (int idx = 0; idx < wLen; ++idx) {
7 v             Arrays.fill(dp[idx], -1);
8 v         }
9 v         return dfs(0, 0, workers, bikes, dp);
10 v     }
11 v
12 v     int dfs(int wIdx, int bIdx, int[][] workers, int[][] bikes, int[][] dp) {
13 v         if (wIdx >= workers.length) {
14 v             return 0;
15 v         }
16 v         if (dp[wIdx][bIdx] != -1) {
17 v             return dp[wIdx][bIdx];
18 v         }
19 v         int min = 1 << 30;
20 v         for (int idx = 0; idx < bikes.length; ++idx) {
21 v             if ((bIdx >> idx & 1) == 1) {
22 v                 continue;
23 v             }
24 v             int dist = Math.abs(workers[wIdx][0] - bikes[idx][0])
25 v                 + Math.abs(workers[wIdx][1] - bikes[idx][1]);
26 v             int next = bIdx | 1 << idx;
27 v             min = Math.min(min, dist + dfs(wIdx + 1, next, workers, bikes, dp));
28 v         }
29 v         dp[wIdx][bIdx] = min;
30 v         return min;
31 v     }
32 v }
```

## 1059. 从始点到终点的所有路径

### 1059. 从始点到终点的所有路径

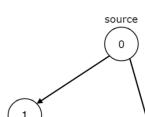
难度 中等 ⚡ 10 ❤️ 💬 📄

给定有向图的边 edges，以及该图的始点 source 和目标终点 destination，确定从始点 source 出发的所有路径是否最终结束于目标终点 destination，即：

- 从始点 source 到目标终点 destination 存在至少一条路径。
- 如果存在从始点 source 到没有出边的节点的路径，则该节点就是路径终点。
- 从始点 source 到目标终点 destination 可能路径数是有限数字。

当从始点 source 出发的所有路径都可以到达目标终点 destination 时返回 true，否则返回 false。

示例 1：



```

1 v class Solution {
2 v     public boolean leadsToDestination(int n, int[][] edges, int source, int destination) {
3 v         List<Integer> adjList = new ArrayList[n];
4 v         for (int idx = 0; idx < n; idx++) {
5 v             adjList[idx] = new ArrayList<O>;
6 v         }
7 v         for (int[] edge : edges) {
8 v             adjList[edge[0]].add(edge[1]);
9 v         }
10 v        return dfs(adjList, source, destination, new boolean[n], new boolean[n]);
11 v    }
12 v
13 v    public boolean dfs(List<Integer> adjList, int src, int dst, boolean[] visited, boolean[] isOnPath) {
14 v        if (adjList[src].size() == 0) {
15 v            return src == dst;
16 v        }
17 v        visited[src] = true;
18 v        isOnPath[src] = true;
19 v        for (int vex : adjList[src]) {
20 v            if ((isOnPath[vex] || (visited[vex] && !dfs(adjList, vex, dst, visited, isOnPath))) {
21 v                return false;
22 v            }
23 v        }
24 v        isOnPath[src] = false;
25 v        return true;
26 v    }
27 v }
```

## 1060. 有序数组中的缺失元素

### 1060. 有序数组中的缺失元素

难度 中等 ⚡ 12 ❤️ 🔍 文章 ⌂

给出一个有序数组  $A$ ，数组中的每个数字都是独一无二的，找出从数组最左边开始的第  $K$  个缺失数字。

示例 1：

输入:  $A = [4, 7, 9, 10]$ ,  $K = 1$   
输出: 5  
解释:  
第一个缺失数字为 5。

示例 2：

输入:  $A = [4, 7, 9, 10]$ ,  $K = 3$   
输出: 8  
解释:  
缺失数字有  $[5, 6, 8, \dots]$ ，因此第三个缺失数字为 8。

```
1 v class Solution {
2 v     public int missingElement(int[] nums, int k) {
3 v         int len = nums.length;
4 v         if (k > miss(len - 1, nums)) {
5 v             return nums[len - 1] + k - miss(len - 1, nums);
6 v         }
7 v         int mid;
8 v         int lid = 0;
9 v         int rid = len - 1;
10 v        while (lid != rid) {
11 v            mid = lid + (rid - lid) / 2;
12 v            if (miss(mid, nums) < k) {
13 v                lid = mid + 1;
14 v            } else {
15 v                rid = mid;
16 v            }
17 v        }
18 v        return nums[lid - 1] + k - miss(lid - 1, nums);
19 v    }
20 v
21 v    private int miss(int idx, int[] nums) {
22 v        return nums[idx] - nums[0] - idx;
23 v    }
24 v
25 v }
```

## 1061. 按字典序排列最小的等效字符串

### 1061. 按字典序排列最小的等效字符串

难度 中等 ⚡ 3 ❤️ 🔍 文章 ⌂

给出长度相同的两个字符串： $A$  和  $B$ ，其中  $A[i]$  和  $B[i]$  是一组等价字符。举个例子，如果  $A = "abc"$  且  $B = "cde"$ ，那么就有  $'a' == 'c'$ ,  $'b' == 'd'$ ,  $'c' == 'e'$ 。

等价字符遵循任何等价关系的一般规则：

- 自反性:  $'a' == 'a'$
- 对称性:  $'a' == 'b'$  则必定有  $'b' == 'a'$
- 传递性:  $'a' == 'b'$  且  $'b' == 'c'$  就表明  $'a' == 'c'$

例如， $A$  和  $B$  的等价信息和之前的例子一样，那么  $S = "eed"$ ,  $"acd"$  或  $"aab"$ ，这三个字符串都是等价的，而  $"aab"$  是  $S$  的按字典序最小的等价字符串。

利用  $A$  和  $B$  的等价信息，找出并返回  $S$  的按字典序排列最小的等价字符串。

示例 1：

输入:  $A = "parker"$ ,  $B = "morris"$ ,  $S = "parser"$   
输出: "makkek"  
解释: 根据  $A$  和  $B$  中的等价信息，我们可以将这些字符分为  $[m,p]$ ,  $[a,o]$ ,  $[k,r,s]$ ,  $[e,i]$  共 4 组。每组中的字符都是等价的，并按字典序排列。所以答案是 "makkek"。

示例 2：

```
1 v class Solution {
2 v     public String smallestEquivalentString(String A, String B, String S) {
3 v         final int num = 26;
4 v         int[] roots = new int[num];
5 v         for (int idx = 0; idx < num; idx++) {
6 v             roots[idx] = idx;
7 v         }
8 v         for (int idx = 0; idx < A.length(); idx++) {
9 v             int aRoot = find(A.charAt(idx) - 'a', roots);
10 v            int bRoot = find(B.charAt(idx) - 'a', roots);
11 v            if (aRoot == bRoot) {
12 v                continue;
13 v            }
14 v            if (aRoot < bRoot) {
15 v                roots[bRoot] = aRoot;
16 v            } else {
17 v                roots[aRoot] = bRoot;
18 v            }
19 v        }
20 v        StringBuilder buffer = new StringBuilder(S.length());
21 v        for (int idx = 0; idx < S.length(); idx++) {
22 v            int root = find(S.charAt(idx) - 'a', roots);
23 v            buffer.append((char) (root + 'a'));
24 v        }
25 v        return buffer.toString();
26 v    }
27 v
28 v    private int find(int index, int[] roots) {
29 v        while (index != roots[index]) {
30 v            index = roots[index];
31 v        }
32 v        return index;
33 v    }
34 v }
```

## 1062. 最长重复子串

### 1062. 最长重复子串

难度 中等 ⚡ 16 ❤️ 🔍 文章 ⌂

给定字符串  $S$ ，找出最长重复子串的长度。如果不存在重复子串就返回 0。

示例 1：

输入: "abcd"  
输出: 0  
解释: 没有重复子串。

示例 2：

输入: "abbaba"  
输出: 2  
解释: 最长的重复子串为 "ab" 和 "ba"，每个出现 2 次。

示例 3：

输入: "aabcaabdaab"  
输出: 3  
解释: 最长的重复子串为 "aab"，出现 3 次。

```
1 v class Solution {
2 v     public int longestRepeatingSubstring(String S) {
3 v         int len = S.length();
4 v         int lid = 1;
5 v         int rid = len;
6 v         while (lid <= rid) {
7 v             int mid = lid + (rid - lid) / 2;
8 v             if (search(mid, len, S) != -1) {
9 v                 lid = mid + 1;
10 v            } else {
11 v                rid = mid - 1;
12 v            }
13 v        }
14 v        return lid - 1;
15 v    }
16 v
17 v    private int search(int mid, int len, String string) {
18 v        HashSet<Integer> set = new HashSet();
19 v        for (int idx = 0; idx < len - mid + 1; idx++) {
20 v            String sub = string.substring(idx, idx + mid);
21 v            int hash = sub.hashCode();
22 v            if (set.contains(hash)) {
23 v                return idx;
24 v            }
25 v            set.add(hash);
26 v        }
27 v        return -1;
28 v    }
29 v
30 v }
```

## 1065. 字符串的索引对

1065. 字符串的索引对

难度 简单 ⚡ 8 ❤ 1 ⚡ 文 ⚡ 单 ⚡ 四

给出字符串 `text` 和字符串列表 `words`, 返回所有的索引对 `[i, j]` 使得在索引对范围内的子字符串 `text[i]...text[j]` (包括 `i` 和 `j`) 属于字符串列表 `words`。

示例 1:

```
输入: text = "thestoryofleetcodeandme", words =  
["story","fleet","leetcode"]  
输出: [[3,7],[9,13],[10,17]]
```

```
1 * class Solution {  
2 *     public int[] indexPairs(String text, String[] words) {  
3 *         ArrayList<int[]> list = new ArrayList<>();  
4 *         for (String word : words) {  
5 *             int len = word.length();  
6 *             int idx = text.indexOf(word, 0);  
7 *             while (idx != -1) {  
8 *                 list.add(new int[]{idx, idx + len - 1});  
9 *                 idx = text.indexOf(word, idx + 1);  
10 *            }  
11 *        }  
12 *        int[] ans = list.toArray(new int[0][]);  
13 *        Arrays.sort(ans, (o1, o2) -> o1[0] == o2[0] ? o1[1] - o2[1] : o1[0] - o2[0]);  
14 *        return ans;  
15 *    }  
16 *
```

## 1067. 范围内的数字计数

1067. 范围内的数字计数

难度 困难 ⚡ 8 ❤ 1 ⚡ 文 ⚡ 单 ⚡ 四

给定一个在 `0` 到 `9` 之间的整数 `d`, 和两个正整数 `low` 和 `high` 分别作为上下界。返回 `d` 在 `low` 和 `high` 之间的整数中出现的次数, 包括边界 `low` 和 `high`。

示例 1:

```
输入: d = 1, low = 1, high = 13  
输出: 6  
解释:  
数字 d=1 在 1,10,11,12,13 中出现 6 次。注意 d=1 在数字  
11 中出现两次。
```

示例 2:

```
输入: d = 3, low = 100, high = 250  
输出: 35  
解释:  
数字 d=3 在 103,113,123,130,131,...,238,239,243 出现  
35 次。
```

## 1085. 最小元素各数位之和

1085. 最小元素各数位之和

难度 简单 ⚡ 1 ❤ 1 ⚡ 文 ⚡ 单 ⚡ 四

给你一个正整数的数组 `A`。

然后计算 `s`, 使其等于数组 `A` 当中最小的那个元素各个数位上数字之和。

最后, 假如 `s` 所得计算结果是 奇数 的请你返回 `0`, 否则请返回 `1`。

```
1 * class Solution {  
2 *     public int digitsCount(int d, int low, int high) {  
3 *         return count(high, d) - count(low - 1, d);  
4 *     }  
5 *  
6 *     /* 计算数字 d 在 1-n 中出现的次数 */  
7 *     public int count(int n, int d) {  
8 *         int cnt = 0;  
9 *         for (int i = 1, k = 0; (k = n / i) != 0; i *= 10) {  
10 *             int hig = k / 10;  
11 *             if (d == 0) {  
12 *                 if (hig != 0) {  
13 *                     hig--;  
14 *                 } else {  
15 *                     break;  
16 *                 }  
17 *             }  
18 *             cnt += hig * i;  
19 *             int cur = k % 10;  
20 *             if (cur > d) {  
21 *                 cnt += i;  
22 *             } else if (cur == d) {  
23 *                 cnt += n - k * i + 1;  
24 *             }  
25 *         }  
26 *         return cnt;  
27 *     }  
28 * }
```

## 1086. 前五科的均分

1086. 前五科的均分

难度 简单 ⚡ 8 ❤ 1 ⚡ 文 ⚡ 单 ⚡ 四

给你一个不同学生的分数列表, 请按 学生的 id 顺序 返回每个学生 最高的五科 成绩的 平均分。

对于每条 `items[i]` 记录, `items[i][0]` 为学生的 id, `items[i][1]` 为学生的分数。平均分请采用整数除法计算。

示例:

```
输入: [[1,91],[1,92],[2,93],[2,97],[1,60],[2,77],  
[1,65],[1,87],[1,100],[2,100],[2,76]]  
输出: [[1,87],[2,88]]  
解释:  
id = 1 的学生平均分为 87。  
id = 2 的学生平均分为 88.6。但由于整数除法的缘故, 平均分会  
被转换为 88。
```

```
1 * class Solution {  
2 *     public int sumOfDigits(int[] A) {  
3 *         int min = Integer.MAX_VALUE;  
4 *         for (int idx = 0; idx < A.length; idx++) {  
5 *             min = Math.min(min, A[idx]);  
6 *         }  
7 *         int sum = 0;  
8 *         while (min != 0) {  
9 *             sum += (min % 10);  
10 *            min /= 10;  
11 *        }  
12 *        return (sum % 2 == 0) ? 1 : 0;  
13 *    }  
14 *
```

```
1 * class Solution {  
2 *     public int[] highFive(int[][] items) {  
3 *         HashMap<Integer, PriorityQueue<Integer>> map = new HashMap<>();  
4 *         for (int idx = 0; idx < items.length; idx++) {  
5 *             PriorityQueue<Integer> heap = map.getOrDefault(items[idx][0],  
6 *                         new PriorityQueue<>((o1, o2) -> o1 - o2));  
7 *             heap.offer(items[idx][1]);  
8 *             if (heap.size() > 5) {  
9 *                 heap.poll();  
10 *            }  
11 *            map.put(items[idx][0], heap);  
12 *        }  
13 *        ArrayList<int[]> list = new ArrayList<>();  
14 *        for (Map.Entry<Integer, PriorityQueue<Integer>> entry : map.entrySet()) {  
15 *            int key = entry.getKey();  
16 *            int val = (int) entry.getValue().stream().mapToInt((x) -> x)  
17 *                .summaryStatistics().getAverage();  
18 *            list.add(new int[]{key, val});  
19 *        }  
20 *        Collections.sort(list, (o1, o2) -> o1[0] - o2[0]);  
21 *        return list.toArray(new int[0][]);  
22 *    }  
23 *
```

## 1087. 字母切换

### 1087. 字母切换

难度 中等 ⚡ 15 ❤️ 🔍 文档 举报

我们用一个特殊的字符串  $S$  来表示一份单词列表，之所以能展开成为一个列表，是因为这个字符串  $S$  中存在一个叫做「选项」的概念：

单词中的每个字母可能只有一个选项或存在多个备选项。如果只有一个选项，那么该字母按原样表示。

如果存在多个选项，就会以花括号包裹来表示这些选项（使它们与其他字母分隔开），例如  $\{a,b,c\}$  表示  $\{ "a", "b", "c" \}$ 。

例子： $\{a,b,c\}d\{e,f\}$  可以表示单词列表  $\{ "ade", "adf", "bde", "bdf", "cde", "cdf" \}$ 。

请你按字典顺序，返回所有以这种方式形成的单词。

示例 1：

输入：  $\{a,b\}c\{d,e\}f$   
输出：  $\{ "acdf", "acef", "bcdf", "bcef" \}$

示例 2：

输入：  $"abcd"$   
输出：  $\{ "abcd" \}$

