# 斜率优化 dp

## 1. 斜率优化 dp 概述

斜率优化（国外称为凸壳优化）是一种优化动态规划状态转移的时间复杂度的方法，主要用于优化形如 dp[i]=minj∈[1,r]{Y(j)−K(i)X(j)}−A(i)的状态转移方程。

其中 K(i),A(i)是跟 i 有关的项

Y(j),X(j)是跟 j 有关的项

而在固定 i 之后，K(i),A(i)的值也就确定了。

### 1.1. 例题 打印文章 （HDU3507）

**【问题描述】**

给出 N 个单词，每个单词有个非负权值 Ci，现要将它们分成连续的若干段，每段的代价为此段单词的权值和的平方，还要加一个常数 M，即(ΣCi)^2+M。现在想求出一种最优方案，使得总费用之和最小。

**【输入】**

包含多组测试数据，对于每组测试数据：

第一行包含两个整数 N 和 M。(0<=N<=500 000，0<=M<=1 000)。

第二行为 N 个整数。

**【输出】**

输出仅一个整数，表示最小的价值。

**【样例输入】**

```
5 5
5 9 5 7 5
```

**【样例输出】**

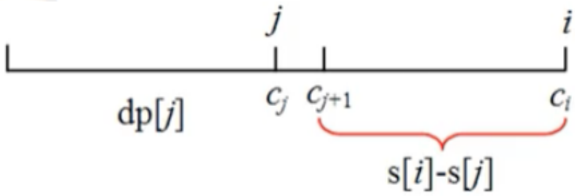
```
230
```

**【问题分析】**

本题求解打印文档的最小成本，可采用动态规划解决。

状态表示：dp[i]表示打印前 i 个单词的最小成本；s[i]表示前 i 个单词的打印成本之和。

状态转移:若前面已打印 j 个单词,当前行打印第 j+1…i 个单词,则 dp[i]等于打印前 j 个单词的最小成本加上打印 j+1…i 个单词的成本。即 dp[i] = min(dp[j] + (s[i] - s[j])^2) + M,0 <= j < i。




若枚举所有状态，则时间复杂度为 O（n^2），n = 500000，n^2 = 2.5 × 1e11，显然会超时，状态转移方程与 i、j 均有关，包含 i、j 有关的乘积，因此考虑斜率优化。

dp[i] = min(dp[j] + (s[i] - s[j])^2) + M，整理方程可得：

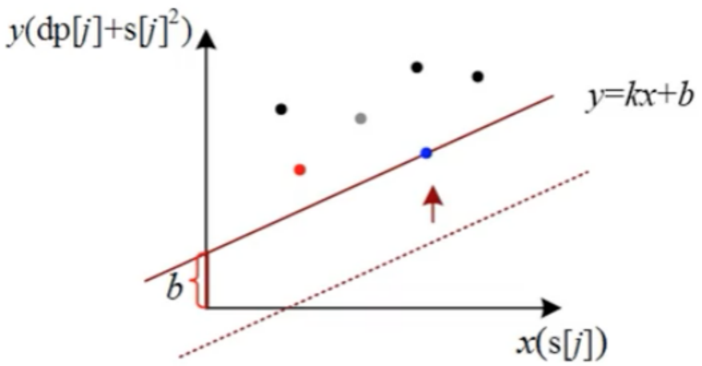
dp[i] = min(dp[j] + s[i]^2 - 2×s[i]×s[j] + s[j]^2) + M。

把仅与 j 有关的项放到等号左侧，其他项放到等号右侧：

dp[j] + s[j]^2 = 2×s[i]×s[j] + dp[i] - s[i]^2 - M。

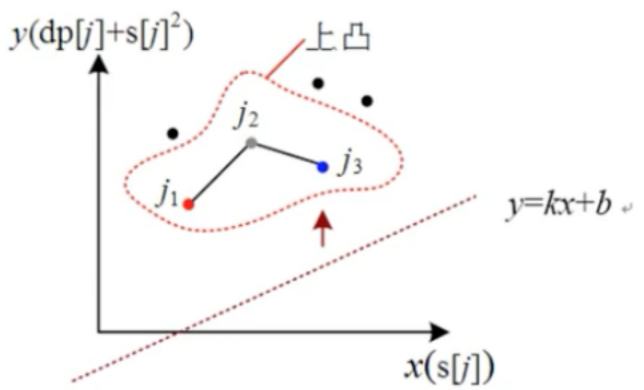
此时可以将上面的公式看作 y = kx + b 的线形表示，其中 y = dp[j] + s[j]^2，k = 2×s[i]，x = s[j]，b = dp[i] - s[i]^2 - M。其中，x 为横坐标，y 为纵坐标。

