1. 图论（待整理）

1.1. 最短路

1.1.1. 迪杰斯特拉（堆优化）

```
struct Dijkstra
{
    struct Edge
    {
        int to;
        LL cost;
        int nextt;
    } edge[maxn];
    int cnt = 0;
    int head[maxn];
    void init(int n)
    {
        V=n;
        fill(head,head+1+V,-1);
        cnt=0;
    }
    void add(int u, int v, LL w)
    {
        edge[cnt].to = v;
        edge[cnt].cost = w;
        edge[cnt].nextt = head[u]; //head[u]指向的是以该点为起点的边里面的最后一条边
        head[u] = cnt++;
    }
    typedef pair<LL, int> P; //first是最短距离，second是顶点的编号
    int V;
    LL d[maxn];
    void dijkstra(int s) //找从s点出发到所有点的最短路
    {
        priority_queue<P, vector<P>, greater<P>> que; //堆按照first从小到大排列
        fill(d, d + V + 1, INF);
        d[s] = 0;
        P temp;
        que.push(P(0, s));
        while (!que.empty())
        {
            P p = que.top();
            que.pop();
            int v = p.second;
```

```
            if (d[v] < p.first)
                continue;
            for (int i = head[v]; i != -1; i = edge[i].nextt)
            {
                if (d[edge[i].to] > d[v] + edge[i].cost)
                {
                    d[edge[i].to] = d[v] + edge[i].cost;
                    que.push(P(d[edge[i].to], edge[i].to));
                }
            }
        }
    }
} ;
```

1.1.2. 生成最短路地图

```
        //先运行一边迪杰斯特拉，然后再遍历每条边，运用d数组找到最短路地图
        for(int i=1;i<=V;i++)//顶点 1~n
        {
            for(int k=head[i];k!=-1;k=edge[k].nextt)
            {
                int to=edge[k].to;
                int cost=edge[k].cost;
                if(d[to]==d[i]+cost)
                {
                    AddEdge(i,to,cost);//给新图添加有向边
                }
            }
        }
```

1.1.3. 分层图最短路

第一种写法

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <queue>
#include <list>
#include <cstring>
#include <map>
#include <limits>
#include <cmath>
#define IN freopen("in.txt","r",stdin);
using namespace std;
typedef long long int ll;
const int maxn = 1e3+10;
const int INF = 0x3f3f3f3f;

struct State
```

```
{                   // 优先队列的结点结构体
    int v, w, cnt; // cnt 表示已经使用多少次免费通行权限
    State() {}
    State(int v, int w, int cnt) : v(v), w(w), cnt(cnt) {}
    bool operator<(const State &rhs) const { return w > rhs.w; }
};
struct node
{
    int v;
    int w;
    int next;
    /* data */
}edge[maxn*4];
priority_queue<State>pq;
int n,t,m,k,u,v,w,s;
int cnt;
bool vis[maxn][1005];
int dis[maxn][1005];
int head[maxn];

void add(int u,int v,int w){    //链式前向星存边
    edge[cnt] = {v,w,head[u]};
    head[u] = cnt++;
}

void dijkstra()
{
    memset(dis, 0x3f, sizeof(dis));
    dis[s][0] = 0;
    pq.push(State(s, 0, 0)); // 到起点不需要使用免费通行权，距离为零
    while (!pq.empty())
    {
        State top = pq.top();
        pq.pop();
        int u = top.v;
        int nowCnt = top.cnt;
        if (vis[u][nowCnt])
            continue;
        vis[u][nowCnt] = true;

        for (int i = head[u]; ~i; i = edge[i].next)
        {
            int v = edge[i].v, w = edge[i].w;
            if (nowCnt < k && dis[v][nowCnt + 1] > dis[u][nowCnt])
            { // 可以免费通行
                dis[v][nowCnt + 1] = dis[u][nowCnt];
                pq.push(State(v, dis[v][nowCnt + 1], nowCnt + 1));
            }
            if (dis[v][nowCnt] > dis[u][nowCnt] + w)
            { // 不可以免费通行
                dis[v][nowCnt] = dis[u][nowCnt] + w;
                pq.push(State(v, dis[v][nowCnt], nowCnt));
            }
        }
    }
}
```

```
int main()
{
    //IN;
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    memset(head,-1,sizeof (head));
    cin >> n >> m;
    cin >> s >> t>>k;
    while (m--)
    {
        cin >> u >> v >> w;
        add(u, v, w);
        add(v, u, w);
    }
    int ans = INF;
    dijkstra();
    for (int i = 0; i <= k; ++i)
        ans = min(ans, dis[t][i]); // 对到达终点的所有情况取最优值
    cout << ans << endl;
}
```

第二种写法

```
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <algorithm>
#include <queue>
#include <vector>
#define ll long long
#define inf 0x3f3f3f3f
#define pii pair<int, int>
const int mod = 1e9+7;
const int maxn = 6e6+7;
using namespace std;
struct node {int to,w,next;} edge[maxn];
int head[maxn], cnt;
int dis[maxn], vis[maxn];
int n, m, s, t, k;
struct Dijkstra
{
    void init()
    {
        memset(head,-1,sizeof(head));
        memset(dis,0x3f,sizeof(dis));
        memset(vis,0,sizeof(vis));
        cnt = 0;
    }

    void add(int u,int v,int w)
    {
        edge[cnt].to = v;
        edge[cnt].w = w;
```

```
        edge[cnt].next = head[u];
        head[u] = cnt ++;
    }

    void dijkstra()
    {
        priority_queue<pii,vector<pii>,greater<pii> > q;
        dis[s] = 0; q.push({dis[s],s});
        while(!q.empty())
        {
            int now = q.top().second;
            q.pop();
            if(vis[now]) continue;
            vis[now] = 1;
            for(int i = head[now]; i != -1; i = edge[i].next)
            {
                int v = edge[i].to;
                if(!vis[v] && dis[v] > dis[now] + edge[i].w)
                {
                    dis[v] = dis[now] + edge[i].w;
                    q.push({dis[v],v});
                }
            }
        }
    }
}dj;

int main()
{
    while(~scanf("%d%d%d", &n, &m, &s))
    {
        dj.init(); scanf("%d%d",&t,&k);
        while(m--)
        {
            int u, v, w;
            scanf("%d%d%d",&u, &v, &w);
            for(int i = 0; i <= k; i++)
            {
                dj.add(u + i * n, v + i * n, w);
                dj.add(v + i * n, u + i * n, w);
                if(i != k)
                {
                    dj.add(u + i * n, v + (i + 1) * n, 0);
                    dj.add(v + i * n, u + (i + 1) * n, 0);
                }
            }
        }
        dj.dijkstra(); int ans = inf;
        for(int i = 0; i <= k; i++)
            ans = min(ans, dis[t + i * n]);

        printf("%d\n",ans);
    }
}
```

1.2. 最小生成树

1.3. 网络流

1.3.1. 最大流最小割

有多个源点和多个汇点就建立一个超级源点和一个超级汇点！！！

权值在点上！！！ 普通的最大流是边上带权，但是这道题的权值在点上，所以我们需要拆点把点拆开，当作是经过这个点就要花费这么多权值。本题把花费当作流量，就是要求最小割，因为最小割等于最大流，所以就是求最大流。 拆点就是把该点变成$a \longrightarrow a'$，边的权值就是该点的花费，表示如果要走到这个点，则一定要有这样的花费，再把a到b的边变成$a' \longrightarrow b$，表示如果要从b到a的话必须要先经过b点（即已经到b'）。然后就是丢进dinic跑最大流。

路径还原 最终的残余网络中的反向边即为水流通过的路径！！！

```
struct Dinic
{
    struct Edge
    {
        int u, v;
        LL c;
        int next;
    } edge[20 * maxn];
    int n, m;
    int cnt;     //边数
    int p[maxn]; //父亲
    int d[maxn]; //深度
    int sp, tp; //原点，汇点
    void init(int V)//初始化函数，顶点下标 1~n
    {
        n=V;
        cnt=0;
        fill(p,p+V+1,-1);
    }
    void addedge(int u, int v, LL c)//调用此函数添加有向边
    {
        edge[cnt].u = u;
        edge[cnt].v = v;
        edge[cnt].c = c;
        edge[cnt].next = p[u];
        p[u] = cnt++;

        edge[cnt].u = v;
        edge[cnt].v = u;
        edge[cnt].c = 0;
        edge[cnt].next = p[v];
```

```
        p[v] = cnt++;
    }
    bool bfs()//从原点进行bfs，如果原点和汇点连通说明还有增广路
    {
        queue<int> q;
        memset(d, -1, sizeof(d));
        d[sp] = 0;
        q.push(sp);
        while (!q.empty())
        {
            int cur = q.front();
            q.pop();
            for (int i = p[cur]; i != -1; i = edge[i].next)
            {
                int u = edge[i].v;
                if (d[u] == -1 && edge[i].c > 0)
                {
                    d[u] = d[cur] + 1;
                    q.push(u);
                }
            }
        }
        return d[tp] != -1;
    }
    LL dfs(int a, LL b)
    {
        LL r = 0;
        if (a == tp)
            return b;
        for (int i = p[a]; i != -1 && r < b; i = edge[i].next)
        {
            int u = edge[i].v;
            if (edge[i].c > 0 && d[u] == d[a] + 1)
            {
                LL x = min(edge[i].c, b - r);
                x = dfs(u, x);
                r += x;
                edge[i].c -= x;
                edge[i ^ 1].c += x;
            }
        }
        if (!r)
            d[a] = -2;
        return r;
    }

    LL dinic(int sp, int tp)//调用此函数求最大流最小割
    {
        this->sp=sp;
        this->tp=tp;
        LL total = 0, t;
        while (bfs())
        {
            while (t = dfs(sp, INF))
                total += t;
        }
        return total;
```

```
    }
};
```

1.4. 最小费用流(迪杰)

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f;
struct edge {
    int to, cap, cost, rev;
    edge() {}
    edge(int to, int _cap, int _cost, int _rev) :to(to), cap(_cap), cost(_cost), rev(_rev) {}
};
int V, H[maxn + 5], dis[maxn + 5], PreV[maxn + 5], PreE[maxn + 5];
vector<edge> G[maxn + 5];
void init(int n) {
    V = n;
    for (int i = 0; i <= V; ++i)G[i].clear();
}
void AddEdge(int from, int to, int cap, int cost) {
    G[from].push_back(edge(to, cap, cost, G[to].size()));
    G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
}
int Min_cost_max_flow(int s, int t, int f, int& flow) {
    int res = 0; fill(H, H + 1 + V, 0);
    while (f) {
        priority_queue <pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > q;
        fill(dis, dis + 1 + V, inf);
        dis[s] = 0; q.push(pair<int, int>(0, s));
        while (!q.empty()) {
            pair<int, int> now = q.top(); q.pop();
            int v = now.second;
            if (dis[v] < now.first)continue;
            for (int i = 0; i < G[v].size(); ++i) {
                edge& e = G[v][i];
                if (e.cap > 0 && dis[e.to] > dis[v] + e.cost + H[v] - H[e.to]) {
                    dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
                    PreV[e.to] = v;
                    PreE[e.to] = i;
                    q.push(pair<int, int>(dis[e.to], e.to));
                }
            }
        }
        if (dis[t] == inf)break;
        for (int i = 0; i <= V; ++i)H[i] += dis[i];
        int d = f;
        for (int v = t; v != s; v = PreV[v])d = min(d, G[PreV[v]][PreE[v]].cap);
        f -= d; flow += d; res += d*H[t];
        for (int v = t; v != s; v = PreV[v]) {
            edge& e = G[PreV[v]][PreE[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
```

```
    }
    return res;
}
```

2. 计算几何（待学习）

几何基础： https://blog.csdn.net/linxilinxilinxi/article/details/81750327 计算几何：https://blog.csdn.net/linxilinxilinxi/article/details/81810944

3. Bitset

定义方法

```
bitset<8> b;
bitset<8> bit(8);
bitset<8> tp1(string("01010101"));
bitset<8> tp2(bit);

cout << "  b = " << b << endl;
cout << "bit = " << bit << endl;
cout << "tp1 = " << tp1 << endl;
cout << "tp2 = " << tp2 << endl;
/*
b = 00000000
bit = 00001000
tp1 = 01010101
tp2 = 00001000
*/
```

相关函数

```
bit.size();         //返回大小（长度）
bit.count();    //返回1的个数
bit.any();         //返回是否有1
bit.none();         //返回是否没有1
bit.set();         //全部置为1
bit.set(p);         //将p+1(下标从0开始!!)位置为1
bit.set(p,n);    //将p+1位置为n
bit.reset();    //全部置为0
bit.reset(p);    //将p+1位置为0
bit.flip();         //全部取反 等同于 (~bit)
bit.flip(p);    //将p+1位取反
bit.to_ulong();    //返回转换为 unsigned long  的结果，超范围会报错
```

```
    bit.to_ullong();//返回转换为 unsigned long long 的结果，超范围报错
    bit.to_string();//返回转换为 string 的结果
```

4. 极大团，最大团

4.1. 团的定义

对于给定图G=(V,E)。其中，V={1,...,n}是图G的顶点集，E是图G的边集。图G的团就是一个两两之间有边的顶点集合。简单地说，团是G的一个完全子图。如果一个团不被其他任一团所包含，即它不是其他任一团的真子集，则称该团为图G的极大团（maximal clique）。顶点最多的极大团，称之为图G的最大团（maximum clique）。最大团问题的目标就是要找到给定图的最大团。

4.2. 求最大团（原图的补图是二分图）

最大团问题和最大独立集问题是互补的问题，通过对原图建立补图，求解补图的最大独立集，然后顶点数减去最大独立集就是最大团

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define sc(x) scanf("%d", &x)
#define scc(x, y) scanf("%d%d", &x, &y)
#define sccc(x, y, z) scanf("%d%d%d", &x, &y, &z)
#define mkp(a, b) make_pair(a, b)
#define F first
#define ep(a) emplace_back(a)
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 10000;
const double pi = acos(-1.0);
int T, n, m;
int a[maxn];
vector<int> ans;
struct Dinic
{
    struct Edge
    {
```

```
    int u, v;
    LL c;
    int next;
} edge[20 * maxn];
int n, m;
int cnt;         //边数
int p[maxn];     //父亲
int d[maxn];     //深度
int sp, tp;      //原点，汇点
void init(int V) //初始化函数，顶点下标 1~n
{
    n = V;
    cnt = 0;
    fill(p, p + V + 1, -1);
}
void addedge(int u, int v, LL c) //调用此函数添加有向边
{
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].c = c;
    edge[cnt].next = p[u];
    p[u] = cnt++;

    edge[cnt].u = v;
    edge[cnt].v = u;
    edge[cnt].c = 0;
    edge[cnt].next = p[v];
    p[v] = cnt++;
}
bool bfs() //从原点进行bfs，如果原点和汇点连通说明还有增广路
{
    queue<int> q;
    memset(d, -1, sizeof(d));
    d[sp] = 0;
    q.push(sp);
    while (!q.empty())
    {
        int cur = q.front();
        q.pop();
        for (int i = p[cur]; i != -1; i = edge[i].next)
        {
            int u = edge[i].v;
            if (d[u] == -1 && edge[i].c > 0)
            {
                d[u] = d[cur] + 1;
                q.push(u);
            }
        }
    }
    return d[tp] != -1;
}
LL dfs(int a, LL b)
{
    LL r = 0;
    if (a == tp)
        return b;
    for (int i = p[a]; i != -1 && r < b; i = edge[i].next)
```

```
        {
            int u = edge[i].v;
            if (edge[i].c > 0 && d[u] == d[a] + 1)
            {
                LL x = min(edge[i].c, b - r);
                x = dfs(u, x);
                r += x;
                edge[i].c -= x;
                edge[i ^ 1].c += x;
            }
        }
        if (!r)
            d[a] = -2;
        return r;
    }

    LL dinic(int sp, int tp) //调用此函数求最大流最小割
    {
        this->sp = sp;
        this->tp = tp;
        LL total = 0, t;
        while (bfs())
        {
            while (t = dfs(sp, INF))
                total += t;
        }
        return total;
    }
} ge;
inline int judge(int tmp)
{
    int tot = 0;
    while (tmp)
    {
        if (tmp & 1)
            tot++;
        tmp >>= 1;
    }
    return tot;
}
int main()
{
    sc(n);
    {
        ans.clear();
        ge.init(n + 2);
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
        for (int i = 1; i <= n; i++)
        {
            int tmp = judge(a[i]);
            if (tmp % 2)
            {
                ge.addedge(0, i, 1);
            //  cout<<"bian"<<0<<" "<<a[i]<<endl;
```

