算法基础（三）：位运算，离散化，区间合并

位运算

基本思想

顾名思义

基本运用

n的二进制表示中第k位是什么

1. 先把第k位移到最后一位，`n>>k`
2. 再看个位是多少`n&1`

```
#include<iostream>
using namespace std;
int main(){
    int n  = 10;
    //将n以二进制输出
    for(int k = 3; k >= 0; k--) cout << ((n >> k) &1)
        return 0;

}
```

`lowbit`操作

`lowbit`返回的是一个二进制数，返回`x`最后一位`1`的位置, 比如`x = 100010`则返回`10`， `x = 100.....1000`返回`1000`

```
n = x &(-x);//x&((~x)+1)
```

代码实现

801. 二进制中1的个数 - AcWing题库

```
#include<iostream>
using namespace std;

const int N = 100010;
int a[N];


int main(){
    int n;
    cin >> n;
    for(int i = 0; i < n; i++) cin >> a[i];

    for(int i = 0; i < n; i++){
        int count = 0;
        while(a[i]){
        //每个数不停地与自己的lowbit进行异或，除去最后一位1
            a[i] = a[i]^((a[i])&(-a[i]));
            count++;
        }
        cout << count << " ";
    }
    return 0;
}
```

离散化

基本思想

条件:

值域很大比如`0~10^9`，但是这些数的个数只有不超过`10^5`个

举一个例子:

数组`a[] = {1, 3, 100, 2000, 500000}`将其中的每个数映射到`0, 1, 2, 3, 4`这个过程就是离散化的过程，有点像哈希

需要解决的问题:

1. `a[]`中可能会出现重复的元素，我们怎么进行去重

    1. 用 c++ 的一个库函数进行去重 `alls.erase(unique(alls.begin(), alls.end()), alls.end())`
2. 我们怎么算出离散化的值，具体来说，是找到 `a[]` 中的某个值对应的下标
    1. 用二分的方法，也就是二分查找

代码实现

```
vector<int> all; //存储所有待离散化的值
sort(alls.begin(), alls.end());//将所有的值进行排序
alls.erase(unique(alls.begin(), alls.end()), alls.end())//将这些元素去重
//unique()这个函数的作用就是将这个数组中的所有重复元素移动到最后，并返回数组中不重复元素的尾端点，erase()函数的作用是去掉这些重复的元素

//这里使用二分的方法对这些值进行离散化，也就是将这些值与他们的下标对应起来
//这里的二分模板找的是右边性质的左端点，mid = l + r >> 1
int find(int x){
    int l = 0, r = alls.size() - 1;
    while(l < r){
        int mid = l + r >> 1;
        if(a[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1;//返回什么值根据题目而定，返回r+1说明是把值映射到1.....n，不是从0开始
}
```

例题：

802. 区间和 - AcWing题库

这个题目的暴力解法是开一个足够大的数组，然后对每个插入操作将对应的数轴上的下标对应的值加一下，然后对一个区间里的数暴力加起来求和，但是存在两个问题：

1. 由于数轴是 `10^9` 级别，并不能开到这么大的数组，但是插入以及查询都是 `10^5` 级别，所以我们可以将每个插入和查询离散化到另外一个数组 `a[]` 上，`a[]` 的下标是插入和查询对应数轴下标离散化的值，这样对数轴下标 `i` 插入 `c` 的时候，我们只需要对 `a[(i离散化的值)]` 插入 `c` 即可，查询的时候因为有序的离散化，并且原数轴的其他的下标对应的值都为 `0`，所以等价于只需要求 `a[(查询的数轴左端点的下标离散化的值)] ~ a[(查询的数轴右端点的下标离散化的值)]` 的和即可
2. 暴力求解的时候时间复杂度过高，我们采用前缀和的方式求解

代码实现：

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

//用一个pair数据结构来存储插入以及查询
typedef pair<int, int> PII;
//这里我们需要离散化的对象其实是数轴的下标，其值域是10^9
//我们的插入操作只操作了10^5个下标，所以数的个数是10^5级别，考虑使用离散化
```

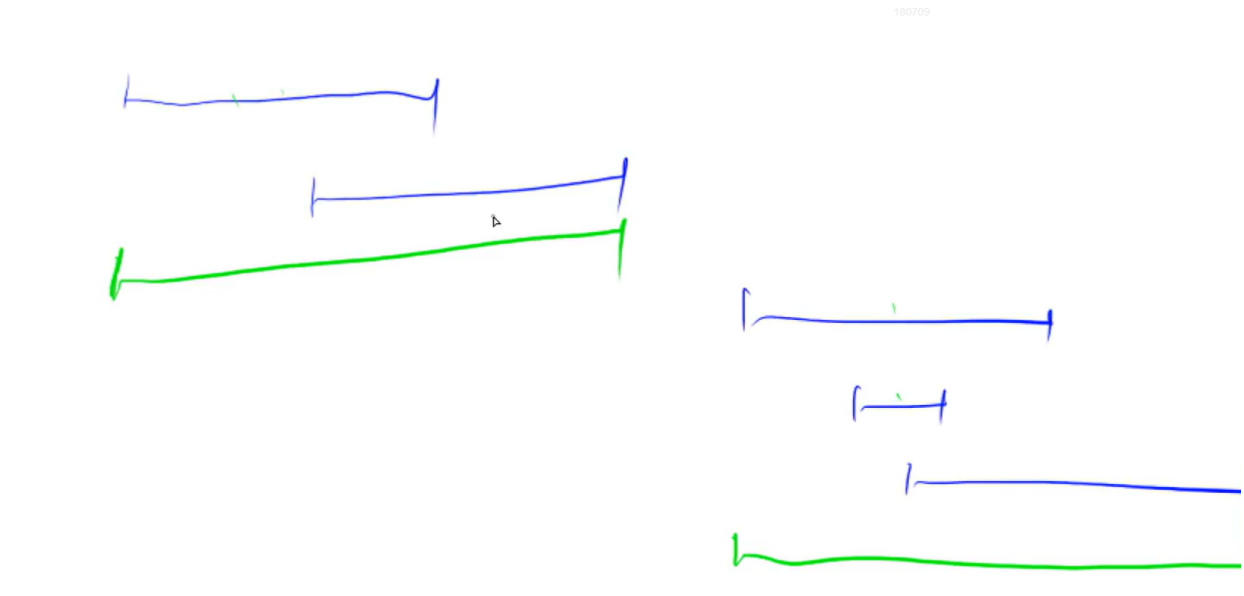
```cpp
//我们插入了10^5个下标，但同时有10^5个查询操作，这些查询的两个端点也需要离散化
//所以开的数组是3*100010级别
const int N = 300010;
int n, m;
//我们求和的时候用前缀和的方式求和，s[N]是a[N]的前缀和
int a[N], s[N];
//用alls来存放每一个插入的数以及查询操作的端点
vector<int> alls;
//pair数据结构来存储插入以及查询
vector<PII> add, query;

