NKOJ2151　烽火传递

【题意】

在某两座城市之间有n个烽火台，每个烽火台发出信号都有一定的代价。为了使情报准确的传递，在每m个烽火台中至少要有一个发出信号。现输入n、m和点燃每个烽火台发信号的代价，请计算总共最少需要花费多少代价，才能使敌军来袭之时，情报能在这两座城市之间准确的传递！！！

【输入格式】

第一行有两个数n,m（1<=n,m<=1000000)分别表示n个烽火台，在m个烽火台中至少要有一个发出信号。

第二行为n个数，表示每一个烽火台的代价。

【输出格式】

一个数，即最小代价。

【样例输入】

5 3

1 2 5 6 2

【样例输出】

4

NKOJ2151 烽火传递【问题分析】

设DP[i]表示将情报从1号烽火台传到i号烽火台所需最小代价，且第i个烽火台要点燃。

由于第i个烽火台被点燃，也就意味着第i-1到第i-m+1号烽火台中，必有一个会被点燃。

于是有方程：

DP[i]=Cost[i]+Min{ DP[i-m],DP[i-m+1],...,DP[i-2],DP[i-1] }

DP[i]=Cost[i]+Min{ DP[k] }　　i-m ≤ k ≤ i-1　　1≤i≤n

但是，时间复杂度是？

O(nm)　1<=n,m<=1000000 显然会超时，怎么办？

Min{ DP[k] }表示求一个长度为m的区间的最小值，显然我们可以用(类似"滑动窗口")单调队列处理。

处理后求Min{ DP[k] }的时间复杂度为O(1),总的时间复杂度为**O(n)**。

【参考代码，STL版】

```
#include<deque>
deque<int>Q;                                        //双端队列，维护一个长度为m的单调递增区间
int DP[1000005],Cost[1000005];
int main()
{
    int n,m,i,ans=2000000000;
    scanf("%d%d",&n,&m);
    for(i=1;i<=n;i++)scanf("%d",&Cost[i]);
    //动态规划
    Q.push_back(0);                                 //初始化，将DP[0]入队
    for(i=1;i<=n;i++)
    {
        while(!Q.empty() && Q.front()<i-m)Q.pop_front();    //将[i-m,i-1]区间以外的数字从队中删除
        DP[i]=DP[Q.front()]+Cost[i];                        //DP[i]=Min{ DP[k] }+Cost[i]
        while(!Q.empty() && DP[Q.back()]>DP[i])Q.pop_back();    //DP[i]入队前，维护队列的单调性
        Q.push_back(i);
    }
    //输出结果
    for(i=n;i>=n-m+1;i--)
        if(ans>DP[i])ans=DP[i];
    printf("%d\n",ans);
}
```

【参考代码，手工队列版】

```
int Q[1000005],DP[1000005],Cost[1000005];

scanf("%d%d",&n,&m);
for(i=1;i<=n;i++)scanf("%d",&Cost[i]);
head=tail=1;
for(i=1;i<=n;i++)
{
        while(head<tail && Q[head]<i-m)head++;
        while(head<tail && DP[Q[tail-1]]>DP[i-1])tail--;
        Q[tail++]=i-1;
        DP[i]=DP[Q[head]]+Cost[i];
}
```

HDU3415　单调队列优化DP

【题意】

　给出一列n个数并将这列数的首数和尾数视为首尾相邻的环状的结构，求这些数字长度不大于k的最大子段和。

样例输入：

6 3（n和k)

-1 2 -6 5 -5 6

样例输出：

7

设Dp[i]表示以i结尾的满足约束的最大子段和，

Sum[i]表示从0到第i个数的和，状态转移方程：

Dp[i]=Sum[i]-min(Sum[j]) } , i-k<=j<i

最后结果为max(Dp[i]).

　　直接二重循环超时，可以考虑用**单调队列**优化。构造一个单调递减队列，维护操作与前例类似

　　min(Sum[j])即是找出长度不超过k的区间中的最小值

【SCOI 2010 DAY1】股票交易 NKOJ2148

最近lxhgww又迷上了投资股票，通过一段时间的观察和学习，他总结出了股票行情的一些规律。

通过一段时间的观察，lxhgww预测到了未来T天内某只股票的走势，第i天的股票买入价为每股APi，第i天的股票卖出价为每股BPi（数据保证对于每个i，都有APi>=BPi），但是每天不能无限制地交易，于是股票交易所规定第i天的一次买入至多只能购买ASi股，一次卖出至多只能卖出BSi股。

另外，股票交易所还制定了两个规定。为了避免大家疯狂交易，股票交易所规定在两次交易（某一天的买入或者卖出均算是一次交易）之间，至少要间隔W天，也就是说如果在第i天发生了交易，那么从第i+1天到第i+W天，均不能发生交易。同时，为了避免垄断，股票交易所还规定在任何时间，一个人的手里的股票数不能超过MaxP。

在第1天之前，lxhgww手里有一大笔钱（可以认为钱的数目无限），但是没有任何股票，当然，T天以后，lxhgww想要赚到最多的钱，聪明的程序员们，你们能帮助他吗？

Input

输入数据第一行包括3个整数，分别是T，MaxP，W。

接下来T行，第i行代表第i-1天的股票走势，每行4个整数，分别表示APi，BPi，ASi，BSi。

Output

输出数据为一行，包括1个数字，表示lxhgww能赚到的最多的钱数。

Sample Input

```
5 2 0
2 1 1 1
2 1 1 1
3 2 1 1
4 3 1 1
5 4 1 1
```

Sample Output

```
3
```

状态：用f[i][j]表示前i天中，第i天收盘时手中还持有j股的股票，能得到的最大收益（当前手中现金的最大数量）。　　枚举W天前买或卖了多少支就行了。

转移方程：f[i][j] = max{
f[i - W - 1][j - k1] - ap[i] * k1 //情况(1)在第i天买了k1股
f[i - W - 1][j + k1] + bp[i] * k2 //情况(2)在第i天卖了k2股
}
其中，k1 <= as[i], k2 <= bs[i]。

复杂度是O(T * maxP ^ 2)，显然过不了极限数据。怎么办？

对于买，令x=j-k1
max{ f[i - W - 1][j - k1] - ap[i] * k1 } 可化简为
max{ f[i - W - 1][x] + x*ap[i] }-j*ap[i]
因为0<=k1<=as[i],所以 0<=x<=j-as[i]
上述式子只跟x有关，可用单调队列优化
当j取值0到MaxP的过程中，用单调队列来快速取出f[i - W - 1][x] + x*ap[i]对应的最值
比如，当as[i]=3时，当j=3时，x讨论的区间为[0,0],当j=4时，x讨论的区间为[0,1],当j=5时，x讨论的区间为[0,2].....

卖同理讨论