LeetCode 1422. 分割字符串的最大得分

1422. 分割字符串的最大得分

难度 简单 4 收藏 分享 切换为英文 关注 反馈

给你一个由若干 0 和 1 组成的字符串 `s` ，请你计算并返回将该字符串分割成两个 **非空** 子字符串（即 **左** 子字符串和 **右** 子字符串）所能获得的最大得分。

「分割字符串的得分」为 **左** 子字符串中 **0** 的数量加上 **右** 子字符串中 **1** 的数量。

示例 1：

```
输入：s = "011101"
输出：5
解释：
将字符串 s 划分为两个非空子字符串的可行方案有：
左子字符串 = "0" 且 右子字符串 = "11101"，得分 = 1 + 4 = 5
左子字符串 = "01" 且 右子字符串 = "1101"，得分 = 1 + 3 = 4
左子字符串 = "011" 且 右子字符串 = "101"，得分 = 1 + 2 = 3
左子字符串 = "0111" 且 右子字符串 = "01"，得分 = 1 + 1 = 2
左子字符串 = "01110" 且 右子字符串 = "1"，得分 = 2 + 1 = 3
```

示例 2：

```
输入：s = "00111"
输出：5
解释：当 左子字符串 = "00" 且 右子字符串 = "111" 时，我们得到最大得分 = 2 + 3 = 5
```

示例 3：

```
输入：s = "1111"
输出：3
```

```cpp
class Solution {
public:
    int maxScore(string s) {
        int n = s.length();
        vector<int> z(n, 0), o(n, 0);
        if (s.front() == '0')
            z[0] = 1;

        for (int i = 1; i < n; i++)
            z[i] = z[i - 1] + int(s[i] == '0');

        if (s.back() == '1')
            o[n - 1] = 1;

        for (int i = n - 2; i >= 0; i--)
            o[i] = o[i + 1] + int(s[i] == '1');

        int ans = 0;
```

```
        for (int i = 1; i < n; i++)
            ans = max(ans, z[i - 1] + o[i]);

        return ans;
    }
};
```

1423. 可获得的最大点数

1423. 可获得的最大点数

难度 中等 8 收藏 分享 切换为英文 关注 反馈

几张卡牌 **排成一行**，每张卡牌都有一个对应的点数。点数由整数数组 `cardPoints` 给出。

每次行动，你可以从行的开头或者末尾拿一张卡牌，最终你必须正好拿 `k` 张卡牌。

你的点数就是你拿到手中的所有卡牌的点数之和。

给你一个整数数组 `cardPoints` 和整数 `k`，请你返回可以获得的最大点数。

示例 1：

输入： `cardPoints = [1,2,3,4,5,6,1], k = 3`
输出： `12`
解释： 第一次行动，不管拿哪张牌，你的点数总是 `1` 。但是，先拿最右边的卡牌将会最大化你的可获得点数。最优策略是拿右边的三张牌，最终点数为 `1 + 6 + 5 = 12` 。

示例 2：

输入： `cardPoints = [2,2,2], k = 2`
输出： `4`
解释： 无论你拿起哪两张卡牌，可获得的点数总是 `4` 。

示例 3：

输入： `cardPoints = [9,7,7,9,7,7,9], k = 7`
输出： `55`
解释： 你必须拿起所有卡牌，可以获得的点数为所有卡牌的点数之和。

示例 4：

输入： `cardPoints = [1,1000,1], k = 1`
输出： `1`
解释： 你无法拿到中间那张卡牌，所以可以获得的最大点数为 `1` 。

```
class Solution {
public:
    int maxScore(vector<int>& cardPoints, int k) {
        int n = cardPoints.size();

        vector<int> s(n + 1, 0);

        for (int i = 1; i <= n; i++)
            s[i] = s[i - 1] + cardPoints[i - 1];
```

```
        int ans = 0;
        for (int i = 0; i <= k; i++)
            ans = max(ans, s[i] + s[n] - s[n - (k - i)]);

        return ans;
    }
};
```

