Chapter 1

数据结构

1.1 哈希

HDU 5918

给定元素个数分别为 $n \leq 10^6$ 和 $m \leq 10^6$ 的数组 $A[]$ 和 $B[]$(下标从 1 开始) 和一个常数 $p \leq 10^6$，求在数组 $A[]$ 中满足 $A[q] = B[1], A[q+p] = B[2], A[q+2*p] = B[3]..., A[q+(m-1)*p] = B[m]$ 的位置 $q(1 \leq q, q+(m-1)*p \leq n)$ 的个数。

```
const ll mod1 = 1034128911ll;
const ll mod2 = 1032728711ll;
const ll prime1 = 95734711;
const ll prime2 = 132134911;
const int MAX_N = 1000010;

int T, n, m, p, cases = 0;
int A[MAX_N], B[MAX_N];

struct Hash {
    ll a, b;
} hesh[MAX_N];

void solve()
{
    ll ret1 = 0, ret2 = 0;
    ll pw1 = 1, pw2 = 1;
    for (int i = 1; i <= m; ++i) {
        ret1 = (ret1 * prime1 % mod1 + B[i]) % mod1;
        if (i > 1) pw1 = pw1 * prime1 % mod1;

        ret2 = (ret2 * prime2 % mod2 + B[i]) % mod2;
        if (i > 1) pw2 = pw2 * prime2 % mod2;
    }
    for (int i = 1; 1ll * (m - 1) * p + i <= n; ++i) {
        if (i <= p) {
            hesh[i].a = hesh[i].b = 0;
            for (int j = 0; j < m; ++j) {
                int pos = i + j * p;
                hesh[i].a = (hesh[i].a * prime1 % mod1 + A[pos]) % mod1;
                hesh[i].b = (hesh[i].b * prime2 % mod2 + A[pos]) % mod2;
            }
        } else {
            hesh[i].a = (hesh[i - p].a - (ll)A[i - p] * pw1 % mod1) * prime1 % mod1;
            hesh[i].a = (hesh[i].a + A[i + (m - 1) * p]) % mod1;
            if (hesh[i].a < 0) hesh[i].a += mod1;

            hesh[i].b = (hesh[i - p].b - (ll)A[i - p] * pw2 % mod2) * prime2 % mod2;
            hesh[i].b = (hesh[i].b + A[i + (m - 1) * p]) % mod2;
            if (hesh[i].b < 0) hesh[i].b += mod2;
        }
```

```
42      }
43      int ans = 0;
44      for (int i = 1; 1ll * (m - 1) * p + i <= n; ++i) {
45          if (hesh[i].a == ret1 && hesh[i].b == ret2) ans++;
46      }
47      printf("Case #%d: %d\n", ++cases, ans);
48  }
49
50  int main()
51  {
52      scanf("%d", &T);
53      while (T--) {
54          scanf("%d%d%d", &n, &m, &p);
55          for (int i = 1; i <= n; ++i) {
56              scanf("%d", &A[i]);
57          }
58          for (int i = 1; i <= m; ++i) {
59              scanf("%d", &B[i]);
60          }
61          solve();
62      }
63      return 0;
64  }
```

1.2 并查集

1.2.1 加权并查集

n 表示 n 个数字，编号 1–n，然后有 m 个区间 $[l,r]$ 和该区间和 s，问在这 m 个区间中有多少个区间和是不正确的？如果不正确就忽略该区间和，否则将该区间和作为已知条件使用。

需要一个数组 val,$val[i]$ 表示 i 到根节点的距离，然后就是在查找根节点的过程更新路径上结点的 val,和在 mix 函数里判断该区间和是否有效。l 到 r 之间的和为 s 可以理解为 l 到 r 的距离为 s。

```
const int maxn = 200010;

int val[maxn], pre[maxn];
int n, m, u, v, w;
//val[i]:i 到根节点的距离;  pre[i]:i 的父节点

int find(int x)
{
    if (pre[x] == x) return x;
    int tmp = find(pre[x]); // tmp 是 x 的根节点
    val[x] = val[x] + val[pre[x]];
    // 在递归时 x 还未连接到根节点上，只连接到父节点上，
    // 所以这时的 val[x] 实际上是到父节点的距离
    // 而 val[pre[x]] 是父节点到根节点的距离，
    // 所以真正的 val[x]=val[x]+val[pre[x]], 左边的 val[x] 是到根节点距离，右边的是到父节点距离
    // 可以从递归倒数第二层往前想
    return pre[x] = tmp;//路径压缩
}

int mix(int x, int y, int z)
{
    int fx = find(x), fy = find(y);
    if (fx != fy) {
        pre[fx] = fy; // 将 x 的根节点 fx 的父节点设为fy
        val[fx] = val[y] - val[x] + z;
        // x 到 y 的距离是 z ， x 到 fx 的距离是 val[x] ， y 到 fy 的距离是 val[y]
        // 那么 fx 到 fy 的距离 val[fx]=x 到 fy 的距离（ z+val[y])- x 到 fx 的距离 val[x]
        return 0;
    } else {
        if (abs(val[y] - val[x]) == z) return 0; // 必须是绝对值才行
        // 因为尽管 y>x ，但是到根节点的距离不确定大小（到根结点路径上个点的值大小不确定）
        return 1;
    }
}

int main()
{
   while (~scanf("%d%d", &n, &m)){
   //因为接下来由 u— 的存在，所以要从开始0
        for (int i = 0; i <= n; i++)
            pre[i] = i;
        memset(val, 0, sizeof(val));
        int ans = 0;
        for (int i = 1; i <= m; i++) {
            scanf("%d%d%d", &u, &v, &w);
            u--; // 这样做符合 val 的含义
            ans += mix(u, v, w);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.2.2 分层并查集

有 n 个动物，编号 1–n，每个动物属于 A,B,C 三种中的一种，并且 A 种动物吃 B 种动物，B 种动物吃 C 种动物，C 种动物吃 A 种动物。有 K 条语句。格式是：d, x, y。

- $d = 1$ 时表示 x 和 y 是属于同一种类
- $d = 2$ 时表示 x 吃 y

这条语句如果满足下列条件之一就是错误的语句：

- 当前的话与前面的某些真的话冲突，就是假话；
- 当前的话中 X 或 Y 比 N 大，就是假话；
- 当前的话表示 X 吃 X，就是假话。

问 K 条语句中一共有多少条语句是错误的？

在数组 $pre[3 * maxn]$ 中，1–n 表示种类 A，$n+1$–$2*n$ 表示种类 B，$2*n+1$–$3*n$ 表示种类 C。
当 $d = 1$ 时：先检查 x 和 y 是否属于不同的种类，如果是不同种类，则 $ans++$，否则就将 x，y 同属于 A,B,C 依次 mix；
当 $d = 2$ 时：先检查 x 和 y 是否属于同一种类和 y 吃 x 的情况，如果是的，则 $ans++$，否则就将 x 属于 A，y 属于 B；x 属于 B，y 属于 C；x 属于 C，y 属于 A 三种情况 mix