```
1 v class Solution {
2 v     public String[] expand(String S) {
3 v         if (S.indexOf("{") < 0)
4 v             return new String[]{S};
5 v         int len = S.length();
6 v         List<String> list = new ArrayList<>();
7 v         backTrace(S, new StringBuilder(), 0, list);
8 v         Collections.sort(list);
9 v         return list.toArray(new String[list.size()]);
10 v    }
11 v    private void backTrace(String str, StringBuilder buf, int idx, List<String> list) {
12 v        if (idx == str.length())
13 v            list.add(buf.toString());
14 v            return;
15 v        if (str.charAt(idx) == '{') {
16 v            int cnt = 0;
17 v            for (int i = idx + 1; str.charAt(i) != '}'; i++) {
18 v                cnt++;
19 v                for (int j = idx + 1; str.charAt(j) != '}'; j++) {
20 v                    char ch = str.charAt(j);
21 v                    if (ch != ',')
22 v                        buf.append(ch);
23 v                    backTrace(str, buf, idx + cnt + 2, list);
24 v                    buf.deleteCharAt(buf.length() - 1);
25 v                }
26 v            }
27 v        } else {
28 v            buf.append(str.charAt(idx));
29 v            backTrace(str, buf, idx + 1, list);
30 v            buf.deleteCharAt(buf.length() - 1);
31 v        }
32 v    }
33 }
```

## 1099. 小于 K 的两数之和

### 1099. 小于 K 的两数之和

难度 简单 ⚡ 12 ❤️ 🔍 文档 举报

给你一个整数数组  $A$  和一个整数  $K$ ，请在该数组中找出两个元素，使它们的和小于  $K$  但尽可能地接近  $K$ ，返回这两个元素的和。

如不存在这样的两个元素，请返回  $-1$ 。

示例 1：

输入：  $A = [34, 23, 1, 24, 75, 33, 54, 8]$ ,  $K = 60$   
输出： 58  
解释：  
34 和 24 相加得到 58，58 小于 60，满足题意。

示例 2：

```
1 v class Solution {
2 v     public int twoSumLessThanK(int[] A, int K) {
3 v         Arrays.sort(A);
4 v         int ans = -1, min = Integer.MAX_VALUE;
5 v         int lid = 0, rid = A.length - 1;
6 v         while (lid < rid) {
7 v             if (A[lid] >= K)
8 v                 break;
9 v             int sum = A[lid] + A[rid];
10 v            if (sum >= K) {
11 v                --rid;
12 v            } else {
13 v                if (min > K - sum) {
14 v                    min = K - sum;
15 v                    ans = sum;
16 v                }
17 v                ++lid;
18 v            }
19 v        }
20 v        return ans;
21 v    }
22 }
```

## 1100. 长度为 K 的无重复字符子串

### 1100. 长度为 K 的无重复字符子串

难度 中等 ⚡ 12 ❤️ 🔍 文档 举报

给你一个字符串  $S$ ，找出所有长度为  $K$  且不含重复字符的子串，请你返回全部满足要求的子串的数目。

示例 1：

输入：  $S = "havefunonleetcode"$ ,  $K = 5$   
输出： 6  
解释：  
这里有 6 个满足题意的子串，分别是：  
 $'havef'$ ,  $'avefu'$ ,  $'vefun'$ ,  $'efuno'$ ,  $'etcod'$ ,  $'tcode'$ 。

示例 2：

输入：  $S = "home"$ ,  $K = 5$   
输出： 0

```
1 v class Solution {
2 v     public int numKLenSubstrNoRepeats(String S, int K) {
3 v         int len = S.length();
4 v         if (len < K)
5 v             return 0;
6 v         int ans = 0, lid = 0, rid = 0;
7 v         int[] cnts = new int[26];
8 v         while (rid < len) {
9 v             int idx = S.charAt(rid) - 'a';
10 v            cnts[idx]++;
11 v            while (cnts[idx] > 1) {
12 v                cnts[S.charAt(lid++) - 'a']--;
13 v            }
14 v            if (rid - lid + 1 == K) {
15 v                ans++;
16 v                cnts[S.charAt(lid++) - 'a']--;
17 v            }
18 v            rid++;
19 v        }
20 v        return ans;
21 v    }
22 }
```

## 1101. 彼此熟识的最早时间

### 1101. 彼此熟识的最早时间

难度 中等 ⚡ 5 ❤ 1 ⚡ ✅

在一个社交圈子当中，有 `N` 个人。每个人都只有一个从 `0` 到 `N-1` 唯一的 id 编号。

我们有一份日志列表 `logs`，其中每条记录都包含一个非负整数的时间戳，以及分属两个人的不同 id，`logs[i] = [timestamp, id_A, id_B]`。

每条日志标识出两个人成为好友的时间，友谊是相互的：如果 A 和 B 是好友，那么 B 和 A 也是好友。

如果 A 是 B 的好友，或者 A 是 B 的好友的好友，那么就可以认为 A 也与 B 熟识。

返回圈子里所有人之间都熟识的最早时间。如果找不到最早时间，就返回 `-1`。

示例：

```
输入: logs = [[20190101,0,1],[20190104,3,4],[20190107,2,3],[20190211,1,5],[20190224,2,4],[20190301,0,3],[20190312,1,2],[20190322,4,5]], N = 6
输出: 20190301
解释:
第一次结交发生在 timestamp = 20190101, 0 和 1 成为好友,
社交媒体圈如下 [0,1], [2], [3], [4], [5]。
第二次结交发生在 timestamp = 20190104, 3 和 4 成为好友,
社交媒体圈如下 [0,1], [2], [3,4], [5]。
第三次结交发生在 timestamp = 20190107, 2 和 3 成为好友,
```

## 1118. 一个月有多少天

### 1118. 一个月有多少天

难度 简单 ⚡ 4 ❤ 1 ✅

指定年份 `Y` 和月份 `M`，请你帮忙计算出该月一共有多少天。

示例 1：

```
1 class Solution {
2     public int numberOfDays(int Y, int M) {
3         int[] leapYear = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // 闰年
4         int[] commonYear = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // 非闰年
5         if ((Y % 100 != 0 && Y % 4 == 0) || Y % 400 == 0) {
6             return leapYear[M-1];
7         }
8         return commonYear[M-1];
9     }
10 }
```

## 1119. 删去字符串中的元音

### 1119. 删去字符串中的元音

难度 简单 ⚡ 6 ❤ 1 ✅

给你一个字符串 `s`，请你删去其中的所有元音字母（'a'，'e'，'i'，'o'，'u'），并返回这个新字符串。

```
1 class Solution {
2     public String removeVowels(String S) {
3         char[] chars = {'a', 'e', 'i', 'o', 'u'};
4         for (char ch : chars) {
5             String str = String.valueOf(ch);
6             S = S.replaceAll(str, "");
7         }
8     }
9 }
```

## 1120. 子树的最大平均值

### 1120. 子树的最大平均值

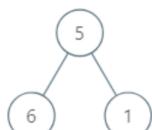
难度 中等 ⚡ 10 ❤ 1 ✅

给你一棵二叉树的根节点 `root`，找出这棵树的每一棵子树的平均值中的最大值。

子树是树中的任意节点和它的所有后代构成的集合。

树的平均值是树中节点值的总和除以节点数。

示例：



输入: [5,6,1]

输出: 6.00000

```
1 class Solution {
2     public double maximumAverageSubtree(TreeNode root) {
3         if (root == null) {
4             return 0;
5         }
6         double[] ans = new double[1];
7         dfs(root, ans);
8         return ans[0];
9     }
10
11     private int[] dfs(TreeNode root, double[] ans) {
12         int[] tmp = new int[2];
13         if (root == null) {
14             return tmp;
15         }
16         int[] lft = dfs(root.left, ans);
17         int[] rgt = dfs(root.right, ans);
18         // 设置当前树的元素和
19         tmp[0] = lft[0] + rgt[0] + root.val;
20         // 设置节点个数
21         tmp[1] = lft[1] + rgt[1] + 1;
22         // 更新平均值
23         ans[0] = Math.max(ans[0], (double) tmp[0] / tmp[1]);
24     }
25 }
```

## 1102. 得分最高的路径

### 1102. 得分最高的路径

难度 中等 通过 23 提交 答案

给你一个 R 行 C 列的整数矩阵 A。矩阵上的路径从 [0, 0] 开始，在 [R-1, C-1] 结束。

路径沿四个基本方向（上、下、左、右）展开，从一个已访问单元格移动到任一相邻的未访问单元格。

路径的得分是该路径上的最小值。例如，路径 8 → 4 → 5 → 9 的值为 4。

找出所有路径中得分 最高 的那条路径，返回其 得分。

示例 1：

5	4	5
1	2	6
7	4	6

输入：[[5,4,5],[1,2,6],[7,4,6]]

输出：4

解释：

得分最高的路径用黄色突出显示。

示例 2：

2	2	1	2	2	2
1	2	2	2	1	2

输入：[[2,2,1,2,2,2],[1,2,2,2,1,2]]

输出：2

```
1 class Solution {
2     public int maximumMinimumPath(int[][] A) {
3         int ans = 0, rows = A.length, cols = A[0].length;
4         int lo = 1, hi = Math.min(A[0][0], A[rows - 1][cols - 1]);
5         int[] dirs = new int[]{0, 1, 0, -1, 0};
6         while (lo <= hi) {
7             int mid = (lo + hi) / 2;
8             boolean[][] visited = new boolean[rows][cols];
9             if (dfs(A, visited, dirs, mid, 0, 0, rows, cols)) {
10                 ans = mid;
11                 lo = mid + 1;
12             } else {
13                 hi = mid - 1;
14             }
15         }
16         return ans;
17     }
18     private boolean dfs(int[][] grid, boolean[][] visited, int[] dirs,
19                         int max, int rid, int cid, int rows, int cols) {
20         visited[rid][cid] = true;
21         if (rid == rows - 1 && cid == cols - 1) {
22             return true;
23         }
24         for (int k = 0; k < 4; k++) {
25             int rIdx = rid + dirs[k];
26             int cIdx = cid + dirs[k + 1];
27             if (rIdx < 0 || rIdx >= rows || cIdx < 0 || cIdx >= cols
28                 || visited[rIdx][cIdx] || grid[rIdx][cIdx] < max) {
29                 continue;
30             }
31             if (dfs(grid, visited, dirs, max, rIdx, cIdx, rows, cols)) {
32                 return true;
33             }
34         }
35         return false;
36     }
37 }
38 }
```

## 1121. 将数组分成几个递增序列

### 1121. 将数组分成几个递增序列

难度 困难 通过 13 提交 答案

给你一个 非递减 的正整数数组 nums 和整数 K，判断该数组是否可以被分成一个或几个 长度至少 为 K 的 不相交的递增子序列。

示例 1：

输入：nums = [1,2,2,3,3,4,4], K = 3

输出：true

解释：

该数组可以分成两个子序列 [1,2,3,4] 和 [2,3,4]，每个子序列的长度都至少是 3。

```
1 class Solution {
2     public boolean canDivideIntoSubsequences(int[] nums, int K) {
3         if (K == 1)
4             return true;
5         int cnt = 0, len = nums.length, pre = nums[0];
6         for (int idx = 1; idx < len; idx++) {
7             if (pre == nums[idx])
8                 cnt++;
9             else {
10                 if (cnt * K > len)
11                     return false;
12                 pre = nums[idx];
13                 cnt = 1;
14             }
15         }
16         return cnt * K <= len;
17     }
18 }
```

## 1133. 最大唯一数

### 1133. 最大唯一数

难度 简单 通过 7 提交 答案

给你一个整数数组 A，请找出并返回在该数组中仅出现一次的最大整数。

如果不存在这个只出现一次的整数，则返回 -1。

示例 1：

## 1134. 阿姆斯特朗数

### 1134. 阿姆斯特朗数

难度 简单 通过 4 提交 答案

假设存在一个 k 位数 N，其每一位上的数字的 k 次幂的总和也是 N，那么这个数是阿姆斯特朗数。

给你一个正整数 N，让你来判定他是否是阿姆斯特朗数，是则返回 true，不是则返回 false。

```
1 class Solution {
2     public int largestUniqueNumber(int[] A) {
3         int[] map = new int[1001];
4         for (int a : A) map[a]++;
5         for (int i = map.length - 1; i >= 0; i--) {
6             if (map[i] != 1) continue;
7             return i;
8         }
9         return -1;
10    }
11 }
```

```
1 class Solution {
2     public boolean isArmstrong(int N) {
3         int len = String.valueOf(N).length();
4         int ans = 0;
5         int tmp = N;
6         while (tmp != 0) {
7             ans += Math.pow(tmp % 10, len);
8             tmp /= 10;
9         }
10        return ans == N;
11    }
12 }
```

## 1135. 最低成本联通所有城市

1135. 最低成本联通所有城市

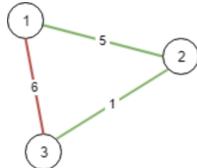
难度 中等 18 热 次数 10

想象一下你是个城市基建规划者，地图上有  $N$  座城市，它们按以 1 到  $N$  的次序编号。

给你一些可连接的选项  $connections$ ，其中每个选项  $connections[i] = [city1, city2, cost]$  表示将城市  $city1$  和城市  $city2$  连接所要的成本。（连接是双向的，也就是说城市  $city1$  和城市  $city2$  相连也同样意味着城市  $city2$  和城市  $city1$  相连）。

返回使得每对城市间都存在将它们连接在一起的连通路径（可能长度为 1 的）最小成本。该最小成本应该是所用全部连接代价的综合。如果根据已知条件无法完成该项任务，则请你返回 -1。

示例 1：



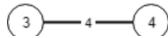
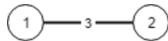
输入:  $N = 3$ ,  $connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]$

输出: 6

解释:

选出任意 2 条边都可以连接所有城市，我们从中选取成本最小的 2 条。

示例 2：



## 1136. 平行课程

1136. 平行课程

难度 困难 7 热 次数 10

已知有  $N$  门课程，它们以 1 到  $N$  进行编号。

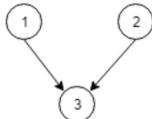
给你一份课程关系表  $relations[i] = [X, Y]$ ，用以表示课程  $X$  和课程  $Y$  之间的先修关系：课程  $X$  必须在课程  $Y$  之前修完。

假设在一个学期里，你可以学习任何数量的课程，但前提是已经学习了将要学习的这些课程的所有先修课程。

请你返回学完全部课程所需的最少学期数。

如果没有办法做到学完全部这些课程的话，就返回 -1。

示例 1：



输入:  $N = 3$ ,  $relations = [[1, 3], [2, 3]]$

输出: 2

解释:

在第一个学期学习课程 1 和 2，在第二个学期学习课程 3。

## 1150. 检查一个数是否在数组中占绝大多数

1150. 检查一个数是否在数组中占绝大多数

难度 简单 16 热 次数 10

给出一个按 非递减 顺序排列的数组  $nums$ ，和一个目标数值  $target$ 。假如数组  $nums$  中绝大多数元素的数值都等于  $target$ ，则返回  $True$ ，否则请返回  $False$ 。

所谓占绝大多数，是指在长度为  $N$  的数组中出现必须超过  $N/2$  次。

```
1 class Solution {
2     public int minimumCost(int N, int[][] connections) {
3         if (N <= 0 || connections.length < N - 1)
4             return -1;
5         Arrays.sort(connections, (a, b) -> a[2] - b[2]);
6         int[] parent = new int[N + 1];
7         for (int idx = 1; idx <= N; ++idx) {
8             parent[idx] = idx;
9         }
10        int ans = 0;
11        for (int[] edge : connections) {
12            if (Union(edge[0], edge[1], parent)) {
13                ans += edge[2];
14            }
15        }
16        int ptr = -1;
17        for (int idx = 1; idx <= N; ++idx) {
18            int root = findRoot(idx, parent);
19            if (ptr == -1) {
20                ptr = root;
21            } else if (ptr != root) {
22                return -1;
23            }
24        }
25        return ans;
26    }
27
28    public int findRoot(int idx, int[] parent) {
29        while (idx != parent[idx]) {
30            parent[idx] = parent[parent[idx]];
31            idx = parent[idx];
32        }
33        return idx;
34    }
35
36    public boolean Union(int vex1, int vex2, int[] parent) {
37        vex1 = findRoot(vex1, parent);
38        vex2 = findRoot(vex2, parent);
39        if (vex1 == vex2)
40            return false;
41        parent[vex1] = vex2;
42        return true;
43    }
44}
```

