数据结构练习 1

线段树合并/分裂

习题一

洛谷p2824

题意

给定由 $1\sim n$ 的排列构成的序列 A，要求支持以下下两种操作：

1. 将 $A_{l\sim r}$ 按升序排列
2. 将 $A_{l\sim r}$ 按降序排列

m 次操作后询问位置 p 上的数值。

题解

考虑初始时构造 n 棵权值线段树，每个线段树维护区间 $[i,i](1\le i\le n)$。

每次排序操作将所有代表区间含于 $[l,r]$ 的线段树合并，发现合并完成的同时也完成了排序。

但由于区间 $[l,b]$ 可能已经属于某个区间 $[a,b]$，所以需要线段树分裂将其分裂为 $[a,l-1],[l,b]$。

同样也需要将区间 $[c,d]$ 分裂为 $[c,r],[r+1,d]$，最后将 $[l,b]\cdots [c,r]$ 等区间合并即可，注意打标记维护区间的升序/降序情况。

初始化时间复杂度 $O(n\log n)$，产生点数 $O(n\log n)$，每次分裂时间复杂度 $O(\log n)$，且增加 $O(\log n)$ 个点。

每次合并操作时间复杂度等于合并的点数，于是一定不会超过初始化和分裂生成的总点数。

对于查询操作，考虑单独将区间 $[p,p]$ 分裂出来然后 dfs 沿唯一路径到达叶子结点即可，时间复杂度 $O(\log n)$。

考虑 ODT 维护区间，总时空间复杂度 $O((n+m)\log n)$。

```
const int MAXN=1e5+5,MAXM=60;
struct Node{
    int ch[2],cnt;
}node[MAXN*MAXM];
int root[MAXN],node_cnt;
void push_up(int
k){node[k].cnt=node[node[k].ch[0]].cnt+node[node[k].ch[1]].cnt;}
void update(int &k,int lef,int rig,int pos){
```

```
    k=++node_cnt;
    node[k].cnt++;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(mid>=pos)
    update(node[k].ch[0],lef,mid,pos);
    else
    update(node[k].ch[1],mid+1,rig,pos);
}
int query(int k,int lef,int rig){
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(node[k].ch[0])
    return query(node[k].ch[0],lef,mid);
    else
    return query(node[k].ch[1],mid+1,rig);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].cnt+=node[k2].cnt,void();
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
void split(int &k1,int &k2,int k){
    if(node[k1].cnt==k)return;
    k2=++node_cnt;
    if(k<=node[node[k1].ch[0]].cnt){
        node[k2].ch[1]=node[k1].ch[1];
        node[k1].ch[1]=0;
        split(node[k1].ch[0],node[k2].ch[0],k);
    }
    else
    split(node[k1].ch[1],node[k2].ch[1],k-node[node[k1].ch[0]].cnt);
    push_up(k1);push_up(k2);
}
struct seg{
    int lef,rig,rev;
    bool operator < (const seg &b)const{
        return lef<b.lef;
    }
    seg(int lef,int rig,int rev):lef(lef),rig(rig),rev(rev){}
    seg(int pos){lef=pos;}
};
set<seg> s;
typedef set<seg>::iterator iter;
iter split2(int pos){
    iter it=s.lower_bound(seg(pos));
    if(it!=s.end()&&it->lef==pos)return it;
```

```
    --it;
    int lef=it->lef,rig=it->rig,rev=it->rev;
    if(rev){
        split(root[lef],root[pos],rig-pos+1);
        swap(root[lef],root[pos]);
    }
    else
    split(root[lef],root[pos],pos-lef);
    s.erase(it);s.insert(seg(lef,pos-1,rev));
    return s.insert(seg(pos,rig,rev)).first;
}
int main()
{
    int n=read_int(),m=read_int();
    _rep(i,1,n)update(root[i],1,n,read_int()),s.insert(seg(i,i,0));
    while(m--){
        int opt=read_int(),lef=read_int(),rig=read_int();
        iter r=split2(rig+1),l=split2(lef);
        for(iter it=++l;it!=r;++it)Merge(root[lef],root[it->lef],1,n);
        s.erase(--l,r);
        s.insert(seg(lef,rig,opt));
    }
    int p=read_int();
    split2(p+1);split2(p);
    enter(query(root[p],1,n));
        return 0;
}
```