```
const int maxn = 50010;

int n, k, d, x, y;
int pre[3 * maxn];

int find(int x)
{
    return pre[x] == x ? x : pre[x] = find(pre[x]);
}

void mix(int x, int y)
{
    int fx = find(x), fy = find(y);
    if (fx != fy){
        pre[fx] = fy;
    }
}

int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= 3 * n; i++)
        pre[i] = i;
    int ans = 0;
    for (int i = 0; i < k; i++){
        scanf("%d%d%d", &d, &x, &y);
        if (x > n || y > n || (d == 2 && x == y)){
            ans++;
            continue;
        }
        if (d == 1){
            if (find(x) == find(y + n) || find(y) == find(x + n)) ans++;
            // 因为当 x 和 y 属于不同种类时无非是 x 吃 y 或者 y 吃 x
            // 而且每种捕食关系的三种种类归属情况同时入队列，
            //所以检查 x 和 y 属于不同种类就是检查 x 和 y+n 与 x+n 和 y 是否属于同一集合
            // （前者是 x 吃 y ，后者是 y 吃 x ）
            else {
                mix(x, y); // 同属A
                mix(x + n, y + n); // 同属 B
                mix(x + 2 * n, y + 2 * n); // 同属 C
            }
        }else {
```

```
            if (find(x) == find(y) || find(y) == find(x + n)) ans++;
            //前者是检查 x 和 y 是否属于同一种类，后者是检查是否是 y 吃 x
            else{
                mix(x, y + n); //x 属于 A , y 属于 B
                mix(x + n, y + 2 * n); // x 属于 B , y 属于 C
                mix(x + 2 * n, y); // 属于x C , y 属于 A
            }
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

有 n 个数字，每个数字非 0 即 1，有 m 条语句，每条语句：$l, r, even/odd$，表示 l 到 r 区间上有奇/偶个 1. 问最多前多少条语句是正确的？

用 $val[i]$ 表示从 i 到根节点路径上含有 1 的数量的奇偶性。在寻找根节点的同时更新路径上的 val。
比较麻烦的是数据范围:$n <= 1000000000$ $m <= 5000$.
1. 可以用 map 来标记各个读入点的次序，相当于有个代号，在 map 里没读入的话就添加进 map
在 $find()$ 和 $mix()$ 函数里相当于是对代号的操作，代号具有唯一性。
由于 $m <= 5000$，所以最多会读入 10000 个相异的数，那么对于 pre 数组和 val 数组都是可以接受的了。
2.$hash$ 思路。同样是代号的想法，把 l(或 r) mod maxn 值相同的归在一类, 用 $head$ 数组的下标记录这类，head 的数值表示最后一类的读入位置。和最短路里链式前向星的查找类似。

```
const int maxn=10010;

int pre[maxn],val[maxn],n,m,l,r,w;
char s[10];

int find(int x)
{
    if(pre[x]==x) return x;
    int tmp=find(pre[x]);
    val[x]=(val[x]+val[pre[x]])%2;
    return pre[x]=tmp;
}

int mix(int x,int y,int z)
{ // 语句正确， mix 返回 1 ，否则返回0
    int fx=find(x);
    int fy=find(y);
    if(fx!=fy)
    {
        pre[fx]=fy;
        val[fx]=(val[y]-val[x]+z)%2;
        return 1;
    }
    else
    {
        if(abs(val[x]-val[y])%2==z) return 1;
        return 0;
    }
}

int main()
{
    while(~scanf("%d",&n)&&n){
        for(int i=0;i<maxn;i++){
            pre[i]=i;
            val[i]=0;
        }
        scanf("%d",&m);
        int ok=0;
        int ans=0;
        int index=0;
```

```
        map<int,int> mp;
        for(int i=1;i<=m;i++) {
            scanf("%d%d%s",&l,&r,s);
            if(s[0]=='e') w=0;
            else w=1;
            l--; // 这样做就可以合并相邻的区间，例如：读入 l=1,r=2 和 l=3,r=4
            if(mp.find(l)==mp.end()) mp[l]=index++;
            // find() 函数返回一个迭代器指向键值为 key 的元素
            // 如果没找到就返回指向 map 尾部的迭代器
            if(mp.find(r)==mp.end()) mp[r]=index++;
            // 假设前 x 条语句是正确的，第 x+1 条是错的
            // 那么前 x 次读入 ok 都为 0 ，第 x+1 次读入由于 ok=0,mix() 返回0
            // 所以执行 else 语句， ok 变为 1 ， ans=i-1=(x+1)-1=x
            // 从 x+2 开始由于 ok=1 ，不会执行 || 后面的判断，恒continue.
            if(ok||mix(mp[l],mp[r],w)) continue;
            else ok=1;
            if(ok) ans=i-1;
        }
        if(ok==0) ans=m; // 所有的语句都是正确的
        printf("%d\n",ans);
    }
    return 0;
}
```