```
            }
            else{
                ge.addedge(i, n + 1, 1);
            //  cout<<"bian"<<a[i]<<" "<<n+1<<endl;
            }
            for (int j = i+1; j<=n; j++)
            {
                int tm = judge(a[i] ^ a[j]);
                //cout<<i<<" J  "<<j<<" "<<tmp<<"    -----------\n";
                if (tm == 1)
                {
                    if (tmp % 2)
                    {
                        //cout<<"bian!"<<a[i]<<" "<<a[j]<<endl;
                        ge.addedge(i, j, 1);
                    }
                    else{
                        ge.addedge(j, i, 1);
                        //cout<<"bian#"<<a[j]<<" "<<a[i]<<endl;
                    }
                }
            }
        }
        int len = n - ge.dinic(0, n + 1);
        printf("%d\n", len);
        for (int i = 1; i <= n; ++i)
        {
            //cout<<'\n'<<d[i]<<" "<<i<<" "<<a[i]<<endl;
            if (ge.d[i] != -1 && (judge(a[i]) % 2))
            {
                printf("%d ", a[i]);
                //  cout<<"---"<<endl;
            }
            else if (ge.d[i] == -1 && !(judge(a[i]) % 2))
               printf("%d ", a[i]);
        }
    }
   // system("pause");
    return 0;
}
```

4.3. 二分图的一些知识

我们将这种两两不含端点的边集合M称为匹配，而元素最多的M则称为最大匹配。 当最大匹配数满足$2|M| = |V|$时，又称为完美匹配。特别的二分图中的匹配又称为二分图匹配。

实际上，可以将二分图最大匹配问题看成是最大流问题的一种特殊情况。可以对原图做如下变形。

将原图中的所有无向边e改成有向边，方向从U到V，容量为1。增加源点s和汇点t，从s向所有的顶点$u \in U$连一条容量为1的边，从所有的顶点$v \in V$向t连一条容量为1的边。

这样变形得到的新图G'中最大$s-t$流的流量就是原二分图G中的最大匹配的匹配数，而$U-V$之间流量为正的集合就是最大匹配。该算法的时间复杂度为$O(|V||E|)$

4.3.1. 一些概念

记图$G=(V,E)$

4.3.1.0.1. 匹配

在G中两两没有公共端点的边集合$M \subseteq E$

4.3.1.0.2. 边覆盖

G中的任意顶点都至少是F中某条边的端点的边集合$F \subseteq E$

4.3.1.0.3. 独立集

在G中两两互不相连的顶点集合$S \subseteq V$

4.3.1.0.4. 顶点覆盖

G中的任意边都有至少一个端点属于S的顶点集合$S \subseteq V$

还满足如下关系： （a）对于不存在孤立点的图，$|\text{最大匹配}|+|\text{最小边覆盖}|=|V|$ （b）$|\text{最大独立集}|+|\text{最小顶点覆盖}|=|V|$ （c）$|\text{最大匹配}|=|\text{最小顶点覆盖}|$ 对于二分图$G=(U \bigcup V,E)$,在通过最大流求解最大匹配所得到的参与网络中,令

$S=(\text{从}s\text{不可达的属于}U\text{的顶点}) \bigcup (\text{从}s\text{可达的属于}V\text{的顶点})$，则$S$就是$G$的一个最小顶点覆盖。

4.3.2. 二分图的Hall's marriage theorem定理

对于二分图（$U+V$，E）最大匹配 $|M|=|U|-max(|S|-|N(S)|)(S \subset U)$

5. 蒙特卡洛随机

5.1. 牛客暑期多校训练营（第二场）A（随机化）

现在有一个长度为未知环，每次你可以向前或者向后走一步。现在有T个回合，每个回合给你两个整数n和m。现在问你，在第i个回合中，在满足第i-1个回合的条件的前提下，在该回合中，将长度为n的环上的所有的点都访问过至少一次并最终落在点m的可能性。

于这类询问概率的问题，我们可以采用蒙特卡洛随机的方法进行随机分析（说白了也就是随机算法）。

我们可以模拟若干次题意的操作，并求解出大概的概率，根据归纳分析，我们发现，当m不等于0的情况下，答案与m的值无关，只与n的值有关且答案为1n−1。因此我们只需要用快速幂求解逆元即可，'''注意上一次回合的答案会累积到本回合'''。时间复杂度O(nlogn)。

需要注意的是，在windows系统下，随机函数rand()在数据较大的情况可能会发生比较大的误差。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int n,m;
int montekalo(int n){
    int cur=0,vis[10005],cnt=1;
    memset(vis,0,sizeof(vis));
    vis[0]=1;
    int tmp=0;
    while(cnt<n){
        int x=rand()%2;
        if(x){
            cur=(cur+1)%n;
        }
        else cur=(cur-1+n)%n;
        if(!vis[cur]){
            vis[cur]=1;
            cnt++;
        }
    }
    return cur;
}
void gen(int n,int m){
    int cnt=0,res=0;
    while(cnt<=100000){
        if(montekalo(n)==m) res++;
        cnt++;
    }
    printf("%d %d %.5f\n",res,cnt,1.0*res/cnt);
}
const int mod=1e9+7;
ll powmod(ll x,ll n){
    ll res=1;
    while(n){
        if(n&1) res=res*x%mod;
        x=x*x%mod;
        n>>=1;
    }
    return res%mod;
}
int main()
{
    int t;
```

```
    /*
     while(~scanf("%d%d",&n,&m)){
        gen(n,m);
    }*/
    scanf("%d",&t);
    ll res=1;
    while(t--){
        ll n,m;
        scanf("%lld%lld",&n,&m);
        if(n==1){
            printf("%lld\n",res);
            continue;
        }
        else if(m==0) res=0;
        else{
            res=res*powmod(n-1,mod-2)%mod;
        }
        printf("%lld\n",res);
    }
    return 0;
}
```

6. 随机数种子设置

```
srand((unsigned int)time(NULL));
```

7. 字符串

7.1. 字符串hash

hash

8. 莫队算法

莫队算法

8.1. 普通莫队算法

我们可以根据运算优先级的知识，把这个：

```
void add(int pos) {
    if(!cnt[aa[pos]]) ++now;
    ++cnt[aa[pos]];
}
void del(int pos) {
    --cnt[aa[pos]];
    if(!cnt[aa[pos]]) --now;
}
```

和这个：

```
while(l < ql) del(l++);
while(l > ql) add(--l);
while(r < qr) add(++r);
while(r > qr) del(r--);
```

硬生生压缩成这个：

```
while(l < ql) now -= !--cnt[aa[l++]];
while(l > ql) now += !cnt[aa[--l]]++;
while(r < qr) now += !cnt[aa[++r]]++;
while(r > qr) now -= !--cnt[aa[r--]];
```

例题:给定一个数组查询[l,r]中有多少不同的数字（也可以主席树做法）

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;

#define maxn 1010000
#define maxb 1010
int aa[maxn], cnt[maxn], belong[maxn];//数组的数值，每个数值的计数器
int n, m, size, bnum, now, ans[maxn];
struct query {
    int l, r, id;
} q[maxn];//查询区间

int cmp(query a, query b) {//区间排序函数
    return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belong[b.l] : ((belong[a.l] & 1) ? a.r
< b.r : a.r > b.r);
}
//输入输出挂
#define isdigit(x) ((x) >= '0' && (x) <= '9')
```

```
int read() {
    int res = 0;
    char c = getchar();
    while(!isdigit(c)) c = getchar();
    while(isdigit(c)) res = (res << 1) + (res << 3) + c - 48, c = getchar();
    return res;
}
void printi(int x) {
    if(x / 10) printi(x / 10);
    putchar(x % 10 + '0');
}
//输入输出挂结束

int main() {
    scanf("%d", &n);
    size = sqrt(n);//分块的数量
    bnum = ceil((double)n / size);
    for(int i = 1; i <= bnum; ++i)//分块
        for(int j = (i - 1) * size + 1; j <= i * size; ++j) {
            belong[j] = i;
        }
    for(int i = 1; i <= n; ++i) aa[i] = read();//读入数值数组
    m = read();
    for(int i = 1; i <= m; ++i) {
        q[i].l = read(), q[i].r = read();
        q[i].id = i;
    }
    sort(q + 1, q + m + 1, cmp);
    int l = 1, r = 0;
    for(int i = 1; i <= m; ++i) {//莫队算法核心代码
        int ql = q[i].l, qr = q[i].r;
        while(l < ql) now -= !--cnt[aa[l++]];
        while(l > ql) now += !cnt[aa[--l]]++;
        while(r < qr) now += !cnt[aa[++r]]++;
        while(r > qr) now -= !--cnt[aa[r--]];
        ans[q[i].id] = now;
    }
    for(int i = 1; i <= m; ++i) printi(ans[i]), putchar('\n');
    return 0;
}
```

8.2. 树上莫队算法(对欧拉序数组进行判断)

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
#define maxn 200200
#define isdigit(x) ((x) >= '0' && (x) <= '9')
inline int read() {
    int res = 0;
```

```
    char c = getchar();
    while(!isdigit(c)) c = getchar();
    while(isdigit(c)) res = (res << 1) + (res << 3) + (c ^ 48), c = getchar();
    return res;
}
int aa[maxn], cnt[maxn], first[maxn], last[maxn], ans[maxn], belong[maxn], inp[maxn],
vis[maxn], ncnt, l = 1, r, now, size, bnum; //莫队相关
int ord[maxn], val[maxn], head[maxn], depth[maxn], fa[maxn][30], ecnt;
int n, m;
struct edge {
    int to, next;
} e[maxn];
void adde(int u, int v) {
    e[++ecnt] = (edge){v, head[u]};
    head[u] = ecnt;
    e[++ecnt] = (edge){u, head[v]};
    head[v] = ecnt;
}
void dfs(int x) {
    ord[++ncnt] = x;
    first[x] = ncnt;
    for(int k = head[x]; k; k = e[k].next) {
        int to = e[k].to;
        if(to == fa[x][0]) continue;
        depth[to] = depth[x] + 1;
        fa[to][0] = x;
        for(int i = 1; (1 << i) <= depth[to]; ++i) fa[to][i] = fa[fa[to][i - 1]][i - 1];
        dfs(to);
    }
    ord[++ncnt] = x;
    last[x] = ncnt;
}
int getlca(int u, int v) {
    if(depth[u] < depth[v]) swap(u, v);
    for(int i = 20; i + 1; --i)
        if(depth[u] - (1 << i) >= depth[v]) u = fa[u][i];
    if(u == v) return u;
    for(int i = 20; i + 1; --i)
        if(fa[u][i] != fa[v][i]) u = fa[u][i], v = fa[v][i];
    return fa[u][0];
}
struct query {
    int l, r, lca, id;
} q[maxn];
int cmp(query a, query b) {
    return (belong[a.l] ^ belong[b.l]) ? (belong[a.l] < belong[b.l]) : ((belong[a.l] & 1) ?
a.r < b.r : a.r > b.r);
}
void work(int pos) {
    vis[pos] ? now -= !--cnt[val[pos]] : now += !cnt[val[pos]]++;
    vis[pos] ^= 1;
}
int main() {
    n = read(); m = read();
    for(int i = 1; i <= n; ++i)
        val[i] = inp[i] = read();
    sort(inp + 1, inp + n + 1);
```

```
    int tot = unique(inp + 1, inp + n + 1) - inp - 1;
    for(int i = 1; i <= n; ++i)
        val[i] = lower_bound(inp + 1, inp + tot + 1, val[i]) - inp;
    for(int i = 1; i < n; ++i) adde(read(), read());
    depth[1] = 1;
    dfs(1);
    size = sqrt(ncnt), bnum = ceil((double) ncnt / size);
    for(int i = 1; i <= bnum; ++i)
        for(int j = size * (i - 1) + 1; j <= i * size; ++j) belong[j] = i;
    for(int i = 1; i <= m; ++i) {
        int L = read(), R = read(), lca = getlca(L, R);
        if(first[L] > first[R]) swap(L, R);
        if(L == lca) {
            q[i].l = first[L];
            q[i].r = first[R];
        }
        else {
            q[i].l = last[L];
            q[i].r = first[R];
            q[i].lca = lca;
        }
        q[i].id = i;
    }
    sort(q + 1, q + m + 1, cmp);
    for(int i = 1; i <= m; ++i) {
        int ql = q[i].l, qr = q[i].r, lca = q[i].lca;
        while(l < ql) work(ord[l++]);
        while(l > ql) work(ord[--l]);
        while(r < qr) work(ord[++r]);
        while(r > qr) work(ord[r--]);
        if(lca) work(lca);
        ans[q[i].id] = now;
        if(lca) work(lca);
    }
    for(int i = 1; i <= m; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

9. 单调栈和单调队列

9.1. 单调队列（队尾插入，队首删除）

单调队列

单调队列一般是具有单调性队列

视具体题目而定，单调队列有单调递增和单调递减两种，一般来讲，队列的队首是整个队列的最大值或最小值

单调队列实现的大致过程： 1、维护队首（对于上题就是如果队首已经是当前元素的m个之前，则队首就应该被删了,head++） 2、在队尾插入（每插入一个就要从队尾开始往前去除冗杂状态，保持单调性）

> 实现一般采用双端队列

代码待补

```
#include<bits/stdc++.h>
using namespace std;
const int MAX=3050;
int n,m,a,b,x,y,z;
long long g[9000050],h[MAX][MAX];
long long minn[MAX][MAX];

int main()
{
    cin>>n>>m>>a>>b;
    cin>>g[0]>>x>>y>>z;
    for(int i=1;i<=n*m;i++)
    g[i]=(g[i-1]*x+y)%z;
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
    h[i][j]=g[(i-1)*m+j-1];
    for(int i=1;i<=n;i++)
    {
        deque<int> dq;
        for(int j=1;j<=m;j++)
        {
            while(!dq.empty() && h[i][j]<=h[i][dq.back()])//队列单调递增
            dq.pop_back();
            dq.push_back(j);
            if(!dq.empty() && j-dq.front()+1>b)//去除冗余元素（视题目而定）
            dq.pop_front();
            minn[i][j]=h[i][dq.front()];
            //获取区间内最小的小于当前数的数，所以取队首元素
        }
    }

    long long ans=0;
    for(int j=1;j<=m;j++)
    {
        deque<int> dq;
        for(int i=1;i<=n;i++)
        {
            while(!dq.empty() && minn[i][j]<=minn[dq.back()][j])
            dq.pop_back();
            dq.push_back(i);
            while(!dq.empty() && i-dq.front()+1>a)
            dq.pop_front();
            if(i>=a && j>=b)
            ans+=(long long)minn[dq.front()][j];
        }
```

```
    }
    cout<<ans;
    return 0;
}
```

> 单调队列有许多作用：
> 比如可以求出一个数组内第一个大于等于一个数x的数
> 也可以通过维护单调性，解决一些区间内最小或最大的问题
> 总之单调队列的应用在根本上要视题目而定的灵活运用

9.2. 单调栈（队尾插入队尾删除）

例题 **hdu1506**

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
const int INF = 0x3f3f3f3f ;
const int maxn = 100100;
const double pi = acos(-1.0);
int T,n,m;
stack<int>sta;
LL num[maxn];
int L[maxn],R[maxn];
int main()
{
    while(cin>>n&&n){
        while(!sta.empty())sta.pop();
        for(int i=1;i<=n;i++)scanf("%lld",&num[i]);
        for(int i=1;i<=n;i++){
            while(!sta.empty()&&num[sta.top()]>=num[i])sta.pop();//单调递增
            if(sta.empty())L[i]=1;
            else L[i]=sta.top()+1;
            sta.push(i);
        }
        while(!sta.empty())sta.pop();
        for(int i=n;i>=1;i--){
            while(!sta.empty()&&num[sta.top()]>=num[i])sta.pop();
            if(sta.empty())R[i]=n;
            else R[i]=sta.top()-1;//获取当前栈中第一个小于当前数的数，所以取栈顶元素
            sta.push(i);
        }
        LL ans=0;
        for(int i=1;i<=n;i++){
           // cout<<i<<" "<<L[i]<<" "<<R[i]<<" "<<(R[i]-L[i])*num[i]<<endl;
            ans=max(ans,(R[i]-L[i]+1)*num[i]);
```

```
            }
            cout<<ans<<endl;
        }
        system("pause");
        return 0;
    }
```

10. 笛卡尔树

简单介绍

10.1. 定义

笛卡尔树是一种二叉搜索树的数据结构。树的每个节点有两个值，分别为key 和 value。

Key类似于二叉搜索树，每个节点的左子树key值都要小于其本身，右子树的key值都比它大

value类似堆，根节点的value是最小（或者最大）的，每个节点的value都比它的子树要小（或者大）。

10.2. 性质

树中的元素满足二叉搜索树性质，要求按照中序遍历得到的序列为原数组序列

树中节点满足堆性质，节点的key值要大于其左右子节点的key值

10.3. 建立

当按照key从1到n的顺序将数组中的每个元素插入到笛卡尔树中时，当前要被插入的元素的key值最大，因此根据二叉搜索的性质需要沿着当前已经完成的笛卡尔树的根的右子树链搜索。

由于笛卡尔树要满足堆的性质（以最大堆为例），父节点的value值要大于子节点的value值，所以沿着树根的右子树链往下走，直到搜索到的节点的value值小于等于当前要插入节点的value值。

此时，便找到了当前结点需要插入的位置，记为Pos。此时Pos下方的节点的value值肯定小于当前被插入节点的value，但是key也小于当前插入节点的key（即需要在二叉搜索树中当前结点之前的位置），所以将当前节点插入到Pos的位置，同时将以Pos为根的子树挂载到新插入的节点的左子树（为了保证Pos及其子树在新插入节点之前被二叉搜索）。

```

```

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int n,v[N];//v数组为键值数组
int fa[N],ls[N],rs[N];//笛卡尔树用到
int s[N],top;//单调栈维护极右链
void Tree()
{
    top=0;
    memset(fa,0,sizeof fa);
    memset(ls,0,sizeof ls);
    memset(rs,0,sizeof rs);
    for(int i = 1; i <= n; i ++)
        {
        scanf("%d",&v[i]);
        while(top && v[s[top]] > v[i])//找到第一个小于v[i]的数值的位置
            ls[i] = s[top], top --;//如果大于v[i]，则向上走
        fa[i] = s[top];//给i赋值父亲节点
        fa[ls[i]] = i;//i节点左孩子的父亲为i
        if(fa[i]) rs[fa[i]] = i;//如果i不是根节点，i为右孩子
        s[++ top] = i;//入栈
    }
}
```

11. 字符串相关

11.1. 序列自动机

序列自动机

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
const int INF = 0x3f3f3f3f;
const int maxn = 101100;
const double pi = acos(-1.0);
int T, n, m;
char s[100500];
char a[1010];
int Next[100500][30];
int main()
{
    for (int i = 0; i <= 100000; i++)
```

```
    {
        for (int j = 0; j < 26; j++)
        {
            Next[i][j] = -1;
        }
    }
    cin >> s;
    int len = strlen(s);
    for(int i=len-1;i>=0;i--){
        for(int j=0;j<26;j++){
            Next[i][j]=Next[i+1][j];
        }
        Next[i][s[i]-'a']=i+1;
    }
    cin >> n;
    while (n--)
    {
        scanf("%s", a);
        int l = strlen(a);
        int f = 1;
        int now=0;
        for (int i = 0; f && i < l; i++)
        {
            now = Next[now][a[i] - 'a'];
            if (now == -1)
            {
                f = 0;
                break;
            }

        }
        if(f)cout<<"YES\n";
        else cout<<"NO\n";
    }
    system("pause");
    return 0;
}
```

