2.保龄球

【算法分析】

由于球没有先后顺序，不妨设球打的位置一定是从左到右的。

记 sum[i]表示[1,i]的球瓶分数和。记 f[i][j]表示第 j 个球打在[i-w+1,i]时的最大值，考虑第 j-1 个球打到的位置的情况。

① 第 j-1 个球打到区间的右端点 x 满足 x≤i-w：这时前 j-1 个球打的具体位置 x 不影响第 j 个球的得分，它的得分只需要记一个最大值$mx=\max\limits_{1\le k\le i-w}\{f[k][j-1]\}$，那么第 j 个球打到的位置[i-w+1,i]上的球瓶是完整的，它的得分为 sum[i]-sum[i-w]，此时 f[i][j]=mx+sum[i]-sum[i-w]。

② 第 j-1 个球打到的区间右端点 x 满足 i-w+1≤x≤i-1：

这时前 j-1 个球的得分为 f[x][j-1]。第 j 个球打到的区间[i-w+1,i]的得分受前 j-1 个球的影响（[i-w+1,x]的球瓶已经被第 j-1 个球打掉了，这段的得分为 0），得分为 sum[i]-sum[x]，此时 f[i][j]=max(f[x][j-1]+sum[i]-sum[x])。

但是 O(NKW)时间复杂度无法承受，考虑怎样优化转移。

观察上面的转移方程式，发现 sum[i]是只和状态有关的值，枚举 x 要得到的是 max(f[x][j-1]-sum[x])，因此可以用单调队列进行优化。显然对于队列中的两个决策点 $x_1<x_2$，若有 f[x_1][j-1]-sum[x_1]<f[x_2][j-1]-sum[x_2]，则 x_1 是不可能被用来更新答案的，可以在加入 x_2 时就让 x_1 从队尾出队。因此这个单调队列的单调性就在于 f[x][j-1]-sum[x]的值随着 x 的递增而递减。

记一个队列 h 表示当前符合要求的 x 的值，每次将队头不符合的答案出队，此时最优转移就在队头，直接从队头转移即可。然后不断将队尾不如当前下标 i 的值优的答案移出队列，将当前下标 i 的值从队尾入队。

③ 最终 f[i][j]的值为 1、2 两种情况的较大值。

解决了“被击倒的球瓶留下的空白位置”，对于“原先球瓶左边和右边的空白位置”，可以直接在“原先球瓶的左右”各加上 w-1 个分数为 0 的球瓶，将球瓶的长度扩展到 n+2*(w-1)，就可以避免判断各种特殊情况。

初始化：f[0][0]=0，其余状态 f 赋为-INF。

答案：由于没有要求每个球都用完，所以每个状态 f[i][j]都有可能成为最大值（有些球瓶的分数可能为负）。ans=max(f[i][j])。

时间复杂度为 O(NK)。

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
```

```
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

const int N = 505, M = 100205;
const int Minn = -1e8;
int TAT, n, K, W, ans, f[N][M], h[M], sum[M], a[M];

template <class T>
inline T Max(T x, T y) {return x > y ? x : y;}
template <class T>
inline void CkMax(T &x, T y) {if (x < y) x = y;}

int main()
{
    read(TAT);
    while (TAT--)
    {
        memset(a, 0, sizeof(a));
        read(n); read(K); read(W);
        for (int i = 1; i <= n; ++i)
            read(a[i + W - 1]);
        n += W + W - 2; ans = 0;
        //在原先球瓶左右加上分数为 0 的球瓶
        for (int i = 1; i <= n; ++i)
            sum[i] = sum[i - 1] + a[i];
        for (int i = 0; i <= K; ++i)
            for (int j = 0; j <= n; ++j)
                f[i][j] = Minn;
        f[0][0] = 0;  //初始化
        for (int i = 1; i <= K; ++i)
        {
            int t = 1, w = 0, res = Minn;
            if (f[i - 1][i - 1] > Minn) h[++w] = i - 1;
            for (int j = i; j <= n; ++j)
```

