单调队列：单调队列即保持队列中的元素单调递增（或递减）的这样一个队列，可以从两头删除，只能从队尾插入。单调队列的具体作用在于，由于保持队列中的元素满足单调性，对于上述问题中的每个 j，可以用 O(1)的时间找到对应的 s[i]。（保持队列中的元素单调增的话，队首元素便是所要的元素了）。

维护方法：对于每个 i，我们插入 s[i-1]（为什么不是 s[i]?自己想想吧），插入时从队尾插入。为了保证队列的单调性，我们从队尾开始删除元素，直到队尾元素比当前需要插入的元素优（本题中是值比待插入元素小，位置比待插入元素靠前，不过后面这一个条件可以不考虑），就将当前元素插入到队尾。之所以可以将之前的队列尾部元素全部删除，是因为它们已经不可能成为最优的元素了，因为当前要插入的元素位置比它们靠前，值比它们小。我们要找的，是满足(i>=j-k+1)的 i 中最小的 s[i]，位置越大越可能成为后面的 j 的最优 s[i]。由于笔者语言文字功底不行，还请朋友们自己思索...

这个队列满足一个性质：数列中的所有元素只会进出单调队列一次。所以，这道题的时间复杂度就变成了 O（N）了。

Problem 1894 志愿者选拔

【题目概述】见下图。

每组数据第一行为”START”，表示面试开始

接下来的数据中有三种情况：

	输入	含义
1	C NAME RP_VALUE	名字为NAME的人品值为RP_VALUE的同学加入面试队伍。(名字长度不大于5，0 <= RP_VALUE <= 1,000,000,000)
2	G	排在面试队伍最前面的同学面试结束离开考场。
3	Q	主面试官John想知道当前正在接受面试的队伍中人品最高的值是多少。

最后一行为”END”，表示所有的面试结束，面试的同学们可以依次离开了。

所有参加面试的同学总人数不超过1,000,000

【题目分析】一个简单的单调队列的题目。模拟就可以。注意队首和队尾的编号的记录。

【题目代码】//Name: fzu_Problem 1894 志愿者选拔

```
#include <iostream>
using namespace std;
const int maxn = 1000002;
struct node {
    int key, tag;
    node(int k=0,int t=0):key(k),tag(t){}
}q[maxn];
int head, tail;
int main() {
```

```
    //freopen("in.in", "r", stdin);
    int t, i, j, val; char s[120]; scanf("%d", &t);
    q[0] = -1;
    while(t-- && scanf("%s", s)) {
        head = 1; tail = 0; i = 0; j = 1;
        while(scanf("%s", s)) {
            if(!strcmp(s, "END")) break;
            if(s[0] == 'C') {
                scanf("%s %d", s, &val);
                while(head <= tail && q[tail].key <= val) tail--;
                q[++tail] = node(val, ++i);
             } else if(s[0] == 'G') {
                while(head <= tail && q[head].tag <= j) head++;
                j++;
             } else
                printf("%d\n", head > tail ? -1:q[head].key);
        }
    }
    return 0;
}
```

Sliding Window

【题目概述】给定一个长度为 N(N<= 10^6) 整型序列，和一个长度为 K 扫面框，求每次扫描的最大和最小值。

The array is [1 3 -1 -3 5 3 6 7], and *k* is 3.

Window position	Minimum value	Maximum value
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

【问题分析】这个题目是学习单调队列的人要做的第一个题目，包含了最大单调队列和最小单调队列。用最大队列来维护区间最大值，最小队列来维护区间最小值。下面叙述最大队列，最小队列做法类似。

最大队列保证队列中各个元素大小单调递减（即，最大元素在对头，最小的在队尾），同时每个元素的下标单调递增（按校标递增的顺序添加这点可以不用考虑）。这样便保证队首元素最大，而且更新的时候队首永远是当前最大。因此，这个队列需要在两头都可以进行删除，在队尾插入。

