

数据结构

树状数组

区间和

```
struct BIT{
    vector<int> tree;
    int n;
    inline int lowbit(int x){
        return x&(-x);
    }
    BIT(int n){
        this->n=n;
        tree.resize(n+1,0);
    }
    void update(int x,int delta){
        for(int i=x;i<=n;i+=lowbit(i)) tree[i]+=delta;
    }
    int query(int x,int y){
        if(x>y) return 0;
        int ans=0;
        for(int i=y;i>=1;i-=lowbit(i)) ans+=tree[i];
        for(int i=x-1;i>=1;i-=lowbit(i)) ans-=tree[i];
        return ans;
    }
};
```

区间最值

```
struct BIT{
    vector<int> tree;
    vector<int> raw;
    int n;
    inline int lowbit(int x){
        return x&(-x);
    }
    BIT(int n){
        this->n=n;
        tree.resize(n+1,0);
        raw.resize(n+1,0);
    }
    void update(int x,int y){
        raw[x]=y;
        for(int i=x;i<=n;i+=lowbit(i)){
            tree[i]=raw[i];
            for(int j=1;j<lowbit(i);j<<=1){
                tree[i]=max(tree[i],tree[i-j]);
            }
        }
    }
    int query(int x,int y){
        if(x>y) return 0;
```

```

int ans=0;
while(x<=y){
    int nx=y-lowbit(y)+1;
    if(nx>=x){
        ans=max(ans,tree[y]);
        y=nx-1;
    }else{
        ans=max(ans,raw[y]);
        --y;
    }
}
return ans;
};

};

```

线段树

```

struct SegmentTree{
    struct edge{
        int sum;
        edge() {
            sum=0;
        }
    };
    vector<int> lazy;
    vector<edge> node;
    int n;
    void pushup(int id,int l,int r){
        node[id].sum=node[id<<1].sum+node[id<<1|1].sum;
    }
    void pushdown(int id,int l,int r){
        if(lazy[id]){
            int mid=l+(r-l>>1);
            lazy[id<<1]+=lazy[id];
            lazy[id<<1|1]+=lazy[id];
            node[id<<1].sum+=(mid-l+1)*lazy[id];
            node[id<<1|1].sum+=(r-mid)*lazy[id];
            lazy[id]=0;
        }
    }
    SegmentTree(int n):n(n){
        node.resize((n<<2)+5);
        lazy.assign((n<<2)+5,0);
    }
    SegmentTree(){}
    void init(vector<int> &v){
        function<void(int,int,int)> buildtree=[&](int id,int l,int r){
            lazy[id]=0;
            if(l==r){
                node[id].sum=v[l];
                return;
            }
            int mid=l+(r-l>>1);
            buildtree(id<<1,l,mid);
            buildtree(id<<1|1,mid+1,r);
        };
        buildtree(0,0,n);
    }
};

```

```

        pushup(id,l,r);
    };
    buildtree(1,1,n);
}

SegmentTree(int n,vector<int> &v):n(n){
    node.resize((n<<2)+5);
    lazy.assign((n<<2+5),0);
    init(v);
}

void update(int id,int l,int r,int x,int y,int delta){
    if(x<=l&&r<=y){
        lazy[id]+=delta;
        node[id].sum+=delta*(r-l+1);
        return;
    }
    pushdown(id,l,r);
    int mid=l+(r-l>>1);
    if(x<=mid) update(id<<1,l,mid,x,y,delta);
    if(y>mid) update(id<<1|1,mid+1,r,x,y,delta);
    pushup(id,l,r);
}

int query(int id,int l,int r,int x,int y){
    if(x<=l&&r<=y) return node[id].sum;
    pushdown(id,l,r);
    int mid=l+(r-l>>1);
    int ans=0;
    if(x<=mid) ans+=query(id<<1,l,mid,x,y);
    if(y>mid) ans+=query(id<<1|1,mid+1,r,x,y);
    return ans;
}
};


```

线段树标记永久化

```

struct SegmentTree{
    struct edge{
        int sum,lazy;
        edge():sum=lazy=0{};
    };
    int n;
    vector<edge> node;
    SegmentTree(int n):n(n){
        node.resize(n*4+5);
    }

    void update(int x,int y,int delta){
        auto upd=[&](auto self,int id,int l,int r,int x,int y,int delta){
            node[id].sum+=delta*(min(y,r)-max(l,x)+1);
            if(x<=l&&r<=y){
                node[id].lazy+=delta;
                return;
            }
            int mid=l+(r-l>>1);
            if(x<=mid) self(self,id<<1,l,mid,x,y,delta);

```

```

        if(y>mid) self(self,id<<1|1,mid+1,r,x,y,delta);
    };
    upd(upd,1,1,n,x,y,delta);
}
int query(int x,int y){
    auto que=[&](auto self,int id,int l,int r,int x,int y,int lz){
        if(x<=l&&r<=y){
            return node[id].sum+lz*(r-l+1);
        }
        int ans=0;
        int mid=l+(r-l>>1);
        lz+=node[id].lazy;
        if(x<=mid){
            ans+=self(self,id<<1,l,mid,x,y,lz);
        }
        if(y>mid){
            ans+=self(self,id<<1|1,mid+1,r,x,y,lz);
        }
        return ans;
    };
    return que(que,1,1,n,x,y,0);
}
};

```

线段树合并

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
struct SegmentTree{
    struct edge{
        int maxn,maxnpos,lson,rson;
        edge():maxn(0),maxnpos(0),lson(0),rson(0){}
    };
    int n;
    vector<edge> node;
    SegmentTree(int n):n(n),node(1){}
    void pushup(int id,int l,int r){
        node[id].maxn=0;
        node[id].maxnpos=0;
        if(node[id].lson){
            if(node[id].maxn<node[node[id].lson].maxn){
                node[id].maxn=node[node[id].lson].maxn;
                node[id].maxnpos=node[node[id].lson].maxnpos;
            }
        }
        if(node[id].rson){
            if(node[id].maxn<node[node[id].rson].maxn){
                node[id].maxn=node[node[id].rson].maxn;
                node[id].maxnpos=node[node[id].rson].maxnpos;
            }
        }
    }
    int update(int id,int l,int r,int x,int delta){
        if(id==0){

```

```

        id=node.size();
        node.emplace_back();
    }
    if(l==r){
        node[id].maxn+=delta;
        node[id].maxnpos=l;
        return id;
    }
    int mid=l+(r-l>>1);
    if(x<=mid) node[id].lson=update(node[id].lson,l,mid,x,delta);
    else node[id].rson=update(node[id].rson,mid+1,r,x,delta);
    pushup(id,l,r);
    return id;
}
int merge(int a,int b,int l,int r){
    if(!a&&!b) return 0;
    if(!a) return b;
    if(!b) return a;
    if(l==r){
        node[a].maxn+=node[b].maxn;
        return a;
    }
    int mid=l+(r-l>>1);
    node[a].lson=merge(node[a].lson,node[b].lson,l,mid);
    node[a].rson=merge(node[a].rson,node[b].rson,mid+1,r);
    pushup(a,l,r);
    return a;
}
int query(int id){
    return node[id].maxnpos;
}
};

void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(n+1);
    vector<int> dep(n+1);
    vector<vector<int>> fa(n+1,vector<int>(25));
    vector<map<int,int>> mp(n+1);
    for(int i=1;i<n;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    function<void(int,int)> dfs=[&](int x,int father){
        dep[x]=dep[father]+1;
        fa[x][0]=father;
        for(int i=1;i<=20;i++){
            fa[x][i]=fa[fa[x][i-1]][i-1];
        }
        for(int &to:v[x]){
            if(to==father) continue;
            dfs(to,x);
        }
    };
}

```

```

dfs(1,0);
auto LCA=[&](int x,int y){
    if(dep[x]<dep[y]) swap(x,y);
    for(int i=20;i>=0;i--){
        if(dep[fa[x][i]]>=dep[y]){
            x=fa[x][i];
        }
    }
    if(x==y) return x;
    for(int i=20;i>=0;i--){
        if(fa[x][i]!=fa[y][i]){
            x=fa[x][i];
            y=fa[y][i];
        }
    }
    return fa[x][0];
};
for(int i=1;i<=m;i++){
    int x,y,z;
    cin>>x>>y>>z;
    mp[x][z]++;
    mp[y][z]++;
    int l=LCA(x,y);
    mp[l][z]--;
    mp[fa[l][0]][z]--;
}
vector<int> head(n+1),ans(n+1);
SegmentTree tree(1e5);
function<void(int,int)> dfs1=[&](int x,int father){
    for(auto &to:v[x]){
        if(to==father) continue;
        dfs1(to,x);
        head[x]=tree.merge(head[x],head[to],1,1e5);
    }
    for(auto &[p,q]:mp[x]){
        head[x]=tree.update(head[x],1,1e5,p,q);
    }
    ans[x]=tree.query(head[x]);
};
dfs1(1,0);
for(int i=1;i<=n;i++) cout<<ans[i]<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

并查集

```

//带权并查集
//sz表示祖先节点个数
struct DSU{

```

```

vector<int> fa;
vector<int> sz;
int n;
DSU(int n){
    this->n=n;
    fa.resize(n+1);
    sz.resize(n+1,0);
    for(int i=0;i<=n;i++) fa[i]=i;
}
int find(int x){
    if(fa[x]==x) return x;
    int fax=fa[x];
    fa[x]=find(fa[x]);
    sz[x]+=sz[fax];
    return fa[x];
}
bool merge(int x,int y){
    int fax=find(x);
    int fay=find(y);
    if(fax==fay) return 0;
    fa[x]=y;
    sz[x]=1;
    return 1;
}
};


```

```

//按秩合并+路径压缩，rank表示子树深度
struct DSU{
    vector<int> fa;
    vector<int> rank;
    int n;
    DSU(int n){
        this->n=n;
        fa=vector<int>(n+1);
        rank=vector<int>(n+1,0);
        for(int i=0;i<=n;i++) fa[i]=i;
    }
    int find(int x){
        return fa[x]==x?x:fa[x]=find(fa[x]);
    }
    bool merge(int x,int y){
        int fax=find(x);
        int fay=find(y);
        if(fax==fay) return 0;
        if(rank[fas]<rank[fay]) fa[fas]=fay;
        else{
            fa[fay]=fax;
            if(rank[fas]==rank[fay]) rank[fas]++;
        }
        return 1;
    }
};


```

可持久化线段树

- 开40倍空间

```
struct PresidentTree{
    vector<int> node;
    vector<int> lson, rson;
    vector<int> head;
    int n;
    int cnt;
    PresidentTree(int n, vector<int> &v) : n(n){
        node.resize(40*n);
        lson.resize(40*n);
        rson.resize(40*n);
        cnt=0;
        function<int(int,int)> buildtree=[&](int l, int r){
            int now=++cnt;
            if(l==r){
                node[now]=v[l];
                return now;
            }
            int mid=l+(r-l>>1);
            lson[now]=buildtree(l, mid);
            rson[now]=buildtree(mid+1, r);
            return now;
        };
        head.push_back(buildtree(1, n));
    }
    void update(int nowid, int baseid, int x, int y){
        function<int(int,int,int,int,int)> updatenode=[&](int base, int l, int r, int x, int y){
            int now=++cnt;
            if(l==r){
                node[now]=y;
                return now;
            }
            int mid=l+(r-l>>1);
            if(x<=mid){
                lson[now]=updatenode(lson[base], l, mid, x, y);
                rson[now]=rson[base];
            }else{
                lson[now]=lson[base];
                rson[now]=updatenode(rson[base], mid+1, r, x, y);
            }
            return now;
        };
        head.push_back(updatenode(head[baseid], 1, n, x, y));
    }
    int query(int id, int x){
        function<int(int,int,int,int)> querynode=[&](int base, int l, int r, int x){
            if(l==r) return node[base];
            int mid=l+(r-l>>1);
            if(x<=mid) return querynode(lson[base], l, mid, x);
            else return querynode(rson[base], mid+1, r, x);
        };
    }
}
```

```

        return querynode(head[id], 1, n, x);
    }
};

```

可持久化01trie tree

```

struct TRIE{
    vector<array<int,2>> v;
    vector<int> head;
    vector<int> num;
    TRIE(){
        v.push_back({-1, -1});
        head.push_back(0);
        num.push_back(0);
        int now=0;
        for(int i=30; i>=0; i--){
            v[now][0]=v.size();
            num.push_back(1);
            v.push_back({-1, -1});
            now=v[now][0];
        }
    }
    void insert(int x){
        int base=head.back(), now=v.size();
        head.push_back(v.size());
        v.push_back({-1, -1});
        num.push_back(0);
        for(int i=30; i>=0; i--){
            int cnt=x>>i&1;
            if(base===-1) v[now][!cnt]==-1;
            else v[now][!cnt]=v[base][!cnt];
            v[now][cnt]=v.size();
            v.push_back({0, 0});
            if(base===-1 || v[base][cnt]==-1){
                num.push_back(1);
            } else{
                num.push_back(num[v[base][cnt]]+1);
            }
            now=v[now][cnt];
            if(base!= -1) base=v[base][cnt];
        }
    }
    int query(int l, int r, int x){
        int lnow, rnow;
        if(l-1<0) lnow=-1;
        else lnow=head[l-1];
        rnow=head[r];
        int ans=0;
        for(int i=30; i>=0; i--){
            int cnt=x>>i&1;
            if(v[rnow][!cnt] != -1 && num[v[rnow][!cnt]] && (lnow== -1 || v[lnow][!cnt]==-1 || num[v[rnow][!cnt]]>num[v[lnow][!cnt]])){
                ans|=1<<i;
                if(lnow!= -1) lnow=v[lnow][!cnt];
                rnow=v[rnow][!cnt];
            }
        }
    }
};

```

```

        }else{
            if(lnow!=-1) lnow=v[lnow][cnt];
            rnow=v[rnow][cnt];
        }
    }
    return ans;
}
};

```

ST 表

```

struct ST{
    static vector<int> Log2;
    vector<vector<int>> dp;
    ST(int n,vector<int> &v){
        for(int i=Log2.size();i<=n;i++){
            if(i==0) Log2.push_back(0);
            else if(i==1) Log2.push_back(0);
            else Log2.push_back(Log2[i>>1]+1);
        }
        dp.resize(n+1);
        for(int i=1;i<=n;i++){
            dp[i].resize(20);
            dp[i][0]=v[i];
        }
        for(int i=1;i<=18;i++){
            for(int j=1;j+(1<<i)-1<=n;j++){
                dp[j][i]=max(dp[j][i-1],dp[j+(1<<i-1)][i-1]);
            }
        }
    }
    int query(int l,int r){
        int k=Log2[r-l+1];
        return max(dp[l][k],dp[r-(1<<k)+1][k]);
    }
};
vector<int> ST::Log2;

```

权值线段树

```

struct SegmentTree{
    struct edge{
        int sum,lson,rson;
    };
    int l,r;
    int cnt;
    vector<edge> node;
    SegmentTree(int l,int r):l(l),r(r){
        cnt=1;
        node.push_back({0,0,0});
        node.push_back({0,0,0});
    }
    void pushup(int id,int l,int r){
        node[id].sum=0;
    }
    void update(int id,int l,int r,int val,int pos,int k=1,int d=0){
        if(l==r){
            node[id].sum+=val;
            return;
        }
        if(d==0)
            k=Log2[r-l+1];
        if(k>=d)
            update(id*2,l,(l+r)/2,val,pos,d-1);
        else
            update(id*2+1,(l+r)/2+1,r,val,pos,d-1);
        node[id].sum=node[id*2].sum+node[id*2+1].sum;
    }
    int query(int id,int l,int r,int pos,int k=1,int d=0){
        if(l==r)
            return node[id].sum;
        if(d==0)
            k=Log2[r-l+1];
        if(k>=d)
            return query(id*2,l,(l+r)/2,pos,d-1);
        else
            return query(id*2+1,(l+r)/2+1,r,pos,d-1);
        return node[id].sum;
    }
};

```

```

        if(node[id].lson) node[id].sum+=node[node[id].lson].sum;
        if(node[id].rson) node[id].sum+=node[node[id].rson].sum;
    }

    void update(int x,int delta){
        function<void(int,int,int,int,int)> upd=[&](int id,int l,int r,int x,int delta){
            if(l==r){
                node[id].sum+=delta;
                return;
            }
            int mid=l+(r-l>>1);
            if(x<=mid){
                if(!node[id].lson){
                    node[id].lson=++cnt;
                    node.push_back({0,0,0});
                }
                upd(node[id].lson,l,mid,x,delta);
            }else{
                if(!node[id].rson){
                    node[id].rson=++cnt;
                    node.push_back({0,0,0});
                }
                upd(node[id].rson,mid+1,r,x,delta);
            }
            pushup(id,l,r);
        };
        upd(1,l,r,x,delta);
    }

    int query(int x,int y){
        function<int(int,int,int,int,int)> que=[&](int id,int l,int r,int x,int y){
            if(x<=l&&r<=y) return node[id].sum;
            int ans=0;
            int mid=l+(r-l>>1);
            if(x<=mid&&node[id].lson){
                ans+=que(node[id].lson,l,mid,x,y);
            }
            if(y>mid&&node[id].rson){
                ans+=que(node[id].rson,mid+1,r,x,y);
            }
            return ans;
        };
        return que(1,l,r,x,y);
    }
}

```

树套树

树状数组套权值线段树

区间kth等的问题，用树状数组来实现权值线段树的区间求和

```

#include<bits/stdc++.h>
using namespace std;
#define int long long

```

```

struct SegmentTree{
    struct edge{
        int sum,lson,rson;
    };
    int l,r;
    int cnt;
    vector<edge> node;
    SegmentTree(int l,int r):l(l),r(r){
        cnt=1;
        node.push_back({0,0,0});
        node.push_back({0,0,0});
    }
    void pushup(int id,int l,int r){
        node[id].sum=0;
        if(node[id].lson) node[id].sum+=node[node[id].lson].sum;
        if(node[id].rson) node[id].sum+=node[node[id].rson].sum;
    }
    void update(int x,int delta){
        function<void(int,int,int,int,int)> upd=[&](int id,int l,int r,int x,int delta){
            //cerr<<l<<" "<<r<<"\n";
            if(l==r){
                node[id].sum+=delta;
                return;
            }
            int mid=l+(r-l>>1);
            if(x<=mid){
                if(!node[id].lson){
                    node[id].lson=++cnt;
                    node.push_back({0,0,0});
                }
                upd(node[id].lson,l,mid,x,delta);
            }else{
                if(!node[id].rson){
                    node[id].rson=++cnt;
                    node.push_back({0,0,0});
                }
                upd(node[id].rson,mid+1,r,x,delta);
            }
            pushup(id,l,r);
        };
        upd(1,l,r,x,delta);
    }
    int query(int x,int y){
        function<int(int,int,int,int,int)> que=[&](int id,int l,int r,int x,int y){
            if(x<=l&&r<=y) return node[id].sum;
            int ans=0;
            int mid=l+(r-l>>1);
            if(x<=mid&&node[id].lson){
                ans+=que(node[id].lson,l,mid,x,y);
            }
            if(y>mid&&node[id].rson){
                ans+=que(node[id].rson,mid+1,r,x,y);
            }
            return ans;
        }
    }
}

```

```

    };
    return que(1, l, r, x, y);
}
};

struct BIT{
    vector<SegmentTree> tree;
    int n;
    inline int lowbit(int x){
        return x&(-x);
    }
    BIT(int n,int l,int r){
        this->n=n;
        tree.resize(n+1,SegmentTree(l,r));
    }
    void update(int x,int y,int delta){
        for(int i=x;i<=n;i+=lowbit(i)){
            tree[i].update(y,delta);
        }
    }
    int query(int x,int y,int a,int b){
        if(x>y) return 0;
        int ans=0;
        for(int i=y;i>=1;i-=lowbit(i)){
            ans+=tree[i].query(a,b);
        }
        for(int i=x-1;i>=1;i-=lowbit(i)){
            ans-=tree[i].query(a,b);
        }
        return ans;
    }
    int que(int l,int r,int x,int y,int k){
        if(x==y) return x;
        int mid=x+(y-x>>1);
        int num=query(l,r,x,mid);
        if(k<=num) return que(l,r,x,mid,k);
        else return que(l,r,mid+1,y,k-num);
    }
};
struct ss{
    char c;
    int x,y,z;
};

void solve(){
    int n,m;
    cin>>n>>m;
    vector<int> a(n+1);
    vector<int> num;
    vector<ss> query(m);
    function<int(int)> getid=[&](int x){
        return lower_bound(num.begin(),num.end(),x)-num.begin()+1;
    };
    for(int i=1;i<=n;i++){
        cin>>a[i];
        num.push_back(a[i]);
    }
    for(int i=0;i<m;i++){

```

```

    cin>>query[i].c;
    if(query[i].c=='Q'){
        cin>>query[i].x>>query[i].y>>query[i].z;
    }else{
        cin>>query[i].x>>query[i].y;
        num.push_back(query[i].y);
    }
}
sort(num.begin(),num.end());
num.erase(unique(num.begin(),num.end()),num.end());
BIT bit(n,1,num.size());
for(int i=1;i<=n;i++) a[i]=getid(a[i]);
for(int i=0;i<m;i++){
    if(query[i].c=='C'){
        query[i].y=getid(query[i].y);
    }
}
for(int i=1;i<=n;i++) bit.update(i,a[i],1);
for(int i=0;i<m;i++){
    if(query[i].c=='Q'){
        cout<<num[bit.que(query[i].x,query[i].y,1,num.size(),query[i].z)-1]
<<"\n";
    }else{
        int x=query[i].x;
        int y=query[i].y;
        bit.update(x,a[x],-1);
        bit.update(x,y,1);
        a[x]=y;
    }
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

线段树套平衡树

区间排名，单点修改，找区间前驱后继

```

#include<bits/stdc++.h>
#include<bits/extc++.h>
using namespace std;
#define int long long
const int INF=1e18;
struct SegmentTree{
    struct edge{
        __gnu_pbds::tree<std::pair<int,int>, __gnu_pbds::null_type,
                        std::less<pair<int,int>>, __gnu_pbds::rb_tree_tag,
                        __gnu_pbds::tree_order_statistics_node_update>
        tree;
    };

```

```

vector<edge> node;
int n;
int clock;
SegmentTree(int n):n(n){
    node.resize((n<<2)+5);
    clock=0;
}
SegmentTree(){}
void init(vector<pair<int,int>> &v){
    function<void(int,int,int)> buildtree=[&](int id,int l,int r){
        for(int i=l;i<=r;i++){
            node[id].tree.insert(v[i]);
        }
        if(l==r) return;
        int mid=l+(r-l>>1);
        buildtree(id<<1,l,mid);
        buildtree(id<<1|1,mid+1,r);
    };
    buildtree(1,1,n);
}
SegmentTree(int n,vector<pair<int,int>> &v):n(n){
    node.resize((n<<2)+5);
    init(v);
}
void update(int w,pair<int,int> x,pair<int,int> y){
    function<void(int,int,int,int,pair<int,int>,pair<int,int>)> upd=[&](int
id,int l,int r,int w,pair<int,int> x,pair<int,int> y){
        node[id].tree.erase(x);
        node[id].tree.insert(y);
        if(l==r){
            return;
        }
        int mid=l+(r-l>>1);
        if(w<=mid) upd(id<<1,l,mid,w,x,y);
        else upd(id<<1|1,mid+1,r,w,x,y);
    };
    upd(1,1,n,w,x,y);
}
int querynum(int id,int l,int r,int x,int y,int t){
    if(x<=l&&r<=y) return (int)node[id].tree.order_of_key({t,0});
    int ans=0;
    int mid=l+(r-l>>1);
    if(x<=mid){
        ans+=querynum(id<<1,l,mid,x,y,t);
    }
    if(y>mid){
        ans+=querynum(id<<1|1,mid+1,r,x,y,t);
    }
    return ans;
};
int queryrnk(int l,int r,int x){
    return querynum(1,1,n,l,r,x)+1;
}
int querybyrnk(int x,int y,int k){
    int l=0,r=1e8,ans=-1;
    while(l<=r){

```