习题二

洛谷p5298

题意

给定一棵以 1 为根的有根树，每个点有一个权值，且度数不超过 2。

若 i 为叶子结点，则 i 的权值等于 v□否则它的权值有 p 的概率为儿子结点中的较大值□$1-p$ 的概率为儿子节点中的较小值。

假设 1 号节点有 m 中可能取值□V_i 为所有可能值中第 i 小的取值□D_i 为取到第 i 小的取值的概率，询问

$$\sum_{i=1}^m iV_iD_i^2$$

数据保证所有叶子节点权值互异。

题解

先对所有权值进行离散化，设 $D(i,j)$ 表示节点 i 取值为 j 的概率。

对树进行 dfs，如果该节点是叶子节点，直接可知该结点 D 数组在该点权值处为 1，其余处为 0。

如果该节点只有一个子节点，则该节点可以直接继承子节点的 D 数组。

如果该节点有两个子节点，对取值 i 由于叶子节点取值互异，不难得到状态转移：

$$D(u,i)=\left(D(l,i)\left(p_i\sum_{j=1}^{i-1}D(r,j)+(1-p_i)\sum_{j=i+1}^mD(r,j)\right)+D(r,i)\left(p_i\sum_{j=1}^{i-1}D(l,j)+(1-p_i)\sum_{j=i+1}^mD(l,j)\right)\right)$$

考虑用权值线段树维护每个节点的 D 数组。对于叶子节点和只有一个子节点的节点，不难维护。

对于有两个子节点的节点和某个确定的取值 i 显然至多只有一个子节点满足该节点取值为 i 的概率不为 0。

于是不可能出现被合并的两棵线段树拥有重复的叶子节点的情况。于是合并的终止条件一定某个线段树该节点为空。

不妨设左节点的线段树非空，且当前线段树节点代表区间为 $[x,y]$ 于是对 $i\in [x,y]$ 有

$$D(u,i)=D(l,i)\left(p_i\sum_{j=1}^{x-1}D(r,j)+(1-p_i)\sum_{j=y+1}^mD(r,j)\right)$$