维护方法：在每次插入的时候，先判断队尾元素，如果不比待插入元素大就删除，不断删除队尾直到队尾元素大于待插入元素或者队空。删除的时候，判断队首，如果队首元素下标小于当前段左边界就删除，不断删除队首直

到队首元素下标大于等于当前段左边界（注意：这时队列肯定不为空），队首元素就是当前段的最优解。

【题目代码】下面是 AC 的程序，5 秒多，呵呵！不知道他们几百秒的人是用什么做的。

2823	Accepted	26600K	5407MS	C++	1565B

```
//Name: pku_2823_Sliding_Window
//Author: longxiaozhi
//Tag: 单调队列
//Root: 最大单调队列+ 最小单调队列。
#include <iostream>
using namespace std;
const int inf = -1u>>1;
const int maxn = 1e6;
struct node {
    int val, tag;
    node(int v=0, int t=0):val(v),tag(t){}
    void pt() { printf("%d %d\n", val, tag); }
}q[2][maxn+1];
int h1, r1, h2, r2;
int key[maxn+1], n, k;
int a[maxn+1], b[maxn+1];
int main() {
    freopen("in.in", "r", stdin);
    q[0][0] = node(-inf, 0);
    q[1][0] = node(+inf, 0);
    int i;
    while(scanf("%d %d", &n, &k) != EOF) {
        if(k > n) k = n;
        for(i = 1; i <= n; i++) scanf("%d", key+i);
        h1 = h2 = 1; r1 = r2 = 0;
        for(i = 1; i < k; i++) {
            while(h1 <= r1 && q[0][r1].val < key[i]) r1--;
            q[0][++r1] = node(key[i], i);
            while(h2 <= r2 && q[1][r2].val > key[i]) r2--;
            q[1][++r2] = node(key[i], i);
        }
        for(i = k; i <= n; i++) {
            while(h1 <= r1 && q[0][r1].val < key[i]) r1--;
            q[0][++r1] = node(key[i], i);
            while(h1 <= r1 && q[0][h1].tag <= i-k) ++h1;
            a[i-k] = q[0][h1].val;
            while(h2 <= r2 && q[1][r2].val > key[i]) r2--;
            q[1][++r2] = node(key[i], i);
            while(h2 <= r2 && q[1][h2].tag <= i-k) ++h2;
            b[i-k] = q[1][h2].val;
        }
        for(i = 0; i < n-k; i++) printf("%d ", b[i]);
```

```
        printf("%d\n", b[i]);
        for(i = 0; i < n-k; i++) printf("%d ", a[i]);
        printf("%d\n", a[i]);
    }
    return 0;
}
```

Max Sum of Max-K-sub-sequence

【题目概述】 给出一个有 N 个数字（-1000..1000，N<=10^5）的环状序列，让你求一个和最大的连续子序列，且这个连续子序列的长度小于等于 K。

【题目分析】 因为序列是环状的，所以可以在序列后面复制一段（或者复制前 k 个数字）。如果用 s[i]来表示复制过后的序列的前 i 个数的和，那么任意一个子序列[i..j]的和就等于[j]-s[i-1]。对于每一个 j，用 s[j]减去最小的一个 s[i](i>=j-k+1)就可以得到以 j 为终点长度不大于 k 的和最大的序列了。将原问题转化为这样一个问题后，就可以用单调队列解决了。