```
1 class Solution {
2     public int minimumSemesters(int N, int[][] relations) {
3         int[] degrees = new int[N];
4         ArrayList<Integer>[] adj = new ArrayList[N];
5         for (int[] edge : relations) {
6             degrees[edge[1] - 1]++;
7             if (adj[edge[0] - 1] == null)
8                 adj[edge[0] - 1] = new ArrayList<>();
9             adj[edge[0] - 1].add(edge[1] - 1);
10        }
11        Queue<Integer> queue = new LinkedList<>();
12        for (int idx = 0; idx < N; idx++)
13            if (degrees[idx] == 0) queue.offer(idx);
14        if (queue.isEmpty()) return -1; // 存在环
15        int cnt = 0;
16        while (!queue.isEmpty()) {
17            int size = queue.size();
18            for (int i = 0; i < size; i++) {
19                int node = queue.poll();
20                if (adj[node] == null) continue;
21                for (int next : adj[node]) {
22                    degrees[next]--;
23                    if (degrees[next] == 0) queue.offer(next);
24                }
25            }
26            cnt++;
27        }
28        for (int idx = 0; idx < N; idx++)
29            if (degrees[idx] > 0) return -1; // 存在环
30        return cnt;
31    }
32}
```

```
1 class Solution {
2     public boolean isMajorityElement(int[] nums, int target) {
3         int cnt = 0, maj = nums.length / 2 + 1;
4         for (int num : nums) {
5             if (num == target) cnt++;
6         }
7         return cnt >= maj;
8     }
9 }
10
```

## 1151. 最少交换次数来组合所有的 1

### 1151. 最少交换次数来组合所有的 1

难度 中等 ⚡ 10 ❤️ ⌂ ⌂ ⌂ ⌂ ⌂

给出一个二进制数组 `data`，你需要通过交换位置，将数组中 **任何位置** 上的 1 组合到一起，并返回所有可能中所需 **最少** 的交换次数。

示例 1：

```
输入: [1,0,1,0,1]
输出: 1
解释:
有三种可能的方法可以把所有的 1 组合在一起:
[1,1,1,0,0], 交换 1 次;
[0,1,1,1,0], 交换 2 次;
[0,0,1,1,1], 交换 1 次。
所以最少的交换次数为 1。
```

```
1 class Solution { // 滑动窗口
2     public int minSwaps(int[] data) {
3         int sum = 0;
4         for (int num : data)
5             sum += num;
6         if (sum <= 1)
7             return 0;
8         int lid = 0, rid = 0, cur = 0;
9         while (rid < sum) {
10             cur += data[rid];
11             rid++;
12         }
13         int ans = data.length + 1, len = ans - 1;
14         while (rid < len) {
15             ans = Math.min(ans, sum - cur);
16             cur = cur - data[lid] + data[rid];
17             rid++;
18             lid++;
19         }
20     }
21     return ans;
22 }
```

## 1152. 用户网站访问行为分析

### 1152. 用户网站访问行为分析

难度 中等 ⚡ 3 ❤️ ⌂ ⌂ ⌂ ⌂ ⌂

为了评估某网站的用户转化率，我们需要对用户的访问行为进行分析，并建立用户行为模型。日志文件中已经记录了用户名、访问时间以及页面路径。

为了方便分析，日志文件中的 `N` 条记录已经被解析成三个长度相同且长度都为 `N` 的数组，分别是：用户名 `username`，访问时间 `timestamp` 和页面路径 `website`。第 `i` 条记录意味着用户名是 `username[i]` 的用户在 `timestamp[i]` 的时候访问了路径是 `website[i]` 的页面。

我们需要找到用户访问网站时的『共性行为路径』，也就是有最多的用户都至少接某种次序访问过一次的三个页面路径。需要注意的是，用户可能不是连续访问这三个路径的。

『共性行为路径』是一个长度为 3 的页面路径列表，列表中的路径不必不同，并且按照访问时间的先后升序排列。

如果有多个满足要求的答案，那么就请返回字典序排列最小的那个。（页面路径列表 `X` 按字典序小于 `Y` 的前提条件是：`X[0] < Y[0]` 或 `X[0] == Y[0]` 且 `(X[1] < Y[1]` 或 `X[1] == Y[1]` 且 `X[2] < Y[2]`）

题目保证一个用户会至少访问 3 个路径一致的页面，并且一个用户不会在同一时间访问两个路径不同的页面。

示例：

### 1152. 用户网站访问行为分析

难度 中等 ⚡ 3 ❤️ ⌂ ⌂ ⌂ ⌂ ⌂

为了评估某网站的用户转化率，我们需要对用户的访问行为进行分析，并建立用户行为模型。日志文件中已经记录了用户名、访问时间以及页面路径。

为了方便分析，日志文件中的 `N` 条记录已经被解析成三个长度相同且长度都为 `N` 的数组，分别是：用户名 `username`，访问时间 `timestamp` 和页面路径 `website`。第 `i` 条记录意味着用户名是 `username[i]` 的用户在 `timestamp[i]` 的时候访问了路径是 `website[i]` 的页面。

我们需要找到用户访问网站时的『共性行为路径』，也就是有最多的用户都至少接某种次序访问过一次的三个页面路径。需要注意的是，用户可能不是连续访问这三个路径的。

『共性行为路径』是一个长度为 3 的页面路径列表，列表中的路径不必不同，并且按照访问时间的先后升序排列。

如果有多个满足要求的答案，那么就请返回字典序排列最小的那个。（页面路径列表 `X` 按字典序小于 `Y` 的前提条件是：`X[0] < Y[0]` 或 `X[0] == Y[0]` 且 `(X[1] < Y[1]` 或 `X[1] == Y[1]` 且 `X[2] < Y[2]`）

题目保证一个用户会至少访问 3 个路径一致的页面，并且一个用户不会在同一时间访问两个路径不同的页面。

示例：

```
1 class Solution {
2     public List<String> mostVisitedPattern(String[] username, int[] timestamp, String[] website) {
3         int len = timestamp.length; List<Log> logs = new ArrayList<Log>();
4         for (int i = 0; i < len; i++) logs.add(new Log(username[i], timestamp[i], website[i]));
5         Collections.sort(logs, new LogComparator());
6         HashMap<String, List<String>> userVisitMap = new HashMap<String, List<String>>();
7         HashMap<List<String>, Integer> patternTimesMap = new HashMap<List<String>, Integer>();
8         for (int i = 0; i < logs.size(); i++) {
9             List<String> visitPattern = userVisitMap.getOrDefault(logs.get(i).username, new ArrayList<String>());
10            visitPattern.add(logs.get(i).website); userVisitMap.put(logs.get(i).username, visitPattern);
11        }
12        List<String> mostVisitPattern = new ArrayList<String>(); int maxFrequency = Integer.MIN_VALUE;
13        for (String user : userVisitMap.keySet()) {
14            List<String> visitPattern = userVisitMap.get(user);
15            int size = visitPattern.size(); HashSet<List<String>> set = new HashSet<List<String>>();
16            for (int i = 0; i < size - 3; i++) {
17                for (int j = i + 1; j < size - 2; j++) {
18                    for (int k = j + 1; k < size - 1; k++) {
19                        List<String> threeSequence = new ArrayList<String>(); threeSequence.add(visitPattern.get(i));
20                        threeSequence.add(visitPattern.get(j)); threeSequence.add(visitPattern.get(k));
21                        if (set.contains(threeSequence))
22                            continue;
23                        set.add(threeSequence); int times = patternTimesMap.getOrDefault(threeSequence, 0);
24                        times++; patternTimesMap.put(threeSequence, times);
25                        if (times > maxFrequency) {
26                            maxFrequency = times; mostVisitPattern = new ArrayList<String>(threeSequence);
27                        } else if (times == maxFrequency)
28                            mostVisitPattern = new ArrayList<String>(getAlphabetSequence(mostVisitPattern, threeSequence));
29                    }
30                }
31            }
32        }
33        return mostVisitPattern;
34    }
}
private List<String> getAlphabetSequence(List<String> mostVisitPattern, List<String> threeSequence) {
    for (int i = 0; i < 3; i++) {
        int index = 0; String word1 = mostVisitPattern.get(i), word2 = threeSequence.get(i);
        while (index < word1.length() && index < word2.length()) {
            if (word1.charAt(index) < word2.charAt(index))
                return mostVisitPattern;
            else if (word1.charAt(index) > word2.charAt(index))
                return threeSequence;
            index++;
        }
        if (index == word1.length() && index < word2.length())
            return mostVisitPattern;
        if (index == word2.length() && index < word1.length())
            return threeSequence;
    }
    return mostVisitPattern;
}
private class Log {
    String username; int timestamp; String website;
    public Log(String _username, int _timestamp, String _website) {
        username = _username; timestamp = _timestamp; website = _website;
    }
}
private class LogComparator implements Comparator<Log> {
    @Override
    public int compare(Log log1, Log log2) {
        if (log1.timestamp < log2.timestamp) return -1;
        else if (log1.timestamp > log2.timestamp) return 1;
        return 0;
    }
}
```

## 1153. 字符串转化

### 1153. 字符串转化

难度 困难 ⚡ 15 ❤️ 🔍 🎁

给出两个长度相同的字符串，分别是 `str1` 和 `str2`。请你帮忙判断字符串 `str1` 能不能在零次或多次转化后变成字符串 `str2`。

每一次转化时，将会一次性将 `str1` 中出现的所有相同字母变成其他任何小写英文字母（见示例）。

只有在字符串 `str1` 能够通过上述方式顺利转化为字符串 `str2` 时才能返回 `True`，否则返回 `False`。

示例 1：

```
输入: str1 = "aabcc", str2 = "ccdee"
输出: true
解释: 将 'c' 变成 'e', 然后把 'b' 变成 'd', 接着再把 'a' 变成 'c'。注意, 转化的顺序也很重要。
```

示例 2：

```
输入: str1 = "leetcode", str2 = "codeleet"
输出: false
解释: 我们没有办法能够把 str1 转化为 str2。
```

```
1 class Solution {
2     public boolean canConvert(String str1, String str2) {
3         int len = str1.length();
4         if (len <= 1 || str1.equals(str2)) {
5             return true;
6         }
7         char[] chars1 = str1.toCharArray();
8         char[] chars2 = str2.toCharArray();
9         char[] lowerReaches = new char['z' + 1];
10        char[] upperReaches = new char['z' + 1];
11        int freeCharNum = 26;
12        for (int idx = 0; idx < len; idx++) {
13            char char1 = chars1[idx];
14            char char2 = chars2[idx];
15            if (lowerReaches[char1] != 0 && lowerReaches[char1] != char2) {
16                return false;
17            } else {
18                char upperReach = upperReaches[char2];
19                if (upperReach == 0) {
20                    if (--freeCharNum == 0) {
21                        return false;
22                    }
23                    upperReaches[char2] = char1;
24                }
25                lowerReaches[char1] = char2;
26            }
27        }
28    }
29    return true;
30 }
```

## 1165. 单行键盘

### 1165. 单行键盘

难度 简单 ⚡ 7 ❤️ 🔍 🎁

我们定制了一款特殊的力扣键盘，所有的键都排列在一行上。

我们可以按从左到右的顺序，用一个长度为 26 的字符串 `keyboard`（索引从 0 开始，到 25 结束）来表示该键盘的键位布局。

现在需要测试这个键盘是否能够有效工作，那么我们就需要个机械手来测试这个键盘。

最初的时候，机械手位于左边起第一个键（也就是索引为 0 的键）的上方。当机械手移动到某一字符所在的键位时，就会在终端上输出该字符。

机械手从索引 `i` 移动到索引 `j` 所需要的时间是  $|i - j|$ 。

当前测试需要你使用机械手输出指定的单词 `word`，请你编写一个函数来计算机械手输出该单词所需的时间。

```
1 class Solution {
2     public int calculateTime(String keyboard, String word) {
3         int sum = 0;
4         int preIdx = 0;
5         int curIdx = 0;
6         int[] hash = new int[26];
7         for (int idx = 0; idx < keyboard.length(); idx++) {
8             hash[keyboard.charAt(idx) - 'a'] = idx;
9         }
10        for (char ch : word.toCharArray()) {
11            curIdx = hash[ch - 'a'];
12            sum += Math.abs(curIdx - preIdx);
13            preIdx = curIdx;
14        }
15    }
16    return sum;
17 }
18 }
```

## 1166. 设计文件系统

### 1166. 设计文件系统

难度 中等 ⚡ 5 ❤️ 🔍 🎁

你需要设计一个能提供下面两个函数的文件系统：

- **create(path, value):** 创建一个新的路径，并尽可能将值 `value` 与路径 `path` 关联，然后返回 `True`。如果路径已经存在或者路径的父路径不存在，则返回 `False`。
- **get(path):** 返回与路径关联的值。如果路径不存在，则返回 `-1`。

“路径”是由一个或多个符合下述格式的字符串连接起来形成的：在 `/` 后跟着一个或多个小写英文字母。

例如 `/leetcode` 和 `/leetcode/problems` 都是有效的路径，但空字符串和 `/` 不是有效的路径。

好了，接下来就请你来实现这两个函数吧！（请参考示例以获得更多信息）

```
1 class FileSystem {
2     private Map<String, Integer> mFileMap;
3     public FileSystem() {
4         mFileMap = new HashMap<>();
5         mFileMap.put("", -1);
6     }
7     public boolean createPath(String path, int value) {
8         if (mFileMap.containsKey(path))
9             return false;
10        int lastIndex = path.lastIndexOf("/");
11        String parentPath = path.substring(0, lastIndex);
12        if (!mFileMap.containsKey(parentPath))
13            return false;
14        mFileMap.put(path, value);
15        return true;
16    }
17    public int get(String path) {
18        return mFileMap.getOrDefault(path, -1);
19    }
20 }
```

## 1167. 连接棒材的最低费用

### 1167. 连接棒材的最低费用

难度 中等 ⚡ 12 ❤️ ⚡ 🌟 ⚡ 🏆 ⚡ 🔒

为了装修新房，你需要加工一些长度为正整数的棒材 sticks。

如果要将长度分别为 x 和 y 的两根棒材连接在一起，你需要支付  $x + y$  的费用。由于施工需要，你必须将所有棒材连接成一根。

返回把你所有棒材 sticks 连成一根所需要的最低费用。注意你可以任意选择棒材连接的顺序。

示例 1：

输入: sticks = [2,4,3]  
输出: 14

```
1 class Solution {
2     public int connectSticks(int[] sticks) {
3         PriorityQueue<Integer> queue = new PriorityQueue<>();
4         for (int num : sticks)
5             queue.offer(num);
6         int ans = 0;
7         while (!queue.isEmpty()) {
8             int num = queue.poll();
9             if (!queue.isEmpty())
10                 int nxt = queue.poll();
11                 queue.offer(num + nxt);
12                 ans += num + nxt;
13             }
14         }
15     return ans;
16 }
17
18
19
```

## 1168. 水资源分配优化

### 1168. 水资源分配优化

难度 困难 ⚡ 17 ❤️ ⚡ 🌟 ⚡ 🏆 ⚡ 🔒

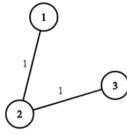
村里面一共有 n 栋房子。我们希望通过建造水井和铺设管道来为所有房子供水。

对于每个房子 i，我们有两种可选的供水方案：

- 一种是直接在房子内建造水井。成本为 wells[i]；
- 另一种是从另一口井铺设管道引水，数组 pipes 给出了在房子间铺设管道的成本，其中每个 pipes[i] = [house1, house2, cost] 代表用管道将 house1 和 house2 连接在一起的成本。当然，连接是双向的。

请你帮忙计算为所有房子都供水的最低总成本。

示例：



