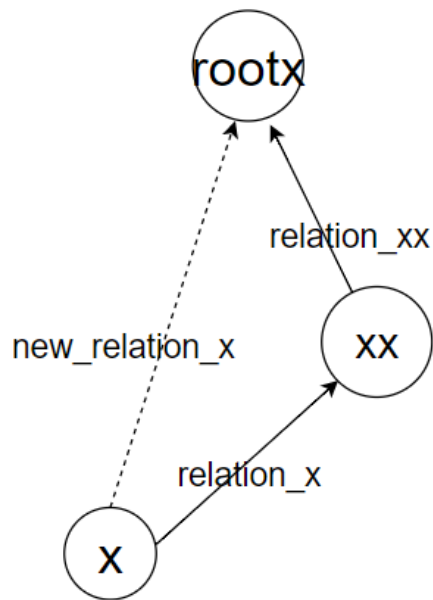


种类并查集

-PKU-1182食物链

压缩路径

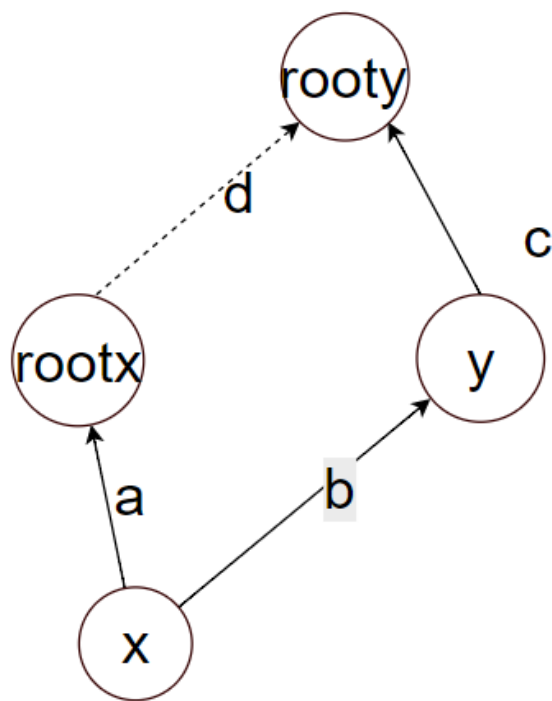


$x \rightarrow \text{rootx} = x \rightarrow \text{xx} + \text{xx} \rightarrow \text{rootx}$

即: $\text{new_relation_x} = (\text{relation_xx} + \text{relation}) \% 3$

<http://blog.csdn.net/DreamPoem>

UNION



a,表示a对于
rootx的关系

$\text{rootx} \rightarrow \text{rooty} = \text{rootx} \rightarrow x + x \rightarrow y + y \rightarrow \text{rooty}$

$= (3 - a + b + c) \% 3$

<http://blog.csdn.net/DreamPoem>

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cstring>
4  using namespace std;
5  const int maxn=50005;
6  int pre[maxn],rela[maxn]; //记录父亲结点、当前结点与父节点的关系
7  void init(int n)
8  {
9      for(int i=1;i<=n;i++)
10     {
11         pre[i]=i;
12         rela[i]=0;
13     }
14 }
15
16 int Find(int x)
17 {
18     if(pre[x]!=x)
19     {
20         int fa=pre[x];
21         pre[x]=Find(fa);
22         rela[x]=(rela[x]+rela[fa])%3;

```

```

23     }
24     return pre[x];
25 }
26
27 int ans,N;
28 void solve(int x,int y,int op)
29 {
30
31     if(x>N||y>N||(op==2&&x==y))
32     {
33         ans++;
34         return;
35     }
36     int fa_x,fa_y;
37     fa_x=Find(x);
38     fa_y=Find(y);
39     if(fa_x==fa_y)//两者在同一个集合中
40     {
41         if(((rela[x]+3-rela[y])%3)!=(op-1))
42             ans++;
43     }
44     else
45     {
46         pre[fa_x]=fa_y;//建立父节点之间的关系
47         rela[fa_x]=(3-rela[x]+op-1+rela[y])%3;
48     }
49 }
50 int main()
51 {
52     int K;
53     cin>>N>>K;
54     init(N);
55     ans=0;
56     while(K--)
57     {
58         int op,x,y;
59         scanf("%d%d%d",&op,&x,&y);
60         solve(x,y,op);
61     }
62     cout<<ans<<endl;
63 }
64

```

1. HDU-3038

判断区间和正误, 与种类并查集一样