```
        {   //单调队列优化 DP
            while (t <= w && j - h[t] >= W) ++t;
            if (j - W >= i - 1)  //保证转移的状态合法
            {
                CkMax(res, f[i - 1][j - W]);
                f[i][j] = res + sum[j] - sum[j - W];
            }
            if  (t  <=  w)  CkMax(f[i][j],  f[i-1][h[t]]  +  sum[j]  -
sum[h[t]]);
           while (t <= w && f[i-1][h[w]]+sum[j]-sum[h[w]] <= f[i-1][j])
                --w;
            h[++w] = j;
        }
    }
    for (int i = 1; i <= K; ++i)
        for (int j = 1; j <= n; ++j)
            CkMax(ans, f[i][j]);
    printf("%d\n", ans);
  }
  return 0;
}
```

3.爬山

【算法分析】

首先，我们求出 le[i]和 ri[i]分别表示(x_i,y_i)向左和向右能看到的最高山顶。

容易得出，le[i]就是按 x 坐标排序后，前 i 个点构成的**上凸壳**（相邻点斜率递减）的倒数第二个（x 坐标次大）点，ri[i]为后 i 个点构成的**上凸壳**的第二个（x 坐标次小）点。

从左向右和从右向左分别用单调栈维护上凸壳即可求出 le[i]和 ri[i]。

这样，我们就能求得 to[i]表示在山顶 i 能看到的最高山顶。如果 i 为最高山顶并且 x 坐标最大则 to[i]为 0。

接下来，我们还要求出：

（1）pre[i]，表示 i 左边满足 to[j]的 y 坐标大于 to[i]的 y 坐标的最大的 j（相同情况下判断 x 坐标大小），如果不存在 j 则 pre[i]=to[i]。

（2）suf[i]，表示 i 右边满足 to[j]的 y 坐标大于 to[i]的 y 坐标的最小的 j（相同情况下判断 x 坐标大小），如果不存在 j 则 suf[i]=to[i]。

同样可以维护一个 to[i]的 y 坐标随 i 递减（递增）的单调栈，每次弹出 to[j]小于 to[i]的元素，再取出栈顶得出 pre[i],suf[i]。

容易得出，如果 to[i]<i，那么在 i 向左走的过程中，走到 pre[i]时可以看到更高的山顶或者已经到达了最高山顶。同样地，如果 to[i]>i，那么在 i 向右走的过程中，走到 suf[i]时可以看到更高的山顶或者已经到达了最高山顶。故：

（1）如果 to[i]<i，将 pre[i]作为 i 的父亲节点，边(pre[i],i)权为 i-pre[i]，表示从 i 走到 pre[i]的距离，中间不会改变方向。

（2）如果 to[i]>i，将 suf[i]作为 i 的父亲节点，边(suf[i],i)权为 suf[i]-i，表示从 i 走到 suf[i]的距离，中间不会改变方向。

建好树之后，根到 i 节点的距离就是山顶 i 的答案。

时间复杂度为 O(n)。

【参考程序】

```
#include <bits/stdc++.h>
#define For(i, a, b) for (i = a; i <= b; i++)
#define Rof(i, a, b) for (i = a; i >= b; i--)
#define Edge(u) for (int e = adj[u], v = go[e]; e; e = nxt[e], v = go[e])
using namespace std;

inline int read()
{
    int res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    return bo ? ~res + 1 : res;
}

typedef long long ll;
const int N = 1e6 + 5;
int X[N], Y[N], stk[N], le[N], ri[N], to[N], cnt[N], top, n, rt,
ecnt, nxt[N], adj[N], go[N];
ll dis[N];

bool slope(int i, int j, int k)   //计算斜率维护凸壳性质
{
    return 1ll * (Y[i] - Y[j]) * (X[j] - X[k]) <
        1ll * (Y[j] - Y[k]) * (X[i] - X[j]);
}

void add_edge(int u, int v)
{   //将 x 作为 y 的父亲节点，边(x,y)权为|x-y|
    nxt[++ecnt] = adj[u]; adj[u] = ecnt; go[ecnt] = v;
    cnt[v]++;
```

```
}

inline void dfs(int u)
{   //求每个点到根的距离
    Edge(u)
        dis[v] = dis[u] + abs(u - v), dfs(v);
}

bool comp(int a, int b)
{   //比较两个山顶的大小
    if (a == b) return 1;
    if (X[a] > X[b] && Y[a] == Y[b] || Y[a] > Y[b]) return 1;
    else return 0;
}

int main()
{
    int i;
    n = read();
    For (i, 1, n) X[i] = read(), Y[i] = read();
    Y[0] = -1;
    stk[top = 1] = 1; le[1] = 0;
    For (i, 2, n)
    {
        while (top > 1 && slope(stk[top - 1], stk[top], i)) top--;
        //用单调栈维护上凸壳
        if (comp(stk[top], i)) le[i] = stk[top];
        stk[++top] = i;
    }
    stk[top = 1] = n; ri[n] = 0;
    Rof (i, n - 1, 1)
    {
        while (top > 1 && slope(i, stk[top], stk[top - 1])) top--;
        if (comp(stk[top], i)) ri[i] = stk[top];
        stk[++top] = i;
    }
    For (i, 1, n) to[i] = Y[le[i]] <= Y[ri[i]] ? ri[i] : le[i];
    stk[top = 1] = 1;
```

