区间和

假定有一个无限长的数轴，数轴上每个坐标上的数都是 0。

现在，我们首先进行 n 次操作，每次操作将某一位置 x 上的数加 c。

接下来，进行 m 次询问，每个询问包含两个整数 l 和 r，你需要求出在区间 $[l, r]$ 之间的所有数的和。

输入格式

第一行包含两个整数 n 和 m。

接下来 n 行，每行包含两个整数 x 和 c。

再接下来 m 行，每行包含两个整数 l 和 r。

输出格式

共 m 行，每行输出一个询问中所求的区间内数字和。

数据范围

$-10^9 \leq x \leq 10^9$,
$1 \leq n, m \leq 10^5$,
$-10^9 \leq l \leq r \leq 10^9$,
$-10000 \leq c \leq 10000$

输入样例：

```
3 3
1 2
3 6
7 5
1 3
4 6
7 8
```

输出样例：

```
8
0
5
```

(2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作

3. 离散化

```
vector<int> alls; // 存储所有待离散化的值
sort(alls.begin(), alls.end()); // 将所有值排序
alls.erase(unique(alls.begin(), alls.end()), alls.end());// 去掉重复元素

// 二分求出x对应的离散化的值
int find(int x) // 找到第一个大于等于x的位置
{
    int l = 0, r = alls.size() - 1;
    while (l < r)
    {
        int mid = l + r >> 1;
        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1; // 映射到1, 2, ...n
}
```

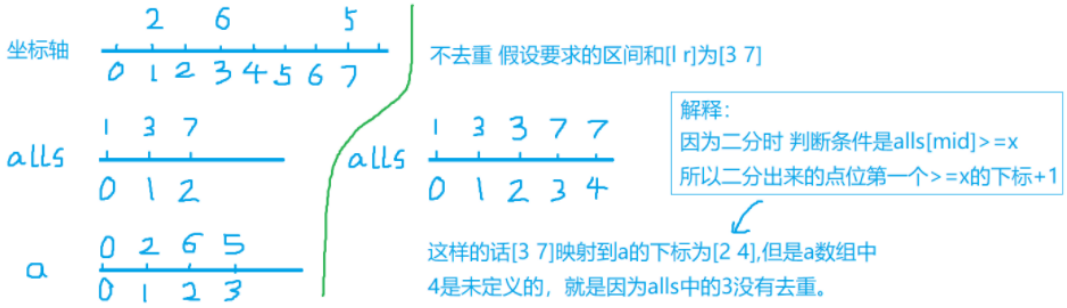
离散化的本质，是映射，将间隔很大的点，映射到相邻的数组元素中。减少对空间的需求，也减少计算量。

2.为什么要排序和去重?

首先要明确 `find` 函数的功能，输入一个离散数组的位置（映射前的位置） `x` 返回连续数组的位置 `+1`（映射后的位置 `+1`）。 `+1` 的目的是为了求区间和时少一步下标为 `0` 的判断。

排序很好理解，因为在 `find` 函数中是使用了二分来查找 `x` 在 `alls` 中的下标 `+1`，想要使用二分 `alls` 就必须具有某种性质这里就可以找一个最简单的办法使他单调（但是y总说过二分!=单调性）。

去重看我下面画的图，图中的例子为本题的样例。（前半部分为在 `x` 处加 `c` 后 `alls` 与 `a` 中的内容，后半部分为假设不去重求[3 7]的区间和）




```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

const int N=300010;//坐标x的数量上限为1e5，两个坐标l,r的数量上限也为1e5,所以加起来为3*1e5;
typedef pair<int,int> PII;//pair<int,int>类似于结构体的简写版，typedef定义了一个新的类型PII(跟结构体定义了一个结构体类型然后使用相似)

int a[N],s[N];

vector<int> alls;
vector<PII> add,query;
//使用二分查找x所在的位置，此时是alls(x,l,r)排好序的,返回的坐标也会是按照x的大小所给出的;
int find(int x)
{
    int l=0,r=alls.size()-1;
    while(l<r)
    {
        int mid=(l+r)/2;
        if(alls[mid]>=x) r=mid;
        else l=mid+1;
    }
    return r+1;//因为后续要使用前缀和，所以返回的坐标要加上1;
}

int main()
{
    int n,m;cin>>n>>m;
    //分别将要操作的四组数据记录在add和query中，将l，r，x的坐标值保存在alls中;
    for(int i=0;i<n;i++)
    {
        int x,c;
        cin>>x>>c;
```

```
        add.push_back({x,c});
        alls.push_back(x);
    }
    for(int i=0;i<m;i++)
    {
        int l,r;
        cin>>l>>r;
        query.push_back({l,r});
        alls.push_back(l);
        alls.push_back(r);
    }
    //将alls进行排序，并将重复的操作删除掉(如进行了两次在x的增值操作，应该去掉一个x保持平衡)；
    sort(alls.begin(),alls.end());
    alls.erase(unique(alls.begin(),alls.end()),alls.end());
    //一个迭代器从1开始直到末尾结束，itdm.first是x，second是r(在上方循环中可知)；
    for(auto itdm : add)
    {
        int x;
        x=find(itdm.first);
        a[x]+=itdm.second;
    }
    //只循环x是因为x的坐标加上了值，而l，r可能有(l或r与x的值相同)，且定义全局变量数组，其余值默认为0，故可以方便计算；
    for(int i=1;i<=alls.size();i++) s[i]=s[i-1]+a[i];

    for(auto itdm : query)
    {
        int l,r;
        l=find(itdm.first);//找出l，r在a中的坐标
        r=find(itdm.second);
        printf("%d\n",s[r]-s[l-1]);//前缀和上方已计算，所以可直接输出，记得加上换行符!
    }
    return 0;
}
```