线段树分治

算法简介

一种将修改和询问操作转移到线段树上，最后通过遍历线段树求解的离线算法。

算法例题

洛谷p5787

题意

初始给定 n 个点，图上没有边。现有 m 条边，每条边出现的时间为 $[l_i,r_i]$询问每个时刻图是否为二分图。

题解

考虑对时间序列建线段树，然后将每条边插入线段树。

遍历线段树，同时使用拓展域并查集动态维护此时是否为二分图。如果当前区间已经不是二分图或者遍历到叶子节点即可得到答案并回溯。

由于修改操作需要支持回溯，所有考虑使用按秩合并的可撤销并查集。

修改次数为 $O(m\log n)$于是总时间复杂度 $O(m\log^2 n)$

```
const int MAXN=1e5+5;
vector<pair<int,int> >a[MAXN<<2];
int lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);build(k<<1|1,M+1,R);
}
void add(int k,int L,int R,pair<int,int> edge){
    if(L<=lef[k]&&rig[k]<=R)
    return a[k].push_back(edge),void();
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)add(k<<1,L,R,edge);
    if(mid<R)add(k<<1|1,L,R,edge);
}
int n,fa[MAXN<<1],dep[MAXN<<1],top;
pair<int,bool> Stack[MAXN<<2];
int Find(int x){return x==fa[x]?x:Find(fa[x]);}
```

```
void Union(int u,int v){
    int x=Find(u),y=Find(v);
    if(x==y)return;
    if(dep[x]>dep[y])swap(x,y);
    Stack[++top]=make_pair(x,dep[x]==dep[y]);
    fa[x]=y,dep[y]+=dep[x]==dep[y];
}
void solve(int k){
    int cur=top;bool flag=true;
    _for(i,0,a[k].size()){
        pair<int,int> temp=a[k][i];
        if(Find(temp.first)==Find(temp.second)){
            _rep(i,lef[k],rig[k])puts("No");
            flag=false;
            break;
        }
        Union(temp.first,temp.second+n),Union(temp.first+n,temp.second);
    }
    if(flag){
        if(lef[k]==rig[k])puts("Yes");
        else solve(k<<1),solve(k<<1|1);
    }
    while(top>cur){
        dep[fa[Stack[top].first]]-=Stack[top].second;
        fa[Stack[top].first]=Stack[top].first;
        top--;
    }
}
int main()
{
    int n=read_int(),m=read_int(),k=read_int();
    ::n=n;
    build(1,1,k);
    _rep(i,1,n<<1)fa[i]=i,dep[i]=1;
    while(m--){
        int u=read_int(),v=read_int(),l=read_int(),r=read_int();
        if(l!=r)add(1,l+1,r,make_pair(u,v));
    }
    solve(1);
    return 0;
}
```

