总结

1. 第三题没思路果断放弃，才能让第四题有足够的时间做出来，第四题AC之后只剩下五分钟，好险
2. 第四题体现了平时积累的重要性，用到的知识蛮多

重复至少 K 次且长度为 M 的模式（简单）

1. 题目描述

   给你一个正整数数组 `arr`，请你找出一个长度为 `m` 且在数组中至少重复 `k` 次的模式。

   模式 是由一个或多个值组成的子数组（连续的子序列），**连续** 重复多次但 **不重叠** 。 模式由其长度和重复次数定义。

   如果数组中存在一个至少重复 `k` 次且长度为 `m` 的模式，则返回 `true` ，否则返回 `false` 。

   示例 1：

   ```
   输入：arr = [1,2,4,4,4,4], m = 1, k = 3
   输出：true
   解释：模式 (4) 的长度为 1 ，且连续重复 4 次。注意，模式可以重复 k 次或更多次，但不能少于 k 次。
   ```

   示例 2：

   ```
   输入：arr = [1,2,1,2,1,1,1,3], m = 2, k = 2
   输出：true
   解释：模式 (1,2) 长度为 2 ，且连续重复 2 次。另一个符合题意的模式是 (2,1) ，同样重复 2 次。
   ```

   示例 3：

   ```
   输入：arr = [1,2,1,2,1,3], m = 2, k = 3
   输出：false
   解释：模式 (1,2) 长度为 2 ，但是只连续重复 2 次。不存在长度为 2 且至少重复 3 次的模式。
   ```

   示例 4：

   ```
   输入：arr = [1,2,3,1,2], m = 2, k = 2
   输出：false
   解释：模式 (1,2) 出现 2 次但并不连续，所以不能算作连续重复 2 次。
   ```

   示例 5：

   ```
   输入：arr = [2,2,2,2], m = 2, k = 3
   输出：false
   解释：长度为 2 的模式只有 (2,2) ，但是只连续重复 2 次。注意，不能计算重叠的重复次数。
   ```

   提示：

- `2 <= arr.length <= 100`
- `1 <= arr[i] <= 100`
- `1 <= m <= 100`
- `2 <= k <= 100`

2. 比赛实现

```
class Solution {
public:
    string toString(vector<int>& arr, int idx, int len){
        string ans = "";
        for(int i = idx; i < idx+len; i++)
            ans += to_string(arr[i]) + ',';
        return ans;
    }
    bool containsPattern(vector<int>& arr, int m, int k) {
        for(int i = 0; i <= int(arr.size())-m*k; i++){
            string pattern = toString(arr, i, m);
            int cnt = 1;
            int j = i;
            while(cnt < k){
                j += m;
                if(j <= int(arr.size()) - m){
                    string cur = toString(arr, j, m);
                    if(pattern == cur)
                        cnt++;
                    else
                        break;
                }
                else
                    break;
            }
            if(cnt == k)
                return true;
        }
        return false;
    }
};
```

乘积为正数的最长子数组长度（中等）

1. 题目描述

给你一个整数数组 `nums` ，请你求出乘积为正数的最长子数组的长度。

一个数组的子数组是由原数组中零个或者更多个连续数字组成的数组。

请你返回乘积为正数的最长子数组长度。

示例 1：

```
输入: nums = [1,-2,-3,4]
输出: 4
解释: 数组本身乘积就是正数，值为 24 。
```

示例 2:

```
输入: nums = [0,1,-2,-3,-4]
输出: 3
解释: 最长乘积为正数的子数组为 [1,-2,-3] ，乘积为 6 。
注意，我们不能把 0 也包括到子数组中，因为这样乘积为 0 ，不是正数。
```

示例 3:

```
输入: nums = [-1,-2,-3,0,1]
输出: 2
解释: 乘积为正数的最长子数组是 [-1,-2] 或者 [-2,-3] 。
```

示例 4:

```
输入: nums = [-1,2]
输出: 1
```

示例 5:

```
输入: nums = [1,2,3,5,-6,4,0,10]
输出: 4
```

提示:

- `1 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`

2. 比赛实现

`dp[i][0/1]`表示以nums[i]为结尾的，乘积为 正/负 的最长子数组长度

但其实dp[i]只与dp[i-1]有关，因此还能再优化空间，此处略去

```
class Solution {
public:
    int getMaxLen(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>> dp(n, vector<int>(2, 0));
        if(nums[0] > 0){
            dp[0][0] = 1;
            dp[0][1] = 0;
        }
        else if(nums[0] < 0){
            dp[0][0] = 0;
            dp[0][1] = 1;
        }
        else{
```

```
            dp[0][0] = 0;
            dp[0][1] = 0;
        }
        int ans = dp[0][0];
        for(int i = 1; i < n; i++){
            if(nums[i] > 0){
                dp[i][0] = dp[i-1][0] + 1;
                if(dp[i-1][1] != 0)//只有前面能是负，这里才可能是负
                    dp[i][1] = dp[i-1][1] + 1;
                else
                    dp[i][1] = 0;
            }
            else if(nums[i] < 0){
                if(dp[i-1][1] != 0)//只有前面能是正，这里才可能是正
                    dp[i][0] = dp[i-1][1] + 1;
                else
                    dp[i][0] = 0;
                dp[i][1] = dp[i-1][0] + 1;
            }
            else{
                dp[i][0] = 0;
                dp[i][1] = 0;
            }
            ans = max(ans, dp[i][0]);
        }
        return ans;
    }
};
```

使陆地分离的最少天数（中等）

1. 题目描述

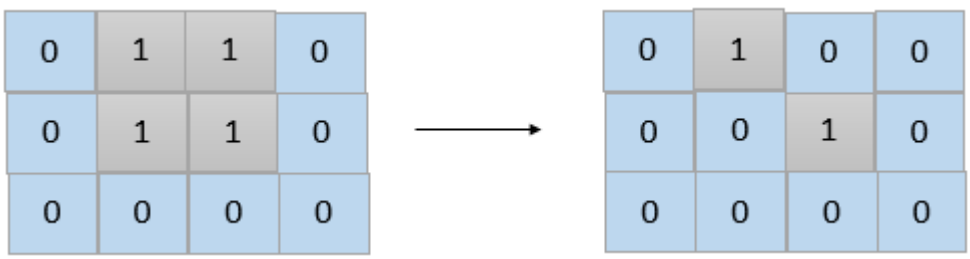
给你一个由若干 `0` 和 `1` 组成的二维网格 `grid` ，其中 `0` 表示水，而 `1` 表示陆地。岛屿由水平方向或竖直方向上相邻的 `1` （陆地）连接形成。

如果 **恰好只有一座岛屿** ，则认为陆地是 **连通的** ；否则，陆地就是 **分离的** 。

一天内，可以将任何单个陆地单元（`1`）更改为水单元（`0`）。

返回使陆地分离的最少天数。

示例 1：



```
输入：grid = [[0,1,1,0],[0,1,1,0],[0,0,0,0]]
输出：2
解释：至少需要 2 天才能得到分离的陆地。
将陆地 grid[1][1] 和 grid[0][2] 更改为水，得到两个分离的岛屿。
```

示例 2：

```
输入：grid = [[1,1]]
输出：2
解释：如果网格中都是水，也认为是分离的 ([[1,1]] -> [[0,0]])，0 岛屿。
```

示例 3：

```
输入：grid = [[1,0,1,0]]
输出：0
```

示例 4：

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

示例 5：

```
输入：grid = [[1,1,0,1,1],
              [1,1,1,1,1],
              [1,1,0,1,1],
              [1,1,1,1,1]]
输出：2
```

提示：

- `1 <= grid.length, grid[i].length <= 30`
- `grid[i][j]` 为 `0` 或 `1`

2. 比赛实现

   没有很明确的思路，担心时间复杂度不够，比赛时在看了困难题可解的情况下，战略性放弃

3. 正确解法

   果然是我担心多了，强行dfs即可。。。不过题解里的桥和角我也确实没推出来，所以不算冤枉

   https://leetcode-cn.com/problems/minimum-number-of-days-to-disconnect-island/solution/5501-shi-lu-di-fen-chi-de-zui-shao-tian-shu-by-lin/

   显然，当原始图有多块岛屿时，这是分离的，返回0。因此我们首先dfs确定连通块的数量。若大于1就返回0。接下来我们思考怎样才能最少地让陆地分离。 显然最多消掉2个陆地就可以分离岛屿了。为什么呢？我们将一个连接两格陆地且位于连通块边缘的陆地称之为"角"。而该陆地消去后连通块分离的陆地称之为“桥”。桥就只需要消掉桥即可分离岛屿。而角则需要将相邻的两个陆地消除即可将角分离出原始岛屿。显然一个联通块至少存在一个角或者桥。 因此我们只需要先检查有没有桥，若有则只需要花费1天即可分离岛屿。否则需要