```
1 class Solution {
2     public int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {
3         int ans = 0; int[] costs = new int[n + 1], roots = new int[n + 1];
4         for (int idx = 1; idx <= n; ++idx) {
5             roots[idx] = idx; costs[idx] = wells[idx - 1];
6             ans += wells[idx - 1];
7         }
8         Arrays.sort(pipes, (o1, o2) -> o1[2] - o2[2]);
9         for (int[] edge : pipes) {
10            int vex0 = edge[0], vex1 = edge[1], cost = edge[2];
11            int root1 = findRoot(vex0, roots), root2 = findRoot(vex1, roots);
12            if (root1 == root2)
13                continue;
14            if (costs[root1] > costs[root2] && costs[root1] > cost) {
15                roots[root1] = root2; ans += cost - costs[root1];
16            } else if (costs[root2] > cost) {
17                roots[root2] = root1; ans += cost - costs[root2];
18            }
19        }
20     return ans;
21 }
22 private int findRoot(int idx, int[] roots) {
23     while (roots[idx] != idx) {
24         roots[idx] = roots[roots[idx]];
25         idx = roots[idx];
26     }
27     return idx;
28 }
29 }
```

## 1176. 健身计划评估

### 1176. 健身计划评估

难度 简单 ⚡ 11 ❤️ ⚡ 🌟 ⚡ 🏆 ⚡ 🔒

你的好友是一位健身爱好者。前段日子，他给自己制定了一份健身计划。现在想请你帮他评估一下这份计划是否合理。

他会有一份计划消耗的卡路里表，其中 calories[i] 给出了你的这位好友第 i 天需要消耗的卡路里总量。

为了更好地评估这份计划，对于卡路里表中的每一天，你都需要计算他「这一天以及之后的连续几天」（共 k 天）内消耗的总卡路里 T：

- 如果  $T < lower$ ，那么这份计划相对糟糕，并失去 1 分；
- 如果  $T > upper$ ，那么这份计划相对优秀，并获得 1 分；
- 否则，这份计划普普通通，分值不做变动。

请返回统计完所有 calories.length 天后得到的总分作为评估结果。

注意：总分可能是负数。

```
1 class Solution {
2     public int dietPlanPerformance(int[] calories, int k, int lower, int upper) {
3         int ans = 0, cal = 0, len = calories.length;
4         for (int idx = 0; idx < k; idx++) { // 计算第一个k天段的消耗值
5             cal += calories[idx];
6         }
7         if (cal < lower) {
8             ans--;
9         } else if (cal > upper) {
10            ans++;
11        }
12        for (int idx = 0; idx < len - k; idx++) { // 滑动窗口，每次右滑一位
13            cal = cal - calories[idx] + calories[idx + k];
14            if (cal < lower) {
15                ans--;
16            } else if (cal > upper) {
17                ans++;
18            }
19        }
20    return ans;
21 }
22 }
```

## 1180. 统计只含单一字母的子串

### 1180. 统计只含单一字母的子串

难度 简单 ⚡ 10 ❤️ ⚡ 🌟 ⚡ 🏆 ⚡ 🔒

给你一个字符串 s，返回只含 单一字母 的子串个数。

示例 1：

输入: "aaaba"  
输出: 8  
解释:  
只含单一字母的子串分别是 "aaa", "aa", "a", "b"。  
"aaa" 出现 1 次。  
"aa" 出现 2 次。

```
1 class Solution {
2     public int countLetters(String S) {
3         char[] chs = S.toCharArray();
4         int ans = 0, idx = 0, len = chs.length;
5         while (idx < len) {
6             int cnt = 1;
7             while (idx + 1 < len && chs[idx] == chs[idx + 1]) {
8                 ++cnt;
9                 ++idx;
10            }
11            ans += (cnt * (cnt + 1) / 2);
12            ++idx;
13        }
14    return ans;
15 }
16 }
```

## 1181. 前后拼接

### 1181. 前后拼接

难度 中等 ⚡ 7 ❤️ ⚡ 🌟 ⚡ 🌟

给你一个「短语」列表 `phrases`，请你帮忙按规则生成拼接后的「新短语」列表。

「短语」`(phrase)`是仅由小写英文字母和空格组成的字符串。「短语」的开头和结尾都不会出现空格，「短语」中的空格不会连续出现。

「前后拼接」`(Before and After puzzles)`是合并两个「短语」形成「新短语」的方法。我们规定拼接时，第一个短语的最后一个单词 和 第二个短语的第一个单词 必须相同。

返回每两个「短语」`phrases[i]` 和 `phrases[j]` ( $i \neq j$ ) 进行「前后拼接」得到的「新短语」。

注意，两个「短语」拼接时的顺序也很重要，我们需要同时考虑这两个「短语」。另外，同一个「短语」可以多次参与拼接，但「新短语」不能再参与拼接。

请你按字典序排列并返回「新短语」列表，列表中的字符串应该是 不重复 的。

示例 1：

输入: `phrases = ["writing code", "code rocks"]`  
输出: `["writing code rocks"]`

示例 2：

输入: `phrases = ["ab", "cd", "ef", "gh"]`  
输出: `["ab", "cd", "ef", "gh"]`

```
1 v class Solution {
2 v     public List<String> beforeAndAfterPuzzles(String[] phrases) {
3 v         Set<String> set = new HashSet<>();
4 v         List<String> list = new ArrayList<>();
5 v         Map<String, List<Integer>> map = new HashMap<>();
6 v         for (int idx = 0; idx < phrases.length; idx++) {
7 v             int sIdx = phrases[idx].indexOf(" ");
8 v             String str = sIdx == -1 ? phrases[idx] : phrases[idx].substring(0, sIdx);
9 v             if (!map.containsKey(str)) {
10 v                 map.put(str, new ArrayList<>());
11 v             }
12 v             map.get(str).add(idx);
13 v         }
14 v         for (int idx = 0; idx < phrases.length; idx++) {
15 v             int sIdx = phrases[idx].lastIndexOf(" ");
16 v             String str = sIdx == -1 ? phrases[idx] : phrases[idx].substring(sIdx + 1);
17 v             if (!map.containsKey(str)) {
18 v                 List<Integer> tmpList = map.get(str);
19 v                 for (int i = 0; i < tmpList.size(); i++) {
20 v                     if (tmpList.get(i) == idx) {
21 v                         continue;
22 v                     }
23 v                     String tmp = phrases[idx].substring(0, phrases[idx].length() - str.length());
24 v                     tmp += phrases[tmpList.get(i)];
25 v                     if (!set.contains(tmp)) {
26 v                         list.add(tmp);
27 v                         set.add(tmp);
28 v                     }
29 v                 }
30 v             }
31 v             Collections.sort(list, (a, b) -> a.compareTo(b));
32 v         }
33 v         return list;
34 v     }
35 }
```

## 1182. 与目标颜色间的最短距离

### 1182. 与目标颜色间的最短距离

难度 中等 ⚡ 7 ❤️ ⚡ 🌟 ⚡ 🌟

给你一个数组 `colors`，里面有 1、2、3 三种颜色。

我们需要在 `colors` 上进行一些查询操作 `queries`，其中每个待查项都由两个整数 `i` 和 `c` 组成。

现在请你帮忙设计一个算法，查找从索引 `i` 到具有目标颜色 `c` 的元素之间的最短距离。

如果不存在解决方案，请返回 `-1`。

示例 1：

输入: `colors = [1,1,2,1,3,2,2,3,3]`, `queries = [[1,3], [2,2], [6,1]]`  
输出: `[3,0,3]`  
解释:  
距离索引 1 最近的颜色 3 位于索引 4 (距离为 3)。  
距离索引 2 最近的颜色 2 就是它自己 (距离为 0)。  
距离索引 6 最近的颜色 1 位于索引 3 (距离为 3)。

示例 2：

输入: `colors = [1,2]`, `queries = [[0,3]]`  
输出: `[-1]`  
解释: colors 中没有颜色 3。

提示:

- $1 \leqslant \text{colors.length} \leqslant 5 \times 10^4$
- $1 \leqslant \text{colors}[i] \leqslant 3$
- $1 \leqslant \text{queries.length} \leqslant 5 \times 10^4$
- $\text{queries}[i].length == 2$
- $0 \leqslant \text{queries}[i][0] < \text{colors.length}$
- $1 \leqslant \text{queries}[i][1] \leqslant 3$

```
1 v class Solution { // 二分搜索
2 v     public List<Integer> shortestDistanceColor(int[] colors, int[][] queries) {
3 v         List<Integer> list1 = new ArrayList<>();
4 v         List<Integer> list2 = new ArrayList<>();
5 v         List<Integer> list3 = new ArrayList<>();
6 v         for (int idx = 0; idx < colors.length; idx++) {
7 v             if (colors[idx] == 1) {
8 v                 list1.add(idx);
9 v             } else if (colors[idx] == 2) {
10 v                 list2.add(idx);
11 v             } else {
12 v                 list3.add(idx);
13 v             }
14 v         }
15 v         List<Integer> ans = new ArrayList<>();
16 v         for (int[] query : queries) {
17 v             List<Integer> tmp;
18 v             if (query[1] == 1) {
19 v                 tmp = list1;
20 v             } else if (query[1] == 2) {
21 v                 tmp = list2;
22 v             } else {
23 v                 tmp = list3;
24 v             }
25 v             if (tmp.size() == 0) {
26 v                 ans.add(-1);
27 v                 continue;
28 v             }
29 v             int idx = Collections.binarySearch(tmp, query[0]);
30 v             if (idx >= 0) {
31 v                 ans.add(0);
32 v                 continue;
33 v             }
34 v             idx = -(idx + 1);
35 v             int min = Integer.MAX_VALUE;
36 v             if (idx - 1 >= 0) {
37 v                 min = query[0] - tmp.get(idx - 1);
38 v             }
39 v             if (idx < tmp.size()) {
40 v                 min = Math.min(min, tmp.get(idx) - query[0]);
41 v             }
42 v             ans.add(min);
43 v         }
44 v         return ans;
45 v     }
46 }
```

## 1183. 矩阵中 1 的最大数量 (公式法)

### 1183. 矩阵中 1 的最大数量

难度 困难 ⚡ 14 ❤️ ⚡ 🌟 ⚡ 🌟

现在有一个尺寸为 `width * height` 的矩阵 `M`，矩阵中的每个单元格的值不是 0 就是 1。

而且矩阵 `M` 中每个大小为 `sideLength * sideLength` 的正方形子阵中，1 的数量不得超过 `maxOnes`。

请你设计一个算法，计算矩阵中最多可以有多少个 1。

```
1 v class Solution { // 公式法
2 v     public int maximumNumberOfOnes(int width, int height, int sideLength, int maxOnes) {
3 v         int nw = width / sideLength, nh = height / sideLength;
4 v         int rw = width % sideLength, rh = height % sideLength;
5 v         int min1 = Math.min(rw * rh, maxOnes); int ans = nw * nh * maxOnes + min1;
6 v         int min2 = Math.min(nw, nh); int area = ((rw + rh) * sideLength - rw * rh);
7 v         ans += min2 * (Math.min(maxOnes, area) + min1);
8 v         if (rw > nh) {
9 v             ans += (nw - nh) * Math.min(maxOnes, rh * sideLength);
10 v        } else if (rw < nh) {
11 v            ans += (nh - nw) * Math.min(maxOnes, rw * sideLength);
12 v        }
13 v    }
14 v    return ans;
15 }
```

### 1183. 矩阵中 1 的最大数量 (贪心算法)

#### 1183. 矩阵中 1 的最大数量

难度 困难 ⚡ 14 ❤️ 🔍 🌐 💬

现在有一个尺寸为  $width * height$  的矩阵  $M$ , 矩阵中的每个单元格的值不是 0 就是 1。

而且矩阵  $M$  中每个大小为  $sideLength * sideLength$  的正方形子阵中, 1 的数量不得超过  $maxOnes$ 。

请你设计一个算法, 计算矩阵中最多可以有多少个 1。

```
1 class Solution { // 贪心算法
2     public int maximumNumberOfOnes(int width, int height,
3                                     int sideLength, int maxOnes) {
4         int ans = 0;
5         PriorityQueue<Integer> queue = new PriorityQueue<>();
6         for (int i = 0; i < sideLength; ++i)
7             for (int x = 0; x < sideLength; x++) {
8                 queue.add((width - 1 - i) / sideLength + 1)
9                         * ((height - 1 - x) / sideLength + 1);
10                if (queue.size() > maxOnes) queue.poll();
11            }
12        while (!queue.isEmpty()) ans += queue.poll();
13    }
14 }
15 }
```

### 1188. 设计有限阻塞队列

#### 1188. 设计有限阻塞队列

难度 中等 ⚡ 8 ❤️ 🔍 🌐 💬

实现一个拥有如下方法的线程安全有限阻塞队列:

- `BoundedBlockingQueue(int capacity)` 构造方法初始化队列, 其中 `capacity` 代表队列长度上限。
- `void enqueue(int element)` 在队首增加一个 `element`. 如果队列满, 调用线程被阻塞直到队列非满。
- `int dequeue()` 返回队尾元素并从队列中将其删除. 如果队列为空, 调用线程被阻塞直到队列非空。
- `int size()` 返回当前队列元素个数。

你的实现将会被多线程同时访问进行测试。每一个线程要么是一个只调用 `enqueue` 方法的生产者线程, 要么是一个只调用 `dequeue` 方法的消费者线程。`size` 方法将会在每一个测试用例之后进行调用。

请不要使用内置的有限阻塞队列实现, 否则面试将不会通过。

#### 示例 1:

```
输入:
1
1
["BoundedBlockingQueue","enqueue","dequeue","dequeue"]
[[2],[1],[],[],[0],[2],[3],[4],[]]

输出:
[1,0,2,2]

解释:
生产者线程数目 = 1
消费者线程数目 = 1

BoundedBlockingQueue queue = new
BoundedBlockingQueue(2); // 使用capacity = 2初始化
队列。

queue.enqueue(1); // 生产者线程将1插入队列。
queue.dequeue(); // 消费者线程调用dequeue并返回1。
```

```
1 class BoundedBlockingQueue {
2     private int head, tail;
3     private int[] queue;
4     private ReentrantLock lock = new ReentrantLock();
5     private Condition full = lock.newCondition();
6     private Condition empty = lock.newCondition();
7     public BoundedBlockingQueue(int capacity) {
8         this.queue = new int[capacity + 1];
9         this.tail = this.head = 0;
10    }
11    public void enqueue(int element) throws InterruptedException {
12        lock.lock();
13        try {
14            while ((tail + 1) % queue.length == head) full.await();
15            tail = (tail + 1) % queue.length;
16            queue[tail] = element;
17            empty.signalAll();
18        } finally {
19            lock.unlock();
20        }
21    }
22    public int dequeue() throws InterruptedException {
23        lock.lock();
24        try {
25            while (head == tail) empty.await();
26            head = (head + 1) % queue.length;
27            full.signalAll();
28            return queue[head];
29        } finally {
30            lock.unlock();
31        }
32    }
33    public int size() {
34        lock.lock();
35        try {
36            if (tail == head) {
37                return 0;
38            } else if (tail > head) {
39                return tail - head;
40            } else {
41                return tail - 0 + (queue.length - head);
42            }
43        } finally {
44            lock.unlock();
45        }
46    }
47 }
```

### 1196. 最多可以买到的苹果数量

#### 1196. 最多可以买到的苹果数量

难度 简单 ⚡ 3 ❤️ 🔍 🌐 💬

楼下水果店正在促销, 你打算买些苹果, `arr[i]` 表示第 `i` 个苹果的单位重量。

你有一个购物袋, 最多可以装 5000 单位重量的东西, 算一算, 最多可以往购物袋里装入多少苹果。