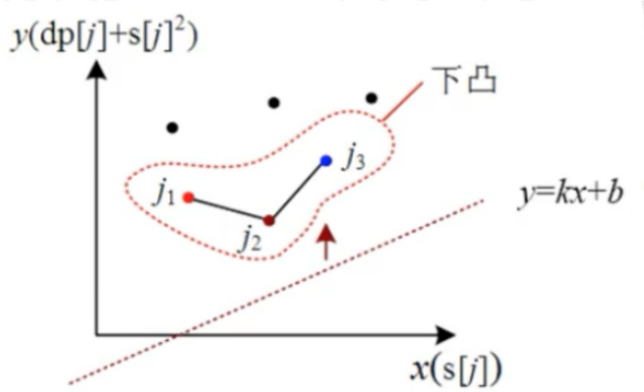
对于一个确定的 i 来说，斜率 k 是定值，b 也是定值，每个决策 j 都对应坐标系中的一个点（s[j]，dp[j] + s[j]^2），如何从众多决策点中找到线形方程的最小值呢？



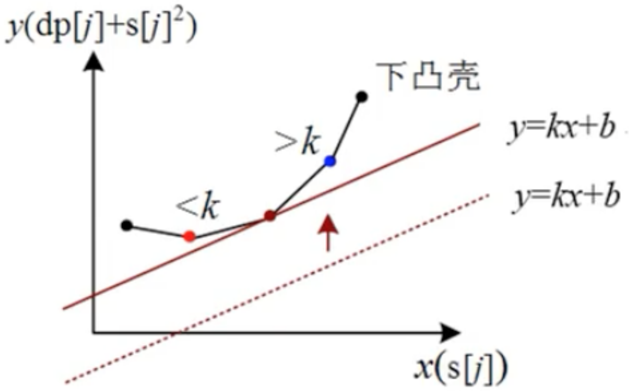

从上图中可以发现，在确定的斜率 k 下，最优决策点为平移后第一次与 j 相遇的点。

对于任意三个决策 j1 < j2 < j3，对应的 x 坐标 s[j]表示前 j 个单词的成本和，成本均为正数，所以 s[j1] < s[j2] < s[j3]。考虑 j2 是否有可能成为最优决策，有以下两种情况：




形成下凸形状的条件是 j1 到 j2 的线段斜率小于 j2 到 j3 的线段斜率，维护相邻两点的线段斜率单调递增即可保证下凸性。相邻两点的线段斜率单调递增的决策点集合叫做“下凸壳”，下凸壳上的点才有可能成为最优决策。



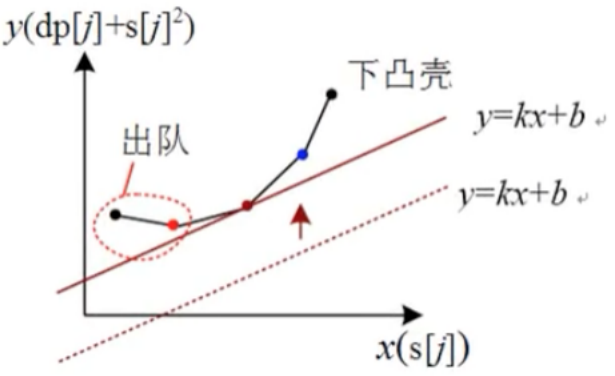

在本题中，0<=j<i，当 i 增加 1 时，j 也增加 1，所以可以省略对 j 的枚举，在枚举 i 时尝试用队列维护 j 的最优决策即可，采用队列时需要注意两个问题。