11.2. 后缀自动机和回文自动机

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=4e5+10;
const int N=2e5+10;
const int maxc=28;
char s[N];
struct Suffix_Automaton {
    int len[maxn * 2]; //最长子串的长度(该节点子串数量=len[x]-len[link[x]])
    int link[maxn * 2];   //后缀链接(最短串前部减少一个字符所到达的状态)
    int cnt[maxn * 2];   //被后缀连接的数
    int nex[maxn * 2][maxc];  //状态转移(尾部加一个字符的下一个状态)(图)
```

```
    int idx; //结点编号
    int last;    //最后结点
    ll num[maxn * 2]; // enpos数（子串出现数量）
    ll a[maxn];     //长度为i的子串出现最大次数

    void init() {   //初始化
        for(int i=1; i<=idx; i++)
            link[i] = len[i] = 0,memset(nex[i],0,sizeof(nex[i]));
        last = idx = 1; //1表示root起始点 空集
    }
//SAM建图
    void extend(int c) {     //插入字符，为字符ascll码值
        int x = ++idx; //创建一个新结点x;
        len[x] = len[last] + 1; //  长度等于最后一个结点+1
        num[x] = 1;  //接受结点子串除后缀连接还需加一
        int p;  //第一个有C转移的结点;
        for (p = last; p && !nex[p][c]; p = link[p])
            nex[p][c] = x;//沿着后缀连接 将所有没有字符c转移的节点直接指向新结点
        if (!p)link[x] = 1, cnt[1]++;  //全部都没有c的转移 直接将新结点后缀连接到起点
        else {
            int q = nex[p][c];    //p通过c转移到的结点
            if (len[p] + 1 == len[q])    //pq是连续的
                link[x] = q, cnt[q]++; //将新结点后缀连接指向q即可,q结点的被后缀连接数+1
            else {
                int nq = ++idx;   //不连续 需要复制一份q结点
                len[nq] = len[p] + 1;   //令nq与p连续
                link[nq] = link[q];   //因后面link[q]改变此处不加cnt
                memcpy(nex[nq], nex[q], sizeof(nex[q]));  //复制q的信息给nq
                for (; p&&nex[p][c] == q; p = link[p])
                    nex[p][c] = nq;    //沿着后缀连接 将所有通过c转移为q的改为nq
                link[q] = link[x] = nq; //将x和q后缀连接改为nq
                cnt[nq] += 2; //  nq增加两个后缀连接
            }
        }
        last = x;  //更新最后处理的结点
    }
    ll getSubNum() {    //求不相同子串数量
        ll ans = 0;
        for (int i = 2; i <= idx; i++)
            ans += len[i]-len[link[i]]; //一状态子串数量等于len[i]-len[link[i]]
        return ans;
    }
} sam;
struct Palindromic_Tree{
    int next[N][26];//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[N];//fail指针，失配后跳转到fail指针指向的节点      最长回文后缀
    int len[N];//len[i]表示节点i表示的回文串的长度
    int S[N];//存放添加的字符
    ll cnt[N];//结点表示的本质不同的回文串的个数(调用count()后)
    int num[N];//结点表示的最长回文串的最右端点为回文串结尾的回文串个数
    int last;;//指向上一个字符所在的节点，方便下一次add
    int n;//字符数组指针
    int p;//节点指针
    int newnode(int x){//新建节点
        memset(next[p],0,sizeof(next[p]));
        cnt[p]=0;
        num[p]=0;
```

```
        len[p]=x;
        return p++;
    }
    void init(){
        p=0;
        newnode(0);
        newnode(-1);
        last=0;
        n=0;
        S[0]=-1;//开头放一个字符集中没有的字符，减少特判
        fail[0]=1;
    }
    int get_fail(int x){//和KMP一样，失配后找一个尽量最长的
        while(S[n-len[x]-1]!=S[n]) x=fail[x];
        return x;
    }
    void add(int c){
        c-='a';
        S[++n]=c;
        int cur=get_fail(last);//通过上一个回文串找这个回文串的匹配位置
        if(!next[cur][c]){//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now=newnode(len[cur]+2);//新建节点
            fail[now]=next[get_fail(fail[cur])][c];//和AC自动机一样建立fail指针，以便失配后跳转
            num[now]=num[fail[now]]+1;
            next[cur][c]=now;
        }
        last=next[cur][c];
        cnt[last]++;
    }
    void count(){
        for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];
        //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！
    }
}pam;
int main(){
    scanf("%s",s);
    int len=strlen(s);
    //建广义SAM
    sam.init();
    for(int i=0;i<len;i++) sam.extend(s[i]-'a');
    sam.last=1;
    for(int i=len-1;i>=0;i--) sam.extend(s[i]-'a');
    ll a=sam.getSubNum();//得到不同的子串的数目
    //建PAM
    pam.init();
    for(int i=0;i<len;i++) pam.add(s[i]);
    int b=pam.p-2;
    printf("%lld\n",(a+b)/2);
    return 0;
}
```

12. 三分查找算法

```
double solve(double parameter)
{
    // 计算函数值，即f(x)
}

double trisection_search(double left, double right)
{
    // 三分搜索，找到最优解（求函数最大值下的自变量值）
    double midl, midr;
    while (right-left > 1e-7)
    {
        midl = (left + right) / 3;
        midr = (midl + right) / 2;
        // 如果是求最小值的话这里判<=即可
        if(solve(midl) >= solve(midr)) right = midr;
        else left = midl;
    }
    return left;
}
```

13. 数学相关模板

13.1. 线性基

13.1.1. 简单理解

- 原序列里面的任意一个数都可以由线性基里面的一些数异或得到
- 线性基里面的任意一些数异或起来都不能得到0 00
- 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

```
bool add(ll x)
{
    for(int i=50;i>=0;i--)
    {
        if(x&(1ll<<i))//注意，如果i大于31，前面的1的后面一定要加ll
        {
            if(d[i])x^=d[i];
            else
            {
                d[i]=x;
                break;//记得如果插入成功一定要退出
            }
        }
    }return x;
}
```

```
ll ans()
{
    ll anss=0;
    for(int i=50;i>=0;i--)//记得从线性基的最高位开始
    if((anss^d[i])>anss)anss^=d[i];
    return anss;
 }

void work()//处理线性基
{
        for(int i=1;i<=60;i++)
        for(int j=1;j<=i;j++)
        if(d[i]&(1<<(j-1)))d[i]^=d[j-1];
}
ll k_th(ll k)
{
        if(k==1&&tot<n)return 0;//特判一下，假如k=1，并且原来的序列可以异或出0，就要返回0，tot表示线性基中的元素个数，n表示序列长度
        if(tot<n)k--;//类似上面，去掉0的情况，因为线性基中只能异或出不为0的解
        work();
        ll ans=0;
        for(int i=0;i<=60;i++)
        if(d[i]!=0)
        {
                if(k%2==1)ans^=d[i];
                k/=2;
        }
}
```

第二种写法

```
struct L_B{
    long long d[61],p[61];
    int cnt;
    L_B()
    {
        memset(d,0,sizeof(d));
        memset(p,0,sizeof(p));
        cnt=0;
    }
    bool insert(long long val)
    {
        for (int i=60;i>=0;i--)
            if (val&(1LL<<i))
            {
                if (!d[i])
                {
                    d[i]=val;
                    break;
                }
                val^=d[i];
            }
```

```
            return val>0;
        }
        long long query_max()
        {
            long long ret=0;
            for (int i=60;i>=0;i--)
                if ((ret^d[i])>ret)
                    ret^=d[i];
            return ret;
        }
        long long query_min()
        {
            for (int i=0;i<=60;i++)
                if (d[i])
                    return d[i];
            return 0;
        }
        void rebuild()
        {
            for (int i=60;i>=0;i--)
                for (int j=i-1;j>=0;j--)
                    if (d[i]&(1LL<<j))
                        d[i]^=d[j];
            for (int i=0;i<=60;i++)
                if (d[i])
                    p[cnt++]=d[i];
        }
        long long kthquery(long long k)//先调用rebuild
        {
            int ret=0;
            if (k>=(1LL<<cnt))
                return -1;
            for (int i=60;i>=0;i--)
                if (k&(1LL<<i))
                    ret^=p[i];
            return ret;
        }
    }
    L_B merge(const L_B &n1,const L_B &n2)
    {
        L_B ret=n1;
        for (int i=60;i>=0;i--)
            if (n2.d[i])
                ret.insert(n1.d[i]);
        return ret;
    }
```

13.1.2. 前缀线性基

贪心地维护序列的前缀线性基(上三角形态)，对于每个线性基，将出现位置靠右的数字尽可能地放在高位，也就是说在插入新数字的时候，要同时记录对应位置上数字的出现位置，并且在找到可以插入的位置的时候，如果新数字比位置上原来的数字更靠右，就将该位置上原来的数字向低位推。

在求最大值的时候，从高位向低位遍历，如果该位上的数字出现在询问中区间左端点的右侧且可以使答案变大，就异或到答案里。

对于线性基的每一位，与它异或过的线性基更高位置上的数字肯定都出现在它右侧(否则它就会被插入在那个位置了)，因此做法的正确性显然。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 500105;
const double pi = acos(-1.0);
int T, n, m;
int pre[maxn][40], a[maxn], pos[maxn][40];
int main()
{
    cin >> T;
    while (T--)
    {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
        for (int i = 0; i < 40; i++)
        {
            pre[0][i] = 0; //初始化
            pos[0][i] = 0;
        }
        for (int i = 1; i <= n; i++)
        {
            int p = i;
            int x = a[i];
            for (int j = 0; j < 40; j++)
            {
                pre[i][j] = pre[i - 1][j];
                pos[i][j] = pos[i - 1][j];
            }
            for (int j = 31; j >= 0; j--)
            {
               // cout<<"FLAG"<<i<<" "<<x<<endl;
                if (x & (1 << j))
                {
                    if (pre[i][j])
                    {
```

```
                    if (pos[i][j] < p)//下传
                    {
                        swap(pre[i][j], x);
                        swap(pos[i][j], p);
                    }
                    x ^= pre[i][j];
                }
                else
                {
                    pre[i][j] = x;
                    pos[i][j] = p;
                    break;
                }
            }
        } //j
    }     //i
    int op, l, r;
    int la=0;
    while (m--)
    {
        scanf("%d%d", &op, &l);
        if (op == 1)
        {
            l=l^la;
            a[++n]=l;
            int p = n;
            int x = a[n];
            for (int j = 0; j < 40; j++)
            {
                pre[n][j] = pre[n - 1][j];
                pos[n][j] = pos[n - 1][j];
            }
            for (int j = 31; j >= 0; j--)
            {
                if (x & (1 << j))
                {
                    if (pre[n][j])
                    {
                        if (pos[n][j] < p)
                        {
                            swap(pre[n][j], x);
                            swap(pos[n][j], p);
                        }
                        x ^= pre[n][j];
                    }
                    else
                    {
                        pre[n][j] = x;
                        pos[n][j] = p;
                        break;
                    }
                }
            } //j
        }
        else
        {
            scanf("%d", &r);
```

```
                l=(l^la)%n+1;
                r=(r^la)%n+1;
                if(l>r)swap(l,r);
                int ans=0;
                for(int i=35;i>=0;i--)
                {
                    if(pos[r][i]>=l&&((ans^pre[r][i])>ans))ans^=pre[r][i];
                }
                la=ans;
                printf("%d\n",ans);
            }
        }
    }
    system("pause");
    return 0;
}
```

13.2. 线性基求交

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define sc(x) scanf("%d",&x)
#define scc(x,y) scanf("%d%d",&x,&y)
#define sccc(x,y,z) scanf("%d%d%d",&x,&y,&z)
#define mkp(a,b) make_pair(a,b)
#define F first
#define ep(a) emplace_back(a)
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const int INF = 0x3f3f3f3f ;
const int maxn = 10000;
const double pi = acos(-1.0);
int T,n,m;
LL x[61];
struct LinearBasis{
    LL basis[61],p[61];
    int cnt;//极大无关组的大小
        int num=0;//数的个数
        int size;//线性基是多少位的
        void push(LL x)
        {
                basis[num++]=x;
        }
        void build()//生成线性基
        {
                cnt=0;
```

```
                num--;
                for(int i=0;i<=num;i++)x[i]=basis[i];
                mem0(basis);
                for(int i=0;i<=num;i++)
                {
                        for(int j=size;i>=0;j--)
                        {
                                if((x[i]>>j)&1)
                                {
                                        if(basis[j]==0)
                                        {
                                                cnt++;
                                                basis[j]=x[i];
                                                break;
                                        }
                                        else x[i]^=basis[j];
                                }
                        }
                }
                num=0;
        }
        bool check(LL xx)//判断一个数在不在线性基中
        {
                for(int i=size;i>=0;i--)
                {
                        if((xx>>i)&1){
                                if(basis[i]==0)break;
                                else xx^=basis[i];
                        }

                }return (xx==0);
        }
    LinearBasis()
    {
                num=0;
        memset(basis,0,sizeof(basis));
        memset(p,0,sizeof(p));
        cnt=0;
    }
        void clear()
        {
                num=0;
                memset(basis,0,sizeof(basis));
        memset(p,0,sizeof(p));
        cnt=0;
        }
    bool insert(long long val)//线性基插入
    {
        for (int i=size;i>=0;i--)
            if (val&(1LL<<i))
            {
                if (!basis[i])
                {
                    basis[i]=val;
                    break;
                }
                val^=basis[i];
```

```
            }
        return val>0;
    }
    long long query_max()
    {
        long long ret=0;
        for (int i=size;i>=0;i--)
            if ((ret^basis[i])>ret)
                ret^=basis[i];
        return ret;
    }
    long long query_min()
    {
        for (int i=0;i<=size;i++)
            if (basis[i])
                return basis[i];
        return 0;
    }
    void rebuild()
    {
        for (int i=size;i>=0;i--)
            for (int j=i-1;j>=0;j--)
                if (basis[i]&(1LL<<j))
                    basis[i]^=basis[j];
        for (int i=0;i<=size;i++)
            if (basis[i])
                p[cnt++]=basis[i];
    }
    long long kthquery(long long k)//先调用rebuild
    {
        int ret=0;
        if (k>=(1LL<<cnt))
            return -1;
        for (int i=size;i>=0;i--)
            if (k&(1LL<<i))
                ret^=p[i];
        return ret;
    }

};
LinearBasis merge(const LinearBasis &n1,const LinearBasis &n2)
{
    LinearBasis ret=n1;
    for (int i=60;i>=0;i--)
        if (n2.basis[i])
            ret.insert(n1.basis[i]);
    return ret;
}
LinearBasis Merge(LinearBasis A,LinearBasis B) {//线性基求交
        LinearBasis All , C , D;
        All.clear();
        C.clear();
        D.clear();
        for (int i = 60;i >= 0;i--) {//先把A线性基放入ALL中
                All.basis[i] = A.basis[i];
                D.basis[i] = 1ll << i;
        }
```

```
        for (int i = 60; i >= 0; i--) {
                if (B.basis[i]) {//对每一个B中的线性基试图插入A中，如果能插入，说明这个B中的线性基和A线性无关，如果不能插入，说明线性有关，就是AB的交
                        LL v = B.basis[i] , k = 0;
                        bool can = true;
                        for (int j = 60; j >= 0; j--) {
                                if (v & (1ll << j)) {//试图插入过程
                                        if (All.basis[j]) {
                                                v ^= All.basis[j];
                                                k ^= D.basis[j];
                                        } else {
                                                can = false;
                                                All.basis[j] = v;
                                                D.basis[j] = k;
                                                break;
                                        }
                                }
                        }

                        if (can) {//线性无关，是交
                                LL v = 0;
                                for (int j = 60; j >= 0; j--) {
                                        if (k & (1ll << j)) {
                                                v ^= A.basis[j];
                                        }
                                }
                                C.push(v);
                        }
                }
        }
        C.build();
        return C;
}

int main()
{
        system("pause");
        return 0;
}
```

13.3. 素数筛法

13.3.1. 埃式筛法

```
//获取0~max_prime内的所有素数
bool is_prime[maxn];
void Prime(int max_prime)//vis[1]=1 代表不是素数 vis[i]=0 代表是素数
{
    int tmp=sqrt(max_prime+0.5);
```

```
    memset(is_prime,0,sizeof is_prime);
    is_prime[0]=is_prime[1]=1;
    for(int i=2;i<=tmp;i++)
    {
        if(!is_prime[i]){
            for(int j=i*i;j<=max_prime;j+=i)is_prime[j]=1;
        }
    }
}
```

13.3.2. 欧式筛法