平面上有 n 个点，每个点可以在东西南北四个方向上与另外一个点连接，（每个点最多和四个直接连通）然后有 m 条语句用以表示这 n 个点之间的位置关系：a,b,d,S 表示 a 点在 b 点南方，距离为 $d(N,W,E$ 分别表示北，西，东）接着有 k 条语句来查询：$a\ b\ t$：由 m 条语句中的前 t 条能否得出 $a\ b$ 两点间的哈夫曼距离，如果由 m 条语句中的前 t 条得出 $a\ b$ 两点不连通，则结果为 -1.

先查询按照查询的 t 由小到大排序，然后将读入数据读入并查集到相应次序，两个点是否连通可通过 $find$ 来判断。$x[i]\ y[i]$ 分别表示点 i 到根节点在横向和竖向的距离。
那么如果 uu,vv 两点连通距离就是 $abs(x[uu]-x[vv])+abs(y[uu]-y[vv])$; 但是输出需要按照查询的顺序输出，所以可以把查询语句用结构体存储，并且结构体中一个变量用于存储查询顺序，那么计算每条语句的 ans 时，把查询语句结构体数组按照 t 排序，输出时，再把查询语句按照查询顺序排序输出。

```
const int maxn=40010;
const int maxk=10010;

int pre[maxn],x[maxn],y[maxn],n,m,k,a,b,d,tx,ty;
char ss[10];

struct Query{
    int index,u,v,t,ans;
    // ans 是该条查询语句的答案
}query[maxk];

struct Read{
    int u,v,xx,yy;
}read[maxn];

bool cmp1(Query Q1,Query Q2)
{ //按查询语句的 t 排序
    if(Q1.t==Q2.t) return Q1.index<Q2.index;
    return Q1.t<Q2.t;
}

bool cmp2(Query Q1,Query Q2)
{ //按查询语句的查询顺序排序
    return Q1.index<Q2.index;
}

int find(int u)
{
    if(pre[u]==u) return u;
    int tmp=find(pre[u]);
```

```
    x[u]=x[u]+x[pre[u]];
    y[u]=y[u]+y[pre[u]];
    return pre[u]=tmp;
}

void mix(int a,int b,int tx,int ty)
{
    int fa=find(a);
    int fb=find(b);
    if(fa!=fb){
        pre[fa]=fb;
        x[fa]=x[b]-x[a]+tx;
        y[fa]=y[b]-y[a]+ty;
    }
}

int main()
{
    while(~scanf("%d%d",&n,&m)&&n){
        for(int i=0;i<=n;i++)
            pre[i]=i;
        memset(x,0,sizeof(x));
        memset(y,0,sizeof(y));
        for(int i=1;i<=m;i++){
            scanf("%d%d%d%s",&a,&b,&d,ss);
            if(ss[0]=='E'){tx=d;ty=0;} // 这里对方向做了统一规定
            else if(ss[0]=='W'){tx=-d;ty=0;}
            else if(ss[0]=='S'){tx=0;ty=d;}
            else if(ss[0]=='N'){tx=0;ty=-d;}
            read[i].u=a;
            read[i].v=b;
            read[i].xx=tx;
            read[i].yy=ty;
        }
        scanf("%d",&k);
        for(int i=1;i<=k;i++){
            scanf("%d%d%d",&query[i].u,&query[i].v,&query[i].t);
            query[i].ans=0;
            query[i].index=i; //查询顺序标记
        }
        sort(query+1,query+k+1,cmp1);
        int now=1;
        for(int i=1;i<=k;i++){
            for(int j=now;j<=query[i].t;j++){
                a=read[j].u;
                b=read[j].v;
                tx=read[j].xx;
                ty=read[j].yy;
                mix(a,b,tx,ty);
            }
            int uu=query[i].u;
            int vv=query[i].v;
            if(find(uu)==find(vv)) // uu 和 vv 连通{
                query[i].ans=abs(x[uu]-x[vv])+abs(y[uu]-y[vv]);
            }
            else query[i].ans=-1;

            now=query[i].t+1;
        }
        sort(query+1,query+1+k,cmp2);
        for(int i=1;i<=k;i++)
            printf("%d\n",query[i].ans);
    }
    return 0;
}
```

