[P1220]关路灯

Algorithm	动态规划 区间DP
Created	@Jul 16, 2020 4:09 PM
Difficulty	提高+/省选-
Related to 近期更新 (Property)	[P1220]关路灯
URL	https://www.luogu.com.cn/problem/P1220

题目链接：

关路灯

某一村庄在一条路线上安装了 n 盏路灯，每盏灯的功率有大有小（即同一段时间内消耗的电量有多有少）。老张就住在这条路中间某一路灯旁，他有一项工作就是每天早上天亮时一盏一盏地关掉这些路灯。 为了给村里节省电费，老张记录下了每盏路灯的位置和功率，他每次关灯时也都是尽快地去关，但是老张不知道怎样去关灯才能够最节省电。他每天都是在天亮时首先关掉自己所处位置的路灯，然

https://www.luogu.com.cn/problem/P1220

题解：

~~这题其实还是挺水的~~

首先，分析一下题目，会发现：**为保证情况最优，因此在任意时刻，被老张关掉的灯都可以组成一个连续的区间。**再分析一下情况，除了初始情况（老张在c点还没动时），每一个灯的开关情况都可以由前一个（或两个）状态转换而来。因此：

区间DP

以区间长度为阶段（外层循环），左端点为里层循环，那么双重循环就产生了。

最大的问题来了，dp数组怎么定？转移方程是什么？

我们可以想想，当只有l,r中间的灯都关了，老张会在哪？

首先，老张不会乱跑，因此，当只有l,r中间的灯都关了时，老张一定在l或r点上。因此我们只需要记录当i,j中间的灯都关了时，老张在i点和在j点分别已消耗的电的最小值，用$f[l][r][k](l,r\in[1,n],k\in 0,1)$三维数组表示。$k=0$时，老张在$i$点，$k=1$时，老张在$j$点。同时，$i$一定在$c$的左边，$j$一定在$c$的右边，这两个条件可以用来剪枝（广义的剪枝）。

再分析，可知$dp[l][r][0]$可以由$dp[l+1][r][0]$和$dp[l+1][r][1]$扩展而来（老张可以不撞南墙不回头，也可以半路会回头关另一端的灯）。设pos为电灯位置数组，设f为电灯频率数组，则稍微动动脑子，便可得到：

$$dp\left[l\right]\left[r\right]\left[0\right]=\max\begin{cases}dp\left[l+1\right]\left[r\right]\left[0\right]+\left(pos\left[l+1\right]-pos\left[l\right]\right)*\left(\sum_{i=1}^{l}f\left[i\right]+\sum_{i=r+1}^{n}f\left[i\right]\right)\\dp\left[l+1\right]\left[r\right]\left[1\right]+\left(pos\left[r\right]-pos\left[l\right]\right)*\left(\sum_{i=1}^{l}f\left[i\right]+\sum_{i=r+1}^{n}f\left[i\right]\right)\end{cases}$$

同理可以得到$dp[l][r][1]$的转移公式。

在写代码时，我们可以记录f数组的前缀和，这样后面的一群$\sum$就好搞了。

最后附上AC代码：

```
//P1220
// Created by admin on 2020/7/16.
//
#include <bits/stdc++.h>
#define int long long
using namespace std;
int n,c,f[100],pos[100],dp[100][100][2],sum[100];
signed main()
{
    cin>>n>>c;
    for(int i=1;i<=n;i++)
        cin>>pos[i]>>f[i];
    memset(dp,0x3f,sizeof(dp));
    for(int i=1;i<=n;i++)
        sum[i]=sum[i-1]+f[i];//记录前缀和
    dp[c][c][0]=dp[c][c][1]=0;
    for(int len=2;len<=n;len++)//穷举区间长度
        for(int l=1;l<=c&&l+len-1<=n;l++)  //穷举区间左端点
        {
            int r=l+len-1;
            if(r>=c)//剪枝
                break;
            dp[l][r][0]=min(dp[l+1][r][0]+(pos[l+1]-pos[l])*(sum[l]+sum[n]-sum[r]),dp[l+1][r][1]+(pos
[r]-pos[l])*(sum[l]+sum[n]-sum[r]));
            dp[l][r][1]=min(dp[l][r-1][1]+(pos[r]-pos[r-1])*(sum[l-1]+sum[n]-sum[r-1]),dp[l][r-1][0]+
(pos[r]-pos[l])*(sum[l-1]+sum[n]-sum[r-1]));
        }
    cout<<min(dp[1][n][0],dp[1][n][1])<<endl;
    return 0;
}
```