考虑线段树合并过程中维护两棵线段树的区间前缀和 $[1,x-1]$ 和后缀和 $[y+1,m]$ 即可。

最后对 1 节点的线段树 dfs 即可得到答案。时空间复杂度 $O(n\log n)$

```
const int MAXN=3e5+5,MAXM=60,Mod=998244353;
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%Mod;
        a=1LL*a*a%Mod;
        b>>=1;
    }
    return ans;
}
const int inv=quick_pow(10000,Mod-2);
struct Node{
    int ch[2],s,lazy;
    void add_lazy(int v){
        s=1LL*s*v%Mod;
        lazy=1LL*lazy*v%Mod;
    }
}node[MAXN*MAXM];
int root[MAXN],node_cnt;
void push_down(int k){
    if(node[k].lazy!=1){
        node[node[k].ch[0]].add_lazy(node[k].lazy);
        node[node[k].ch[1]].add_lazy(node[k].lazy);
```

```
        node[k].lazy=1;
    }
}
void push_up(int
k){node[k].s=(node[node[k].ch[0]].s+node[node[k].ch[1]].s)%Mod;}
void update(int &k,int lef,int rig,int pos){
    k=++node_cnt;
    node[k].lazy=node[k].s=1;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(mid>=pos)
    update(node[k].ch[0],lef,mid,pos);
    else
    update(node[k].ch[1],mid+1,rig,pos);
}
int n,temp_p,p[MAXN],m;
void Merge(int &k1,int k2,int lpre,int lsuf,int rpre,int rsuf){
    if(!k1&&!k2)return;
    else if(k1&&k2){
        push_down(k1);push_down(k2);
        int t1=node[node[k1].ch[0]].s,t2=node[node[k2].ch[0]].s;
Merge(node[k1].ch[0],node[k2].ch[0],lpre,(lsuf+node[node[k1].ch[1]].s)%Mod,
rpre,(rsuf+node[node[k2].ch[1]].s)%Mod);
Merge(node[k1].ch[1],node[k2].ch[1],(lpre+t1)%Mod,lsuf,(rpre+t2)%Mod,rsuf);
        push_up(k1);
    }
    else{
        if(k1)node[k1].add_lazy((1LL*rpre*temp_p+1LL*rsuf*(Mod+1-
temp_p))%Mod);
        else node[k1=k2].add_lazy((1LL*lpre*temp_p+1LL*lsuf*(Mod+1-
temp_p))%Mod);
    }
}
struct Node_2{
    int ch[2],v;
}node2[MAXN];
void dfs(int k){
    if(!node2[k].ch[0])
    update(root[k],1,m,node2[k].v);
    else if(!node2[k].ch[1]){
        dfs(node2[k].ch[0]);
        root[k]=root[node2[k].ch[0]];
    }
    else{
        dfs(node2[k].ch[0]);dfs(node2[k].ch[1]);
        temp_p=node2[k].v;
        root[k]=root[node2[k].ch[0]];
        Merge(root[k],root[node2[k].ch[1]],0,0,0,0);
    }
}
int dfs2(int k,int lef,int rig){
```

```
    if(!k)return 0;
    push_down(k);
    int mid=lef+rig>>1;
    if(lef==rig)return 1LL*mid*p[mid]%Mod*node[k].s%Mod*node[k].s%Mod;
    return (dfs2(node[k].ch[0],lef,mid)+dfs2(node[k].ch[1],mid+1,rig))%Mod;
}
int main()
{
    n=read_int();
    _rep(i,1,n){
        int f=read_int();
        if(f){
            if(node2[f].ch[0])node2[f].ch[1]=i;
            else node2[f].ch[0]=i;
        }
    }
    _rep(i,1,n){
        node2[i].v=read_int();
        if(node2[i].ch[0])node2[i].v=1LL*node2[i].v*inv%Mod;
        else p[++m]=node2[i].v;
    }
    sort(p+1,p+m+1);
    m=unique(p+1,p+m+1)-p;
    _rep(i,1,n)if(!node2[i].ch[0])
    node2[i].v=lower_bound(p+1,p+m,node2[i].v)-p;
    dfs(1);
    enter(dfs2(1,1,m));
    return 0;
}
```

可持久化数据结构

习题一

洛谷p2633

题意

给定一棵点权树，每次询问某条路径上第 k 小的权值，强制在线。

题解

类比区间第 k 小查询，考虑差分，对每个结点建权值线段树维护该结点到根节点之间的所有结点的权值集合。

对于路径询问操作跑 $u+v-\text{LCA}(u,v)-\text{fa}(\text{LCA}(u,v))$ 构成的线段树即可。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5,MAXM=40;
struct Edge{
    int to,next;
```

```
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
namespace LCA{
    int d[MAXN],sz[MAXN],f[MAXN];
    int h_son[MAXN],mson[MAXN],p[MAXN];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u])
            h_son[u]=v,mson[u]=sz[v];
        }
    }
    void dfs_2(int u,int top){
        p[u]=top;
        if(mson[u])dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void init(int root){dfs_1(root,0,0);dfs_2(root,root);}
    int query_lca(int u,int v){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])swap(u,v);
            u=f[p[u]];
        }
        return d[u]<d[v]?u:v;
    }
};
struct Node{
    int ch[2],val;
}node[MAXN*MAXM];
int node_cnt;
int nodecopy(int k){
    node[++node_cnt]=node[k];
    return node_cnt;
}
void update(int &k,int p,int lef,int rig,int pos){
    k=nodecopy(p);
```