```

        int mid=l+(r-l>>1);
        int num=querynum(1,1,n,x,y,mid);
        if(num<=k-1){
            ans=mid;
            l=mid+1;
        }else r=mid-1;
    }
    return ans;
}
int querypre(int x,int y,int k){
    function<int(int,int,int,int,int,int)> query=[&](int id,int l,int r,int
x,int y,int t){
        if(x<=l&&r<=y){
            auto it=node[id].tree.lower_bound({t,0});
            if(it==node[id].tree.begin()) return -214748364711;
            --it;
            return it->first;
        }
        int mid=l+(r-l>>1);
        int ans=-214748364711;
        if(x<=mid) ans=max(ans,query(id<<1,l,mid,x,y,t));
        if(y>mid) ans=max(ans,query(id<<1|1,mid+1,r,x,y,t));
        return ans;
    };
    return query(1,1,n,x,y,k);
}
int querynxt(int x,int y,int k){
    function<int(int,int,int,int,int,int)> query=[&](int id,int l,int r,int
x,int y,int t){
        if(x<=l&&r<=y){
            auto it=node[id].tree.upper_bound({t,INF});
            if(it==node[id].tree.end()) return 214748364711;
            return it->first;
        }
        int mid=l+(r-l>>1);
        int ans=214748364711;
        if(x<=mid) ans=min(ans,query(id<<1,l,mid,x,y,t));
        if(y>mid) ans=min(ans,query(id<<1|1,mid+1,r,x,y,t));
        return ans;
    };
    return query(1,1,n,x,y,k);
}
void solve(){
    int n,m;
    cin>>n>>m;
    vector<pair<int,int>> a(n+1);
    int clock=0;
    for(int i=1;i<=n;i++){
        cin>>a[i].first;
        a[i].second=++clock;
    }
    SegmentTree tree(n,a);
    while(m--){
        int op;
        cin>>op;

```

```

if(op==1){
    int l,r,k;
    cin>>l>>r>>k;
    cout<<tree.queryrank(l,r,k)<<"\n";
}else if(op==2){
    int l,r,k;
    cin>>l>>r>>k;
    cout<<tree.querybyrank(l,r,k)<<"\n";
}else if(op==3){
    int pos,k;
    cin>>pos>>k;
    tree.update(pos,a[pos],{k,++clock});
    a[pos]={k,clock};
}else if(op==4){
    int l,r,k;
    cin>>l>>r>>k;
    cout<<tree.querypre(l,r,k)<<"\n";
}else{
    int l,r,k;
    cin>>l>>r>>k;
    cout<<tree.querynext(l,r,k)<<"\n";
}
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

猫树

$O(n \log n)$ 次合并，查询时 $O(1)$ 次合并查询

具体做法是，每个节点， l, mid, r ,存储mid往前的后缀贡献，和mid往后的前缀贡献

查询区间 $[l, r]$ ，则只需要在他们的lca上查找前缀后缀即可 $O(1)$ 查询

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
struct CatTree{
    int n;
    vector<int> pos; // [i, i]所在的位置
    vector<pair<int,int>> lr;
    vector<array<vector<int>,2>> maxn; // 某个区间，从mid分开的后缀前缀贡献
    int lca(int x,int y){
        int xx=__builtin_clz(x);
        int yy=__builtin_clz(y);
        if(xx<yy) x>>=yy-xx;
        else y>>=xx-yy;
        return x>>(__lg(x^y)+1);
    }
}

```

```

CatTree(int n,vector<int> &v):n(n),pos(n+1,0),maxn((n<<2)+10),lr((n<<2)+10){
    auto buildtree=[&](auto &&self,int id,int l,int r){
        int mid=l+(r-l>>1);
        maxn[id][0].reserve(mid-1+2);
        maxn[id][1].reserve(r-mid+1);
        maxn[id][0].push_back(-1e18);
        maxn[id][1].push_back(-1e18);
        lr[id]={l,r};
        if(l==r){
            pos[l]=id;
            maxn[id][0].push_back(v[l]);
            return;
        }
        for(int i=mid;i>=l;i--){
            maxn[id][0].push_back(max(maxn[id][0].back(),v[i]));
        }
        for(int i=mid+1;i<=r;i++){
            maxn[id][1].push_back(max(maxn[id][1].back(),v[i]));
        }
        self(self,id<<1,l,mid);
        self(self,id<<1|1,mid+1,r);
    };
    buildtree(buildtree,1,1,n);
}
int query(int x,int y){
    int l=lca(pos[x],pos[y]);
    int mid=(lr[l].first+lr[l].second)>>1;
    return max(maxn[l][0][mid-x+1],maxn[l][1][y-mid]);
}
void solve(){
    int n,m;
    cin>>n>>m;
    vector<int> v(n+1);
    for(int i=1;i<=n;i++){
        cin>>v[i];
    }
    CatTree cattree(n,v);
    while(m--){
        int x,y;
        cin>>x>>y;
        cout<<cattree.query(x,y)<<"\n";
    }
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

字符串

序列自动机

$nxt[i][j]$ 表示从第*i*个位置开始，字符串*j*出现的第一个位置

```
struct SubsequenceAutomaton{
    string s;
    int n;
    vector<array<int,26>> nxt;
    SubsequenceAutomaton(string ss):s(ss){
        n=s.size();
        nxt.resize(n+1);
        for(int i=0;i<26;i++) nxt[n][i]=-1;
        for(int i=n-1;i>=0;i--){
            nxt[i]=nxt[i+1];
            if(s[i]>='a'&&s[i]<='z') nxt[i][s[i]-'a']=i;
        }
    }
    inline int query(int pos,string t){
        if(pos>=n) return -1;
        for(int i=0;i<t.size();i++){
            int p=t[i]-'a';
            if(nxt[pos][p]==-1) return -1;
            pos=nxt[pos][p];
            if(i!=t.size()-1) pos++;
        }
        return pos;
    }
};
```

AC自动机

多模式串匹配

```
template<int Base>
struct ACAutomaton{
    vector<array<int,26>> tree;
    vector<int> ed;
    vector<int> fail;
    vector<vector<int>> id;
    vector<int> exist;
    vector<int> tag;
    vector<int> in;
    int cnt;
    void insert(string &s,int num){
        int u=0;
        for(char &c:s){
            if(!tree[u][c-Base]){
                tree[u][c-Base]=++cnt;
                tree.emplace_back();
                tree.back().fill(0);
                ed.emplace_back(0);
                fail.emplace_back(0);
            }
            u=tree[u][c-Base];
        }
    }
};
```

```

        id.emplace_back();
        in.emplace_back(0);
    }
    u=tree[u][c-Base];
}
ed[u]++;
id[u].push_back(num);
}

void build(){
queue<int> q;
for(int i=0;i<26;i++){
    if(tree[0][i]) q.push(tree[0][i]);
}
while(!q.empty()){
    int u=q.front();
    q.pop();
    for(int i=0;i<26;i++){
        if(tree[u][i]){
            fail[tree[u][i]]=tree[fail[u]][i];
            in[tree[fail[u]][i]]++;
            q.push(tree[u][i]);
        }else{
            tree[u][i]=tree[fail[u]][i];
        }
    }
}
}

void topo(){
queue<int> q;
for(int i=1;i<=cnt;i++){
    if(!in[i]) q.push(i);
}
while(!q.empty()){
    int f=q.front();
    q.pop();
    for(int &p:id[f]) exist[p]=tag[f];
    int u=fail[f];
    tag[u]+=tag[f];
    if(!(--in[u])) q.push(u);
}
}

void query(string &s){
tag.resize(cnt+1,0);
int u=0,ans=0;
for(int i=0;i<s.size();i++){
    u=tree[u][s[i]-Base];
    tag[u]++;
}
topo();
}

ACAutomaton(vector<string> &v){
tree.resize(1);
in.resize(1,0);
exist.resize(v.size(),0);
id.resize(v.size());
cnt=0;
}

```

```

        tree.back().fill(0);
        fail.push_back(0);
        ed.push_back(0);
        for(int i=0;i<v.size();i++) insert(v[i],i);
        build();
    }
};

```

Manacher

求回文串长度

(s的下标+1)*2对应p数组

p数组的值-1对应了回文串长度

```

vector<int> manacher(string s){
    string cur="^";
    for(char &c:s){
        cur+= '#';
        cur+=c;
    }
    cur+= '#';
    cur+= '@';
    vector<int> p(cur.size(),0);
    int r=0,mid=0;
    for(int i=1;i<cur.size()-1;i++){
        p[i]=i<=r?min(p[2*mid-i],r-i+1):1;
        while(i-p[i]>0&&i+p[i]<cur.size()-1&&cur[i-p[i]]==cur[i+p[i]]) p[i]++;
        if(i+p[i]-1>r){
            r=i+p[i]-1;
            mid=i;
        }
    }
    return p;
}

```

Trie Tree

```

struct TRIE{
    int tot=0,sz=0;
    vector<vector<int>> tree;
    vector<int> ed;
    TRIE(vector<string> &v,int sz){
        this->sz=sz;
        tree.push_back(vector<int>(sz,0));
        ed.push_back(0);
        for(string &p:v) insert(p);
    }
    void insert(string s){
        int now=0;
        for(char &c:s){
            if(tree[now][ma[c]]==0){
                tree[now][ma[c]]=++tot;

```

```

        tree.push_back(vector<int>(sz, 0));
        ed.push_back(0);
    }
    now=tree[now][ma[c]];
}
ed[now]++;
}

int query(string s){
    int now=0, ans=0;
    for(char &c:s){
        if(tree[now][ma[c]]==0) return 0;
        ans=ed[tree[now][ma[c]]];
        now=tree[now][ma[c]];
    }
    return ans;
}
};


```

扩展KMP

对于一个长度为n的字符串，定义函数 $z[i]$ ，表示 s 和 $s[i, n-1]$ （即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被称为 s 的 z 函数，其中 $z[0]=0$ 。

```

vector<int> z_function(string s){
    int n=(int)s.size();
    vector<int> z(n);
    for(int i=1, l=0, r=0; i<n; i++){
        if(i<=r&&z[i-1]<r-i+1){
            z[i]=z[i-1];
        }else{
            z[i]=max(0, r-i+1);
            while(i+z[i]<n&&s[z[i]]==s[i+z[i]]) ++z[i];
        }
        if(i+z[i]-1>r) l=i, r=i+z[i]-1;
    }
    return z;
}

```

本质不同的字串数

给定一个长度为n的字符串 s ，计算 s 本质不同的子串的数量。

每次在后面新增一个字符 c ，计算新增的本质不同的子串的数量（以 c 结尾且之前未出现过的子串）。令 t 为 $s+c$ 的反串（将原来的字符串倒序排列），计算出 t 的 z_{max} ，新增本质不同的子串的数量即为 $|t| - z_{max}$

前缀函数

$\pi[i]$ 表示子串 $[0, i]$ 最长的相等的真前缀与真后缀的长度

其中 $\pi[0]=0$

```

vector<int> prefix_function(string s){
    int n=(int)s.length();
    vector<int> pi(n);
    pi[0]=0;
    for(int i=1;i<n;i++){
        int j=pi[i-1];
        while(j>0&&s[i]!=s[j]) j=pi[j-1];
        if(s[i]==s[j]) j++;
        pi[i]=j;
    }
    return pi;
}

```

KMP函数

给定一个文本text和一个字符串pattern，找到并展示s在t中的所有出现位置，时间复杂度O(n+m)

```

vector<int> kmp(string text,string pattern){
    string cur=pattern+'#'+text;
    int sz1=text.size(),sz2=pattern.size();
    vector<int> v;
    vector<int> lps=prefix_function(cur);
    for(int i=sz2+1;i<=sz1+sz2;i++){
        if(lps[i]==sz2) v.push_back(i-sz2);
    }
    return v;
}

```

字符串哈希

```

const int HASHMOD[2]={998244353,(int)1e9+7};
const int BASE[2]={29,31};
struct Stringhash{
    static vector<int> qpow[2];
    vector<int> hash[2];
    void init(){
        qpow[0].push_back(1);
        qpow[1].push_back(1);
        for(int i=1;i<=1e6;i++){
            for(int j=0;j<2;j++){
                qpow[j].push_back(qpow[j].back()*BASE[j]%HASHMOD[j]);
            }
        }
    }
    Stringhash(string s,int base){
        for(int i=0;i<2;i++){
            hash[i]=vector<int>(s.size()+1);
            hash[i][0]=0;
        }
        if(qpow[0].empty()) init();
        for(int i=1;i<=s.size();i++){
            for(int j=0;j<2;j++){

```

```

        hash[j][i]=(hash[j][i-1]*BASE[j]%HASHMOD[j]+s[i-1]-
base)%HASHMOD[j];
    }
}
pair<int,int> gethash(int x,int y){
    pair<int,int> result={0,0};
    for(int i=0;i<2;i++){
        int k=((hash[i][y]-hash[i][x-1]*qpow[i][y-
x+1])%HASHMOD[i]+HASHMOD[i])%HASHMOD[i];
        if(i==0) result.first=k;
        else result.second=k;
    }
    return result;
}
vector<int> Stringhash::qpow[2];

```

字符串的周期

如果s长度为r的前缀和长度为r的后缀相等，那么s长度为r的前缀是s的一个border，n-r是s的一个周期。

$\pi[n-1], \pi[\pi[n-1]], \dots$ 为所有border的长度

最小周期为 $n-\pi[n-1]$

最小表示法

$s[i\dots n]+s[1\dots i-1]=T$

s的最小表示法就是与S循环同构的字典序最小的字符串

```

int k=0,i=0,j=1;
while(k<n&&i<n&&j<n){
    if(v[(i+k)%n]==v[(j+k)%n]){
        k++;
    }else{
        if(v[(i+k)%n]>v[(j+k)%n]){
            i+=k+1;
        }else{
            j+=k+1;
        }
        if(i==j) i++;
        k=0;
    }
}
i=min(i,j);

```

后缀数组 (SA)

$sa[i]$ 表示所有后缀排序后第*i*小的后缀的编号， $rk[i]$ 表示后缀*i*的排名， $height[i]$ 表示第*i*名后缀与前一名的后缀的最长公共前缀

$$height[rk[i]] \geq height[rk[i-1]] - 1$$

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    string s;
    cin>>s;
    auto getsa=[&](const string &s){
        int n=s.size();
        vector<int> sa(n),rk(n);
        iota(sa.begin(),sa.end(),0);
        sort(sa.begin(),sa.end(),[&](int a,int b){
            return s[a]<s[b];
        });
        rk[sa[0]]=0;
        for(int i=1;i<n;i++){
            rk[sa[i]]=rk[sa[i-1]];
            if(s[sa[i]]!=s[sa[i-1]]) rk[sa[i]]++;
        }
        for(int len=1;len<n;len<=1){
            vector<tuple<int,int,int>> v;//第一关键字，第二关键字，下标
            v.reserve(n);
            for(int i=0;i<n;i++){
                v.emplace_back(rk[i],(i+len<n?rk[i+len]:-1),i);
            }
            sort(v.begin(),v.end(),[](const auto &a,const auto &b){
                if(get<0>(a)!=get<0>(b)) return get<0>(a)<get<0>(b);
                if(get<1>(a)!=get<1>(b)) return get<1>(a)<get<1>(b);
                return get<2>(a)<get<2>(b);
            });
            for(int i=0;i<n;i++){
                sa[i]=get<2>(v[i]);
            }
            rk[sa[0]]=0;
            for(int i=1;i<n;i++){
                rk[sa[i]]=rk[sa[i-1]];
                if(get<0>(v[i-1])!=get<0>(v[i])||get<1>(v[i-1])!=get<1>(v[i])){
                    rk[sa[i]]++;
                }
            }
        }
        vector<int> height(n);
        for(int i=0,k=0;i<n;i++){
            if(!rk[i]){
                k=0;
                continue;
            }
            if(k) --k;
            while(i+k<n&&sa[rk[i]-1]+k<n&&s[i+k]==s[sa[rk[i]-1]+k]) k++;
            height[rk[i]]=k;
        }
        return make_pair(sa,height);
    };
    auto [sa,height]=getsa(s);
    for(int i=0;i<s.size();i++) cout<<sa[i]<<" ";
    cout<<"\n";
    for(int i=0;i<s.size();i++) cout<<height[i]<<" ";
}

```

```

        cout<<"\n";
    }
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

SAM

```

#include<bits/stdc++.h>
using namespace std;
using ll=long long;
struct SAM{
    struct edge{
        int len,link,cnt;
        array<int,26> nxt;
        edge(int len):len(len),link(-1),cnt(0){nxt.fill(0);}
    };
    vector<edge> node;
    int last;
    SAM():last(0),node(1,0){}
    void add(char c){
        int cur=node.size();
        node.emplace_back(node[last].len+1);
        node[cur].cnt=1;
        int p=last;
        while(p!=-1&&!node[p].nxt[c-'a']){
            node[p].nxt[c-'a']=cur;
            p=node[p].link;
        }
        if(p==-1){
            node[cur].link=0;
        }else{
            int q=node[p].nxt[c-'a'];
            if(node[q].len==node[p].len+1){
                node[cur].link=q;
            }else{
                int clone=node.size();
                node.emplace_back(node[q]);
                node[clone].cnt=0;
                node[clone].len=node[p].len+1;
                node[q].link=node[cur].link=clone;
                while(p!=-1&&node[p].nxt[c-'a']==q){
                    node[p].nxt[c-'a']=clone;
                    p=node[p].link;
                }
            }
        }
        last=cur;
    }
    void solve(){

```

```

string s;
cin>>s;
SAM sam;
for(char &c:s) sam.add(c);
vector<vector<int>> son(sam.node.size());
vector<ll> dp(sam.node.size());
ll ans=0;
for(int i=1;i<sam.node.size();i++){
    son[sam.node[i].link].push_back(i);
}
vector<int> id(sam.node.size());
iota(id.begin(),id.end(),0);
sort(id.begin(),id.end(),[&](int &a,int &b){
    return sam.node[a].len>sam.node[b].len;
});
for(auto &x:id){
    dp[x]=sam.node[x].cnt;
    for(auto &p:son[x]){
        dp[x]+=dp[p];
    }
    if(dp[x]>1){
        ans=max(ans,(ll)dp[x]*sam.node[x].len);
    }
}
cout<<ans<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

数学

高精度

压位高精度。在int下，加法压9位，乘法压3位，long long压4位

```

//压位高精，压Base位
template<int Base>
struct BigNum{
    constexpr int pow(int x,int y){
        int ans=1,base=x;
        while(y){
            if(y&1) ans=ans*base;
            base*=base;
            y>>=1;
        }
        return ans;
    }
    const int mod;
    vector<int> v;
};

```

```

BigNum(int n):mod(pow(10,Base)){
    if(n==0) v.push_back(0);
    while(n){
        v.push_back(n%mod);
        n/=mod;
    }
}
vector<int> stos(string &s) const{
    vector<int> v;
    int len=s.size();
    for(int i=len-Base;i+Base-1>=0;i-=Base){
        string k=s.substr(max(0ll,i),min(i+Base-1-max(0ll,i)+1,Base));
        v.push_back(stoi(k));
    }
    return v;
}
BigNum(string &s):mod(pow(10,Base)),v(stos(s)){}
BigNum(vector<int> &_):v(_),mod(pow(10,Base)){};
BigNum():mod(pow(10,Base)){v.push_back(0);}
BigNum operator+(const BigNum &e) const{
    vector<int> ans;
    int i;
    for(i=0;i<min(e.v.size(),v.size());i++){
        ans.push_back(e.v[i]+v[i]);
    }
    for(;i<e.v.size();i++){
        ans.push_back(e.v[i]);
    }
    for(;i<v.size();i++){
        ans.push_back(v[i]);
    }
    for(int i=0;i<ans.size();i++){
        if(ans[i]>=mod){
            if(i+1==ans.size()) ans.push_back(0);
            ans[i+1]+=ans[i]/mod;
            ans[i]%=mod;
        }
    }
    while(ans.back()==0) ans.pop_back();
    return BigNum(ans);
}
BigNum operator-(const BigNum &e) const{
    vector<int> ans;
    int i;
    for(i=0;i<min(e.v.size(),v.size());i++){
        ans.push_back(v[i]-e.v[i]);
    }
    for(;i<v.size();i++){
        ans.push_back(v[i]);
    }
    for(int i=0;i<ans.size();i++){
        if(ans[i]<0){
            int t=(-ans[i]+mod-1)/mod;
            ans[i]+=t*mod;
            ans[i+1]-=t;
        }
    }
}

```

```

    }
    while(ans.back()==0) ans.pop_back();
    return BigNum(ans);
}

BigNum operator*(const BigNum &e) const{
    vector<int> ans;
    for(int i=0;i<v.size();i++){
        for(int j=0;j<e.v.size();j++){
            while(i+j==ans.size()) ans.push_back(0);
            ans[i+j]+=v[i]*e.v[j];
        }
    }
    for(int i=0;i<ans.size();i++){
        if(ans[i]>=mod){
            if(i+1==ans.size()) ans.push_back(0);
            ans[i+1]+=ans[i]/mod;
            ans[i]%=mod;
        }
    }
    while(ans.back()==0) ans.pop_back();
    return BigNum(ans);
}

void operator+=(const BigNum &e){
    v=(*this+e).v;
}

void operator-=(const BigNum &e){
    v=(*this-e).v;
}

void operator*=(const BigNum &e){
    v=(*this*e).v;
}

void operator=(int x){
    v.clear();
    if(x==0) v.push_back(0);
    else{
        while(x){
            v.push_back(x%mod);
            x/=mod;
        }
    }
}

void operator=(const BigNum &e){
    v=e.v;
}

BigNum operator+(int x) const{
    return (*this)+BigNum<Base>(x);
}

BigNum operator*(int x) const{
    return (*this)*BigNum<Base>(x);
}

int getlen() const{
    if(v.empty()) return 0;
    int len=0;
    if(v.size()==1){
        if(v.front()==0) return 1;
        int tmp=v.front();
        if(tmp>=mod) len++;
    }
}

```

```

        while(tmp){
            len++;
            tmp/=10;
        }
        return len;
    }else{
        int tmp=v.back();
        while(tmp){
            len++;
            tmp/=10;
        }
        return len+Base*(v.size()-1);
    }
}
bool operator<(const BigNum &e) const{
    int len1=getlen(),len2=e.getlen();
    if(len1!=len2) return len1<len2;
    for(int i=v.size()-1;i>=0;i--){
        if(v[i]<e.v[i]) return 1;
        if(v[i]>e.v[i]) return 0;
    }
    return 0;
}
friend ostream& operator<<(ostream& os,const BigNum& obj){
    for(int i=obj.v.size()-1;i>=0;i--){
        if(i!=obj.v.size()-1){
            for(int j=obj.mod/10;j>=1;j/=10){
                os<<obj.v[i]/j%10;
            }
        }else{
            os<<obj.v[i];
        }
    }
    return os;
}
friend istream& operator>>(istream& is,BigNum& obj){
    string s;
    is>>s;
    obj.v=obj.stos(s);
    return is;
}
};

```

```

const int base = 1000;
const int base_digits = 3; // 分解为九个数位一个数字
struct bigint {
    vector<int> a;
    int sign;

    bigint() : sign(1) {}
    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }
}

```

```

bigint(long long v) {
    *this = v;
}
bigint(const string &s) {
    read(s);
}
void operator=(const bigint &v) {
    sign = v.sign;
    a = v.a;
}
void operator=(long long v) {
    a.clear();
    sign = 1;
    if (v < 0)
        sign = -1, v = -v;
    for (; v > 0; v = v / base) {
        a.push_back(v % base);
    }
}

// 基础加减乘除
bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;
        for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.size()) || carry; ++i) {
            if (i == (int)res.a.size())
                res.a.push_back(0);
            res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
            carry = res.a[i] >= base;
            if (carry)
                res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}
bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i) {
                res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
                carry = res.a[i] < 0;
                if (carry)
                    res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}

```

```

}

void operator*(int v) {
    check(v);
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size())
            a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
    }
    trim();
}

void operator/(int v) {
    check(v);
    for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long)base;
        a[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i) {
        m = (a[i] + m * (long long)base) % v;
    }
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}

void operator-=(const bigint &v) {
    *this = *this - v;
}

bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

void operator%=(const int &v) {
    *this = *this % v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
}

```

