深信服24届笔试题

Written By CTENET In 2023-09-07

Contact me at www.github.com/EASYGOING45 Or www.happygoing.cc

填空

涉及思维逻辑问题、二叉树前中后序遍历序列反推、链表、哈希表适用场景选取等，

编程题

一共四道

1、最长连续递减序列-ACM模式

> 给定一个未经排序的整数数组，找到最长且连续递减的子序列，并返回该序列的长度。
>
> 连续递减的子序列可以由两个下标l和r确定(l<r)，如果对于每个l<=i<r,都有nums[i]<nums[i+1]，那么子序列[nums[l],nums[l+1],...nums[r-1],nums[r]]就是连续递减子序列，
>
> 输入一个未经排序的整数数组
>
> 输出连续递减子序列的长度
>
> 例如:
> 输入 7 4 5 3 1
> 输出 3

```
/*
解决一道算法题：
给定一个未经排序的整数数组，找到最长且连续递减的子序列，并返回该序列的长度。
```

```
连续递减的子序列可以由两个下标l和r确定(l<r)，如果对于每个l<=i<r，都有nums[i]<nums[i+1]，那么子序列[nums[l],nums[l+1],...nums[r-1],nums[r]]就是连续递减子序列，
输入一个未经排序的整数数组
输出连续递减子序列的长度
例如：
输入 7 4 5 3 1
输出 3
*/
#include <iostream>
#include <vector>
using namespace std;

int longestDecreasingSubsequence(vector<int>& nums) {
    int n = nums.size();
    if (n == 0) {
        return 0;
    }

    int maxLength = 1;
    int currentLength = 1;

    for (int i = 1; i < n; i++) {
        if (nums[i] < nums[i-1]) {
            currentLength++;
        } else {
            maxLength = max(maxLength, currentLength);
            currentLength = 1;
        }
    }

    maxLength = max(maxLength, currentLength);

    return maxLength;
}

int main() {
    vector<int> nums = {7, 4, 5, 3, 1};
    int length = longestDecreasingSubsequence(nums);
```

```
    cout<<length<<endl;

     return 0;
}
```

代码的思路是使用一个变量maxLength来记录当前找到的最长子序列的长度，使用另一个变量currentLength来记录当前正在计算的子序列的长度。

首先，代码会检查数组的长度，如果数组为空，则直接返回0。

然后，代码从数组的第二个元素开始遍历，对于每个元素，如果它小于前一个元素，则说明当前子序列仍然是递减的，currentLength加1。如果当前元素不小于前一个元素，则说明当前子序列的递减趋势被打破，需要更新maxLength为当前子序列的长度，并将currentLength重置为1。

最后，代码会再次更新maxLength为当前子序列的长度，以确保最后一个子序列的长度也被考虑在内。

2、找朋友开黑-ACM模式

> *小张是一个游戏高手，想要在游戏中组建一个工会。小张有n个游戏好友，每个好友都愿意和他一起组工会，每个好友都有自己的游戏等级和与对小张的好感度，小张希望组建完工会后，所有人对自己的总好感度最大，如果在工会中有某个好友比另外的好友高至少d个等级，那么等级低的好友会感到自己的游戏水平菜，就不会对小张有好感度，请帮助他找到最佳的游戏队友*
>
> *输入描述：*
>
> *第一行输入包含两个整数n和d，分别是小张的朋友数量和感到游戏等级低的最小差异*
>
> *接下来n行是校长的朋友的信息，第(i+1)行包含第i个朋友的信息mi，si，分别是游戏等级和好感度*
>
> *输出好友对小张的总好感度*

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```cpp

using namespace std;

struct Friend {
    int level;
    int like;
};

bool compareFriends(const Friend& a, const Friend& b) {
    return a.level < b.level;
}

int main() {
    int n, d;
    cin >> n >> d;

    vector<Friend> friends(n);
    for (int i = 0; i < n; i++) {
        cin >> friends[i].level >> friends[i].like;
    }

    sort(friends.begin(), friends.end(), compareFriends);

    int maxLike = 0;
    int currentLike = 0;
    int j = 0;

    for (int i = 0; i < n; i++) {
        while (friends[i].level - friends[j].level >= d) {
            currentLike -= friends[j].like;
            j++;
        }
        currentLike += friends[i].like;
        maxLike = max(maxLike, currentLike);
    }

    cout << maxLike << endl;

```

```
    return 0;
}
```

这个问题可以通过贪心算法来解决。

首先，我们需要将朋友按照游戏等级进行排序，这样可以保证我们在遍历朋友列表时，等级低的朋友在等级高的朋友之前。

然后，我们使用两个指针来维护一个滑动窗口。初始时，两个指针都指向列表的第一个朋友。

我们通过移动右指针来扩展窗口，直到窗口中的朋友等级差异小于d。在扩展窗口的过程中，我们累加窗口中朋友的好感度。

一旦窗口中的朋友等级差异大于等于d，我们开始移动左指针来缩小窗口。在缩小窗口的过程中，我们减去窗口中最左边朋友的好感度。

在遍历过程中，我们记录下最大的好感度总和。

最后，输出最大的好感度总和即可。

3、所有奇数长度子数组的和-ACM模式

> *已知子数组定义为原数组中的一个连续子序列，现给定一个正整数数组arr，请计算该数组内所有可能的奇数长度子数组的数值之和*
>
> *输入一个正整数数组arr，输出所有可能的奇数长度子数组的和*