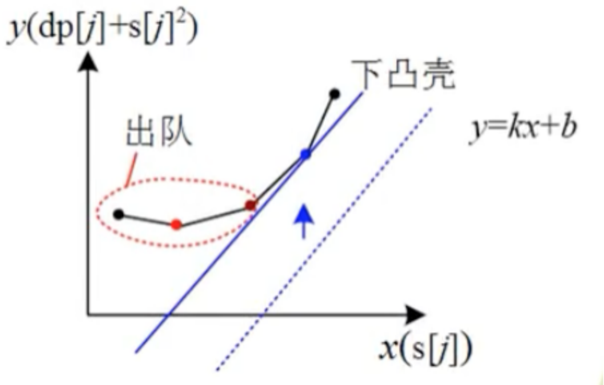
（1） 处理过时决策。

（2） 维护下凸壳。

### 1.1.1. 处理过时决策

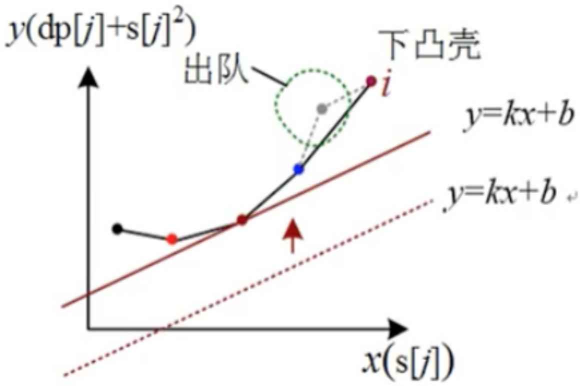
本题斜率 k = 2 × s[i]，s[i]为打印成本的前缀和，因此 k 随着 i 的递增而单调递增。每次将相邻两点线段斜率小于或者等于 k 的过时决策出队，此时队头就是最优决策，如下图所示。




### 1.1.2. 维护下凸壳

横坐标 s[j]随着 j 的递增而单调递增，新的决策 i 必然出现在下凸壳的最右端，检查队列尾部两点和第 i 个点是否满足下凸性，队列按照横坐标递增，维护相邻两点斜率递增的下凸壳。如下图所示：




## 1.2. 算法实现

（1） 枚举 i=1…n，k = 2 × s[i]。

（2） 处理过时决策。检查单调队列头部相邻两点的斜率，若小于等于 k，则队头出队，直到大于 k 为止。

（3） 取队头 j 为最优决策，计算 dp[i] = dp[j] + (s[i] - s[j])^2 + M。

（4） 维护下凸壳，若队列尾部两点和第 i 点不满足下凸性，则队尾出队，直到满足下凸性，将 i 入队。

（5） 最优解为 dp[n]。

### 1.2.1. 计算横纵坐标差值

dp[j] + s[j]^2 = 2×s[i]×s[j] + dp[i] - s[i]^2 - M。

计算相邻两点的斜率情况：

```
int GetY(int k1,int k2){ //计算 y2-y1
    return dp[k2]+s[k2]*s[k2]-(dp[k1]+s[k1]*s[k1]);
}

int GetX(int k1,int k2){ //计算 x2-x1
    return s[k2]-s[k1];
}
```

### 1.2.2. 计算 dp[i]

dp[i] = dp[j] + (s[i] - s[j])^2 + M。

```
int GetVal(int i,int j){
    return dp[j]+(s[i]-s[j])*(s[i]-s[j])+m;
}
```

### 1.2.3. 斜率优化 处理过时决策，维护下凸壳

```
int head=0,tail=0;
q[tail++]=0;
    for(int i=1;i<=n;i++) {
         while(head+1<tail&&GetY(q[head],q[head+1])<=2*s[i]*GetX(q[head],q[head+1]))
             head++;
         dp[i]=GetVal(i,q[head]);
         while(head+1<tail&&GetY(q[tail-1],i)*GetX(q[tail-2],q[tail-1])<=GetY(q[tail-2],q[ta
il-1])*GetX(q[tail-1],i))
          tail--;
         q[tail++]=i;
}
```