```
1 class Solution { // 贪心算法
2     public int maxNumberOfApples(int[] arr) {
3         Arrays.sort(arr);
4         int cnt = 0, sum = 0;
5         for (int i = 0; i < arr.length; i++) {
6             sum += arr[i];
7             if (sum <= 5000) {
8                 cnt++;
9             } else {
10                 break;
11             }
12         }
13         return cnt;
14     }
15 }
```

## 1197. 进击的骑士

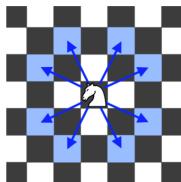
## 1197. 进击的骑士

难度 中等

一个坐标可以从 `-infinity` 延伸到 `+infinity` 的无限大的棋盘上，你的骑士驻扎在坐标为 `[0, 0]` 的方格里。

骑士的走法和中国象棋中的马相似，走“日”字：即先向左（或右）走1格，再向上（或下）走2格；或先向左（或右）走2格，再向上（或下）走1格。

每次移动，他都可以按图示八个方向之一前进。



现在，骑士需要前去征服坐标为  $[x, y]$  的部落，请你为他规划路线。

最后返回所需的最小移动次数即可。本题确保答案是一定存在的。

### 示例 1：

输出: 1

10

輸入:  $x = 5, y = 5$   
輸出: 4  
解釋:  $[0, 0] \rightarrow [2, 1] \rightarrow [4, 2] \rightarrow [3, 4] \rightarrow [5, 5]$

```

class Solution {
    public int minKnightMoves(int x, int y) {
        Queue<int[]> pQueue = new LinkedList<int[]>;
        Queue<Integer> lQueue = new LinkedList<Integer>;
        pQueue.add(new int[]{0, 0});
        lQueue.add(0);
        while (pQueue.size() > 0) {
            int[] currPos = pQueue.poll();
            int currLevel = lQueue.poll();
            int[] dis = {x - currPos[0], y - currPos[1]};
            int xx, yy;
            if (Math.abs(dis[0]) > Math.abs(dis[1]) && (dis[0] >= 5 || dis[0] <= -5)) {
                xx = currPos[0] + 2 * dis[0] / Math.abs(dis[0]);
                if (dis[1] == 0)
                    yy = currPos[1] + 1;
                else
                    yy = currPos[1] + dis[1] / Math.abs(dis[1]);
                pQueue.add(new int[]{xx, yy});
                lQueue.add(currLevel + 1);
                continue;
            }
            if (dis[1] >= 5 || dis[1] <= -5) {
                yy = currPos[1] + 2 * dis[1] / Math.abs(dis[1]);
                if (dis[0] == 0) {
                    xx = currPos[0] + 1;
                } else {
                    xx = currPos[0] + dis[0] / Math.abs(dis[0]);
                }
                pQueue.add(new int[]{xx, yy});
                lQueue.add(currLevel + 1);
                continue;
            }
            final int[][] dirs = {{1, 2}, {2, 1}, {1, -2}, {2, -1}, {-1, -2}, {-2, -1}, {-1, 2}, {-2, 1}};
            for (int[] i : dirs) {
                if (i[0] * dis[0] + i[1] * dis[1] <= 0)
                    continue;
                else {
                    xx = currPos[0] + i[0];
                    yy = currPos[1] + i[1];
                    if (xx == x && yy == y)
                        return currLevel + 1;
                    pQueue.add(new int[]{xx, yy});
                    lQueue.add(currLevel + 1);
                }
            }
        }
        return 0;
    }
}

```

### 1198. 找出所有行中最小公共元素

### 1198. 找出所有行中最小公共元素

難度由繁到簡 由易到難

给你一个矩阵 `mat`，其中每一行的元素都已经按 递增 顺序排好了。请你帮忙找出在所有这些行中 最小的公共元素。

如果矩阵中没有这样的公共元素，就请返回 -1。

示例：

```
输入: mat = [[1,2,3,4,5],[2,4,5,8,10],[3,5,7,9,11],  
[1,3,5,7,9]]  
输出: 5
```

```
1 class Solution {
2     public int smallestCommonElement(int[][] mat) {
3         int[] times = new int[10001];
4         int rows = mat.length;
5         int cols = mat[0].length;
6         for (int rid = 0; rid < rows; rid++) {
7             for (int cid = 0; cid < cols; cid++) {
8                 ++times[mat[rid][cid]];
9             }
10        }
11        for (int idx = 1; idx <= 10000; idx++) {
12            if (times[idx] == rows) {
13                return idx;
14            }
15        }
16        return -1;
17    }
18 }
```

### 1199. 建造街区的最短时间

### 1199. 建造街区的最短时间

难度 困难 ✓ 11 ❤ 10 章 1

你是个城市规划工作者，手里负责管辖一系列的街区。在这个街区列表中 `blocks[i] = t` 意味着第 `i` 个街区需要 `t` 个单位的时间来建造。

由于一个街区只能由一个工人来完成建造。

所以，一个工人要么需要再召唤一个工人（工人数增加 1）；要么建造完一个街区后回家。这两个决定都需要花费一定的时间。

一个工人再召唤一个工人所花费的时间由整数 `split` 给出

注意：如果两个工人同时召唤别的工人，那么他们的行为是并行的，所以时间花费仍然是  $O(n^2)$ 。

最开始的时候只有一个工人，请你最后输出建造完所有街区所需要的最少时间。

```
class Solution {
    public int minBuildTime(int[] blocks, int split) {
        PriorityQueue<Integer> queue = new PriorityQueue<>();
        for (int block : blocks) {
            queue.add(block);
        }
        while (queue.size() > 1) {
            queue.poll();
            int block = queue.poll();
            queue.add(split + block);
        }
        return queue.peek();
    }
}
```

### 1213. 三个有序数组的交集

#### 1213. 三个有序数组的交集

难度 简单 ⚡ 10 ❤ 1 文章 ⌂ 回

给出三个均为 **严格递增排列** 的整数数组 `arr1`, `arr2` 和 `arr3`。

返回一个由 **仅** 在这三个数组中 **同时出现** 的整数所构成的有序数组。

示例:

```
输入: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]
输出: [1,5]
```

### 1214. 查找两棵二叉搜索树之和

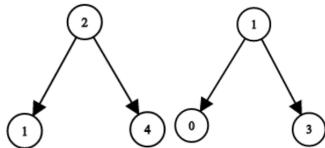
#### 1214. 查找两棵二叉搜索树之和

难度 中等 ⚡ 8 ❤ 1 文章 ⌂ 回

给出两棵二叉搜索树，请你从两棵树中各找出一个节点，使得这两个节点的值之和等于目标值 `Target`。

如果可以找到返回 `True`，否则返回 `False`。

示例 1:



### 1215. 步进数

#### 1215. 步进数

难度 中等 ⚡ 7 ❤ 1 文章 ⌂ 回

如果一个整数上的每一位数字与其相邻位上的数字的绝对差都是 `1`，那么这个数就是一个「步进数」。

例如，`321` 是一个步进数，而 `421` 不是。

给你两个整数，`low` 和 `high`，请你找出在 `[low, high]` 范围内的所有步进数，并返回 **排序后** 的结果。

示例:

```
输入: low = 0, high = 21
输出: [0,1,2,3,4,5,6,7,8,9,10,12,21]
```

### 1216. 验证回文字符串 III

#### 1216. 验证回文字符串 III

难度 困难 ⚡ 14 ❤ 1 文章 ⌂ 回

给出一个字符串 `s` 和一个整数 `k`，请你帮忙判断这个字符串是不是一个「K回文」。

所谓「K回文」：如果可以通过从字符串中删去最多 `k` 个字符将其转换为回文，那么这个字符串就是一个「K回文」。

示例:

```
输入: s = "abcdeca", k = 2
输出: true
解释: 删除字符 "b" 和 "e"。
```

```
1 class Solution {
2     public List<Integer> arraysIntersection(
3         int[] arr1, int[] arr2, int[] arr3) {
4         int[] map = new int[2001];
5         for (int n : arr1) map[n]++;
6         for (int n : arr2) map[n]++;
7         for (int n : arr3) map[n]++;
8         List<Integer> list = new ArrayList<>();
9         for (int i = 0; i < map.length; i++) {
10            if (map[i] == 3) list.add(i);
11        }
12    }
13 }
14
15 }
```

```
1 class Solution {
2     public boolean twoSumBSTs(TreeNode root1, TreeNode root2, int target) {
3         if (root1 == null)
4             return false;
5         return isExist(root2, target - root1.val) ||
6             twoSumBSTs(root1.left, root2, target) ||
7             twoSumBSTs(root1.right, root2, target);
8     }
9
10 private boolean isExist(TreeNode node, int val) {
11     if (node == null)
12         return false;
13     if (node.val > val) {
14         return isExist(node.left, val);
15     } else if (node.val < val) {
16         return isExist(node.right, val);
17     } else {
18         return true;
19     }
20 }
```

```
1 class Solution { // BFS
2     public List<Integer> countSteppingNumbers(int low, int high) {
3         List<Integer> ans = new ArrayList<>();
4         Queue<Integer> queue = new LinkedList<>();
5         if (low == 0) ans.add(0);
6         for (int i = 1; i <= 9; i++) queue.offer(i);
7         while (!queue.isEmpty()) {
8             int cur = queue.poll();
9             if (cur >= low && cur <= high) ans.add(cur);
10            if (cur > high / 10) continue;
11            int rem = cur % 10;
12            if (rem != 9 && cur * 10 + rem + 1 <= high)
13                queue.offer(cur * 10 + rem + 1);
14            if (rem != 0 && cur * 10 + rem - 1 <= high)
15                queue.offer(cur * 10 + rem - 1);
16        }
17        Collections.sort(ans);
18    }
19 }
```

```
1 class Solution {
2     public boolean isValidPalindrome(String s, int k) {
3         return s.length() - longestPalindromeSubseq(s) <= k;
4     }
5
6     public int longestPalindromeSubseq(String str) {
7         char[] ss = str.toCharArray();
8         int n = str.length();
9         if (n == 0) return 0;
10        int[][] dp = new int[n][n];
11        for (int i = n - 1; i >= 0; i--) {
12            dp[i][i] = 1;
13            for (int j = i + 1; j < n; j++) {
14                if (ss[i] == ss[j]) {
15                    dp[i][j] = dp[i + 1][j - 1] + 2;
16                } else {
17                    dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);
18                }
19            }
20        }
21        return dp[0][n - 1];
22    }
}
```

## 1228. 等差数列中缺失的数字

### 1228. 等差数列中缺失的数字

难度 简单 击 4 喜欢 10 收藏 1

有一个数组，其中的值符合等差数列的数值规律，也就是说：

- 在  $0 \leq i < arr.length - 1$  的前提下， $arr[i+1] - arr[i]$  的值都相等。

我们会从该数组中删除一个既不是第一个也不是最后一个的值，得到一个新的数组  $arr$ 。

给你这个缺值的数组  $arr$ ，请你帮忙找出被删除的那个数。

```
1 class Solution {
2     public int missingNumber(int[] arr) {
3         int diff1 = arr[1] - arr[0];
4         int diff2 = arr[2] - arr[1];
5         int diff = diff1;
6         if (Math.abs(diff1) > Math.abs(diff2))
7             diff = diff2;
8         int pre = arr[0];
9         for (int i = 1; i < arr.length; ++i)
10            if (arr[i] - arr[i - 1] != diff)
11                pre = arr[i - 1];
12     }
13 }
```

## 1229. 安排会议日程

### 1229. 安排会议日程

难度 中等 击 6 喜欢 10 收藏 1

你是一名行政助理，手里有两位客户的空闲时间表： $slots1$  和  $slots2$ ，以及会议的预计持续时间  $duration$ ，请你为他们安排合适的会议时间。

「会议时间」是两位客户都有空参加，并且持续时间能够满足预计时间  $duration$  的最早的时间间隔。

如果没有满足要求的会议时间，就请返回一个空数组。

「空闲时间」的格式是  $[start, end]$ ，由开始时间  $start$  和结束时间  $end$  组成，表示从  $start$  开始，到  $end$  结束。

题目保证数据有效：同一个人的空闲时间不会出现交叠的情况，也就是说，对于同一个人的两个空闲时间  $[start1, end1]$  和  $[start2, end2]$ ，要么  $start1 > end2$ ，要么  $start2 > end1$ 。

```
1 class Solution {
2     public List<Integer> minAvailableDuration(int[][] slots1, int[][] slots2, int duration) {
3         List<Integer> ans = new ArrayList<>();
4         Arrays.sort(slots1, (o1, o2) -> (o1[0] == o2[0] ? o1[1] - o2[0] : o1[0] - o2[0]));
5         Arrays.sort(slots2, (o1, o2) -> (o1[0] == o2[0] ? o1[1] - o2[0] : o1[0] - o2[0]));
6         for (int i1 = 0, i2 = 0; i1 < slots1.length && i2 < slots2.length; ) {
7             while (slots1[i1][0] > slots2[i2][1]) i2++;
8             while (slots1[i1][1] < slots2[i2][0]) i1++;
9             int right = Math.min(slots1[i1][1], slots2[i2][1]);
10            int left = Math.max(slots1[i1][0], slots2[i2][0]);
11            int len = right - left;
12            if (len >= duration) {
13                ans.add(left);
14                ans.add(left + duration);
15                return ans;
16            } else {
17                if (slots2[i2][1] > slots1[i1][1]) i1++;
18                else if (slots2[i2][1] < slots1[i1][1]) i2++;
19            }
20        }
21        return ans;
22    }
23 }
```

## 1230. 抛掷硬币

### 1230. 抛掷硬币

难度 中等 击 15 喜欢 10 收藏 1

有一些不规则的硬币。在这些硬币中， $prob[i]$  表示第  $i$  枚硬币正面朝上的概率。

请对每一枚硬币抛掷一次，然后返回正面朝上的硬币数等于  $target$  的概率。

```
1 class Solution { // 动态规划
2     public double probabilityOfHeads(double[] prob, int target) {
3         double[] dp = new double[target + 1];
4         dp[0] = 1;
5         for (int i = 0; i < prob.length; i++) {
6             for (int x = Math.min(i + 1, target); x > 0; x--)
7                 dp[x] = dp[x - 1] * prob[i] + dp[x] * (1 - prob[i]);
8             dp[0] = dp[0] * (1 - prob[i]);
9         }
10        return dp[target];
11    }
12 }
```

## 1231. 分享巧克力

### 1231. 分享巧克力

难度 困难 击 20 喜欢 10 收藏 1

你有一大块巧克力，它由一些甜度不完全相同的小块组成。我们用数组  $sweetness$  来表示每一小块的甜度。

你打算和  $K$  名朋友一起分享这块巧克力，所以你需要将切割  $K$  次才能得到  $K+1$  块，每一块都由一些连续的小块组成。

为了表现出你的慷慨，你将会吃掉总甜度最小的一块，并将其余几块分给你的朋友们。

请找出一个最佳的切割策略，使得你所分得的巧克力总甜度最大，并返回这个最大总甜度。

示例 1：

```
输入: sweetness = [1,2,3,4,5,6,7,8,9], K = 5
输出: 6
解释: 你可以把巧克力分成 [1,2,3], [4,5], [6], [7], [8], [9]。
```

```
1 class Solution {
2     private boolean check(int[] nums, int k, int mid) {
3         int sum = 0;
4         int cnt = 0;
5         for (int i = 0; i < nums.length; i++) {
6             sum += nums[i];
7             if (sum >= mid) {
8                 cnt++;
9                 sum = 0;
10            }
11        }
12        return cnt > k;
13    }
14    public int maximizeSweetness(int[] sweetness, int K) {
15        int lid = 1, rid = Integer.MAX_VALUE;
16        while (lid < rid) {
17            int mid = (rid - lid) / 2 + lid;
18            if (check(sweetness, K, mid)) {
19                lid = mid + 1;
20            } else {
21                rid = mid;
22            }
23        }
24        return lid - 1;
25    }
26 }
```

## 1236. 网络爬虫

1236. 网络爬虫

难度 中等 通过数 10 收藏 分享

给定一个链接 `startUrl` 和一个接口 `UrlParser`，请你实现一个网络爬虫，以实现爬取同 `startUrl` 拥有相同 域名标签 的全部链接。该爬虫得到的全部链接可以 任何顺序 返回结果。

你的网络爬虫应当按照如下模式工作：

- 自链接 `startUrl` 开始爬取
- 调用 `UrlParser.getUrls(url)` 来获得链接 `url` 页面中的全部链接
- 同一个链接最多只爬取一次
- 只输出 域名 与 `startUrl` 相同 的链接集合

<http://example.org:8888/foo/bar#bang>

hostname

如上所示的一个链接，其域名为 `example.org`。简单起见，你可以假设所有的链接都采用 `http` 协议 并没有指定 端口号。例如，链接 `http://leetcode.com/problems` 和 `http://leetcode.com/contests` 属于同一个 域名 下的，而链接

