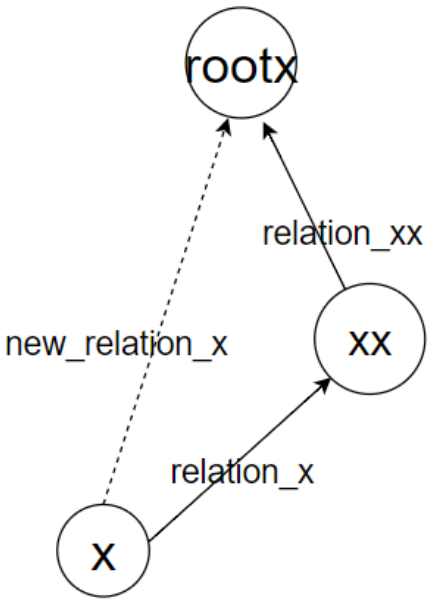
种类并查集

-PKU-1182食物链

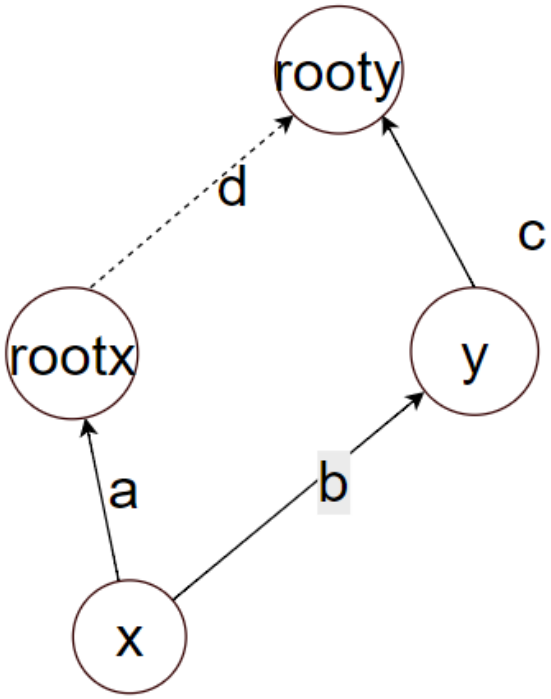
压缩路径



x->rootx=x->xx+xx->rootx

即：new_relation_x=(relation_xx+relation) %3;

UNION



a,表示a对于rootx的关系

rootx->rooty=rootx->x+x->y+y->rooty

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


```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=50005;
int pre[maxn],rela[maxn];//记录父亲结点、当前结点与父节点的关系
void init(int n)
{
    for(int i=1;i<=n;i++)
    {
        pre[i]=i;
        rela[i]=0;
    }
}

int Find(int x)
{
    if(pre[x]!=x)
    {
        int fa=pre[x];
        pre[x]=Find(fa);
        rela[x]=(rela[x]+rela[fa])%3;
```

```
    }
    return pre[x];
}

int ans,N;
void solve(int x,int y,int op)
{

    if(x>N||y>N||(op==2&&x==y))
        {
            ans++;
            return;
        }
    int fa_x,fa_y;
    fa_x=Find(x);
    fa_y=Find(y);
    if(fa_x==fa_y)//两者在同一个集合中
    {
        if(((rela[x]+3-rela[y])%3)!=(op-1))
            ans++;
    }
    else
    {
        pre[fa_x]=fa_y;//建立父节点之间的关系
        rela[fa_x]=(3-rela[x]+op-1+rela[y])%3;
    }
}
int main()
{
    int K;
    cin>>N>>K;
    init(N);
    ans=0;
    while(K--)
    {
        int op,x,y;
        scanf("%d%d%d",&op,&x,&y);
        solve(x,y,op);
    }
    cout<<ans<<endl;
}

```

1. HDU-3038

判断区间和正误, 与种类并查集一样

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=200005;
int pre[maxn],sum[maxn];//sum数组存储当前结点到父节点的和
void init(int n)
{
    for(int i=0; i<=n; i++)
    {
        pre[i]=i;
        sum[i]=0;
    }
}

int Find(int x)
{

    if(x!=pre[x])
    {

        int fa=pre[x];
        pre[x]=Find(fa);
        sum[x]=(sum[x]+sum[fa]);
    }
    return pre[x];
}

int ans;
void solve(int x,int y,int s)
{
    int fa_x,fa_y;
    fa_x=Find(x);
    fa_y=Find(y);
    if(fa_x==fa_y)
    {
        if(sum[x]-sum[y]!=s)
            ans++;
    }
    else
    {
        pre[fa_x]=fa_y;
        sum[fa_x]=-sum[x]+s+sum[y];
```