```cpp
/*
已知子数组定义为原数组中的一个连续子序列，现给定一个正整数数组arr，请计算该数组内所有可能的奇数长度子数组的数值之和
输入一个正整数数组arr，输出所有可能的奇数长度子数组的和
*/
#include <iostream>
#include <vector>
using namespace std;

int longestDecreasingSubsequence(vector<int>& nums) {
    int n = nums.size();
    if (n == 0) {
        return 0;
    }
```

```
    int maxLength = 1;
    int currentLength = 1;

    for (int i = 1; i < n; i++) {
        if (nums[i] < nums[i-1]) {
            currentLength++;
        } else {
            maxLength = max(maxLength, currentLength);
            currentLength = 1;
        }
    }

    maxLength = max(maxLength, currentLength);

    return maxLength;
}

int main() {
    std::vector<int> arr;
    int num;

    while (std::cin >> num) {
        arr.push_back(num);
        if (std::cin.get() == '\n') {
            break;
        }
    }
    int length = longestDecreasingSubsequence(arr);
    cout<<length<<endl;

    return 0;
}
```

这个问题要求计算给定正整数数组中所有可能的奇数长度子数组的数值之和。

首先，我们需要遍历数组中的每个元素，作为子数组的起始位置。

然后，我们使用两个指针来维护一个滑动窗口，初始时，左指针和右指针都指向起始位置。

我们通过移动右指针来扩展窗口，直到窗口的长度为奇数。在扩展窗口的过程中，我们累加窗口中元素的值。

一旦窗口的长度为奇数，我们开始移动左指针来缩小窗口。在缩小窗口的过程中，我们减去窗口中最左边元素的值。

在遍历过程中，我们记录下所有窗口中元素值的总和。

最后，输出所有窗口中元素值的总和即可。

4、分割回文字符串-核心代码模式

> *给你一个字符串string,string仅由小写英文字母组成，且长度不超过1000，请你讲string分割成一些子串，使每个 子串都是回文，返回符合要求的最少分割次数*

```
/*
解决一道算法题：
给你一个字符串string,string仅由小写英文字母组成，且长度不超过1000，请你讲string分割成一些子串，使每个 子串都是回文，返回符合要求的最少分割次数
*/
class Solution {
public:
    int min_cut_string(string str) {
        int n = str.length();

        // 创建一个二维DP表，用于存储从i到j的子串所需的最小分割次数
        vector<vector<int>> dp(n, vector<int>(n, 0));

        // 创建一个一维DP表，用于存储从0到i的子串所需的最小分割次数
        vector<int> cuts(n, 0);

        // 初始化DP表
        for (int i = 0; i < n; i++) {
            dp[i][i] = 1;
```

```
        }

        // 计算所有可能子串的最小分割次数
        for (int len = 2; len <= n; len++) {
            for (int i = 0; i < n - len + 1; i++) {
                int j = i + len - 1;

                if (str[i] == str[j] && (len == 2 || dp[i+1][j-1] == 1)) {
                    dp[i][j] = 1;
                }
            }
        }

        // 计算所需的最小分割次数
        for (int i = 0; i < n; i++) {
            int minCuts = i;

            for (int j = 0; j <= i; j++) {
                if (dp[j][i] == 1) {
                    minCuts = min(minCuts, (j == 0) ? 0 : cuts[j-1] + 1);
                }
            }

            cuts[i] = minCuts;
        }

        return cuts[n-1];
    }
};
```

算法的思路如下：

\1. 首先，创建一个二维的DP表，用于存储从字符串的第i个字符到第j个字符的子串所需的最小分割次数。同时，创建一个一维的cuts表，用于存储从字符串的第0个字符到第i个字符的子串所需的最小分割次数。

2. 初始化DP表，将对角线上的元素都设置为1，表示单个字符是回文。

\3. 接下来，计算所有可能子串的最小分割次数。使用两个循环，外层循环控制子串的长度，内层循环控制子串的起始位置。对于每个子串，判断首尾字符是否相等，并且如果子串长度大于2，则判断去掉首尾字符后的子串是否是回文。如果满足条件，则将DP表对应位置的值设置为1，表示该子串是回文。

\4. 最后，计算所需的最小分割次数。使用两个循环，外层循环遍历字符串的每个字符，内层循环从0遍历到外层循环的位置。对于每个位置，如果该位置到外层循环位置的子串是回文，则更新最小分割次数。最后，将最小分割次数存储到cuts表中。

\5. 返回cuts表的最后一个元素，即为符合要求的最少分割次数。

这个算法的时间复杂度为O(n^2)，其中n是字符串的长度。