有 n 个人玩石头剪刀布，有且只有一个裁判。除了裁判每个人的出拳形式都是一样的。

- $a < b$ 表示 b 打败 a
- $a = b$ 表示 a 和 b 出拳一样，平手
- $a > b$ 表示 a 打败 b

给出 m 个回合的游戏结果，问能否判断出谁是裁判？如果能还要输出是在哪个回合之后判断出谁是裁判。

枚举和加权并查集。
对于每个人假设其为裁判，然后去掉所有和他有关的匹配，判断是否会出现矛盾。

- $val[i] = 0$:i 和根节点属于同一集合
- $val[i] = 1$: 根节点打败 i
- $val[i] = 2$:i 打败根节点

在寻找根节点的 $find()$ 函数中，val 的更新函数是：$val[x] = (val[x] + val[pre[x]])\%3$
举个例子：找到根节点之前 $val[x] = 1$，$val[pre[x]] = 2$：表示父节点打败 x，父节点也打败父节点的父节点。（注意此时 $val[x]$ 是 x 与父节点的关系）所以按照递归的思路，从递归倒数第二层开始 $pre[x]$ 就表示为根节点了，那么 $pre[x]$ 打败根节点。又因为 $pre[x]$ 也打败 x 所以 $val[x] = 0 = (1 + 2)\%3$; 递归在往上一层时 $pre[x]$ 又表示为根节点了。
再来看看合并操作时的 val 关系。fa, fb 分别为 aa, bb 的根节点，ww 是 aa 与 bb 的关系。
当 $fa = fb$ 时，令 $pre[fa] = fb$. 假设 $val[aa] = 1$, 即 fa 打败 aa...(1),$val[bb] = 2$，即 bb 打败 fb...(2)，$ww = 1$，即 bb 打败 aa...(3)。则由 (1)(2) 的 aa 和 fb 是同一集合。再由 (1) 得 fa 打败 fb。即 $val[fa] = 2$. 也就是 $val[fa] = (val[bb] - val[aa] + ww)\%3$, 但是由于有可能 $val[bb] - val[aa] + ww < 0$，所以正确的方程是：$val[fa] = (val[bb] - val[aa] + ww + 3)\%3$. 当 $fa == fb$ 时，那么就要判断是否出现矛盾，如果出现矛盾那么说明 i 不能作为裁判。判断矛盾是：$(val[aa] - val[bb] + 3)\%3$ 和 ww 是否相等。如果不矛盾，那么就接着读入输入到最后。

还要注意一点就是可能会出现多个裁判，那就是 Can not determine。

```
const int maxn = 510;
const int maxm = 2010;

int n,m,aa,bb,ww;
int pre[maxn],val[maxn];
int line,tmpline, ans,cnt,flag;
char s;

struct Read {
    int a, b, w;
}read[maxm];

void init()
{
    for (int i = 0; i <maxn; i++){
        pre[i] = i;
        val[i]=0;
    }
}

int find(int x)
{
    if(pre[x]==x) return x;
    int tmp=find(pre[x]);
    val[x]=(val[x]+val[pre[x]])%3;
    return pre[x]=tmp;
}

int main()
{
    while (~scanf("%d%d", &n, &m)){
        if (m == 0){
            //printf("case 1\n");
```

```
34              if(n==1)
35      printf("Player 0 can be determined to be the judge after 0 lines\n");
36              else printf("Can not determine\n");
37              continue;
38          }
39          for (int i = 0; i<m; i++){
40              scanf("%d%c%d", &aa, &s, &bb);
41              read[i].a=aa;
42              read[i].b=bb;
43              if (s == '=') read[i].w=0;
44              else if(s=='<') read[i].w=1;
45              else if(s=='>') read[i].w=2;
46          }
47          line=-1;
48          cnt=0;
49          for(int i=0;i<n;i++)//枚举每个人{
50              init();
51              tmpline=-1;
52              flag=0;
53              for(int j=0;j<m;j++) {
54                  aa=read[j].a;
55                  bb=read[j].b;
56                  ww=read[j].w;
57                  if(aa==i||bb==i) continue;
58                  //去掉的影响看是否还会出现矛盾i
59                  int fa=find(aa);
60                  int fb=find(bb);
61                  if(fa!=fb){
62                      pre[fa]=fb;
63                      val[fa]=(val[bb]-val[aa]+ww+3)%3;
64                  } else{
65                      if((val[aa]-val[bb]+3)%3!=ww)//出现矛盾{
66                          tmpline=j+1;
67                          //出现矛盾所在行
68                          flag=1;
69                          break;
70                      }
71                  }
72                  if(flag) break;
73              }
74              if(flag==0)//没出现矛盾{
75                  ans=i;//可以是裁判i
76                  cnt++;
77                  if(cnt>=2) break;//裁判数量 >=2
78              }
79              else line=max(line,tmpline);//
80          }
81          if(cnt==0){
82              //printf("case 2\n");
83              printf("Impossible\n");
84          }else if(cnt>=2){
85              //printf("case 3\n");
86              printf("Can not determine\n");
87          }else{
88              //printf("case 4\n");
89   printf("Player %d can be determined to be the judge after %d lines\n",ans,line);
90          }
91      }
92      return 0;
93  }
```