```
    node[k].val++;
    if(lef==rig)
    return;
    int mid=lef+rig>>1;
    if(mid<pos)
    update(node[k].ch[1],node[p].ch[1],mid+1,rig,pos);
    else
    update(node[k].ch[0],node[p].ch[0],lef,mid,pos);
}
int query(int k1,int k2,int k3,int k4,int lef,int rig,int rk){
    int mid=lef+rig>>1;
    if(lef==rig)
    return mid;
    int
lc1=node[k1].ch[0],lc2=node[k2].ch[0],lc3=node[k3].ch[0],lc4=node[k4].ch[0]
;
    int lz=node[lc1].val+node[lc2].val-node[lc3].val-node[lc4].val;
    if(rk>lz)
    return
query(node[k1].ch[1],node[k2].ch[1],node[k3].ch[1],node[k4].ch[1],mid+1,rig
,rk-lz);
    else
    return
query(node[k1].ch[0],node[k2].ch[0],node[k3].ch[0],node[k4].ch[0],lef,mid,r
k);
}
int a[MAXN],b[MAXN],root[MAXN],n2;
void dfs(int u,int fa){
    update(root[u],root[fa],1,n2,a[u]);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs(v,u);
    }
}
int query2(int u,int v,int k){
    int lca=LCA::query_lca(u,v);
    return query(root[u],root[v],root[lca],root[LCA::f[lca]],1,n2,k);
}
int main()
{
    int n=read_int(),m=read_int();
    _rep(i,1,n)a[i]=b[i]=read_int();
    sort(b+1,b+n+1);
    n2=unique(b+1,b+n+1)-b;
    _rep(i,1,n)a[i]=lower_bound(b+1,b+n2,a[i])-b;
    _for(i,1,n){
        int u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    LCA::init(1);
```

```
    dfs(1,0);
    int last=0;
    while(m--){
        int u=read_int()^last,v=read_int();
        enter(last=b[query2(u,v,read_int())]);
    }
    return 0;
}
```

习题二

洛谷p2839

题意

给定一个长度为 n 的序列。每次询问给定两个区间 $[l_1,r_1]$ 和 $[l_2,r_2]$□询问所有满足 $l\in [l_1,r_1],r\in [l_2,r_2]$ 的区间 $[l,r]$ 的中位数的最大值。

其中序列下标从 0 开始，中位数定义为所选区间中第 $\lfloor \frac {r-l+1}2\rfloor$ 小的数，强制在线。(数据保证 $l_1\lt r_1\lt l_2\lt r_2$)

题解

考虑二分答案，对每个答案 x□将小于 x 的数设为 -1，其他的数设为 1，于是 x 可以满足条件当且仅当所选区间满足区间和不小于 0。

首先 $[r_1+1,l_2-1]$ 必选，然后考虑取 $[l_1,r_1]$ 的最大后缀和 $[l_2,r_2]$ 的最大前缀即可，这可以用线段树维护。

考虑答案 x 和 $x+1$ 的线段树大部分相同，考虑可持久化线段树维护所有可能答案，答案 $x+1$ 的线段树只需要在答案 x 的线段树上修改即可。

将数据值域离散化即可，时间复杂度 $O(n\log n+q\log^2 n)$□空间复杂度 $O(n\log n)$□

```
#define lch(k) node[node[k].ch[0]]
#define rch(k) node[node[k].ch[1]]
const int MAXN=2e4+5,MAXM=40;
struct Node{
    int ch[2],pre,suf,sum;
}node[MAXN*MAXM];
int a[MAXN],b[MAXN],root[MAXN],node_cnt;
vector<int>c[MAXN];
void push_up(int k){
    node[k].pre=max(lch(k).pre,lch(k).sum+rch(k).pre);
    node[k].suf=max(rch(k).suf,rch(k).sum+lch(k).suf);
    node[k].sum=lch(k).sum+rch(k).sum;
}
```