```
1  #include<iostream>
2  #include<cstdio>
3  #include<cstring>
4  using namespace std;
5  const int maxn=200005;
6  int pre[maxn],sum[maxn]; //sum数组存储当前结点到父节点的和
7  void init(int n)
8  {
9      for(int i=0; i<=n; i++)
10     {
11         pre[i]=i;
12         sum[i]=0;
13     }
14 }
15
16 int Find(int x)
17 {
18
19     if(x!=pre[x])
20     {
21
22         int fa=pre[x];
23         pre[x]=Find(fa);
24         sum[x]=(sum[x]+sum[fa]);
25     }
26     return pre[x];
27 }
28
29 int ans;
30 void solve(int x,int y,int s)
31 {
32     int fa_x,fa_y;
33     fa_x=Find(x);
34     fa_y=Find(y);
35     if(fa_x==fa_y)
36     {
37         if(sum[x]-sum[y]!=s)
38             ans++;
39     }
40     else
41     {
42         pre[fa_x]=fa_y;
43         sum[fa_x]=-sum[x]+s+sum[y];
```

```

44     }
45 }
46 int main()
47 {
48     int n,m;
49     while(cin>>n>>m)
50     {
51         init(n);
52         ans=0;
53         while(m-->0)
54         {
55             int x, y,s;
56             scanf("%d%d%d",&x,&y,&s);
57             x--; //将区间离散化, 对于区间问题来说很常用
58             solve(x,y,s);
59         }
60         cout<<ans<<endl;
61     }
62     return 0;
63 }
64

```

2. 单点更新+区间求和

```

1  const int maxn=5e4+5;
2  int tree[maxn<<2];
3  int fa[maxn<<2];
4
5  void pushup(int i)
6  {
7      tree[i]=tree[i<<1]+tree[i<<1|1];
8  }
9  void build(int l,int r,int i)
10 {
11     if(l==r)
12     {
13         fa[l]=i;//记录叶子节点的下标
14         scanf("%d",&tree[i]);
15         return;
16     }
17     int m=(l+r)>>1;
18     build(lson);
19     build(rson);
20     pushup(i);
21 }

```

```

22 void updata(int i,int e)
23 { //自下而上更新
24     if(i==1)
25     {
26         tree[i]+=e;
27         return;
28     }
29     tree[i]+=e;
30     updata(i>>1,e); //更新父节点
31 }
32
33
34 int query(int L,int R,int l,int r,int i)
35 { // [L..R]是目标区间
36     //cout<<l<<' '<<r<<' '<<tree[i]<<endl;
37     if(L<=l&&R>=r)
38         return tree[i];
39     int ans=0;
40     int m=(l+r)>>1;
41     if(L<=m)
42         ans+=query(L,R,lson);
43     if(R>m)
44         ans+=query(L,R,rson);
45     return ans;
46 }

```

3. 单点更新+区间比较

```

1 void pushup(int i)
2 {
3     tree[i]=max(tree[i<<1],tree[i<<1|1]);
4 }
5
6 void updata(int i)
7 { //叶子节点的值在主函数中已经被更新，只需递归向上更细致根节点就好
8     if(i==1)
9         return ;
10    pushup(i>>1); //更新叶子结点的父节点，不可以更新叶子节点！！
11    updata(i>>1);
12 }
13

```

4. 区间更新+区间查询

```

1  #define lson l,m,i<<1
2  #define rson m+1,r,i<<1|1
3  typedef long long ll;
4  using namespace std;
5  const int maxn=1e7+5;
6  ll tree[maxn<<2],seg[maxn<<2];
7
8  void pushup(int i)
9  {
10     tree[i]=tree[i<<1]+tree[i<<1|1];
11 }
12
13 void pushdown(int l,int r,int i)
14 { //下放标记
15     if(seg[i]==0)
16         return ;
17     int m=(l+r)>>1;
18     seg[i<<1]+=seg[i]; //子结点的标记叠加
19     tree[i<<1]+=seg[i]*(m-l+1); //子结点的和直接更新
20     seg[i<<1|1]+=seg[i]; //子结点的标记叠加
21     tree[i<<1|1]+=seg[i]*(r-m); //子结点的和直接更新
22     seg[i]=0; //当前结点的标记 解除
23 }
24
25 void build(int l,int r,int i)
26 {
27     if(l==r)
28     {
29         scanf("%lld",&tree[i]);
30         return ;
31     }
32     int m=(l+r)>>1;
33     build(lson);
34     build(rson);
35     pushup(i);
36 }
37
38 void updata(int L,int R,int l,int r,int i,int e)
39 { //自定向下更新
40     if(L<=l&&R>=r)
41     {
42         seg[i]+=e;
43         tree[i]+=(ll)(e*(r-l+1));
44         return;
45     }
46     pushdown(l,r,i); //下方沿途标记

```