```

    if (a.size() != v.a.size())
        return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[i] * sign;
    return false;
}
bool operator>(const bigint &v) const {
    return v < *this;
}
bool operator<=(const bigint &v) const {
    return !(v < *this);
}
bool operator>=(const bigint &v) const {
    return !(*this < v);
}
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}
void check(int v) { // 检查输入的是否为负数
    if (v < 0) {
        sign = -sign;
        v = -v;
    }
}
void trim() { // 去除前导零
    while (!a.empty() && !a.back()) a.pop_back();
    if (a.empty())
        sign = 1;
}
bool isZero() const { // 判断是否等于零
    return a.empty() || (a.size() == 1 && !a[0]);
}
friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}
void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
        if (s[pos] == '-')
            sign = -sign;
        ++pos;
    }
    if (pos < s.size() && (s[pos] >='0' && s[pos] <='9')) {
        string num;
        while (pos < s.size() && (s[pos] >='0' && s[pos] <='9'))
            num += s[pos++];
        a = num;
    }
}

```

```

    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i; j++) x = x * 10 +
s[j] - '0';
        a.push_back(x);
    }
    trim();
}

friend istream &operator>>(istream &stream, bigint &v) {
string s;
stream >> s;
v.read(s);
return stream;
}

friend ostream &operator<<(ostream &stream, const bigint &v) {
if (v.sign == -1)
    stream << '-';
stream << (v.a.empty() ? 0 : v.a.back());
for (int i = (int)v.a.size() - 2; i >= 0; --i)
    stream << setw(base_digits) << setfill('0') << v.a[i];
return stream;
}

/* 大整数乘除大整数部分 */
typedef vector<long long> vll;
bigint operator*(const bigint &v) const { // 大整数乘大整数
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size()) a.push_back(0);
    while (b.size() < a.size()) b.push_back(0);
    while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int)c.size(); i++) {
        long long cur = c[i] + carry;
        res.a.push_back((int)(cur % 1000000));
        carry = (int)(cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
friend pair<bigint, bigint> divmod(const bigint &a1,
                                    const bigint &b1) { // 大整数除大整数，同时返

```

回答案与余数

```

int norm = base / (b1.a.back() + 1);
bigint a = a1.abs() * norm;
bigint b = b1.abs() * norm;
bigint q, r;
q.a.resize(a.a.size());
for (int i = a.a.size() - 1; i >= 0; i--) {
    r *= base;
    if (r >= b) {
        q.a[i] = r / b;
        r -= q.a[i] * b;
    }
}
q.trim();
r.trim();
return {q, r};
}

```

```

        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
        int d = ((long long)base * s1 + s2) / b.a.back();
        r -= b * d;
        while (r < 0) r += b, --d;
        q.a[i] = d;
    }
    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}
static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back((int)(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back()) res.pop_back();
    return res;
}
static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                res[i + j] += a[i] * b[j];
            }
        }
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);

```

```

        for (int i = 0; i < k; i++) a2[i] += a1[i];
        for (int i = 0; i < k; i++) b2[i] += b1[i];

        v11 r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];

        for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
        for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
        return res;
    }

    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }
    bigint operator%=(const bigint &v) const {
        return divmod(*this, v).second;
    }
    void operator%=(const bigint &v) {
        *this = *this % v;
    }
};


```

矩阵相关

```

const double eps=1e-8;
struct matrix{
    vector<vector<double>> v;
    int n,m;
    matrix(int n,int m):n(n),m(m){
        v.assign(n,vector<double>(m));
    }
    matrix(vector<vector<double>> &v):v(v),n(v.size()),m(v[0].size()){}
    //第n行第m列变成x
    void set(int n,int m,double x){
        v[n-1][m-1]=x;
    }
    matrix operator*(const matrix &e) const{
        matrix ans(n,e.m);
        for(int i=0;i<n;i++){
            for(int j=0;j<e.m;j++){
                for(int k=0;k<m;k++){
                    ans.v[i][j]+=v[i][k]*e.v[k][j];
                }
            }
        }
        return ans;
    }
};


```

```

//高斯消元
//无解-1, 无穷解0, 有唯一解1
int Gauss(){
    int column=0;
    for(int i=0;i<n;i++){
        while(column<m){
            int line=i;
            double maxn=v[i][column];
            for(int j=i+1;j<n;j++){
                if(fabs(v[j][column])>maxn){
                    maxn=v[j][column];
                    line=j;
                }
            }
            swap(v[i],v[line]);
            if(fabs(v[i][column])<eps){
                column++;
                continue;
            }
            double k=v[i][column];
            for(int j=column;j<m;j++){
                v[i][j]/=k;
            }
            for(int j=0;j<n;j++){
                if(j==i) continue;
                k=v[j][column];
                for(int z=column;z<m;z++){
                    v[j][z]-=k*v[i][z];
                }
            }
            break;
        }
    }
    int inf=0;
    for(int i=0;i<n;i++){
        bool ok=0;
        for(int j=0;j<m-1;j++){
            if(fabs(v[i][j])>eps){
                ok=1;
                break;
            }
        }
        if(!ok){
            if(fabs(v[i][m-1])>eps) return -1;
            inf++;
        }
    }
    return inf==0;
}
};


```

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
int quickpow(int x,int y,int mod){


```

```

int ans=1,base=x;
while(y){
    if(y&1) ans=ans*base%mod;
    base=base*base%mod;
    y>>=1;
}
return ans;
}

void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(n,vector<int>(n+1));
    for(int i=0;i<n;i++){
        int k;
        cin>>k;
        v[i][i]=1;
        while(k--){
            int x;
            cin>>x;
            v[x-1][i]=1;
        }
    }
    vector<int> s(n),t(n);
    for(int i=0;i<n;i++){
        cin>>s[i];
    }
    for(int i=0;i<n;i++){
        cin>>t[i];
    }
    for(int i=0;i<n;i++){
        v[i][n]=((t[i]-s[i])%m+m)%m;
    }
    vector<pair<int,int>> zy;
    auto gauss=[&](int N,int M){
        int cur=0;
        for(int i=0;i<M;i++){
            int now=cur;
            while(now<n&&!v[now][i]) now++;
            if(now==n) continue;
            if(now!=cur) swap(v[now],v[cur]);
            zy.emplace_back(cur,i);
            for(int j=0;j<n;j++){
                if(j==cur||!v[j][i]) continue;
                const int inv=quickpow(v[cur][i],m-2,m)*v[j][i]%m;
                for(int k=i;k<M;k++){
                    v[j][k]=((v[j][k]-inv*v[cur][k])%m+m)%m;
                }
            }
            cur++;
        }
    };
    gauss(n,n+1);
    if(zy.size()&&zy.back().second==n){
        cout<<"niuza\n";
        return;
    }
}

```

```

vector<int> ans(n);
int pre=n;
auto combine=[&](int x,int t){
    for(int i=0;i<n;i++){
        v[i][n]=((v[i][n]-t*v[i][x])%m+m)%m;
    }
};
for(int i=(int)(zy.size()-1;i>=0;i--){
    const int inv=quickpow(v[zy[i].first][zy[i].second],m-2,m)*v[zy[i].first]
[n]%m;
    ans[zy[i].second]=inv;
    combine(zy[i].second,inv);
}
for(int &p:ans) cout<<p<<" ";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

高斯消元解异或方程组，所有的加减乘除操作变成异或

```

//异或版本
struct matrix{
    vector<vector<int>> v;
    int n,m;
    matrix(int n,int m):n(n),m(m){
        v.assign(n,vector<int>(m));
    }
    matrix(vector<vector<int>> &v):v(v),n(v.size()),m(v[0].size()){}
    //第n行第m列变成x
    void set(int n,int m,int x){
        v[n-1][m-1]=x;
    }
    matrix operator*(const matrix &e) const{
        vector<vector<int>> ans(n,vector<int>(e.m,0));
        for(int i=0;i<n;i++){
            for(int j=0;j<e.m;j++){
                for(int k=0;k<m;k++){
                    ans[i][j]+=v[i][k]*e.v[k][j];
                }
            }
        }
        return ans;
    };
    //高斯消元
    //无解-1，无穷解0，有唯一解1
    int Gauss(){
        int column=0;
        for(int i=0;i<n;i++){
            while(column<m){
                int line=i;

```

```

        int maxn=v[i][column];
        for(int j=i+1;j<n;j++){
            if(fabs(v[j][column])>maxn){
                maxn=v[j][column];
                line=j;
            }
        }
        swap(v[i],v[line]);
        if(v[i][column]==0){
            column++;
            continue;
        }
        for(int j=0;j<n;j++){
            if(j==i) continue;
            int k=v[j][column];
            for(int z=column;z<m;z++){
                v[j][z]^=k&v[i][z];
            }
        }
        break;
    }
}
int inf=0;
for(int i=0;i<n;i++){
    bool ok=0;
    for(int j=0;j<m-1;j++){
        if(v[i][j]!=0){
            ok=1;
            break;
        }
    }
    if(!ok){
        if(v[i][m-1]!=0) return -1;
        inf++;
    }
}
return inf==0;
}
};

```

```

#include<bits/stdc++.h>
using namespace std;
void solve(){
    int n,k;
    cin>>n>>k;
    vector<vector<int>> v(k+1,vector<int>(n));
    vector<pair<int,int>> zy;
    auto gauss=[&](int n,int m){
        int cur=0;
        for(int i=0;i<m;i++){
            int now=cur;
            while(now<n&&!v[now][i]) now++;
            if(now==n) continue;
            if(now!=cur) swap(v[now],v[cur]);
            zy.emplace_back(cur,i);
        }
    };
}
```

```

        for(int j=0;j<n;j++){
            if(cur!=j&&v[j][i]){
                for(int k=0;k<m;k++){
                    v[j][k]^=v[cur][k];
                }
            }
            cur++;
        }
        return;
    };
    for(int i=0;i<n;i++){
        string s;
        cin>>s;
        for(int j=0;j<k;j++){
            v[j][i]=s[j]-'0';
        }
        v[k][i]=1;
    }
    gauss(k+1,n);
    if(zy.size()==n){
        cout<<"*`\n";
    }else{
        vector<int> pre(k+1),ans;
        int now=zy.size()-1;
        auto combine=[&](int x){
            for(int i=0;i<=k;i++){
                pre[i]^=v[i][x];
            }
        };
        for(int i=n-1;i>=0;i--){
            //如果是主元列
            if(i==zy[now].second){
                if(pre[zy[now].first]==1){
                    ans.push_back(i);
                    combine(i);
                }
                now--;
            }else{
                ans.push_back(i);
                combine(i);
            }
        }
        assert(count(pre.begin(),pre.end(),1)==0);
        vector<int> result(n);
        for(int i=0;i<ans.size();i++){
            if(i<ans.size()/2) result[ans[i]]=1;
            else result[ans[i]]=2;
        }
        for(int &p:result) cout<<p;
    }
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
}

```

```

    while(t--) solve();
    return 0;
}

```

卡特兰数

有一个大小为n*n的方格图，左下角为(0,0)，右上角为(n,n)，从左下角开始每次只能向右或者向上走一个单位，不能走到y=x上方（但可以触碰），有几种可能的路径

1 1 2 5 14 42

递推式：

$$\begin{aligned}
 H_n &= \frac{\binom{2n}{n}}{n+1} \quad (n \geq 2, n \in N_+) \\
 H_n &= \begin{cases} \sum_{i=1}^n H_{i-1} H_{n-i}, & n \geq 2, \quad n \in N_+ \\ 1, & n = 0, 1 \end{cases} \\
 H_n &= \frac{H_{n-1}(4n-2)}{n+1} \\
 H_n &= \binom{2n}{n} - \binom{2n}{n-1}
 \end{aligned}$$

MillerRabin

判断某个数是否是质数

```

struct MillerRabin{
    vector<int> Prime;
    MillerRabin():Prime({2,3,5,7,11,13,17,19,23}){}
    static constexpr int mulp(const int &a,const int &b,const int &p){
        int res=a*b-(int)(1.L*a*b/P)*P;
        res%=P;
        res+=(res<0?P:0);
        return res;
    }
    static constexpr int powp(int a,int mi,const int &mod){
        int ans = 1;
        for(;mi;mi>>=1){
            if(mi&1) ans=mulp(ans,a,mod);
            a=mulp(a,a,mod);
        }
        return ans;
    }
    bool operator()(const int &v){
        if(v<2||v!=2&&v%2==0) return false;
        int s=v-1;
        while(!(s&1)) s>>=1;
        for(int x:Prime){
            if(v==x) return true;
            int t=s,m=powp(x,s,v);
            while(t!=v-1&&m!=1&&m!=v-1) m=mulp(m,m,v),t<<=1;
            if(m!=v-1&&(t&1)) return false;
        }
        return true;
    }
}

```

```
    }  
};
```

PollardRho

判断质数（使用millerrabin判断），计算因子

```
struct PollardRho:public MillerRabin{  
    mt19937 myrand;  
    PollardRho():myrand(time(0)){}  
    int rd(int l,int r){  
        return myrand()%(r-l+1)+l;  
    }  
    int operator()(int n) { //返回n的随机一个[2, n-1]内的因子，或者判定是质数  
        if(n==4) return 2;  
        MillerRabin &super=*this;  
        //如果n是质数直接返回n  
        if(super(n)) return n;  
        while(1){  
            int c=rd(l,n-1);  
            auto f=[&](int x){  
                return (super.mul(x,x,n)+c)%n;  
            };  
            int t=0,r=0,p=1,q;  
            do{  
                for(int i=0;i<128;i++){  
                    t=f(t),r=f(f(r));  
                    if(t==r||(q=super.mul(p,abs(t-r),n))==0) break;  
                    p=q;  
                }  
                int d=__gcd(p,n);  
                if(d>1) return d;  
            }while(t!=r);  
        }  
    }  
};
```

超快质因数分解&求约数

时间复杂度 $O(n^{\frac{1}{4}})$

```
stack<int> st;  
st.push(x);  
map<int,int> ma;  
while(!st.empty()){  
    int f=st.top();  
    st.pop();  
    int k=rho(f);  
    if(k==f){  
        ma[k]++;  
    }else{  
        st.push(k);  
        st.push(f/k);  
    }  
}
```

```

    }
    vector<pair<int,int>> v;
    for(auto &[p,q]:ma) v.push_back({p,q});
    function<void(int,int)> dfs=[&](int id,int now){
        if(id==v.size()){
            if(i<now&&cal(now)==i) ans++;
            return;
        }
        for(int i=0;i<=v[id].second;i++){
            dfs(id+1,now);
            now*=v[id].first;
        }
    };
}

```

线性筛质数

```

struct Eulersieve{
    vector<int> prime;
    vector<int> v;
    int n;
    Eulersieve(int n):v(n+1){
        this->n=n;
        for(int i=2;i<=n;i++){
            if(v[i]==0){
                prime.push_back(i);
                v[i]=i;
            }
            for(int &p:prime){
                if(i*p>n) break;
                v[i*p]=p;
                if(i%p==0) break;
            }
        }
    }
    vector<int> getdiv(int x) const{
        vector<int> _div(1,1);
        while(x>1){
            int d=v[x];
            int l=0,r=_div.size();
            while(x%d==0){
                for(int k=l;k<r;k++){
                    _div.push_back(_div[k]*d);
                }
                x/=d;
                l=r;
                r=_div.size();
            }
        }
        return _div;
    }
};

```

线性筛欧拉函数

小于等于n且与n互质的正整数数量

```
struct Eulersieve{
    vector<int> prime;
    vector<bool> isPrime;
    vector<int> phi;
    int n;
    Eulersieve(int n){
        this->n=n;
        isPrime=vector<bool>(n+1,1);
        phi=vector<int>(n+1);
        isPrime[1]=0;
        for(int i=2;i<=n;i++){
            if(isPrime[i]){
                prime.push_back(i);
                phi[i]=i-1;
            }
            for(int &p:prime){
                if(i*p>n) break;
                isPrime[i*p]=0;
                if(i%p==0){
                    phi[i*p]=phi[i]*p;
                    break;
                }else{
                    phi[i*p]=phi[i]*phi[p];
                }
            }
        }
    }
};
```

直接求欧拉函数

```
int phi(int n){
    int ans=n;
    for(int i=2;i*i<=n;i++){
        if(n%i==0){
            ans=ans/i*(i-1);
            while(n%i==0) n/=i;
        }
    }
    if(n>1) ans=ans/n*(n-1);
    return ans;
}
```

莫比乌斯函数

$$\mu(n) = \begin{cases} 1, & \text{若 } n = 1 \\ 0, & \text{若 } n \text{ 能被一个素数的平方整除} \\ (-1)^r, & \text{若 } n \text{ 是 } r \text{ 个 } (r \geq 1) \text{ 不同素数的乘积} \end{cases}$$

组合数学

```
template<int MOD>
struct Comb{
    vector<int> jc,ijc;
    int quickpow(int x,int y){
        x%=MOD;
        if(x==0) return 0;
        int ans=1,base=x;
        while(y){
            if(y&1) ans=ans*base%MOD;
            base=base*base%MOD;
            y>>=1;
        }
        return ans;
    }
    Comb(int n){
        jc.resize(n+1);
        ijc.resize(n+1);
        jc[0]=1;
        for(int i=1;i<=n;i++) jc[i]=jc[i-1]*i%MOD;
        ijc[n]=quickpow(jc[n],MOD-2);
        for(int i=n-1;i>=0;i--) ijc[i]=ijc[i+1]*(i+1)%MOD;
    }
    int C(int n,int k){
        if(n<0||k<0||n<k) return 0;
        return jc[n]*ijc[k]%MOD*ijc[n-k]%MOD;
    }
    int A(int n,int k){
        if(n<0||k-1<0||n<k) return 0;
        return jc[n]*ijc[n-k]%MOD;
    }
    int CLucas(int n,int m){
        if(m==0) return 1;
        return C(n%MOD,m%MOD)*CLucas(n/MOD,m/MOD)%MOD;
    }
    int Stirling2(int n,int m){
        int ans=0;
        for(int i=0;i<=m;i++){
            ans=(ans+((m-i)%2==0?1:-1)*quickpow(i,n)%MOD*ijc[i]%MOD*ijc[m-i]%MOD)%MOD;
        }
        return ans;
    }
};
```

Lucas

用于求解问题规模很大，而模数是一个不大的质数的时候的组合数问题， p 为质数

$$C_n^m \bmod p = \begin{cases} 1 & m=0 \\ C_{\left[\frac{n}{p}\right]}^{\left[\frac{m}{p}\right]} * C_{n \bmod p}^{m \bmod p} \bmod p \end{cases}$$

$$C_n^m \bmod p = C_{a_0}^{b_0} \cdot C_{a_1}^{b_1} \cdot C_{a_2}^{b_2} \cdots C_{a_k}^{b_k}$$

$$n = a_0 + a_1 p + a_2 p^2 + a_3 p^3$$

$$m = b_0 + b_1 p + b_2 p^2 + b_3 p^3$$

m、n按p进制展开

$n \& m == m$, C_n^m 为奇数

上指标求和

$$\sum_{i=0}^n \binom{i}{k} = \binom{n+1}{k+1}$$

考虑添加一个虚拟的第 $k+1$ 个元素，枚举这个元素所占据的位置为 i ，则前 k 个元素应在前 $i-1$ 个位置排列， i 的范围为 $[1, n+1]$ ，则有

$$\binom{n+1}{k+1} = \sum_{i=1}^{n+1} \binom{i-1}{k} = \sum_{i=0}^n \binom{i}{k}$$

二项式反演

$$g_n = \sum_{i=0}^n C_n^i f_i$$

$$f_n = \sum_{i=0}^n C_n^i (-1)^{n-i} g_i$$

$$g_k = \sum_{i=k}^n C_i^k f_i$$

$$f_k = \sum_{i=k}^n C_i^k (-1)^{i-k} g_i$$

第二类斯特林数

将n个两两不同的元素，划分为k个互不区分的非空子集的方案数

康托展开

用于全排列的状态压缩，是一个全排列到一个自然数的映射，康托展开的实质是计算当前排列在所有从小到大的排列中的次序编号

康托展开的表达式为 $X = a_n(n-1)! + a_{n-1}(n-2)! + \dots + a_1 \cdot 0!$

其中X为比当前排列小的全排列个数， $(X+1)$ 即为当前排列的次序编号，n表示全排列的长度， a_i 表示原排列中的第*i*位（从右往左从低到高）在当前未出现（剩下未被选择）的元素集合中比其小的元素个数

时间复杂度 n^2 ，用树状数组可优化为 $n \log n$

还原：先让排名-1，从高位开始，每轮整除 $i!$ ，即可得到当前位有多少个数小于他（去掉已经存在的），线段树优化为 $\log n$

```
struct Cantor{
    struct SegmentTree{
        vector<int> tree;
        int n;
        SegmentTree(int n):tree((n<<2)+10, 0), n(n){}
    }
}
```

```

    void update(int x,int delta){
        function<void(int,int,int,int,int)> realupdate=[&](int id,int l,int r,int x,int delta){
            tree[id]+=delta;
            if(l==r){
                return;
            }
            int mid=l+(r-l>>1);
            if(x<=mid) realupdate(id<<1,l,mid,x,delta);
            else realupdate(id<<1|1,mid+1,r,x,delta);
        };
        realupdate(1,1,n,x,delta);
    }
    int querysum(int x,int y){
        if(x>y) return 0;
        function<int(int,int,int,int,int)> realquerysum=[&](int id,int l,int r,int x,int y){
            if(x<=l&&r<=y) return tree[id];
            int mid=l+(r-l>>1);
            int ans=0;
            if(x<=mid) ans+=realquerysum(id<<1,l,mid,x,y);
            if(y>mid) ans+=realquerysum(id<<1|1,mid+1,r,x,y);
            return ans;
        };
        return realquerysum(1,1,n,x,y);
    }
    int querykth(int k){
        function<int(int,int,int,int)> realquerykth=[&](int id,int l,int r,int k){
            if(l==r) return l;
            int mid=l+(r-l>>1);
            int lsum=mid-l+1-tree[id<<1];
            if(k<=lsum) return realquerykth(id<<1,l,mid,k);
            else return realquerykth(id<<1|1,mid+1,r,k-lsum);
        };
        return realquerykth(1,1,n,k);
    }
};

vector<int> fac;
int n;
const int mod;
SegmentTree tree;
Cantor(int n,int mod=9e18):n(n),mod(mod),tree(n){
    fac.resize(n+1);
    fac[0]=1;
    for(int i=1;i<=n;i++) fac[i]=fac[i-1]*i%mod;
}
//排名
int get(vector<int> &v){
    int ans=0;
    int sz=v.size();
    for(int i=0;i<sz;i++){
        ans=(ans+fac[sz-i-1]*(((v[i]-1-
tree.querysum(1,v[i]-1))%mod+mod)%mod)%mod)%mod;
        tree.update(v[i],1);
    }
}

```

```

        for(int i=0;i<sz;i++) tree.update(v[i], -1);
        return (ans+1)%mod;
    }
    //x: 排名, w: 位数
    vector<int> restore(int x,int w){
        vector<int> ans;
        --x;
        for(int i=w-1;i>=0;i--){
            int l=x/fac[i];
            ans.push_back(tree.querykth(l+1));
            tree.update(ans.back(), 1);
            x-=l*fac[i];
        }
        for(int i=1;i<=w;i++) tree.update(i, -1);
        return ans;
    }
};

```

快速幂

```

int quickpow(int x,int y,int mod){
    if(x==0) return 0;
    int ans=1,base=x;
    while(y){
        if(y&1) ans=ans*base%mod;
        base=base*base%mod;
        y>>=1;
    }
    return ans;
}

```

模意义下大整数乘法

计算 $a \times b \bmod p$, $a, b \leq p \leq 10^{18}$

- 龟速乘 $O(\log_2 b)$

```

// O(log2(b))
inline ll mul(ll a, ll b, const ll p) {
    assert(p > 0);
    a %= p, b %= p;
    if (a < 0) a += p; // 保证正数
    if (b < 0) b += p; // 保证正数
    ll ans = 0;
    while (b) {
        if (b & 1) {
            ans += a;
            if (ans > p) ans -= p; // 直接取模会慢很多
        }
        a <<= 1;
        if (a > p) a -= p;
        b >>= 1;
    }
    return ans;
}

```

```
}
```

- 快速乘: $a \times b \bmod p = a \times b - \lfloor a \times b/p \rfloor \times p$, 可以处理模数在 `long long` 范围内、不需要使用黑科技 `__int128` 的、复杂度为 $O(1)$