### 1.2.4. 完整代码实现

```
#include<cstdio>
using namespace std;
const int MAXN=5e5+5;
int s[MAXN],q[MAXN],dp[MAXN];
int n,m;

int GetY(int k1,int k2){
     return dp[k2]+s[k2]*s[k2]-(dp[k1]+s[k1]*s[k1]);
}

int GetX(int k1,int k2){
     return s[k2]-s[k1];
}

int GetVal(int i,int j){
    return dp[j]+(s[i]-s[j])*(s[i]-s[j])+m;
}

int main(){
    while(~scanf("%d%d",&n,&m)){
        s[0]=0;
        dp[0]=0;
        for(int i=1;i<=n;i++){
            scanf("%d",&s[i]);
            s[i]+=s[i-1];
        }
        int head=0,tail=0;
```

```
        q[tail++]=0; //因为可能是前面 i 个全部作为一段才是最小值
        for(int i=1;i<=n;i++) {//head+1<tail 保证队列里面至少有两个值
            while(head+1<tail&&GetY(q[head],q[head+1])<=2*s[i]*GetX(q[head],q[head+1]))
                head++; //head+1 比 head 更优
            dp[i]=GetVal(i,q[head]);
            while(head+1<tail&&GetY(q[tail-1],i)*GetX(q[tail-2],q[tail-1])<=GetY(q[tail-2]
,q[tail-1])*GetX(q[tail-1],i))
                tail--;//维护凹包，凸包已经被证明中间的不符合
            q[tail++]=i;
        }
        printf("%d\n",dp[n]);
    }
    return 0;
}
```

# 2. 基础题目

## 2.1. 洛谷 P2365 任务安排 1+2

**【问题描述】**

有 N 个任务排成一个序列在一台机器上等待执行，它们的顺序不得改变。机器会把这 N 个任务分成若干批，每一批包含连续的若干个任务。从时刻 0 开始，任务被分批加工，执行第 i 个任务所需的时间是 Ti。另外，在每批任务开始前，机器需要 S 的启动时间，故执行一批任务所需的时间是启动时间 S 加上每个任务所需时间之和。

一个任务执行后，将在机器中稍作等待，直至该批任务全部执行完毕。也就是说，同一批任务将在同一时刻完成。每个任务的费用是它的完成时刻乘以一个费用系数 Ci。

请为机器规划一个分组方案，使得总费用最小。

**【输入】**

第一行是 N。第二行是 S。

下面 N 行每行有一对正整数，分别为 Ti 和 CI，均为不大于 100 的正整数，表示第 i 个任务单独完成所需的时间是 Ti 及其费用系数 Ci。

**【输出】**

一个数，最小的总费用。

**【样例输入】**

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

**【样例输出】**

```
153
```

数据规模 1<N<=5 000, （数据规模 1<N<=3*10^5）， 1<=S<=50,1<=TI,Ci<=1 00。

**【题目分析】**

定义 dp[i][j]表示前 i 个任务被分为 j 批的最小费用值。

定义 sumt[i]表示 t 的前缀和,sumf[i]表示 f 的前缀和。

易得

dp[i][j]=min{dp[k][j−1]+(sumt[i]+s×j)(sumf[i]−sumf[k])},k∈[0,i)

初始化 dp[0][0]=0,其余均为一个极大值。于是便可以照这个写出一个 O(n3)的 TLE 代码。

```
for(int i=1;i<=n;i++) {
    for(int j=1;j<=i;j++) {
        for(int k=0;k<i;k++) {
            dp[i][j]=min(dp[i][j],dp[k][j-1]+(sumt[i]+s*j)*(sumf[i]-sumf[k]));
        }
        if(i==n) ans=min(ans,dp[i][j]);
    }
}
```

观察式子，发现 j 的作用仅是为了计算此前个过程中的启动时间和，但事实上，既然这个时间要乘上此后的所有 f，不如提前加入其中。因为若分完前 j 个任务后，要等待 s 秒，则后续费用一定会加上(sumf[n]−sumf[j])×s，于是可以提前加进去，这样 dp 数组可以省去一维，状转方程变为 dp[i]=min{dp[j]+sumt[i]×(sumf[i]−sumf[j])+s×(sumf[n]−sumf[j])},j∈[0,i)。这样又可以写出 O(n^2)的代码。

```
for(int i=1;i<=n;i++) {
    for(intj=0;j<i;j++)
        dp[i]=min(dp[i],dp[j]+sumt[i]*(sumf[i]-sumf[j])+s*(sumf[n]-sumf[j]));
}
```

