

1221. 分割平衡字符串（简单）

1. 题目描述

在一个「平衡字符串」中，'L' 和 'R' 字符的数量是相同的。

给出一个平衡字符串 `s`，请你将它分割成尽可能多的平衡字符串。返回可以通过分割得到的平衡字符串的最大数量。

示例 1:

输入: `s = "RLRLLRLRL"`

输出: 4

解释: `s` 可以分割为 `"RL"`, `"RRLL"`, `"RL"`, `"RL"`, 每个子字符串中都包含相同数量的 'L' 和 'R'。

示例 2:

输入: `s = "RLLLLRRRLR"`

输出: 3

解释: `s` 可以分割为 `"RL"`, `"LLLR"`, `"RRRL"`, 每个子字符串中都包含相同数量的 'L' 和 'R'。

示例 3:

输入: `s = "LLLLRRRR"`

输出: 1

解释: `s` 只能保持原样 `"LLLLRRRR"`。

提示:

- `1 <= s.length <= 1000`
- `s[i] = 'L' 或 'R'`

2. 简单实现

```
class Solution {
public:
    int balancedStringSplit(string s) {
        int ans = 0;
        int l = 0, r = 0;
        for(int i = 0; i < s.size(); i++){
            if(s[i] == 'L')
                l++;
            else
                r++;
            if(l == r){
                ans++;
                l = r = 0;
            }
        }
        return ans;
    }
};
```

```
}  
};
```

1222. 可以攻击国王的皇后（中等）

1. 题目描述

在一个 **8x8** 的棋盘上，放置着若干「黑皇后」和一个「白国王」。

「黑皇后」在棋盘上的位置分布用整数坐标数组 `queens` 表示，「白国王」的坐标用数组 `king` 表示。

「黑皇后」的行棋规定是：横、直、斜都可以走，步数不受限制，但是，不能越子行棋。

请你返回可以直接攻击到「白国王」的所有「黑皇后」的坐标（任意顺序）。

示例 1：

K	Q			Q			
Q							
				Q			
			Q				
Q							

输入：queens = [[0,1],[1,0],[4,0],[0,4],[3,3],[2,4]], king = [0,0]

输出：[[0,1],[1,0],[3,3]]

解释：

[0,1] 的皇后可以攻击到国王，因为他们在同一行上。

[1,0] 的皇后可以攻击到国王，因为他们在同一列上。

[3,3] 的皇后可以攻击到国王，因为他们在同一条对角线上。

[0,4] 的皇后无法攻击到国王，因为她被位于 [0,1] 的皇后挡住了。

[4,0] 的皇后无法攻击到国王，因为她被位于 [1,0] 的皇后挡住了。

[2,4] 的皇后无法攻击到国王，因为她和国王不在同一行/列/对角线上。

示例 2：

Q							
	Q						
		Q					
			K	Q	Q		
				Q	Q		

输入: queens = [[0,0],[1,1],[2,2],[3,4],[3,5],[4,4],[4,5]], king = [3,3]
输出: [[2,2],[3,4],[4,4]]

示例 3:

Q	Q		Q	Q	Q	Q	Q
Q	Q	Q		Q		Q	Q
	Q	Q	Q			Q	Q
				K			Q
Q		Q	Q		Q	Q	
Q	Q	Q		Q		Q	
	Q		Q	Q			
					Q		Q

输入: queens = [[5,6],[7,7],[2,1],[0,7],[1,6],[5,1],[3,7],[0,3],[4,0],[1,2],[6,3],[5,0],[0,4],[2,2],[1,1],[6,4],[5,4],[0,0],[2,6],[4,5],[5,2],[1,4],[7,5],[2,3],[0,5],[4,2],[1,0],[2,7],[0,1],[4,6],[6,1],[0,6],[4,3],[1,7]], king = [3,4]
输出: [[2,3],[1,4],[1,6],[3,7],[4,3],[5,4],[4,5]]

提示:

- `1 <= queens.length <= 63`
- `queens[0].length == 2`
- `0 <= queens[i][j] < 8`

- `king.length == 2`
- `0 <= king[0], king[1] < 8`
- 一个棋盘格上最多只能放置一枚棋子。

2. 简单实现

将 `8*8` 的格子扩展成 `10*10`，将坐标化为数字，向八个方向探查

也可以直接用坐标来做

```
class Solution {
public:
    vector<vector<int>> queensAttacktheKing(vector<vector<int>>& queens,
vector<int>& king) {
        vector<vector<int>> ans;
        vector<bool> q = vector<bool>(100, false);
        for(int i = 0; i < queens.size(); i++)
            q[queens[i][0]*10 + queens[i][1] + 11] = true;
        int k = king[0]*10 + king[1] + 11;

        vector<int> direction = {-11, -10, -9, -1, 1, 9, 10, 11}; //八个方向
        int cur;
        for(int i = 0; i < direction.size(); i++) //遍历各个方向
            for(int idx = 1; ; idx++){ //向该方向走idx步
                cur = k + idx * direction[i]; //走后的位置
                if(cur/10 == 0 || cur/10 == 9 || cur%10 == 0 || cur%10 == 9)
                    break; //到达边界，说明该方向没有黑皇后
                if(q[cur] == true){ //找到最近的可攻击的黑皇后
                    ans.push_back({cur/10 - 1, cur%10 - 1});
                    break;
                }
            }
        return ans;
    }
};
```

1223. 掷骰子模拟（中等）

1. 题目描述

有一个骰子模拟器会每次投掷的时候生成一个 1 到 6 的随机数。

不过我们在使用它时有个约束，就是使得投掷骰子时，**连续** 掷出数字 `i` 的次数不能超过 `rollMax[i]` (`i` 从 1 开始编号)。

现在，给你一个整数数组 `rollMax` 和一个整数 `n`，请你来计算掷 `n` 次骰子可得到的不同点数序列的数量。

假如两个序列中至少存在一个元素不同，就认为这两个序列是不同的。由于答案可能很大，所以请返回 模 $10^9 + 7$ 之后的结果。

示例 1:

输入: $n = 2$, $\text{rollMax} = [1,1,2,2,2,3]$

输出: 34

解释: 我们掷 2 次骰子, 如果没有约束的话, 共有 $6 * 6 = 36$ 种可能的组合。但是根据 rollMax 数组, 数字 1 和 2 最多连续出现一次, 所以不会出现序列 (1,1) 和 (2,2)。因此, 最终答案是 $36 - 2 = 34$ 。

示例 2:

输入: $n = 2$, $\text{rollMax} = [1,1,1,1,1,1]$

输出: 30

示例 3:

输入: $n = 3$, $\text{rollMax} = [1,1,1,2,2,3]$

输出: 181

提示:

- $1 \leq n \leq 5000$
- $\text{rollMax.length} == 6$
- $1 \leq \text{rollMax}[i] \leq 15$

2. 正确解法——动态规划

1, $\text{dp}[i][j]$ 代表截止到某一次掷骰子, 掷出的数字为 i , 且截止到目前为止一共出现了连续 j 个 i 的情况数

2, 状态转移方程

$\text{dp}[i][j] = \text{dp}[i][j - 1] \quad (j \geq 2)$

$\text{dp}[i][1] = \sum \{ \text{dp}[u][v] \mid u \neq i; 1 \leq v < \text{rollMax}[k] \}$

```
class Solution {
public:
    const long M = 1e9 + 7;
    int dieSimulator(int n, vector<int>& rollMax) {
        long dp1[6][16] = {0}; // 记录上一次投掷的情况
        long dp2[6][16] = {0}; // 记录本此投掷的情况
        long sums1[6] = {0}; // 记录上一次投掷中以 i 结尾的总情况数
        long sums2[6] = {0};
        long tot1 = 0; // 记录上一次投掷的总情况数
        long tot2 = 0;
        for (int i = 0; i < 6; ++i) { // 初始化第一次投掷
            dp1[i][1] = 1;
            sums1[i] = 1;
            ++tot1;
        }
        for (--n; n > 0; --n) { // 再投掷 n-1 次
            tot2 = 0;
            for (int i = 0; i < 6; ++i) {
                sums2[i] = 0;
                for (int j = 2; j <= rollMax[i]; ++j) {
                    dp2[i][j] = dp1[i][j - 1];
                    sums2[i] += dp2[i][j];
                }
            }
        }
    }
};
```

```

        dp2[i][1] = (M + tot1 - sums1[i]) % M; //由于取模后tot1可能小于
sums1[i], 故加M
        sums2[i] += dp2[i][1];
        sums2[i] %= M;
        tot2 += sums2[i];
    }
    tot2 %= M;
    swap(dp1, dp2);
    swap(sums1, sums2);
    swap(tot1, tot2);
}
return tot1;
}
};

