C++方向编程题答案

第八周

day47

> **45830 合唱团**
>
> 链接：https://www.nowcoder.com/questionTerminal/661c49118ca241909add3a11c96408c8
>
> **【题目解析】：**
>
> 题目要求n各学生中选择k个，使这k个学生的能力值乘积最大。这是一个最优化的问题。另外，在优化过程中，提出了相邻两个学生的位置编号差不超过d的约束。
>
> 解决的方法是采用动态规划（理由：1.求解的是最优化问题；2.可以分解为最优子结构）
>
> **【解题思路】：**
>
> 对该问题的分解是关键。 从n个学生中，选择k个，可以看成是：先从n个学生里选择最后1个，然后在剩下的里选择k-1个，并且让这1个和前k-1个满足约束条件 记第k个人的位置为one,则可以用 `f[one][k]` 表示从n个人中选择k个的方案。然后，它的子问题，需要从one前面的left个人里面，选择k-1个，这里left表示k-1个人中最后一个（即第k-1个）人的位置，因此，子问题可以表示成 `f[left][k-1]` .
>
> 一般的动态规划题目，中间使用的表的最后一个元素， `dp[N][K]` 就是所求的结果。 但这个题目不能这样，因为如果那样建表，子问题就成了"在前n个学生中，取k个，使乘积最大" 然而，本题目有额外的限制条件"相邻两个学生的位置编号的差不超过d"就没有办法代入递推公式了， 因为子问题中本身并不包含位置信息。
>
> 从n个学生中，选择k个，可以看成是：先从n个学生里选择最后1个，然后在剩下的里选择k-1个，并且让这1个和前k-1个满足约束条件 记第k个人的位置为one,则可以用 `f[one][k]` 表示从n个人中选择k个的方案。然后，它的子问题，需要从one前面的left个人里面，选择k-1个，这里left表示k-1个人中最后一个（即第k-1个）人的位置，因此，子问题可以表示成 `f[left][k-1]` .
>
> 其次，求最大乘积比求最大和的问题要复杂许多。求最大和的话，子问题中也只需要求最大和就行了。 但求最大乘积的时候，在子问题中，每一步需要求最大正积和最小负积。 因为如果某学生的能力值为负数，乘以前面求得的最小负积，结果才是最大乘积。

```
#include <iostream>
#include <string>
#include <vector>
#include <limits.h>         // 宏 LONG_MIN LONG_MAX

using namespace std;

// NK表存放的是，在前n个学生中，取k个（必取第n个），所等得到的最大正乘积和最小负乘积。
long long NK_zheng[51][11];
long long NK_fu[51][11];

int main() {

 int N;

 cin >> N;
```

```
    vector<int> V(N + 1);
    for (int i = 1; i <= N; i++) {
        cin >> V[i];
    }

    int K, D;
    cin >> K >> D;

    // 对数组先初始化
    for (int n = 1; n <= 50; n++) {
        for (int k = 1; k <= 10; k++) {
            NK_zheng[n][k] = LLONG_MIN;      // 代表无效值
            NK_fu[n][k] = LLONG_MAX;
        }
    }

    for (int n = 1; n <= N; n++) {
        if (V[n] >= 0)
            NK_zheng[n][1] = V[n];
        else
            NK_fu[n][1] = V[n];
    }

    for (int n = 2; n <= N; n++) {
        for (int k = 2; k <= K && k <= n; k++) {
            // 找到NK[...][k-1]中最大的正数和最小的负数。
            long long max = LLONG_MIN, min = LLONG_MAX;
            for (int temp = (k - 1 > n - D) ? (k - 1) : (n - D); temp <= n - 1; temp++) {
// 起始位置很重要
                if (max < NK_zheng[temp][k - 1])
                    max = NK_zheng[temp][k - 1];
                if (min > NK_fu[temp][k - 1])
                    min = NK_fu[temp][k - 1];
            }
            if (max != LLONG_MIN) {
                if (V[n] >= 0)
                    NK_zheng[n][k] = max * V[n];
                else
                    NK_fu[n][k] = max * V[n];
            }
            if (min != LLONG_MAX) {
                if (V[n] < 0 && NK_zheng[n][k] < min * V[n])
                    NK_zheng[n][k] = min * V[n];
                else if (NK_fu[n][k] > min * V[n])
                    NK_fu[n][k] = min * V[n];
            }
        }
    }

    // 在两个表的最后一列中，找出最大的乘积。
    long long max1 = LLONG_MIN, max2 = LLONG_MIN;

    for (int n = K; n <= N; n++) {
```

```
        if (max1 < NK_zheng[n][K])
            max1 = NK_zheng[n][K];
        if (max2 < NK_fu[n][K] && NK_fu[n][K] != LLONG_MAX)
            max2 = NK_fu[n][K];
    }

    max1 = max1 > max2 ? max1 : max2;

    cout << max1 << endl;
}
```

36484 马戏团

链接: https://www.nowcoder.com/questionTerminal/c2afcd7353f84690bb73aa6123548770

【题目解析】：

注意! 体重相同时， 只有身高相同才能叠. 体重升序排列， 体重相同时，按身高降序排列 接下来就是按照身高数据进行最大升序子序列

【解题思路】：

参看 https://www.cnblogs.com/wxjor/p/5524447.html

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
struct player
{
 int w;
 int h;
};
bool com_w(player p1,player p2)
{
  //按照体重从小到大排序
  if(p1.w != p2.w)
  return p1.w <= p2.w;
  //在体重相等的条件下，身高高的在前（在上）
  else
  return p1.h>p2.h;
}
int main(void)
{
 int N;
 int h;
 int w;
 int index;
 vector<player> p;
 while(cin>>N)
 {
     p.clear();

     // 处理数据
```

```
    for(int i = 0;i<N;i++)
    {
       player pt;
       cin>>index>>w>>h;
       pt.w = w;
       pt.h = h;
       p.push_back(pt);
    }
    sort(p.begin(),p.end(),com_w);

    //按照身高求最大上升子序列(此处为核心代码)
    //关于最大上升子序列问题的讲解, 参看 https://www.cnblogs.com/wxjor/p/5524447.html
    int dp2[N+1];
    int max = 0;
    dp2[0] = 1;
    for(int i = 1;i<N;i++)
    {
        dp2[i] = 1;
        for(int j = 0;j<i;j++)
        {
            if(p[j].h <= p[i].h && dp2[j]+1 > dp2[i])
            dp2[i] = dp2[j] + 1;
        }
    }

    // 找出 dp2 中最大的数据, 即为最终结果
    for(int i = 0;i<N;i++)
           if(max < dp2[i])
           max = dp2[i];
    cout<<max<<endl;
 }
 system("pause");
 return 0;
}
```