```
    }
}
int main()
{
    int n,m;
    while(cin>>n>>m)
    {
        init(n);
        ans=0;
        while(m--)
        {
            int x, y,s;
            scanf("%d%d%d",&x,&y,&s);
            x--; //将区间离散化，对于区间问题来说很常用
            solve(x,y,s);
        }
        cout<<ans<<endl;
    }
    return 0;
}

```

2. 单点更新+区间求和

```
const int maxn=5e4+5;
int tree[maxn<<2];
int fa[maxn<<2];

void pushup(int i)
{
    tree[i]=tree[i<<1]+tree[i<<1|1];
}
void build(int l,int r,int i)
{
    if(l==r)
    {
        fa[l]=i;//记录叶子节点的下标
        scanf("%d",&tree[i]);
        return;
    }
    int m=(l+r)>>1;
    build(lson);
    build(rson);
    pushup(i);
}
```

```
void updata(int i,int e)
{//自下而上更新
    if(i==1)
    {
        tree[i]+=e;
        return;
    }
    tree[i]+=e;
    updata(i>>1,e);//更新父节点
}


int query(int L,int R,int l,int r,int i)
{//[L..R]是目标区间
    //cout<<l<<' '<<r<<' '<<tree[i]<<endl;
    if(L<=l&&R>=r)
        return tree[i];
    int ans=0;
    int m=(l+r)>>1;
    if(L<=m)
        ans+=query(L,R,lson);
    if(R>m)
        ans+=query(L,R,rson);
    return ans;
}
```

3. 单点更新+区间比较

```
void pushup(int i)
{
    tree[i]=max(tree[i<<1],tree[i<<1|1]);
}

void updata(int i)
{//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
    if(i==1)
        return ;
    pushup(i>>1);//更新叶子结点的父节点，不可以更新叶子节点！！
    updata(i>>1);
}

```

4. 区间更新+区间查询

```cpp
#define lson l,m,i<<1
#define rson m+1,r,i<<1|1
typedef long long ll;
using namespace std;
const int maxn=1e7+5;
ll tree[maxn<<2],seg[maxn<<2];

void pushup(int i)
{
    tree[i]=tree[i<<1]+tree[i<<1|1];
}

void pushdown(int l,int r,int i)
{//下放标记
    if(seg[i]==0)
        return ;
    int m=(l+r)>>1;
    seg[i<<1]+=seg[i];//子结点的标记叠加
    tree[i<<1]+=seg[i]*(m-l+1);//子结点的和直接更新
    seg[i<<1|1]+=seg[i];//子结点的标记叠加
    tree[i<<1|1]+=seg[i]*(r-m);//子结点的和直接更新
    seg[i]=0;//当前结点的标记  解除
}

void build(int l,int r,int i)
{
    if(l==r)
    {
        scanf("%lld",&tree[i]);
        return ;
    }
    int m=(l+r)>>1;
    build(lson);
    build(rson);
    pushup(i);
}

void updata(int L,int R,int l,int r,int i,int e)
{//自定向下更新
    if(L<=l&&R>=r)
    {
        seg[i]+=e;
        tree[i]+=(ll)(e*(r-l+1));
        return;
    }
    pushdown(l,r,i);//下方沿途标记
```

```
    int m=(l+r)>>1;
    if(L<=m)
        updata(L,R,lson,e);
    if(R>m)
        updata(L,R,rson,e);
    pushup(i);//记得向上更新
}

ll query(int L,int R,int l,int r,int i)
{
    if(L<=l&&R>=r)
        return tree[i];
    ll ans=0;
    pushdown(l,r,i);//查询时消除沿途标记
    int m=(l+r)>>1;
    if(L<=m)
        ans+=query(L,R,lson);
    if(R>m)
        ans+=query(L,R,rson);
    return ans;
}
```

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

```
using namespace std;
const int maxn=5e4+5;
int len[maxn<<2],llen[maxn<<2],rlen[maxn<<2];
void pushup(int l,int r,int i)
{//更新当前区间的最大连续区间
    int m=(l+r)>>1;
    len[i]=max(len[i<<1],len[i<<1|1]);
    len[i]=max(len[i],rlen[i<<1]+llen[i<<1|1]);
    llen[i]=llen[i<<1];
    rlen[i]=rlen[i<<1|1];