【题目代码】下面是两种 AC 的代码，一个用了二分查找 O(logn)，另外一个 O(N)；

Accepted	3415	203MS	2924K	1494 B	C++	longxiaozhi
Accepted	3415	171MS	2924K	1487 B	C++	longxiaozhi

```
//Name: hdu_3415_Max Sum of Max-K-sub-sequence
//Author: longxiaozhi
//Tag: 单调队列
#include <iostream>
using namespace std;
const int maxn = 100005;
struct node {
    int val, tag;
    node(int v=0,int t=0):val(v),tag(t){}
    void pt() { printf("%d %d\n", tag, val); }
}q[maxn<<1];
int head, rear;
//q[0] = node(-1<<30, 0);
int s[maxn<<2], a[maxn], n, k;
int st, ed, sum;
int bSearch(int v) {
    int mn = head, mx = rear, md;
    while(mx >= mn) {
        md = (mn + mx)>>1;
        if(q[md].val > v) mx = md-1;
        else mn = md+1;
    }
    return mx;
```

```
}
int main() {
    q[0] = node(-1<<30, 0);
    //freopen("in.in", "r", stdin);
    int t, i, tmp; scanf("%d", &t);
    while(t-- && scanf("%d %d", &n, &k)) {
        for(i = 1; i <= n; i++) {
            scanf("%d", a+i);
            s[i] = s[i-1]+ a[i];
        }
        for(i = i; i <= n+k; i++)
            s[i] = s[i-1] + a[i-n];
        q[head=rear = 1] = node(0, 1);
        st = ed = 1; sum = s[1];
        for(i = 2; i <= n+k; i++) {
            //while(head <= rear && q[rear].val > s[i-1]) rear--;
            rear = bSearch(s[i-1]);
            q[++rear] = node(s[i-1], i);
            while(head <= rear && q[head].tag <= i-k) head++;
            tmp = s[i] - q[head].val;
            if(tmp > sum) sum = tmp, st = q[head].tag, ed = i;
        }
        if(st > n) st -= n;
        if(ed > n) ed -= n;
        printf("%d %d %d\n", sum, st, ed);
    }
    return 0;
}
```
http://acm.hdu.edu.cn/showproblem.php?pid=3415

Necklace

【题目概述】告诉一个有字母 C 和字母 J 组成的长度不大于 1E6 的环行排列，现在从某个地方断开，由此向前或者向后，如果可以使得其中之一满足任何时刻数的的 C 不小于 J.那么就成合法的切开位置，求合法的位置的数目。
【题目分析】先将环行的排列展开，在展开的地方在续上一个展开串，使之和环行排列等价。 然后以任意位置为起点，记录下 C 的个数剪掉 J 的个数 s[i], 如果 i 位置的左边或右边有 sL[i] >= 0 或者右边 SR[i] >= 0（SL 和 SR 都是相对的个数），那么就可以说段开点 i 是合法的。这里可以用一个最小单点队列（队首元素为区间最小值）来表示。
【题目代码】

```
//Name: hdu_3474_Necklace
//Author: longxiaozhi
//Tag: 单调队列
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
```

```
const int maxn = 1E6;
struct node {
    int val, tag;
    node(int v=0,int t=0):val(v),tag(t){}
    void pt() { printf("%d %d\n", val, tag);}
}q[(maxn<<1)+1];
int head, rear;
bool v[2][maxn];
char neck[(maxn<<1)+1];
int s[(maxn<<1)+1], n, m, ans;
void solve(int n, int m, int k) {
    for(int i = 1; i < m; i++)
        s[i] = s[i-1] + (neck[i] == 'C' ? 1:-1);
    head = 1; rear = 0;
    for(int i = 1; i < m; i++){
        while(head <= rear && q[rear].val >= s[i]) rear--;
        q[++rear] = node(s[i], i);
        if(i >= n) {
            while(head <= rear && q[head].tag <= i-n) ++head;
            if(s[i-n] - q[head].val <= 0)  v[k][i-n] = 1;
        }
    }
}

int main() {
    //freopen("in.in", "r", stdin);
    int t, i, ca = 0; scanf("%d", &t);
    while(t-- && scanf("%s", neck+1)) {
        n = strlen(neck + 1);
        m = 2 * n + 1;
        for(i = 1; i <= n; i++)
            neck[i+n] = neck[i];
        neck[m] = 0;
        memset(v, 0, sizeof(v));
        solve(n, m, 0);
        reverse(neck+1, neck+m);
        solve(n, m, 1);
        ans = 0;
        for(i = 1; i <= n; i++)
            if(v[0][i] || v[1][n-i]) ans++;
        printf("Case %d: %d\n",++ca, ans);
    }
}
```