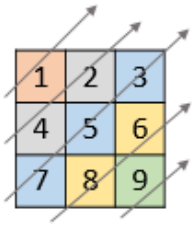
1424. 对角线遍历 II

1424. 对角线遍历 II

难度 中等 8 收藏 分享 切换为英文 关注 反馈

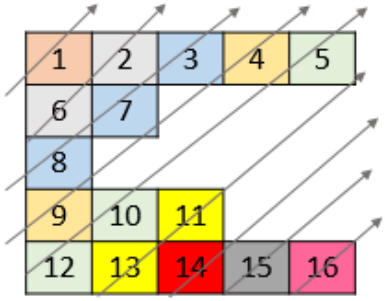
给你一个列表 `nums` ，里面每一个元素都是一个整数列表。请你依照下面各图的规则，按顺序返回 `nums` 中对角线上的整数。

示例 1：



输入： `nums = [[1,2,3],[4,5,6],[7,8,9]]`
输出： `[1,4,2,7,5,3,8,6,9]`

示例 2：



输入： `nums = [[1,2,3,4,5],[6,7],[8],[9,10,11],[12,13,14,15,16]]`
输出： `[1,6,2,8,7,3,9,4,12,10,5,13,11,14,15,16]`

```
class Solution {
public:
    vector<int> findDiagonalOrder(vector<vector<int>>& nums) {
        int n = nums.size();

        vector<int> pre(n + 1), nxt(n + 1);

        int s = 0;
```

```
        for (int i = 0; i <= n; i++) {
            pre[i] = i - 1;
            nxt[i] = i + 1;
        }

        int d = 0;
        vector<int> ans;

        while (s < n) {
            vector<int> cur;
            for (int i = s; i < min(d + 1, n); i = nxt[i]) {
                if (d - i < nums[i].size()) {
                    cur.push_back(nums[i][d - i]);
                    if (d - i == nums[i].size() - 1) {
                        if (pre[i] == -1) s = nxt[i];
                        else nxt[pre[i]] = nxt[i];

                        pre[nxt[i]] = pre[i];
                    }
                }
            }

            ans.insert(ans.end(), cur.rbegin(), cur.rend());
            d++;
        }

        return ans;
    }
};
```

1425. 带限制的子序列和

1425. 带限制的子序列和

难度 困难 23 收藏 分享 切换为英文 关注 反馈

给你一个整数数组 `nums` 和一个整数 `k` ，请你返回 **非空** 子序列元素和的最大值，子序列需要满足：子序列中每两个 **相邻** 的整数 `nums[i]` 和 `nums[j]` ，它们在原数组中的下标 `i` 和 `j` 满足 `i < j` 且 `j - i <= k` 。

数组的子序列定义为：将数组中的若干个数字删除（可以删除 0 个数字），剩下的数字按照原本的顺序排布。

示例 1：

```
输入：nums = [10,2,-10,5,20], k = 2
输出：37
解释：子序列为 [10, 2, 5, 20] 。
```

示例 2：

```
输入：nums = [-1,-2,-3], k = 1
输出：-1
解释：子序列必须是非空的，所以我们选择最大的数字。
```

示例 3：

```
输入：nums = [10,-2,-10,-5,20], k = 2
输出：23
解释：子序列为 [10, -2, -5, 20] 。
```

```cpp
class Solution {
public:
    int constrainedSubsetSum(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> f(n);

        deque<int> q;
        int ans = nums[0];

        f[0] = nums[0];
        q.push_back(0);

        for (int i = 1; i < n; i++) {
            while (!q.empty() && i - q.front() > k)
                q.pop_front();

            f[i] = max(nums[i], f[q.front()] + nums[i]);
            ans = max(ans, f[i]);

            while (!q.empty() && f[i] >= f[q.back()])
                q.pop_back();

            q.push_back(i);
        }
        return ans;
    }
```

```
};
```