```

47     int m=(l+r)>>1;
48     if(L<=m)
49         updata(L,R,lson,e);
50     if(R>m)
51         updata(L,R,rson,e);
52     pushup(i);//记得向上更新
53 }
54
55 // query(int L,int R,int l,int r,int i)
56 {
57     if(L<=l&&R>=r)
58         return tree[i];
59     // ans=0;
60     pushdown(l,r,i);//查询时消除沿途标记
61     int m=(l+r)>>1;
62     if(L<=m)
63         ans+=query(L,R,lson);
64     if(R>m)
65         ans+=query(L,R,rson);
66     return ans;
67 }

```

5. 区间合并查询某个点所在的最大区间长度,

初始化区间都是连续的

```

1  using namespace std;
2  const int maxn=5e4+5;
3  int len[maxn<<2],llen[maxn<<2],rlen[maxn<<2];
4  void pushup(int l,int r,int i)
5  { //更新当前区间的最大连续区间
6      int m=(l+r)>>1;
7      len[i]=max(len[i<<1],len[i<<1|1]);
8      len[i]=max(len[i],rlen[i<<1]+llen[i<<1|1]);
9      llen[i]=llen[i<<1];
10     rlen[i]=rlen[i<<1|1];
11
12     if(len[i<<1]==m-l+1)//左子树连续: 当前区间的连续长度等于左子树的右连续长度+右子树的
        做连续长度
13         llen[i]+=llen[i<<1|1];
14     if(len[i<<1|1]==r-m)//右子树连续
15         rlen[i]+=rlen[i<<1];
16 }
17 void build(int l,int r,int i)
18 { //初始化区间连续

```



```

19     len[i]=rlen[i]=llen[i]=r-l+1;
20     if(r==l)
21         return;
22     int m=(l+r)>>1;
23     build(lson);
24     build(rson);
25 }
26
27 void updata(int l,int r,int i,int p,int e)
28 { //查找目标叶子节点p, 重新设置结点长度为e
29     //更新叶子节点的长度
30     if(l==r&&l==p)
31     {
32         len[i]=rlen[i]=llen[i]=e;
33         return ;
34     }
35     int m=(l+r)>>1;
36     if(p<=m)
37         updata(lson,p,e);
38     else
39         updata(rson,p,e);
40     pushup(l,r,i);
41 }
42 int query(int l,int r,int i,int p)
43 {
44     if(len[i]==0 || len[i]==(r-l+1))//包含p点的区间完全连续, 或者完全不连续 不用在往
    下找
45         return len[i];
46     int m=(l+r)>>1;
47     if(p<=m)
48         //p<=m    p>m-rlen[i] 说明p在左子树的右连续区间
49         return p>m-rlen[i]<<1?rlen[i<<1]+llen[i<<1|1]:query(lson,p);
50         //p>m    p<m+llen[i<<1|1] 说明p在右子树的左连续区间
51     return p<=m+llen[i<<1|1]?rlen[i<<1]+llen[i<<1|1]:query(rson,p);
52 }

```

一维树状数

组最大用处是维护前缀和

1. 1. 单点修改+区间查询

$$C1 = A1;$$

$$C2 = A1 + A2;$$

$$C3 = A3;$$

$$C4 = A1 + A2 + A3 + A4;$$

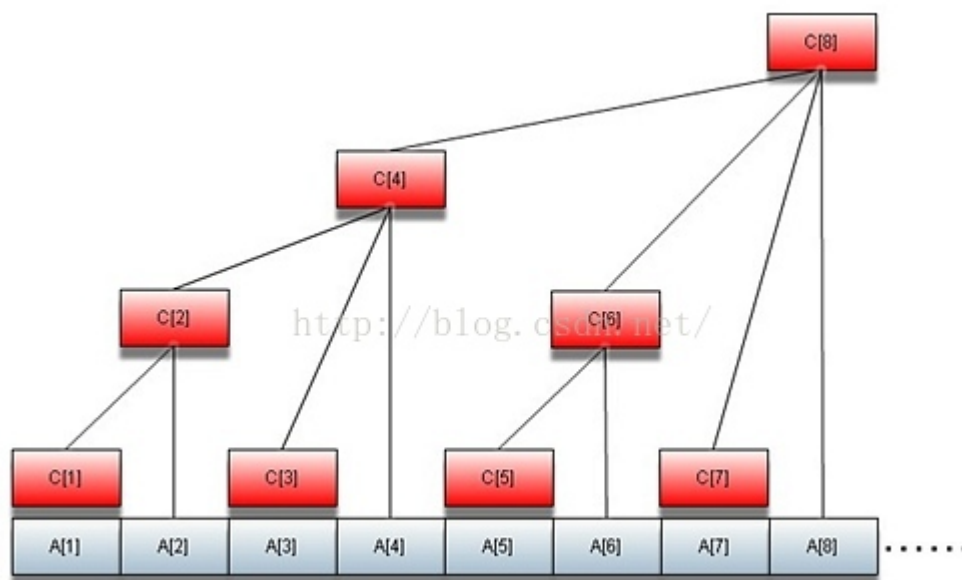
$$C5 = A5;$$

$$C6 = A5 + A6.$$

.....

$$C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;$$

.....



注意：没有A[0]。

如图可知：为奇数的时候他是代表他本身，而为偶数的时候则是代表着自己以及属于它管辖区域的和。

p 管辖的区域大小为 $p \& (-p)$