```
ll prime[maxn];      //就是个素数表
bool sf[maxn];        //判断这个数是不是素数，sf[i]中的i是从1到maxn的数
void sushu()
{         //核心 欧拉筛代码
    ll num=0;        //num 用来记筛到第几个质数
    memset(sf,true,sizeof(sf));
        sf[1] = false;
    sf[0] = false;  //1 0 特判
    for(int i = 2;i <= maxn; i ++)
        {          //外层枚举1~maxn
        if(sf[i]) prime[++num] = i;      //如果是质数就加入素数表
        for(int j = 1;j <= num;j ++)
                {       //内层枚举num以内的质数
            if(i * prime[j] > maxn) break; //筛完结束
            sf[i * prime[j]] = false;      //筛掉...
            if(i % prime[j] == 0) break;   //避免重复筛
        }
    }
}
```

13.4. 求逆元

逆元直接乘上就行

13.4.1. 扩展欧几里得求逆元（mod可以不为质数，逆元不一定存在）

给定模数m，求a的逆相当于求解ax=1(mod m) 这个方程可以转化为ax-my=1 然后套用求二元一次方程的方法，用扩展欧几里得算法求得一组x0,y0和gcd 检查gcd是否为1 gcd不为1则说明逆元不存在 若为1，则调整x0到0~m-1的范围中即可 PS：这种算法效率较高，常数较小，时间复杂度为O(ln n)

```
typedef  long long ll;
void extgcd(ll a,ll b,ll& d,ll& x,ll& y){
```

```
    if(!b){ d=a; x=1; y=0;}
    else{ extgcd(b,a%b,d,y,x); y-=x*(a/b); }
}
ll inverse(ll a,ll n){
    ll d,x,y;
    extgcd(a,n,d,x,y);
    return d==1?(x+n)%n:-1;
}
```

13.4.2. 费马小定理求逆元(mod必须为质数！！！)

在模为素数p的情况下，有费马小定理 a^(p-1)=1（mod p） 那么a^(p-2) =a^-1(mod p) 也就是说a的逆元为a^(p-2) 而在模不为素数p的情况下，有欧拉定理 a^phi(m)=1（mod m）(a⊥m) 同理a^-1=a^(phi(m)-1) 因此逆元x便可以套用快速幂求得了x=a^(phi(m)-1) 但是似乎还有个问题？如何判断a是否有逆元呢？检验逆元的性质，看求出的幂值x与a相乘是否为1即可 PS:这种算法复杂度为O（log2N）在几次测试中，常数似乎较上种方法大 当p比较大的时候需要用快速幂求解

```
typedef  long long ll;
ll pow_mod(ll x, ll n, ll mod){
    ll res=1;
    while(n>0){
        if(n&1)res=res*x%mod;
        x=x*x%mod;
        n>>=1;
    }
    return res;
}
ll inverse(ll a,ll mod){
    return pow_mod(a,mod-2);
}
```

13.4.3. 连续逆元(mod必须为质数！！！)

```
typedef  long long ll;
const int N = 1e5 + 5;
int inv[N];
void inverse(int n, int p) {//p为模值 必须为质数！！！
    inv[1] = 1;
    for (int i=2; i<=n; ++i) {
        inv[i] = (ll) (p - p / i) * inv[p%i] % p;
    }
}
```

13.5. 卢卡斯定理模板

```
#include<bits/stdc++.h>
using namespace std;
#define mem(a,b) memset(a,b,sizeof(a))
#define mkp(a,b) make_pair(a,b)
#define pii pair<int ,int>
typedef long long LL;
const double PI = acos(-1.0);
const double eps = 1e-6;
const int INF = 0x3f3f3f3f;
const int maxn = 100100;
int T,n,m;
int fact[maxn];//需要预处理出n!%mod
const int mod=1e5+3;
void init()//预处理出n!%mod
{
    fact[0]=1;
    for(int i=1;i<maxn;i++)
    {
        fact[i]=(fact[i-1]*i)%mod;
    }
}
int extgcd(int a,int b,int&x,int&y)
{
    int d=a;
    if(b!=0)
    {
        d=extgcd(b,a%b,y,x);
        y-=(a/b)*x;
    }
    else {
        x=1;
        y=0;
    }
    return d;
}
int mod_inverse(int a,int m)
{
    int x,y;
    extgcd(a,m,x,y);
    return (m+x%m)%m;
}
int mod_fact(int n,int p,int&e)
{
    e=0;
    if(n==0)return 1;
    int res=mod_fact(n/p,p,e);
    e+=n/p;
    if(n/p%2!=0)return res*(p-fact[n%p])%p;
    return res*fact[n%p]%p;
}
int mod_comb(int n,int k,int p)
{
```

```
    if(n<0||k<0||n<k)return 0;
    int e1,e2,e3;
    int a1=mod_fact(n,p,e1);
    int a2=mod_fact(k,p,e2);
    int a3=mod_fact(n-k,p,e3);
    if(e1>e2+e3)return 0;
    return a1*mod_inverse(a2*a3%p,p)%p;
}
int main()
{
    init();
    cout<<mod_comb(6,2,mod);
    system("pause");//提交题目之前注释掉这句话！！！
    return 0;
}
```

13.6. 二分乘法

```
LL multi(LL a,LL b,LL m)
{
    LL ans = 0;
    a %= m;
    while(b)
    {
        if(b & 1)
        {
            ans = (ans + a) % m;
            b--;
        }
        b >>= 1;
        a = (a + a) % m;
    }
    return ans;
}
```

13.7. 快速幂

```
long long modexp(long long a, long long b, LL mod)
{
    LL res = 1;
    while (b > 0)
    {
        if (b & 1)
            res = res * a % mod;
        b = b >> 1;
        a = a * a % mod;
    }
```

```
    return res;
}
```

13.8. 高斯消元

```
//高斯消元模板
LL inv(LL bs){//求逆元
    LL ans=1,x=mod-2;
    while(x){
        if(x&1)ans=ans*bs%mod;
        bs=bs*bs%mod;
        x>>=1;
    }
    return ans;
}

LL ans[100];//存放最终的解
LL b[100][100];
void Guess(int n){//高斯消元模板
    int k,i,j;
    LL t;
    for(i=1;i<=n;i++)
    {
        for(k=i,j=i+1;j<=n;j++)if(b[j][i]>b[k][i])k=j;
        if(k!=i)for(j=i;j<=n+1;j++)swap(b[i][j],b[k][j]);
        for(j=i+1;j<=n;j++)
            for(t=(b[j][i]*inv(b[i][i])%mod),k=i;k<=n+1;k++)
            {
                b[j][k]=(b[j][k]-(b[i][k]*t)%mod+mod)%mod;
            }
    }
    for(ans[n]=(b[n][n+1]*inv(b[n][n]))%mod,i=n-1;i;i--)
    {
        for(ans[i]=b[i][n+1],j=n;j>i;j--)ans[i]=(ans[i]+mod-(ans[j]*b[i][j])%mod)%mod;
        ans[i]=(ans[i]*inv(b[i][i]))%mod;
    }
}
```

13.9. 矩阵快速幂

```
/*
矩阵快速幂模板
by chsobin
*/
struct Matrix//一定注意初始化！！！！！！！！！！
{
```

```
    LL a[M][M];
    void init()//初始化为0矩阵
    {
        for(int i=0;i<M;i++)for(int j=0;j<M;j++)a[i][j]=0;
    }
    void init2() //初始化为单位矩阵
    {
        mem0(a);
        for(int i=1;i<M;i++)
        {
            a[i][i]=1;
        }
    }
    Matrix operator*( Matrix b)
    {
        Matrix ans;
        ans.init();
        for (int i = 0; i < M; ++i)
        {
            for (int j = 0; j < M; ++j)
            {
                ans.a[i][j] = 0;
                for (int k = 0; k < M; ++k)
                {
                    ans.a[i][j] += a[i][k] * b.a[k][j];
                    ans.a[i][j] %= mod;
                }
            }
        }
        return ans;
    }
};


//矩阵快速幂
Matrix qpow(Matrix a, LL n)
{
    Matrix ans;
    ans.init2();
    while (n)
    {
        if (n & 1)
            ans =ans*a;
        a = a*a;
        n /= 2;
    }
    return ans;
}
```

13.10. 组合数（待学）

https://blog.csdn.net/qq_40772692/article/details/81835414

```
typedef long long LL;
const LL mod=998244353;
LL comb[maxn][(maxn>>1)+1];
inline LL C(int n,int k){
    if(k>n)return 0;
    k = min(k,n-k);
    return comb[n][k];
}
void solve(){
    for(int i = 0;i<maxn;i++){
        comb[i][0] = 1;
        for(int j = 1;j<=(i>>1);j++){//j<=i/2
            comb[i][j] = (C(i-1,j-1) + C(i-1,j))%mod;
        }
    }
}
```

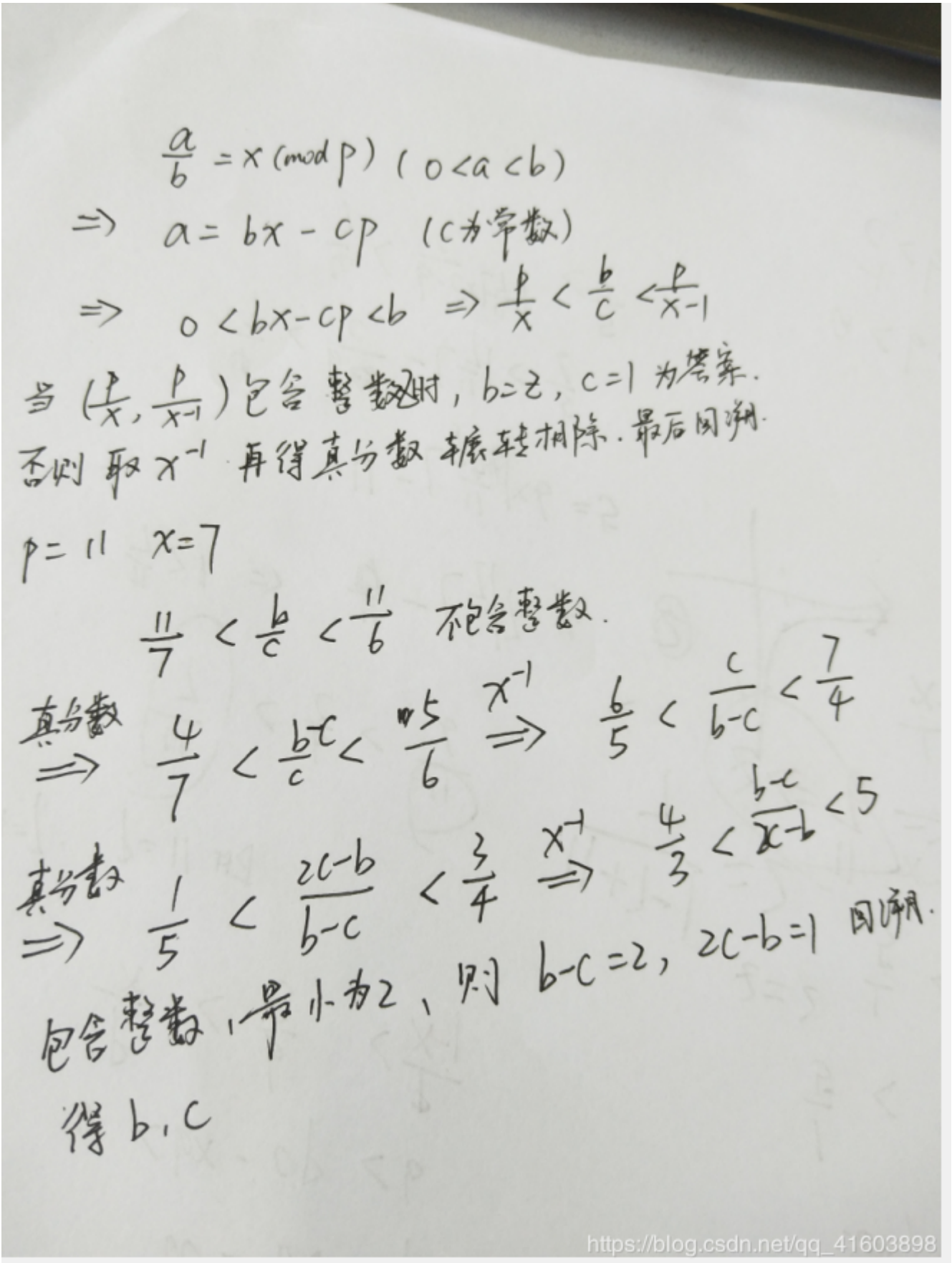
13.11. 蔡勒公式

W=[C/4]-2C+Y+[Y/4]+[13×(M+1)/5]+D-1， 或者是 W=Y+[Y/4]+[C/4]-2C+[26×(M+1)/10]+D-1 公式都是基于 公历 的置闰规则来考虑。 公式中的符号含义如下： • W：星期 • C：世纪数减一 (年份前两位数) • Y：年(年份后两位数) • M：月(M的取值范围为3至14，即在蔡勒公式中，某年的1、2月要看作上一年的13、14月来计算，比如2003 年1月1日要看作 2002 年的13月1日来计算) • D： 日 • []：称作高斯符号，代表取整，即只要整数部份 • mod：同余这里代表括号里的答案除以7后的余数 算出来的除以7，余数是几就是星期几。如果余数是0，则为星期日。

```
int cai(int p)
{
        int Y=f[data[p][0]-'A']*1000+f[data[p][1]-'A']*100+f[data[p][2]-'A']*10+f[data[p][3]-'A'];
        int M=f[data[p][5]-'A']*10+f[data[p][6]-'A'];
        int D=f[data[p][8]-'A']*10+f[data[p][9]-'A'];
        //C-=1;
        if(Y<1600||Y>9999)return 0;
        if((Y%100!=0&&Y%4==0)||Y%400==0)tian[2]=29;
        else tian[2]=28;
        if(M>12||M<1)return 0;
        if(D>tian[M]||D<1)return 0;
        if(M<=2){
                M+=12;
                Y--;
        }

        int C=Y/100;
        Y=Y%100;
        int W=(Y+Y/4+C/4-2*C+(26*(M+1))/10+D-1)%7;
        W=(W+7)%7;
```

```
        return W;
}
```

13.12. 辗转相除法解不等式(待学习)




```
#include <bits/stdc++.h>
#define ll long long
using  namespace std;
void gao(ll pa,ll pb,ll qa,ll qb,ll &x,ll &y){
        ll z = (pa+pb-1)/pb;
```

```
        if(z<=qa/qb){
                x = z;  y = 1;
                return ;
        }
        pa-=(z-1)*pb;   qa-=(z-1)*qb;
        gao(qb,qa,pb,pa,y,x);
        x+=(z-1)*y;
}
ll p,a,x,y;
int main()
{
        int _;
        for(scanf("%d",&_);_;_--){
                scanf("%lld %lld",&p,&a);
                gao(p,a,p,a-1,x,y);
                ll z = x*a - y*p;
                printf("%lld/%lld\n",z,x);
        }
        return 0;
}
```

14. RMQ

```
//也可以是区间最大值，倍增思想
void rmq_init()//RMQ初始化，数组下标从1开始
{
    for(int i=1;i<=N;i++)
        dp[i][0]=arr[i];//初始化
    for(int j=1;(1<<j)<=N;j++)
        for(int i=1;i+(1<<j)-1<=N;i++)
            dp[i][j]=min(dp[i][j-1],dp[i+(1<<j-1)][j-1]);
}

int rmq(int l,int r)//O(1)查询
{
    int k=log2(r-l+1);
    return min(dp[l][k],dp[r-(1<<k)+1][k]);
}
```

15. 树状数组

15.0.1. 一维

```
LL bit[maxn];//记录的节点从下标1开始
void add(int i,LL x)
{
    while(i<=n){
        shu[i]+=x;
        i+=i&-i;
    }
}
LL sum(int i)
{
    LL s=0;
    while(i>0){
        s+=shu[i];
        i-=i&-i;
    }
    return s;
}
```

15.0.2. 二维

```
ll a[maxn][maxn];
ll c[maxn][maxn];
int lowbit(int x){
    return x&(-x);
}
void modify(int x, int y, int val){
    a[x][y] += val;
    for(int i=x;i<=row;i+=lowbit(i))
        for(int j=y;j<=col;j+=lowbit(j))
            c[i][j] += val;
}
ll sum(int x,int y){
   ll ans=0;
   for(int i=x;i;i-=lowbit(i))
        for(int j=y;j;j-=lowbit(j))
            ans += c[i][j];
   return ans;
}
int main(){
    int i,j;
    int op;
    //freopen("F://2.txt","r",stdin);
    while(~scanf("%d",&op)){
        if(op==0){
            scanf("%d",&n);
            row=col=n;
            CL(c,0);
        }
        if(op==3)
            break;
        if(op==1){
            int x,y,t;
```

```
            scanf("%d %d %d",&x,&y,&t);
            modify(x+1,y+1,t);///BIT从1开始
        }
        if(op==2){
            int l,b,r,t;
            scanf("%d%d%d%d",&l,&b,&r,&t);
            printf("%lld\n",sum(r+1,t+1)+sum(l,b)-sum(l,t+1)-sum(r+1,b));///类似容斥
        }
    }
    return 0;
}
```

16. 线段树

16.1. 区间修改（加减）,区间求和

```
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
const int INF = 0x3f3f3f3f ;
const int maxn = 200000+100;
const double pi = acos(-1.0);
int T,n,m;
LL a[maxn];
LL sum[maxn*3]={0};//线段树的区间和
LL add[maxn*3]={0};//懒惰标记，标记这个区间变化了多少
void pushup(int o)////pushup函数，该函数本身是将当前结点用左右子节点的信息更新，此处求区间和，用于update中将结点信息传递完返回后更新父节点
{
    sum[o]=sum[o<<1]+sum[o<<1|1];
}
void pushdown(int o,int l,int r)////pushdown函数，将o结点的信息传递到左右子节点上
{
    if(add[o])////当父节点有更新信息时才向下传递信息
    {
        LL c=add[o];
        add[o<<1]+=c;////左右儿子结点均加上父节点的更新值
        add[o<<1|1]+=c;
        int m=l+((r-l)>>1);
        sum[o<<1]+=1ll*(m-l+1)*c;////左右儿子结点均按照需要加的值总和更新结点信息
        sum[o<<1|1]+=1ll*(r-m)*c;
        add[o]=0;//信息传递完之后就可以将父节点的更新信息删除
    }
```