算法练习

习题一

牛客暑期多校(第八场) A 题

题意

现有 n 个球员，m 个球迷。每个球员有 k_i 个粉丝。

球迷 i 喜欢球员 j 当且仅当以下两个条件至少有一个满足：

1. 球迷 i 是球员 j 的粉丝
2. 球迷 i 和 i' 都喜欢球员 j' 且球迷 i' 喜欢球员 j

接下来 q 次修改操作，每次修改球迷 x 与球员 y 之间粉丝关系。

每次修改后询问至少需要选择多少个球员才能保证每个球迷至少有一个喜欢的球员。如果没有满足条件的方案，输出 -1。

题解

把球员和球迷当成点，粉丝关系视为边，于是题目答案等价于连通块个数减去孤立球员个数。

特殊情况为如果存在孤立球迷，此时答案为 -1。

考虑对时间序列建线段树，然后可撤销并查集维护连通块个数、孤立球员个数、孤立球迷个数即可。

时间复杂度 $O\left((q+\sum_{i=1}^n k_i)\log q\log n\right)$。

```
const int MAXN=2e5+5,MAXK=7e5+5;
vector<pair<int,int> >a[MAXN<<2];
int lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);build(k<<1|1,M+1,R);
}
void add(int k,int L,int R,pair<int,int> edge){
    if(L<=lef[k]&&rig[k]<=R)
    return a[k].push_back(edge),void();
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)add(k<<1,L,R,edge);
    if(mid<R)add(k<<1|1,L,R,edge);
}
int m,fa[MAXN<<1],sz[MAXN<<1],top,Stack[MAXK],block_cnt,Iosn,Iosm;
int Find(int x){return x==fa[x]?x:Find(fa[x]);}
void Union(int u,int v){
    int x=Find(u),y=Find(v);
    if(x==y)return;
    block_cnt--;
    if(sz[x]==1){
        if(x>m)Iosn--;
        else Iosm--;
    }
```

```
    if(sz[y]==1){
        if(y>m)Iosn--;
        else Iosm--;
    }
    if(sz[x]>sz[y])swap(x,y);
    Stack[++top]=x;
    fa[x]=y,sz[y]+=sz[x];
}
void Inv_union(int x,int y){
    block_cnt++;
    sz[y]-=sz[x];
    fa[x]=x;
    if(sz[x]==1){
        if(x>m)Iosn++;
        else Iosm++;
    }
    if(sz[y]==1){
        if(y>m)Iosn++;
        else Iosm++;
    }
    top--;
}
void solve(int k){
    int cur=top;
    _for(i,0,a[k].size()){
        pair<int,int> temp=a[k][i];
        Union(temp.first,temp.second);
    }
    if(lef[k]==rig[k]){
        if(Iosm)
        puts("-1");
        else
        enter(block_cnt-Iosn);
    }
    else
    solve(k<<1),solve(k<<1|1);
    while(top>cur)Inv_union(Stack[top],fa[Stack[top]]);
}
map<pair<int,int>,int> st;
int main()
{
    int n=read_int(),m=read_int(),q=read_int(),u,v;
    pair<int,int> temp;
    ::m=m;
    _rep(i,1,n+m)fa[i]=i,sz[i]=1;
    block_cnt=n+m,Iosn=n,Iosm=m;
    build(1,1,q);
    _rep(i,1,n){
        int k=read_int();
        while(k--)
```

```
        st[make_pair(read_int(),i+m)]=1;
    }
    _rep(i,1,q){
        u=read_int(),v=read_int()+m;
        temp=make_pair(u,v);
        if(st.count(temp)){
            int l=st[temp];
            if(l!=i)add(1,l,i-1,temp);
            st.erase(temp);
        }
        else
        st[temp]=i;
    }
    for(auto it=st.begin();it!=st.end();++it)
    add(1,it->second,q,it->first);
    solve(1);
    return 0;
}
```

习题二

洛谷p4585

题意

有 n 个商店，编号为 $1\sim n$每个商店有一个价格为 v_i 的特殊商品。接下来 m 个操作：

1. 时间流逝一个单位，同时商店 s 额外进货一种价格为 v 的商品
2. 给定 l,r,d,x 表示一个顾客的信息，询问该顾客的最大满意度。(满意度等于 x 异或顾客可以购买的商品价格)

记询问时刻为 t该顾客只能在编号为 $[l,r]$ 的商店购物，且只能购买特殊商品或进货时间为 $[t-d+1,t]$ 的商品，问该顾客的最大满意度。

注意初始时刻为 0，且询问不会导致时间流逝。

题解

对与区间询问最大异或和操作可以用可持久化字典树 $O(\log v)$ 解决，于是可以先 $O((n+m)\log v)$ 处理出只考虑特殊商品时每个询问的答案。