    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
        llen[i]+=llen[i<<1|1];
    if(len[i<<1|1]==r-m)//右子树连续
        rlen[i]+=rlen[i<<1];
}
void build(int l,int r,int i)
{//初始化区间连续
```

```
    len[i]=rlen[i]=llen[i]=r-l+1;
    if(r==l)
        return;
    int m=(l+r)>>1;
    build(lson);
    build(rson);
}

void updata(int l,int r,int i,int p,int e)
{   //查找目标叶子节点p，重新设置结点长度为e
    //更新叶子节点的长度
    if(l==r&&l==p)
    {
        len[i]=rlen[i]=llen[i]=e;
        return ;
    }
    int m=(l+r)>>1;
    if(p<=m)
        updata(lson,p,e);
    else
        updata(rson,p,e);
    pushup(l,r,i);
}
int query(int l,int r,int i,int p)
{
    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
        return len[i];
    int m=(l+r)>>1;
    if(p<=m)
    //p<=m   p>m-rlen[i]  说明p在左子树的右连续区间
        return p>m-rlen[i<<1]?rlen[i<<1]+llen[i<<1|1]:query(lson,p);
    //p>m   p<m+llen[i<<1|1] 说明p在右子树的左连续区间
    return p<=m+llen[i<<1|1]?rlen[i<<1]+llen[i<<1|1]:query(rson,p);
}
```

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;

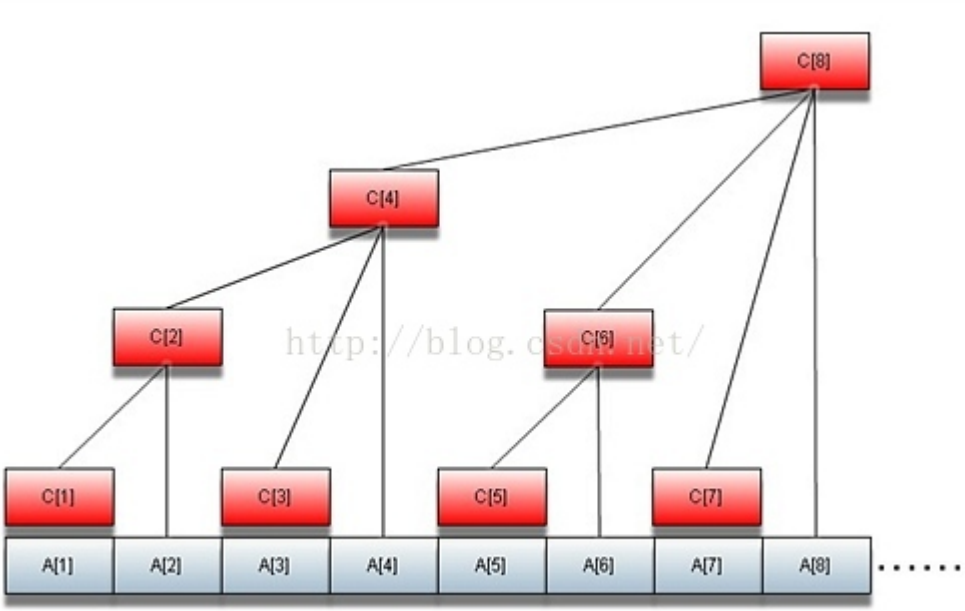
C5 = A5;

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$p\&(-p)$

```
void add(int p,int x)//sum[p]+=x
{//维护sum数组
    while(p<=n)
    {
        sum[p]+=x;
        p+=p&(-p);
    }
}

int ask(int p)
{//前p项元素的和
    int ans=0;
    while(p)
    {
         ans+=sum[p];
         p-=p&(-p);
    }
    return ans;
}

int range_sum(int l,int r)
{
    return ask(r)-ask(l-1);
}
```

2. 2. 区间修改+单点查询

维护一个差值数组$d[1..n]$，定义

$$d[i]=a[i]-a[i-1] \quad 、a[0]=0 \tag{1}$$

- **查询**：$a\left[i\right]=\sum_{1=1}^{i}d\left[j\right]$，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，$d[l]+=x, d[r+1]-=x$