```
1 void add(int p,int x)//sum[p]+=x
2 { //维护sum数组
3     while(p<=n)
4     {
5         sum[p]+=x;
6         p+=p&(-p);
7     }
8 }
9
10 int ask(int p)
11 { //前p项元素的和
12     int ans=0;
13     while(p)
14     {
15         ans+=sum[p];
16         p-=p&(-p);
17     }
18     return ans;
19 }
20
21 int range_sum(int l,int r)
22 {
23     return ask(r)-ask(l-1);
24 }
```

2. 2. 区间修改+单点查询

维护一个差值数组 $d[1..n]$ ，定义

$$d[i] = a[i] - a[i - 1] \quad , \quad a[0] = 0 \quad (1)$$

- **查询：** $a[i] = \sum_{j=1}^i d[j]$ ，通过 $d[j]$ 的前缀和求出 $a[i]$ 单点的值。
- **更新：**通过 $d[i]$ 的性质我们可以知道，当更新 $a[l..r] += x$ 时， $d[l] += x, d[r + 1] -= x$

```
1
2 void add(int p,int x)
3 { //tree[]=d[]
4     while(p<=n)
```

```

5     {
6         tree[p]+=x;
7         p+=p&(-p);
8     }
9 }
10 void rang_add(int l,int r,int x)
11 {
12     add(l,x);
13     add(r+1,-x);
14 }
15 void ask(int p)
16 {
17     int sum=0;
18     while(p)
19     {
20         sum+=tree[p];
21         p-=p&(-p);
22     }
23     return p;
24 }
25
26 void init(int n)
27 {
28     for(int i=1;i<=n;i++)
29         add(i,a[i]-a[i-1]);
30 }

```

3. 3. 区间修改+区间查询

基于2的思想我们可以得到区间查询的式子

$$\sum_{i=1}^p a[i] = \sum_{i=1}^p \sum_{j=1}^i d[j] \quad (2)$$

观察后发现： $d[1]$ 计算了 p 次， $d[2]$ 计算了 $p - 1$ 次， $d[3]$ 计算了 $p - 2$ 次， ...。

- **查询：** 基于此我们得到了查询的式子。

$$\begin{aligned}
 \sum_{i=1}^p a[i] &= \sum_{i=1}^p \sum_{j=1}^i d[j] \\
 &= \sum_{i=1}^p (d[i] \times (p - i + 1)) \\
 &= (p + 1) \times \sum_{i=1}^p d[i] - \sum_{i=1}^p (d[i] \times i)
 \end{aligned} \quad (3)$$

因此我们维护**两个树状数组**即可,

初始化: $sum1[i] = d[i]$, $sum2[i] = d[i] * i$ 。

- 修改:

对 $sum1[i]$ 的修改同上2对 $d[i]$ 的修改;

对 $sum2[i]$ 的修改根据定义有: $sum2[l] = l * x, sum2[r + 1] = (r + 1) * x$ 。