再整理，可得

dp[i]=min{dp[j]-(sumt[i]+s)×sumf[j]}+sumt[i]×sumf[i]+s×sumf[n],j∈[0,i)

对于每一个 i，min 外都为常量。

观察 min 内的式子 dp[j]-(sumt[i]+s)×sumf[j]，令 k = sumt[i] + s，则原式变为-k×sumf[j]+dp[j]。

就凑成了我们上面所讲过的一次函数。将其看作一条斜率为 k 且过点（sumf[j]，dp[j]）的直线。

截距为 dp[j]-(sumt[i]+s)×sumf[j]。

令 k < j < i ，如果转移 j 比转移 k 要更优的话，则

dp[j]-(sumt[i]+s)×sumf[j]<dp[k]-(sumt[i]+s)×sumf[k]

将两项移到同侧：

dp[j]-dp[k]<(sumt[i]+s)(sumf[j]-sumf[k])

即 j 与 k 的连线段的斜率小于 sumt[i]+s 时，k 就不需要了。

同时维护下凸性，令 j1 < j2 < j3 ，若 j2 有可能成为最优决策，则其满足下凸性，j1 与 j2 连成线段的斜率要小于 j 2 与 j 3 连成线段的斜率。

便可以以此建立单调队列，维护这个下凸壳。

每个元素只入队一次，时间复杂度 O(n)

**【参考代码】**

```
#include<bits/stdc++.h>
#define N 5010

using namespace std;

int n,s,f[N],t[N],sumf[N],sumt[N],dp[N],q[N],head=1,tail=1;

int main() {
    cin>>n;
```

```
    cin>>s;
    for(int i=1;i<=n;i++) {
        cin>>t[i]>>f[i];
        sumf[i]=sumf[i-1]+f[i];
        sumt[i]=sumt[i-1]+t[i];
    }

    memset(dp,0x3f,sizeof(dp));
    q[head]=0;
    dp[0]=0;
    for(int i=1;i<=n;i++) {
        while(head<tail&&dp[q[head+1]]-dp[q[head]]<=(sumt[i]+s)*(sumf[q[head+1]]-sumf[q[hea
d]])) head++;
        dp[i]=dp[q[head]]-(sumt[i]+s)*sumf[q[head]]+sumt[i]*sumf[i]+s*sumf[n];
        while(head<tail&&(dp[q[tail]]-dp[q[tail-1]])*(sumf[i]-sumf[q[tail]])>=(dp[i]-dp[q[t
ail]])*(sumf[q[tail]]-sumf[q[tail-1]])) tail--;
        q[++tail]=i;
    }

    cout<<dp[n]<<endl;

    return 0;
}
```

## 2.2. 洛谷 P5785 任务安排 3

**【问题描述】**

有 N 个任务排成一个序列在一台机器上等待执行，它们的顺序不得改变。机器会把这 N 个任务分成若干批，每一批包含连续的若干个任务。从时刻 0 开始，任务被分批加工，执行第 i 个任务所需的时间是 Ti。另外，在每批任务开始前，机器需要 S 的启动时间，故执行一批任务所需的时间是启动时间 S 加上每个任务所需时间之和。

一个任务执行后，将在机器中稍作等待，直至该批任务全部执行完毕。也就是说，同一批任务将在同一时刻完成。每个任务的费用是它的完成时刻乘以一个费用系数 Ci。

请为机器规划一个分组方案，使得总费用最小。

**【输入】**

第一行是 N。第二行是 S。

下面 N 行每行有一对正整数，分别为 Ti 和 CI，均为不大于 100 的正整数，表示第 i 个任务单独完成所需的时间是 Ti 及其费用系数 Ci。

**【输出】**

一个数，最小的总费用。

**【样例输入】**

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

**【样例输出】**

```
153
```

数据规模 1<N<=3*10^5, 0<=S,C <=512,-512<=T<=512。

**【题目分析】**

本题与之前的那道任务安排不同在于数据量增大了，且 t 的值可能为负的。因此 O（n2）与 O(n3)肯定无法通过，同时考虑斜率优化，由于 t 值可能为负，所以 sumt[i]+s 不再具有单调性，那么上面保存的相邻两点连线段斜率大于 sumt[i]+s 的方法便不再适用。