```

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

```
    {
        tree[p]+=x;
        p+=p&(-p);
     }
}
void rang_add(int l,int r,int x)
{
    add(l,x);
    add(r+1,-x);
}
void ask(int p)
{
    int sum=0;
    while(p)
    {
        sum+=tree[p];
        p-=p&(-p);
    }
    return p;
}

void init(int n)
{
    for(int i=1;i<=n;i++)
        add(i,a[i]-a[i-1]);
}
```

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

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

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。

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

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：$sum2[l]=l*x$,$sum2[r+1]=(r+1)*x$。

```
//a[] 维护输入数据的前缀和
//注意：我们维护的时更新值的区间和，因此初始化时d[i]=0
//询问总的区间和：rang_ask(l,r)+(a[r]-a[l-1])   ！！！
//详情看例题
void add(int p,int x)
{
    int i=p;
    while(i<=n)
    {
        sum1[i]+=x;
        sum2[i]+=p*x;
        i+=i&(-i);
    }
}
void rang_add(int l,int r,int x)
{
    add(l,x);
    add(r+1,-x);
}

long long ask(int p)
{
    long long ans=0;
    int i=p;
    while(i)
    {
       ans+=(p+1)*sum1[i]-sum2[i];
        i-=i&(-i);
    }
    return sum;
}
long long rang_ask(int l,int r,int p)
{
    return ask(r)-ask(l-1);
}
```

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。

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
const int maxn=32005;
int star[maxn],ans[maxn];
//star[] 维护一个关于x轴坐标星星数目的树状数组,
//ans[ ]记录该层星星的个数
int n;
void add(int i,int a)
{
    while(i<=maxn)
    {
        star[i]+=a;
        i+=i&(-i);
    }
}

int ask(int i)
{
    int sum=0;
    while(i)
    {
        sum+=star[i];
        i-=i&(-i);
    }
    return sum;

}
int main()
{
```

```
    int N;
    scanf("%d",&N);
    memset(star,0,sizeof(star));
    memset(ans,0,sizeof(ans));
    for(int i=1;i<=N;i++)
    {
        int x;int y;
        scanf("%d%d",&x,&y);
        x++;//x不要是0
        add(x,1);//增加一个星星，更新树状数组
        int s=ask(x);
        ans[s-1]++;//该层星星数量+1
    }
    for(int i=0;i<N;i++)
        printf("%d\n",ans[i]);

}

```

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。**

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
const int maxn=5e5+5;
int tree[maxn],B[maxn];
struct node
{
    int val,pos;
    bool operator<(struct node a) const
    {
        return val<a.val;
    }
}A[maxn];
int n;
```

```cpp
void add(int x)
{
    while(x<=n)//离散后的数据上限
    {
        tree[x]+=1;
        x+=x&(-x);
    }
}

int ask(int x)
{
    int sum=0;
    while(x)
    {
        sum+=tree[x];
        x-=x&(-x);
    }
    return sum;
}

int main()
{
    while(scanf("%d",&n)&&n)
    {
        for(int i=1;i<=n;i++)
        {
            scanf("%d",&A[i].val);
            A[i].pos=i;
        }
        sort(A+1,A+n+1);
        for(int i=1;i<=n;i++)
            B[A[i].pos]=i;
        long long ans=0;
        memset(tree,0,sizeof(tree));
        for(int i=1;i<=n;i++)
        {
            add(B[i]);
            ans+=i-ask(B[i]);
        }
        cout<<ans<<endl;
    }
    return 0;
}

```

4.3. HDU1556 区间更新+单点查询

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=1e5+5;
int d[maxn];
int N;
void add(int p,int x)
{
    while(p<=N)
    {
        d[p]+=x;
        p+=p&(-p);
    }
}
void rang_add(int l,int r)
{
    add(l,1);
    add(r+1,-1);
}
int ask(int p)
{
    int sum=0;
    while(p)
    {
        sum+=d[p];
        p-=p&(-p);
    }
    return sum;
}
int main()
{
    while(scanf("%d",&N))
    {
        if(N==0)  break;
        memset(d,0,sizeof(d));
        for(int i=1;i<=N;i++)
        {
            int a,b;
            scanf("%d%d",&a,&b);
            rang_add(a,b);
        }
        for(int i=1;i<N;i++)
```

```
            printf("%d ",ask(i));
        printf("%d\n",ask(N));
    }
    return 0;