```
}
void build(int o,int l,int r)
{
    add[o]=0;
    sum[o]=a[l];
    if(l==r) return ;
    int m=(l+r)/2;
    build(o<<1,l,m);
    build((o<<1)|1,m+1,r);
    sum[o]=sum[o<<1]+sum[(o<<1)|1];
}
void Update(int o,int l,int r,int ql,int qr,LL c)//ql、qr为需要更新的区间左右端点，c为需要增加的值
{
    if(ql<=l&&qr>=r)//当前区间处于要修改的区间中
    {
        add[o]+=c;//修改懒惰标记
        sum[o]+=1ll*(r-l+1)*c;//修改区间和
        return ;
    }
    pushdown(o,l,r);//向下更新信息
    int m=l+((r-l)>>1);
    if(ql<=m)Update(o<<1,l,m,ql,qr,c);//更新区间与左半区间有重合部分
    if(qr>=m+1)Update((o<<1)|1,m+1,r,ql,qr,c);//更新区间与右半区间有重合部分
    pushup(o);//向上更新信息
    //cout<<o<<"!!"<<sum[o]<<"\n";
}
LL Query(int o,int l,int r,int ql,int qr)
{
    if(ql<=l&&qr>=r)return sum[o];//当前区间处于查询区间中，直接返回区间值
    pushdown(o,l,r);
    int m=l+((r-l)>>1);
    LL ans=0;
    if(ql<=m)ans+=Query(o<<1,l,m,ql,qr);
    if(qr>=m+1)ans+=Query(o<<1|1,m+1,r,ql,qr);//若所需查询的区间与当前结点的右子节点有交集，则结果加上查询其右子节点的结果
    return ans;
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)scanf("%lld",&a[i]);
    build(1,1,n);
    string s;
    int a,b;
    while(m--)
    {
        cin>>s>>a>>b;
        if(s[0]=='Q')cout<<Query(1,1,n,a,b)<<endl;
        else
        {
            int c;
            cin>>c;
            Update(1,1,n,a,b,c);
        }
    }
```

```
    return 0;
}
```

16.2. 无区间修改，查询区间内最小值和最大值

```
using namespace std;
#define LL int
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
const int INF = 0x3f3f3f3f ;
const int maxn = 100000+100;
const double pi = acos(-1.0);
int T,n;
int h[maxn];
int maxh[maxn*4];
int minh[maxn*4];
void build(int o,int l,int r)
{
    if(l==r)
    {
        maxh[o]=minh[o]=h[l];//建树时读入
        return ;
    }
    int m=l+((r-l)>>1);
    build(o<<1,l,m);
    build(o<<1|1,m+1,r);
    maxh[o]=max(maxh[o<<1],maxh[o<<1|1]);
    minh[o]=min(minh[o<<1],minh[o<<1|1]);
}
void Query(int o,int l,int r,int ql,int qr,int&minans,int&maxans)
{
    if(ql<=l&&qr>=r)
    {
        maxans=max(maxans,maxh[o]);
        minans=min(minans,minh[o]);
        return ;
    }
    int m=l+((r-l)>>1);
    if(ql<=m)Query(o<<1,l,m,ql,qr,minans,maxans);
    if(qr>m)Query(o<<1|1,m+1,r,ql,qr,minans,maxans);
}
int main()
{
    int m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&h[i]);
```

```
    }
    build(1,1,n);
    int a,b;
    while(m--)
    {
        int c=INF;
        int d=0;
        scanf("%d%d",&a,&b);
        Query(1,1,n,a,b,c,d);
        cout<<d-c<<endl;
    }
    return 0;
}
```

16.3. 区间覆盖，查询有多少不同颜色（贴海报问题）

```
using namespace std;
#define LL int
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
const int INF = 0x3f3f3f3f ;
const int maxn = 2*10000+100;
const double pi = acos(-1.0);
int T,n,m;
int mp[10000000+10];//离散化端点坐标
bool book[maxn];//标记答案颜色是否出现
int f[maxn*4];//标记整个区间是什么颜色//0代表没有颜色或者区间颜色不相同
int l[maxn];//记录数据的左端点
int r[maxn];//记录数据的右端点
void build(int o,int l,int r)//建树
{
    if(l==r)
    {
        f[o]=0;//初始化标记
        return ;
    }
    int m=l+((r-l)>>1);
    build(o<<1,l,m);
    build(o<<1|1,m+1,r);
}
void pushdown(int o,int l,int r)//向下更新
{
    if(f[o]==0)return;//如果区间颜色相同，则向下更新，同时此区间的颜色置0，相当于把海报从中切开，完
整的海报不存在了
    f[o<<1]=f[o<<1|1]=f[o];
    f[o]=0;
}
void Update(int o,int l,int r,int ql,int qr,int c)//区间更新颜色
```

```
{
    if(ql<=l&&qr>=r)
    {
        f[o]=c;
        return ;
    }
    pushdown(o,l,r);//向下更新
    int mid=(l+r)/2;
    if(ql<=mid)Update(o<<1,l,mid,ql,qr,c);
    if(qr>mid)Update(o<<1|1,mid+1,r,ql,qr,c);
}
void Query(int o,int l,int r)
{
    if(l==r)
    {
        book[f[o]]=1;
        return;
    }
    pushdown(o,l,r);
    int m=l+((r-l)>>1);
    Query(o<<1,l,m);
    Query(o<<1|1,m+1,r);
}
int main()
{
    cin>>T;
    while(T--)
    {
        qu.clear();
        mem0(f);
        mem0(book);
        cin>>n;
        m=0;
        for(int i=1;i<=n;i++)
        {
            scanf("%d%d",&l[i],&r[i]);
            a[m++]=l[i];
            a[m++]=r[i];
        }
        sort(a,a+m);
        m=unique(a,a+m)-a;
        int t=0;
        for(int i=0;i<m;i++)
        {
            mp[a[i]]=++t;
        }
        for(int i=1;i<=n;i++)
        {
            Update(1,1,t,mp[l[i]],mp[r[i]],i);
        }
        Query(1,1,t);
        int ans=0;
        for(int i=1;i<=t;i++)
        {
            if(book[i])ans++;
        }
        cout<<ans<<endl;
```

```
    }
    return 0;
}
```

16.4. 区间每个数开根号（单点更新+剪枝），区间查询求和

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
const int INF = 0x3f3f3f3f ;
const int maxn = 100000+100;
const double pi = acos(-1.0);
int T,n,m;
LL sum[maxn*4];//线段树区间和，无懒惰标记
void pushup(int o)//向上更新函数
{
    sum[o]=sum[o<<1]+sum[o<<1|1];
}
void build(int o,int l,int r)//建树
{
    if(l==r)
    {
        scanf("%lld",&sum[o]);//建树时读入每个数据
        return;
    }
    int m=MID(l,r);
    build(o<<1,l,m);
    build(o<<1|1,m+1,r);
    pushup(o);//向上更新区间和（因为有非0的初始数据）
}
void update(int o,int l,int r,int ql,int qr)//更新函数，对区间内的每个数进行开根号
{
    if(ql<=l&&qr>=r&&sum[o]==r-l+1)return;//此区间处于查询区间中，且此区间的已经等于区间长度，就算
再进行开根运算，也不会改变区间和，函数终止
    if(ql<=l&&qr>=r&&l==r)//进行点更新
    {
        sum[o]=sqrt(sum[o]);
        return ;
    }
    int m=MID(l,r);
    if(ql<=m)update(o<<1,l,m,ql,qr);
    if(qr>=m+1)update(o<<1|1,m+1,r,ql,qr);
    pushup(o);
}
```

```
LL query(int o,int l,int r,int ql,int qr)//普通区间求和函数
{
    if(ql<=l&&qr>=r)
    {
        return sum[o];
    }
    int m=MID(l,r);
    LL ans=0;
    if(ql<=m)ans+=1ll*query(o<<1,l,m,ql,qr);
    if(qr>m)ans+=1ll*query(o<<1|1,m+1,r,ql,qr);
    return ans;
}
int main()
{
    int kase=0;
    while(cin>>n)
    {
        printf("Case #%d:\n",++kase);
        build(1,1,n);
        cin>>m;
        int a,b,c;
        while(m--)
        {
            scanf("%d%d%d",&c,&a,&b);
            if(a>b)swap(a,b);
            if(c==0)
            {
                update(1,1,n,a,b);
            }
            else
            {
                cout<<query(1,1,n,a,b)<<endl;
            }

        }
        cout<<endl;
    }
   // system("pause");
    return 0;
}
```

16.5. 区间修改加法，乘法，置换，区间查询每个数的幂的和

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
```

```
#define MID(l,r) (l+((r-l)>>1))
const int INF = 0x3f3f3f3f ;
const int maxn = 100010;
const double pi = acos(-1.0);
const int mod=10007;
int T,n,m;
bool flag[maxn*4];//此问题懒惰标记标记为此区间的所有数字是否相同，因为运算较多，难以对每种运算都进行懒惰标记处理
int num[maxn*4];//区间的数字
int quick_pow(int a,int b)//快速幂取模模板
{
    int ans=1;
    while(b)
    {
        if(b&1)ans=((ans%mod)*(a%mod))%mod;
        a=(a%mod)*(a%mod)%mod;
        b>>=1;
    }
    return ans;
}
void pushdown(int o)//向下更新
{
    if(flag[o])//如果此区间的所有数相同，向下更新
    {
        num[o<<1]=num[o<<1|1]=num[o];
        flag[o<<1]=flag[o<<1|1]=1;
        flag[o]=0;//标记清零
    }
}
void pushup(int o)//向上更新
{
    if(flag[o<<1]&&flag[o<<1|1]&&num[o<<1]==num[o<<1|1]){
        flag[o]=1;
        num[o]=num[o<<1];
    }
    else flag[o]=0;
}
void update(int o,int l,int r,int ql,int qr ,int c,int op)//区间更新
{

    if(ql<=l&&qr>=r&&flag[o])//只有此区间的所有数字相同时才能进行运算
    {
        if(op==1)
        {
            num[o]=(num[o]%mod+c%mod)%mod;
        }
        else if(op==2)
        {
            num[o]=((num[o]%mod)*(c%mod))%mod;
        }
        else if(op==3)
        {
            num[o]=c;
        }
        return;
    }
    pushdown(o);
```

```
    int m=MID(l,r);
    if(ql<=m)update(o<<1,l,m,ql,qr,c,op);
    if(qr>m)update(o<<1|1,m+1,r,ql,qr,c,op);
    pushup(o);
}
int query(int o,int l,int r,int ql,int qr,int p)//区间查询
{
    if(ql<=l&&qr>=r&&flag[o])//只有此区间的所有数字相同时才能进行幂的求和运算
    {

        int ans=quick_pow(num[o],p);
        ans=(ans*(r-l+1))%mod;
        return ans;
    }
    pushdown(o);
    int m=MID(l,r);
    int ans=0;
    if(ql<=m)ans+=query(o<<1,l,m,ql,qr,p);
    if(qr>m)ans+=query(o<<1|1,m+1,r,ql,qr,p);
    return ans%mod;
}
int main()
{
    while (cin >> n >> m)
    {
        if(n==0&&m==0)break;
        mem(flag,1);
        mem0(num);
        int a, b, c, d;
        while (m--)
        {
            scanf("%d%d%d%d", &a, &b, &c, &d);
            if(a<=3)
            {
                update(1,1,n,b,c,d,a);
            }
            else
            {
                printf("%d\n",query(1,1,n,b,c,d));
            }
        }
    }
   // system("pause");
    return 0;
}
```

16.6. 区间修改是否覆盖，区间查询相邻剩余空间左右端点值

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
```

```
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
const int INF = 0x3f3f3f3f ;
const int maxn = 50005;
const double pi = acos(-1.0);
int T,n,m;
int sum[maxn*4];//标记区间内已经覆盖范围的大小
void pushdown(int o,int l,int r)//向下更新
{
    if(sum[o]==r-l+1)//如果区间全覆盖
    {
        int m=MID(l,r);
        sum[o<<1]=m-l+1;
        sum[o<<1|1]=r-m;
    }
    if(sum[o]==0)//如果区间全没覆盖
    {
        sum[o<<1]=sum[o<<1|1]=0;
    }
}
void pushup(int o)//向上更新
{
    sum[o]=sum[o<<1]+sum[o<<1|1];
}
void update(int o,int l,int r,int ql,int qr,int c)//更新函数c={0,1}1代表覆盖了
{
    if(ql<=l&&qr>=r)
    {
        sum[o]=(r-l+1)*c;
        return;
    }
    pushdown(o,l,r);
    int m=MID(l,r);
    if(ql<=m)update(o<<1,l,m,ql,qr,c);
    if(qr>m)update(o<<1|1,m+1,r,ql,qr,c);
    pushup(o);
}
int query(int o,int l,int r,int ql,int qr)//查询函数
{
    if(ql<=l&&qr>=r)
    {
        return sum[o];
    }
    pushdown(o,l,r);
    int m=MID(l,r);
    int ans=0;
    if(ql<=m)ans+=query(o<<1,l,m,ql,qr);
    if(qr>m)ans+=query(o<<1|1,m+1,r,ql,qr);
    return ans;
}
int find(int qi,int num)//找从起点出发向右的端点，要想覆盖num的范围，需要覆盖到的最近的右端点
{
    //查询的区间内的剩余容量必须大于等于查询的容量，如果小于，需在调用find函数时修改查询的num的大小
```

```
    int l=qi,r=n;
    int ans=INF;
    while(l<=r)//二分查找
    {
        int m=MID(l,r);
        int shu=query(1,1,n,qi,m);
        int kong=m-qi+1-shu;
        if(kong>=num)
        {
            ans=min(ans,m);
            r=m-1;
        }
        else l=m+1;
    }
    return ans;
}
int main()
{
    cin>>T;
    while (T--)
    {
        mem0(sum);
        cin>>n>>m;
        int k,a,b;
        while(m--)
        {
            scanf("%d%d%d",&k,&a,&b);
            if(k==1)//charu
            {
                a++;
                int tmp=query(1,1,n,a,n);
                if(tmp==n-a+1)//如果查询区间全被覆盖
                {
                    printf("Can not put any one.\n");
                    continue;
                }
                int kong=n-a+1-tmp;
                if(kong<b)b=kong;//如果查询区间的剩余容量小于查询容量，修改查询容量
                int f1=find(a,1);
                int f2=find(a,b);
                update(1,1,n,f1,f2,1);
                cout<<f1-1<<" "<<f2-1<<endl;
            }
            else if(k==2)
            {
                a++,b++;
                cout<<query(1,1,n,a,b)<<endl;
                update(1,1,n,a,b,0);
            }
        }
        cout<<endl;
    }

    //system("pause");
    return 0;
}
```

16.7. 区间修改覆盖/清空，区间查询相应长的连续空间是否存在

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<string>
#include<vector>
#include<stack>
#include<bitset>
#include<cstdlib>
#include<cmath>
#include<set>
#include<list>
#include<deque>
#include<map>
#include<queue>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
const int INF = 0x3f3f3f3f ;
const int maxn = 50100;
const double pi = acos(-1.0);
int T,n,m;
int ls[maxn*3];//线段树区间从左端点开始最长连续区间的长度
int rs[maxn*3];//线段树区间从右端点开始向左最长连续区间的长度
int maxs[maxn*3];//此区间内最长连续区间的长度
int cover[maxn*3];//具体操作是当这个区间全部是1时color置1，全部为0时color置0，否则置-1。在pushup()
的时候会用到。
void build(int o,int l,int r)
{
    cover[o]=-1;//建树
    ls[o]=r-l+1;
    rs[o]=r-l+1;
    maxs[o]=r-l+1;
    if(l==r)return ;
    int m=MID(l,r);
    build(o<<1,l,m);
    build(o<<1|1,m+1,r);
}
void pushdown(int o,int l,int r)
{
    int k=r-l+1;
    if(cover[o]!=-1)//如果区间内全为1或者0，则向下更新
    {
        cover[o<<1]=cover[o<<1|1]=cover[o];
```

```
        ls[o<<1]=rs[o<<1]=maxs[o<<1]=cover[o]?0:(k-(k>>1));
        ls[o<<1|1]=rs[o<<1|1]=maxs[o<<1|1]=cover[o]?0:(k>>1);
        cover[o]=-1;//标记清零
    }
}
void pushup(int o,int l,int r)
{
    向上更新
    int k=r-l+1;
    ls[o]=ls[o<<1];
    rs[o]=rs[o<<1|1];
    if(ls[o]==k-(k>>1))ls[o]+=ls[o<<1|1];//如果当前ls长度等于左半区间的长度,ls加上右半区间的ls
    if(rs[o]==k>>1)rs[o]+=rs[o<<1];//同理
    maxs[o]=max(rs[o<<1]+ls[o<<1|1],max(maxs[o<<1],maxs[o<<1|1]));//区间的最大区间长度

}
void update(int o,int l,int r,int ql,int qr,int c)
{
    if(ql<=l&&qr>=r)
    {
        ls[o]=rs[o]=maxs[o]=c?0:(r-l+1);
        cover[o]=c;
        return;
    }
    pushdown(o,l,r);
    int m=MID(l,r);
    if(ql<=m)update(o<<1,l,m,ql,qr,c);
    if(qr>m)update(o<<1|1,m+1,r,ql,qr,c);
    pushup(o,l,r);
}
int query(int o,int l,int r,int len)
{
    if(l==r)return l;
    pushdown(o,l,r);
    int m=(l+r)>>1;
    if(maxs[o<<1]>=len)return query(o<<1,l,m,len);//左
    else if(rs[o<<1]+ls[o<<1|1]>=len)return m-rs[o<<1]+1;//中
    else return query(o<<1|1,m+1,r,len);//右
}
int main()
{
    while(cin>>n>>m)
    {
        build(1,1,n);
        while(m--)
        {
            int a,b,c;
            scanf("%d%d",&c,&a);
            if(c==1)
            {
                if(maxs[1]<a)
                {
                    cout<<"0"<<endl;
                    continue;
                }
                int pos=query(1,1,n,a);
                cout<<pos<<endl;
```