```
void build(int &k,int lef,int rig){
    k=++node_cnt;
    if(lef==rig){
        node[k].sum=node[k].pre=node[k].suf=1;
        return;
    }
    int mid=lef+rig>>1;
    build(node[k].ch[0],lef,mid);
    build(node[k].ch[1],mid+1,rig);
    push_up(k);
}
void update(int &k,int p,int lef,int rig,int pos){
    k=++node_cnt;
    node[k]=node[p];
    if(lef==rig){
        node[k].sum=node[k].pre=node[k].suf=-1;
        return;
    }
    int mid=lef+rig>>1;
    if(mid>=pos)
    update(node[k].ch[0],node[p].ch[0],lef,mid,pos);
    else
    update(node[k].ch[1],node[p].ch[1],mid+1,rig,pos);
    push_up(k);
}
int query_sum(int k,int L,int R,int ql,int qr){
    if(ql>qr)return 0;
    if(ql<=L&&R<=qr)return node[k].sum;
    int M=L+R>>1;
    if(M>=qr)return query_sum(node[k].ch[0],L,M,ql,qr);
    else if(M<ql)return query_sum(node[k].ch[1],M+1,R,ql,qr);
    else return
query_sum(node[k].ch[0],L,M,ql,qr)+query_sum(node[k].ch[1],M+1,R,ql,qr);
}
pair<int,int> query_pre(int k,int L,int R,int ql,int qr){
    if(ql<=L&&R<=qr)return make_pair(node[k].pre,node[k].sum);
    int M=L+R>>1;
    if(M>=qr)return query_pre(node[k].ch[0],L,M,ql,qr);
    else if(M<ql)return query_pre(node[k].ch[1],M+1,R,ql,qr);
    else{
        pair<int,int>
l=query_pre(node[k].ch[0],L,M,ql,qr),r=query_pre(node[k].ch[1],M+1,R,ql,qr)
,s;
        s.first=max(l.first,l.second+r.first);
        s.second=l.second+r.second;
        return s;
    }
}
pair<int,int> query_suf(int k,int L,int R,int ql,int qr){
    if(ql<=L&&R<=qr)return make_pair(node[k].suf,node[k].sum);
    int M=L+R>>1;
```

```
    if(M>=qr)return query_suf(node[k].ch[0],L,M,ql,qr);
    else if(M<ql)return query_suf(node[k].ch[1],M+1,R,ql,qr);
    else{
        pair<int,int>
l=query_suf(node[k].ch[0],L,M,ql,qr),r=query_suf(node[k].ch[1],M+1,R,ql,qr)
,s;
        s.first=max(l.first+r.second,r.first);
        s.second=l.second+r.second;
        return s;
    }
}
int opt[4];
int main()
{
    int n=read_int();
    _for(i,0,n)a[i]=b[i]=read_int();
    sort(b,b+n);
    int m=unique(b,b+n)-b;
    _for(i,0,n){
        a[i]=lower_bound(b,b+m,a[i])-b;
        c[a[i]].push_back(i);
    }
    build(root[0],0,n-1);
    int pos,last=root[0];
    _for(i,1,n){
        _for(j,0,c[i-1].size()){
            update(pos,last,0,n-1,c[i-1][j]);
            last=pos;
        }
        root[i]=last;
    }
    last=0;
    int q=read_int();
    while(q--){
        _for(i,0,4)opt[i]=(read_int()+last)%n;
        sort(opt,opt+4);
        int lef=0,rig=m-1,mid,ans;
        while(lef<=rig){
            mid=lef+rig>>1;
if(query_suf(root[mid],0,n-1,opt[0],opt[1]).first+query_sum(root[mid],0,n-1
,opt[1]+1,opt[2]-1)+query_pre(root[mid],0,n-1,opt[2],opt[3]).first>=0){
                lef=mid+1;
                ans=mid;
            }
            else
            rig=mid-1;
        }
        enter(last=b[ans]);
    }
    return 0;
```

```
}
```

习题三

洛谷p4094

题意

给定一个长度为 n 的字符串 S 和 m 个询问。每次询问所有 $S[a,b]$ 的子串和 $S[c,d]$ 的 LCP 的最大值。

题解

发现所有 $S[a,b]$ 的子串和 $S[c,d]$ 的 LCP 的最大值等价于所有 $S[a,b]$ 的后缀和 $S[c,d]$ 的 LCP 的最大值。

考虑二分答案，对每个答案 x只需要存在 p 满足 $\text{LCP}(S[p,b],S[c,d])\ge x$ 即可。

而要满足上式必有 $\min(b-p+1,d-c+1)\ge x$于是上式等价于 $\text{LCP}(S[p,n],S[c,n])\ge x,p\le b-x+1$

考虑建立后缀数组，于是满足 $\text{LCP}(S[p,n],S[c,n])\ge x$ 的 p 必然是 sa 数组中的一段连续区间，可以通过二分得到该连续区间。

然后对 sa 数组建立主席树查询该连续区间是否存在点属于 $[a,b-x+1]$ 即可。

时间复杂度 $O(n\log n+q\log^2 n)$空间复杂度 $O(n\log n)$