```
1 v class Solution {
2 v     public List<String> crawl(String startUrl, HtmlParser htmlParser) {
3 v         Set<String> set = new HashSet<>();
4 v         set.add(startUrl);
5 v         String startDomain = getDomain(startUrl);
6 v         dfs(set, startUrl, htmlParser, startDomain);
7 v         return new ArrayList<>(set);
8 v     }
9 v
10 v    void dfs(Set<String> set, String startUrl, HtmlParser htmlParser, String startDomain) {
11 v        List<String> urls = htmlParser.getUrls(startUrl);
12 v        for (String url : urls) {
13 v            if (!set.contains(url) && startDomain.equals(getDomain(url))) {
14 v                set.add(url);
15 v                dfs(set, url, htmlParser, startDomain);
16 v            }
17 v        }
18 v    }
19 v
20 v    String getDomain(String url) {
21 v        int startIndex = url.indexOf(".");
22 v        if (startIndex == -1) startIndex = "http://".length();
23 v        int endIndex = url.indexOf("/", startIndex);
24 v        if (endIndex == -1) endIndex = url.length();
25 v        String domain = url.substring(startIndex, endIndex);
26 v        return domain;
27 v    }
28 }
```

## 1242. 多线程网页爬虫

1242. 多线程网页爬虫

难度 中等 通过数 11 收藏 分享

给你一个初始地址 `startUrl` 和一个 HTML 解析器接 口 `UrlParser`，请你实现一个 多线程 的网页爬虫，用于获取与 `startUrl` 有 相同主机名 的所有链接。

以 任意 顺序返回爬虫获取的路径。

爬虫应该遵循：

- 从 `startUrl` 开始
- 调用 `UrlParser.getUrls(url)` 从指定网页路径获得的所有 路径。
- 不要抓取相同的链接两次。
- 仅浏览与 `startUrl` 相同 主机名 的链接。

<http://example.org:8888/foo/bar#bang>

hostname

如上图所示，主机名是 `example.org`。简单起见，你可以假设所有链接都采用 `http` 协议，并且没有指定 端口号。举个例子，链接 `http://leetcode.com/problems` 和链 接 `http://leetcode.com/contest` 属于同一个 主机名，而 `http://example.org/test` 与 `http://example.com/abc` 并不属于同一个 主机名。

`UrlParser` 的接口定义如下：

```
interface HtmlParser {
    // Return a list of all urls from a webpage of
    // given url.
    // This is a blocking call. that means it will
```

```
1 v class Solution {
2 v     public List<String> crawl(String startUrl, HtmlParser htmlParser) {
3 v         Set<String> validUrls = new ConcurrentHashMap<>().newKeySet();
4 v         validUrls.add(startUrl);
5 v         String hostname;
6 v         int slashAfterHostNameIndex = startUrl.indexOf('/', 7);
7 v         if (slashAfterHostNameIndex == -1) {
8 v             hostname = startUrl;
9 v         } else {
10 v             hostname = startUrl.substring(0, slashAfterHostNameIndex);
11 v         }
12 v         crawlHelper(startUrl, htmlParser, hostname, validUrls);
13 v         return new ArrayList<>(validUrls);
14 v     }
15 v
16 v     public void crawlHelper(String url, HtmlParser htmlParser,
17 v                             String hostname, Set<String> validUrls) {
18 v         List<String> adjacentUrls = htmlParser.getUrls(url);
19 v         List<Thread> threads = new ArrayList<>();
20 v         for (String adjacentUrl : adjacentUrls) {
21 v             if (!adjacentUrl.startsWith(hostname) || validUrls.contains(adjacentUrl)) {
22 v                 continue;
23 v             }
24 v             validUrls.add(adjacentUrl);
25 v             threads.add(new Thread(() -> {
26 v                 crawlHelper(adjacentUrl, htmlParser, hostname, validUrls);
27 v             }));
28 v         }
29 v         for (Thread thread : threads) {
30 v             thread.start();
31 v         }
32 v         for (Thread thread : threads) {
33 v             try {
34 v                 thread.join();
35 v             } catch (InterruptedException e) {
36 v                 e.printStackTrace();
37 v             }
38 v         }
39 v     }
40 }
```

## 1243. 数组变换

1243. 数组变换

难度 简单 通过数 6 收藏 分享

首先，给你一个初始数组 `arr`。然后，每天你都要根据前一天的数组生成一个新的数组。

第 `i` 天所生成的数组，是由你对第 `i-1` 天的数组进行如下操作所得的：

1. 假如一个元素小于它的左右邻居，那么该元素自增 1。
2. 假如一个元素大于它的左右邻居，那么该元素自减 1。
3. 首、尾元素 永不 改变。

过些时日，你会发现数组将会不再发生变化，请返回最终所得到的数组。

示例 1：

```
输入: [6,2,3,4]
输出: [6,3,3,4]
解释:
第一天, 数组从 [6,2,3,4] 变为 [6,3,3,4]。
无法再对该数组进行更多操作。
```

```
1 v class Solution {
2 v     public List<Integer> transformArray(int[] arr) {
3 v         int[] narr = new int[arr.length];
4 v         narr[0] = arr[0];
5 v         narr[arr.length - 1] = arr[arr.length - 1]; //首尾
6 v         for (int i = 1; i < arr.length - 1; i++) {
7 v             if ((arr[i] < arr[i + 1]) && (arr[i] < arr[i - 1])) {
8 v                 narr[i] = arr[i] + 1;
9 v             } else if ((arr[i] > arr[i + 1]) && (arr[i] > arr[i - 1])) {
10 v                narr[i] = arr[i] - 1;
11 v            } else {
12 v                narr[i] = arr[i];
13 v            }
14 v        }
15 v        if (Arrays.equals(narr, arr)) {
16 v            List<Integer> ans = new ArrayList<>();
17 v            for (int i = 0; i < narr.length; i++) {
18 v                ans.add(narr[i]);
19 v            }
20 v            return ans; //新旧相等返回
21 v        } else {
22 v            return transformArray(narr); //不相等进行下一天
23 v        }
24 v    }
25 }
```

## 1244. 力扣排行榜

### 1244. 力扣排行榜

难度 中等 通过 8 提交 38 通过 4 失败 4

新一轮的「力扣杯」编程大赛即将启动，为了动态显示参赛者的得分数据，需要设计一个排行榜 Leaderboard。

请你帮忙来设计这个 Leaderboard 类，使得它有如下 3 个函数：

1. addScore(playerId, score)：  
◦ 假如参赛者已经在排行榜上，就给他的当前得分增加 score 点分值并更新排行。  
◦ 假如该参赛者不在排行榜上，就把他添加到榜单上，并且将分数设置为 score。
2. top(K)：返回前 K 名参赛者的得分总和。
3. reset(playerId)：将指定参赛者的成绩清零。题目保证在调用此函数前，该参赛者已有成绩，并且在榜单上。

请注意，在初始状态下，排行榜是空的。

#### 示例 1：

输入：  
["Leaderboard", "addScore", "addScore", "addScore", "addScore", "[[], [1, 73], [2, 56], [3, 39], [4, 51], [5, 4], [1], [1], [2], [2, 51], [3]]]  
输出：  
[null, null, null, null, null, null, 73, null, null, null, 141]

```

1 v class Leaderboard {
2   Map<Integer, Integer> freq; // how many players in a score
3   Map<Integer, Integer> players; // user, score
4 v   public Leaderboard() {
5     players = new HashMap<>();
6     freq = new HashMap<>();
7   }
8 v   public void addScore(int playerId, int score) {
9     if (!players.containsKey(playerId)) players.put(playerId, 0);
10    int oriScore = players.get(playerId), newScore = oriScore + score;
11    if (oriScore > 0) freq.put(oriScore, freq.get(oriScore) - 1);
12    freq.put(newScore, freq.getOrDefault(newScore, 0) + 1);
13    players.put(playerId, newScore);
14  }
15 v   public int top(int K) {
16     int[] scores = new int[freq.size()];
17     int i = 0;
18     for (Integer score : freq.keySet()) scores[i++] = score;
19     Arrays.sort(scores);
20     int sum = 0;
21     for (int j = scores.length - 1; j >= 0; j--) {
22       if (K == 0) return sum;
23       int cnt = freq.get(scores[j]);
24       while (cnt-- > 0 && K-- > 0) sum += scores[j]; // cnt first
25     }
26     return sum;
27   }
28 v   public void reset(int playerId) {
29     int score = players.get(playerId);
30     freq.put(score, freq.get(score) - 1);
31     players.put(playerId, 0);
32   }
33 }
```

## 1245. 树的直径

### 1245. 树的直径

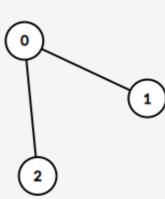
难度 中等 通过 28 提交 28 通过 20 通过 4 失败 4

给你这棵「无向树」，请你测算并返回它的「直径」：这棵树上最长简单路径的边数。

我们用一个由所有「边」组成的数组 edges 来表示一棵无向树，其中 edges[i] = [u, v] 表示节点 u 和 v 之间的双向边。

树上的节点都已经用 {0, 1, ..., edges.length} 中的数做了标记，每个节点上的标记都是独一无二的。

#### 示例 1：



输入：edges = [[0,1],[0,2]]  
输出：2  
解释：

这棵树上最长的路径是 1 - 0 - 2，边数为 2。

#### 示例 2：

```

1 v class Solution {
2   public int treeDiameter(int[][] edges) {
3     if (edges == null || edges.length == 0 || edges[0].length == 0) {
4       return 0;
5     }
6     List<List<Integer>> graph = new ArrayList<>();
7     for (int i = 0; i < edges.length + 1; i++) {
8       graph.add(new ArrayList<Integer>());
9     }
10    for (int[] edge : edges) {
11      graph.get(edge[0]).add(edge[1]);
12      graph.get(edge[1]).add(edge[0]);
13    }
14    int[] ans = new int[1];
15    boolean[] visited = new boolean[graph.size()];
16    dfs(graph, 0, visited, ans);
17    return ans[0];
18  }
19 v private int dfs(List<List<Integer>> graph, int idx, boolean[] visited, int[] max) {
20   visited[idx] = true;
21   List<Integer> nodes = graph.get(idx);
22   int max1 = 0;
23   int max2 = 0;
24 v   for (int next : nodes) {
25     if (visited[next]) {
26       continue;
27     }
28     int dist = dfs(graph, next, visited, max);
29 v     if (dist > max1) {
30       max2 = max1;
31       max1 = dist;
32     } else if (dist > max2) {
33       max2 = dist;
34     }
35   }
36   max[0] = Math.max(max[0], max1 + max2);
37   return Math.max(max1, max2) + 1;
38 }
39 }
```

## 1246. 删除回文子数组

### 1246. 删除回文子数组

难度 困难 通过 19 提交 28 通过 10 通过 4 失败 4

给你一个整数数组 arr，每一次操作你都可以选择并删除它的一个回文子数组 arr[i], arr[i+1], ..., arr[j] ( $i \leq j$ )。

注意，每当你删除掉一个子数组，右侧元素都会自行向前移动填补空位。

请你计算并返回从数组中删除所有数字所需的最少操作次数。

#### 示例 1：

输入：arr = [1,2]  
输出：2

#### 示例 2：

输入：arr = [1,3,4,1,5]  
输出：3  
解释：先删除 [4]，然后删除 [1,3,1]，最后再删除 [5]。

```

1 v class Solution {
2   public int minimumMoves(int[] arr) {
3     int len = arr.length;
4     int[][] dp = new int[len][len];
5     for (int i = 0; i < len; i++) {
6       dp[i][i] = 1;
7     }
8 v     for (int j = 1; j < len; j++) {
9       for (int i = j - 1; i >= 0; i--) {
10         if (i == j - 1) {
11           dp[i][j] = arr[i] == arr[j] ? 1 : 2;
12           continue;
13         }
14         int min = Integer.MAX_VALUE;
15         if (arr[i] == arr[j]) {
16           min = dp[i + 1][j - 1];
17         }
18         for (int k = i; k < j; k++) {
19           min = Math.min(min, dp[i][k] + dp[k + 1][j]);
20         }
21         dp[i][j] = min;
22       }
23     }
24   return dp[0][len - 1];
25 }
```

## 1256. 加密数字

1256. 加密数字

难度 中等 通过数 10 收藏 复制 举报

给你一个非负整数 `num`，返回它的「加密字符串」。

加密的过程是把一个整数用某个未知函数进行转化，你需要从下表推测出该转化函数：

n	f(n)
0	""
1	"0"
2	"1"
3	"00"
4	"01"
5	"10"
6	"11"
7	"000"

```
1 class Solution {
2     public String encode(int num) {
3         return Integer.toBinaryString(num + 1).substring(1);
4     }
5 }
6
7
8
9
```

## 1257. 最小公共区域

1257. 最小公共区域

难度 中等 通过数 9 收藏 复制 举报

给你一些区域列表 `regions`，每个列表的第一个区域都包含这个列表内所有其他区域。

很自然地，如果区域 `x` 包含区域 `y`，那么区域 `x` 比区域 `y` 大。

给定两个区域 `region1` 和 `region2`，找到同时包含这两个区域的最小区域。

如果区域列表中 `r1` 包含 `r2` 和 `r3`，那么数据保证 `r2` 不会包含 `r3`。

数据同样保证最小公共区域一定存在。

示例 1：

```
输入：  
regions = [["Earth","North America","South
```

```
1 class Solution {
2     public String findSmallestRegion(List<List<String>> regions,
3         String region1, String region2) {
4         if (regions.size() == 0) return null;
5         Map<String, String> map = new HashMap<>();
6         for (List<String> reg : regions) {
7             String str = reg.get(0);
8             for (int i = 1; i < reg.size(); i++) {
9                 map.put(reg.get(i), str);
10            }
11        }
12        Set<String> ll = new HashSet<>();
13        while (map.get(region1) != null) {
14            ll.add(region1);
15            region1 = map.get(region1);
16        }
17        ll.add(region1);
18        while (region2 != null) {
19            if (ll.contains(region2)) return region2;
20            region2 = map.getOrDefault(region2, null);
21        }
22    }
23 }
24 }
```

## 1258. 近义词句子

### 1258. 近义词句子

难度 中等  
通过 11 提交 20  
  
给你一个近义词表 `synonyms` 和一个句子 `text`，`synonyms` 表中是一些近义词对，你可以将句子 `text` 中每个单词用它的近义词来替换。  
请你找出所有用近义词替换后的句子，按字典序排序 后返回。

示例 1：

```
输入:  
synonyms = [["happy","joy"],["sad","sorrow"],  
["joy","cheerful"]],  
text = "I am happy today but was sad yesterday"  
输出:  
["I am cheerful today but was sad yesterday",  
"I am cheerful today but was sorrow yesterday",  
"I am happy today but was sad yesterday",  
"I am happy today but was sorrow yesterday",  
"I am joy today but was sad yesterday",  
"I am joy today but was sorrow yesterday"]
```

提示：

- $0 \leq \text{synonyms.length} \leq 10$
- $\text{synonyms}[i].length == 2$
- $\text{synonyms}[0] != \text{synonyms}[1]$
- 所有单词仅包含英文字母，且长度最多为 10。
- `text` 最多包含 10 个单词，且单词间用单个空格分隔。

通过次数 690 | 提交次数 1,240

在真实的面试中遇到过这道题？

是

否

贡献者