```
                update(1,1,n,pos,pos+a-1,1);
            }
            else
            {
                scanf("%d",&b);
                update(1,1,n,a,a+b-1,0);
            }
        }
    }
    return 0;
}
```

16.8. 求二维图形的边长（数据离散化）

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
#include <stack>
#include <bitset>
#include <cstdlib>
#include <cmath>
#include <set>
#include <list>
#include <deque>
#include <map>
#include <queue>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 10000 + 100;
const double pi = acos(-1.0);
int T, n, m;
struct Edge
{
    int l, r, f, h;
    bool operator<(const Edge &tmp) const
    {
        return h < tmp.h;
    }
    void get(int lt, int rt, int ht, int ft)
```

```
    {
        h = ht, l = lt, r = rt, f = ft;
    }
} edge1[maxn * 2], edge2[maxn * 2];
int x[maxn * 2], y[maxn * 2];
int xt, yt;
int len[maxn * 3];
int tag[maxn * 3];
void build(int o, int l, int r)
{
    len[o] = tag[o] = 0;
    if (l == r)
        return;
    int m = MID(l, r);
    build(ll(o), l, m);
    build(rr(o), m + 1, r);
}
void pushup(int o, int l, int r)
{
    if (tag[o])
        len[o] = x[r] - x[l - 1];
    else
        len[o] = len[ll(o)] + len[rr(o)];
}
void update(int o, int l, int r, int ql, int qr, int c)
{
    if(ql>r||qr<l)return;
    if (ql <= l && qr >= r)
    {
        tag[o] += c;
        pushup(o, l, r);
        return;
    }
    int m = MID(l, r);
    update(ll(o), l, m, ql, qr, c);
    update(rr(o), m + 1, r, ql, qr, c);
    pushup(o, l, r);
}

int t = 0;
int main()
{
    while (cin >> n)
    {
        t = 0;
        int a1, b1, a2, b2;
        xt = yt = 0;
        for (int i = 0; i < n; i++)
        {
            scanf("%d%d%d%d", &a1, &b1, &a2, &b2);
            edge1[t].get(a1, a2, b1, 1);
            edge2[t++].get(b1, b2, a1, 1);
            edge1[t].get(a1, a2, b2, -1);
            edge2[t++].get(b1, b2, a2, -1);
            x[xt++] = a1;
            x[xt++] = a2;
            y[yt++] = b1;
```

```
            y[yt++] = b2;
        }
        sort(edge1, edge1 + t);
        sort(edge2, edge2 + t);
        sort(x, x + xt);
        sort(y, y + yt);
        xt = unique(x, x + t) - x;
        yt = unique(y, y + t) - y;
        int ans = 0;
        int last = 0;
        build(1,1,xt);
        for (int i = 0; i < t; i++)
        {

            int l = lower_bound(x, x + xt, edge1[i].l) - x + 1;
            int r = lower_bound(x, x + xt, edge1[i].r) - x;
            update(1,1,xt,l,r,edge1[i].f);
            int tmp = len[1];
            ans+=abs(tmp-last);
            last=tmp;


        }
        xt=yt;
        for(int i=0;i<xt;i++)x[i]=y[i];
        last=0;
        build(1,1,xt);
        for (int i = 0; i < t; i++)
        {
            int l = lower_bound(x, x + xt, edge2[i].l) - x + 1;
            int r = lower_bound(x, x + xt, edge2[i].r) - x;
            update(1,1,xt,l,r,edge2[i].f);
            int tmp = len[1];
            ans+=abs(tmp-last);
            last=tmp;

        }
        cout<<ans<<endl;
    }
  //  system("pause");
    return 0;
}
/*
2
10 10 20 20
15 15 25 25
 */
```

16.9. 求二维空间的面积

```
using namespace std;
typedef long long int LL;
```

```
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 10000 + 100;
const double pi = acos(-1.0);
int T, n, m;
struct Edge
{
    int l, r, f, h;
    bool operator<(const Edge &tmp) const
    {
        return h < tmp.h;
    }
    void get(int lt, int rt, int ht, int ft)
    {
        h = ht, l = lt, r = rt, f = ft;
    }
} edge1[maxn * 2], edge2[maxn * 2];
int x[maxn * 2], y[maxn * 2];
int xt, yt;

int len[maxn * 3];
int tag[maxn * 3];
void build(int o, int l, int r)
{
    len[o] = tag[o] = 0;
    if (l == r)
        return;
    int m = MID(l, r);
    build(ll(o), l, m);
    build(rr(o), m + 1, r);
}
void pushup(int o, int l, int r)
{
    if (tag[o])
        len[o] = x[r] - x[l - 1];
    else
        len[o] = len[ll(o)] + len[rr(o)];
}
void update(int o, int l, int r, int ql, int qr, int c)
{
    if(ql>r||qr<l)return;
    if (ql <= l && qr >= r)
    {
        tag[o] += c;
        pushup(o, l, r);
        return;
    }
    int m = MID(l, r);
    update(ll(o), l, m, ql, qr, c);
```

```
    update(rr(o), m + 1, r, ql, qr, c);
    pushup(o, l, r);
}

int t = 0;
int main()
{
    while (cin >> n)
    {
        t = 0;
        int a1, b1, a2, b2;
        xt = yt = 0;
        for (int i = 0; i < n; i++)
        {
            scanf("%d%d%d%d", &a1, &b1, &a2, &b2);
            edge1[t].get(a1, a2, b1, 1);
            edge2[t++].get(b1, b2, a1, 1);
            edge1[t].get(a1, a2, b2, -1);
            edge2[t++].get(b1, b2, a2, -1);
            x[xt++] = a1;
            x[xt++] = a2;
            y[yt++] = b1;
            y[yt++] = b2;
        }
        sort(edge1, edge1 + t);
        sort(edge2, edge2 + t);
        sort(x, x + xt);
        sort(y, y + yt);
        xt = unique(x, x + t) - x;
        yt = unique(y, y + t) - y;
        int ans = 0;
        int last = 0;
        build(1,1,xt);
        for (int i = 0; i < t; i++)
        {

            int l = lower_bound(x, x + xt, edge1[i].l) - x + 1;
            int r = lower_bound(x, x + xt, edge1[i].r) - x;
            update(1,1,xt,l,r,edge1[i].f);
            int tmp = len[1];
            ans+=abs(tmp-last);
            last=tmp;


        }
        xt=yt;
        for(int i=0;i<xt;i++)x[i]=y[i];
        last=0;
        build(1,1,xt);
        for (int i = 0; i < t; i++)
        {
            int l = lower_bound(x, x + xt, edge2[i].l) - x + 1;
            int r = lower_bound(x, x + xt, edge2[i].r) - x;
            update(1,1,xt,l,r,edge2[i].f);
            int tmp = len[1];
            ans+=abs(tmp-last);
            last=tmp;
```

```
        }
        cout<<ans<<endl;
    }
  //  system("pause");
    return 0;
}
/*
2
10 10 20 20
15 15 25 25
 */
```

16.10. 最大连续区间和

```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
inline int read() {
    int x = 0, f = 1; char c = getchar();
    while (c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}
    while (c <= '9' && c >= '0') {x = x*10 + c-'0'; c = getchar();}
    return x * f;
}
const int maxn = 5e4+3, inf = 2147483647;
int n, m, opt, l, r;
struct node {
    int l, r, gss, gssr, gssl, sum;
}tree[maxn << 2];
struct TREE {
    #define Lson (k << 1)
    #define Rson ((k << 1) + 1)
    inline int MAX(int a, int b, int c) {
        return max(max(a, b), c);
    }
    inline void build(int k, int ll, int rr) {
        tree[k].l = ll, tree[k].r = rr;
        if(tree[k].l == tree[k].r) {
            tree[k].sum = read();
            tree[k].gss = tree[k].gssr = tree[k].gssl = tree[k].sum;
            return ;
        }
        int mid = (tree[k].l + tree[k].r) >> 1;
        build (Lson, tree[k].l, mid);
        build (Rson, mid+1, tree[k].r);
        tree[k].sum = tree[Lson].sum + tree[Rson].sum;
        tree[k].gss = MAX(tree[Lson].gss, tree[Rson].gss, tree[Lson].gssr+tree[Rson].gssl);
        tree[k].gssr = max(tree[Rson].gssr, tree[Rson].sum+tree[Lson].gssr);
        tree[k].gssl = max(tree[Lson].gssl, tree[Lson].sum+tree[Rson].gssl);
    }
    inline void update(int k, int pos, int num) {
```

```
        if(tree[k].l == tree[k].r && tree[k].l == pos) {
            tree[k].sum = num;
            tree[k].gss = tree[k].gssr = tree[k].gssl = tree[k].sum;
            return ;
        }
        int mid = (tree[k].l + tree[k].r) >> 1;
        if(pos <= mid) update(Lson, pos, num);
        else update(Rson, pos, num);
        tree[k].sum = tree[Lson].sum + tree[Rson].sum;
        tree[k].gss = MAX(tree[Lson].gss, tree[Rson].gss, tree[Lson].gssr+tree[Rson].gssl);
        tree[k].gssr = max(tree[Rson].gssr, tree[Rson].sum+tree[Lson].gssr);
        tree[k].gssl = max(tree[Lson].gssl, tree[Lson].sum+tree[Rson].gssl);
    }
    inline node query(int k, int L, int R) {
        if(tree[k].l == L && tree[k].r == R) return tree[k];
        int mid = (tree[k].l + tree[k].r) >> 1;
        if(L > mid) return query(Rson, L, R);
        else if(R <= mid) return query(Lson, L, R);
        else {
            node lson, rson, res;
            lson = query(Lson, L, mid);
            rson = query(Rson, mid+1, R);
            res.sum = lson.sum + rson.sum;
            res.gss = MAX(lson.gss, rson.gss, lson.gssr+rson.gssl);
            res.gssl = max(lson.gssl, lson.sum+rson.gssl);
            res.gssr = max(rson.gssr, rson.sum+lson.gssr);
            return res;
        }
    }
}T;
int main() {
    n = read(), T.build(1, 1, n);
    m = read();
    for(int i=1; i<=m; i++) {
        opt = read(), l = read(), r = read();
        if(opt == 1) printf("%d\n", T.query(1, l, r).gss);
        else T.update(1, l, r);
    }
}
```

网上代码

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int N = 50008;
#define mid (l+r>>1)
#define lc d<<1
#define rc d<<1|1

struct Tr{
    int v,lv,rv,ans;
```

```
}st[N<<2];
void Push(int d){
    st[d].v = st[lc].v+st[rc].v;
    st[d].lv = max(st[lc].lv,st[lc].v+st[rc].lv);
    st[d].rv = max(st[rc].rv,st[rc].v+st[lc].rv);
    st[d].ans = max(max(st[lc].ans,st[rc].ans),st[lc].rv+st[rc].lv);
    return;
}
void build(int l, int r, int d){
    if(l==r){
        scanf("%d",&st[d].ans);
        st[d].v = st[d].lv = st[d].rv = st[d].ans;
        return;
    }
    build(l,mid,lc);
    build(mid+1,r,rc);
    Push(d);
}
Tr query(int L, int R, int l, int r, int d)
{
    if(l==L&&R==r){
        return st[d];
    }
    else if(R<=mid) return query(L,R,l,mid,lc);
    else if(L>mid) return query(L,R,mid+1,r,rc);
    Tr la = query(L,mid,l,mid,lc);
    Tr ra = query(mid+1,R,mid+1,r,rc);
    Tr re;
    re.v = la.v+ra.v;
    re.lv = max(la.lv,la.v+ra.lv);
    re.rv = max(ra.rv,ra.v+la.rv);
    re.ans = max(max(ra.ans,la.ans),la.rv+ra.lv);
    return re;
}
int main()
{
    int n,m;
    int ll,rr;
    while(~scanf("%d",&n))
    {
        build(1,n,1);
        scanf("%d",&m);
        while(m--)
        {
            scanf("%d%d",&ll,&rr);
            Tr fn = query(ll,rr,1,n,1);
            printf("%d\n",fn.ans);
        }
    }
    return 0;
}
```

自己写的代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const int INF = 0x3f3f3f3f ;
const int maxn = 50000+100;
const double pi = acos(-1.0);
int T,n,m;
LL a[maxn];
LL ls[maxn*3];
LL rs[maxn*3];
LL sum[maxn*3];
LL ms[maxn*3];
void pushup(int o,int l,int r)
{
    int lson=ll(o);
    int rson=rr(o);
    int mid=MID(l,r);
    sum[o]=sum[lson]+sum[rson];
    ls[o]=max(ls[lson],sum[lson]+ls[rson]);
    rs[o]=max(rs[rson],sum[rson]+rs[lson]);
    ms[o]=ls[rson]+rs[lson];
    ms[o]=max(ms[o],max(ms[lson],ms[rson]));
}
void build(int o,int l,int r)
{
    sum[0]=-INF;
    if(l==r)
    {
        ms[o]=sum[o]=ls[o]=rs[o]=a[l];
        return ;
    }
    int mid=MID(l,r);
    build(o<<1,l,mid);
    build(o<<1|1,mid+1,r);
    pushup(o,l,r);
}
LL ans=-INF;
LL ansr;
LL ansm;
void update(int o,int l,int r,int v,LL num)
{
    if(l==v&&r==v)
    {
        sum[o]=ms[o]=ls[o]=rs[o]=num;
        return ;
    }
    int mid=MID(l,r);
```

```
    if(v<=mid)update(ll(o),l,mid,v,num);
    else update(rr(o),mid+1,r,v,num);
    pushup(o,l,r);
}
void query(int o,int l,int r,int ql,int qr)
{
    if(ql<=l&&qr>=r)
    {
        ans=max(ans,max(ansr+ls[o],ansr+sum[o]));
        ans=max(ans,rs[o]);
        ans=max(ans,ms[o]);
        ansr=max(rs[o],ansr+sum[o]);
        ansr=max(0ll,ansr);
        return ;
    }
    int mid=MID(l,r);
    if(ql<=mid)query(ll(o),l,mid,ql,qr);
    if(qr>mid) query(rr(o),mid+1,r,ql,qr);
}
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",&a[i]);
    }
    build(1,1,n);
    cin>>m;
    int l,r;
    int op;
    while(m--)
    {
        scanf("%d%d%d",&op,&l,&r);
        if(op==1){
            ansr=ansm=0;
            ans=-INF;
            query(1,1,n,l,r);
            printf("%lld\n",ans);
        }
        else
        {
            update(1,1,n,l,r);
        }
    }
    system("pause");
    return 0;
}
```

16.11. 斐波那契(待补题)

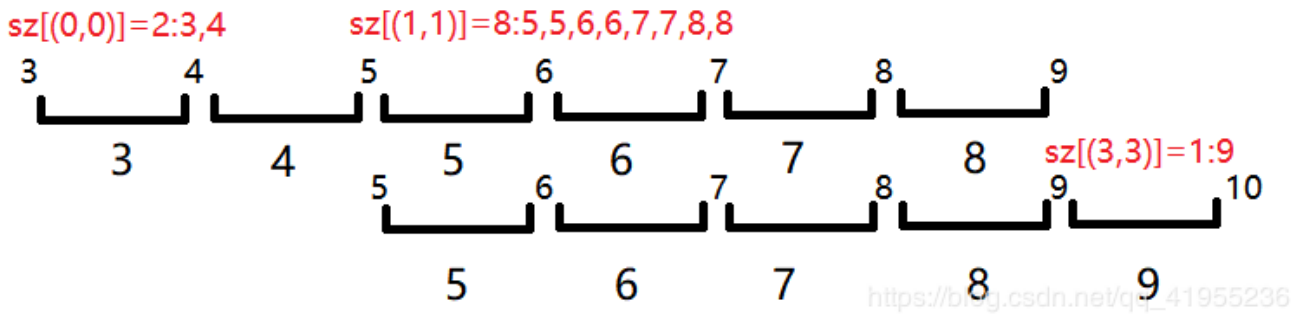
斐波那契通项公式 $$a_n = 1/\sqrt{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

16.12. 权值线段树

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define ep(a) emplace_back(a)
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const int INF = 0x3f3f3f3f ;
const int maxn = 200100;
const double pi = acos(-1.0);
int T,n,m;
LL a[maxn];
int sz=0;
LL num[maxn<<2];
int tot[maxn<<2];
LL b[maxn];
void build(int o,int l,int r)
{
    num[o]=0;
    tot[o]=0;
    if(l==r)return;
    int mid=MID(l,r);
    build(ll(o),l,mid);
    build(rr(o),mid+1,r);
}
inline pup(int o)
{
    num[o]=num[ll(o)]+num[rr(o)];
    tot[o]=tot[ll(o)]+tot[rr(o)];
}
void upd(int o,int l,int r,int x)
{
    if(l==r)
    {
        //cout<<b[x]<<"UPD\n";
        num[o]+=b[x];
        tot[o]+=1;
        return;
    }
    int mid=MID(l,r);
    if(x<=mid)upd(ll(o),l,mid,x);
    else upd(rr(o),mid+1,r,x);
    pup(o);
}
int query(int o,int l,int r,LL x)
{
```

```
    //if(l==r)return tot[o];
    //cout<<l<<" "<<r<<" "<<num[o]<<"!!!"<<x<<endl;
    if(num[o]<=x)
    {
        return tot[o];
    }
    if(l==r){
        //cout<<"\nTOT "<<tot[o]<<" "<<x<<" "<<b[l]<<endl;
        return min(1ll*tot[o],x/b[l]);
    }
    int mid=MID(l,r);
    int ans=0;
    if(num[ll(o)]>=x)return query(ll(o),l,mid,x);
    else  {
       // cout<<x<<" "<<num[ll(o)]<<"JIAN\n";
       return tot[ll(o)]+query(rr(o),mid+1,r,x-num[ll(o)]);
    }
}

int main()
{
    cin>>T;
    LL pre=0;
    while(T--)
    {
        mem0(tot);
        mem0(num);
        pre=0;
        //build()
        scanf("%d%d",&n,&m);