```
1 //a[] 维护输入数据的前缀和
2 //注意: 我们维护的时更新值的区间和, 因此初始化时d[i]=0
3 //询问总的区间和: rang_ask(l,r)+(a[r]-a[l-1])    !!!
4 //详情看例题
5 void add(int p,int x)
6 {
7     int i=p;
8     while(i<=n)
9     {
10         sum1[i]+=x;
11         sum2[i]+=p*x;
12         i+=i&(-i);
13     }
14 }
15 void rang_add(int l,int r,int x)
16 {
17     add(l,x);
18     add(r+1,-x);
19 }
20
21 long long ask(int p)
22 {
23     long long ans=0;
24     int i=p;
25     while(i)
26     {
27         ans+=(p+1)*sum1[i]-sum2[i];
28         i-=i&(-i);
29     }
30     return ans;
31 }
32 long long rang_ask(int l,int r,int p)
33 {
34     return ask(r)-ask(l-1);
35 }
```

4. 一维树状数组例题

4.1. [POJ2352](#)，裸一维数组

给你N个星星的坐标，判断当前星星是第几层，结果输出每层星星的个数。

层的定义是有多少个星星，位于该星星的左下方区域（按照二位直角坐标系划分）。

思路：由于输入是按照y坐标非递减，排序的；因此只需统计当前横坐标之前的所有星星个数（星星个数的**前缀和**）

因此只需维护x方向的星星数量的树状数组即可。x方向的树状数组，表示了当前x坐标左侧，（包括当前x坐标）的星星数目。

```
1  #include<iostream>
2  #include<cstring>
3  #include<cstdio>
4  using namespace std;
5  const int maxn=32005;
6  int star[maxn],ans[maxn];
7  //star[] 维护一个关于x轴坐标星星数目的树状数组,
8  //ans[] 记录该层星星的个数
9  int n;
10 void add(int i,int a)
11 {
12     while(i<=maxn)
13     {
14         star[i]+=a;
15         i+=i&(-i);
16     }
17 }
18
19 int ask(int i)
20 {
21     int sum=0;
22     while(i)
23     {
24         sum+=star[i];
25         i-=i&(-i);
26     }
27     return sum;
28 }
29
30 int main()
31 {
```

```

32     int N;
33     scanf("%d",&N);
34     memset(star,0,sizeof(star));
35     memset(ans,0,sizeof(ans));
36     for(int i=1;i<=N;i++)
37     {
38         int x,int y;
39         scanf("%d%d",&x,&y);
40         x++;//x不要是0
41         add(x,1);//增加一个星星，更新树状数组
42         int s=ask(x);
43         ans[s-1]++;//该层星星数量+1
44     }
45     for(int i=0;i<N;i++)
46         printf("%d\n",ans[i]);
47
48 }
49

```

4.2. [POJ2299](#) 树状数组求前缀和

题意：给你一个序列，输出该序列中的所有逆序对数目。

思路：**读入数据时**， $tree[i]$ 记录数字 i 的个数，输入时我们可以找出小于 i 的个数，输入数据 i 的**下标** $-idx$ ，即为 i 的逆序对数目。

处理前先对输入数据 i 进行**离散化**。

```

1  #include<iostream>
2  #include<cstring>
3  #include<cstdio>
4  #include<algorithm>
5  using namespace std;
6  const int maxn=5e5+5;
7  int tree[maxn],B[maxn];
8  struct node
9  {
10     int val,pos;
11     bool operator<(struct node a) const
12     {
13         return val<a.val;
14     }
15 }A[maxn];
16 int n;

```

```
17 void add(int x)
18 {
19     while(x<=n)//离散后的数据上限
20     {
21         tree[x]+=1;
22         x+=x&(-x);
23     }
24 }
25
26 int ask(int x)
27 {
28     int sum=0;
29     while(x)
30     {
31         sum+=tree[x];
32         x-=x&(-x);
33     }
34     return sum;
35 }
36
37 int main()
38 {
39     while(scanf("%d",&n)&&n)
40     {
41         for(int i=1;i<=n;i++)
42         {
43             scanf("%d",&A[i].val);
44             A[i].pos=i;
45         }
46         sort(A+1,A+n+1);
47         for(int i=1;i<=n;i++)
48             B[A[i].pos]=i;
49         long long ans=0;
50         memset(tree,0,sizeof(tree));
51         for(int i=1;i<=n;i++)
52         {
53             add(B[i]);
54             ans+=i-ask(B[i]);
55         }
56         cout<<ans<<endl;
57     }
58     return 0;
59 }
60
```


4.3. [HDU1556](#) 区间更新+单点查询