}

```

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。**

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=1e5+5;
long long A[maxn];
long long sum1[maxn],sum2[maxn];
int n;
void add(int p,int x)
{
    int i=p;
    while(i<=n)
    {
        sum1[i]+=x;
        sum2[i]+=p*x;
        i+=i&(-i);
    }
}

void rang_add(int l,int r,int x)
{
    add(l,x);
    add(r+1,-x);
}

long long ask(int p)
{
    long long s=0;
    int i=p;
    while(i)
    {
        s+=(p+1)*sum1[i]-sum2[i];
        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--)
57          {
58              char op;
59              long long a;
60              long long b;
61              scanf("%*c%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

在二位树状数组中，$tree[x][y]$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**。

```
void add(int x,int y,int z)//将点(x,y)加上z
{//n*m的矩阵
    int i=x,j=y;
    wwhile(i<=n)
    {
        j=y;
        while(j<=m)
        {
            tree[i][j]+=z;
            j+=j&(-j);
        }
        i+=i&(-i);
    }
}


void ask(int x,int y)
{
    int s=0;
    int i=x,j=y;
    while(i)
    {
        j=y;
        while(j)
        {
            s+=tree[i][j];
            j-=j&(-j);
        }
        i-=i&(-i);
    }
    return s;
}
```

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。

```
//输入一个n*n的矩阵，区间更新，查询某点的值
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=1e3+5;
int tree[maxn][maxn];

int n;
void add(int x,int y,int z)
{
    int i=x,j=y;
    while(i<=n)
    {
        j=y;
        while(j<=n)
        {
            tree[i][j]+=z;
            j+=j&(-j);
        }
        i+=i&(-i);
    }
}

void rang_add(int x1,int y1,int x2,int y2,int z)
{
    add(x1,y1,z);
    add(x1,y2+1,-z);
    add(x2+1,y1,-z);
    add(x2+1,y2+1,z);
}

int ask(int x,int y)
{
```

```
    int i=x,j=y;
    int s=0;
    while(i)
    {
        j=y;
        while(j)
        {
            s+=tree[i][j];
            j-=j&(-j);
        }
        i-=i&(-i);
    }
    return s;
}

int main()
{

       cin>>n;
       for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
       {
           int e;
           cin>>e;
           rang_add(i,j,i,j,e);
       }
    int x1,y1,x2,y2,z;
    cin>>x1>>y1>>x2>>y2>>z;
    rang_add(x1,y1,x2,y2,z);
    cout<<ask(2,3);
   return 0;

}


```

3. 区间修改+区间查询

```
//5*5矩阵区间更新+区间查询历程
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#include <iostream>
using namespace std;
```

```cpp
typedef long long ll;
const int N = 205;
ll n, m, Q;
ll t1[N][N], t2[N][N], t3[N][N], t4[N][N];
void add(ll x, ll y, ll z)
{
    for(int X = x; X <= n; X += X & -X)
        for(int Y = y; Y <= m; Y += Y & -Y)
        {
            t1[X][Y] += z;
            t2[X][Y] += z * x;
            t3[X][Y] += z * y;
            t4[X][Y] += z * x * y;
        }
}
void range_add(ll xa, ll ya, ll xb, ll yb, ll z)  //(xa, ya) 到 (xb, yb) 的矩形
{
    add(xa, ya, z);
    add(xa, yb + 1, -z);
    add(xb + 1, ya, -z);
    add(xb + 1, yb + 1, z);
}
ll ask(ll x, ll y)
{
    ll res = 0;
    for(int i = x; i; i -= i & -i)
        for(int j = y; j; j -= j & -j)
            res += (x + 1) * (y + 1) * t1[i][j]
                   - (y + 1) * t2[i][j]
                   - (x + 1) * t3[i][j]
                   + t4[i][j];
    return res;
}
ll range_ask(ll xa, ll ya, ll xb, ll yb)
{
    return ask(xb, yb) - ask(xb, ya - 1) - ask(xa - 1, yb) + ask(xa - 1, ya - 1);
}
int main()
{
    n=5,m=5;
    for(int i=1; i<=n; i++)
        for(int j=1; j<=n; j++)
        {
            int z;
```

```
            cin>>z;
            range_add(i,j,i,j,z);
        }

    int x1,x2,y1,y2,z;
    cin>>x1>>y1>>x2>>y2>>z;
    range_add(x1,y1,x2,y2,z);
    cout<<range_ask(1,1,5,5);
    return 0;
}

```