1.3 *RMQ*

用于解决区间最值问题，需要 $O(n \log n)$ 的预处理。

```
int n;
int data[MAX_N], dp[MAX_N][20];

void RMQ()
{
    memset(dp, 0, sizeof(dp));
    int k = (int)log2(n * 1.0);
    for (int i = 0; i < n; ++i) { dp[i][0] = data[i]; }
    for (int j = 1; j <= k; ++j) { // 注意枚举顺序
        for (int i = 0; i + (1 << j) - 1 < n; ++i) {
        //一定要 -1 ，因为从 i 开始的第 1<<j 个元素下标是 i + (1 << j) - 1
            int k = i + (1 << (j - 1));
            dp[i][j] = max(dp[i][j - 1], dp[k][j - 1]);
        }
    }
}

// 查询区间 [left, right] 的最值
int len = right - left + 1;
int e = (int)log2(len * 1.0);
int k = right - (1 << e);
res = max(dp[left][e], dp[k][e]);
```

1.4 线段树

1.4.1 矩形并的周长

```
#define lson(x) (x<<1)
#define rson(x) ((x<<1)|1)
const int maxn = 5050;
// maxn 是最大矩形数

int n, x1, x2, yy1, y2;
int xx[maxn << 2];

struct SegTree{
    int left, right, len, num, flag;
    // len 是有效长度， num 是有几条线段, flag 记录状态
    bool lcover, rcover;
    // 记录区间左右端点是否存在竖边
}segtree[maxn << 4];

struct Line{
    int x1, x2, y, flag;
    bool operator < (const Line a) const{
        return y < a.y;
    }
}line[maxn << 2];

void build(int left, int right, int cur)
{
    segtree[cur].left = left;
    segtree[cur].right = right;
    segtree[cur].len = segtree[cur].num = segtree[cur].flag = 0;
    segtree[cur].lcover = segtree[cur].rcover = false;
    if(left + 1 == right) return ;
    int mid = (left+right) >> 1;
    build(left, mid, lson(cur));
    build(mid, right, rson(cur));
}

void calc_len(int cur)
{
    int left = segtree[cur].left;
    int right = segtree[cur].right;
    if(segtree[cur].flag > 0){
        segtree[cur].len = xx[right] - xx[left];
        segtree[cur].num = 1;
        segtree[cur].lcover = segtree[cur].rcover = true;
    }else if(left + 1 == right){
        segtree[cur].len = segtree[cur].num = 0;
        segtree[cur].lcover = segtree[cur].rcover = false;
    }else{
        segtree[cur].len = segtree[lson(cur)].len + segtree[rson(cur)].len;
        segtree[cur].num = segtree[lson(cur)].num + segtree[rson(cur)].num;
        segtree[cur].lcover = segtree[lson(cur)].lcover;
        segtree[cur].rcover = segtree[rson(cur)].rcover;
        if(segtree[lson(cur)].rcover && segtree[rson(cur)].lcover)
        // 左右儿子的线段可以衔接
            segtree[cur].num--;//合并成一个线段，所以线段数量-1
    }
    return;
}

void update(int a, int b, int flag, int cur)
{
    int left = segtree[cur].left;
    int right = segtree[cur].right;
```

```
62      if(left==a && right==b){
63          segtree[cur].flag += flag;
64          calc_len(cur);
65          return;
66      }
67      if(left + 1 == right) return;
68      int mid = (left + right) >> 1;
69      if(b <= mid) update(a, b, flag, lson(cur));
70      else if(a >= mid) update(a, b, flag, rson(cur));
71      else{
72          update(a, mid, flag, lson(cur));
73          update(mid, b, flag, rson(cur));
74      }
75      calc_len(cur);
76  }
77
78  int main()
79  {
80      while(~scanf("%d", &n)){
81          int tot = 0;
82          for(int i = 0; i < n; i++){
83              scanf("%d%d%d%d", &x1, &yy1, &x2, &y2);
84              line[tot].x1 = x1, line[tot].x2 = x2;
85              line[tot].y = yy1, line[tot].flag = 1;
86              line[tot+1].x1 = x1, line[tot+1].x2 = x2;
87              line[tot+1].y = y2, line[tot+1].flag = -1;
88              xx[tot] = x1, xx[tot+1] = x2;
89              tot += 2;
90          }
91          sort(xx, xx + tot);
92          int m = unique(xx, xx + tot) - xx;
93          build(0, m - 1, 1);
94          sort(line, line + tot);
95          int ans = 0, last = 0, a, b;
96          for(int i=0;i<tot-1;i++){
97              a=lower_bound(xx,xx+m,line[i].x1)-xx;
98              b=lower_bound(xx,xx+m,line[i].x2)-xx;
99              update(a,b,line[i].flag,1);
100             ans+=(line[i+1].y-line[i].y)*segtree[1].num*2;
101             //竖边数量是线段数量的两倍
102             ans+=abs(segtree[1].len-last);
103             / 因为加上边时/ flag 为 -1 ，这时线段树中这条边就不存在了
104             // 但是求周长时是应该加上的
105             // 而且在上一棵线段树中是有的，所以要用abs
106             last=segtree[1].len;
107         }
108         a=lower_bound(xx,xx+m,line[tot-1].x1)-xx;
109         b=lower_bound(xx,xx+m,line[tot-1].x2)-xx;
110         update(a,b,line[tot-1].flag,1);
111         ans+=abs(segtree[1].len-last);
112         printf("%d\n",ans);
113     }
114     return 0;
115 }
```