因此不能弹出队头，而是要维护整个凸壳。所维护的凸壳具有下凸性，因此最优的决策点一定是左侧的线段斜率小于 sumt[i]+s，右侧的线段斜率大于 sumt[i]+s，所以便可以用二分查找，来寻找这个最优决策点。

队尾的操作维护与上题相同，对于队头我们不再将其弹出。

**【参考代码】**

```
#include<bits/stdc++.h>
#define N 300010
#define ll long long

using namespace std;

int n,s,head=1,tail=1,q[N];
ll f[N],t[N],sumf[N],sumt[N],dp[N];

int binary_search(int i) {
    if(head==tail) return q[head];
    int l=head,r=tail;
    while(l<r) {
        int mid=l+r>>1;
        if(dp[q[mid+1]]-dp[q[mid]]<=(sumt[i]+s)*(sumf[q[mid+1]]-sumf[q[mid]])) l=mid+1;
        else r=mid;
    }

    return q[l];
}

int main() {
    cin>>n;
    cin>>s;
    for(int i=1;i<=n;i++) {
        cin>>t[i]>>f[i];
        sumf[i]=sumf[i-1]+f[i];
        sumt[i]=sumt[i-1]+t[i];
    }

    memset(dp,0x3f,sizeof(dp));
    q[head]=0;
    dp[0]=0;
    for(int i=1;i<=n;i++) {
        int p=binary_search(i);
        dp[i]=dp[p]-(sumt[i]+s)*sumf[p]+sumt[i]*sumf[i]+s*sumf[n];
        while(head<tail&&(dp[q[tail]]-dp[q[tail-1]])*(sumf[i]-sumf[q[tail]])>=(dp[i]-dp[q[ta
il]])*(sumf[q[tail]]-sumf[q[tail-1]])) tail--;
        q[++tail]=i;
    }
```

```
    cout<<dp[n]<<endl;

    return 0;
}
```

# 3. 强化练习

## 3.1. 洛谷 P3195 玩具装箱

**【问题描述】**

P 教授要去看奥运，但是他舍不得他的玩具，于是他决定把所有的玩具运到北京。他使用自己的压缩器进行压缩。这个压缩器可以将任意物品变成一维，再放到一种特殊的一维容器中。P 教授有编号为 1…N 件玩具，玩具经过压缩后会变成一维，第 i 件件玩具压缩后一维长度为 CI。为了方便整理，P 教授要求：在一个一维容器中，玩具的编号是连续的。如果一个一维容器中有多个玩具，那么两件玩具之间要加入一个单位长度的填充物。形式地说，如果要将 i 号玩具到 j 号玩具放到同一个容器中，则容器长度将为：X=j-i+Sigma（$C_k$），i<=K<=j。

制作容器的费用与容器的长度有关，根据教授研究，如果容器长度为 X，其制作费用为(X-L)^2，其中 L 是一个常量。P 教授不关心容器的数目，他可以制作出任意长度的容器，甚至超过 L。但他希望费用最小。

**【输入】**

第一行输入两个整数 N, L。

接下来 N 行，输入整数 Ci。

**【输出】**

输出最小费用。

**【样例输入】**

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

**【样例输出】**

```
1
```

**【算法分析】**

f[i]=min{f[j]+(sum[i]+i−(sum[j]+j+L+1))^2}

j∈(0, i)