花费2天将角分离出来。 检查桥的方法则是将该陆地先消掉，dfs确定连通块的数量。若有多个说明这个陆地是桥。 时间复杂度为O(n^4)

```
int dirx[] = {0,0,1,-1};
int diry[] = {-1,1,0,0};
class Solution {
public:
    int n,m;
    vector< vector<int> > copy;
    void dfs(int i,int j){
        if(copy[i][j] == 0) return ;
        copy[i][j] = 0;
        for(int k=0;k<4;k++){
            if(i + dirx[k] >= 0 && i + dirx[k] < n &&  j + diry[k] >= 0 && j +
diry[k] < m){
                dfs(i + dirx[k],j + diry[k]);
            }
        }
        return ;
    }
    int minDays(vector<vector<int>>& grid) {
        n = grid.size(), m = grid[0].size();
        copy = grid;
        int ct = 0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(copy[i][j] != 0) ct++,dfs(i,j);
            }
        }
        if(ct > 1) return 0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j] == 0) continue;
                copy = grid;
                copy[i][j] = 0;
                ct = 0;
                for(int k=0;k<n;k++){
                    for(int l=0;l<m;l++){
                        if(copy[k][l] != 0) ct++,dfs(k,l);
                    }
                }
                if(ct > 1) return 1;
            }
        }
        return 2;
    }
};
```

将子数组重新排序得到同一个二叉查找树的方案数（困难）

1. 题目描述

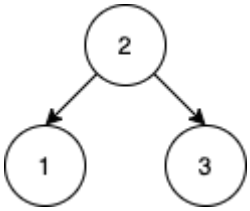
给你一个数组 `nums` 表示 `1` 到 `n` 的一个排列。我们按照元素在 `nums` 中的顺序依次插入一个初始为空的二叉查找树（BST）。请你统计将 `nums` 重新排序后，统计满足如下条件的方案数：重排后得到的二叉查找树与 `nums` 原本数字顺序得到的二叉查找树相同。

比方说，给你 `nums = [2,1,3]`，我们得到一棵 2 为根，1 为左孩子，3 为右孩子的树。数组 `[2,3,1]` 也能得到相同的 BST，但 `[3,2,1]` 会得到一棵不同的 BST 。

请你返回重排 `nums` 后，与原数组 `nums` 得到相同二叉查找树的方案数。

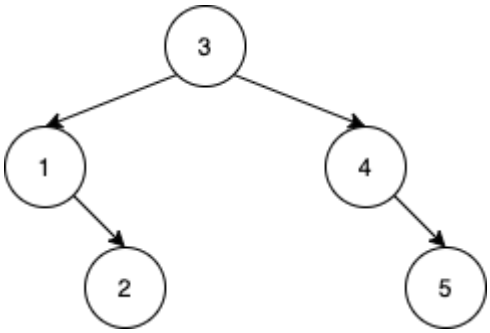
由于答案可能会很大，请将结果对 `10^9 + 7` 取余数。

示例 1：



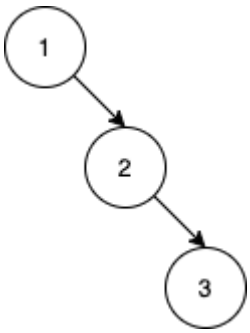

```
输入：nums = [2,1,3]
输出：1
解释：我们将 nums 重排， [2,3,1] 能得到相同的 BST 。没有其他得到相同 BST 的方案了。
```

示例 2：



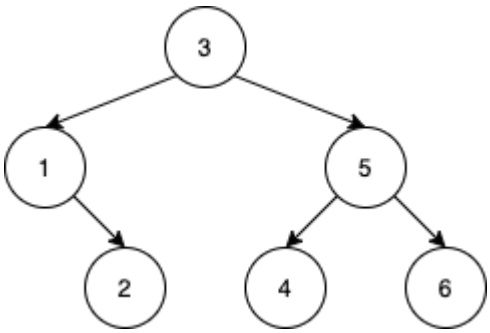

```
输入：nums = [3,4,5,1,2]
输出：5
解释：下面 5 个数组会得到相同的 BST：
[3,1,2,4,5]
[3,1,4,2,5]
[3,1,4,5,2]
[3,4,1,2,5]
[3,4,1,5,2]
```

示例 3：



```
输入：nums = [1,2,3]
输出：0
解释：没有别的排列顺序能得到相同的 BST 。
```

示例 4：




```
输入：nums = [3,1,2,5,4,6]
输出：19
```

示例 5：

```
输入：nums = [9,4,2,1,3,6,5,7,8,14,11,10,12,13,16,15,17,18]
输出：216212978
解释：得到相同 BST 的方案数是 3216212999。将它对 10^9 + 7 取余后得到 216212978。
```

提示：

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <= nums.length`
- `nums` 中所有数 **互不相同** 。