```
const int MAXN=1e5+5,MAXM=30;
namespace SA{
    int sa[MAXN],rk[MAXN],height[MAXN],X[MAXN],Y[MAXN],c[MAXN];
    int d[MAXN][MAXM],lg2[MAXN];
    void get_sa(char *s,int n,int m){
        int *x=X,*y=Y;
        _rep(i,0,m)c[i]=0;
        _rep(i,1,n)c[x[i]=s[i]]++;
        _rep(i,1,m)c[i]+=c[i-1];
        for(int i=n;i;i--)sa[c[x[i]]--]=i;
        for(int k=1;k<n;k<<=1){
            int pos=0;
            _rep(i,n-k+1,n)y[++pos]=i;
            _rep(i,1,n)if(sa[i]>k)y[++pos]=sa[i]-k;
            _rep(i,0,m)c[i]=0;
            _rep(i,1,n)c[x[i]]++;
            _rep(i,1,m)c[i]+=c[i-1];
            for(int i=n;i;i--)sa[c[x[y[i]]]--]=y[i],y[i]=0;
            swap(x,y);
```

```
            pos=0,y[n+1]=0;
_rep(i,1,n)x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k])?pos:++
pos;
            if(pos==n)break;
            m=pos;
        }
        _rep(i,1,n)rk[sa[i]]=i;
    }
    void get_height(char *s,int n){
        for(int i=1,k=0;i<=n;i++){
            if(k)k--;
            while(s[i+k]==s[sa[rk[i]-1]+k])k++;
            height[rk[i]]=k;
        }
    }
    void build_st(int n){
        lg2[1]=0;
        _rep(i,2,n)
        lg2[i]=lg2[i>>1]+1;
        _rep(i,1,n)
        d[i][0]=height[i];
        for(int j=1;(1<<j)<=n;j++){
            _rep(i,1,n+1-(1<<j))
            d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);
        }
    }
    int lcp(int lef,int rig){
        lef++;
        int len=rig-lef+1;
        return min(d[lef][lg2[len]],d[rig-(1<<lg2[len])+1][lg2[len]]);
    }
}
struct Node{
    int ch[2],cnt;
}node[MAXN*MAXM];
int root[MAXN],node_cnt;
void update(int &k,int p,int lef,int rig,int pos){
    k=++node_cnt;
    node[k]=node[p];
    node[k].cnt++;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(mid>=pos)
    update(node[k].ch[0],node[p].ch[0],lef,mid,pos);
    else
    update(node[k].ch[1],node[p].ch[1],mid+1,rig,pos);
}
int query(int k1,int k2,int L,int R,int ql,int qr){
    if(ql<=L&&R<=qr)return node[k2].cnt-node[k1].cnt;
    int M=L+R>>1;
```

```
    if(M>=qr)return query(node[k1].ch[0],node[k2].ch[0],L,M,ql,qr);
    else if(M<ql)return query(node[k1].ch[1],node[k2].ch[1],M+1,R,ql,qr);
    else return
query(node[k1].ch[0],node[k2].ch[0],L,M,ql,qr)+query(node[k1].ch[1],node[k2
].ch[1],M+1,R,ql,qr);
}
bool check(int x,int l,int r,int pos,int n){
    int ql=SA::rk[pos],qr=SA::rk[pos],lef,rig,mid;
    lef=1,rig=SA::rk[pos]-1;
    while(lef<=rig){
        mid=lef+rig>>1;
        if(SA::lcp(mid,SA::rk[pos])>=x){
            rig=mid-1;
            ql=mid;
        }
        else
        lef=mid+1;
    }
    lef=SA::rk[pos]+1,rig=n;
    while(lef<=rig){
        mid=lef+rig>>1;
        if(SA::lcp(SA::rk[pos],mid)>=x){
            lef=mid+1;
            qr=mid;
        }
        else
        rig=mid-1;
    }
    return query(root[ql-1],root[qr],1,n,l,r);
}
char buf[MAXN];
int main()
{
    int n=read_int(),q=read_int();
    scanf("%s",buf+1);
    SA::get_sa(buf,n,'z');
    SA::get_height(buf,n);
    SA::build_st(n);
    _rep(i,1,n)
    update(root[i],root[i-1],1,n,SA::sa[i]);
    while(q--){
        int l1=read_int(),r1=read_int(),l2=read_int(),r2=read_int();
        int lef=0,rig=min(r1-l1+1,r2-l2+1),mid,ans;
        while(lef<=rig){
            mid=lef+rig>>1;
            if(check(mid,l1,r1-mid+1,l2,n)){
                lef=mid+1;
                ans=mid;
            }
            else
            rig=mid-1;
```

