Java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。

```java
import java.util.Scanner;

public class Main {
 public static void main(String[] args){
     Scanner sc = new Scanner(System.in);
     while(sc.hasNext()) {
         //总人数
         int n = sc.nextInt();
         //学生能力值数组，第i个人直接对应arr[i]
         int[] arr = new int[n + 1];
         //初始化
         for (int i = 1; i <= n; i++) {//人直接对应坐标
             arr[i] = sc.nextInt();
         }
         //选择的学生数
```

```
            int kk = sc.nextInt();
            //间距
            int dd = sc.nextInt();

            /**
                * 递推的时候，以f[one][k]的形式表示
                * 其中：one表示最后一个人的位置，k为包括这个人，一共有k个人
                * 原问题和子问题的关系：f[one][k]=max{f[left][k-1]*arr[one],g[left][k-
1]*arr[one]}
             */
            //规划数组
            long[][] f = new long[n + 1][kk + 1];//人直接对应坐标,n和kk都要+1
            long[][] g = new long[n + 1][kk + 1];
            //初始化k=1的情况
            for(int one = 1;one<=n;one++){
                f[one][1] = arr[one];
                g[one][1] = arr[one];
            }
            //自底向上递推
            for(int k=2;k<=kk;k++){
                for(int one = k;one<=n;one++){
                    //求解当one和k定的时候，最大的分割点
                    long tempmax = Long.MIN_VALUE;
                    long tempmin = Long.MAX_VALUE;
                    for(int left = Math.max(k-1,one-dd);left<=one-1;left++){
                        if(tempmax<Math.max(f[left][k-1]*arr[one],g[left][k-1]*arr[one])){
                            tempmax=Math.max(f[left][k-1]*arr[one],g[left][k-1]*arr[one]);
                        }
                        if(tempmin>Math.min(f[left][k-1]*arr[one],g[left][k-1]*arr[one])){
                            tempmin=Math.min(f[left][k-1]*arr[one],g[left][k-1]*arr[one]);
                        }
                    }
                    f[one][k] = tempmax;
                    g[one][k] = tempmin;
                }
            }
            //n选k最大的需要从最后一个最大的位置选
            long result = Long.MIN_VALUE;
            for(int one = kk;one<=n;one++){
                if(result<f[one][kk]){
                    result = f[one][kk];
                }
            }
            System.out.println(result);
        }
    }
}
```

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

```
import java.util.*;

public class Main {

 static class People {
     int height;
     int weight;

     public People(int weight, int height) {
         this.height = height;
         this.weight = weight;
     }
 }

 public static void main(String[] args) {

     Scanner scan = new Scanner(System.in);
     while (scan.hasNext()) {
         int n = scan.nextInt();
         People[] array = new People[n];
         for (int i = 0; i < n; ++i) {
             int index = scan.nextInt();
             array[index - 1] = new People(scan.nextInt(), scan.nextInt());
         }
           // 对所有的人信息进行排序
         Arrays.sort(array, new Comparator<People>() {
             public int compare(People p1, People p2) {
                 int result = Integer.compare(p1.height, p2.height);
                 if (result != 0)
                     return result;
                 else
                     return Integer.compare(p1.weight, p2.weight);
             }
         });

         int[] dp = new int[n];
         int max = Integer.MIN_VALUE;
         for (int i = 0; i < dp.length; ++i) {
             dp[i] = 1;
             for (int j = i - 1; j >= 0; --j) {
                 if (array[i].weight > array[j].weight
                     || (array[i].weight == array[j].weight && array[i].height ==
array[j].height)) {
                     dp[i] = Math.max(dp[i], dp[j] + 1);
                 }
             }
```

```
            max = Math.max(dp[i], max);
        }
        System.out.println(max);
    }
 }
}
```