```
1 class Solution {  
2     public List<String> generateSentences(List<List<String>> synonyms, String text) {  
3         Map<String, Set<String>> synonymMap = new HashMap();  
4         for (List<String> synonym : synonyms) {  
5             for (String wordOne : synonym) {  
6                 Set<String> synonymSet = synonymMap.containsKey(wordOne)  
7                     ? synonymMap.get(wordOne) : new HashSet();  
8                 if (synonymMap.containsKey(wordOne)) {  
9                     for (String other : synonymSet) {  
10                         if (other.equals(wordOne)) {  
11                             Set<String> otherSet = synonymMap.containsKey(other)  
12                                 ? synonymMap.get(other) : new HashSet();  
13                             otherSet.add(wordTwo);  
14                             synonymMap.put(other, otherSet);  
15                         }  
16                     }  
17                 }  
18                 synonymSet.add(wordTwo);  
19                 synonymMap.put(wordOne, synonymSet);  
20             }  
21         }  
22         List<String> ans = new ArrayList();  
23         String[] words = text.split(" ");  
24         String[] res = new String[words.length];  
25         dfs(words, res, synonymMap, ans, 0);  
26         Collections.sort(ans);  
27         return ans;  
28     }  
29     private void dfs(String[] words, String[] res, Map<String, Set<String>> synonymMap,  
30                     List<String> ans, int index) {  
31         if (index == words.length) {  
32             ans.add(String.join(" ", res));  
33             return;  
34         }  
35         if (!synonymMap.containsKey(words[index])) {  
36             res[index] = words[index];  
37             dfs(words, res, synonymMap, ans, index + 1);  
38         } else {  
39             for (String synonym : synonymMap.get(words[index])) {  
40                 res[index] = synonym;  
41                 dfs(words, res, synonymMap, ans, index + 1);  
42             }  
43         }  
44     }  
45 }  
46 }  
47 }
```

## 1259. 不相交的握手

### 1259. 不相交的握手

难度 困难  
通过 9 提交 20

偶数个人站成一个圆，总人数为 `num_people`。每个人与除自己外的一个人握手，所以总共有  $\frac{\text{num\_people}}{2}$  次握手。

将握手的人之间连线，请你返回连线不会相交的握手方案数。

由于结果可能会很大，请你返回答案 模  $10^{9+7}$  后的结果。

示例 1：

```
1 class Solution {  
2     public int numberOfWays(int n) {  
3         long[] dp = new long[n / 2 + 2];  
4         dp[1] = 1;  
5         long mod = (long) 1e9 + 7, res = 1;  
6         for (int i = 2; i < n / 2 + 2; ++i) {  
7             dp[i] = mod - mod / i * dp[(int) mod % i] % mod;  
8         }  
9         for (int i = 1; i <= n / 2; ++i) {  
10            res = res * (i + n / 2) % mod;  
11            res = res * dp[i] % mod;  
12        }  
13        return (int) (res * dp[n / 2 + 1] % mod);  
14    }  
15 }
```

## 1265. 逆序打印不可变链表

### 1265. 逆序打印不可变链表

难度 中等  
通过 5 提交 20

给您一个不可变的链表，使用下列接口逆序打印每个节点的值：

- `ImmutableListNode`：描述不可变链表的接口，链表的头节点已给出。

您需要使用以下函数来访问此链表（您不能直接访问 `ImmutableListNode`）：

- `ImmutableListNode.printValue()`：打印当前节点的值。
- `ImmutableListNode.getNext()`：返回下一个节点。

输入只用来内部初始化链表。您不可以通过修改链表解决问题。也就是说，您只能通过上述 API 来操作链表。

```
1 class Solution {  
2     public void printLinkedListInReverse(ImmutableListNode head) {  
3         if (head == null) {  
4             return;  
5         }  
6         printLinkedListInReverse(head.getNext());  
7         head.printValue();  
8     }  
9 }
```

## 1271. 十六进制魔术数字

### 1271. 十六进制魔术数字

难度 简单 ⚡ 4 ❤ 10 🌟 1 🏆 1

你有一个十进制数字，请按照此规则将它变成「十六进制魔术数字」：首先将它变成字母大写的十六进制字符串，然后将所有的数字 0 变成字母 O，将数字 1 变成字母 I。

如果一个数字在转换后只包含 {"A", "B", "C", "D", "E", "F", "I", "O"}，那么我们就认为这个转换是有效的。

给你一个字符串 num，它表示一个十进制数 N，如果它的十六进制魔术数字转换是有效的，请返回转换后的结果，否则返回 "ERROR"。

示例 1：

```
输入: num = "257"
输出: "IOI"
解释: 257 的十六进制表示是 101。
```

示例 2：

1272. 删区间

### 1272. 删区间

难度 中等 ⚡ 8 ❤ 10 🌟 1 🏆 1

给你一个有序的不相交区间列表 intervals 和一个要删除的区间 toBeRemoved，intervals 中的每一个区间 intervals[i] = [a, b] 都表示满足  $a \leq x < b$  的所有实数 x 的集合。

我们将 intervals 中任意区间与 toBeRemoved 有交集的部分都删除。

返回删除所有交集区间后，intervals 剩余部分的有序列表。

示例 1：

```
输入: intervals = [[0,2],[3,4],[5,7]], toBeRemoved = [1,6]
输出: [[0,1],[6,7]]
```

示例 2：

```
输入: intervals = [[0,5]], toBeRemoved = [2,3]
输出: [[0,2],[3,5]]
```

```
1 class Solution {
2     public String toHexspeak(String num) {
3         char[] ches = num.toCharArray();
4         long n = 0;
5         for (int i = 0; i < ches.length; i++) {
6             n *= 10;
7             n += (ches[i] - '0');
8         }
9         StringBuilde buf = new StringBuilde();
10        ches = new char[]{ 'A', 'B', 'C', 'D', 'E', 'F' };
11        for (long mask = 15; n > 0; n = n >> 4) {
12            int s = (int) (n & mask);
13            if (s > 1 && s < 10) {
14                return "ERROR";
15            }
16            if (s == 0) {
17                buf.append('0');
18            } else if (s == 1) {
19                buf.append('I');
20            } else {
21                buf.append(ches[s - 10]);
22            }
23        }
24        return buf.reverse().toString();
25    }
26 }
```

```
1 class Solution {
2     public List<List<Integer>> removeInterval(int[][] intervals, int[] toBeRemoved) {
3         List<List<Integer>> res = new ArrayList<>();
4         for (int i = 0; i < intervals.length; i++) {
5             if (intervals[i][0] >= toBeRemoved[1] || intervals[i][1] <= toBeRemoved[0]) {
6                 res.add(buildInterval(intervals[i][0], intervals[i][1]));
7             } else if (intervals[i][0] < toBeRemoved[0]) {
8                 int start = intervals[i][0];
9                 int end = toBeRemoved[0];
10                if (start < end) {
11                    res.add(buildInterval(start, end));
12                }
13                intervals[i][0] = toBeRemoved[0];
14                i--;
15            } else if (intervals[i][1] > toBeRemoved[1]) {
16                int start = toBeRemoved[1];
17                res.add(buildInterval(start, intervals[i][1]));
18            }
19        }
20        return res;
21    }
22    private List<Integer> buildInterval(int start, int end) {
23        List<Integer> list = new ArrayList<>();
24        list.add(start);
25        list.add(end);
26        return list;
27    }
28 }
29 }
```

### 1273. 删除树节点

#### 1273. 删除树节点

难度 中等 通过数 10 提交

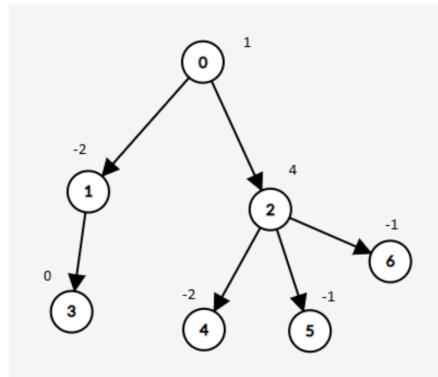
给你一棵以节点 0 为根节点的树，定义如下：

- 节点的总数为 nodes 个；
- 第 i 个节点的值为 value[i]；
- 第 i 个节点的父节点是 parent[i]。

请你删除节点值之和为 0 的每一棵子树。

在完成所有删除之后，返回树中剩余节点的数目。

示例：



输入: nodes = 7, parent = [-1,0,0,1,2,2,2], value = [1,-2,4,0,-2,-1,-1]  
输出: 2

```
1 v class Solution {
2 v     public int deleteTreeNodes(int num, int[] parent, int[] value) {
3 v         List<Set<Integer>> children = new ArrayList<>();
4 v         for (int i = 0; i < num; i++) {
5 v             children.add(new HashSet<>());
6 v         }
7 v         // 子节点表示法：如本题目示例中，根节点有两个子节点，分别为1和2
8 v         for (int i = 1; i < num; i++) {
9 v             children.get(parent[i]).add(i);
10 v        }
11 v        // 结算以0为根节点的“删除和为0的所有子树”后剩余结点数量
12 v        if (dfs(0, value, children) == 0) {
13 v            return 0;
14 v        }
15 v        // 广度优先遍历所有的剩余结点的数量
16 v        Queue<Integer> queue = new LinkedList<>();
17 v        queue.offer(0);
18 v        int cnt = 0;
19 v        while (!queue.isEmpty()) {
20 v            int idx = queue.poll();
21 v            for (int child : children.get(idx)) {
22 v                queue.offer(child);
23 v            }
24 v            cnt++;
25 v        }
26 v        return cnt;
27 v    }
28 v    // 递归计算：以idx为根节点的“删除和为0的所有子树”后剩余结点数量
29 v    private int dfs(int idx, int[] value, List<Set<Integer>> children) {
30 v        int sum = value[idx];
31 v        List<Integer> toDeleteList = new ArrayList<>();
32 v        for (int child : children.get(idx)) {
33 v            int tmp = dfs(child, value, children);
34 v            // 以child为父节点的子树的所有结点之和为0，则直接删除child结点
35 v            // 因为是有向无环图，父节点删除后，子节点无法被访问
36 v            if (tmp == 0) {
37 v                toDeleteList.add(child);
38 v            }
39 v            sum += tmp;
40 v        }
41 v        for (int i : toDeleteList) {
42 v            children.get(idx).remove(i);
43 v        }
44 v        return sum;
45 v    }
46 v}
```

### 1274. 矩形内船只的数目

#### 1274. 矩形内船只的数目

难度 困难 通过数 11 提交

(此题是 交互式问题)

在用笛卡尔坐标系表示的二维海平面上，有一些船。每一艘船都在一个整数点上，且每一个整数点最多只有 1 艘船。

有一个函数 Sea.hasShips(topRight, bottomLeft)，输入参数为右上角和左下角两个点的坐标。当且仅当这两个点所表示的矩形区域（包含边界）内至少有一艘船时，这个函数才返回 true，否则返回 false。

给你矩形的右上角 topRight 和左下角 bottomLeft 的坐标，请你返回此矩形内船只的数目。题目保证矩形内 至多只有 10 艘船。

调用函数 hasShips 超过400 次的提交将被判为 错误答案 (Wrong Answer)。同时，任何尝试绕过评测系统的行为都将被取消比赛资格。

示例：



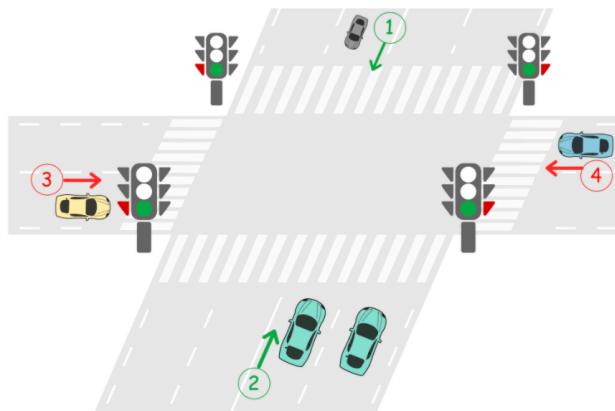
```
1 v public class Solution {
2 v     public int countShips(Sea sea, int[] topRight, int[] bottomLeft) {
3 v         // 用 X 表示横坐标，用 Y 表示纵坐标
4 v         int topRightX = topRight[0];
5 v         int topRightY = topRight[1];
6 v         int bottomLeftX = bottomLeft[0];
7 v         int bottomLeftY = bottomLeft[1];
8 v         // 先写递归终止条件，即考虑什么样的右上角的点和左下角的点不构成区间
9 v         // 1. 不构成区间：(1) 左下角的点的横坐标 > 右上角的点的横坐标；(2) 左下角的点的纵坐标 > 右上角的点的纵坐标
10 v        // 2. 像扫雷一样，扫不到船只，就返回 0
11 v        if (bottomLeftX > topRightX || bottomLeftY > topRightY
12 v            || !sea.hasShips(topRight, bottomLeft)) {
13 v            return 0;
14 v        }
15 v        int res = 0;
16 v        if (bottomLeftX == topRightX && bottomLeftY == topRightY) {
17 v            return 1;
18 v        }
19 v        // 分治
20 v        int midX = (bottomLeftX + topRightX) >>> 1;
21 v        int midY = (bottomLeftY + topRightY) >>> 1;
22 v        // 左上角
23 v        res += countShips(sea, new int[]{midX, topRightY}, new int[]{bottomLeftX, midY + 1});
24 v        // 右上角
25 v        res += countShips(sea, new int[]{topRightX, topRightY}, new int[]{midX + 1, midY + 1});
26 v        // 右下角
27 v        res += countShips(sea, new int[]{topRightX, midY}, new int[]{midX + 1, bottomLeftY});
28 v        // 左下角
29 v        res += countShips(sea, new int[]{midX, midY}, new int[]{bottomLeftX, bottomLeftY});
30 v        return res;
31 v    }
32 v}
```

## 1279. 红绿灯路口

### 1279. 红绿灯路口

难度 简单 ✓ 1 ❤️ ⚡ A ⚡

这是两条路的交叉路口。第一条路是 A 路，车辆可沿 1 号方向由北向南行驶，也可沿 2 号方向由南向北行驶。第二条路是 B 路，车辆可沿 3 号方向由西向东行驶，也可沿 4 号方向由东向西行驶。



## 1426. 数元素

### 1426. 数元素

难度 简单 ✓ 0 ❤️ ⚡ A ⚡

给你一个整数数组 arr，对于元素 x，只有当  $x + 1$  也在数组 arr 里时，才能记为 1 个数。

如果数组 arr 里有重复的数，每个重复的数单独计算。

示例 1：

输入: arr = [1,2,3]  
输出: 2

解释: 1 和 2 被计算次数因为 2 和 3 在数组 arr 里。

## 1427. Perform String Shifts

### 1427. Perform String Shifts

难度 简单 ✓ 0 ❤️ ⚡ A ⚡

You are given a string s containing lowercase English letters, and a matrix shift, where  $\text{shift}[i] = [\text{direction}, \text{amount}]$ :

- direction can be 0 (for left shift) or 1 (for right shift).
- amount is the amount by which string s is to be shifted.
- A left shift by 1 means remove the first character of s and append it to the end.
- Similarly, a right shift by 1 means remove the last character of s and add it to the beginning.

Return the final string after all operations.

## 1428. Leftmost Column with at Least a One

### 1428. Leftmost Column with at Least a One

难度 中等 ✓ 0 ❤️ ⚡ A ⚡

(This problem is an *interactive problem*.)

A binary matrix means that all elements are 0 or 1. For each individual row of the matrix, this row is sorted in non-decreasing order.

Given a row-sorted binary matrix binaryMatrix, return leftmost column index(0-indexed) with at least a 1 in it. If such index doesn't exist, return -1.

You can't access the Binary Matrix directly. You may only access the matrix using a `BinaryMatrix` interface:

- `BinaryMatrix.get(row, col)` returns the element of the matrix at index (row, col) (0-indexed).
- `BinaryMatrix.dimensions()` returns a list of 2 elements [rows, cols], which means the matrix is rows \* cols.