```
        }
        enter(ans);
    }
    return 0;
}
```

习题四

洛谷p3899

题意

给定一棵以 1 为根的树，接下来 q 次询问。每次询问给定 a,k有多少个二元组 (b,c) 满足：

1. 结点 a,b,c 互异
2. 结点 a 和结点 b 是结点 c 的祖先
3. 结点 a 和结点 b 的距离不超过 k

题解

对每次查询操作，答案分为 b 为 a 祖先和 a 为 b 祖先两种。对于 b 为 a 祖先的情况，易知答案为 $(\text{sz}_a-1)\min(d_a-1,k)$

对于 a 为 b 祖先的情况，易知每个 a 的子树中距离 a 不超过 k 的结点的贡献为该结点的子树大小减一。

于是考虑将每个结点的权值设为该结点的子树大小减一，于是贡献和为所有深度为 $[d_a+1,d_a+k]$ 且 dfs 序属于 a 的子树的结点的权值和。

于是问题转化为二维偏序问题，考虑主席树统计答案。时间复杂度 $O((n+q)\log n)$

```
const int MAXN=3e5+5,MAXM=40;
struct Node{
    LL s;
    int ch[2];
}node[MAXN*MAXM];
int root[MAXN],n,node_cnt;
void update(int &k,int p,int pos,int v,int lef=1,int rig=n){
    node[k=++node_cnt]=node[p];
    node[k].s+=v;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(pos<=mid)
    update(node[k].ch[0],node[p].ch[0],pos,v,lef,mid);
    else
    update(node[k].ch[1],node[p].ch[1],pos,v,mid+1,rig);
}
```

```
LL query(int k1,int k2,int ql,int qr,int lef=1,int rig=n){
    if(ql<=lef&&rig<=qr)return node[k1].s-node[k2].s;
    int mid=lef+rig>>1;
    if(mid>=qr)return query(node[k1].ch[0],node[k2].ch[0],ql,qr,lef,mid);
    else if(mid<ql)return
query(node[k1].ch[1],node[k2].ch[1],ql,qr,mid+1,rig);
    else return
query(node[k1].ch[0],node[k2].ch[0],ql,qr,lef,mid)+query(node[k1].ch[1],nod
e[k2].ch[1],ql,qr,mid+1,rig);
}
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int dfs_t,dfn[MAXN],sz[MAXN],d[MAXN],inv_dfn[MAXN];
void dfs(int u,int fa,int dep){
    dfn[u]=++dfs_t;
    sz[dfs_t]=1;d[dfs_t]=dep;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs(v,u,dep+1);
        sz[dfn[u]]+=sz[dfn[v]];
    }
}
int main()
{
    n=read_int();
    int q=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    dfs(1,0,1);
    _rep(i,1,n)
    update(root[i],root[i-1],d[i],sz[i]-1);
    while(q--){
        int p=read_int(),k=read_int(),u=dfn[p];
enter(1LL*(sz[u]-1)*min(d[u]-1,k)+query(root[u+sz[u]-1],root[u],d[u],min(d[
u]+k,n)));
    }
    return 0;
}
```

习题五

洛谷p4197

题意

给定 n 个点和 m 条边，每个点给定一个点权，每条边给定一个边权。接下来 q 个询问。

每次询问从点 v 开始不经过边权超过 x 的边可以走到的点中第 k 大的权值。

题解

考虑 Kruskal 算法重构树，即求取最小生成树过程中将加边操作改为生成一个新结点同时将新结点作为两个连通块的根节点的祖先。

将重构树上的新结点的点权设为该结点代表边的边权，原图中结点点权设为 0。于是重构树上的点权从叶子结点到根节点单增。

于是每次询问从原图某结点开始可以到达的边权不超过 x 的结点等价于询问该结点的权值不超过 x 的最远的祖先结点的子树。

于是倍增求出满足条件的祖先结点，最后利用 dfs 序和主席树处理第 k 大询问即可。时间复杂度 $O((n+q)\log n+m\log m)$

另外这题可以以离线询问，将询问和边然后按权值排序，然后通过线段树合并处理加边操作和询问操作。