2. 比赛实现

观察例2，可以发现递归规律：

- 要求以root为根的树可能对应的输入数组，可以假设知道其左子树和右子树的可能输入数组，来找递归规律
- 对于给定的长度分别为a和b的某一左和右子树输入数组，将其融合构建以root为根的树，发现，只要保证各子树内部数字顺序相同即可，例如2中，只要1在2之前，且4在5之前，就能构建符合要求的树。因此，这两个输入数组的混合方式有 `A(a+b,a+b)/(A(a,a)*A(b,b))` 种
- 因此，若左右子树分别有c，d种输入数组，则可以构成以root为根的输入数组数目为 `c*d*A(a+b,a+b)/(A(a,a)*A(b,b))`

因此衍生出两个问题：

- 二叉搜索树的生成和遍历：简单
- 阶乘的求解，且答案可能会很大，且取模公式含除法
  - 阶乘的求解按记忆化递归即可，每做一次乘法都要取模
  - 含除法的取模公式：`(a/b)*mod = a*pow(b,mod-1)%mod`
    - 衍生出快速幂的求解
  - 另外涉及到大数的，直接用unsigned long long代替int，防止溢出

最后记得减一，即减去题目里给出的一个输入数组

```
TreeNode* insertIntoBST(TreeNode* root, int val) {//向二叉搜索树插入
    if(!root) return new TreeNode(val);
    TreeNode* cur = root;
    while(1){
        if(cur->val < val){
            if((cur->right)) cur = cur->right;
            else{
                cur->right = new TreeNode(val);
                return root;
            }
        }
        else{
            if(cur->left) cur = cur->left;
            else{
                cur->left = new TreeNode(val);
                return root;
            }
        }
    }
    return NULL;
}
class Solution {
public:
    unsigned long long MOD = 1e9+7;
    vector<unsigned long long> jc;//阶乘结果
    unsigned long long get_jc(unsigned long long i){//求阶乘
        if(jc[i] > 0)
            return jc[i];
        unsigned long long ans = get_jc(i - 1) * i % MOD;
        jc[i] = ans;
        return ans;
    }

    unsigned long long fastPow(unsigned long long x, unsigned long long n){//快速幂
        if(x == 1)
            return x;
        if(n == 0)
            return 1;
        unsigned long long ans = fastPow(x, n/2);
        if(n % 2 == 0)
            return (ans * ans) % MOD;//注意%优先级高于*
        else
            return ((ans * ans) % MOD * x) % MOD;//注意%优先级高于*
    }

    unsigned long long  cal(unsigned long long a, unsigned long long b){//求
A(a+b,a+b)/(A(a,a)*A(b,b))，即给定左右子树长度分别为a,b的输入数组，将他们混合在一起的方案，要
求混合后对应二叉树不变
        unsigned long long m = get_jc((a + b) % MOD);
        unsigned long long n = (get_jc(a) * get_jc(b)) % MOD;
        unsigned long long ans =  (m * (fastPow(n, MOD-2) % MOD)) % MOD;
        return ans;
    }
```

```
    TreeNode* build(vector<int>& nums){//建立二叉搜索树
        TreeNode* root = new TreeNode(nums[0]);
        for(int i = 1; i < nums.size(); i++)
            root = insertIntoBST(root, nums[i]);
        return root;
    }

    vector<unsigned long long> helper(TreeNode* root){//求以root为根的树 <对应的输入数组可能有多少种，树中的节点数>
        if(!root) return {0, 0};
        if(!root->left && !root->right) return {1, 1};
        vector<unsigned long long> l = helper(root->left);
        vector<unsigned long long> r = helper(root->right);
        if(l[0] == 0) return {r[0], r[1]+1};
        if(r[0] == 0) return {l[0], l[1]+1};
        unsigned long long nums = (cal(l[1], r[1]) * ((l[0] * r[0]) % MOD)) % MOD;
        return {nums, l[1]+r[1]+1};
    }

    int numOfWays(vector<int>& nums) {
        jc = vector<unsigned long long>(1001, 0);
        jc[1] = 1;
        TreeNode* root = build(nums);
        int ans = helper(root)[0];
        return ans - 1;
    }
};
```

赛后看了题解明悟，其实树是不用真的建立出来的，我在前面也推导=出来了，只要相对顺序保持不变即可保证二叉搜索树不变，因此可以每次以第一个数为根，划分左右子树的

这里羡慕一下python，同样的思路，不用考虑这么多

```
class Solution:
    def helper(self, nums: List[int]) -> int:
        if not nums:
            return 1

        pivot = nums[0]
        nums_left = [num for num in nums if num < pivot]
        nums_right = [num for num in nums if num > pivot]
        return self.combine(len(nums) - 1, len(nums_left)) * self.helper(
            nums_left) * self.helper(nums_right) % (10 ** 9 + 7)

    def combine(self, n, m):
        return math.factorial(n) // (
                math.factorial(m) * math.factorial(n - m))

    def numOfWays(self, nums: List[int]) -> int:
        return self.helper(nums) - 1
```