K-Anonymous Sequence

【题目概述】将题目转化下：将一个升序的，有 N 个元素的序列，分组。要求每组的元素不少于 K 个，计算出组内各元素与最小元素的之差的和，将每组的这个值加起来，其和要最小。

【题目分析】详细的过程见我的《斜率优化 DP +单调队列》的总结，这里只有大致过程（某神牛的）

1. 状态转移方程：设$dp[i]$ 表示前 i 个元素进行划分（最多为 i 组）的最小的值，则，

$$dp[i] = \min(dp[i] + sum[i] - sum[j] - a[j+1](i-j), \quad \leq j \leq i-1)$$

1. 证明较优决策点对后续状态影响的持续性

2. 求斜率方程:一般化为左边是 J，K，右边是 I 的形式

假设 J<K，且在 K 点的决策比 J 好，则有

$$dp\quad[j] + sum[i] - sum[j] - (i-j) * arr[j+1] >= dp[k] + sum[i] - sum[k] - (i-k) * a[k+1]$$

化简得：

$$(dp[k]dp[j]) - (sum[k] - sum[j]) + k \times a[k+1] - j \times a[j+1] \leq i * (a[k+1] - a[j+1])$$

$$G(k,j) = (dp[k]dp[j]) - (sum[k] - sum[j]) + k \times a[k+1] - j \times a[j+1]$$

$$S(k,j) = a[k+1] - a[j+1]$$

则上式化为$G(k,j) \leq i * S(k,j)$ 即 $X(i,\ j) = G(k,j)/S(k,j) <= i$

所以斜率方程：$X(k,j) <= i$

3. 规定队列的维护规则

1) 队首维护:

假设 A,B(A<B)是队首元素,若$X(B,A) <= i$,则 B 比 A 好,删除 A,否则不需维护.

2) 队尾维护:☹队尾元素不改变决策的单调性（不存在极值）

假设 A,B,C (A<B<C)是队尾元素

a.若 X(B,A)<=i,且 X(C,B)<=i,则 C 比 B 好,B 比 A 好

b.若 X(B,A)<=i,且 X(C,B)>i,则 B 比 C 好,B 比 A 好,B 为极大值

c.若 X(B,A)>i,A 比 B 好

4. 延时操作

下考虑每组不少于 K 个元素这个限制。

要解决这个限制，只需延迟加入的时机即可。

若延迟 K-1 个回合加入，有可能使前一组的个数少于 K 个。

若延迟 2*k-1 个回合加入，则不会出现这情况。但此时加入的数应是 i-k+1（假设是第 I 回合）

【题目代码】下面是 AC 的代码：

```
//Name: pku_3709_K-Anonymous Sequence
#include <iostream>
using namespace std;
#define i64 long long
const int maxn = 500005;
int q[maxn], head, tail;
i64 a[maxn], sum[maxn];
i64 dp[maxn];
int n, k;
```

```
i64 G(int k, int j) {return dp[k]-dp[j]-sum[k]+sum[j]+k*a[k+1]-j*a[j+1];}
i64 S(int k, int j) {return a[k+1] - a[j+1]; }
int main() {
    //freopen("in.in", "r", stdin);
    int t, i, j, x, y, z; scanf("%d", &t);
    while(t-- && scanf("%d %d", &n, &k)) {
        for(i = 1; i <= n; i++){
            scanf("%d", a+i);
            sum[i] = sum[i-1] + a[i];
        }
        dp[0] = 0;
        head = 0; q[tail=1] = 0;
        for(i = 1; i <= n; i++) {
            while(head<tail-1 && G(q[head+1],q[head])<=i*S(q[head+1],q[head]))
                head++;
            j = q[head];
            dp[i] = dp[j] + sum[i] - sum[j] - (i-j)*a[j+1];
            if(i >= 2*k-1) q[tail++] = i-k+1;
            for(j = tail-2; j > head; j--) {
                x = q[j-1], y = q[j], z = q[j+1];
                if(!(G(y, x)*S(z, y) < G(z, y)*S(y,x)))
                    q[j] = q[--tail];
                else break;
            }
        }
        printf("%lld\n", dp[n]);
    }
    return 0;
}
```

