LeetCode 1403. 非递增顺序的最小子序列

1403. 非递增顺序的最小子序列

难度 简单 7 收藏 分享 切换为英文 关注 反馈

给你一个数组 `nums`，请你从中抽取一个子序列，满足该子序列的元素之和 **严格** 大于未包含在该子序列中的各元素之和。

如果存在多个解决方案，只需返回 **长度最小** 的子序列。如果仍然有多个解决方案，则返回 **元素之和最大** 的子序列。

与子数组不同的地方在于，「数组的子序列」不强调元素在原数组中的连续性，也就是说，它可以通过从数组中分离一些（也可能不分离）元素得到。

注意，题目数据保证满足所有约束条件的解决方案是 **唯一** 的。同时，返回的答案应当按 **非递增顺序** 排列。

示例 1：

```
输入：nums = [4,3,10,9,8]
输出：[10,9]
解释：子序列 [10,9] 和 [10,8] 是最小的、满足元素之和大于其他各元素之和的子序列。但是 [10,9] 的元素之和最大。
```

示例 2：

```
输入：nums = [4,4,7,6,7]
输出：[7,7,6]
解释：子序列 [7,7] 的和为 14 ，不严格大于剩下的其他元素之和（14 = 4 + 4 + 6）。因此，[7,6,7] 是满足题意的最小子序列。注意，元素按非递增顺序返回。
```

示例 3：

```
输入：nums = [6]
输出：[6]
```

```cpp
class Solution {
public:
    vector<int> minSubsequence(vector<int>& nums) {
        int n = nums.size();
        int sum = 0;

        for(int i = 0; i < n;i ++)
            sum += nums[i];

        sort(nums.begin(),nums.end());
        reverse(nums.begin(),nums.end());

        int s = 0;
        vector<int> ans;
        for(int i = 0;i < n;i ++)
        {
```

```
            s += nums[i];
            ans.push_back(nums[i]);
            if(s + s > sum)
                return ans;
        }
        return ans;
    }
};
```

LeetCode 1404. 将二进制表示减到 1 的步骤数

1404. 将二进制表示减到 1 的步骤数

难度 中等 8 收藏 分享 切换为英文 关注 反馈

给你一个以二进制形式表示的数字 `s` 。请你返回按下述规则将其减少到 1 所需要的步骤数：

- 如果当前数字为偶数，则将其除以 2 。
- 如果当前数字为奇数，则将其加上 1 。

题目保证你总是可以按上述规则将测试用例变为 1 。

示例 1：

```
输入：s = "1101"
输出：6
解释："1101" 表示十进制数 13 。
Step 1) 13 是奇数，加 1 得到 14
Step 2) 14 是偶数，除 2 得到 7
Step 3) 7  是奇数，加 1 得到 8
Step 4) 8  是偶数，除 2 得到 4
Step 5) 4  是偶数，除 2 得到 2
Step 6) 2  是偶数，除 2 得到 1
```

示例 2：

```
输入：s = "10"
输出：1
解释："10" 表示十进制数 2 。
Step 1) 2 是偶数，除 2 得到 1
```

示例 3：

```
输入：s = "1"
输出：0
```

```
class Solution {
public:
    int numSteps(string s) {
        int ans = 0;
        reverse(s.begin(), s.end());

        while (s.length() > 1) {
            ans++;
```

```
            if (s[0] == '0') {
                for (int i = 0; i < s.length() - 1; i++)
                    s[i] = s[i + 1];
                s.pop_back();
            } else {
                s[0]++;
                int i = 0;
                while (i < s.length() - 1 && s[i] == '2') {
                    s[i] = '0';
                    s[i + 1]++;
                    i++;
                }

                if (s.back() == '2') {
                    s.back() = '0';
                    s += '1';
                }
            }
        }

        return ans;
    }
};
```

1405. 最长快乐字符串

1405. 最长快乐字符串

难度 中等 19 收藏 分享 切换为英文 关注 反馈

如果字符串中不含有任何 'aaa'，'bbb' 或 'ccc' 这样的字符串作为子串，那么该字符串就是一个「快乐字符串」。

给你三个整数 a，b ，c，请你返回 **任意一个** 满足下列全部条件的字符串 s：

- s 是一个尽可能长的快乐字符串。
- s 中 **最多** 有 a 个字母 'a'、b 个字母 'b'、c 个字母 'c' 。
- s 中只含有 'a'、'b' 、'c' 三种字母。