```
ll mul(ll a, ll b, ll p) {
    ll ans = a * b - ((1LL * a * b / p) * p);
    ans %= p;
    if (ans < 0) ans += p;
    return ans;
}
```

线性基

解决异或问题。原序列里的每一个数都可以由线性基里面的一些数异或得到，线性基里面任意一些数异或起来不等于0，线性基里面的数的个数唯一，且数的个数是最小的。

插入

`d[i]`存最高位1在第`i`位的数

对于每个数`x`，假设最高位的1在第`i`位，如果`d[i]`等于0，`d[i]=x`，插入完成，否则`x^=d[i]`，继续插入

```
void insert(int x){
    for(int i=51;i>=0;i--){
        if(x>>i&1){
            if(p[i]) x^=p[i];
            else{
                p[i]=x;
                break;
            }
        }
    }
}
```

查询某个数能否被异或出来

```
bool ask(int x){
    for(int i=51;i>=0;i--){
        if(x>>i&1){
            x^=p[i];
        }
    }
    return x==0;
}
```

查询异或最大值

```
int askmx(int x){
    int ans=0;
    for(int i=51;i>=0;i--){
        if((ans^p[i])>ans) ans^=p[i];
    }
    return ans;
}
```

查询异或最小值

```
int askminn(int x){
    for(int i=51;i>=0;i--){
        if(!p[i]){
            return p[i];
        }
    }
    for(int i=0;i<=51;i++){
        if(p[i]) return p[i];
    }
}
```

查询异或第k小

重构一个各个位之间互不影响的d数组

```
void rebuild(){
    for(int i=51;i>=0;i--){
        for(int j=i-1;j>=0;j--){
            if(p[i]>>j&1) p[i]^=p[j];
        }
    }
    for(int i=0;i<=51;i++){
        if(p[i]){
            d.push_back(p[i]);
        }
    }
}
```

```
//flag表示数组里面有没有0
int querykth(int k){
    if(flag){
        if(k==1) return 0;
        k--;
    }
    int ans=0;
    for(int i=d.size()-1;i>=0;i--){
        if(k>>i&1) ans^=d[i];
    }
    return ans;
}
```

扩展欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \text{if } \gcd(a, m) = 1, \\ a^b, & \text{if } \gcd(a, m) \neq 1 \text{ and } b < \varphi(m), \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \text{if } \gcd(a, m) \neq 1 \text{ and } b \geq \varphi(m). \end{cases}$$

扩展欧几里得

```
// ax+by=gcd(a,b)
// mod非素数，扩展欧几里得
// ax+by = gcd(a,b)
// 返回 d=gcd(a,b); 和对应于等式ax+by=d中的x,y
int exgcd(int a,int b,int &x,int &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    int x1,y1;
    int p=exgcd(b,a%b,x1,y1);
    x=y1;
    y=(x1-a/b*y1);
    return p;
}
// lzy
11 exgcd(11 a, 11 b, 11& x, 11& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    11 g = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
}
```

中国剩余定理 CRT

求解如下形式的一元线性同余方程组 (其中 n_1, n_2, \dots, n_k 两两互质)

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

```
int CRT(vector<int> &a, vector<int> &r) {
    int n=1,ans=0;
    for(int i=0;i<r.size();i++) n=n*r[i];
    for(int i=0;i<a.size();i++){
        int m=n/r[i],b,y;
        exgcd(m,r[i],b,y);
        ans=(ans+a[i]*m*b%n)%n;
    }
    return (ans%n+n)%n;
```

```

}

// lzy: 优先使用i128, 比龟速乘更快
static inline ll mul(ll a, ll b, const ll mod) {
    return (i128)a * b % mod;
}

ll CRT(vector<int>& a, vector<int>& r) {
    ll n = 1, ans = 0;
    for (int i = 0; i < r.size(); i++) n = n * r[i];
    for (int i = 0; i < a.size(); i++) {
        ll m = n / r[i], b, y;
        exgcd(m, r[i], b, y);
        ans = (ans + mul(mul(a[i], m, n), b, n)) % n;
    }
    return (ans % n + n) % n;
}

```

扩展CRT

处理模数不互质的情况

```

#include<bits/stdc++.h>
using namespace std;
#define int __int128
int exgcd(int a,int b,int &x,int &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    int x1,y1;
    int p=exgcd(b,a%b,x1,y1);
    x=y1;
    y=(x1-a/b*y1);
    return p;
}
int ExCRT(vector<int> &a,vector<int> &r){
    int a1=0,r1=1;
    for(int i=0;i<a.size();i++){
        int a2=(a[i]%r[i]+r[i])%r[i],r2=r[i];
        int x,y;
        int g=exgcd(r1,r2,x,y);
        if((a2-a1)%g!=0){
            return -1;
        }
        x+=(a1-a2)/g;
        //y+=(a2-a1)/g;
        a1=a1-r1*x;
        //y+=(a2-a1)/g;
        r1=r2/g*r1;
        a1=(a1%r1+r1)%r1;
        //cerr<<a1<<" "<<r1<<" "<<g<<"\n";
    }
    return (a1%r1+r1)%r1;
}
void get(int &x){
    x=0;
}

```

```

char ch=getchar();
while(ch<'0' || ch>'9'){
    ch=getchar();
}
while(ch>='0'&&ch<='9'){
    x=x*10+ch-'0';
    ch=getchar();
}
void print(__int128 x){
    if(x==0){
        putchar('0');
        return;
    }
    stack<char> st;
    while(x){
        st.push(x%10+'0');
        x/=10;
    }
    while(!st.empty()){
        putchar(st.top());
        st.pop();
    }
}
void solve(){
    int k;
    get(k);
    vector<int> a(k),b(k);
    for(int i=0;i<k;i++){
        get(a[i]);
        get(b[i]);
    }
    print(EXCRT(b,a));
}
signed main(){
    int t=1;
    while(t--) solve();
    return 0;
}

```

切比雪夫距离与曼哈顿距离之间的转化

曼哈顿意义下的坐标 (x,y) , 可以转化成切比雪夫意义下的 $(x+y, x-y)$

切比雪夫意义下的坐标 (x,y) , 可以转化成曼哈顿意义下的坐标 $(\frac{x+y}{2}, \frac{x-y}{2})$

积性函数

定义在正整数域上的函数 $f(x)$ 对于任意的 $\gcd(a,b)=1$ 均满足 $f(ab)=f(a)*f(b)$

可以用欧拉筛线性求值

常见积性函数

$\mu(n)$: 莫比乌斯函数

$\varphi(n)$: 欧拉函数

$d(n)$: 一个数n的约数个数

$\sigma(n)$: 一个数n的约数和

$f(x)=x^k(k \in \mathbb{N})$: 这个玩意儿也是积性函数

多项式

快速傅里叶变换 FFT

- 实数 (`double`) 系数的线性卷积/多项式乘法: `Poly C = A * B;`, 默认得到浮点数系数的多项式;
- 整数系数: 用 `llround()` (返回整型), 对负数也正确四舍五入, 避免 `(int)(x+0.5)` 的坑;
- 默认使用 `std::complex` 作为复数类型;

```
template <class T, template <class G> class Complex>
class Polynomial : public std::vector<T> {
    using Comp = Complex<T>;
    // C++17及以上, 可以用inline static, 省掉类外初始化
    inline static std::vector<Comp> w[2] = {};// 单位根表 0: 正向FFT, 1: 逆向FFT
    inline static std::vector<int> r = {};// 比特倒序排列数组

    // 初始化单位根表和比特倒序排列数组
    static void init(int _log) {
        if (r.size() == (1u << _log)) {
            return;
        }

        int n = 1 << _log;
        r.assign(n, 0);
        // 比特倒序排列
        for (int i = 1; i < n; i++) {
            r[i] = (r[i >> 1] >> 1) | ((i & 1) << (_log - 1));
        }
    }

    w[0].assign(n, Comp());
    w[1].assign(n, Comp());

    // 单位根预处理
    const T PI = acosl(-1);
    for (int i = 0; i < n; i++) {
        auto th = PI * i / n;
        auto cth = std::cos(1.L * th);
        auto sth = std::sin(1.L * th);
        w[0][i] = Comp(cth, sth);
        w[1][i] = Comp(cth, -sth);
    }
}

// op=0: 正向FFT, op=1: 逆向FFT
static void fft(std::vector<Comp> &a, int op) {
    int n = a.size();
    init(std::lg(n));
```

```

        for (int i = 0; i < n; i++) {
            if (i < r[i]) {
                std::swap(a[i], a[r[i]]);
            }
        }
        for (int mid = 1; mid < n; mid <= 1) {
            const int d = n / mid;
            for (int R = mid < 1, j = 0; j < n; j += R) {
                for (int k = 0; k < mid; k++) {
                    Comp x = a[j + k];
                    Comp y = w[op][d * k] * a[j + mid + k];
                    a[j + k] = x + y;
                    a[j + mid + k] = x - y;
                }
            }
        }
    }

public:
    using std::vector<T>::vector; // 继承vector的构造函数

    constexpr friend Polynomial operator*(const Polynomial &a, const Polynomial
&b) {
        if (a.size() == 0 or b.size() == 0) {
            return Polynomial();
        }
        int n = a.size() + b.size() - 1;
        int _log = std::__lg(2 * n - 1);
        int s = 1 << _log;
        if (std::min(a.size(), b.size()) < 128) {
            Polynomial res(n);
            for (auto i = 0U; i < a.size(); i++) {
                for (auto j = 0U; j < b.size(); j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
            return res;
        }

        std::vector<Comp> p(s), q(s);
        for (auto i = 0U; i < a.size(); i++) {
            p[i] = Comp(a[i], 0);
        }
        for (auto i = 0U; i < b.size(); i++) {
            q[i] = Comp(b[i], 0);
        }

        fft(p, 0);
        fft(q, 0);
        for (int i = 0; i < s; i++) {
            p[i] = p[i] * q[i]; // 点值乘法
        }
        fft(p, 1);

        Polynomial res(n);
        for (int i = 0; i < n; i++) {

```

```

        res[i] = p[i].real() / s; // 默认浮点数
        // 卷积后调用 llround() 得到整型
    }
    return res;
}

friend std::istream &operator>>(std::istream &is, Polynomial &a) {
    int n = a.size();
    for (int i = 0; i < n; i++) {
        is >> a[i];
    }
    return is;
}
friend std::ostream &operator<<(std::ostream &os, const Polynomial &a) {
    int n = a.size();
    for (int i = 0; i < n; i++) {
        os << a[i] << " \n"[i == n - 1];
    }
    return os;
}
};

using Poly = Polynomial<double, std::complex>;

```

快速数论变换 NTT

- NTT是整数域上的快速傅里叶变换，用于高效求解模意义下的多项式卷积。
- 要求模数 P 是形如 $k \cdot 2^n + 1$ 的质数（如 $P = 998244353 = 119 \cdot 2^{23} + 1$ ），且有原根 g ；如果模数为 $10^9 + 7$ ，需要三模数 NTT + CRT，在3个NTT友好质数下分别做NTT卷积，再用Garner/CRT合并回 $10^9 + 7$ 。常用3个NTT质数：
 - $P_1 = 167772161 = 5 \cdot 2^{25} + 1$
 - $P_2 = 469762049 = 7 \cdot 2^{26} + 1$
 - $P_3 = 1224736769 = 73 \cdot 2^{24} + 1$
- 单位根： $w = g^{(P-1)/n} \pmod{P}$ 。

```

template <class T>
struct Polynomial : public std::vector<T> {
    #define self (*this)
    inline static std::vector<T> w = {};
    static constexpr auto P = T::getMod();

    // 初始化单位根
    static void initw(int r) {
        if (static_cast<int>(w.size()) >= r) {
            return;
        }

        w.assign(r, 0);
        w[r >> 1] = 1;
        T s = T(3).pow((P - 1) / r); // 原根 3
        for (int i = r / 2 + 1; i < r; i++) {
            w[i] = w[i - 1] * s;
        }
    }
}

```

```

        for (int i = r / 2 - 1; i > 0; i--) {
            w[i] = w[i * 2];
        }
    }

// 正变换
friend void dft(Polynomial &a) {
    const int n = a.size();
    assert((n & (n - 1)) == 0);
    initw(n);

    for (int k = n >> 1; k; k >>= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; j++) {
                T v = a[i + j + k];
                a[i + j + k] = (a[i + j] - v) * w[k + j];
                a[i + j] = a[i + j] + v;
            }
        }
    }
}

// 逆变换
friend void idft(Polynomial &a) {
    const int n = a.size();
    assert((n & (n - 1)) == 0);
    initw(n);

    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; j++) {
                T x = a[i + j];
                T y = a[i + j + k] * w[j + k];
                a[i + j + k] = x - y;
                a[i + j] = x + y;
            }
        }
    }
}

a *= P - (P - 1) / n;
std::reverse(a.begin() + 1, a.end());
}

public:
    using std::vector<T>::vector;

// 多项式截断 mod x^k, 保留前k项系数
Polynomial mod(int k) const {
    Polynomial p = self;
    p.resize(k);
    return p;
}

// 多项式加法
friend Polynomial operator+(const Polynomial &a, const Polynomial &b) {
    Polynomial p(std::max(a.size(), b.size()));

```

```

        for (auto i = 0U; i < a.size(); i++) p[i] += a[i];
        for (auto i = 0U; i < b.size(); i++) p[i] += b[i];
        return p;
    }

    // 多项式减法
    friend Polynomial operator-(const Polynomial &a, const Polynomial &b) {
        Polynomial p(std::max(a.size(), b.size()));
        for (auto i = 0U; i < a.size(); i++) p[i] += a[i];
        for (auto i = 0U; i < b.size(); i++) p[i] -= b[i];
        return p;
    }

    // 多项式取负 (系数取反)
    friend Polynomial operator-(const Polynomial &a) {
        int n = a.size();
        Polynomial p(n);
        for (int i = 0; i < n; i++) p[i] = -a[i];
        return p;
    }

    // 多项式乘法 a * f(x)
    friend Polynomial operator*(T a, Polynomial b) {
        for (auto i = 0U; i < b.size(); i++) b[i] *= a;
        return b;
    }

    // 多项式乘法 f(x) * b
    friend Polynomial operator*(Polynomial a, T b) {
        for (auto i = 0U; i < a.size(); i++) a[i] *= b;
        return a;
    }

    // 多项式卷积 f(x) * g(x)
    friend Polynomial operator*(const Polynomial &a, const Polynomial &b) {
        if (a.size() == 0 or b.size() == 0) {
            return Polynomial();
        }

        int n = a.size() + b.size() - 1;
        int s = 1 << std::__lg(2 * n - 1);
        if (((P - 1) & (s - 1)) != 0 or std::min(a.size(), b.size()) < 128) {
            Polynomial p(n);
            for (auto i = 0U; i < a.size(); i++) {
                for (auto j = 0U; j < b.size(); j++) {
                    p[i + j] += a[i] * b[j];
                }
            }

            return p;
        }

        Polynomial f = a.mod(s);
        Polynomial g = b.mod(s);
        dft(f), dft(g);
        for (int i = 0; i < s; i++) {
            f[i] *= g[i];
        }
    }
}

```

```

    }

    idft(f);
    return f.mod(n);
}

friend Polynomial operator/(Polynomial a, T b) {
    b = b.inv();
    for (auto i = 0U; i < a.size(); i++) a[i] *= b;
    return a;
}

// // 多项式除法 f(x) / g(x), 返回商和余数 需要debug
// pair<Polynomial, Polynomial> divmod(const Polynomial &g) const {
//     int n = size(), m = g.size();
//     if (n < m) return {Polynomial{0}, *this};

//     Poly fr = *this;
//     reverse(fr.begin(), fr.end());
//     Poly gr = g;
//     reverse(gr.begin(), gr.end());

//     Poly qrev = (fr * gr.inv(n - m + 1)).mod(n - m + 1);
//     reverse(qrev.begin(), qrev.end());

//     Poly r = (*this - qrev * g).mod(m);
//     return {qrev, r};
// }

// f(x) * x^k, 多项式整体向高次移动k位
Polynomial mulxk(int k) const {
    assert(k >= 0);
    Polynomial b = self;
    b.insert(b.begin(), k, 0);
    return b;
}

// f(x) / x^k, 多项式整体向低次移动k位
Polynomial divxk(int k) const {
    assert(k >= 0);
    if (static_cast<int>(self.size()) <= k) {
        return Polynomial{};
    }
    return Polynomial(self.begin() + k, self.end());
}

// 多项式求值 f(x)
T at(T x) const {
    T ans = T{0};
    for (int i = static_cast<int>(self.size()) - 1; i >= 0; i--) {
        ans = ans * x + self[i];
    }
    return ans;
}

Polynomial &operator+=(Polynomial b) { return self = self + b; }
Polynomial &operator-=(Polynomial b) { return self = self - b; }

```

```

Polynomial &operator*(Polynomial b) { return self = self * b; }
Polynomial &operator*(T b) { return self = self * b; }
Polynomial &operator/(T b) { return self = self / b; }

// 求导
Polynomial deriv() const {
    int n = self.size();
    if (n <= 1) return Polynomial();

    Polynomial p(n - 1);
    for (int i = 1; i < n; i++) {
        p[i - 1] = i * self[i];
    }
    return p;
}

// 积分
Polynomial integr() const {
    int n = self.size();
    Polynomial p(n + 1);
    std::vector<T> _inv(n + 1);
    _inv[1] = 1;
    for (int i = 2; i <= n; i++) {
        _inv[i] = _inv[P % i] * (P - P / i);
    }
    for (int i = 0; i < n; ++i) {
        p[i + 1] = self[i] * _inv[i + 1];
    }
    return p;
}

// 多项式模  $x^m$  的逆元
// assert(self[0] != 0);
Polynomial inv(int m = -1) const {
    const int n = self.size();
    m = m < 0 ? n : m;
    Polynomial p = Polynomial{self.at(0).inv()};
    p.reserve(4 * m);
    for (int k = 2; k / 2 < m; k <= 1) {
        Polynomial q = Polynomial(self.begin(), self.begin() + std::min(k,
n)).mod(2 * k);
        p.resize(2 * k);
        dft(q), dft(p);
        for (int i = 0; i < 2 * k; i++) {
            p[i] = p[i] * (2 - p[i] * q[i]);
        }
        idft(p);
        p.resize(k);
    }
    return p.mod(m);
}

Polynomial ln(int m = -1) const {
    m = m < 0 ? self.size() : m;
    return (deriv() * inv(m)).integr().mod(m);
}

```

```

// exp(f(x)): 1 + f(x) + f(x)^2/2! + f(x)^3/3! + ...
Polynomial exp(int m = -1) const {
    m = m < 0 ? self.size() : m;
    Polynomial p{1};
    int k = 1;
    while (k < m) {
        k <= 1;
        p = (p * (Polynomial{1} - p.ln(k) + mod(k))).mod(k);
    }
    return p.mod(m);
}

// 多项式幂运算 f(x)^k, k >= 0
Polynomial pow(long long k, int m = -1) const {
    m = m < 0 ? self.size() : m;
    assert(0 <= k);
    k = k % P;
    if (0 <= k and k < 6) {
        Polynomial p = self;
        Polynomial ans{1};
        for (; k; k /= 2) {
            if (k & 1) {
                ans = (ans * p).mod(m);
            }
            p = (p * p).mod(m);
        }
        return ans.mod(m);
    }

    unsigned int i = 0;
    while (i < self.size() and self[i] == T{}) {
        i += 1;
    }
    if (i == self.size() or __int128(k) * i >= m) {
        return Polynomial(m, T{});
    }
    T v = self[i];
    Polynomial f = divxk(i) / v;
    return (f.ln(m - i * k) * k).exp(m - i * k).mulxk(i * k) * v.pow(k);
}

Polynomial sqrt(int m = -1) const {
    m = m < 0 ? self.size() : m;
    Polynomial p{1};
    int k = 1;
    const T INV2 = T(1) / 2;
    while (k < m) {
        k <= 1;
        p = (p + (mod(k) * p.inv(k)).mod(k)) * INV2;
    }
    return p.mod(m);
}

friend std::istream &operator>>(std::istream &is, Polynomial &a) {
    int n = a.size();

```

```

        for (int i = 0; i < n; i++) {
            is >> a[i];
        }
        return is;
    }

    friend std::ostream &operator<<(std::ostream &os, const Polynomial &a) {
        int n = a.size();
        for (int i = 0; i < n; i++) {
            os << a[i] << " \n"[i == n - 1];
        }
        return os;
    }
    #undef self
};

template <class T, T P>
struct ModInt {
    // static_assert(std::is_integral_v<T>, "ModInt requires integral type.");
    T x;

    ModInt(ll x = 0) : x(norm(x % P)) {}

    // 规范化 x 到 [0, P)
    static constexpr T norm(T x) {
        if (x < 0) return x + P;
        if (x >= P) return x - P;
        return x;
        // return (x < 0 ? x + getMod() : (x >= getMod() ? x - getMod() : x));
    }

    static constexpr T getMod() { return P; }

    // 模乘(适配 int 和 ll)
    static constexpr int mul(int a, int b, int p) {
        return 1LL * a * b % p;
    }

    static constexpr ll mul(ll a, ll b, ll p) {
        ll res = a * b - ll(1.L * a * b / p) * p;
        res %= p;
        if (res < 0) res += p;
        return res;
    }

    explicit constexpr operator T() const { return x; }

    constexpr ModInt operator-() const { return ModInt(-x); }
    constexpr ModInt pow(ll m) const {
        ModInt a = *this;
        ModInt res = 1;

        if (m < 0) {
            a = a.inv();
            m = -m;
        }

```

```

        while (m) {
            if (m & 1) res *= a;
            a *= a;
            m >>= 1;
        }
        return res;
    }

constexpr ModInt inv() const {
    assert(x != 0);      // 0 没有逆元
    return pow(P - 2);   // 费马小定理 P 为质数
}

constexpr ModInt &operator+=(const ModInt &b) {
    x = norm(x + b.x);
    return *this;
}

constexpr ModInt &operator-=(const ModInt &b) {
    x = norm(x - b.x);
    return *this;
}

constexpr ModInt &operator*=(const ModInt &b) {
    x = mul(x, b.x, P);
    return *this;
}

constexpr ModInt &operator/=(const ModInt &b) {
    return *this *= b.inv();
}

// 运算符重载
friend constexpr ModInt operator+(ModInt a, const ModInt &b) { return a += b; }
friend constexpr ModInt operator-(ModInt a, const ModInt &b) { return a -= b; }
friend constexpr ModInt operator*(ModInt a, const ModInt &b) { return a *= b; }
friend constexpr ModInt operator/(ModInt a, const ModInt &b) { return a /= b; }
friend constexpr bool operator==(const ModInt &a, const ModInt &b) { return a.x == b.x; }
friend constexpr bool operator!=(const ModInt &a, const ModInt &b) { return a.x != b.x; }

friend std::istream &operator>>(std::istream &is, ModInt &a) {
    ll v;
    is >> v;
    a = ModInt(v);
    return is;
}

friend std::ostream &operator<<(std::ostream &os, const ModInt &a) {
    return os << a.x;
}
};

```

```
using Z = ModInt<11, 998244353>;
using Poly = Polynomial<Z>;
```

快速沃尔什变换 FWT

适用范围：（配合二进制状态压缩）

- 按位运算卷积： $C[i] = \sum_{j \oplus k=i} A[j]B[k]$, 其中 \oplus 是二元位运算的一种
- SOS DP / 子集-超集和：子集/超集 zeta & Möbius 变换，与 FWT_or / FWT_and 等价实现。
- 布尔立方体上的运算：在 $\{0, 1\}^m$ 上的线性变换、计数、期望、异或型 DP 等。
- 与 NTT 不同：FWT 的“长度”是按位数 n 的 2^N , 服务于位掩码/集合运算；不是多项式的循环/线性卷积。

复杂度：

- 单次变换 $O(N \log N)$, 其中 $N = 2^n$
- 卷积（正变换 $A, B \rightarrow$ 按位乘 \rightarrow 逆变换）： $O(n \log n)$ (三个变换 + $O(n)$ 点乘)
- 实战规模：常见 $n \leq 20$ ($N \leq 2^{20}$) 较稳； $n = 22$ 需注意常数与内存。

模板注意事项：

- 长度必须是 $N = 2^n$, 否则需要补 0 到最近的 2^k ;
- XOR 逆变换最后统一乘 $1/N$, 逆元实现；OR / AND 不需要归一化。