接下来考虑对时间建立线段树，同时将每个询问加入线段树，遍历线段树时更新答案。

对于修改即进货操作，考虑先对其根据商店编号进行排序，这样就可以使用可持久化字典树加离散化处理询问。

然后遍历线段树时处理完当前节点询问后可以根据修改的时间分配到左右子树处理，每次处理询问重建字典树即可。

时间复杂度 $O((n+m)\log v\log m)$

```
const int MAXN=1e5+5,MAXM=17;
int root[MAXN],ch[MAXN*20][2],sz[MAXN*20],cnt;
void Insert(int &k,int p,int v,int pos=MAXM-1){
    k=++cnt;
    sz[k]=sz[p]+1;
    if(pos<0)return;
    int dir=(v>>pos)&1;
    ch[k][!dir]=ch[p][!dir];
    Insert(ch[k][dir],ch[p][dir],v,pos-1);
}
int query(int lef,int rig,int v){
    int pos=MAXM-1,k1=root[lef-1],k2=root[rig],ans=0;
    while(~pos){
        int dir=(v>>pos)&1;
        if(sz[ch[k2][!dir]]-sz[ch[k1][!dir]])
        ans|=(1<<pos),k1=ch[k1][!dir],k2=ch[k2][!dir];
        else
        k1=ch[k1][dir],k2=ch[k2][dir];
        pos--;
    }
    return ans;
}
struct Query{
    int ql,qr,tl,tr,x,id;
}q[MAXN];
struct Upd{
    int pos,v,t;
    bool operator < (const Upd &b)const{
        return pos<b.pos;
    }
}upd[MAXN];
int lef[MAXN<<2],rig[MAXN<<2];
vector<int>a[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);build(k<<1|1,M+1,R);
}
void update(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return a[k].push_back(v);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
}
int opt[MAXN],cur,ans[MAXN];
```

```
int Find(int p,int l,int r){
    int lef=l,rig=r,mid,ans=l-1;
    while(lef<=rig){
        mid=lef+rig>>1;
        if(upd[opt[mid]].pos<=p)ans=mid,lef=mid+1;
        else rig=mid-1;
    }
    return ans-l+1;
}
void cal(int k,int L,int R){
    cnt=cur=0;
    _rep(i,L,R)Insert(root[cur+1],root[cur],upd[opt[i]].v),cur++;
    _for(i,0,a[k].size()){
        int l=Find(q[a[k][i]].ql-1,L,R)+1,r=Find(q[a[k][i]].qr,L,R);
if(l<=r)ans[q[a[k][i]].id]=max(ans[q[a[k][i]].id],query(l,r,q[a[k][i]].x));
    }
}
int t1[MAXN],t2[MAXN];
void solve(int k,int L,int R){
    if(L>R)return;
    cal(k,L,R);
    if(lef[k]==rig[k])return;
    int cnt1=0,cnt2=0,mid=lef[k]+rig[k]>>1,M=L-1;
    _rep(i,L,R){
        if(upd[opt[i]].t<=mid)t1[cnt1++]=opt[i];
        else t2[cnt2++]=opt[i];
    }
    _for(i,0,cnt1)opt[++M]=t1[i];
    _for(i,0,cnt2)opt[M+1+i]=t2[i];
    solve(k<<1,L,M);solve(k<<1|1,M+1,R);
}
int main()
{
    int n=read_int(),m=read_int(),cnt1=0,cnt2=0,k;
    _rep(i,1,n)Insert(root[i],root[i-1],read_int());
    while(m--){
        k=read_int();
        if(k==0){
            cnt1++;
upd[cnt1].pos=read_int(),upd[cnt1].v=read_int(),upd[cnt1].t=cnt1,opt[cnt1]=
cnt1;
        }
        else{
            cnt2++;
q[cnt2].ql=read_int(),q[cnt2].qr=read_int(),q[cnt2].x=read_int(),q[cnt2].id
=cnt2;
            q[cnt2].tl=max(1,cnt1-read_int()+1),q[cnt2].tr=cnt1;
            ans[cnt2]=query(q[cnt2].ql,q[cnt2].qr,q[cnt2].x);
        }
    }
    if(!cnt1){
```

```
        _rep(i,1,cnt2)enter(ans[i]);
        return 0;
    }
    build(1,1,cnt1);
    _rep(i,1,cnt2)if(q[i].tl<=q[i].tr)
    update(1,q[i].tl,q[i].tr,i);
    sort(upd+1,upd+cnt1+1);
    solve(1,1,cnt1);
    _rep(i,1,cnt2)enter(ans[i]);
    return 0;
}
```