如果不存在这样的字符串 s ，请返回一个空字符串 ""。

示例 1：

```
输入：a = 1, b = 1, c = 7
输出："ccaccbcc"
解释："ccbccacc" 也是一种正确答案。
```

示例 2：

```
输入：a = 2, b = 2, c = 1
输出："aabbc"
```

示例 3：

```
输入：a = 7, b = 1, c = 0
输出："aabaa"
解释：这是该测试用例的唯一正确答案。
```

```cpp
class Solution {
public:
    void insert(string &ans, char c) {
        for (int j = 0; j <= ans.length(); j++) {
            if (!( j > 1 && ans[j - 2] == c && ans[j - 1] == c
                || j > 0 && ans[j - 1] == c && ans[j] == c
                || j < ans.length() - 1 && ans[j] == c && ans[j + 1] == c)) {
                ans.insert(j, 1, c);
                break;
            }
        }
    }

    string longestDiverseString(int a, int b, int c) {
        string ans;

        int x = a + b + c, r = 0;
        for (int i = 1; i <= x; i++) {
            if (r == 0) {
                if (a > 0) {
                    insert(ans, 'a');
                    a--;
                } else {
```

```
                    i--;
                }
            }

            if (r == 1) {
                if (b > 0) {
                    insert(ans, 'b');
                    b--;
                } else {
                    i--;
                }
            }

            if (r == 2) {
                if (c > 0) {
                    insert(ans, 'c');
                    c--;
                }
                else {
                    i--;
                }
            }

            r = (r + 1) % 3;
        }

        return ans;
    }
};
```

LeetCode 1406. 石子游戏 III

1406. 石子游戏 III

难度 困难 33 收藏 分享 切换为英文 关注 反馈

Alice 和 Bob 用几堆石子在做游戏。几堆石子排成一行，每堆石子都对应一个得分，由数组 `stoneValue` 给出。

Alice 和 Bob 轮流取石子，**Alice** 总是先开始。在每个玩家的回合中，该玩家可以拿走剩下石子中的的前 **1、2 或 3 堆石子** 。比赛一直持续到所有石头都被拿走。

每个玩家的最终得分为他所拿到的每堆石子的对应得分之和。每个玩家的初始分数都是 **0** 。比赛的目标是决出最高分，得分最高的选手将会赢得比赛，比赛也可能会出现平局。

假设 Alice 和 Bob 都采取 **最优策略** 。如果 Alice 赢了就返回 "Alice" ，Bob 赢了就返回 "Bob"，平局（分数相同）返回 "Tie" 。

示例 1：

```
输入：values = [1,2,3,7]
输出："Bob"
解释：Alice 总是会输，她的最佳选择是拿走前三堆，得分变成 6 。但是 Bob 的得分为 7，
Bob 获胜。
```

示例 2：

```
输入：values = [1,2,3,-9]
输出："Alice"
解释：Alice 要想获胜就必须在第一个回合拿走前三堆石子，给 Bob 留下负分。
如果 Alice 只拿走第一堆，那么她的得分为 1，接下来 Bob 拿走第二、三堆，得分为 5 。
之后 Alice 只能拿到分数 -9 的石子堆，输掉比赛。
如果 Alice 拿走前两堆，那么她的得分为 3，接下来 Bob 拿走第三堆，得分为 3 。之后
Alice 只能拿到分数 -9 的石子堆，同样会输掉比赛。
注意，他们都应该采取 最优策略 ，所以在这里 Alice 将选择能够使她获胜的方案。
```

```cpp
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue) {
        int n = stoneValue.size();
        vector<int> f(n + 1);

        f[n] = 0;

        for (int i = n - 1; i >= 0; i--) {
            f[i] = stoneValue[i] - f[i + 1];
            if (i < n - 1)
                f[i] = max(f[i], stoneValue[i] + stoneValue[i + 1] - f[i + 2]);

            if (i < n - 2)
                f[i] = max(f[i], stoneValue[i] + stoneValue[i + 1] +
stoneValue[i + 2]
                                    - f[i + 3]);
        }

        if (f[0] == 0)
            return "Tie";
        else if (f[0] > 0)
```

```
            return "Alice";
        else return "Bob";
    }
};
```