不难发现对于将题意抽象为几何模型就是在一个二维坐标系内，每个苹果有一个坐标（t，x），同时你有一个大小为$S \times L$的矩形，题目要求输出这个矩形最多能罩住几个苹果。

其实对数据结构有更深入研究的同学一眼就可以看出这是一道线段树扫描线的模板题，不过不会扫描线也没关系，正常线段树也可以写，此处讲述下正常线段树的写法。

对横坐标t进行排序后，我们可以使用一个滑动窗口来维护当前的矩形的长d，对不在当前矩形长内的点进行删除，对于矩形的宽的维护，此时就可以使用线段树了，设线段树的端点i携带的信息为纵坐标在$i, i+1 ... i+s-1$范围内的苹果的数量最大值。不难发现通过滑动窗口+线段树我们能通过$o(nlogn)$的时间来维护这个矩形内的苹果数量。

```
struct SegmentTree{
    struct node{
        int l, r;
        LL sum, add;
    };
    vector<int> a;
    vector<node> tr;
    SegmentTree(int n) : a(n + 1), tr(4 * n + 1) {}
    void pull(int u){
        tr[u].sum = max(tr[2 * u].sum,tr[2 * u + 1].sum);
    }
    void apply(int v, int u){
        tr[v].sum += tr[u].add;
        tr[v].add += tr[u].add;
    }
    void push(int u){
        apply(2 * u, u);
        apply(2 * u + 1, u);
        tr[u].add = 0;
    }
    void build(int u, int l, int r){
        tr[u] = {l, r};
        if (l == r){
            tr[u].sum = a[r];
            tr[u].add = 0;
            return;
        }
        int mid = (l + r) >> 1;
        build(2 * u, l, mid);
        build(2 * u + 1, mid + 1, r);
        pull(u);
    }
    void modify(int u, int l, int r, LL v){
        if (tr[u].l >= l && tr[u].r <= r){
            tr[u].sum = tr[u].sum + v;
            tr[u].add = tr[u].add + v;
            return;
        }
        push(u);
        int mid = (tr[u].l + tr[u].r) >> 1;
        if (l <= mid) modify(u << 1, l, r, v);
        if (r > mid) modify(u << 1 | 1, l, r, v);
        pull(u);
    }
    void modify(int l, int r, LL v){
        return modify(1, l, r, v);
    }
    LL query(int u, int l, int r){
        if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
        push(u);
        int mid = (tr[u].l + tr[u].r) >> 1;
```

```
        LL sum = 0;
        if (l <= mid) sum = max(sum,query(u << 1, l, r));
        if (r > mid) sum = max(sum,query(u << 1 | 1, l, r));
        return sum;
    }
    LL query(int l, int r){
        return query(1, l, r);
    }
};
struct node{
    LL t,x;
};

int cmp(node x,node y){
    return x.t < y.t;
}

void solve(){
    LL n,d,w;
    cin >> n >> d >> w;
    SegmentTree seg(N);
    seg.build(1,1,N);
    vector<node> a(n + 1);
    for(int i = 1;i <= n;i ++){
        cin >> a[i].t >> a[i].x;
    }
    sort(a.begin() + 1,a.end(),cmp);
    deque<int> q;
    LL ans = 0;
    for(int i = 1;i <= n;i ++){
        while(q.size() != 0 && a[q.front()].t < a[i].t + 1 - d){
            seg.modify(max(1LL,a[q.front()].x - w + 1),a[q.front()].x,-1);
            q.pop_front();
        }
        q.push_back(i);
        seg.modify(max(1LL,a[i].x - w + 1),a[i].x,1);
        ans = max(ans,seg.query(1,N));
    }
    cout << ans << endl;
}
```