习题三

洛谷p3733

题意

给定一个带边权连通图，接下来三种操作：

1. 新增一条边，新增边从 1 开始编号
2. 删除某条新增边，保证新增边一定存在
3. 修改某条新增边的权值，保证新增边一定存在

要求输出初始时所有经过点 1 的回路的边权异或和最大值和每次操作后的最大值。其中边权为二进制数，且长度 $L\le 1000$。

题解 1

首先建立 dfs 树，于是题目转化为求所有非树边代表环的边权的集合的子集的异或和最大值。

不难想到线性基加 bitset 维护。考虑对时间序列建立线段树，操作 $1,2,3$ 不难转化为修改操作加入线段树。

同时考虑保存线段树分治时当前遍历的链上的所有线性基即可完成回溯操作。时间复杂度 $O\left(\cfrac{(m+q)L^2\log q}{w}\right)$。

```
const int MAXN=1005,MAXD=15;
typedef bitset<MAXN> bt;
struct LB{
    bt p[MAXN];
    void insert(bt x){
        for(int i=MAXN-1;i>=0;i--){
            if(x[i]){
                if(p[i].any())
                x^=p[i];
                else
```

```
                return p[i]=x,void();
            }
        }
    }
    bt query(){
        bt ans(0);
        for(int i=MAXN-1;i>=0;i--){
            if(!ans[i]&&p[i][i])
            ans^=p[i];
        }
        return ans;
    }
}lb[MAXD];
void Enter_bitset(const bt& x){
    bool flag=false;
    for(int i=MAXN-1;i>=0;i--){
        if(x[i])flag=true;
        if(flag)putchar('0'+x[i]);
    }
    if(!flag)putchar('\0');
    putchar('\n');
}
char buf[MAXN];
bt Read_bitset(){
    bt ans(0);
    scanf("%s",buf);
    int len=strlen(buf);
    reverse(buf,buf+len);
    _for(i,0,len)
    ans[i]=buf[i]-'0';
    return ans;
}
vector<bt> s[MAXN<<2];
int lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void update(int k,int L,int R,const bt& v){
    if(L<=lef[k]&&rig[k]<=R){
        s[k].push_back(v);
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)
    update(k<<1,L,R,v);
    if(mid<R)
    update(k<<1|1,L,R,v);
```

```
}
void solve(int k,int d){
    _for(i,0,s[k].size())
    lb[d].insert(s[k][i]);
    if(lef[k]==rig[k]){
        Enter_bitset(lb[d].query());
        return;
    }
    lb[d+1]=lb[d];
    solve(k<<1,d+1);
    lb[d+1]=lb[d];
    solve(k<<1|1,d+1);
}
struct Edge{
    int to,next;
    bt w;
}edge[MAXN];
int head[MAXN],edge_cnt;
void AddEdge(int u,int v,const bt& w){
    edge[++edge_cnt]=Edge{v,head[u],w};
    head[u]=edge_cnt;
}
bt dis[MAXN];
int dfn[MAXN],dfs_t;
void dfs(int u){
    dfn[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(!dfn[v]){
            dis[v]=dis[u]^edge[i].w;
            dfs(v);
        }
        else if(dfn[v]<=dfn[u])
        lb[1].insert(dis[u]^dis[v]^edge[i].w);
    }
}
struct Edge2{
    int u,v,last;
    bt w;
}edge2[MAXN];
int main()
{
    int n=read_int(),m=read_int(),q=read_int();
    build(1,0,q);
    while(m--){
        int u=read_int(),v=read_int();
        bt w=Read_bitset();
        AddEdge(u,v,w);
        AddEdge(v,u,w);
    }
```

```
    dis[1]=bt(0);
    dfs(1);
    int k=0;
    _rep(i,1,q){
        scanf("%s",buf);
        if(buf[0]=='A'){
            k++;
            edge2[k].u=read_int(),edge2[k].v=read_int();
            edge2[k].w=Read_bitset(),edge2[k].last=i;
        }
        else if(buf[1]=='a'){
            int t=read_int();
update(1,edge2[t].last,i-1,dis[edge2[t].u]^dis[edge2[t].v]^edge2[t].w);
            edge2[t].last=0;
        }
        else{
            int t=read_int();
update(1,edge2[t].last,i-1,dis[edge2[t].u]^dis[edge2[t].v]^edge2[t].w);
            edge2[t].w=Read_bitset();
            edge2[t].last=i;
        }
    }
    _rep(i,1,k){
        if(edge2[i].last)
update(1,edge2[i].last,q,dis[edge2[i].u]^dis[edge2[i].v]^edge2[i].w);
    }
    solve(1,1);
    return 0;
}
```