```
template <int MOD>
struct FWT {
    enum class Type { OR, AND, XOR };

    static inline int add(int a, int b) {
        a += b;
        if (a >= MOD) a -= MOD;
        return a;
    }
    static inline int sub(int a, int b) {
        a -= b;
        if (a < 0) a += MOD;
        return a;
    }
    static inline int mul(int a, int b) {
        return 1LL * a * b % MOD;
    }
    // 只有 XOR 需要求逆元
    static inline int pow(int a, ll e) {
        ll r = 1;
        while (e) {
            if (e & 1) r = mul(r, a);
            a = mul(a, a);
            e >>= 1;
        }
        return (int)r;
    }
    // FWT 就地变换: opt = 1 为正变换, opt = -1 为逆变换
    static void transform(vector<int>& a, Type type, int opt) {
```

```

const int n = a.size();
for (int len = 1; len < n; len <= 1) {
    for (int i = 0; i < n; i += (len < 1)) {
        for (int j = 0; j < len; j++) {
            int &x = a[i + j], &y = a[i + j + len];
            if (type == Type::OR) {
                // forward: (x, y) -> (x, x+y)
                // inverse: (x, y) -> (x, y-x)
                if (opt == 1) {
                    y = add(y, x);
                } else {
                    y = sub(y, x);
                }
            } else if (type == Type::AND) {
                // forward: (x, y) -> (x+y, y)
                // inverse: (x, y) -> (x-y, y)
                if (opt == 1) {
                    x = add(x, y);
                } else {
                    x = sub(x, y);
                }
            } else if (type == Type::XOR) {
                // (x, y) -> (x+y, x-y) ; 逆变换后再整体乘 1/n
                int u = x, v = y;
                x = add(u, v);
                y = sub(u, v);
            }
        }
    }
}
// xor 逆变换，归一化，所有元素乘 1/n = (1/2)^{log2 n}
if (type == Type::XOR && opt == -1) {
    const int inv2 = (MOD + 1) / 2;
    int k = __builtin_ctz(n); // log2(n)
    int invn = pow(inv2, k); // (1/2)^k
    for (int& x : a) {
        x = mul(x, invn);
    }
}

// 点乘
static void pointwise(vector<int>& a, const vector<int>& b) {
    const int n = a.size();
    for (int i = 0; i < n; ++i) {
        a[i] = mul(a[i], b[i]);
    }
}

// 卷积 (OR / AND / XOR)
static vector<int> convolution(vector<int> a, vector<int> b, Type type) {
    // 需要 A、B 长度相等且为 2 的幂；若需要，也可在外部补零到 2^k
    assert(a.size() == b.size());
    assert((a.size() & (a.size() - 1)) == 0); // 检查是否为 2 的幂

    transform(a, type, 1); // 正变换
}

```

```

        transform(b, type, 1); // 正变换
        pointwise(a, b); // 点乘
        transform(a, type, -1); // 逆变换
        return a;
    }
};

using fwt = FWT<998244353>

int main() {
    int n; cin >> n;
    int N = 1 << n; // 需要保证长度是2的幂次，否则补0
    vector<int> a(N), b(N);
    for (int i = 0; i < N; i++) cin >> a[i];
    for (int i = 0; i < N; i++) cin >> b[i];

    auto OR = fwt::convolution(a, b, fwt::Type::OR);
    auto AND = fwt::convolution(a, b, fwt::Type::AND);
    auto XOR = fwt::convolution(a, b, fwt::Type::XOR);

    for (int i = 0; i < N; i++) cout << OR[i] << " \n"[i == N - 1];
    //...
}

```

求n!质因数p的个数

1-n每个数都可以贡献质因数p

p的倍数贡献一个， p^2 的倍数继续额外贡献一个

$$cnt = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \dots + \left\lfloor \frac{n}{p^k} \right\rfloor$$

二次剩余（模意义下开根）

$$\gcd(n, p) = 1$$

$$x^2 \equiv n \pmod{p}$$

若存在x，则n为模p的二次剩余（p是奇素数），否则就是二次非剩余

Euler判定

a是p的二次剩余当且仅当

$$n^{\frac{p-1}{2}} \equiv 1 \pmod{p}$$

是p的二次非剩余当且仅当

$$n^{\frac{p-1}{2}} \equiv -1 \pmod{p}$$

如果

%4=3

$n^{(p+1)/4} \pmod{p}$ 是一个解

这个方程只有两个解且它们互为相反数。一个二次剩余对应一对模意义下不同的相反数 (p 为奇素数所以相反数的奇偶性不同)。因为模 p 意义下可以找到 $\frac{p-1}{2}$ 对非零的相反数, 所以在模 p 意义下共有 $\frac{p-1}{2}$ 个二次剩余。

Cipolla算法

先找到一个 a 使得 $a^2 - n$ 是二次非剩余, 令 $w^2 \equiv a^2 - n \pmod{p}$, 则 $(a + w)^{\frac{p+1}{2}}$ 即是方程的一个解, 其相反数则是另一个解。

在 $[0,p)$ 中随机一个a, 并用欧拉准则判断 $a^2 - n$ 是否是二次非剩余, 因为二次非剩余有 $p/2$ 个, 所以期望两次找到。

```

int quickpow(int x,int y,int mod){
    int ans=1,base=x;
    x%=mod;
    if(x==0) return 0;
    while(y){
        if(y&1) ans=ans*base%mod;
        base=base*base%mod;
        y>>=1;
    }
    return ans;
}
pair<int,int> Cipolla(int n,int mod){
    if(n==0) return make_pair(0,0);
    n%=mod;
    if(quickpow(n,(mod-1)/2,mod)==mod-1) return {-1,-1}; // 二次非剩余, 无解
    int a=0,w;
    while(1){
        a=rand()%mod;
        w=((a*a%mod-n)%mod+mod)%mod;
        if(quickpow(w,(mod-1)/2,mod)==mod-1) break;
    }
    auto mulcomplex=[&](pair<int,int> x,pair<int,int> y,int mod){
        return
        make_pair(((x.first*y.first%mod+w*x.second%mod*y.second%mod)%mod+mod)%mod,
        (x.first*y.second%mod+x.second*y.first%mod)%mod);
    };
    auto quickpowcomplex=[&](pair<int,int> x,int y,int mod){
        pair<int,int> base=x,ans={1,0};
        while(y){
            if(y&1) ans=mulcomplex(ans,base,mod);
            base=mulcomplex(base,base,mod);
            y>>=1;
        }
        return ans;
    };
    pair<int,int> result;
    result.first=quickpowcomplex(make_pair(a,1),(mod+1)/2,mod).first;
    result.second=(-result.first%mod+mod)%mod;
    if(result.first>result.second) swap(result.first,result.second);
    return result; // 返回两个根并排好序
}

```

Min25筛

在 $O\left(\frac{n^{3/4}}{\log n}\right)$ 时间内求出积性函数f的前缀和，且需要满足

- $f(x)$ 在 x 为质数的时候有一个简单多项式表示
- $f(x^c)$ 在 x 为质数的时候可以快速计算

f: 原函数 (积性函数)

fpi: 新函数 (完全积性函数)，质数处与f取值相同

1. 先求出 $g[n][i]$, 即 $x \in [2, n]$ 且 (x 为质数或 x 的最小质因数 $> prime_i$) 时, 所有 fpi(x) 的和
2. $s[n][i]$ 表示, $x \in [2, n]$ 且最小质因数 $> prime_i$ 时, 所有 f(x) 的和

$$g(n, j) = g(n, j - 1) - p_j^k \left(g\left(\frac{n}{p_j}, j - 1\right) - g(p_{j-1}, j - 1) \right)$$

$$S(n, x) = g(n) - sp_x + \sum_{\substack{p_k^e \leq n \\ k > x}} f(p_k^e) \left(S\left(\frac{n}{p_k^e}, k\right) + [e \neq 1] \right)$$

即质数贡献 (可能需要对某些值进行修改) +合数贡献

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MOD=1e9+7;
int f(int ind,int x){
    x%=MOD;
    if(ind==0){
        return x;
    }else{
        return x*x%MOD;
    }
}
int fpi(int ind,int x){
    x%=MOD;
    if(ind==0){
        return x;
    }else{
        return x*x%MOD;
    }
}
int inv6=166666668,inv2=500000004;
int fpisum(int ind,int x){
    x%=MOD;
    if(ind==0){
        return (((1+x)%MOD*x%MOD*inv2%MOD-1)%MOD+MOD)%MOD;
    }else{
        return ((x%MOD*(x+1)%MOD*(2*x%MOD+1)%MOD*inv6%MOD-1)%MOD+MOD)%MOD;
    }
}
template<int items>
struct Min25{
    struct Eulersieve{
        vector<int> prime;
        vector<int> v;
    }
```

```

vector<array<int,items>> sp;
int n;
Eulersieve(int n,const int MOD):v(n+1){
    this->n=n;
    for(int i=2;i<=n;i++){
        if(v[i]==0){
            prime.push_back(i);
            v[i]=i;
            sp.emplace_back();
            for(int j=0;j<items;j++){
                if(sp.size()>=2) sp.back()[j]=sp[sp.size()-2][j];
                sp.back()[j]=(sp.back()[j]+fpi(j,i))%MOD;
            }
        }
        for(int &p:prime){
            if(i*p>n) break;
            v[i*p]=p;
            if(i%p==0) break;
        }
    }
}
int n,sqr;
Eulersieve eulersieve;
vector<int> ind1,ind2,w;
vector<array<int,items>> g;
int result=0;
//f(i,x) 多项式拆成若干个单项式, 第i个单项式(x=p^t)
//fpi(i,x) 多项式拆成若干个单项式, 和f[i]相等的完全积性函数
//fpisum(i,x) 多项式拆成若干个单项式, fpi[i](x)前缀和(不含fpi[i][1])
//sign[i]第i个单项式的系数(符号)
Min25(int n,array<int,items>
&sign):n(n),sqr(sqrt(n)),ind1(sqr+10),ind2(sqr+10),eulersieve(sqr,MOD){
    w.reserve(2*sqr+10);
    g.reserve(2*sqr+10);
    ind1.reserve(sqr+10);
    ind2.reserve(sqr+10);
    for(int i=1;i<=n;){
        int j=n/(n/i);
        w.emplace_back(n/i);
        g.emplace_back();
        for(int j=0;j<items;j++){
            g.back()[j]=fpisum(j,w.back());
        }
        if(n/i<=sqr) ind1[n/i]=w.size()-1;
        else ind2[n/(n/i)]=w.size()-1;
        i=j+1;
    }
    for(int t=0;t<eulersieve.prime.size();t++){
        int p=eulersieve.prime[t];
        for(int i=0;i<w.size()&&p<=w[i]/p;i++){
            int k=w[i]/p<=sqr?ind1[w[i]/p]:ind2[n/(w[i]/p)];
            for(int j=0;j<items;j++){
                g[i][j]=(fpi(j,p)*(g[k][j])%MOD-(t-1>=0?eulersieve.sp[t-1]
[j]:0)%MOD)%MOD;
                g[i][j]=(g[i][j]%MOD+MOD)%MOD;
            }
        }
    }
}

```

```

        }
    }
}

auto s=[&](auto &&self,int x,int y){
    if(y>=1&&eulersieve.prime[y-1]>=x) return 011;
    int k=x<=sqr?ind1[x]:ind2[n/x];
    int ans=0;
    for(int i=0;i<items;i++){
        int t=y>0?eulersieve.sp[y-1][i]:0;
        ans=(ans+sign[i]*(g[k][i]-t)%MOD)%MOD;
    }
    for(int i=y+1;i<=eulersieve.prime.size()&&eulersieve.prime[i-1]
<=x/eulersieve.prime[i-1];i++){
        int pe=eulersieve.prime[i-1];
        for(int e=1;pe<=x;e++,pe*=eulersieve.prime[i-1]){
            int xx=pe%MOD;
            int sum=0;
            for(int j=0;j<items;j++){
                sum=(sum+sign[j]*f(j,pe))%MOD;
            }
            ans=(ans+sum*(self(self,x/pe,i)+(e!=1))%MOD)%MOD;
            ans=(ans+MOD)%MOD;
        }
    }
    return ans;
};

result=((s(s,n,0)+1)%MOD+MOD)%MOD;
}

int get(){
    return result;
}

void solve(){
    int n;
    cin>>n;
    array<int,2> sign={-1,1};
    Min25<2> min25(n,sign);
    cout<<min25.get()<<"\n";
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MOD=1e9+7;
int f(int x,int t){
    return (x^t)%MOD;
}
int fpi(int ind,int x){

```

```

x%=MOD;
if(ind==0){
    return x;
}else{
    return 1;
}
int inv2=500000004;
int fpisum(int ind,int x){
    x%=MOD;
    if(ind==0){
        return (((1+x)%MOD*x%MOD*inv2%MOD-1)%MOD+MOD)%MOD;
    }else{
        return ((x-1)%MOD+MOD)%MOD;
    }
}
template<int items>
struct Min25{
    struct Eulersieve{
        vector<int> prime;
        vector<int> v;
        vector<array<int,items>> sp;
        int n;
        Eulersieve(int n,const int MOD):v(n+1){
            this->n=n;
            for(int i=2;i<=n;i++){
                if(v[i]==0){
                    prime.push_back(i);
                    v[i]=i;
                    sp.emplace_back();
                    for(int j=0;j<items;j++){
                        if(sp.size()>=2) sp.back()[j]=sp[sp.size()-2][j];
                        sp.back()[j]=(sp.back()[j]+fpisum(j,i))%MOD;
                    }
                }
                for(int &p:prime){
                    if(i*p>n) break;
                    v[i*p]=p;
                    if(i%p==0) break;
                }
            }
        }
    };
    int n,sqr;
    Eulersieve eulersieve;
    vector<int> ind1,ind2,w;
    vector<array<int,items>> g;
    int result=0;
    //f[i](x) 多项式拆成若干个单项式, 第i个单项式(x=p^t)
    //fpisum[i](x) 多项式拆成若干个单项式, 和f[i](x)相等的完全积性函数
    //fpisum[i](x) 多项式拆成若干个单项式, fpisum[i](x)前缀和(不含fpisum[i][1])
    //sign[i]第i个单项式的系数(符号)
    Min25(int n,array<int,items>
&sign):n(n),sqr(sqrt(n)),ind1(sqr+10),ind2(sqr+10),eulersieve(sqr,MOD){
        w.reserve(2*sqr+10);
        g.reserve(2*sqr+10);
    }
}

```

```

        ind1.reserve(sqr+10);
        ind2.reserve(sqr+10);
        for(int i=1;i<=n;){
            int j=n/(n/i);
            w.emplace_back(n/i);
            g.emplace_back();
            for(int j=0;j<items;j++){
                g.back()[j]=fpisum(j,w.back());
            }
            if(n/i<=sqr) ind1[n/i]=w.size()-1;
            else ind2[n/(n/i)]=w.size()-1;
            i=j+1;
        }

        for(int t=0;t<eulersieve.prime.size();t++){
            int p=eulersieve.prime[t];
            for(int i=0;i<w.size()&&p<=w[i]/p;i++){
                int k=w[i]/p<=sqr?ind1[w[i]/p]:ind2[n/(w[i]/p)];
                for(int j=0;j<items;j++){
                    g[i][j]=(fp(i,j,p)*(g[k][j]%MOD-(t-1>=0?eulersieve.sp[t-1]
[j]:011))%MOD)%MOD;
                    g[i][j]=(g[i][j]%MOD+MOD)%MOD;
                }
            }
        }
        auto s=[&](auto &&self,int x,int y){
            if(y>=1&&eulersieve.prime[y-1]>=x) return 011;
            int k=x<=sqr?ind1[x]:ind2[n/x];
            int ans=0;
            for(int i=0;i<items;i++){
                int t=y>0?eulersieve.sp[y-1][i]:0;
                ans=(ans+sign[i]*(g[k][i]-t)%MOD)%MOD;
            }
            if(x>=2&&y==0){
                ans=(ans+2)%MOD;
            }
            for(int i=y+1;i<=eulersieve.prime.size()&&eulersieve.prime[i-1]
<=x/eulersieve.prime[i-1];i++){
                int pe=eulersieve.prime[i-1];
                for(int e=1;pe<=x;e++,pe*=eulersieve.prime[i-1]){
                    int xx=pe%MOD;
                    int sum=f(eulersieve.prime[i-1],e);
                    ans=(ans+sum*(self(self,x/pe,i)+(e!=1))%MOD)%MOD;
                    ans=(ans+MOD)%MOD;
                }
            }
            return ans;
        };
        result=((s(s,n,0)+1)%MOD+MOD)%MOD;
    }
    int get(){
        return result;
    }
};

void solve(){
    int n;

```

```

cin>>n;
if(n==1){
    cout<<1<<"\n";
    return;
}else if(n==2){
    cout<<4<<"\n";
    return;
}
array<int,2> sign={1,-1};
Min25<2> min25(n,sign);
cout<<min25.get()<<"\n";
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

离线算法

莫队

树上莫队

分块大小为 $\sqrt{2n}$ 最优

题目大意：给出一棵树，树的每个节点有一个字母，有m个询问，每个询问给出两节点的编号u, v，回答两个节点之间的简单路径所包含的字母是否可以通过重新排列组合成回文串（所有字母都要用上）。

Format:

Input: 第一行输出n, m, n代表树的节点数, m代表询问的次数, 第二行输入一个字符串（均为小写字母），第i个字母表示第i个节点的字母是什么，接下来n-1行每行给出u, v表示一条边。接下来m行每行给出一堆节点编号u, v代表一次询问

Out: 如果可以组成回文串，输出yes，否则输出no

Samples:

```

5 5
abaac
1 2
2 3
2 4
4 5
1 3
2 4
1 5
1 4
5 5

```

```
yes
no
no
yes
yes
```

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
struct edge{
    int x,y,id;
    bool needLCA;
};
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int n,m;
    string s;
    cin>>n>>m;
    cin>>s;
    s=" "+s;
    vector<vector<int>> v(n+1);
    for(int i=1;i<=n-1;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    vector<int> dep(n+1,0),cnt,in(n+1),out(n+1),dfscnt(2*n+5);
    vector<vector<int>> fa(n+1,vector<int>(21));
    function<void(int,int)> dfs=[&](int x,int father){
        dep[x]=dep[father]+1;
        fa[x][0]=father;
        for(int i=1;i<=20;i++){
            fa[x][i]=fa[fa[x][i-1]][i-1];
        }
        in[x]=cnt.size();
        dfscnt[in[x]]=x;
        cnt.push_back(x);
        for(int &p:v[x]){
            if(p==father) continue;
            dfs(p,x);
        }
        out[x]=cnt.size();
        dfscnt[out[x]]=x;
        cnt.push_back(x);
    };
    function<int(int,int)> LCA=[&](int x,int y){
        if(dep[x]<dep[y]) swap(x,y);
        for(int i=20;i>=0;i--){
            if(dep[fa[x][i]]>=dep[y]) x=fa[x][i];
        }
        if(x==y) return x;
        for(int i=20;i>=0;i--){
            if(fa[x][i]!=fa[y][i]){
                x=fa[x][i];
            }
        }
    };
}
```

```

        y=fa[y][i];
    }
}
return fa[x][0];
};

dfs(1,0);
vector<edge> query(m);
for(int i=0;i<m;i++){
    cin>>query[i].x>>query[i].y;
    query[i].id=i;
    int t=LCA(query[i].x,query[i].y);
    if(t==query[i].x||t==query[i].y){
        query[i].needLCA=0;
        query[i].x=in[query[i].x];
        query[i].y=in[query[i].y];
        if(query[i].x>query[i].y) swap(query[i].x,query[i].y);
    }else{
        query[i].needLCA=1;
        if(out[query[i].x]>in[query[i].y]) swap(query[i].x,query[i].y);
        query[i].x=out[query[i].x];
        query[i].y=in[query[i].y];
    }
}
const int blocksize=sqrt(2*n);
vector<int> chcnt(26,0);
sort(query.begin(),query.end(),[&](edge &a,edge &b){
    if(a.x/Blocksize!=b.x/Blocksize) return a.x/Blocksize<b.x/Blocksize;
    else if(a.x/Blocksize%2==0) return a.y<b.y;
    else return a.y>b.y;
});
int l=1,r=0;
vector<bool> use(n+1,0);
auto upd=[&](int x){
    use[dfscnt[x]]=!use[dfscnt[x]];
    if(use[dfscnt[x]]) chcnt[s[dfscnt[x]]-'a']++;
    else chcnt[s[dfscnt[x]]-'a']--;
};
vector<bool> ans(m);
auto getans=[&](){
    int a=0,b=0;
    for(int i=0;i<26;i++){
        if(chcnt[i]==0) continue;
        if(chcnt[i]%2==0) a++;
        else b++;
    }
    return b<=1;
};
for(int i=0;i<m;i++){
    while(l>query[i].x){
        upd(--l);
    }
    while(r<query[i].y){
        upd(++r);
    }
    while(l<query[i].x){
        upd(l++);
    }
}

```

```

    }
    while(r>query[i].y){
        upd(r--);
    }
    if(query[i].needLCA){
        int t=LCA(dfscnt[query[i].x],dfscnt[query[i].y]);
        upd(in[t]);
        ans[query[i].id]=getans();
        upd(in[t]);
    }else ans[query[i].id]=getans();
}
for(auto p:ans){
    cout<<(p?"yes":"no")<<"\n";
}
}

```

回滚莫队&不删除莫队

只增加或者只删除，其中分块大小取 n/\sqrt{m} 最优， n 为数组长度， m 为询问次数

为了防止分块大小等于0，可以把分块大小取为 $\max(1,n/\sqrt{m})$ ；

给定一个序列，多次询问一段区间 $[l, r]$ ，求区间中相同的数的最远间隔距离
 序列中两个元素的间隔距离指的是两个元素下标差的绝对值
 第一行一个整数 n ，表示序列长度。第二行 n 个整数，描述这个序列。第三行一个整数 m ，表示询问个数。之后 m 行，每行两个整数 l, r 表示询问区间。输出 m 行，表示答案，如果区间内不存在两个数相同，输出0

输入：

```
8
1 6 2 2 3 3 1 6
```

```
5
```

```
1 4
```

```
2 5
```

```
2 8
```

```
5 6
```

```
1 7
```

输出：

```
1
```

```
1
```

```
6
```

```
1
```

```
6
```

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
struct edge{
    int l,r,id,blockid;
};
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int n;
    cin>>n;
    vector<int> a(n+1),num;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
}
```

```

        num.push_back(a[i]);
    }
    sort(num.begin(), num.end());
    num.erase(unique(num.begin(), num.end()), num.end());
    auto getid=[&](int x){
        return lower_bound(num.begin(), num.end(), x) - num.begin();
    };
    for(int i=1;i<=n;i++) a[i]=getid(a[i]);
    int m;
    cin>>m;
    const int blocksize=max(111,(int)(n/sqrt(m)));
    vector<edge> query(m);
    for(int i=0;i<m;i++){
        query[i].id=i;
        cin>>query[i].l>>query[i].r;
        query[i].blockid=query[i].l/blocksize;
    }
    sort(query.begin(), query.end(), [](edge a, edge b){
        if(a.blockid!=b.blockid) return a.blockid<b.blockid;
        else return a.r<b.r;
    });
    int lastblock=-1,r=-1;
    vector<int>
cntright(num.size(),-1),cntleft(num.size(),-1),cnt(num.size(),-1);
    vector<bool> cntback(num.size(),0);
    vector<int> ans(m);
    int maxn=0;
    for(int i=0;i<m;i++){
        if(query[i].l/blocksize==query[i].r/blocksize){
            int tmp=0;
            for(int j=query[i].l;j<=query[i].r;j++){
                if(cnt[a[j]]==-1) cnt[a[j]]=j;
                else tmp=max(tmp,j-cnt[a[j]]);
            }
            ans[query[i].id]=tmp;
            for(int j=query[i].l;j<=query[i].r;j++){
                cnt[a[j]]=-1;
            }
        }else{
            if(lastblock!=query[i].blockid){
                if(lastblock!=-1){
                    for(int j=(lastblock+1)*blocksize;j<=r;j++){
                        cntleft[a[j]]=cntright[a[j]]=-1;
                    }
                }
                lastblock=query[i].blockid;
                r=(lastblock+1)*blocksize-1;
                maxn=0;
            }
            while(r<query[i].r){
                ++r;
                if(cntright[a[r]]==-1){
                    cntleft[a[r]]=cntright[a[r]]=r;
                }else{
                    cntright[a[r]]=r;
                    maxn=max(maxn,r-cntleft[a[r]]));
                }
            }
        }
    }
}

```