对式子转化成 y = kx+b 的形式

f[i]=min{f[j]+(sum[i]+i−(sum[j]+j+L+1))^2}

A[i]=sum[i]+i, B[j]=sum[j]+j+L+1

我们希望把 i 和 j 打包，让求得的 f[i]变成截距

f[j]+B[j]^2=2*A[i]*B[j]+f[i]−A[i]^2（把 i 和 j 一定要分离开）

然后 f[i]−A[i]^2 变成了截距 b，B[j]看成 x，f[j]+B[j]^2 看成 y

则 y=(2*A[i])x+b，其中 x, y 都是只与 j 有关的不变量，这就在坐标系内确定了一个点。

对答案产生贡献的点一定在一个下凸包上

所以我们用单调队列通过斜率维护下凸包即可（斜率和 x 具有单调性），可以用滑动窗口解决。

**【参考代码】**

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5e4 + 10;
#define LL long long
double sum[maxn],f[maxn];
int n,L;
int head=1,tail=0,q[maxn];
inline double A(int i){return sum[i]+i;}
inline double B(int i){return A(i)+L+1;}
inline double X(int i){return B(i);}
inline double Y(int i){return f[i]+B(i)*B(i);}
inline double k(int i,int j){
    return (double)(Y(j)-Y(i))/(X(j)-X(i));
}
int main(){
    scanf("%d%d",&n,&L);
    for(int i=1;i<=n;i++){
        scanf("%lf",&sum[i]);sum[i]+=sum[i-1];
    }
    q[++tail]=0;
    for(int i=1;i<=n;i++){
        while(head<tail && k(q[head],q[head+1])<2*A(i))head++;
        f[i]=f[q[head]]+(A(i)-B(q[head]))*(A(i)-B(q[head]));
        while(head<tail && k(q[tail-1],i)<k(q[tail-1],q[tail]))tail--;
        q[++tail]=i;
    }
    printf("%lld\n",(LL)f[n]);
    return 0;
}
```

## 3.2. 洛谷 P3628 特别行动队

**【问题描述】**

你有一支由 n 名预备役士兵组成的部队，士兵分别编号为 1 到 n，要将他们拆分成若干特别行动队调入战场。出于默契的考虑，同一支特别行动队中队员的编号应该连续，即为形如（i，i+1+⋯+k）的序列。

编号为 i 的士兵的初始战斗力为 $x_i$，一支特别行动队的初始战斗力 x 为队内士兵初始战斗力之和，即 x=（$x_i$）+（$x_{i+1}$）+⋯+（$x_{i+k}$）。

通过长期的观察，你总结出一支特别行动队的初始战斗力 x 将按如下经验公式修正为 x'：x' =ax^2+bx+c，其中 a，b，c（a<0）是已知的系数。

作为部队统帅，现在你要为这支部队进行编队，使得所有特别行动队修正后战斗力之和最大。试求出这个最大和。

例如，你有 4 名士兵，x1=1，X2=2，X3=3，X4=4。经验公式中的参数为 a=-1，b=10，c=-20。此时，最佳方案是将士兵组成 3 个特别行动队：第一队包含士兵 1 和士兵 2，第二队包含士兵 3，第三队包含士兵 4。特别行动队的初始战斗力分别为 4，3，4，修正后的战斗力分别为 4，1，4。修正后的战斗力和为 9，没有其它方案能使修正后的战斗力和更大。

**【输入】**

输入由三行组成。

第一行包含一个整数 n，表示士兵的总数。

第二行包含三个整数 a，b，c，经验公式中各项的系数。

第三行包含 n 个用空格分隔的整数 x1，x2，…xn，分别表示编号为 1，2…n 的士兵的初始战斗力。

**【输出】**

输出一个整数，表示所有特别行动队修正后战斗力之和的最大值。

**【样例输入】**

```
4
-1 10 -20
2 2 3 4
```

**【样例输出】**

```
9
```

**【算法分析】**

dp[i]表示前 i 个人分为若干个队伍的最大战斗力，s[i]维护前缀和。

容易得到状态转移方程

转移方程：$dp[i]=max(dp[j]+A*(s[i]-s[j])^2+B*(s[i]-s[j])+C)$

写成可以斜率优化的式子：$dp[j]+A*s[j]^2-B*s[j]+C=2*A*s[i]*s[j]+dp[i]-A*s[j]^2-B*s[i]$

然后求 dp[i]最大值，于是维护上凸包；横坐标单调增，斜率单调减，所以直接上单调队列即可。

**【参考代码】**

```
#include <bits/stdc++.h>
using namespace std;
typedef long long lint;
#define N 1000010
lint n,a,b,c,d[N],s[N],dp[N];
lint X(int i)
{
    return 2*a*s[i];
}
lint Y(int i)
{
    return dp[i]+a*s[i]*s[i]-b*s[i];
}
lint G(int i,int j)
{
    return dp[j]+a*s[i]*s[i]-2*a*s[i]*s[j]+a*s[j]*s[j]+b*s[i]-b*s[j]+c;
}
double K(int i,int j)
{
    return (double)(Y(i)-Y(j))/(double)(X(i)-X(j));
}
lint q[N],head,tail;
int main()
{
    scanf("%lld%lld%lld%lld",&n,&a,&b,&c);
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",&d[i]);
    }
    for(int i=1;i<=n;i++)
```