【小结】这个题目还是比较经典的一个题目，我通过这个题目是学到了很多的东西，不仅仅是学会了处理一类问题。这个题目是斜率优化 dp 里面比较困难的一个。很经典。值得学习，下面的一个就相对比较简单了，但是为了学习，还是应该好好的学习巩固一下。

[HNOI2008]玩具装箱 toy

【问题描述】P 教授要去看奥运，但是他舍不下他的玩具，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，其可以将任意物品变成一堆，再放到一种特殊的一维容器中。P 教授有编号为 1...N 的 N 件玩具，第 i 件玩具经过压缩后变成一维长度为 Ci.为了方便整理，P 教授要求在一个一维容器中的玩具编号是连续的。同时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，形式地说如果将第 i 件玩具到第 j 个玩具放到一个容器中，那么容器的长度将为 $\mathbf{x = j - i + Sigma(Ck)}$ $\mathbf{i} <= K <= j$ 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$\mathbf{(X - L)\hat{}2}$.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.

【问题分析】下面的分析过程延续上面一题的分析过程。

1. 状态转移方程。 这个题目的状态转移方程很好想。

   设dp[i] 表示前 i 将物品分组后的最小花费，那么

$$dp[i] = \min(dp[j] + w[j][i]| , \quad 0 \le j \le i - 1)$$

代价函数

$$w[j][i] = （i - j - 1 + sum[i] - sum[j] - L）^2$$
$$w[j][i] = ((i + sum[i]) - (j + sum[j]) - (1 + L))^2$$

不妨记$s[x] = x + sum[x]$, $C = 1 + L$， 则

$$w[j][i] = (s[i] - s[j] - C)^2$$

$$dp[i] = \min(dp[j] + (s[i] - s[j] - C)^2) | , \quad 0 \le j \le i - 1)$$

由此，我们看到这是一个 O(N^2)的算法，显然不符合 5 万， 1 秒的时限。必须优化。考虑，单调性。

2. 设当前点为 i, 两个决策点为 j， k 且 j < k. 那么有 dp 的非递减序列，知道 $dp[j] < dp[k]$的。假设当前的最优决策为 k，那么有

$$dp[j] + w[j][i] \ge dp[k] + w[k][i]$$

所以有 $w[j][i] \ge w[k][i]$ 由此说明对于当前的点 i 来说，两个决策点 j、k（j<k）, 如果 k 比 j 好，那么在后面的点钟 k 始终比 j 好，即 $w[j][i] > w[k][i]$。 所以，我们可以抛去 j 点不顾，只考虑 k 点，这样就是单调队列解决这个问题的原因。说明了这一点，下面就可以进行判定了！

3. 确定判定最优决策的依据：

由 j < k ，我们知道

$$dp[j] + w[j][i] \ge dp[k] + w[k][i]$$

$$dp[j] + (s[i] - s[j] - C)^2 \ge dp[k] + (s[i] - s[k] - C)^2$$

展开化简得，

$$dp[k] - dp[j] + s[k]^2 - s[j]^2 + 2C \times (s[k] - s[j]) \le s[i] \times 2(s[k] - s[j])$$

令$G(k,j) = dp[k] - dp[j] + s[k]^2 - s[j]^2 + 2 * C * (s[k] - s[j])$

$$S(k,j) = 2(s[k] - s[j])$$

故

$$\frac{(k,j)}{S(k,j)} \le S(k,j) * s[i]$$

若 k, j(K > j) 满足上式，我们就说 k 比 j 优。

至此，判断最优决策的条件已经决定。

4. 规定队列的维护规则 设三个决策点 A < B < C, 当前点为 i,

队首维护 记$X(k,j) = G(k,j)/S(k,j)$,那么若$X(B,A) <= s[i]$ 那么 B 比 A 优，A 出对。否则，B 入队。

队尾维护

a) 若$X(B,A) \le S[i]$ 且 $X(C,B) \le S[i]$ 则 C 比 B 优，B 比 A 优，B 出队。
b) 若$X(B,A) \le S[i]$且 $X(C,B) \ge S[i]$, 则 B 比 C 优，B 比 A 优。维护完毕。
c) 若$X(B,A) \ge S[i]$ 那么直接 B 出对。