```
    For (i, 2, n)
    {
        while (top && comp(to[i], to[stk[top]])) --top; //用单调栈维护，求出le[i]
        if (to[i] && to[i] < i) add_edge(top ? stk[top] : to[i], i);
        stk[++top] = i;
    }
    stk[top = 1] = n;
    Rof (i, n - 1, 1)
    {
        while (top && comp(to[i], to[stk[top]])) top--; //用单调栈维护，求出ri[i]
        if (to[i] && to[i] > i) add_edge(top ? stk[top] : to[i], i);
        stk[++top] = i;
    }
    For (i, 1, n)
        if (!cnt[i])  //入度为0的点为建出树的根
        {
            rt = i; break;
        }
    dfs(rt);
    For (i, 1, n) printf("%lld\n", dis[i]);
    return 0;
}
```

4.分班

【算法分析】

显然分在同一个班的小朋友 $g[i]$ 相同，并且 $\sum_{i=l}^{r}(X[i]-Average)^2$ 可以预处理前缀和 $O(1)$ 计算，记作 $s[i]$，则把第 $l \sim r$ 个小朋友分在第 k 个班的代价为 $(s[r]-s[l-1])\times G[k]$ 。

于是考虑设计状态 $f[k][i]$ 表示处理到第 k 个班，第 i 个小朋友一定被分在第 k 个班末尾的最小代价，则转移为 $f[k][i]=\min\limits_{i-B\le j\le i-A}\{f[k-1][j]+(s[i]-s[j])\times G[k]\}$，表示枚举分在第 $k-1$ 个班末尾的小朋友 j ，将第 $j+1\sim i$ 个小朋友分入第 k 个班。

枚举状态复杂度为 $O(NM)$，转移复杂度为 $O(M)$，总复杂度为 $O(NM^2)$，难以通过。

考虑怎样优化转移，先把转移式子中不含 j 的项移到 $\min$ 外边：

$$f[k][i]=\min_{i-B\le j\le i-A}\{f[k-1][j]-s[j]\times G[k]\}+s[i]\times G[k]$$

转移的 j 显然是一个区间，并且当 i 右移到 $i+1$ 时，能够转移的 j 的区间的左右端点也随之向右移。

因此维护一个 j 和 $f[k-1][j]-s[j]\times G[k]$（记作 $Q[j]$）递增的单调队列 q。

对于每个 $f[k][i]$，每次把队首元素 $q[head]$ (满足 $q[head]<i-B$)删除，并把 $i-A$ 加入队列，如果队尾元素 $q[tail]$ 满足 $Q[q[tail]]\ge Q[i-A]$，显然从 $q[tail]$ 转移不如从 $i-A$ 转移优，因此可以删去 $q[tail]$，则每次从队首元素转移最优。

单调队列中每个元素至多被插入和删除一次，因此枚举 i 并转移的复杂度为 $O(M)$，总复杂度为 $O(NM)$。

现在我们已经知道了最小代价，考虑怎样输出方案，先找到最小的分班数，另外对每个 $f[k][i]$ 记录最优的转移 j，如果我们在往队列中加入元素时已经把相同代价的队尾元素删去，那么最优的转移 j 一定能保证每次分班人数尽量少。

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

template <class T>
inline void put(T x)
{
    if (x > 9) put(x / 10);
    putchar(x % 10 + 48);
}
```

```
typedef long long ll;
const ll Maxn = 1000000000000000000ll;
const int N = 1e4 + 5;
int m, n, A, B, TAT, sum, h[N], fr[205][N];
int x[N], g[N];
ll s[N], f[205][N];
template <class T>
inline void CkMin(T &x, T y) {if (x > y) x = y;}

int main()
{
    read(TAT);
    while (TAT--)
    {
        read(m); read(n); read(A); read(B);
        sum = 0;
        for (int i = 1; i <= m; ++i)
            read(x[i]), sum += x[i];
        for (int i = 1; i <= n; ++i)
            read(g[i]);
        sum /= m;
        for (int i = 1; i <= m; ++i)
            s[i] = s[i - 1] + 1ll * (sum - x[i]) * (sum - x[i]);
        for (int k = 0; k <= n; ++k)
            for (int i = 0; i <= m; ++i)
                f[k][i] = Maxn;
        f[0][0] = 0;  //初始化
        for (int k = 1; k <= n; ++k)
        {
            int t = 1, w = 0;
            for (int i = k; i <= m; ++i)
            {   //注意状态中要保证能分成 k 个班，i 要从 k 开始循环
                while (t <= w && i - h[t] > B) ++t;
                if (i - A >= k - 1 && f[k - 1][i - A] < Maxn)
                {   //同样插入 i-A 时也要保证能分成 k-1 个班
                    while (t <= w && f[k - 1][h[w]] - s[h[w]] * g[k]
                              >= f[k - 1][i - A] - s[i - A] * g[k]) --
w;
                    h[++w] = i - A;
                }
                if (t <= w)
                  f[k][i]=f[k-1][h[t]]+(s[i]-s[h[t]])*g[k],fr[k][i]=h[t];
            }
```

```
        }
        ll ans = Maxn;
        for (int i = 1; i <= n; ++i)
            CkMin(ans, f[i][m]);
        for (int i = 1; i <= n; ++i)
            if (f[i][m] == ans)
            {
                if (ans < 0) ans = -ans, putchar('-');
                put(ans); putchar(' ');
                put(i); putchar(' ');
                put(m - fr[i][m]), putchar('\n');
                break;
            }
    }
    return 0;
}
```