```

        }
    }
    int maxnback=maxn;
    for(int j=(lastblock+1)*blocksize-1;j>=query[i].l;j--){
        if(cntright[a[j]]==-1){
            cntback[a[j]]=1;
            cntright[a[j]]=j;
        }else{
            maxn=max(maxn,cntright[a[j]]-j);
        }
    }
    ans[query[i].id]=maxn;
    maxn=maxnback;
    for(int j=(lastblock+1)*blocksize-1;j>=query[i].l;j--){
        if(cntback[a[j]]){
            cntright[a[j]]=-1;
            cntback[a[j]]=0;
        }
    }
}
for(int &p:ans) cout<<p<<"\n";
}

```

异或哈希

Zobrist Hash

用于棋盘状态压缩，每个位置的每个棋子状态（例如 (1,1) 位置为黑棋）使用mt19937_64赋予一个随机值，最后整个棋盘的状态等于所有棋子的异或和

图论

Cayley 凯莱定理

一个完全图不同的生成树的数量有 n^{n-2} 种

最小生成树

Kruskal

时间复杂度 $O(m \log m)$

```

struct edge{
    int x,y,k;
};
vector<edge> v(n+1);
DSU dsu(n);
int kruskal(){
    int ans=0,t=0;
    for(int i=0;i<v.size();i++){
        int fax=dsu.find(v[i].x);
        int fay=dsu.find(v[i].y);
        if(fax==fay) continue;

```

```

        merge(fax,fay);
        ans+=v[i].k;
        ++t;
        if(t==n-1){
            return ans;
        }
    }
    return -1;
}

```

判负环

SPFA判负环，一旦一个点入队次数大于等于n，即存在负环，时间复杂度O(nm)

```

const int INF=1e18;
//bfs SPFA判是否存在经过s的负环
//1表示存在负环，0表示不存在负环
auto SPFA=[&](int s){
    vector<bool> vis(n+1,0);
    vector<int> dis(n+1,INF);
    vector<int> in(n+1,0);
    queue<int> q;
    dis[s]=0;
    q.push(s);
    while(!q.empty()){
        int f=q.front();
        q.pop();
        vis[f]=0;
        for(auto &[to,w]:v[f]){
            if(dis[to]>dis[f]+w){
                dis[to]=dis[f]+w;
                if(!vis[to]){
                    if(++in[to]>n) return 1;
                    vis[to]=1;
                    q.push(to);
                }
            }
        }
    }
    return 0;
};

```

```

//存在负环返回1，否则返回0
auto check=[&](){
    bool flag=0;
    vector<int> dis(n+1,0);
    vector<bool> vis(n+1,0);
    function<int(int)> dfs=[&](int x){
        if(vis[x]){
            flag=1;
            return 1;
        }
        vis[x]=1;
        for(auto &[to,w]:v[x]){

```

```

        if(dis[to]>dis[x]+w){
            dis[to]=dis[x]+w;
            dfs(to);
            if(flag) return 1;
        }
    }
    vis[x]=0;
};

for(int i=1;i<=n;i++){
    dfs(i);
    if(flag) return 1;
}
return 0;
};

```

网络流

网络指一个特殊的有向图 $G=(V,E)$, 其与一般有向图的不同之处在于有容量和源汇点 (源点s, 汇点t)。任意节点净流量为0, 且流经该边的流量不得超过该边的容量 (边 (u,v) 的容量记作 $c(u,v)$)。定义f的流量为源点s的净流量。

最大流

使流量f尽可能大, dinic算法, 时间复杂度 $O(mn^2)$

二分图最大匹配时间复杂度 $O(m\sqrt{n})$

```

struct Flow{
    const int n;
    const int MAXN=1e18;
    vector<pair<int,int>> e;
    vector<vector<int>> g;
    vector<int> cur,dep;
    Flow(int n):n(n),g(n+1){}
    bool bfs(int s,int t){
        dep.assign(n+1,-1);
        queue<int> q;
        dep[s]=0;
        q.push(s);
        while(!q.empty()){
            const int u=q.front();
            q.pop();
            for(int i:g[u]){
                auto [v,c]=e[i];
                if(c>0&&dep[v]==-1){
                    dep[v]=dep[u]+1;
                    if(v==t) return 1;
                    q.push(v);
                }
            }
        }
        return 0;
    }
    int dfs(int u,int t,int f){
        if(u==t) return f;

```

```

int res=f;
for(int &i=cur[u];i<g[u].size();i++){
    const int j=g[u][i];
    auto [v,c]=e[j];
    if(c>0&&dep[v]==dep[u]+1){
        int out=dfs(v,t,min(res,c));
        e[j].second-=out;
        e[j^1].second+=out;
        res-=out;
        if(res==0) return f;
    }
}
return f-res;
}
void add(int u,int v,int c){
    g[u].push_back(e.size());
    e.emplace_back(v,c);
    g[v].push_back(e.size());
    e.emplace_back(u,0);
}
int work(int s,int t){
    int ans=0;
    while(bfs(s,t)){
        cur.assign(n+1,0);
        ans+=dfs(s,t,MAXN);
    }
    return ans;
}
};


```

最小割

网络G=(V,E)的一个割{S,T}， S和T是点的一个划分， $s \in S, t \in T$ ， {S,T}的容量= $\sum_{u \in S} \sum_{v \in T} c(u, v)$

最小割要找到一个割，使得容量尽可能小

根据最大流最小割定理，最大流=最小割，直接套用最大流即可

$dep[x]!=-1$ ，表示属于割边，x与s联通

最小费用最大流

在网络上对每条边(u,v)给定一个权值w(u,v)，称为费用，含义是单位流量通过(u,v)所花费的代价，对于G所有可能的最大流中总费用最小的为最小费用最大流，SSP算法， $O(nm + n^2f)$ ，其中f为网络最大流

```

struct MinCostFlow{
    const int MAXN=1e18;
    struct edge{
        int y,f,c;
        edge(int y,int f,int c):y(y),f(f),c(c){}
    };
    const int n;
    vector<edge> e;
    vector<vector<int>> g;
    vector<int> h,dis;
    vector<int> pre;
};


```

```

void spfa(int s,int t){
    queue<int> q;
    vector<bool> vis(n+1,0);
    h[s]=0,vis[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        vis[u]=0;
        for(int i:g[u]){
            const auto &[y,f,c]=e[i];
            if(f&&h[y]>h[u]+c){
                h[y]=h[u]+c;
                if(!vis[y]){
                    vis[y]=1;
                    q.push(y);
                }
            }
        }
    }
}

bool dijkstra(int s,int t){
    dis.assign(n+1,MAXN);
    pre.assign(n+1,-1);
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<>> q;
    dis[s]=0;
    q.emplace(0,s);
    while(!q.empty()){
        auto [D,x]=q.top();
        q.pop();
        if(dis[x]<D) continue;
        for(int i:g[x]){
            const auto &[y,f,c]=e[i];
            if(f&&dis[y]>D+h[x]-h[y]+c){
                dis[y]=D+h[x]-h[y]+c;
                pre[y]=i;
                q.emplace(dis[y],y);
            }
        }
    }
    return dis[t]!=MAXN;
}

MinCostFlow(int n):n(n),g(n+1){}
//x->y f:流量 c:费用
void add(int x,int y,int f,int c){
    g[x].push_back(e.size());
    e.emplace_back(y,f,c);
    g[y].push_back(e.size());
    e.emplace_back(x,0,-c);
}
pair<int,int> work(int s,int t){
    int flow=0;
    int cost=0;
    h.assign(n+1,MAXN);
    spfa(s,t);
    while(dijkstra(s,t)){

```

```

        for(int i=0;i<=n;i++) h[i]+=dis[i];
        int aug=MAXN;
        for(int i=t;i!=s;i=e[pre[i]^1].y){
            aug=min(aug,e[pre[i]].f);
        }
        for(int i=t;i!=s;i=e[pre[i]^1].y){
            e[pre[i]].f-=aug;
            e[pre[i]^1].f+=aug;
        }
        flow+=aug;
        cost+=aug*h[t];
    }
    return make_pair(flow,cost);
}
};

```

差分约束

n元一次不等式组，包含n个变量 $x_1 \dots x_n$ ，以及m个约束条件，形如 $x_i - x_j \leq c_k$ ，其中 c_k 为常量。令 dis_0 等于0，0向所有的点连一条点权为0的边， $dis[i] \leq dis[j] + c_k$ ，则j到i连一条长度为 c_k 的边。如果存在负环则无解。

强连通分量

```

function<void(int)> tarjan=[&](int x){
    dfn[x]=low[x]=++cnt;
    st.push(x);
    instack[x]=1;
    for(int &p:v[x]){
        if(!dfn[p]){
            tarjan(p);
            low[x]=min(low[x],low[p]);
        }else if(instack[p]){
            low[x]=min(low[x],dfn[p]);
        }
    }
    if(low[x]==dfn[x]){
        ans.push_back({});
        while(!st.empty()&&st.top()!=x){
            int f=st.top();
            st.pop();
            ans.back().push_back(f);
            belong[f]=ans.size();
            instack[f]=0;
        }
        st.pop();
        ans.back().push_back(x);
        instack[x]=0;
        belong[x]=ans.size();
    }
};

for(int i=1;i<=n;i++){
    if(!dfn[i]) tarjan(i);
}

```

割点与桥

如果某个顶点u，存在一个子节点v使得 $\text{low}_v > \text{dfn}_u$ ，不能回到祖先，则u为割点，根节点需要单独考虑，如果遍历了一个子节点就可以将所有点都遍历完，那根节点就不是割点，否则是割点

```
int cnt=0;
vector<int> dfn(n+1,0), low(n+1,0);
vector<bool> flag(n+1,0);
function<void(int,int)> tarjan=[&](int x,int fa){
    int son=0;
    low[x]=dfn[x]=++cnt;
    for(int &p:v[x]){
        if(!dfn[p]){
            son++;
            tarjan(p,x);
            low[x]=min(low[x],low[p]);
            if(low[p]>dfn[x]){
                flag[x]=1;
            }
        }else if(p!=fa){
            low[x]=min(low[x],dfn[p]);
        }
    }
    if(!fa&&son<=1){
        flag[x]=0;
    }
};
for(int i=1;i<=n;i++){
    if(!dfn[i]){
        tarjan(i,0);
    }
}
```

如果某个顶点u，存在一个子节点v使得 $\text{low}_v > \text{dfn}_u$ ，则u-v是割边，不用特判根节点

$\text{flag}[x]=1$ ，表示 $\text{fa}[x] \rightarrow x$ 是桥

```
int cnt=0;
vector<int> dfn(n+1,0), low(n+1,0);
vector<bool> flag(n+1,0);
function<void(int,int)> tarjan=[&](int x,int fa){
    int son=0;
    low[x]=dfn[x]=++cnt;
    for(int &p:v[x]){
        if(!dfn[p]){
            son++;
            tarjan(p,x);
            low[x]=min(low[x],low[p]);
            if(low[p]>dfn[x]){
                flag[p]=1;
            }
        }else if(p!=fa){
            low[x]=min(low[x],dfn[p]);
        }
    }
}
```

```

};

for(int i=1;i<=n;i++){
    if(!dfn[i]){
        tarjan(i,0);
    }
}

```

双联通分量

边双联通分量

先求出桥，把割点删去，剩下的极大联通子图就是边双联通分量，不能用常规方法存图

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(n+1);
    vector<int> e;
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(e.size());
        e.push_back(y);
        v[y].push_back(e.size());
        e.push_back(x);
    }
    int cnt=0;
    vector<int> dfn(n+1,0),low(n+1,0);
    vector<bool> flag(e.size(),0);
    function<void(int,int)> tarjan=[&](int x,int laste){
        low[x]=dfn[x]=++cnt;
        for(int &p:v[x]){
            int to=e[p];
            if(p==(laste^1)) continue;
            if(!dfn[to]){
                tarjan(to,p);
                low[x]=min(low[x],low[to]);
                if(low[to]>dfn[x]){
                    flag[p]=flag[p^1]=1;
                }
            }else{
                low[x]=min(low[x],dfn[to]);
            }
        }
    };
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            tarjan(i,2*m);
        }
    }
    vector<bool> vis(n+1,0);
    vector<vector<int>> ebcc;

```

```

function<void(int,int)> dfs=[&](int x,int id){
    ebcc[id].push_back(x);
    vis[x]=1;
    for(int &p:v[x]){
        if(vis[e[p]]||flag[p]) continue;
        dfs(e[p],id);
    }
};

for(int i=1;i<=n;i++){
    if(!vis[i]){
        ebcc.push_back({});
        dfs(i,ebcc.size()-1);
    }
}
cout<<ebcc.size()<<"\n";
for(auto &p:ebcc){
    cout<<p.size()<<" ";
    for(auto &q:p){
        cout<<q<<" ";
    }
    cout<<"\n";
}
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(n+1);
    vector<int> e;
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(e.size());
        e.push_back(y);
        v[y].push_back(e.size());
        e.push_back(x);
    }
    int cnt=0;
    vector<int> dfn(n+1,0),low(n+1,0);
    vector<bool> flag(e.size(),0);
    stack<int> st;
    vector<vector<int>> ebcc;
    function<void(int,int)> tarjan=[&](int x,int laste){
        low[x]=dfn[x]=++cnt;
        st.push(x);

```

```

        for(int &p:v[x]){
            int to=e[p];
            if(p==(laste^1)) continue;
            if(!dfn[to]){
                tarjan(to,p);
                low[x]=min(low[x],low[to]);
            }else{
                low[x]=min(low[x],dfn[to]);
            }
        }
        if(dfn[x]==low[x]){
            ebcc.push_back({});
            while(!st.empty()&&st.top()!=x){
                ebcc.back().push_back(st.top());
                st.pop();
            }
            ebcc.back().push_back(x);
            st.pop();
        }
    };
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            tarjan(i,2*m);
        }
    }
    cout<<ebcc.size()<<"\n";
    for(auto &p:ebcc){
        cout<<p.size()<<" ";
        for(auto &q:p){
            cout<<q<<" ";
        }
        cout<<"\n";
    }
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

点双联通分量

两个点双最多有一个公共点，且一定是割点。

对于一个点双，他在dfs搜索树中dfn值最小的一定是割点或者树根

当一个点是割点的时候，他一定是点双的根

当一个点是树根的时候：

如果有两个及以上的子树，他是割点

只有一个子树，他是一个点双的根

没有子树，他自己是一个点双

需要特判自环的情况

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(n+1);
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        if(x!=y){
            v[x].push_back(y);
            v[y].push_back(x);
        }
    }
    int cnt=0;
    vector<int> dfn(n+1,0),low(n+1,0);
    stack<int> st;
    vector<vector<int>> pbcc;
    function<void(int,int)> tarjan=[&](int x,int root){
        low[x]=dfn[x]=++cnt;
        st.push(x);
        if(x==root&&v[x].size()==0){
            pbcc.push_back({});
            pbcc.back().push_back(x);
            return;
        }
        for(int &p:v[x]){
            if(!dfn[p]){
                tarjan(p,root);
                low[x]=min(low[x],low[p]);
                if(low[p]>=dfn[x]){
                    pbcc.push_back({});
                    while(!st.empty()&&st.top()!=p){
                        pbcc.back().push_back(st.top());
                        st.pop();
                    }
                    pbcc.back().push_back(p);
                    st.pop();
                    pbcc.back().push_back(x);
                }
            }else{
                low[x]=min(low[x],dfn[p]);
            }
        }
    };
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            tarjan(i,i);
        }
    }
    cout<<pbcc.size()<<"\n";
    for(auto &p:pbcc){
```

```

        cout<<p.size()<<" ";
        for(auto &q:p){
            cout<<q<<" ";
        }
        cout<<"\n";
    }
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

最短路

SPFA

```

queue<int> q;
vector<bool> vis(n+1, 0);
vector<int> dis(n+1, 1e18);
dis[0]=0;
q.push(0);
vis[0]=1;
while(!q.empty()){
    int f=q.front();
    q.pop();
    vis[f]=0;
    for(auto &[to,w]:v[f]){
        if(dis[to]>dis[f]+w){
            dis[to]=dis[f]+w;
            if(!vis[to]){
                vis[to]=1;
                q.push(to);
            }
        }
    }
}

```

dijkstra

时间复杂度 $O(m \log m)$

```

priority_queue<pair<int,int>,vector<pair<int,int>>,greater<>> q;
vector<bool> vis(n+1, 0);
vector<int> dis(n+1, INF);
dis[1]=0;
q.push({0,1});
while(!q.empty()){
    auto [d,x]=q.top();
    q.pop();
    if(vis[x]) continue;
    vis[x]=1;

```

```

        for(auto &[to,w]:v[x]){
            if(dis[to]>dis[x]+w){
                dis[to]=dis[x]+w;
                q.push({dis[to],to});
            }
        }
    }
}

```

Johnson

建立虚点0，向每个点连一条边权为0的有向边。先进行一次SPFA，求出虚点0到每个点i的最短路 $h[i]$ ，将每条边 $v[i][j]$ 的边权设置为 $v[i][j] + h[i] - h[j]$ ，边权变为非负数，跑dijkstra，到k的最短路为 $dis[k] + h[k] - h[s]$

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
const int INF=1e9;
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<pair<int,int>>> v(n+1);
    for(int i=1;i<=m;i++){
        int x,y,k;
        cin>>x>>y>>k;
        v[x].push_back({y,k});
    }
    queue<int> q;
    vector<int> h(n+1,INF);
    vector<bool> vis(n+1,0);
    vector<int> cnt(n+1,0);
    q.push(0);
    vis[0]=1;
    h[0]=0;
    bool ok=1;
    while(!q.empty()){
        int f=q.front();
        q.pop();
        vis[f]=0;
        if(++cnt[f]>n+1){
            ok=0;
            break;
        }
        if(f==0){
            for(int i=1;i<=n;i++){
                if(h[i]>h[f]){
                    h[i]=h[f];
                    if(!vis[i]){
                        vis[i]=1;
                        q.push(i);
                    }
                }
            }
        }
    }
    }else{
        for(auto &[to,w]:v[f]){

```

```

        if(h[to]>h[f]+w){
            h[to]=h[f]+w;
            if(!vis[to]){
                vis[to]=1;
                q.push(to);
            }
        }
    }
}

if(!ok){
    cout<<-1<<"\n";
    return;
}

for(int i=1;i<=n;i++){
    for(auto &[to,w]:v[i]){
        w=w+h[i]-h[to];
    }
}

for(int i=1;i<=n;i++){
    fill(vis.begin(),vis.end(),0);
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<>> q;
    vector<int> dis(n+1,INF);
    dis[i]=0;
    q.push({0,i});
    while(!q.empty()){
        auto [d,x]=q.top();
        q.pop();
        if(vis[x]) continue;
        vis[x]=1;
        for(auto &[to,w]:v[x]){
            if(dis[to]>dis[x]+w){
                dis[to]=dis[x]+w;
                q.push({dis[to],to});
            }
        }
    }
    int sum=0;
    for(int j=1;j<=n;j++){
        if(dis[j]==INF) sum+=j*INF;
        else sum+=j*(dis[j]+h[j]-h[i]);
    }
    cout<<sum<<"\n";
}
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

普通环计数

n个点m条边无向图，求简单环数量

$dp[i][j]$ 状压，表示i的状态下，从i的`_builtin_ctz`点作为起点，有多少种情况。i表示经过的点，j表示现在在的点

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> dp(1<<20, vector<int>(n));
    vector<vector<int>> v(n);
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        --x,--y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    for(int i=0;i<n;i++){
        dp[1<<i][i]=1;
    }
    int ans=0;
    for(int i=1;i<(1<<n);i++){
        for(int j=0;j<n;j++){
            if(!dp[i][j]) continue;
            for(int &p:v[j]){
                if((i&-i)>(1<<p)) continue;
                if(i&(1<<p)){
                    if((i&-i)==(1<<p)){
                        ans+=dp[i][j];
                    }
                }else{
                    dp[i|(1<<p)][p]+=dp[i][j];
                }
            }
        }
    }
    cout<<(ans-m)/2<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}
```

三元环计数

给所有边定向，从度数小的指向度数大的，度数相同的从编号小的指向编号大的，此时图变成有向无环图DAG。枚举u和u指向的点v，再枚举v指向的点w，检验u, w是否相连，时间复杂度 $O(m\sqrt{m})$

给一个n个点m条边的简单无向图，求三元环个数

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<pair<int,int>> v(m);
    vector<int> cnt(n+1,0);
    vector<set<int>> son(n+1);
    for(int i=0;i<m;i++){
        cin>>v[i].first>>v[i].second;
        cnt[v[i].first]++;
        cnt[v[i].second]++;
    }
    for(int i=0;i<m;i++){
        if(cnt[v[i].first]==cnt[v[i].second]){
            if(v[i].first>v[i].second) swap(v[i].first,v[i].second);
        }else{
            if(cnt[v[i].first]>cnt[v[i].second]){
                swap(v[i].first,v[i].second);
            }
        }
    }
    vector<vector<int>> node(n+1);
    for(int i=0;i<m;i++){
        node[v[i].first].push_back(v[i].second);
    }
    int ans=0;
    vector<bool> vis(n+1,0);
    for(int i=1;i<=n;i++){
        for(int &p:node[i]) vis[p]=1;
        for(int &p:node[i]){
            for(int &q:node[p]){
                if(vis[q]) ans++;
            }
        }
        for(int &p:node[i]) vis[p]=0;
    }
    cout<<ans<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}
```

2-SAT问题

n个集合，每个集合有两个元素，已知若干个 $\langle a, b \rangle$ ，表示a与b矛盾（a, b不属于同一个集合），需要从每个集合中选择一个元素，判断能否选n个两两不矛盾元素。

a1和b2有矛盾，则建有向边a1->b1,b2->a2，tarjan缩点判断是否有一个集合中的两个元素都在同一个强联通块，如果是则不可能。

选择的时候，优先选择dfs序大的，即scc编号小的

也可以爆搜

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<int>> v(2*n+1);
    while(m--){
        int i,a,j,b;
        cin>>i>>a>>j>>b;
        if(a==0){
            if(b==0){
                v[i+n].push_back(j);
                v[j+n].push_back(i);
            }else{
                v[i+n].push_back(j+n);
                v[j].push_back(i);
            }
        }else{
            if(b==0){
                v[i].push_back(j);
                v[j+n].push_back(i+n);
            }else{
                v[i].push_back(j+n);
                v[j].push_back(i+n);
            }
        }
    }
    vector<int> low(2*n+1,0),dfn(2*n+1,0),belong(2*n+1,0);
    vector<bool> instack(2*n+1,0);
    int cnt=0;
    int id=0;
    stack<int> st;
    function<void(int)> tarjan=[&](int x){
        low[x]=dfn[x]=++cnt;
        st.push(x);
        instack[x]=1;
        for(int &p:v[x]){
            if(!dfn[p]){
                tarjan(p);
                low[x]=min(low[x],low[p]);
            }else if(instack[p]){
                low[x]=min(low[x],dfn[p]);
            }
        }
        if(low[x]==dfn[x]){
            id++;
            while(!st.empty()&&st.top()!=x){
                int f=st.top();
                if(f>x)
                    belong[f]=id;
                else
                    belong[belong[f]]=id;
                st.pop();
            }
        }
    };
    if(low[x]==dfn[x]){
        id++;
        while(!st.empty()&&st.top()!=x){
            int f=st.top();
            if(f>x)
                belong[f]=id;
            else
                belong[belong[f]]=id;
            st.pop();
        }
    }
}
```

```

        st.pop();
        instack[f]=0;
        belong[f]=id;
    }
    belong[st.top()]=id;
    instack[st.top()]=0;
    st.pop();
}
};

for(int i=1;i<=2*n;i++){
    if(!dfn[i]){
        tarjan(i);
    }
}

for(int i=1;i<=n;i++){
    if(belong[i]==belong[i+n]){
        cout<<"IMPOSSIBLE\n";
        return;
    }
}
cout<<"POSSIBLE\n";
for(int i=1;i<=n;i++){
    if(belong[i]>belong[i+n]) cout<<1<<" ";
    else cout<<0<<" ";
}
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

树链剖分

两次dfs，第一次求出fa, dep, son, sz, 第二次求出dfn, top, rnk (dfs序对应的点编号)。

点权，要先init