【题目代码】：下面是 AC 的代码。

```
//Name: HNOJ_[HNOI2008]玩具装箱 toy
//Author: longxiaozhi
#include <iostream>
using namespace std;
#define SQR(x) ((x)*(x))
#define i64 __int64
const int maxn = 50005;
i64 dp[maxn], sum[maxn];
```

```
int q[maxn], head, tail;
int n, L, C;
i64 G(int k,int j){ return dp[k]-dp[j]+SQR(sum[k])-SQR(sum[j])+2*C*(sum[k]-sum[j]);}
i64 S(int k,int j){ return 2*(sum[k]-sum[j]); }
int main() {
    int i, j, val, x, y, z;
    while(scanf("%d %d", &n, &L) != EOF) {
        C = L + 1;
        for(i = 1; i <= n; i++) {
            scanf("%d", &val);
            sum[i] = sum[i-1] + val;
        }
        for(i = 1; i <= n; i++) sum[i] += i;
        head = 0; q[tail=1] = 0;
        for(i = 1; i <= n; i++) {
            while(head < tail-1 && G(q[head+1], q[head]) <= sum[i]*S(q[head+1],
q[head]))
                head++;
            j = q[head];
            dp[i] = dp[j] + SQR(sum[i] - sum[j] - C);
            q[tail++] = i;
            for(j = tail-2; j > head; j--) {
                x = q[j-1]; y = q[j], z = q[j+1];
                if(!(G(y, x)*S(z,y) < G(z,y)*S(y,x)))
                    q[j] = q[--tail];
                else break;
            }
        }
        printf("%I64d\n", dp[n]);
    }
    return 0;
}
```

Batch Scheduling

【问题概述】编号 1~N(N <= 10000)的 N 个工作，要在一个机器上进行加工，每个工作都有一个加工时间 Tx(1…N) 和一个花费 Fx(1…N) ，现在对工作进行分组，每一组的工作都会有一个共同的结束时间，如果开始的时间为 S，第 i 到第 j 个工作分到一组，那么该组工作的结束时间为 O = S + Ti + Ti+1…… + Tj. 每个工作的花费为 O * Fx. 每组工作的加工互不影响，求加工的最少时间。

【问题分析】对概率优化 DP 的题目也模仿着做了几题了，该有点自己的东西了，要不然对不起自己的时间。

1. 这个题目采用逆推的方式，

设 dp[i]表示加工第 i…N 工作完成所需要的时间， sumT[i]表示第 i~N 工作时间的总和，sumF[i]表示工作花费第 i~N 的总和。则，状态转移方程为，

$$\mathrm{dp[i]} = \min\{\, \mathrm{dp[j]} + (\mathrm{s} + \mathrm{sumT[i]} - \mathrm{sumT[j]}) * \mathrm{sumF[i]} \mid \mathrm{i} < j \le N+1 \,\}$$

（dp[i] 相当于在 dp[j]中的每个工作的即出上有增加了 s + sumT[i] – sumT[j]的工作时间的花费）

其中代价函数表示$\mathrm{w[j][i]} = \mathrm{s} + (\mathrm{sumT[i]} - \mathrm{sumT[j]}) \times \mathrm{sumF[i]}$． 表示有状态 j 转移到状态 i 的代价花费，取所有花费中最小的。显然这个时间复杂度为$O(n^2)$，需要优化。