        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        for(int i=1;i<=n;i++){
            b[i]=a[i];
        }

        sort(b+1,b+1+n);
        sz=unique(b+1,b+1+n)-b-1;
        build(1,1,sz);
        for(int i=1;i<=n;i++)
        {

            int k=query(1,1,sz,m-a[i]);
            //cout<<"("<<k<<")";
            cout<<i-k-1<<" ";
            upd(1,1,sz,(lower_bound(b+1,b+1+sz,a[i])-b));
           // cout<<"LOW"<<lower_bound(b+1,b+1+sz,a[i])-b<<endl;
        }
        cout<<endl;
    }
    system("pause");
    return 0;
}
```

16.13. 区间转点



接下来详细讲一下如何建立线段树，区间的范围很大，我们肯定需要离散化，解法也很直观，肯定是线段树维护吗，就维护区间出现点的个数，这样我们query的时候就可以根据数量一直走到叶子节点，求出答案，所以我们每个叶子节点就是代表一段小区间了。这里算是一个小技巧吧，我们把所有的区间都看成是左开右闭的，这样叶子节点的区间就不怕冲突了。比如给出区间[1,3],[2,7],我们将其看成[1,4),[2,8);经过我们离散化会得到 1，2，4，8，那我们线段树的叶子节点所代表的区间就是$[1,2),[2,4),[4,8)$,离散化之后有4个数字,但是区间只有$4-1$个。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define sc(x) scanf("%d",&x)
#define scc(x,y) scanf("%d%d",&x,&y)
#define sccc(x,y,z) scanf("%d%d%d",&x,&y,&z)
#define mkp(a,b) make_pair(a,b)
#define F first
#define ep(a) emplace_back(a)
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const LL INF = 0x3f3f3f3f ;
const int maxn = 600010;
const double pi = acos(-1.0);
int T,n;
LL x[maxn],y[maxn],l[maxn],r[maxn];
LL a[3],b[3],c[3];
LL m[3];
LL po[maxn*2];
int sz=0;
LL tot[maxn<<2];
LL lazy[maxn<<2];
inline void pushup(int o)
{
    tot[o]=tot[ll(o)]+tot[rr(o)];
}
inline void pushdown(int o,int l,int r)
```

```
{
    if(lazy[o]==0)return;

    int mid=MID(l,r);
    tot[ll(o)]+=lazy[o]*(po[mid+1]-po[l]);
    tot[rr(o)]+=lazy[o]*(po[r+1]-po[mid+1]);
    lazy[ll(o)]+=lazy[o];
    lazy[rr(o)]+=lazy[o];
    lazy[o]=0;
}
inline void upd(int o,int l,int r,int ql,int qr)
{

    if(ql<=l&&qr>=r)
    {
        lazy[o]++;
        tot[o]+=po[r+1]-po[l];
        return ;
    }
    pushdown(o,l,r);
    int mid=MID(l,r);
    if(ql<=mid)upd(ll(o),l,mid,ql,qr);
    if(qr>mid)upd(rr(o),mid+1,r,ql,qr);
    pushup(o);
}

void build(int o,int l,int r)
{

    lazy[o]=0;
    tot[o]=0;
    if(l==r)return;
    int mid=MID(l,r);
    build(ll(o),l,mid);
    build(rr(o),mid+1,r);
}
int query(int o,int l,int r,LL k)
{
    if(l==r)
    {
        LL tmp=tot[o]/(po[r+1]-po[l]);
        return po[l]+(tmp+k-1)/tmp-1;
    }
    int mid=MID(l,r);
    int ans;
    pushdown(o,l,r);
    if(tot[ll(o)]>=k)return query(ll(o),l,mid,k);
    else return query(rr(o),mid+1,r,k-tot[ll(o)]);
}
int32_t main()
{
    sc(n);
    scanf("%lld%lld%lld%lld%lld%lld",&x[1],&x[2],&a[1],&b[1],&c[1],&m[1]);
    scanf("%lld%lld%lld%lld%lld%lld",&y[1],&y[2],&a[2],&b[2],&c[2],&m[2]);
    for(int i=3;i<=n;i++)x[i]=(a[1]*x[i-1]+b[1]*x[i-2]+c[1])%m[1];
    for(int i=3;i<=n;i++)y[i]=(a[2]*y[i-1]+b[2]*y[i-2]+c[2])%m[2];
    for(int i=1;i<=n;i++)
```

```
        {
            l[i]=min(x[i],y[i])+1;
            r[i]=max(x[i],y[i])+1+1;
            po[++sz]=l[i];po[++sz]=r[i];
        }
        sort(po+1,po+1+sz);
        po[0]=0;
        sz=unique(po+1,po+1+sz)-(po+1);
        for(int i=1;i<=n;i++)
        {
            l[i]=lower_bound(po+1,po+1+sz,l[i])-po;
            r[i]=lower_bound(po+1,po+1+sz,r[i])-po;
        }
        build(1,1,sz-1);
        LL cnt=0;
        for(int i=1;i<=n;i++){
            int ans;
            upd(1,1,sz-1,l[i],r[i]-1);
            cnt=tot[1];
            if(cnt%2==1)ans=query(1,1,sz-1,cnt/2ll+1);
            else
            {
                ans=query(1,1,sz-1,cnt/2ll);
            }
            printf("%d\n",ans);
        }
        system("pause");
        return 0;
    }
```

17. 主席树

17.1. 区间不同的数的个数

题目大意： 给你 n 个数，然后有 q 个询问，每个询问会给你[l,r]，输出[l,r]之间有多少种数字。

题目分析： 首先我们还是思考对于右端点固定的区间（即R确定的区间），我们如何使用线段树来解决这个问题。

我们可以记录每个数字最后一次出现的位置。比如，5这个数字最后一次出现在位置3上，就把位置3记录的信息++（初始化为0）。比如有一个序列 1 2 2 1 3 那么我们记录信息的数列就是 0 0 1 1 1 （2最后出现的位置是位置3 1最后出现的位置是位置4 3最后出现的位置是位置5）。

那么对着区间右端点会变化的题目，我们应该怎么办呢？先思考一下如果右端点有序的话，我们可以怎么做。对R不同的区间，向线段树或者树状数组中添加元素，知道右端点更新为新的

R，在添加的过程中，如果这个元素之前出现过，就把之前记录的位置储存的信息 -1，然后在新的位置储存的信息 +1，这样就可以保证在新的右端点固定的区间里，记录的是数字最后一次出现的位置的信息，这样题目就解决了。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 100000 + 100;
const double pi = acos(-1.0);
int node_cnt, n, m;
int sum[maxn << 5], rt[maxn << 5], lc[maxn << 5], rc[maxn << 5];
//主席树用，分别为，节点中的不同数的个数，第i个版本主席树的根节点位置，左右孩子节点编号
int a[maxn];
map<int, int> mp;//映射每个数字当前最后出现的位置
void build(int &t, int l, int r)//建树（空树）
{
    if(l==1&&r==n)node_cnt=0;
    t = ++node_cnt;//给新节点
    sum[t] = 0;
    if (l == r)return;
    int mid = MID(l, r);
    build(lc[t], l, mid);
    build(rc[t], mid + 1, r);
}
int update(int o, int l, int r, int k, int v)//建树过程
{
    int oo = ++node_cnt;
    lc[oo] = lc[o];
    rc[oo] = rc[o];
    sum[oo] = sum[o] + v;
    if (l == r)
        return oo;
    int mid = MID(l, r);
    if (k <= mid)//需要更新左节点
        lc[oo] = update(lc[oo], l, mid, k, v);
    else//需要更新右节点
        rc[oo] = update(rc[oo], mid + 1, r, k, v);
    return oo;
}
int query(int o, int l, int r, int k)//主席树查询函数
{
    int ans, mid = MID(l, r);
    if (l >= k)//区间左端点>=要查询的区间的左端点，（参考线段树求和）返回节点值
    {
```

```
        return sum[o];
    }
    if (k <= mid)//查询区间左端点在左半区间
        return query(lc[o], l, mid, k) + sum[rc[o]];
        //答案为查询到的左半区间+右半区间的全部的值
    else
        return query(rc[o], mid + 1, r, k);
        //左半区间不包含答案
}
int main()
{

    int p, q, ans, l, r;
    while (cin >> n)
    {
        mp.clear();//清空位置映射
        for (int i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
        }
        build(rt[0], 1, n);
        for (int i = 1; i <= n; i++)
        {
            if (mp.count(a[i]) == 0)//这个数没有出现过
            {
                rt[i] = update(rt[i - 1], 1, n, i, 1);
            }
            else
            {
                rt[(maxn << 5) - 1] = update(rt[i - 1], 1, n, mp[a[i]], -1);
                //旧的存储位置 -1，用临时主席树节点记录，为了
                rt[i] = update(rt[(maxn << 5) - 1], 1, n, i, 1);
                //然后在新的位置储存的信息 +1
                //使用临时节点为了保证主席树节点数量不变
            }
            mp[a[i]] = i;//更新最后出现的位置
        }
        cin >> m;
        while (m--)
        {
            scanf("%d%d", &l, &r);
            ans = query(rt[r], 1, n, l);//主席树版本为右端点的编号
            printf("%d\n", ans);
        }
    }
    system("pause");
    return 0;
}
```

17.2. 静态第k个数字

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
#include <stack>
#include <bitset>
#include <cstdlib>
#include <cmath>
#include <set>
#include <list>
#include <deque>
#include <map>
#include <queue>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 200010;
const double pi = acos(-1.0);
/*
将原始数组复制一份，然后排序好，然后去掉多余的数，即将数据离散化。推荐使用C++的STL中的unique函数；
以离散化数组为基础，建一个全0的线段树，称作基础主席树；
对原数据中每一个[1,i]区间统计，有序地插入新节点（题目中i每增加1就会多一个数，仅需对主席树对应的节点增加1即可）；
对于查询[1,r]中第k小值的操作，找到[1,r]对应的根节点，我们按照线段树的方法操作即可（这个根节点及其子孙构成的必定是一颗线段树）。
 */
int node_cnt, n, m;
int sum[maxn << 5], rt[maxn << 5], lc[maxn << 5], rc[maxn << 5]; //线段树相关 sum记录线段树权值
int a[maxn], b[maxn];                                            //原数组和离散数组
                                                    //修改点
void build(int &t, int l, int r)//递归建树
{
    if(l==1&&r==n)node_cnt=0;
    t = ++node_cnt;
    sum[t]=0;
    if (l == r)
        return;
    int mid = MID(l, r);
    build(lc[t], l, mid);
    build(rc[t], mid + 1, r);
}
int update(int o, int l, int r,int p)
{
    int oo = ++node_cnt; //生成新的节点
    lc[oo] = lc[o];
```

```
    rc[oo] = rc[o];
    sum[oo]=sum[o]+1;//因为新的版本的线段树一定比旧的版本多一个数，所以权值加一
    //" "<<l<<" "<<r<<" "<<lc[oo]<<" "<<rc[oo]<<" "<<sum[oo]<<endl;
    if (l == r)
        return oo;
    int mid = MID(l, r);
    if (p <= mid)//更新的值在左半区间
        lc[oo] = update(lc[oo], l, mid,p);
    else//跟新的值在右半区间内
        rc[oo] = update(rc[oo], mid + 1, r,p);
    return oo;
}
int query(int u,int v,int l,int r,int k)
{
    int ans,mid=MID(l,r);
    int x=sum[lc[v]]-sum[lc[u]];//获取区间[1,v]-[1,u-1]的左子树上的权值
    if(l==r)return l;
    if(x>=k)//权值大于查询的值，数在左半区间
        ans= query(lc[u],lc[v],l,mid,k);
    else
        ans= query(rc[u],rc[v],mid+1,r,k-x);
    return ans;
}
int main()
{
    int l, p, r, k, q, ans;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]), b[i] = a[i];
    }
    //进行数据离散化
    sort(b + 1, b + 1 + n);
    q = unique(b + 1, b + 1 + n) - b - 1;
    build(rt[0], 1, q);
    //建立基础主席树
    for (int i = 1; i <= n; i++)
    {
        p = lower_bound(b + 1, b + q + 1, a[i]) - b;
        rt[i] = update(rt[i - 1], 1, q,p);//更新[1,i],i=1,2,3,···, n
    }
    for(int i=1;i<=node_cnt;i++)
    {
        cout<<"root"<<i<<" "<<lc[i]<<" "<<rc[i]<<endl;
    }
    cout<<node_cnt<<"!!!!\n";
    while(m--)
    {
        scanf("%d%d%d",&l,&r,&k);
        ans = query(rt[l-1],rt[r],1,q,k);//进行查询
        printf("%d\n",b[ans]);
    }
    system("pause");
    return 0;
}
//4 1 1 2 8 9 4 4 3
```

17.3. 树上路径第k大

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int DEG = 20;
const int maxn = 100000 + 10;
const double pi = acos(-1.0);

struct Edge //链式前向星的边结构体
{
    int to, next;
} edge[maxn * 2];
int head[maxn];  //链式前向星需要
int tot = 0;     //记录链式前向星中边的条数
int node_cnt, n; //主席树的节点个数、数的个数
int m;
int sum[maxn << 5], rt[maxn << 5], lc[maxn << 5], rc[maxn << 5]; //线段树相关 sum记录线段树权值
int a[maxn], b[maxn];                                            //原数组和离散数组
int q;                                                           //离散数组的长度
int fa[maxn][DEG];                                               //LCA用 fa[i][j]表示 i的第2^j
个祖先
int deg[maxn];                                                   //LCA用，表示第i个节点的深度，
根节点的深度
void Init()                                                      //链式前向星初始化
{
    tot = 0;
    mem(head, -1);
}
void addedge(int u, int v) //链式前向星加边
{
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}

int update(int o, int l, int r, int k)//主席树更新
{
    int oo = ++node_cnt; //生成新的节点
    lc[oo] = lc[o];
    rc[oo] = rc[o];
    sum[oo] = sum[o] + 1; //因为新的版本的线段树一定比旧的版本多一个数，所以权值加一
```

```
    if (l == r)
        return oo;
    int mid = MID(l, r);
    if (k <= mid) //更新的值在左半区间
        lc[oo] = update(lc[oo], l, mid, k);
    else //更新的值在右半区间内
        rc[oo] = update(rc[oo], mid + 1, r, k);
    return oo;//返回新节点的标号
}
int query(int u, int v, int u1, int v1, int l, int r, int k)
{
    if (l == r)
        return l;
    int ans, mid = MID(l, r);
    int x = sum[lc[v]] + sum[lc[u]] - sum[lc[u1]] - sum[lc[v1]]; //获取区间[1,v]-[1,u-1]的左子
树上的权值
    if (x >= k)                                                  //权值大于查询的值，数在左半区
间
        ans = query(lc[u], lc[v], lc[u1], lc[v1], l, mid, k);
    else
        ans = query(rc[u], rc[v], rc[u1], rc[v1], mid + 1, r, k - x);
    return ans;
}
int get(int x)//获得x在离散数组中的位置
{
    return lower_bound(b + 1, b + 1 + q, x) - b;
}
void DFS(int root, int fu) //DFS求深度 LCA用
{

    deg[root] = deg[fu] + 1;
    fa[root][0] = fu;
    for (int i = 1; i < DEG; i++)
    {
        fa[root][i] = fa[fa[root][i - 1]][i - 1];
    }
    rt[root] = update(rt[fu], 1, q, get(a[root]));
    for (int i = head[root]; i != -1; i = edge[i].next)
    {
        int v = edge[i].to;
        if (v == fu)
            continue;
        DFS(v, root);
    }
}
int LCA(int u, int v)
{
    if (deg[u] > deg[v])
        swap(u, v);
    int hu = deg[u], hv = deg[v];
    int tu = u;
    int tv = v;
    for (int det = hv - hu, i = 0; det; det >>= 1, i++)//让tv和tu处于同一深度
        if (det & 1)
            tv = fa[tv][i];//深的节点向上走
    if (tu == tv)
        return tv;
```