```

struct HLD{
    int n;
    vector<vector<int>> v;
    vector<int> fa,dep,son,sz,dfn,top,rnk;
    int cnt,root;
    SegmentTree tree;
    HLD(int n,int root):n(n),root(root){
        v.resize(n+1);
        fa.resize(n+1,0);
        dep.resize(n+1,0);
        son.resize(n+1,-1);
        sz.resize(n+1,1);
        dfn.resize(n+1,0);
        top.resize(n+1,0);
        rnk.resize(n+1,0);
    }
};

```

```

cnt=0;
}
void addedge(int x,int y){
    v[x].push_back(y);
    v[y].push_back(x);
}
void dfs1(int x,int father){
    dep[x]=dep[father]+1;
    fa[x]=father;
    for(int &p:v[x]){
        if(p==father) continue;
        dfs1(p,x);
        sz[x]+=sz[p];
        if(sz[x]==-1||sz[p]>sz[son[x]]) son[x]=p;
    }
}
void dfs2(int x,int father,int u){
    dfn[x]=++cnt;
    rnk[cnt]=x;
    top[x]=u;
    if(son[x]==-1) return;
    dfs2(son[x],x,u);
    for(int &p:v[x]){
        if(p==father||p==son[x]) continue;
        dfs2(p,x,p);
    }
}
void init(vector<int> &v){
    dfs1(root,0);
    dfs2(root,0,root);
    vector<int> vv(n+1);
    for(int i=1;i<=n;i++){
        vv[dfn[i]]=v[i];
    }
    tree=SegmentTree(n,vv);
}
int LCA(int x,int y){
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        x=fa[top[x]];
    }
    return dep[x]>dep[y]?y:x;
}
void updaterroute(int x,int y,int delta){
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        tree.update(1,1,n,dfn[top[x]],dfn[x],delta);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    tree.update(1,1,n,dfn[x],dfn[y],delta);
}
int queryroute(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);

```

```

        ans+=tree.query(1,1,n,dfn[top[x]],dfn[x]);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    return ans+tree.query(1,1,n,dfn[x],dfn[y]);
}
void updatesubtree(int x,int delta){
    tree.update(1,1,n,dfn[x],dfn[x]+sz[x]-1,delta);
}
int querysubtree(int x){
    return tree.query(1,1,n,dfn[x],dfn[x]+sz[x]-1);
}
};


```

```

int p;
struct SegmentTree{
    struct edge{
        int sum;
    };
    vector<int> lazy;
    vector<edge> node;
    int n;
    void pushup(int id,int l,int r){
        node[id].sum=(node[id<<1].sum+node[id<<1|1].sum)%p;
    }
    void pushdown(int id,int l,int r){
        if(lazy[id]){
            int mid=l+(r-l>>1);
            lazy[id<<1]+=lazy[id];
            lazy[id<<1|1]+=lazy[id];
            node[id<<1].sum+=(mid-l+1)*lazy[id]%p;
            node[id<<1|1].sum+=(r-mid)*lazy[id]%p;
            lazy[id<<1]%=p;
            lazy[id<<1|1]%=p;
            node[id<<1].sum%=p;
            node[id<<1|1].sum%=p;
            lazy[id]=0;
        }
    }
    SegmentTree(int n):n(n){
        node.resize((n<<2)+5);
        lazy.assign((n<<2)+5,0);
    }
    SegmentTree(){}
    void init(vector<int> &v){
        function<void(int,int,int)> buildtree=[&](int id,int l,int r){
            lazy[id]=0;
            if(l==r){
                node[id].sum=v[l]%p;
                return;
            }
            int mid=l+(r-l>>1);
            buildtree(id<<1,l,mid);
            buildtree(id<<1|1,mid+1,r);
            pushup(id,l,r);
        };
        buildtree(0,0,n);
    }
};


```

```

    };
    buildtree(1,1,n);
}
SegmentTree(int n,vector<int> &v):n(n){
    node.resize((n<<2)+5);
    lazy.assign((n<<2)+5,0);
    init(v);
}
void update(int id,int l,int r,int x,int y,int delta){
    if(x<=l&&r<=y){
        lazy[id]+=delta;
        node[id].sum+=delta*(r-l+1);
        lazy[id]%=p;
        node[id].sum%=p;
        return;
    }
    pushdown(id,l,r);
    int mid=l+(r-l>>1);
    if(x<=mid) update(id<<1,l,mid,x,y,delta);
    if(y>mid) update(id<<1|1,mid+1,r,x,y,delta);
    pushup(id,l,r);
}
int query(int id,int l,int r,int x,int y){
    if(x<=l&&r<=y) return node[id].sum;
    pushdown(id,l,r);
    int mid=l+(r-l>>1);
    int ans=0;
    if(x<=mid) ans+=query(id<<1,l,mid,x,y);
    ans%=p;
    if(y>mid) ans+=query(id<<1|1,mid+1,r,x,y);
    ans%=p;
    return ans;
}
};

struct HLD{
    int n;
    vector<vector<int>> v;
    vector<int> fa,dep,son,sz,dfn,top,rnk;
    int cnt,root;
    SegmentTree tree;
    HLD(int n,int root):n(n),root(root){
        v.resize(n+1);
        fa.resize(n+1,0);
        dep.resize(n+1,0);
        son.resize(n+1,-1);
        sz.resize(n+1,1);
        dfn.resize(n+1,0);
        top.resize(n+1,0);
        rnk.resize(n+1,0);
        cnt=0;
    }
    void addedge(int x,int y){
        v[x].push_back(y);
        v[y].push_back(x);
    }
    void dfs1(int x,int father){

```

```

dep[x]=dep[father]+1;
fa[x]=father;
for(int &p:v[x]){
    if(p==father) continue;
    dfs1(p,x);
    sz[x]+=sz[p];
    if(son[x]==-1||sz[p]>sz[son[x]]) son[x]=p;
}
void dfs2(int x,int father,int u){
dfn[x]=++cnt;
rnk[cnt]=x;
top[x]=u;
if(son[x]==-1) return;
dfs2(son[x],x,u);
for(int &p:v[x]){
    if(p==father||p==son[x]) continue;
    dfs2(p,x,p);
}
}
void init(vector<int> &v){
dfs1(root,0);
dfs2(root,0,root);
vector<int> vv(n+1);
for(int i=1;i<=n;i++){
    vv[dfn[i]]=v[i];
}
tree=SegmentTree(n,vv);
}
int LCA(int x,int y){
while(top[x]!=top[y]){
    if(dep[top[x]]<dep[top[y]]) swap(x,y);
    x=fa[top[x]];
}
return dep[x]>dep[y]?y:x;
}
void updaterroute(int x,int y,int delta){
delta%=p;
while(top[x]!=top[y]){
    if(dep[top[x]]<dep[top[y]]) swap(x,y);
    tree.update(1,1,n,dfn[top[x]],dfn[x],delta);
    x=fa[top[x]];
}
if(dep[x]>dep[y]) swap(x,y);
tree.update(1,1,n,dfn[x],dfn[y],delta);
}
int queryroute(int x,int y){
int ans=0;
while(top[x]!=top[y]){
    if(dep[top[x]]<dep[top[y]]) swap(x,y);
    ans+=tree.query(1,1,n,dfn[top[x]],dfn[x]);
    ans%=p;
    x=fa[top[x]];
}
if(dep[x]>dep[y]) swap(x,y);
return (ans+tree.query(1,1,n,dfn[x],dfn[y]))%p;
}

```

```

    }
    void updatesubtree(int x,int delta){
        delta%=p;
        tree.update(1,1,n,dfn[x],dfn[x]+sz[x]-1,delta);
    }
    int querysubtree(int x){
        return tree.query(1,1,n,dfn[x],dfn[x]+sz[x]-1);
    }
}

```

表达式树

中序表达式转换为逆波兰式

符号优先级: ^:3, */:2, +-:1, ():0

s[i]中:

①如果是数字, 压入结果栈

②如果是乘方, 压入符号栈

③如果是+*/-, 将符号栈栈顶比他优先级高或相同的符号——弹出, 并压入结果栈, 然后将s[i]压入符号栈

④如果是左括号, 压入符号栈

⑤如果是右括号, 将符号栈顶部第一个左括号之前的符号弹出并压入结果栈, 并弹出左括号

最后将符号栈中剩余的符号都弹出并压入结果栈

树哈希

用于判断树是否同构

以某个节点为根的子树的哈希值, 就是以它的所有儿子为根的子树的哈希值构成的多重集的哈希值

$$h_x = f(\{h_i | i \in son(x)\})$$

其中哈希函数为: $f(S) = (c + \sum_{x \in S} g(x)) \bmod m$

其中c为常数, 一般使用1。g为整数到整数的映射

m棵树, 每行第一个数n表示点数, 接下来n个数表示每个点的父节点。找出与每个树同构的树的最小编号。

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MOD=1e9+7;
mt19937_64 myrnd(time(0));
int rnd(){
    return myrnd()>>1;
}
map<int,int> mp;
int gethash(int x){
    if(mp.find(x)!=mp.end()) return mp[x];
    else return mp[x]=rnd();
}

```

```

void solve(){
    int m;
    cin>>m;
    map<int,int> mp1;
    for(int i=0;i<m;i++){
        int n;
        cin>>n;
        vector<vector<int>> v(n+1);
        int root;
        for(int j=1;j<=n;j++){
            int x;
            cin>>x;
            if(x==0) root=j;
            else v[x].push_back(j);
        }
        vector<int> hash(n+1);
        function<void(int,int)> dfs1=[&](int x,int fa){
            hash[x]=1;
            for(int &p:v[x]){
                if(p==fa) continue;
                dfs1(p,x);
                hash[x]=(hash[x]+gethash(hash[p]))%MOD;
            }
        };
        dfs1(root,0);
        set<int> se;
        function<void(int,int)> dfs2=[&](int x,int fa){
            se.insert(hash[x]);
            for(int &p:v[x]){
                if(p==fa) continue;
                int a=hash[x],b=hash[p];
                hash[x]=((hash[x]-gethash(hash[p]))%MOD+MOD)%MOD;
                hash[p]=(hash[p]+gethash(hash[x]))%MOD;
                dfs2(p,x);
                hash[x]=a,hash[p]=b;
            }
        };
        dfs2(root,0);
        int ans=i+1;
        for(auto &p:se){
            if(mp1.find(p)==mp1.end()){
                mp1[p]=i+1;
            }else{
                ans=min(ans,mp1[p]);
            }
        }
        cout<<ans<<"\n";
    }
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

欧拉通路&回路

Hierholzer

```
auto dfs=[&](auto self,int x)->void{
    for(;pos[x]<v[x].size();){
        pos[x]++;
        self(self,v[x][pos[x]-1]);
    }
    st.push(x);
};
```

```
stack<int> st;
auto dfs=[&](auto self,int x)->void{
    while(pos[x]<v[x].size()){
        int g=v[x][pos[x]];
        if(use[g]){
            pos[x]++;
            continue;
        }
        use[g]=1;
        use[g^1]=1;
        pos[x]++;
        self(self,e[g]);
    }
    st.push(x);
};
dfs(dfs,s);
```

```
void solve(){
    int m;
    cin>>m;
    vector<int> e;
    vector<int> in(501,0);
    vector<int> pos(501,0);
    vector<bool> use;
    vector<vector<int>> v(501);
    for(int i=1;i<=m;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(e.size());
        e.push_back(y);
        v[y].push_back(e.size());
        e.push_back(x);
        in[x]++;
        in[y]++;
        use.push_back(0);
        use.push_back(0);
    }
    for(int i=1;i<=500;i++){
        sort(v[i].begin(),v[i].end(),[&](int &a,int &b){
            return e[a]<e[b];
        });
    }
}
```

```

    });
}

vector<int> tmp,tmp1;
for(int i=1;i<=500;i++){
    if(in[i]==0) continue;
    if(in[i]%2==0){
        tmp.push_back(i);
    }else{
        tmp1.push_back(i);
    }
}
sort(tmp.begin(),tmp.end());
sort(tmp1.begin(),tmp1.end());
int s;
if(tmp1.size()!=0){
    s=tmp1[0];
}else{
    s=tmp[0];
}
stack<int> st;
auto dfs=[&](auto self,int x)->void{
    while(pos[x]<v[x].size()){
        if(use[v[x][pos[x]]]){
            pos[x]++;
            continue;
        }
        use[v[x][pos[x]]]=1;
        use[v[x][pos[x]]^1]=1;
        pos[x]++;
        self(self,e[v[x][pos[x]-1]]);
    }
    st.push(x);
};
dfs(dfs,s);
while(!st.empty()){
    cout<<st.top()<<"\n";
    st.pop();
}
}
}

```

最小路径覆盖

最小路径覆盖为用最少的路径走过有向图所有点

最小路径覆盖=点数-拆点后二分图最大匹配

```

//每行输出一条路径，最后一行输出路径条数
#include<bits/stdc++.h>
using namespace std;
#define int long long
struct Flow{
    const int n;
    const int MAXN=1e18;
    vector<pair<int,int>> e;
    vector<vector<int>> g;

```

```

vector<int> cur,dep;
Flow(int n):n(n),g(n+1){}
bool bfs(int s,int t){
    dep.assign(n+1,-1);
    queue<int> q;
    dep[s]=0;
    q.push(s);
    while(!q.empty()){
        const int u=q.front();
        q.pop();
        for(int i:g[u]){
            auto [v,c]=e[i];
            if(c>0&&dep[v]==-1){
                dep[v]=dep[u]+1;
                if(v==t) return 1;
                q.push(v);
            }
        }
    }
    return 0;
}
int dfs(int u,int t,int f){
    if(u==t) return f;
    int res=f;
    for(int &i=cur[u];i<g[u].size();i++){
        const int j=g[u][i];
        auto [v,c]=e[j];
        if(c>0&&dep[v]==dep[u]+1){
            int out=dfs(v,t,min(res,c));
            e[j].second-=out;
            e[j^1].second+=out;
            res-=out;
            if(res==0) return f;
        }
    }
    return f-res;
}
void add(int u,int v,int c){
    g[u].push_back(e.size());
    e.emplace_back(v,c);
    g[v].push_back(e.size());
    e.emplace_back(u,0);
}
int work(int s,int t){
    int ans=0;
    while(bfs(s,t)){
        cur.assign(n+1,0);
        ans+=dfs(s,t,MAXN);
    }
    return ans;
}
};

void solve(){
    int n,m;
    cin>>n>>m;
    Flow flow(2*n+2);

```

```

for(int i=0;i<m;i++){
    int x,y;
    cin>>x>>y;
    flow.add(x+n,y,1);
}
for(int i=1;i<=n;i++){
    flow.add(2*n+1,n+i,1);
    flow.add(i,2*n+2,1);
}
int ans=n-flow.work(2*n+1,2*n+2);
vector<int> in(n+1,0);
vector<vector<int>> v(n+1);
for(int i=1;i<=n;i++){
    for(int id:flow.g[i+n]){
        if(flow.e[id].first>n) continue;
        if(flow.e[id].second){
            v[i].push_back(flow.e[id].first);
            in[flow.e[id].first]++;
        }
    }
}
for(int i=1;i<=n;i++){
    if(in[i]) continue;
    int now=i;
    while(1){
        cout<<now<<" ";
        if(v[now].empty()) break;
        now=v[now].front();
    }
    cout<<"\n";
}
cout<<ans<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

树的重心

最大独立集

重心的所有子树中最大子数树节点数最少 ($\leq n/2$)

点分治

路径分为两种，一种经过rt，一种不经过rt

每次对于一棵树，取他的重心，求这个树里面每个点到根的距离。

继续对于他的每个儿子节点作为根的子树，继续找重心，继续上面的操作。

//树上是否存在边权和为k的路径

```

#include<bits/stdc++.h>
using namespace std;
void solve(){
    int n,m;
    cin>>n>>m;
    vector<vector<pair<int,int>>> v(n+1);
    for(int i=0;i<n-1;i++){
        int x,y,t;
        cin>>x>>y>>t;
        v[x].emplace_back(y,t);
        v[y].emplace_back(x,t);
    }
    vector<int> que(m),sz(n+1),dp(n+1);
    vector<bool> p1(1e7+5),ans(m),vis(n+1);
    p1[0]=1;
    for(int &p:que) cin>>p;
    function<void(int,int)> getsize=[&](int x,int fa){
        sz[x]=1;
        for(auto &[to,w]:v[x]){
            if(to==fa||vis[to]) continue;
            getsize(to,x);
            sz[x]+=sz[to];
        }
    };
    auto getzx=[&](int x,int fa,int tot){
        int rt=0;
        function<void(int,int)> dfs=[&](int x,int fa){
            int maxn=0;
            for(auto &[to,w]:v[x]){
                if(to==fa||vis[to]) continue;
                dfs(to,x);
                maxn=max(maxn,sz[to]);
            }
            maxn=max(maxn,tot-sz[x]);
            if(maxn<=tot/2) rt=x;
        };
        dfs(x,fa);
        return rt;
    };
    function<void(int)> calc=[&](int x){
        vector<int> p2;
        stack<int> st;
        function<void(int,int)> getdis=[&](int x,int fa){
            p2.push_back(dp[x]);
            for(auto &[to,w]:v[x]){
                if(to==fa||vis[to]) continue;
                dp[to]=dp[x]+w;
                getdis(to,x);
            }
        };
        for(auto &[to,w]:v[x]){
            p2.clear();
            if(vis[to]) continue;
            dp[to]=w;
            getdis(to,x);
            for(int i=0;i<m;i++){
                if(p2[i]==p2.back())

```

```

        if(ans[i]) continue;
        for(int &p:p2){
            if(que[i]-p>=0&&que[i]-p<=1e7&&p1[que[i]-p]){
                ans[i]=1;
                break;
            }
        }
        for(int &p:p2){
            if(p>1e7) continue;
            p1[p]=1;
            st.push(p);
        }
    }
    while(!st.empty()){
        p1[st.top()]=0;
        st.pop();
    }
};

function<void(int)> solve=[&](int x){
    vis[x]=1;
    calc(x);
    for(auto &[to,w]:v[x]){
        if(vis[to]) continue;
        getsize(to,x);
        int rt=getzx(to,x,sz[to]);
        solve(rt);
    }
};
getsize(1,0);
int rt=getzx(1,0,n);
solve(rt);
for(int i=0;i<m;i++){
    if(ans[i]) cout<<"AYE\n";
    else cout<<"NAY\n";
}
}

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

点分树

按照点分治的重心，构建一棵新树，也就是对于每一个找到的重心，将上一层分治的重心设置为他的父节点，得到一棵大小不变，最多 $\log(n)$ 层的虚树。

每个点子树大小的和为 $n \log n$ 。

Kruskal重构树

```
struct DSU{
```

```

vector<int> fa, sz;
int n;
DSU(int n) : n(n){
    fa.resize(n + 1);
    sz.resize(n + 1, 1);
    for (int i = 1; i <= n; i++){
        fa[i] = i;
    }
}
int find(int a){
    return fa[a] == a ? a : fa[a] = find(fa[a]);
}
void merge(int x, int y){
    int fx = find(x), fy = find(y);
    if (fx == fy)
        return;
    if (sz[fx] < sz[fy])
        swap(fx, fy);
    fa[fy] = fx;
    sz[fx] += sz[fy];
    return;
}
};

struct edge {
    int x, y, k;
};

struct Ex_kruskal {
    int n;
    vector<vector<int>> fa;
    vector<int> val;
    vector<int> dep;
    DSU dsu;
    int LCA(int x, int y) {
        // 不连通返回-1
        if (dsu.find(x) != dsu.find(y))
            return -1;
        if (dep[x] < dep[y])
            swap(x, y);
        for (int i = 20; i >= 0; i--) {
            if (dep[fa[x][i]] >= dep[y]) {
                x = fa[x][i];
            }
        }
        if (x == y) return x;
        for (int i = 20; i >= 0; i--) {
            if (fa[x][i] != fa[y][i]) {
                x = fa[x][i];
                y = fa[y][i];
            }
        }
    }
    return val[fa[x][0]];
}
Ex_kruskal(int &_n, vector<edge> &_v) : n(_n), dsu(2 * _n), fa(2 * _n + 1,
vector<int>(21)), val(2 * _n + 1), dep(2 * _n + 1) {
    sort(_v.begin(), _v.end(), [&](edge &a, edge &b) { return a.k < b.k; });
    vector<vector<int>> v(2 * _n + 1);

```

```

vector<bool> vis(2 * _n + 1);
int cnt = 1;
for (int i = 0; i < _v.size(); i++) {
    int fax = dsu.find(_v[i].x);
    int fay = dsu.find(_v[i].y);
    if (fax != fay) {
        v[fax].emplace_back(n + cnt);
        v[fay].emplace_back(n + cnt);
        v[n + cnt].emplace_back(fax);
        v[n + cnt].emplace_back(fay);
        val[n + cnt] = _v[i].k;
        dsu.fa[fax] = n + cnt;
        dsu.fa[fay] = n + cnt;
        cnt++;
    }
    if (cnt == n) {
        break;
    }
}
auto dfs = [&](auto &&self, int x, int father) -> void {
    vis[x] = 1;
    dep[x] = dep[father] + 1;
    fa[x][0] = father;
    for (int i = 1; i <= 20; i++) {
        fa[x][i] = fa[fa[x][i - 1]][i - 1];
    }
    for (int &to : v[x]) {
        if (to == father)
            continue;
        self(self, to, x);
    }
};
for (int i = 2 * n - 1; i > 0; i--) {
    if (!vis[i])
        dfs(dfs, i, 0);
}
};

```

最近公共祖先

倍增

时间复杂度 $O(\log n)$

```

vector<int> dep(n + 1, 0);
vector<vector<int>> fa(n + 1, vector<int>(21));
auto dfs = [&](auto &&self, int x, int father) -> void
{
    dep[x] = dep[father] + 1;
    fa[x][0] = father;
    for (int i = 1; i <= 20; i++)
    {
        fa[x][i] = fa[fa[x][i - 1]][i - 1];
    }
}

```

```

    for (int &p : v[x])
    {
        if (p == father)
            continue;
        self(self, p, x);
    }
};

auto LCA = [&](int x, int y)
{
    if (dep[x] < dep[y])
        swap(x, y);
    for (int i = 20; i >= 0; i--)
    {
        if (dep[fa[x][i]] >= dep[y])
            x = fa[x][i];
    }
    if (x == y)
        return x;
    for (int i = 20; i >= 0; i--)
    {
        if (fa[x][i] != fa[y][i])
        {
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
};
dfs(dfs, 1, 0);

```

tarjan

预处理 $O(n + q)$, 询问 $O(1)$

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
struct DSU{
    vector<int> fa;
    int n;
    DSU(int n){
        this->n=n;
        fa=vector<int>(n+1);
        for(int i=0;i<=n;i++) fa[i]=i;
    }
    int find(int x){
        return fa[x]==x?x:fa[x]=find(fa[x]);
    }
    bool merge(int x,int y){
        int fax=find(x);
        int fay=find(y);
        if(fax==fay) return 0;
        fa[fax]=fay;
        return 1;
    }
};