```
    {
        s[i]=s[i-1]+d[i];
    }
    for(int i=1;i<=n;i++)
    {
        while(head<tail&&K(q[head],q[head+1])<=s[i]) head++;
        dp[i]=G(i,q[head]);
        while(head<tail&&K(i,q[tail])<=K(q[tail],q[tail-1])) tail--;
        q[++tail]=i;
    }
    printf("%lld\n",dp[n]);
    return 0;
}
```

## 3.3. 洛谷 P4360 锯木厂选址

**【问题描述】**

从山顶上到山底下沿着一条直线种植了 n 棵老树。当地的政府决定把他们砍下来。为了不浪费任何一棵木材，树被砍倒后要运送到锯木厂。

木材只能朝山下运。山脚下有一个锯木厂。另外两个锯木厂将新修建在山路上。你必须决定在哪里修建这两个锯木厂，使得运输的费用总和最小。假定运输每公斤木材每米需要一分钱。

你的任务是编写一个程序，读入树的个数和他们的重量与位置，计算最小运输费用。

**【输入】**

输入的第一行为一个正整数 n，表示树的个数 。树从山顶到山脚按照 1，2，…n 标号。

接下来 n 行，每行有两个整数 wi 和 di。分别表示第 i 棵树的重量（公斤为单位）和第 i 棵树和第 i+1 棵树之间的距离，1<=wi <=10 000，1<= di <=10 000。最后一个数 di，表示第 n 棵树到山脚的锯木厂的距离。

保证所有树运到山脚的锯木厂所需要到费用小于 2 000 000 000 分

**【输出】**

输出仅一个数，表示最小的运输费用。

**【样例输入】**

```
9
1 2
2 1
3 3
1 1
3 2
1 6
2 1
1 2
1 1
```

**【样例输出】**

```
26
```

**【算法分析】**

令 $f_i$ 表示仅在 $i$ 位置建立一个锯木厂的最小费用，$dis_i$ 表示从山脚到 $i$ 位置的距离，$sum_i$ 表示从山顶到 $i$ 位置的树的重量和，可以直接预处理出来。

那么第二个锯木厂建立在位置$i$的费用就是 $\min\{f_j-dis_i\times(sum_i-sum_j)\mid j<i\}$

考虑两个决策点$j,k\,(j<k)$，若$j$对于当前点更优，那么：

$fj-disi \times (sumi-sumj) < fk-disi \times (sumi-sumk)$
$fj-fk < disi \times (sumk-sumj)$
$(fj-fk)/(sumk-sumj) < disi$
$(fk-fj)/(sumk-sumj) > -disi$
维护 $(sum, f)$ 的下凸包。

**【参考代码】**

```
#include<bits/stdc++.h>
using namespace std;
#define N 20005
#define Db double
#define Min(x,y) ((x)<(y)?x:y)
int sum[N],dis[N],f[N],que[N];
inline Db slope(int j,int k)
{
    return Db(f[k]-f[j])/Db(sum[k]-sum[j]);
}
int main()
{
    int n,i,ans=INT_MAX,head=1,tail=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    scanf("%d%d",&sum[i],&dis[i]),sum[i]+=sum[i-1];
    for(int i=n;i>=1;i--)
    dis[i]+=dis[i+1],f[0]+=dis[i]*(sum[i]-sum[i-1]);
    for(int i=1;i<=n;i++)
    {
        f[i]=f[0]-sum[i]*dis[i];
        while(head<tail&&slope(que[head],que[head+1])<=-dis[i])head++;
        if(i>1)ans=Min(ans,f[que[head]]-dis[i]*(sum[i]-sum[que[head]]));
        while(head<tail&&slope(que[tail-1],que[tail])>=slope(que[tail],i))tail--;
        que[++tail]=i;
    }
    printf("%d",ans);
    return 0;
}
```