1.4.2 矩形交的面积

```
1  #define lson(x) (x<<1)
2  #define rson(x) ((x<<1)|1)
3  using namespace std;
4
5  const int maxn=5050;
6
7  int T,n;
```

```
8  double x1,x2,yy1,y2,xx[maxn<<2];
9
10 struct SegTree{
11     int left,right,flag;
12     double len1,len2;
13     // len1 是覆盖一次的长度， len2 是覆盖不止一次的长度
14 }segtree[maxn<<4];
15
16 struct Line{
17     double x1,x2,y;
18     int flag;
19     bool operator < (const Line a) const{
20         return y<a.y;
21     }
22 }line[maxn<<2];
23
24 inline void build(int left,int right,int cur)
25 {
26     segtree[cur].left=left;
27     segtree[cur].right=right;
28     segtree[cur].flag=0;
29     segtree[cur].len1=segtree[cur].len2=0;
30     if(left+1==right) return ;
31     int mid=(left+right)>>1;
32     build(left,mid,lson(cur));
33     build(mid,right,rson(cur));
34 }
35
36 inline void calc_len(int cur)
37 {
38     int left=segtree[cur].left;
39     int right=segtree[cur].right;
40     if(segtree[cur].flag>0) segtree[cur].len1=xx[right]-xx[left];
41     else if(left+1==right) segtree[cur].len1=0;
42     else segtree[cur].len1=segtree[lson(cur)].len1+segtree[rson(cur)].len1;
43     // flag>1 说明整个区间有不止一次覆盖, flag=1 说明整个区间是完全覆盖,
44     //但是在这个区间下可能有小区间之前已经被完全覆盖了，所以要加上子树中被覆盖一次的的区间长度
45     // flag<1 那么就是左右子树覆盖两次长度之和
46     if(segtree[cur].flag>1) segtree[cur].len2=xx[right]-xx[left];
47     else if(left+1==right) segtree[cur].len2=0;
48     // 叶子结点且 flag<=1 ，则覆盖两次的长度只能是0
49     else if(segtree[cur].flag==1)
50         segtree[cur].len2=segtree[lson(cur)].len1+segtree[rson(cur)].len1;
51     else segtree[cur].len2=segtree[lson(cur)].len2+segtree[rson(cur)].len2;
52
53     return;
54 }
55
56 inline void update(int a,int b,int flag,int cur)
57 {
58     int left=segtree[cur].left;
59     int right=segtree[cur].right;
60     if(left==a&&right==b){
61         segtree[cur].flag+=flag;
62         calc_len(cur);
63         return;
64     }
65     if(left + 1 == right) return;
66     int mid=(left+right)>>1;
67     if(b<=mid) update(a,b,flag,lson(cur));
68     else if(a>=mid) update(a,b,flag,rson(cur));
69     else{
70         update(a,mid,flag,lson(cur));
71         update(mid,b,flag,rson(cur));
72     }
```

```
    calc_len(cur);
}

int main()
{
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        int tot=0;
        for(int i=0;i<n;i++){
            scanf("%lf %lf %lf %lf",&x1,&yy1,&x2,&y2);
            line[tot].x1=x1,line[tot].x2=x2;
            line[tot].y=yy1,line[tot].flag=1;
            line[tot+1].x1=x1,line[tot+1].x2=x2;
            line[tot+1].y=y2,line[tot+1].flag=-1;
            xx[tot]=x1,xx[tot+1]=x2;
            tot+=2;
        }
        sort(xx,xx+tot);
        int m=unique(xx,xx+tot)-xx;
        build(0,m-1,1);
        sort(line,line+tot);
        double ans=0;
        int a,b;
        for(int i=0;i<tot-1;i++){
            a=lower_bound(xx,xx+m,line[i].x1)-xx;
            b=lower_bound(xx,xx+m,line[i].x2)-xx;
            update(a,b,line[i].flag,1);
            //printf("i=%d len=%.2f\n",i,segtree[1].len2);
            ans+=segtree[1].len2*(line[i+1].y-line[i].y);
        }
        printf("%.2f\n",ans);
    }
    return 0;
}
```