题解 2

同样考虑线性基维护，将所有修改操作按左端点排序，然后按右端点贪心处理。时间复杂度 $O\left(\cfrac{(m+q)L^2}w\right)$

```
const int MAXN=1005,MAXD=15;
typedef bitset<MAXN> bt;
namespace LB{
    bt p[MAXN];
    int pos[MAXN];
    void insert(bt x,int y){
        for(int i=MAXN-1;i>=0;i--){
            if(x[i]){
                if(!p[i].any()){
                    p[i]=x;
                    pos[i]=y;
                    break;
                }
                else if(pos[i]<y){
```

```
                    swap(p[i],x);
                    swap(pos[i],y);
                }
                x^=p[i];
            }
        }
    }
    bt query(int l){
        bt ans(0);
        for(int i=MAXN-1;i>=0;i--){
            if(!ans[i]&&p[i][i]&&pos[i]>=l)
            ans^=p[i];
        }
        return ans;
    }
};
void Enter_bitset(const bt& x){
    bool flag=false;
    for(int i=MAXN-1;i>=0;i--){
        if(x[i])flag=true;
        if(flag)putchar('0'+x[i]);
    }
    if(!flag)putchar('\0');
    putchar('\n');
}
char buf[MAXN];
bt Read_bitset(){
    bt ans(0);
    scanf("%s",buf);
    int len=strlen(buf);
    reverse(buf,buf+len);
    _for(i,0,len)
    ans[i]=buf[i]-'0';
    return ans;
}
struct Edge{
    int to,next;
    bt w;
}edge[MAXN];
int head[MAXN],edge_cnt;
void AddEdge(int u,int v,const bt& w){
    edge[++edge_cnt]=Edge{v,head[u],w};
    head[u]=edge_cnt;
}
bt dis[MAXN];
int dfn[MAXN],dfs_t;
void dfs(int u){
    dfn[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
```

```
        if(!dfn[v]){
            dis[v]=dis[u]^edge[i].w;
            dfs(v);
        }
        else if(dfn[v]<=dfn[u])
        LB::insert(dis[u]^dis[v]^edge[i].w,MAXN);
    }
}
struct Edge2{
    int u,v,last;
    bt w;
}edge2[MAXN];
struct Node{
    bt x;
    int l,r;
}node[MAXN];
int idx[MAXN];
struct cmp{
    bool operator () (const int a,const int b)const{
        return node[a].l<node[b].l;
    }
};
int main()
{
    int n=read_int(),m=read_int(),q=read_int();
    while(m--){
        int u=read_int(),v=read_int();
        bt w=Read_bitset();
        AddEdge(u,v,w);
        AddEdge(v,u,w);
    }
    dis[1]=bt(0);
    dfs(1);
    int k=0,q2=0;
    _rep(i,1,q){
        scanf("%s",buf);
        if(buf[0]=='A'){
            k++;
            edge2[k].u=read_int(),edge2[k].v=read_int();
            edge2[k].w=Read_bitset(),edge2[k].last=i;
        }
        else if(buf[1]=='a'){
            int t=read_int();
            node[q2].x=dis[edge2[t].u]^dis[edge2[t].v]^edge2[t].w;
            node[q2].l=edge2[t].last,node[q2].r=i-1;
            q2++;
            edge2[t].last=0;
        }
        else{
            int t=read_int();
            node[q2].x=dis[edge2[t].u]^dis[edge2[t].v]^edge2[t].w;
```

```
            node[q2].l=edge2[t].last,node[q2].r=i-1;
            q2++;
            edge2[t].w=Read_bitset();
            edge2[t].last=i;
        }
    }
    _rep(i,1,k){
        if(edge2[i].last){
            node[q2].x=dis[edge2[i].u]^dis[edge2[i].v]^edge2[i].w;
            node[q2].l=edge2[i].last,node[q2].r=q;
            q2++;
        }
    }
    _for(i,0,q2)
    idx[i]=i;
    sort(idx,idx+q2,cmp());
    int pos=0;
    _rep(i,0,q){
        while(pos<q2&&node[idx[pos]].l<=i){
            LB::insert(node[idx[pos]].x,node[idx[pos]].r);
            pos++;
        }
        Enter_bitset(LB::query(i));
    }
    return 0;
}
```