```

1224. 最大相等频率 (困难)

1. 题目描述

给出一个正整数数组 `nums`，请你帮忙从该数组中找出能满足下面要求的 **最长** 前缀，并返回其长度：

- 从前缀中 **删除一个** 元素后，使得所剩下的每个数字的出现次数相同。

如果删除这个元素后没有剩余元素存在，仍可认为每个数字都具有相同的出现次数（也就是 0 次）。

示例 1：

输入：nums = [2,2,1,1,5,3,3,5]

输出：7

解释：对于长度为 7 的子数组 [2,2,1,1,5,3,3]，如果我们从中删去 nums[4]=5，就可以得到 [2,2,1,1,3,3]，里面每个数字都出现了两次。

示例 2：

输入：nums = [1,1,1,2,2,2,3,3,3,4,4,4,5]

输出：13

示例 3：

输入：nums = [1,1,1,2,2,2]

输出：5

示例 4：

输入：nums = [10,2,8,9,3,8,1,5,2,3,7,6]

输出：8

提示：

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

2. 简单实现

因为要取最长前缀，故可以从后往前依次去掉各个数字，查看当前所剩前缀是否符合条件

```
class Solution {
public:
    int maxEqualFreq(vector<int>& nums) {
        if(nums.size() == 1) return 1;
        unordered_map<int, int> num2cnt; //记录当前前缀中各个数字的次数
        unordered_map<int, int> cnt2num; //记录当前前缀中出现次数为cnt的数字有几个
        for(int i = 0; i < nums.size(); i++){ //初始将整个数组做为前缀
            if(num2cnt.count(nums[i]) <= 0)
                num2cnt[nums[i]] = 1;
            else
                num2cnt[nums[i]]++;
        }
        for(auto it = num2cnt.begin(); it != num2cnt.end(); it++){
            if(cnt2num.count(it->second) <= 0)
                cnt2num[it->second] = 1;
            else
                cnt2num[it->second]++;
        }
        for(int i = nums.size()-1; i >= 0; i--){ //从后往前依次去掉各个数字
            if(cnt2num.size() == 2){
                auto it1 = cnt2num.begin();
                auto it2 = cnt2num.begin();
                it2++;
                if(it1->first == 1 && it1->second == 1) //eg.{1:1, 3:4}
                    return i+1;
                if(it2->first == 1 && it2->second == 1) //同上
                    return i+1;
                if((it1->first - it2->first == 1 && it1->second == 1) //eg.{3:1,
2:2}
                    || (it2->first - it1->first == 1 && it2->second == 1)) //同上
                    return i+1;
            }
            else if(cnt2num.size() == 1){
                auto it = cnt2num.begin();
                if(it->first == 1 //eg.{1:3}
                    || it->second == 1) //eg.{4:1}
                    return i+1;
            }
            int cnt = num2cnt[nums[i]]--; //从前缀中去掉nums[i]
            //更新cnt2num
            if(cnt2num[cnt] == 1)
                cnt2num.erase(cnt);
            else
                cnt2num[cnt]--;
            if(cnt > 1){
                if(cnt2num.count(cnt-1) <= 0)
                    cnt2num[cnt-1] = 1;
                else
                    cnt2num[cnt-1]++;
            }
        }
    }
}
```

```
    }  
    return -1;  
}  
};
```