1.4.3 矩形并面积

```
#define lson(x) (x<<1)
#define rson(x) ((x<<1)|1)
using namespace std;

const int maxn=110;

int n,cases=0;
double x1,x2,yy1,y2,xx[maxn<<2];

struct SegTree{
    int left,right,flag;
    // 只有 flag=1 时该区间才会完全是有效长度
    double len;
}segtree[maxn<<4];

struct Line{
    int flag;
    double x1,x2,y;
    bool operator < (const Line a) const{
        return y<a.y;
    }// 将横边按所在 y 值从小到大排序
}line[maxn<<2];

void build(int left,int right,int cur)
{
    segtree[cur].left=left;
```

```
    segtree[cur].right=right;
    segtree[cur].flag=0;
    segtree[cur].len=0;// len 是区间所存的有效横边长度
    if(left+1==right) return;
    int mid=(left+right)>>1;
    build(left,mid,lson(cur));
    build(mid,right,rson(cur));
    // 如果是按照 [left,mid],[mid+1,right] 建树
    // 那么会有一段横坐标 [mid,mid+1] 丢失
}

void calc_len(int cur)
{
    int flag=segtree[cur].flag;
    int left=segtree[cur].left;
    int right=segtree[cur].right;
    if(flag) segtree[cur].len=xx[right]-xx[left];
    // 整个区间长度是区间左右端点所代表的横坐标之差
    else if(left+1==right) segtree[cur].len=0; // 叶子结点且 flag<=0
    else segtree[cur].len=segtree[lson(cur)].len+segtree[rson(cur)].len;
}

void update(int a,int b,int flag,int cur)
{
    int left=segtree[cur].left;
    int right=segtree[cur].right;
    if(left==a&&right==b){
        segtree[cur].flag+=flag;
        calc_len(cur);
        return;
    }
    if(left + 1 == right) return;
    int mid=(left+right)>>1;
    if(b<=mid) update(a,b,flag,lson(cur));
    else if(a>=mid){
        update(a,b,flag,rson(cur));
    }else {
        int mid=(left+right)>>1;
        update(a,mid,flag,lson(cur));
        update(mid,b,flag,rson(cur));
    }
    calc_len(cur);
}

int main()
{
    while((~scanf("%d",&n))&&n){
        int tot=0;
        for(int i=0;i<n;i++){
            scanf("%lf %lf %lf %lf",&x1,&yy1,&x2,&y2);
            xx[tot]=x1,xx[tot+1]=x2;
            line[tot].x1=x1,line[tot].x2=x2;
            line[tot].y=yy1,line[tot].flag=1;
            line[tot+1].x1=x1,line[tot+1].x2=x2;
            line[tot+1].y=y2,line[tot+1].flag=-1;
            tot+=2;
        }
        sort(xx,xx+tot);
        int m=unique(xx,xx+tot)-xx; // 将横坐标去重
        build(0,m-1,1);
        double ans=0;
        sort(line,line+tot);
        //for(int i=0;i<tot;i++)
        //  printf("%lf %lf %lf\n",line[i].x1,line[i].x2,line[i].y);
        for(int i=0;i<tot-1;i++){
```

```
92              int a=lower_bound(xx,xx+m,line[i].x1)-xx;
93              int b=lower_bound(xx,xx+m,line[i].x2)-xx;
94              update(a,b,line[i].flag,1);
95              ans+=segtree[1].len*(line[i+1].y-line[i].y);
96          }
97          printf("Test case #%d\n",++cases);
98          printf("Total explored area: %.2f\n\n",ans);
99      }
100     return 0;
101 }
```

1.5 树状数组

```
C[i] = a[i - 2^k + 1] + a[i - 2^k] + ... + a[i];
// k 是 i 二进制表示中末尾连续的 0 的个数

int lowbit(int x)
{
    return x & (-x); // 一个负数的二进制表示与其相反数的二进制之和等于2^32
    // return x & (x ^ (x - 1));
    // return x & (~(x - 1));
}
```

1.5.1 单点更新，区间求和

更新所有牵动的区间，求和时求所有区间的所有更新，每个区间覆盖若干子区间但是每个子区间的更新都会体现在父区间中。

定理：$A[k]$ 所牵动的序列为：$C[p_1], C[p_2]...$，其中 $p_1 = k, p_{i+1} = p_i + 2^{l_i} (i \geq 1, l_i$ 指 p_i 二进制中末尾 0 的个数)。

典型应用：求逆序对数。

1.5.2 区间更新，单点求值

比如区间 $[L, R]$ 要加上 $value$，此时数组 $C[]$ 存的是 C 所覆盖的区间改变值之和，那么查询所有覆盖 x 的 C。

区间更新和单点查询复杂度：$O(\log n)$。

1.5.3 一维区间更新和区间求和

C 中记录的是区间该变量，$query(x)$ 是 $data[x]$ 的该变量，设求 $[1, R]$ 区间和，原数据前缀和为 $pre[]$，则：

$$\begin{aligned} Ans(R) &= pre[R] + \sum_{i=1}^{R} query(i) \\ &= pre[R] + \sum_{i=1}^{i=R} C[i] * (R - i + 1) \\ &= pre[R] + (R+1) * \sum_{i=1}^{i=R} C[i] - \sum_{i=1}^{i=R} i * C[i] \end{aligned}$$