```
    for (int i = DEG - 1; i >= 0; i--)
    {
        if (fa[tu][i] == fa[tv][i])
            continue;
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];//tu的父亲节点即为LCA(u,v)
}
int main()
{
    node_cnt = 0;
    int l, r, k, ans;
    cin >> n >> m;
    Init();
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]);
        b[i] = a[i];
    }
    sort(b + 1, b + 1 + n);
    q = unique(b + 1, b + 1 + n) - b - 1;
    for (int i = 0; i < n - 1; i++)
    {
        scanf("%d%d", &l, &r);
        addedge(l, r);
        addedge(r, l);
    }
    mem0(fa);//初始化
    mem0(deg);//
    DFS(1, 0);
    while (m--)
    {
        scanf("%d%d%d", &l, &r, &k);
        int lca = LCA(l, r);
        cout << b[query(rt[l], rt[r], rt[lca], rt[fa[lca][0]], 1, q, k)] << endl;
    }
    system("pause");
    return 0;
}
```

17.4. 动态第k大

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a, b) make_pair(a, b)
#define F first
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
```

```
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const int INF = 0x3f3f3f3f;
const int maxn = 60100;
const double pi = acos(-1.0);
int node_cnt, n, m;
int sum[maxn << 5], rt[maxn << 5], lc[maxn << 5], rc[maxn << 5]; //线段树相关 sum记录线段树权值
int a[maxn], b[maxn];                                           //原数组和离散数组
int S[maxn];                                                    //S为动态主席树的根节点,
int btmp;                                                       //离散化后的数的个数
int use[maxn];                                                  //use是用来存储更新时,沿着
lowbit上升时经过的树
struct Query
{
    int l, r, op;
    int k;
} que[maxn];
int getid(int x)
{
    return lower_bound(b + 1, b + 1 + btmp, x) - b;
}
inline int lowbit(int x)
{
    return x & (-x);
}
void build(int &t, int l, int r)
{
    t = ++node_cnt;
    sum[t] = 0;
    if (l == r)
        return;
    int mid = MID(l, r);
    build(lc[t], l, mid);
    build(rc[t], mid + 1, r);
}
int update(int o, int l, int r, int p, int v) //建立一棵新的树
{
    int oo = ++node_cnt;
    lc[oo] = lc[o];
    rc[oo] = rc[o];
    sum[oo] = sum[o] + v;
    if (l == r)
        return oo;
    int mid = MID(l, r);
    if (p <= mid)
        lc[oo] = update(lc[oo], l, mid, p, v);
    else
        rc[oo] = update(rc[oo], mid + 1, r, p, v);
    return oo;
}
void add(int x, int pos, int v)
{
    while (x <= n)
    {
        S[x] = update(S[x], 1, btmp, pos, v);
        x += lowbit(x);
```

```
        }
    }
    int Sum(int x)
    {
        int s = 0;
        while (x > 0)
        {
            s += sum[lc[use[x]]];
            x -= lowbit(x);
        }
        return s;
    }
    int query(int root_left, int root_right, int left, int right, int l, int r, int k)
    {
        if (l == r)
            return l;
        int tot = 0;
        int mid = MID(l, r);
        tot = Sum(right) - Sum(left - 1) + sum[lc[root_right]] - sum[lc[root_left]];
        if (tot >= k)
        {
            for (int i = right; i > 0; i -= lowbit(i))
                use[i] = lc[use[i]];
            for (int i = left - 1; i > 0; i -= lowbit(i))
                use[i] = lc[use[i]];
            return query(lc[root_left], lc[root_right], left, right, l, mid, k);
        }
        else
        {
            for (int i = right; i > 0; i -= lowbit(i))
                use[i] = rc[use[i]];
            for (int i = left - 1; i > 0; i -= lowbit(i))
                use[i] = rc[use[i]];
            return query(rc[root_left], rc[root_right], left, right, mid + 1, r, k - tot);
        }
    }
    int main()
    {
        int T;
        cin >> T;
        while (T--)
        {
            btmp = 0;
            node_cnt = 0;
            scanf("%d%d", &n, &m);
            for (int i = 1; i <= n; i++)
            {
                scanf("%d", &a[i]);
                b[++btmp] = a[i];
            }
            char op[10];
            for (int i = 1; i <= m; i++)
            {
                scanf("%s", op);
                if (op[0] == 'Q')
                {
                    que[i].op = 0;
```

```
                scanf("%d%d%d", &que[i].l, &que[i].r, &que[i].k);
            }
            else
            {
                que[i].op = 1;
                scanf("%d%d", &que[i].l, &que[i].r);
                b[++btmp] = que[i].r;
            }
        }
        //至于为什么要先把所有的修改的节点找出来,才进行建树,那是因为只有知道序列的范围才可以建树,无他

        sort(b + 1, b + 1 + btmp);
        btmp = unique(b + 1, b + 1 + btmp) - b - 1;
        build(rt[0], 1, btmp);
        for (int i = 1; i <= n; i++)
        {
            rt[i] = update(rt[i - 1], 1, btmp, getid(a[i]), 1);
        }
        for (int i = 1; i <= n; i++)
            S[i] = rt[0]; //为每个树状数组根节点初始化
        for (int i = 1; i <= m; i++)
        {
            if (que[i].op == 0)
            {
                for (int j = que[i].r; j > 0; j -= lowbit(j))
                    use[j] = S[j];
                for (int j = que[i].l - 1; j > 0; j -= lowbit(j))
                    use[j] = S[j];
                int ans = query(rt[que[i].l - 1], rt[que[i].r], que[i].l, que[i].r, 1, btmp,
que[i].k);
                printf("%d\n", b[ans]);
            }
            else
            {
                add(que[i].l, getid(a[que[i].l]), -1); //先消除影响
                add(que[i].l, getid(que[i].r), 1);     //再新建影响
                a[que[i].l] = que[i].r;
            }
        }
    }
    system("pause");
    return 0;
}
```

17.5. 区间修改，区间求和（自己写的代码未过）

```
#include <string.h>
#include <algorithm>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
```

```
using namespace std;
typedef long long int LL;
const int maxn=1e5;
int rt[maxn*35+5];
int ls[maxn*35+5];
int rs[maxn*35+5];
int p;
LL sum[maxn*35+5];
LL pos[maxn*35+5];
int n,m,t;
int newnode()
{
    ls[p]=rs[p]=sum[p]=pos[p]=0;
    return p++;
}
void build(int &node,int begin,int end)
{
    if(!node) node=newnode();
    if(begin==end)
    {
        scanf("%lld",&sum[node]);
        return;
    }
    int mid=(begin+end)>>1;
    build(ls[node],begin,mid);
    build(rs[node],mid+1,end);
    sum[node]=sum[ls[node]]+sum[rs[node]];
}
void update(int &node,int begin,int end,int left,int right,int val)
{
    sum[p]=sum[node];ls[p]=ls[node];rs[p]=rs[node];
    pos[p]=pos[node];
    node=p;p++;
    sum[node]+=1LL*val*(right-left+1);
    if(left==begin&&end==right)
    {
        pos[node]+=val;
        return;
    }
    int mid=(begin+end)>>1;
    if(right<=mid) update(ls[node],begin,mid,left,right,val);
    else if(left>mid) update(rs[node],mid+1,end,left,right,val);
    else
    {
        update(ls[node],begin,mid,left,mid,val);
        update(rs[node],mid+1,end,mid+1,right,val);
    }
}
LL query(int node,int begin,int end,int left,int right)
{
    if(left<=begin&&end<=right) return sum[node];
    LL ret=1LL*pos[node]*(right-left+1);
    int mid=(begin+end)>>1;
    if(right<=mid)  ret+=query(ls[node],begin,mid,left,right);
    else if(left>mid)  ret+=query(rs[node],mid+1,end,left,right);
    else
    {
```

```
        ret+=(query(ls[node],begin,mid,left,mid)+query(rs[node],mid+1,end,mid+1,right));

    }
    return ret;
}
int main()
{
    char x;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        t=0;
        p=0;
    build(rt[0],1,n);
    int l,r,d,time;

    LL ans;
    for(int i=1;i<=m;i++)
    {
        cin>>x;
        if(x=='C')
        {
            scanf("%d%d%d",&l,&r,&d);
            update(rt[++t]=rt[t-1],1,n,l,r,d);
        }
        else if(x=='Q')
        {
            scanf("%d%d",&l,&r);
            ans=query(rt[t],1,n,l,r);
            printf("%lld\n",ans);
        }
        else if(x=='H')
        {
            scanf("%d%d%d",&l,&r,&time);
            ans=query(rt[time],1,n,l,r);
            printf("%lld\n",ans);
        }
        else
        {
            scanf("%d",&time);
            t=time;
        }

    }
    }
    return 0;
}
```

18. 最近公共祖先（最小公共祖先）（LCA）

18.1. 倍增法

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define mkp(a,b) make_pair(a,b)
#define F first
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const int INF = 0x3f3f3f3f ;
const int maxn = 10000;
const double pi = acos(-1.0);
int T,n,m;
const int DEG=20;
struct Edge//链式前向星的边结构体
{
    int to,next;
}edge[maxn*2];
int head[maxn],tot;
void addedge(int u,int v)//链式前向星加边
{
    edge[tot].to=v;
    edge[tot].next=head[u];
    head[u]=tot++;
}
void init()//初始化
{
    tot=0;
    mem(head,-1);
}
int fa[maxn][DEG];//表示节点i的第2^j个祖先
int deg[maxn];//深度数组
void BFS(int root)//BFS求深度
{
    queue<int>que;
    deg[root]=0;//
    fa[root][0]=root;//初始化根节点的父亲是root
    que.push(root);
    while(!que.empty())
    {
        int tmp=que.front();
        que.pop();
        for(int i=1;i<DEG;i++)//找到这个节点所有的祖先节点
        {
            fa[tmp][i]=fa[fa[tmp][i-1]][i-1];//2^i=2^(i-1)+2^(i-1)表示 tmp向上跳2的i次的节点
=tmp向上跳i-1次方的这个节点向上跳i-1次方
        }
        for(int i=head[tmp];i!=-1;i=edge[i].next)
        {
```

```
            int v=edge[i].to;
            if(v==fa[tmp][0])continue;//v等于tmp的父亲节点，跳过
            deg[v]=deg[tmp]+1;//记录深度
            fa[v][0]=tmp;//记录父亲
            que.push(v);//入队
        }
    }
}
void DFS(int root,int fu)//DFS求深度
{
    deg[root]=deg[fu]+1;
    fa[root][0]=fu;
    for(int i=1;i<DEG;i++)
    {
        fa[root][i]=fa[fa[root][i-1]][i-1];
    }
    for(int i=head[root];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v==fu)continue;
        DFS(v,root);
    }
}
int LCA(int u,int v)
{
    if(deg[u]>deg[v])swap(u,v);//
    int hu=deg[u],hv=deg[v];
    int tu=u;
    int tv=v;
    for(int det=hv-hu,i=0;det;det>>=1,i++)//让u和v跳到同一个深度
        if(det&1)
            tv=fa[tv][i];
    if(tu==tv)return tu;//如果tu==tv说明在同一边
    for(int i=DEG-1;i>=0;i--)
    {
        if(fa[tu][i]==fa[tv][i])continue;
        tu=fa[tu][i];
        tv=fa[tv][i];
    }
    return fa[tu][0];//父亲节点即为lca
}
int main()
{
    system("pause");
    return 0;
}
```

19. 最长上升子序列

时间复杂度 $nlog(n)$

```

```

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define sc(x) scanf("%d",&x)
#define scc(x,y) scanf("%d%d",&x,&y)
#define sccc(x,y,z) scanf("%d%d%d",&x,&y,&z)
#define mkp(a,b) make_pair(a,b)
#define F first
#define ep(a) emplace_back(a)
#define S second
#define pii pair<int,int>
#define mem0(a) memset(a,0,sizeof(a))
#define mem(a,b) memset(a,b,sizeof(a))
#define MID(l,r) (l+((r-l)>>1))
#define ll(o) (o<<1)
#define rr(o) (o<<1|1)
const LL INF = 0x3f3f3f3f ;
const int maxn = 500010;
const double pi = acos(-1.0);
int T,n,m;
int a[maxn],k[maxn];
int cool[maxn];
int ans[maxn];
int vis[maxn];
int lis[maxn];
int path[maxn];
int solve()
{
    int len=0;
    for(int i=1;i<=n;i++)
    {
        if(cool[i])continue;
        int p=lower_bound(lis,lis+len,a[i])-lis;
        if(p==len)
        {
            lis[len++]=a[i];
            path[i]=len;
        }
        else
        {
            lis[p]=a[i];
            path[i]=p+1;
        }
        vis[i]=0;
    }
    int tmp=len;
    for(int i=n;i>=1;--i)
    {
        if(cool[i])continue;
        if(path[i]==tmp)vis[a[i]]=1,tmp--;
    }
    return len;
}
int main()
{
    cin>>T;
```

```
    while(T--)
    {
        scanf("%d",&n);
        for(int i=1;i<=n;i++)vis[i]=cool[i]=0;
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        for(int i=1;i<=n;i++)scanf("%d",&k[i]);
        ans[n]=solve();
        for(int i=n-1;i>=1;i--)
        {
            cool[k[i+1]]=1;
            if(vis[a[k[i+1]]]){
                ans[i]=solve();
                //cout<<i<<" !!!\n";
            }
            else ans[i]=ans[i+1];
        }
        for(int i=1;i<=n;i++){
            printf("%d",ans[i]);
            if(i!=n)printf(" ");
            else printf("\n");
        }
    }
    system("pause");
    return 0;
}
```

```
#include<iostream>
using namespace std;
// 使用二分法的前提是 数组是已经排好序的（刚好在这里我们的d数组就是递增数组）
// 查找返回d数组中第一个比x大的值的下标
int er_method(int a[],int len,int x)
{
    int mid,l=1;
    while(l<=len)
    {
        mid = (l+len)/2;
        if(a[mid]<=x)
            l = mid+1;
        else
            len = mid-1;
    }
    return l;// 此时mid等于l
}
int main()
{
    int n;
    while(cin>>n)
    {
        //待测的数组-------    a
        //存放最长子序列---     d
        //记录最长子序列的长度-- len
        int a[101],d[101],len=1;
        for(int i=1;i<=n;i++)
        {
```

```
            cin>>a[i];
        }
        d[1]=a[1];   // 第一个待测数字就d的第一个
        if(n<2)  // 如果待测数字只有一个 ，那就这个数字就是最长子序列
        {
            cout<<1<<endl;
            continue;
        }
        for(int i=2;i<=n;i++)
        {
            if(a[i]>d[len]) // 如果第 i个数大于d数组的最大的数，直接接在d数组的后面
                d[++len] = a[i];
            else    // 比d数组的最大数值小的的话，那就在d数组中找到一个比它大的数，替换掉
                d[er_method(a,len,a[i]] = a[i]; // 这一步不改变d 数组的长度
        }
        cout<<len<<endl;
    }
    return 0;
}
```

20. 可撤销并查集

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
#define sc(x) scanf("%d", &x)
#define scc(x, y) scanf("%d%d", &x, &y)
#define sccc(x, y, z) scanf("%d%d%d", &x, &y, &z)
#define mkp(a, b) make_pair(a, b)
#define F first
#define ep(a) push_back(a)
#define S second
#define pii pair<int, int>
#define mem0(a) memset(a, 0, sizeof(a))
#define mem(a, b) memset(a, b, sizeof(a))
#define MID(l, r) (l + ((r - l) >> 1))
#define ll(o) (o << 1)
#define rr(o) (o << 1 | 1)
const LL INF = 0x3f3f3f3f;
const int maxn =200010;
const double pi = acos(-1.0);
int T, n, m;
vector<int> vec[maxn<<2];
struct Edge
{
    int u, v, l, r;
} edge[maxn];
int fa[maxn];
int Rank[maxn];
int ans = 0;
int sz[maxn];
```

```
int cnt=0;
int par(int u)
{
    return fa[u] == u ? u : par(fa[u]);
}
void dfs(int o, int l, int r)
{
    vector<int> che;
    int len = vec[o].size();
    for (int i = 0; i < len; i++)
    {
        int id = vec[o][i];
        int u = edge[id].u;
        int v = edge[id].v;
        //并查集合并
        u = par(u);
        v = par(v);
        if (Rank[u] > Rank[v])
            swap(u, v);
        fa[u] = v;
        Rank[v] += Rank[u];
        che.ep(u);
    }
    int mid = MID(l, r);
   // cout<<"----- "<<r<<" "<<l<<" "<<par(1)<<" "<<par(n)<<endl;
    if (par(1) == par(n))
    {
        ans += sz[r+1] - sz[l];
    }
    else if (l < r)
    {
        dfs(ll(o), l, mid);
        dfs(rr(o), mid + 1, r);
    }
    //并查集撤销
    len = che.size();
    for (int i = len - 1; i >= 0; i--)
    {
        int x = che[i];
        fa[x] = x;
    }
    che.clear();
    //撤销结束
}

void Insert(int o, int l, int r, int L, int R, int x)
{
    if (L <= l && r <= R)
    {
        vec[o].ep(x);
        return;
    }
    int mid = MID(l, r);
    if (L <= mid)
        Insert(ll(o), l, mid, L, R, x);
    if (R > mid)
        Insert(rr(o), mid + 1, r, L, R, x);
```

```
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
    {
        fa[i] = i;
        Rank[i] = 1;
    }
    for (int i = 1; i <= m; i++)
    {
        scanf("%d%d%d%d", &edge[i].u, &edge[i].v, &edge[i].l, &edge[i].r);
        edge[i].r++;
        sz[++cnt]=edge[i].l,sz[++cnt]=edge[i].r;
        //
    }
    sort(sz+1,sz+1+cnt);
    cnt=unique(sz+1,sz+1+cnt)-sz-1;
    for(int i=1;i<=m;i++)
    {
        edge[i].l=lower_bound(sz+1,sz+1+cnt,edge[i].l)-sz;
        edge[i].r=lower_bound(sz+1,sz+1+cnt,edge[i].r)-sz;
        Insert(1, 1, cnt-1, edge[i].l, edge[i].r-1, i);
    }
    dfs(1,1,cnt-1);
    cout<<ans;
    //system("pause");
    return 0;
}
```

21. 差分

差分就是将数列中的每一项分别与前一项数做差，例如：

一个序列1 2 5 4 7 3，差分后得到1 1 3 -1 3 -4 -3

这里注意得到的差分序列第一个数和原来的第一个数一样（相当于第一个数减0）

差分序列最后比原序列多一个数（相当于0减最后一个数）

性质：

1、差分序列求前缀和可得原序列

2、将原序列区间[L,R]中的元素全部+1，可以转化操作为差分序列L处+1，R+1处-1

3、按照性质2得到，每次修改原序列一个区间+1，那么每次差分序列修改处增加的和减少的相同

22. 数位dp

22.1. 模板

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)//不是每个题都要判断前导零
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数我枚举完了
    if(pos==-1) return 1;/*这里一般返回1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回1 */
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面会讲*/
    int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
    ll ans=0;
    //开始计数
    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
        /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
        大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
        要求数位上不能有62连续出现,那么就是state就是要保存前一位pre,然后分类，
        前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法*/
    }
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
    return ans;
}
ll solve(ll x)
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛
}
int main()
```

```
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化dp数组为-1,这里还有更加优美的优化,后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```