//用二分的方式进行离散化
int find(int x){
    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 l + 1;
}


int main(){
    cin >> n >> m;
    for(int i = 0; i < n; i++){
        int x, c;
        cin >> x >> c;
        //读取插入的每一个数轴上的下标以及插入的值
        add.push_back({x, c});

        //将数轴的下标作为值存入alls这个数组中
        alls.push_back(x);
    }


    for(int i = 0; i < m; i++){
        int l, r;
        cin >> l >> r;
        //读取每一个查询操作的端点
        query.push_back({l, r});
        //将查询操作的端点（同样是数轴的下标）存入alls的数组中，统一进行离散化
        alls.push_back(l);
        alls.push_back(r);
    }

    //对数组进行排序去重操作
    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());

    //将alls中的数值进行离散化，并利用类似hash的策略将离散化对应的数组的值加上插入的数
```

```
    for(auto item:add){
        int x = find(item.first);
        a[x] += item.second;
    }

    //对a[]求前缀和
    for(int i = 1; i <= alls.size(); i++) s[i] = s[i-1] + a[i];

    //得到结果
    for(auto item:query){
        int l = find(item.first), r = find(item.second);
        cout << s[r] - s[l - 1] <<endl;

    }


}
```

区间合并

基本思想

快速地对有交集的区间进行合并，比如：

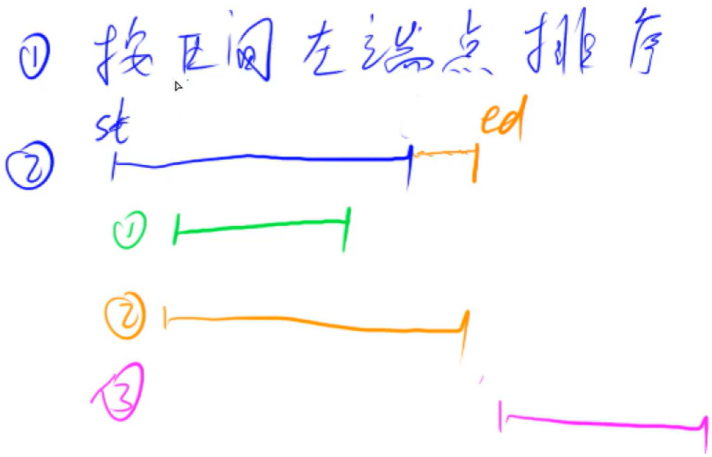


将五个蓝颜色的区间合并成两个绿颜色的区间

算法：

1. 按区间进行排序
2. 从前往后扫描区间，对当前维护的区间有三种情况如下：
    1. 下一个区间在当前维护的区间里面，此时区间不改动
    2. 下一个区间与当前的区间有交叠，更新右端点

3. 下一个区间与当前维护的区间没有关系，则将当前维护的区间更新为下一个区间



代码实现

803. 区间合并 - AcWing题库

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

using namespace std;

typedef pair<int, int> PII;

const int N = 100010;

int n;
//用pair来存储一个区间
vector<PII> segs;

void merge(vector<PII> &segs){
    vector<PII> res;
    //先对区间进行排序，对pair型数据进行排序是先排第一个元素，再排第二个元素
    sort(segs.begin(), segs.end());

    //st与ed表示我们当前维护的区间，最开始的时候设置为负无穷
    int st = -2e9, ed = -2e9;


    //遍历排序好的区间
    for(auto seg : segs){
        //这是表示当前维护的区间与遍历的区间无法合并，没有交集
        //则此时需要将当前维护的区间加入到答案中，并切换为遍历的区间
        if(ed < seg.first){

            //if这里的作用表示的是我们维护区间是从负无穷开始的
            //遍历的区间显然与最开始的没有交集
            //但是我们不能将这个负无穷到负无穷区间加入到结果中
```

```
            if(ed != -2e9) res.push_back({st, ed});

            //更新区间
            st = seg.first, ed = seg.second;
        }else{
            //有交集的情况
            ed = max(ed, seg.second);
        }
    }

    //退出for循环的时候当前维护的区间并没有加入到答案中
    //所以这里需要再加入答案
    //if这里的作用是若题目没有输入任何区间，我们得避免把负无穷到负无穷这个开始区间加入
    if(st != -2e9) res.push_back({st, ed});
    //cout << res.size() <<endl;;
    segs = res;
}

int main(){
    int n;
    int l, r;
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> l >> r;
        segs.push_back({l, r});
    }

    merge(segs);

    cout << segs.size() << endl;

    return 0;
}
```