另开一个数组 $B[i] = i * C[i]$，记录并更新即可。

```
// POJ 3468
int n, m;
ll pre[MAX_N], B[MAX_N], C[MAX_N];

struct BIT {
    ll C[MAX_N], B[MAX_N];

    void init() {
        memset(B, 0, sizeof(B));
        memset(C, 0, sizeof(C));
    }
    int lowbit(const int& x) const {
        return x & (-x);
    }
    ll seg_query(const int& x) const {
```

```
        ll ret1 = 0, ret2 = 0;
        for (int i = x; i > 0; i -= lowbit(i)) {
            ret1 += C[i], ret2 += B[i];
        }
        return ret1 * (x + 1) - ret2;
    }
    void seg_update(const int& x, const ll& value)  {
        for (int i = x; i <= n; i += lowbit(i)) {
            C[i] += value, B[i] += value * x; // 注意这里更新 value * x
        }
    }
} bit;

int main()
{
    while (~scanf("%d%d", &n, &m)) {
        bit.init();
        for (int i = 1; i <= n; ++i) {
            ll t;
            scanf("%lld", &t);
            if (i == 1) pre[1] = t;
            else pre[i] = pre[i - 1] + t;
        }
        for (int i = 0; i < m; ++i) {
            char s[10];
            int L, R;
            ll t;
            scanf("%s", s);
            if (s[0] == 'Q') {
                scanf("%d%d", &L, &R);
                printf("%lld\n", bit.seg_query(R) -
                        bit.seg_query(L - 1) + pre[R] - pre[L - 1]);
            } else {
                scanf("%d%d%lld", &L, &R, &t);
                bit.seg_update(L, t);
                bit.seg_update(R + 1, -t);
            }
        }
    }
    return 0;
}
```

1.5.4 二维区间更新和区间求和

二维的区间更新和单点求值和一维的类似。

$$\begin{aligned}\sum_{i=1}^{i=x}\sum_{j=1}^{j=y} query(i,j) &= \sum_{i=1}^{i=x}\sum_{j=1}^{j=y} C[i][j]*(x-i+1)*(y-j+1)\\ &= \sum_{i=1}^{i=x}\sum_{j=1}^{j=y} C[i][j]*(x+1)*(y+1) - (y+1)*\sum_{i=1}^{i=x}\sum_{j=1}^{j=y} i*C[i][j]\\ &\quad -(x+1)*\sum_{i=1}^{i=x}\sum_{j=1}^{j=y} j*C[i][j] + \sum_{i=1}^{i=x}\sum_{j=1}^{j=y} i*j*C[i][j]\end{aligned}$$

```
// 支持区间更新和区间求和操作，测试BZOJ3132
struct BIT_2D { // 如果必要且空间允许注意开long long
    int row, col; // 矩阵大小
    int B[MAX_N][MAX_N], C[MAX_N][MAX_N], D[MAX_N][MAX_N], E[MAX_N][MAX_N];

    void init() {
        memset(B, 0, sizeof(B));
        memset(C, 0, sizeof(C));
```

```
        memset(D, 0, sizeof (D));
        memset(E, 0, sizeof (E));
    }
    int lowbit(const int& x) const {
        return x & (-x);
    }
    void seg_update(const int& x, const int& y, const int& value) {
        for (int i = x; i <= row; i += lowbit(i)) {
            for (int j = y; j <= col; j += lowbit(j)) {
                C[i][j] += value;
                B[i][j] += value * x;
                D[i][j] += value * y;
                E[i][j] += value * x * y;
            }
        }
    }
    int seg_query(const int& x, const int& y) const {
        int ret1 = 0, ret2 = 0, ret3 = 0, ret4 = 0;
        for (int i = x; i > 0; i -= lowbit(i)) {
            for (int j = y; j > 0; j -= lowbit(j)) {
                ret1 += C[i][j], ret2 += B[i][j];
                ret3 += D[i][j], ret4 += E[i][j];
            }
        }
        return ret1 * (x + 1) * (y + 1) - ret2 * (y + 1) - ret3 * (x + 1) + ret4;
    }
} bit;

// 矩阵左上角为 (a, b) 右下角为 (c, d) , 矩阵从上到下从左到右递增
// 查询
ans = bit.seg_query(c, d) + bit.seg_query(a - 1, b - 1);
ans -= (bit.seg_query(a - 1, d) + bit.seg_query(c, b - 1));
printf("%d\n", ans);
// 更新
scanf("%d", &value);
bit.seg_update(a, b, value);
bit.seg_update(c + 1, d + 1, value);
bit.seg_update(a, d + 1, -value);
bit.seg_update(c + 1, b, -value);
```

```
// CF341D: 子矩阵中每个元素都异或 value 和查询子矩阵所有元素异或和
int n, m;
ll C[4][MAX_N][MAX_N];

inline int lowbit(int x)
{
    return x & (-x);
}

int GetId(int x, int y)
{
    int ret = 0;
    if (x & 1) ret += 1;
    if (y & 1) ret += 2;
    return ret;
}

void update(int x, int y, ll value)
{
    int id = GetId(x, y);
    for (int i = x; i <= n; i += lowbit(i)) {
        for (int j = y; j <= n; j += lowbit(j)) {
            C[id][i][j] ^= value;
        }
    }
```

```
}

ll query(int x, int y)
{
    ll ret = 0;
    int id = GetId(x, y);
    for (int i = x; i > 0; i -= lowbit(i)) {
        for (int j = y; j > 0; j -= lowbit(j)) {
            ret ^= C[id][i][j];
        }
    }
    return ret;
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int id, a, b, c, d;
        ll value, ans;
        scanf("%d%d%d%d%d", &id, &a, &b, &c, &d);
        if (id == 1) {
            ans = query(c, d);
            ans ^= query(a - 1, d);
            ans ^= query(c, b - 1);
            ans ^= query(a - 1, b - 1);
            printf("%lld\n", ans);
        } else {
            scanf("%lld", &value);
            update(a, b, value);
            update(c + 1, d + 1, value);
            update(a, d + 1, value);
            update(c + 1, b, value);
        }
    }
    return 0;
}
```