```
1  #include<iostream>
2  #include<cstdio>
3  #include<cstring>
4  using namespace std;
5  const int maxn=1e5+5;
6  int d[maxn];
7  int N;
8  void add(int p,int x)
9  {
10     while(p<=N)
11     {
12         d[p]+=x;
13         p+=p&(-p);
14     }
15 }
16 void rang_add(int l,int r)
17 {
18     add(l,1);
19     add(r+1,-1);
20 }
21 int ask(int p)
22 {
23     int sum=0;
24     while(p)
25     {
26         sum+=d[p];
27         p-=p&(-p);
28     }
29     return sum;
30 }
31 int main()
32 {
33     while(scanf("%d",&N))
34     {
35         if(N==0) break;
36         memset(d,0,sizeof(d));
37         for(int i=1;i<=N;i++)
38         {
39             int a,b;
40             scanf("%d%d",&a,&b);
41             rang_add(a,b);
42         }
43         for(int i=1;i<N;i++)
```

```

44         printf("%d ",ask(i));
45         printf("%d\n",ask(N));
46     }
47     return 0;
48
49 }
50

```

4.4. [POJ3468](#) 区间更新+区间求和

树状数组维护更新值，而不是实际值。

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cstring>
4  using namespace std;
5  const int maxn=1e5+5;
6  long long A[maxn];
7  long long sum1[maxn],sum2[maxn];
8  int n;
9  void add(int p,int x)
10 {
11     int i=p;
12     while(i<=n)
13     {
14         sum1[i]+=x;
15         sum2[i]+=p*x;
16         i+=i&(-i);
17     }
18 }
19
20 void rang_add(int l,int r,int x)
21 {
22     add(l,x);
23     add(r+1,-x);
24 }
25
26 long long ask(int p)
27 {
28     long long s=0;
29     int i=p;
30     while(i)
31     {
32         s+=(p+1)*sum1[i]-sum2[i];
33         i-=i&(-i);

```

```

34
35     }
36     return s;
37 }
38 long long rang_ask(int l,int r)
39 {
40     return ask(r)-ask(l-1);
41 }
42 int main()
43 {
44     int q;
45     while(cin>>n>>q)
46     {
47
48         memset(A,0,sizeof(A));
49         memset(sum1,0,sizeof(sum1));
50         memset(sum2,0,sizeof(sum2));
51         for(int i=1; i<=n; i++)
52         {
53             scanf("%lld",&A[i]);
54             A[i]+=A[i-1]; //输入数据的前缀和
55         }
56         while(q-->0)
57         {
58             char op;
59             long long a;
60             long long b;
61             scanf("%c%lld%lld",&op,&a,&b);
62             if(op=='Q')
63                 printf("%lld\n",A[b]-A[a-1]+rang_ask(a,b)); //真正的区间和
64             else
65             {
66                 int c;
67                 scanf("%d",&c);
68                 rang_add(a,b,c);
69             }
70         }
71     }
72 }
73 return 0;
74 }
75

```

二维树状数组

在二维树状数组中, $tree[x][y]$ 记录的是右下角 (x, y) , 高为 $lowbit(x)$, 宽为 $lowbit(y)$ 的区间的区间和。

1. 单点修改+区间查询

根据树状数组的性质, 更新 (x, y) 点时, 只需同时更新以 (x, y) 为原点划分的**右下角矩阵**。

查询 (x, y) 点时, 查询以 (x, y) 为原点划分得到**左上角矩阵**。

```
1 void add(int x,int y,int z)//将点(x,y)加上z
2 { //n*m的矩阵
3     int i=x,j=y;
4     while(i<=n)
5     {
6         j=y;
7         while(j<=m)
8         {
9             tree[i][j]+=z;
10            j+=j&(-j);
11        }
12        i+=i&(-i);
13    }
14 }
15
16
17 void ask(int x,int y)
18 {
19     int s=0;
20     int i=x,j=y;
21     while(i)
22     {
23         j=y;
24         while(j)
25         {
26             s+=tree[i][j];
27             j-=j&(-j);
28         }
29         i-=i&(-i);
30     }
31     return s;
32 }
```