```
#define rch(k) node[node[k].ch[1]]
const int MAXN=1e5+5,MAXM=5e5+5,MAXL=80;
struct Node{
    int s,ch[2];
}node[MAXN*MAXL];
int root[MAXN<<1],n,seg_n,node_cnt;
void update(int &k,int p,int pos,int lef=1,int rig=seg_n){
    node[k=++node_cnt]=node[p];
    node[k].s++;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(pos<=mid)
    update(node[k].ch[0],node[p].ch[0],pos,lef,mid);
    else
    update(node[k].ch[1],node[p].ch[1],pos,mid+1,rig);
}
int query(int k1,int k2,int rk,int lef=1,int rig=seg_n){
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(rk<=rch(k1).s-rch(k2).s)
    return query(node[k1].ch[1],node[k2].ch[1],rk,mid+1,rig);
    else
    return query(node[k1].ch[0],node[k2].ch[0],rk-
rch(k1).s+rch(k2).s,lef,mid);
```

```
}
struct Edge{
    int u,v,w;
    bool operator < (const Edge &b)const{
        return w<b.w;
    }
}edge[MAXM];
int a[MAXN],b[MAXN],p[MAXN<<1],ch[MAXN<<1][2],dfn[MAXN<<1][2],dfs_t;
namespace LCA{
    const int MAXN=2e5+5;
    int d[MAXN],anc[MAXN][MAXL],w[MAXN],log2[MAXN];
    void get_log(int n){
        int log=0;
        while((1<<log)<n){
            for(int i=1<<log,j=min(1<<(log+1),n);i<j;i++)
            log2[i]=log;
            log++;
        }
    }
    void init(int n){
        get_log(n);
        for(int j=1;(1<<j)<n;j++){
            _rep(i,1,n){
                if(!anc[i][j-1])
                continue;
                anc[i][j]=anc[anc[i][j-1]][j-1];
            }
        }
    }
    int query(int p,int k){
        for(int i=log2[d[p]];i>=0;i--){
            if(anc[p][i]&&w[anc[p][i]]<=k)
            p=anc[p][i];
        }
        return p;
    }
};
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
void dfs(int u,int fa,int dep){
    if(!u)return;
    dfn[u][0]=++dfs_t;LCA::d[u]=dep;
    if(u<=n)
    update(root[dfs_t],root[dfs_t-1],a[u]);
    else
    root[dfs_t]=root[dfs_t-1];
    dfs(ch[u][0],u,dep+1);
    dfs(ch[u][1],u,dep+1);
    dfn[u][1]=dfs_t;
}
int main()
{
```

```
    n=read_int();
    int m=read_int(),q=read_int();
    _rep(i,1,n)a[i]=b[i]=read_int();
    sort(b+1,b+n+1);
    seg_n=unique(b+1,b+n+1)-b;
    _rep(i,1,n)a[i]=lower_bound(b+1,b+seg_n,a[i])-b;
    _for(i,0,m){
        edge[i].u=read_int();
        edge[i].v=read_int();
        edge[i].w=read_int();
    }
    sort(edge,edge+m);
    int cur=n+1;
    _for(i,1,2*n)p[i]=i;
    _for(i,0,m){
        int x=Find(edge[i].u),y=Find(edge[i].v);
        if(x!=y){
            p[x]=p[y]=cur;
            ch[cur][0]=x,ch[cur][1]=y;
            LCA::anc[x][0]=LCA::anc[y][0]=cur,LCA::w[cur]=edge[i].w;
            cur++;
        }
    }
    _for(i,1,cur){
        if(!root[Find(i)])
        dfs(Find(i),0,0);
    }
    LCA::init(cur-1);
    while(q--){
        int v=read_int(),x=read_int(),k=read_int();
        int rt=LCA::query(v,x);
        if(node[root[dfn[rt][1]]].s-node[root[dfn[rt][0]-1]].s<k)
        enter(-1);
        else
        enter(b[query(root[dfn[rt][1]],root[dfn[rt][0]-1],k)]);
    }
    return 0;
}
```