```

1 class TrafficLight {
2     private int mCurrent;
3
4     public TrafficLight() {
5         mCurrent = 1;
6     }
7
8     public void carArrived(
9         int carId,           // ID
10        int roadId,          // ID
11        int direction,       // Dir
12        Runnable turnGreen, // Use
13        Runnable crossCar   // Use
14    ) {
15        synchronized (this) {
16            if (mCurrent == roadId) {
17                crossCar.run();
18            } else {
19                mCurrent = roadId;
20                turnGreen.run();
21                crossCar.run();
22            }
23        }
24    }
25 }
```

```

1 class Solution {
2     public int countElements(int[] arr) {
3         int[] times = new int[1002];
4         for (int i = 0; i < arr.length; i++) {
5             times[arr[i]]++;
6         }
7         int ans = 0;
8         for (int i = 0; i < arr.length; i++) {
9             if (times[arr[i]] > 0 && times[arr[i] + 1] > 0) {
10                 ans++;
11             }
12         }
13     }
14     return ans;
15 }
16 }
```

```

1 class Solution {
2     public String stringShift(String s, int[] shift) {
3         int len = s.length();
4         int leftShift = 0;
5         for (int[] sft : shift) {
6             if (sft[0] == 0) {
7                 leftShift += sft[1];
8             } else {
9                 leftShift -= sft[1];
10            }
11        }
12        leftShift %= len;
13        if (leftShift < 0) {
14            leftShift += len;
15        }
16        return s.substring(leftShift) + s.substring(0, leftShift);
17    }
18 }
```

```

1 class Solution {
2     public int leftMostColumnWithOne(BinaryMatrix binaryMatrix) {
3         List<Integer> dim = binaryMatrix.dimensions();
4         int n = dim.get(0);
5         int m = dim.get(1);
6         int lid = 0;
7         int rid = m - 1;
8         int min = m;
9         while (lid < n && rid >= 0) {
10            if (binaryMatrix.get(lid, rid) == 0) {
11                lid++;
12            } else {
13                min = Math.min(min, rid);
14                rid--;
15            }
16        }
17        return min == m ? -1 : min;
18    }
19 }
20 }
```

## 1429. First Unique Number

### 1429. First Unique Number

难度 中等

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the `FirstUnique` class:

- `FirstUnique(int[] nums)` Initializes the object with the numbers in the queue.
- `int showFirstUnique()` returns the value of the first unique integer of the queue, and returns -1 if there is no such integer.
- `void add(int value)` insert value to the queue.

**Example 1:**

```
Input:
["FirstUnique","showFirstUnique","add","showFirstUnique"]
[[[2,3,5]],[],[],[2],[],[3],[]]
Output:
[null,2,null,2,null,3,null,-1]
Explanation:
FirstUnique firstUnique = new
FirstUnique([2,3,5]);
firstUnique.showFirstUnique(); // return 2
firstUnique.add(5);          // the queue is now
[2,3,5,5]
```

## 1430. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

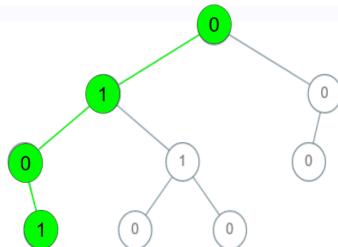
### 1430. Check If a String Is a Valid Sequence from Root to Leaves Path

难度 中等

Given a binary tree where each path going from the root to any leaf form a **valid sequence**, check if a given string is a **valid sequence** in such binary tree.

We get the given string from the concatenation of an array of integers `arr` and the concatenation of all values of the nodes along a path results in a **sequence** in the given binary tree.

**Example 1:**



```
Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr =
[0,1,0,1]
Output: true
```

```
1 class FirstUnique {
2     private List<Integer> mList;
3     private Map<Integer, Integer> mMap;
4     public FirstUnique(int[] nums) {
5         this.mMap = new HashMap<>();
6         this.mList = new ArrayList<>();
7         for (int num : nums) {
8             int cnt = mMap.getOrDefault(num, 0);
9             if (cnt == 0) {
10                 mList.add(num);
11             }
12             mMap.put(num, cnt + 1);
13         }
14     }
15     public int showFirstUnique() {
16         if (mList.isEmpty()) {
17             return -1;
18         }
19         for (int num : mList) {
20             if (mMap.get(num) == 1) {
21                 return num;
22             }
23         }
24         return -1;
25     }
26     public void add(int value) {
27         int cnt = mMap.getOrDefault(value, 0);
28         if (cnt == 0) {
29             mList.add(value);
30         }
31         mMap.put(value, cnt + 1);
32     }
33 }
```

```
1 class Solution {
2     public boolean isValidSequence(TreeNode root, int[] arr) {
3         return checkValidSequence(root, arr, 0);
4     }
5
6     private boolean checkValidSequence(TreeNode root, int[] arr, int index) {
7         // 数值不相同，返回false
8         if (root == null || root.val != arr[index]) {
9             return false;
10        }
11        // 最后一个元素的特判，必须是叶子节点
12        if (index == arr.length - 1) {
13            return root.left == null && root.right == null;
14        }
15        // 递归判断左右子树
16        return checkValidSequence(root.left, arr, index + 1)
17            || checkValidSequence(root.right, arr, index + 1);
18    }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36 }
```

## 770. 基本计算器 IV

### 770. 基本计算器 IV

难度 困难 收藏 分享 切换为英文 关注 反馈

给定一个表达式 expression 如 expression = "e + 8 - a + 5" 和一个求值映射，如 {"e": 1} (给定的形式为 evalvars = ["e"] 和 evalints = [1] )，返回表示简化表达式的标记列表，例如 [-1\*a, "14"]

- 表达式交替使用块和符号。每个块和符号之间有一个空格。
  - 块要么是括号中的表达式，要么是变量，要么是非负整数。
  - 块是括号中的表达式，变量或非负整数。
  - 变量是一个由小写字母组成的字符串（不包括数字）。请注意，变量可以是多个字母，并注意变量从不具有像 “2x” 或 “-x” 这样的前导系数或一元运算符。
- 表达式按通常顺序进行求值：先是括号，然后求乘法，再计算加法和减法。例如，expression = "1 + 2 \* 3" 的答案是 ["7"]。

输出格式如下：

对于系数非零的每个自变量项，我们按字典排序的顺序将自变量写在一个项中。例如，我们永远不会写像 “b\*a\*c” 这样的项，只写 “a\*b\*c”。

项的次数等于被乘的自变量的数目，并计算重复项。（例如，“a\*a\*b\*c”的次数为4。），我们先写出答案的最大次数项，用字典顺序打破关系，此时忽略词的前导系数。

项的前导系数直接放在左边，用星号将它与变量分隔开（如果存在的话）。前导系数1仍然要打印出来。

格式良好的一个示例答案是 [-2\*a\*a\*a, "3\*a\*a\*b", "3\*a\*b", "4\*a", "5\*c", "-"]。

系数为0的项（包括常数项）不包括在内。例如，“0”的表达式输出为[]。

```
10  @ public List<String> basicCalculatorIV(String expression, String[] evalvars, int[] evalints) {
11      ....
12      ....
13      ....
14      ....
15      ....
16      ....
17      ....
18      ....
19      ....
20      ....
21      ....
22      ....
23      ....
24      ....
25      ....
26      ....
27      ....
28      ....
29      ....
30      ....
31      ....
32      ....
33      ....
34      ....
35      ....
36      ....
37      ....
38      ....
39      ....
40      ....
41      ....
42      ....
43      ....
44      ....
45      ....
46      ....
47      ....
48      ....
49      ....
50      ....
51      ....
52      ....
53      ....
54      ....
55      ....
56      ....
57      ....
58      ....
59      ....
60      ....
61      ....
62      ....
63      ....
64      ....
65      ....
66      ....
67      ....
68      ....
69      ....
70      ....
71      ....
72      ....
73      ....
74      ....
75      ....
76      ....
77      ....
78      ....
79      ....
80      ....
81      ....
82      ....
83      ....
84      ....
85      ....
86      ....
87      ....
```

```

89  class Monomial implements Comparable<Monomial> {
90      int coeff;
91      ArrayList<String> factors;
92      public Monomial(int coeff) {
93          this.coeff = coeff;
94          this.factors = new ArrayList<>();
95      }
96      public Monomial() {
97          this.coeff = 0;
98          this.factors = new ArrayList<>();
99      }
100     public Monomial(int coeff, String f) {
101         this.coeff = coeff;
102         this.factors = new ArrayList<>();
103         factors.add(f);
104     }
105     @Override
106     public Monomial mul(Monomial item) {
107         Monomial res = new Monomial();
108         res.coeff = coeff * item.coeff;
109         res.factors.addAll(factors);
110         res.factors.addAll(item.factors);
111         res.factors.sort(String::compareTo);
112         return res;
113     }
114     @Override
115     public int compareTo(Monomial item) {
116         if (factors.size() == item.factors.size()) {
117             int idx = 0, len = factors.size();
118             while (idx < len && factors.get(idx).compareTo(item.factors.get(idx)) == 0)
119                 idx++;
120             return idx == len ? 0 : factors.get(idx).compareTo(item.factors.get(idx));
121         } else {
122             return item.factors.size() - factors.size();
123         }
124     }
125     @Override
126     public String toString() {
127         StringBuilder buffer = new StringBuilder();
128         buffer.append(coeff);
129         for (String factor : factors) {
130             buffer.append("*").append(factor);
131         }
132         return buffer.toString();
133     }
134
135     private class Polynomial {
136         ArrayList<Monomial> items;
137         public Polynomial(Monomial item) {
138             this.items = new ArrayList<>();
139             items.add(item);
140         }
141         void add(Polynomial pol) {
142             items.addAll(pol.items);
143             items.sort(Monomial::compareTo);
144             clean();
145         }
146         void mul(Polynomial pol) {
147             ArrayList<Monomial> res = new ArrayList<>();
148             for (Monomial item1 : items)
149                 for (Monomial item2 : pol.items)
150                     res.add(item1.mul(item2));
151             this.items = res;
152             items.sort(Monomial::compareTo);
153             clean();
154         }
155         Polynomial clean() {
156             for (int idx = 0; idx < items.size(); idx++) {
157                 while (idx + 1 < items.size() && items.get(idx).compareTo(items.get(idx + 1)) == 0) {
158                     items.get(idx).coeff += items.get(idx + 1).coeff;
159                     items.remove(index: idx + 1);
160                 }
161                 if (idx < items.size() && items.get(idx).coeff == 0)
162                     items.remove(idx--);
163             }
164             return this;
165         }
166         Polynomial operate(Polynomial pol, String opr) {
167             switch (opr) {
168                 case "+":
169                     mul(pol);
170                     break;
171                 case "-":
172                     add(pol);
173                     break;
174                 case "*":
175                     for (Monomial item : pol.items)
176                         item.coeff *= -1;
177                     add(pol);
178                     break;
179             }
180             return this;
181         }
182     }

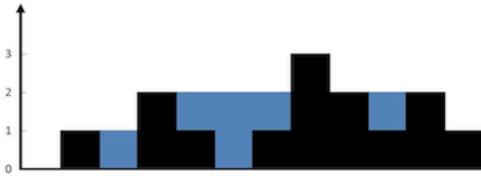
```

## 42. 接雨水

42. 接雨水

难度 困难 1254

给定  $n$  个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。



上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。感谢 Marcos 贡献此图。

示例:

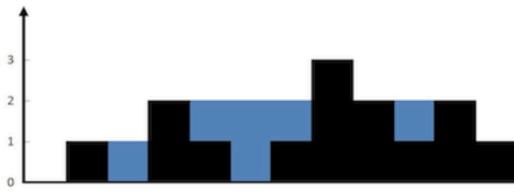
输入: [0,1,0,2,1,0,1,3,2,1,2,1]

输出: 6

## 42. 接雨水

难度 困难 1254

给定  $n$  个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。



上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。感谢 Marcos 贡献此图。

示例:

输入: [0,1,0,2,1,0,1,3,2,1,2,1]

输出: 6

## 407. 接雨水 II (BFS)

407. 接雨水 II

难度 困难 170

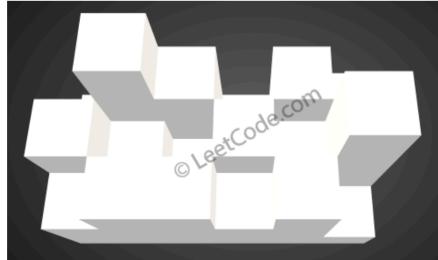
给你一个  $m \times n$  的矩阵，其中的值均为非负整数，代表二维高度图每个单元的高度，请计算图中形状最多能接多少体积的雨水。

示例:

给出如下  $3 \times 6$  的高度图:

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

返回 4。



如上图所示，这是下雨前的高度图 [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]] 的状态。

```
1 class Solution { // 栈应用
2     public int trap(int[] height) {
3         int sum = 0;
4         int cid = 0; // 当前位置下标
5         Stack<Integer> stack = new Stack<Integer>();
6         while (cid < height.length) {
7             // 如果栈不空并且当前指向的高度大于栈顶高度就一直循环
8             while (!stack.empty() && height[cid] > height[stack.peek()]) {
9                 int hgt = height[stack.pop()]; // 出栈
10                if (stack.empty()) { // 空栈，中断循环
11                    break;
12                }
13                int dist = cid - stack.peek() - 1; // 计算距离
14                int min = Math.min(height[stack.peek()], height[cid]);
15                sum = sum + dist * (min - hgt);
16            }
17            stack.push(cid); // 当前位置下标入栈
18            cid++; // 当前位置下标后移
19        }
20        return sum;
21    }
22}
```

```
1 class Solution { // 双指针
2     public int trap(int[] height) {
3         int sum = 0;
4         int[] lMax = new int[height.length];
5         int[] rMax = new int[height.length];
6         for (int idx = 1; idx < height.length - 1; idx++) {
7             lMax[idx] = Math.max(lMax[idx - 1], height[idx - 1]);
8         }
9         for (int idx = height.length - 2; idx > 0; idx--) {
10            rMax[idx] = Math.max(rMax[idx + 1], height[idx + 1]);
11        }
12        for (int idx = 1; idx < height.length - 1; idx++) {
13            int min = Math.min(lMax[idx], rMax[idx]);
14            if (height[idx] < min) {
15                sum += (min - height[idx]);
16            }
17        }
18    }
19    return sum;
20}
21
22
23
24
25
```

```
1 class Solution { // BFS
2     public int trapRainWater(int[][] heights) {
3         if (heights == null || heights.length == 0)
4             return 0;
5         int rows = heights.length, cols = heights[0].length;
6         boolean[][] visited = new boolean[rows][cols];
7         // 优先队列中存放三元组 [x,y,h] 坐标和高度，按高度排序
8         PriorityQueue<int[]> queue = new PriorityQueue<int[]>((o1, o2) -> o1[2] - o2[2]);
9         // 先把最外一圈放进优先队列
10        for (int rid = 0; rid < rows; rid++)
11            for (int cid = 0; cid < cols; cid++)
12                if (rid == 0 || rid == rows - 1 || cid == 0 || cid == cols - 1) {
13                    queue.offer(new int[]{rid, cid, heights[rid][cid]});
14                    visited[rid][cid] = true;
15                }
16        int ans = 0;
17        final int[] xDirs = {-1, 1, 0, 0};
18        final int[] yDirs = {0, 0, -1, 1};
19        while (!queue.isEmpty()) {
20            int[] node = queue.poll();
21            for (int i = 0; i < 4; i++) { // 审视四个方向
22                int x = node[0] + xDirs[i]; // 横坐标x
23                int y = node[1] + yDirs[i]; // 纵坐标y
24                if (x < 0 || x >= rows || y < 0 || y >= cols || visited[x][y])
25                    continue;
26                // 该圈中最小的高度比当前坐标位置的高度更高，说明能往里灌水
27                if (node[2] > heights[x][y])
28                    ans += node[2] - heights[x][y];
29                queue.offer(new int[]{x, y, Math.max(heights[x][y], node[2])});
30                visited[x][y] = true;
31            }
32        }
33    }
34    return ans;
35}
```