> 本文转载自我在洛谷上P10903的题解。

前置知识：前缀和与差分

前缀和

简单而言，给定数组 $a_1,a_2,a_3,\cdots,a_n$，现在想要**快速求出** $a_l,a_{l+1},a_{l+2},\cdots,a_r$ 的和。

- 朴素算法遍历 a_l 至 a_r，时间复杂度$\mathcal{O}(r-l+1)$。
- 使用前缀和算法，时间复杂度 $\mathcal{O}(1)$。
  记数组 $b_1,b_2,b_3,\cdots,b_n$ 使得 $b_i=b_{i-1}+a_i$，即 $b_i=a_i+a_2+a_3+\cdots+a_i$。
  这样 b_r-b_{l-1} 即为区间和。
  更多相关内容见此。

差分

简单而言，给定数组 $a_1,a_2,a_3,\cdots,a_n$，现在想要**快速使得** $a_l,a_{l+1},a_{l+2},\cdots,a_r$ 的值增加 x。

- 朴素算法遍历 a_l 至 a_r，时间复杂度 $\mathcal{O}(r-l+1)$。~~（这不一样的吗？？）~~
- 使用差分算法，时间复杂度 $\mathcal{O}(1)$。
  记数组 $b_1,b_2,b_3,\cdots,b_n$ 使得 $b_i=a_i-a_{i-1}$。
  这样修改 b_l 为 b_l+x，b_{r+1} 为 $b_{r+1}-x$ 即可。
  因为 a_l 至 a_r 全部增加 x，相邻差值不变，而 a_l 与 a_{l-1} 的差值就增加了 x；倘若不更改 b_{r+1} 减去 x，那么还原时 a_{r+1} 至 a_n 全体都会增加 x，因此 b_{r+1} 要减去 x。
  更多相关内容见此。

处理策略

1.朴素算法

输入完成后枚举 1 至 m 哪个操作不做，暴力增加 1，最后统计 0 的个数。
时间复杂度：$\mathcal{O}(m(mn+n))$。
显然超时。

2.差分算法

输入完成后枚举 1 至 m 哪个操作不做，每次都差分维护区间 $[l,r]$ 增加，最后还原时统计 0 的数量即可。
时间复杂度：$\mathcal{O}(m(m+n))$。
考虑到数据范围$1\le n,m\le 3\times 10^5$，**仍会超时。**
得分：20pts。
部分代码：

```
const int N=3e5;
struct node{
    int l,r;
}a[N+1];//其实不用结构体也行
int n,m,cnt,cf[N+2];//差分数组
int main(){
    /*freopen("test.in","r",stdin);
    freopen("test.out","w",stdout);*/
```

```
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++)scanf("%d %d",&a[i].l,&a[i].r);
    for(int i=1;i<=m;i++){
        fill(cf+1,cf+n+1,0);//初始化
        cnt=0;
        //差分维护
        for(int j=1;j<=m;j++){
            if(j==i)continue;
            cf[a[j].l]++,cf[a[j].r+1]--;
        }//差分还原:统计0的数量
        for(int j=1;j<=n;j++){
            cf[j]+=cf[j-1];
            if(cf[j]==0)cnt++;
        }printf("%d\n",cnt);
    }

    /*fclose(stdin);
    fclose(stdout);*/
    return 0;
}
```

3.前缀和+差分优化

输入 n，m 后输入 l_i，r_i，输入的时候便直接使用差分维护增加 1。
维护完成后还原，还原时统计 0 的个数 $cnt0$。这时，只需要**加上 $[l_i,r_i]$ 内不执行操作 i 产生的 0 的数量 pl 即可。**
显然对于区间 $[l_i,r_i]$，最坏查找可以达到 $\mathcal{O}(n)$，那么时间复杂度便达到了 $\mathcal{O}(mn)$。
考虑到 mn 最大为 $(3\times10^5)\times(3\times10^5)=6\times10^{10}$，显然~~又双叒叕~~**具有超时的风险。**~~(这个超时代码就不贴了)~~
如果我们使用一个 $cnt1_i$ 记录 i 号位置是否为 1，那么显然
$pl=cnt1_{l_i}+cnt1_{l_i+1}+cnt1_{l_i+2}+\cdots+cnt1_{r_i}$。
仔细看看，不难发现，这就是**快速**求 $cnt1_{l_i}$ 至 $cnt1_{r_i}$ 的和。那么我们**更改 $cnt1$ 的定义**，使 $cnt1_i$ 为 1 至 i 号位里 1 的总数，则 $pl=cnt1_{r_i}-cnt1_{l_{i-1}}$。
则最终答案为 $cnt0+cnt1_{r_i}-cnt1_{l_{i-1}}$。
时间复杂度：$\mathcal{O}(n+m)$。

AC代码

```
//#include<bits/stdc++.h>
#include<algorithm>//个人习惯，忽略即可
#include<iostream>
#include<cstring>
#include<iomanip>
#include<cstdio>
#include<string>
#include<vector>
#include<cmath>
#include<ctime>
#include<deque>
#include<queue>
#include<stack>
#include<list>
using namespace std;
const int N=3e5;
int n,m,l[N+1],r[N+1],cnt0,cf[N+2],cnt1[N+1];
```

```
int main(){
    /*freopen("test.in","r",stdin);
    freopen("test.out","w",stdout);*/

    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        scanf("%d %d",l+i,r+i);
        cf[l[i]]++,cf[r[i]+1]--;//差分维护
    }
    for(int i=1;i<=n;i++){
        cf[i]+=cf[i-1];//差分还原
        cnt1[i]=cnt1[i-1];//cnt1前缀和
        if(cf[i]==0)cnt0++;//统计0的数量
        if(cf[i]==1)cnt1[i]++;//前缀和统计1~i里1的数量
    } //输出，含义如上文所述
    for(int i=1;i<=m;i++){
        printf("%d\n",cnt0+(cnt1[r[i]]-cnt1[l[i]-1]));
    }

    /*fclose(stdin);
    fclose(stdout);*/
    return 0;
}
```