算法设计与应用基础 作业3

19335174 施天予

1.【Leetcode455】分发饼干

算法思路

从胃口最小的孩子开始尽量满足，用小的饼干尝试满足，逐个遍历，直到胃口最大的孩子被满足或者饼干没有了。这样可以满足的孩子最多。
将胃口 g 数组和饼干大小数组 s ，进行从小到大的排序。
从小到大开始遍历两个数组：
如果 s[j]>=g[i] ，则可以满足该孩子的胃口， ans++ ， i++ , j++ 。
否则， j++ ，看下一个饼干的大小能否满足该孩子的胃口。
当 i 或 j 到达对应数组末尾时，结束循环，返回结果ans。

复杂度分析

时间复杂度：O(nlogn)，n是g和s数组长度更大的值。

空间复杂度：O(1)，开辟常数空间。

代码

```
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
        sort(g.begin(), g.end());
        sort(s.begin(), s.end());
        int i = 0, j = 0, ans = 0;
        while (i < g.size() && j < s.size()) {
            if (s[j] >= g[i]) {
                ans++;
                i++;
                j++;
            }
            else
                j++;
        }
        return ans;
    }
};
```

截图

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	32 ms	17 MB	C++	2021/06/04 12:03	添加备注

2.【Leetcode984】不含 AAA 或 BBB 的字符串

算法思路

我们可以尝试填充一个字符串，一共a+b个字符。设两个变量i和j，分别代表当前填充'a'和'b'的数量，是否有超过2。如果a>=b且当前'a'的个数小于2，或者已经有连续两个'b'，则可以填充a，反之同理。最后，就能得到想要的结果。

复杂度分析

时间复杂度：O(a+b)，共循环a+b次。

空间复杂度：O(1)，开辟常数空间。

代码

```
class Solution {
public:
    string strWithout3a3b(int a, int b) {
        int i = 0, j = 0;
        string ans = "";
        while (a > 0 || b > 0) {
            if (a >= b && i < 2 || j == 2) {
                i++;
                j = 0;
                ans += 'a';
                a--;
            }
            else {
                j++;
                i = 0;
                ans += 'b';
                b--;
            }
        }
        return ans;
    }
};
```

截图

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	0 ms	6.1 MB	C++	2021/06/04 12:09	添加备注

3.【Leetcode120】三角形最小路径和

算法思路

这道题的目标是找到三角形triangle自顶向下的最小路径和。因为最小路径和只有一种可能，而且三角形的顶只有一个元素，所以我们可以反过来思考，自底向上寻找最小路径的可能值，逐步比较，到三角形顶端我们就能找到最小路径和。

复杂度分析

时间复杂度：O(n^2)，n为三角形的行数。

空间复杂度：O(1)，开辟常数空间。

代码

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        int n = triangle.size();
        for (int i = n-2; i >= 0; --i)
            for (int j = 0; j <= i; ++j)
                triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1]);
        return triangle[0][0];
    }
};
```

截图

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	8 ms	8.2 MB	C++	2021/06/04 12:15	添加备注

4.【Leetcode714】买卖股票的最佳时机含手续费

算法思路

运用动态规划方法。定义dp0表示第i天交易完后手里没有股票的最大利润，dp1表示第i天交易完后手里持有一只股票的最大利润（i从0开始）。每一次买股票都要手续费，我们可以列出状态转移方程：

```
temp = dp0;
dp0 = max(dp0, dp1 + prices[i]);
dp1 = max(dp1, temp - prices[i] - fee);
```

初始时收益dp0 = 0，花费dp1 = -prices[0]-fee。

因此我们只要从前往后以此计算状态，全部交易结束时的dp0就是股票的最大收益。

复杂度分析

时间复杂度：O(n)，n为数组长度。

空间复杂度：O(1)，开辟常数空间。

代码

```
class Solution {
public:
    int maxProfit(vector<int>& prices, int fee) {
        int n = prices.size();
        int dp0 = 0;
        int dp1 = -prices[0]-fee;
        for (int i = 1; i < n; i++) {
            int temp = dp0;
            dp0 = max(dp0, dp1 + prices[i]);
            dp1 = max(dp1, temp - prices[i] - fee);
        }
        return dp0;
    }
};
```

截图

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	104 ms	53.8 MB	C++	2021/06/04 12:35	添加备注

5.【Leetcode91】解码游戏

算法思路

这道题也是运用动态规划。先初始化一个f数组， f[i] 记录 s[i] 位置的解码方法数。
状态转移：
当 s[i-1] 不是 “1” 或 “2” 的时候，说明 s[i] 不可以和 s[i-1] 联合解码
当 s[i] 是 “0”的时候， s[i] 无法解码，返回0
当 s[i] 不是“0”的时候， s[i] 位置独立解码，此时 f[i]=f[i-1]
当 s[i-1] 是“1”或“2”的时候，说明 s[i] 可以和 s[i-1] 联合解码
如果 s[i-1]s[i] 组成的数字小于等于26，则可以联合解码， f[i]=f[i-1]+f[i-2]
否则，还是不可以联合解码， s[i] 独立解码， f[i]=f[i-1]
最后 v[n-1] 的值就是我们要的结果。

复杂度分析

时间复杂度：O(n)，n为数组长度。

空间复杂度：O(n)，n为数组长度。

代码

```
class Solution {
public:
    int numDecodings(string s) {
        int n = s.size();
        if (s[0] == '0') return 0;
        if (n == 1) return 1;
        vector<int> f(n,0);
        f[0] = 1;
        if (s.substr(0, 2) <= "26")
            f[1] = s[1] == '0' ? 1 : 2;
        else
            f[1] = s[1] == '0' ? 0 : 1;
        for (int i = 2; i < n; ++i)
            if (s[i-1] == '1' || s[i-1] == '2')
                if (s[i] == '0')
                    f[i] = f[i-2];
                else
                    f[i] = s.substr(i-1, 2)<="26" ? f[i-1]+f[i-2] : f[i-1];
```

```
            else
                if (s[i] == '0')
                    return 0;
                else
                    f[i] = f[i-1];
        return f[n-1];
    }
};
```

截图

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	0 ms	6.1 MB	C++	2021/06/04 13:28	添加备注