2. 能优化的地方，就是决策 j 的选择，即，选择一个最小的决策 j。最有决策显然具有单调性，也就是说如果 j > k ,且 k 为一个最优的决策，那么，由 dp[j] < dp[k]得到。

$$dp[j] + w[j][i] \geq dp[k] + w[k][i];$$

所以，$w[j][i] \geq w[k][i]$. 也就是说 k 之后的（1..k-1）的任何以一个状态的都会有，k 转移到的 比 j 转移的优秀。将上式的代价函数带入得到。

$$dp[j] + (s + sumT[i] - sumT[j]) \times sumF[i] \geq dp[k] + (s + sumT[i] - sumT[k]) * sumF[i]$$

继续化简，得

$$dp[k] - dp[j] \leq (sumT[k] - sumT[j]) * sumF[i]$$

记 $G(k,j) = dp[k] - dp[j]$, $S(k,j) = sumT[k] - sumT[j], X(k,j) = G(k,j)/S(k,j)$， 则

$$G(k,j) \leq S(k,j) * sumF[i]$$

$$X(k,j) \leq sumF[i]$$

也就是说，满足 k 比 j 优的一个必要条件是必须满足上式 X(k, j) <= sumF[i]. 也就是说，如果我们用单调队列来记录决策，那么，对于队首的两个决策的选择必须满足上面的式子，如果记队列第一个元素为 a， 第二个元素为 b, (注意这里是 a > b, 不满足上面的 j > k, 所以要注意符号的问题)，那么，如果 X(b, a) <= sumF[i]. 那么 b 将会比 a 优，我们要删除 a，这样知道队首的两元素满足 X(b, a) >= sumF[i]， 记队首为首选。

3. 考虑队尾，同上，记队尾元素为 a，队尾第二个元素为 b，那么

如果$X(i, a) \leq sumF[i]$ 且 $X(a,b) > sumF[i]$， 即 i 比 a 优，b 比 a 优,,那么 a 为级差值（极小值），那么没必要保留 a（因为下次肯定不会选择 a，b 优于 a，i 优于 a，利用上面的一个结论（对于当前的点 i 来说，两个决策点 j、k（j<k）,如果 k 比 j 好，那么在后面的点钟 k 始终比 j 好，即 $w[j][i] > w[k][i]$。那么没必要包留 j）

【题目代码】：下面是 AC 的代码，还是很高效的。

9	7343486	longxiaozhi	300K	0MS	C++	1154B	2010-08-04 08:37:19

```
////Name: pku_1180_Batch Scheduling
#include <iostream>
using namespace std;
#define i64  long long
const int maxn = 10001;
i64 dp[maxn];
int sumF[maxn], sumT[maxn];
int n, st, q[maxn], head, tail;
i64 G(int k, int j) { return dp[k] - dp[j]; }
i64 S(int k, int j) { return sumT[k] - sumT[j];}
int main(){
      freopen("in.in", "r", stdin);
      scanf("%d %d", &n, &st);
      for(int i = 1; i <= n; i++)
            scanf("%d %d", &sumT[i], &sumF[i]);
      for(int i = n-1; i >= 1; i--) {
            sumF[i] += sumF[i + 1];
            sumT[i] += sumT[i + 1];
      }
      head = tail = 0;
      for(int i = n; i >= 1; i--) {
        while(head < tail && G(q[head + 1],q[head]) <=S(q[head + 1],q[head]) * sumF[i])
```

```
                    head ++;
            int j = q[head];
            dp[i] = (st + sumT[i] - sumT[j]) * sumF[i] + dp[j];
            while(head < tail && G(i,q[tail])*S(q[tail],q[tail-1])<=
                  S(i ,q[tail]) * G(q[tail], q[tail - 1]))
                        tail--;
            q[++tail] = i;
      }
      printf("%I64d\n", dp[1]);
      return(0);
}
```

Picnic Cows

【题目概述】题目大意和解法同 K-Anonymous Sequence。这里主要说明优化的重要性。

14	longxiaozhi	906MS	9620K	1256B	C++	2010-08-04 12:18:10

这个题目主要注意数据范围，可能超 64 位。题目代码和分析略，详见上。