2. 区间修改+单点查询

令 $tree[i][j]$ 数组,记录主对角线为 $(1, 1)$ 、 (i, j) 的增量和。

我们已有的更新操作是,更新以 (x, y) 为原点划分的右下角的矩阵;

查询 (i, j) 点: $sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + a[i][j]$;

要求 $sum[i][j]$ 是查询点的增量值, $a[i][j]$ 是原来的值。

```
1 //输入一个n*n的矩阵, 区间更新, 查询某点的值
2 #include<iostream>
3 #include<cstdio>
4 #include<cstring>
5 using namespace std;
6 const int maxn=1e3+5;
7 int tree[maxn][maxn];
8
9 int n;
10 void add(int x,int y,int z)
11 {
12     int i=x,j=y;
13     while(i<=n)
14     {
15         j=y;
16         while(j<=n)
17         {
18             tree[i][j]+=z;
19             j+=j&(-j);
20         }
21         i+=i&(-i);
22     }
23 }
24
25 void rang_add(int x1,int y1,int x2,int y2,int z)
26 {
27     add(x1,y1,z);
28     add(x1,y2+1,-z);
29     add(x2+1,y1,-z);
30     add(x2+1,y2+1,z);
31 }
32
33 int ask(int x,int y)
34 {
```

```

35     int i=x,j=y;
36     int s=0;
37     while(i)
38     {
39         j=y;
40         while(j)
41         {
42             s+=tree[i][j];
43             j-=j&(-j);
44         }
45         i-=i&(-i);
46     }
47     return s;
48 }
49
50 int main()
51 {
52
53     cin>>n;
54     for(int i=1;i<=n;i++)
55         for(int j=1;j<=n;j++)
56         {
57             int e;
58             cin>>e;
59             rang_add(i,j,i,j,e);
60         }
61     int x1,y1,x2,y2,z;
62     cin>>x1>>y1>>x2>>y2>>z;
63     rang_add(x1,y1,x2,y2,z);
64     cout<<ask(2,3);
65     return 0;
66
67 }
68
69

```

3. 区间修改+区间查询

```

1  //5*5矩阵区间更新+区间查询历程
2  #include <cstdio>
3  #include <cmath>
4  #include <cstring>
5  #include <algorithm>
6  #include <iostream>
7  using namespace std;

```

```

8  typedef long long ll;
9  const int N = 205;
10 ll n, m, Q;
11 ll t1[N][N], t2[N][N], t3[N][N], t4[N][N];
12 void add(ll x, ll y, ll z)
13 {
14     for(int X = x; X <= n; X += X & -X)
15         for(int Y = y; Y <= m; Y += Y & -Y)
16         {
17             t1[X][Y] += z;
18             t2[X][Y] += z * x;
19             t3[X][Y] += z * y;
20             t4[X][Y] += z * x * y;
21         }
22 }
23 void range_add(ll xa, ll ya, ll xb, ll yb, ll z) //(xa, ya) 到 (xb, yb) 的
矩形
24 {
25     add(xa, ya, z);
26     add(xa, yb + 1, -z);
27     add(xb + 1, ya, -z);
28     add(xb + 1, yb + 1, z);
29 }
30 ll ask(ll x, ll y)
31 {
32     ll res = 0;
33     for(int i = x; i; i -= i & -i)
34         for(int j = y; j; j -= j & -j)
35             res += (x + 1) * (y + 1) * t1[i][j]
36                 - (y + 1) * t2[i][j]
37                 - (x + 1) * t3[i][j]
38                 + t4[i][j];
39     return res;
40 }
41 ll range_ask(ll xa, ll ya, ll xb, ll yb)
42 {
43     return ask(xb, yb) - ask(xb, ya - 1) - ask(xa - 1, yb) + ask(xa - 1, ya
- 1);
44 }
45 int main()
46 {
47     n=5,m=5;
48     for(int i=1; i<=n; i++)
49         for(int j=1; j<=n; j++)
50         {
51             int z;

```

```
52         cin>>z;
53         range_add(i,j,i,j,z);
54     }
55
56     int x1,x2,y1,y2,z;
57     cin>>x1>>y1>>x2>>y2>>z;
58     range_add(x1,y1,x2,y2,z);
59     cout<<range_ask(1,1,5,5);
60     return 0;
61 }
62
```