```

```

};

void solve(){
    int n,m,s;
    cin>>n>>m>>s;
    vector<vector<int>> v(n+1);
    for(int i=1;i<n;i++){
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    DSU dsu(n);
    vector<vector<pair<int,int>>> query(n+1);
    vector<int> lca(m);
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        query[x].emplace_back(y,i);
        query[y].emplace_back(x,i);
    }
    vector<int> dfn(n+1),low(n+1);
    stack<int> st;
    vector<bool> instack(n+1);
    int cnt=0;
    function<void(int)> tarjan=[&](int x){
        dfn[x]=low[x]=++cnt;
        st.push(x);
        instack[x]=1;
        for(int &p:v[x]){
            if(!dfn[p]){
                tarjan(p);
                low[x]=min(low[x],low[p]);
                dsu.merge(p,x);
            }else if(instack[p]){
                low[x]=min(low[x],dfn[p]);
            }
        }
        if(low[x]==dfn[x]){
            while(!st.empty()&&st.top()!=x){
                int f=st.top();
                st.pop();
                instack[f]=0;
            }
            st.pop();
            instack[x]=0;
        }
        for(auto &[y,id]:query[x]){
            if(lca[id]||!dfn[y]) continue;
            lca[id]=dsu.find(y);
        }
    };
    tarjan(s);
    for(int i=0;i<m;i++){
        cout<<lca[i]<<"\n";
    }
}

```

```

signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

计算几何

二维几何

```

struct Point{
    double x,y;
    double operator*(const Point &e) const{
        return x*e.x+y*e.y;
    };
    Point operator*(const double k) const{
        return {x*k,y*k};
    }
    double operator^(const Point &e) const{
        return x*e.y-e.x*y;
    }
    Point operator+(const Point &e) const{
        return {x+e.x,y+e.y};
    }
    Point operator-(const Point &e) const{
        return {x-e.x,y-e.y};
    }
    Point operator/(const double &k) const{
        return {x/k,y/k};
    }
    //象限
    inline int quad() const{
        if(x>0&&y>0) return 1;
        if(x<=0&&y>0) return 2;
        if(x<0&&y<=0) return 3;
        if(x>=0&&y<0) return 4;
        return 5;
    }
    inline static bool sortxupyup(const Point &a,const Point &b){
        if(a.x!=b.x) return a.x<b.x;
        else return a.y<b.y;
    }
    //极角排序
    inline static bool sortPointAngle(const Point &a,const Point &b){
        if(a.quad()!=b.quad()) return a.quad()<b.quad();
        return (a^b)>0;
    }
    //模长
    inline double norm() const{
        return sqrtl(x*x+y*y);
    }
}

```

```

}

//向量方向
//1 a在b逆时针方向
//0 同向或反向
//2 a在b顺时针方向
int ordervector(const Point &e){
    double p=(*this)^e;
    if(p>0) return 1;
    else if(p==0.0) return 0;
    else return 2;
}
//逆时针旋转alpha角
inline Point Spin(double alpha){
    double sinalpha=sin(alpha);
    double cosinalpha=cos(alpha);
    return {x*cosinalpha-y*sinalpha,x*sinalpha+y*cosinalpha};
}
inline double dis(const Point &e){
    Point c=(*this)-e;
    return c.norm();
}
double getangle(const Point &e) const{
    return fabs(atan2l(*this^e,*this*e));
}
Point(double x,double y):x(x),y(y){}
Point(){}
};

struct Line{
    //过x点，方向向量为y
    Point x,y;
    //type=0,点和方向向量
    //type=1, 点和点
    Line(const Point &a,const Point &b,int type){
        if(type==0){
            x=a,y=b;
        }else{
            x=a;
            y=b-a;
        }
    }
    inline double distancetopoint(const Point &e) const{
        return fabs((e-x)^y)/y.norm();
    }
};

struct Segment{
    //过x,y
    Point x,y;
    Segment(Point a,Point b):x(a),y(b){}
    inline double distancetopoint(const Point &e) const{
        Point l=y-x;
        double t=l*(e-x)/(l*l);
        if(t<0) t=0;
        else if(t>1) t=1;
        return (e-(x+l*t)).norm();
    }
};

```

```

//要先getConvex求凸包，其他的才能用
struct Polygon{
    vector<Point> p;
    vector<Point> convexhull;
    int n;
    Polygon(int n,vector<Point> &v):n(n),p(v){}
    Polygon(int n):n(n),p(n){}
    void input(){
        for(int i=0;i<n;i++){
            cin>>p[i].x>>p[i].y;
        }
    }
    void getConvex(){
        sort(p.begin(),p.end(),Point::sortxupyup);
        p.erase(unique(p.begin(),p.end(),[])(const Point &a,const Point &b){
            return a.x==b.x&&a.y==b.y;
        }),p.end());
        n=p.size();
        if(n==0) return;
        if(n==1){
            convexhull.push_back(p.front());
            convexhull.push_back(p.front());
            return;
        }
        vector<int> st(2*n+5,0);
        vector<bool> used(n,0);
        int tp=0;
        st[tp]=0;
        for(int i=1;i<n;i++){
            while(tp>=2&&((p[st[tp]]-p[st[tp-1]])&(p[i]-p[st[tp]]))<=0){
                used[st[tp--]]=0;
            }
            used[i]=1;
            st[tp]=i;
        }
        int tmp=tp;//下凸壳大小
        for(int i=n-2;i>=0;i--){
            if(!used[i]){
                while(tp>tmp&&((p[st[tp]]-p[st[tp-1]])&(p[i]-p[st[tp]]))<=0){
                    used[st[tp--]]=0;
                }
                used[i]=1;
                st[tp]=i;
            }
        }
        for(int i=1;i<=tp;i++){
            convexhull.push_back(p[st[i]]);
        }
    }
    double getPerimeter(){
        double ans=0;
        for(int i=1;i<convexhull.size();i++){
            ans+=convexhull[i].dis(convexhull[i-1]);
        }
        return ans;
    }
}

```

```

    double getArea(){
        if(convexhull.size()<4) return 0;
        double ans=0;
        for(int i=1;i<convexhull.size()-2;i++){
            ans+=(convexhull[i]-convexhull[0])^(convexhull[i+1]-convexhull[0])/2;
        }
        return ans;
    }

    //旋转卡壳求直径
    double getLongest(){
        if(convexhull.size()<4){
            return convexhull[0].dis(convexhull[1]);
        }
        int j=0;
        const int sz=convexhull.size();
        double ans=0;
        for(int i=0;i<convexhull.size()-1;i++){
            Line line(convexhull[i],convexhull[i+1],1);
            while(line.distancetopoint(convexhull[j])
<=line.distancetopoint(convexhull[(j+1)%sz])){
                j=(j+1)%sz;
            }
            ans=max({ans,(convexhull[i]-convexhull[j]).norm(),(convexhull[i+1]-
convexhull[j]).norm()});
        }
        return ans;
    }

    //旋转卡壳最小矩形覆盖
    pair<double,vector<Point>> minRectangleCover(){
        vector<Point> p;
        if(convexhull.size()<4) return {0,p};
        int j=1,l=1,r=1;
        double ans=1e18;
        const int sz=convexhull.size();
        for(int i=1;i<convexhull.size();i++){
            Line line(convexhull[i-1],convexhull[i],1);
            while(line.distancetopoint(convexhull[j])
<=line.distancetopoint(convexhull[(j+1)%sz])){
                j=(j+1)%sz;
            }
            while((convexhull[i]-convexhull[i-1])*(convexhull[(r+1)%sz]-
convexhull[i-1])>=(convexhull[i]-convexhull[i-1])*(convexhull[r]-convexhull[i-
1])){
                r=(r+1)%sz;
            }
            if(i==1) l=r;
            while((convexhull[i-1]-convexhull[i])*(convexhull[(l+1)%sz]-
convexhull[i])>=(convexhull[i-1]-convexhull[i])*(convexhull[l]-convexhull[i])){
                l=(l+1)%sz;
            }
            Point t1=convexhull[i]-convexhull[i-1];
            Point t2=convexhull[r]-convexhull[i];
            Point t3=convexhull[l]-convexhull[i-1];
            double a=line.distancetopoint(convexhull[j]);
            double b=t1.norm()+t1*t2/t1.norm()-t1*t3/t1.norm();
            double tmp=a*b;
        }
    }

```

```

        if(ans>tmp){
            ans=tmp;
            p.clear();
            p.push_back(t1*((t1*t3)/(t1.norm()*t1.norm())+convexhull[i-1]));
            p.push_back(t1*(1+(t1*t2)/(t1.norm()*t1.norm())+convexhull[i-1]));
            Point tmp=Point{-(p[1]-p[0]).y,(p[1]-p[0]).x}*a/b;
            p.push_back(tmp+p[1]);
            p.push_back(tmp+p[0]);
        }
    }
    return {ans,p};
}
};


```

```

// by trilliverse
using ll = long long;
using ld = long double;

const double eps = 1e-8; // 视情况调整eps
const double inf = 1e20;
const double pi = acos(-1.0);

int sgn(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    else return 1;
}

struct Point {
    double x, y;
    Point() {}
    Point(double _x, double _y) : x(_x), y(_y) {}

    void input() { cin >> x >> y; }
    void output() { cerr << fixed << setprecision(10) << x << " " << y << '\n'; }

    bool operator==(Point b) const { return sgn(x - b.x) == 0 && sgn(y - b.y) == 0; }
    bool operator<(Point b) const { return sgn(x-b.x)==0?sgn(y-b.y)<0:sgn(x-b.x)<0; }
    Point operator+(const Point &b) const { return Point(x + b.x, y + b.y); }
    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); }
    Point operator*(const double &k) const { return Point(x * k, y * k); }
    Point operator/(const double &k) const { return Point(x / k, y / k); }

    double operator*(const Point &b) const { return x * b.x + y * b.y; } // 点乘
    double operator^(const Point &b) const { return x * b.y - y * b.x; } // 叉乘

    // 象限
    int quad() const {
        if (x > 0 && y >= 0) return 1;
        if (x <= 0 && y > 0) return 2;
        if (x < 0 && y <= 0) return 3;
        if (x >= 0 && y < 0) return 4;
    }
};


```

```

        return 5;
    }
    double len() { return hypot(x, y); }      // 到原点的距离
    double len2() { return x * x + y * y; }   // 到原点距离平方
    double dis(Point b) { return hypot(x - b.x, y - b.y); } // 两点间距离(可能会被
卡精度!)
    double dis2(Point b) { return (x - b.x) * (x - b.x) + (y - b.y) * (y - b.y); }
}
// pa ^ pb
double cross(Point a, Point b) {
    Point p = {x, y};
    return (a - p) ^ (b - p);
}
// pa * pb
double dot(Point a, Point b) {
    Point p = {x, y};
    return (a - p) * (b - p);
}
// 点p看a,b所成夹角 弧度制
double rad(Point a, Point b) {
    Point p = *this;
    return fabs(atan2(fabs((a - p) ^ (b - p)), (a - p) * (b - p)));
}
// 角度制
double ang(Point a, Point b) {
    return rad(a, b) / pi * 180.0;
}
// 考虑叉乘方向的夹角
double drad(Point a, Point b) {
    Point p = *this;
    double angle = p.rad(a, b); // 计算夹角大小 [0, π]
    // 根据叉积符号确定方向
    if (sgn(cross(a, b)) < 0) {
        angle = 2 * pi - angle;
    }
    return angle;
}
Point rotleft() { return Point(-y, x); } // 逆时针旋转90
Point rotright() { return Point(y, -x); } // 顺时针旋转90
// 绕着点p逆时针旋转angle弧度
Point rotate(Point p, double angle) {
    Point v = (*this) - p;
    double c = cos(angle);
    double s = sin(angle);
    return Point(p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
}
// 化为长度为r的向量
Point trunc(double r) {
    double l = len();
    if (!sgn(l))
        return *this;
    r /= l;
    return Point(x * r, y * r);
}
};


```

```

struct Line {
    Point u, v;
    Line() {}
    Line(Point _u, Point _v) : u(_u), v(_v) {}
    bool operator==(Line l) { return (u == l.u) && (v == l.v); }

    // 点和倾斜角 $\alpha \in [0, \pi)$ 
    Line(Point p, double angle) {
        u = p;
        if (sgn(angle - pi / 2) == 0)
            v = u + Point(0, 1); // 斜率不存在
        else
            v = u + Point(1, tan(angle));
    }
    //  $ax+by+c=0$ 
    Line(double a, double b, double c) {
        if (sgn(a) == 0) {
            u = Point(0, -c / b);
            v = Point(1, -c / b);
        } else if (sgn(b) == 0) {
            u = Point(-c / a, 0);
            v = Point(-c / a, 1);
        } else {
            u = Point(0, -c / b);
            v = Point(1, (-c - a) / b);
        }
    }
}

void input() { u.input(), v.input(); }
void adjust() {
    if (v < u)
        swap(u, v);
}
double len() { return u.dis(v); } // 线段长度
// 方向向量
Point dir() { return v - u; }
// 直线倾斜角  $\alpha \in [0, \pi)$  弧度制
double rad() {
    double theta = atan2(v.y - u.y, v.x - u.x);
    if (sgn(theta) < 0) theta += pi;
    if (sgn(theta - pi) == 0) theta -= pi;
    return theta;
}
// 直线倾斜角 角度制
double rad(bool f) { return rad() * 180 / pi; }
// 点和直线(线段)关系
// 1:左侧 2:右侧 3:线上 4:线段前 5:线段后
int relation(Point p) {
    int c = sgn((p - u) ^ (v - u));
    if (c < 0) return 1;
    else if (c > 0) return 2;
    else { // on (line or segment)
        if (sgn((p - u) * (p - v)) <= 0) return 3; // on segment
        if (sgn((p - u) * (v - u)) > 0) return 4; // on line front
        else return 5; // on line back
    }
}

```

```

}

// 直线与直线关系
// 0:平行 1:重合 2:正交 3:其他
int linecrossline(Line l) {
    if (sgn((v - u) * (l.v - l.u)) == 0) return 2;
    // 叉乘为0 → 共线向量(平行or重合)
    if (sgn((v - u) ^ (l.v - l.u)) == 0) return l.relation(u) > 2;
    return 3;
}

// 线段与线段关系
// 2 规范相交 1 非规范相交 0 不相交
int segcrossseg(Line l) {
    int d1 = sgn((v - u) ^ (l.u - u));
    int d2 = sgn((v - u) ^ (l.v - u));
    int d3 = sgn((l.v - l.u) ^ (u - l.u));
    int d4 = sgn((l.v - l.u) ^ (v - l.u));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
    return (d1 == 0 && sgn((l.u - u) * (l.u - v)) <= 0) ||
           (d2 == 0 && sgn((l.v - u) * (l.v - v)) <= 0) ||
           (d3 == 0 && sgn((u - l.u) * (u - l.v)) <= 0) ||
           (d4 == 0 && sgn((v - l.u) * (v - l.v)) <= 0);
}

// 直线与线段关系
// *this:line; seg:segment
// 2 规范相交 1 非规范相交 0 不相交
int linecrossseg(Line seg) {
    int d1 = sgn((v - u) ^ (seg.u - u));
    int d2 = sgn((v - u) ^ (seg.v - u));
    if ((d1 ^ d2) == -2) return 2;
    return (d1 == 0 || d2 == 0);
}

// 求两直线交点 (要保证两直线不平行或重合)
Point crosspoint(Line l) {
    double a1 = (l.v - l.u) ^ (u - l.u);
    double a2 = (l.v - l.u) ^ (v - l.u);
    Point p = {(u.x * a2 - v.x * a1) / (a2 - a1), (u.y * a2 - v.y * a1) / (a2 - a1)};
    p.x = fabs(p.x) < eps ? 0 : p.x; // 可能会输出-0.00 特判
    p.y = fabs(p.y) < eps ? 0 : p.y;
    return p;
}

// 点到直线的距离
double dispointtoline(Point p) {
    return fabs((p - u) ^ (v - u)) / len();
}

// 点到线段的距离
double dispointtoseg(Point p) {
    if (sgn((p - u) * (v - u)) < 0 || sgn((p - v) * (u - v)) < 0)
        return min(p.dis(u), p.dis(v));
    return dispointtoline(p);
}

// 线段到线段的距离
double dissegstoseg(Line l) {
    if (segcrossseg(l))

```

```

        return 0; // 线段相交，距离就是0
        return min(min(dispointtoseg(l.u), dispointtoseg(l.v)),
                   min(l.dispointtoseg(u), l.dispointtoseg(v)));
    }

// 点p在直线上的投影
Point project(Point p) {
    return u + (((v - u) * ((v - u) * (p - u))) / ((v - u).len2()));
    // return p*
}
// 点p关于直线的对称点
Point symmetry(Point p) {
    Point q = project(p); // 连线中点
    return Point(2 * q.x - p.x, 2 * q.y - p.y);
}
};

struct Polygon {
    int n;
    vector<Point> p;
    vector<Line> l;
    Polygon() : n(0) {}
    Polygon(int _n) {
        n = _n;
        p.resize(n);
        l.resize(n);
    }

    void input() {
        for (int i = 0; i < n; i++) p[i].input();
    }
    void output() { // debug
        cerr << "n=" << n << '\n';
        for (int i = 0; i < n; i++) p[i].output();
    }
    void add(Point q) {
        p.push_back(q);
        n++;
    }
    void getline() {
        for (int i = 0; i < n; i++) {
            l[i] = Line(p[i], p[(i + 1) % n]);
        }
    }
    // 周长
    double getCircumference() {
        double sum = 0;
        for (int i = 0; i < n; i++) sum += p[i].dis(p[(i + 1) % n]);
        return sum;
    }
    // 面积（三角剖分）
    double getArea() {
        double sum = 0;
        for (int i = 0; i < n; i++) sum += (p[i] ^ p[(i + 1) % n]);
        return fabs(sum) / 2;
    }
}

```

```

// 重心 UVA 10002
Point getBarycentre() {
    Point ret(0, 0);
    double area = 0;
    for (int i = 1; i < n - 1; i++) {
        double tmp = (p[i] - p[0]) ^ (p[i + 1] - p[0]);
        if (sgn(tmp) == 0) continue;
        area += tmp;
        ret.x += (p[0].x + p[i].x + p[i + 1].x) / 3 * tmp;
        ret.y += (p[0].y + p[i].y + p[i + 1].y) / 3 * tmp;
    }
    if (sgn(area)) ret = ret / area;
    return ret;
}

// 极角排序
void norm() {
    sort(p.begin(), p.end(), [] (const Point &a, const Point &b) {
        if (a.quad() != b.quad()) return a.quad() < b.quad();
        return (a ^ b) > 0;
    });
}

// Andrew得到凸包 点编号0~n-1
void getConvex(Polygon &convex) {
    vector<int> st(2 * n + 5, 0);
    vector<bool> used(n, 0);
    int top = 0;
    sort(p.begin(), p.end());
    st[++top] = 0;
    for (int i = 1; i < n; i++) {
        while (top >= 2 && ((p[st[top]] - p[st[top - 1]]) ^ (p[i] - p[st[top]])) <= 0) {
            used[st[top--]] = 0;
        }
        used[i] = 1;
        st[++top] = i;
    }
    int tmp = top; // 下凸壳大小
    for (int i = n - 2; i >= 0; i--) {
        if (!used[i]) {
            while (top > tmp && ((p[st[top]] - p[st[tmp]])) ^ (p[i] - p[st[top]])) <= 0) {
                used[st[top--]] = 0;
            }
            used[i] = 1;
            st[++top] = i;
        }
    }
    if (top == 1) convex.add(p[st[1]]); // 特判只有1个点
    else {
        for (int i = 1; i < top; i++) {
            convex.add(p[st[i]]);
            if (top - 1 == 2 && p[st[i]] == p[st[i + 1]]) break; // 特判所有都
        }
    }
}

```

为重点

```

bool isConvex() {
    bool s[2];
    memset(s, false, sizeof(s));
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sgn((p[j] - p[i]) ^ (p[k] - p[i])) + 1] = true;
        if (s[0] && s[2]) return false;
    }
    return true;
}

// 点q在多边形内的关系 0:外部 1:内部 2:边上 3:点上
int relationpoint(Point q) {
    for (int i = 0; i < n; i++) {
        if (p[i] == q)
            return 3;
    }
    getline();
    for (int i = 0; i < n; i++) {
        if (l[i].pointonseg(q))
            return 2;
    }
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = sgn((q - p[j]) ^ (p[i] - p[j]));
        int u = sgn(p[i].y - q.y);
        int v = sgn(p[j].y - q.y);
        if (k > 0 && u < 0 && v >= 0)
            cnt++;
        if (k < 0 && v < 0 && u >= 0)
            cnt--;
    }
    return cnt != 0;
}

// 旋转卡壳求凸包直径
double getDiameter() {
    if (n == 1) return 0.0;
    if (n == 2) return p[0].dis(p[1]);
    // assert(n >= 3);
    double ans = 0.0;
    int j = 1;
    for (int i = 0; i < n; i++) {
        while (fabs((p[(i + 1) % n] - p[i]) ^ (p[(j + 1) % n] - p[i])) >
               fabs((p[(i + 1) % n] - p[i]) ^ (p[j] - p[i]))) {
            j = (j + 1) % n;
        }
        ans = max({ans, p[i].dis(p[j]), p[(i + 1) % n].dis(p[(j + 1) % n])});
    }
    return ans;
}

// 旋转卡壳求最小矩形覆盖
// 必须是凸包(逆时针) UVA 10173
double minRectangleCover() {
    if (n <= 2) return 0.0;

```

```

        double ans = -1;
        int j = 1, r = 1, l;
        for (int i = 0; i < n; i++) {
            // 卡出离边p[i]~p[i+1]最远点
            while (sgn(p[i].cross(p[(i + 1) % n], p[(j + 1) % n])
                    - p[i].cross(p[(i + 1) % n], p[j])) >= 0) {
                j = (j + 1) % n;
            }
            // 卡出边p[i]~p[i+1]方向上正向最远点
            while (sgn(p[i].dot(p[(i + 1) % n], p[(r + 1) % n])
                    - p[i].dot(p[(i + 1) % n], p[r])) >= 0) {
                r = (r + 1) % n;
            }
            if (i == 0) l = r;
            // 卡出边p[i]~p[i+1]方向上负向最远点
            while (sgn(p[i].dot(p[(i + 1) % n], p[(l + 1) % n])
                    - p[i].dot(p[(i + 1) % n], p[l])) <= 0) {
                l = (l + 1) % n;
            }
            double d = (p[i] - p[(i + 1) % n]).len2();
            double tmp = p[i].cross(p[(i + 1) % n], p[j])
                * (p[i].dot(p[(i + 1) % n], p[r]) - p[i].dot(p[(i + 1) % n],
                p[l])) / d;
            if (ans < 0 || ans > tmp) ans = tmp;
        }
        return ans;
    }

    // 求凸包最小内角
    double minAngle() {
        if (n <= 2) return 0.0;
        double minn = 200.0;
        for (int i = 0; i < n; i++) {
            // ang 角度制; rad 弧度制
            minn = min(minn, p[i].ang(p[(i + n - 1) % n], p[(i + 1) % n]));
        }
        return minn;
    }
};

struct halfplane : public Line {
    double angle;
    halfplane() {}
    //`表示向量u->v逆时针(左侧)的半平面`
    halfplane(Point _u, Point _v) {
        u = _u;
        v = _v;
    }
    halfplane(Line l) {
        u = l.u;
        v = l.v;
    }
    void calc_angle() {
        angle = atan2(v.y - u.y, v.x - u.x);
    }
    bool operator<(const halfplane &b) const {
        return angle < b.angle;
    }
};

```

```

    }

};

const int maxp = 1005;
struct halfplanes {
    int n;
    // halfplane hp[maxp];
    vector<halfplane> hp;
    Point p[maxp];
    int que[maxp];
    int st, ed;

    void add(halfplane l) {
        hp.push_back(l);
        n++;
    }

    // 去重
    void unique() {
        int m = 1;
        for (int i = 1; i < n; i++) {
            if (sgn(hp[i].angle - hp[i - 1].angle) != 0)
                hp[m++] = hp[i];
            else if (sgn((hp[m - 1].v - hp[m - 1].u) ^ (hp[i].u - hp[m - 1].u)) >
0)
                hp[m - 1] = hp[i];
        }
        n = m;
    }

    bool halfplaneinsert() {
        for (int i = 0; i < n; i++) hp[i].calc_angle();
        sort(hp.begin(), hp.end());
        unique();
        que[st = 0] = 0;
        que[ed = 1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for (int i = 2; i < n; i++) {
            while (st < ed && sgn(hp[i].u.cross(hp[i].v, p[ed])) < 0) ed--;
            while (st < ed && sgn(hp[i].u.cross(hp[i].v, p[st + 1])) < 0) st++;
            que[++ed] = i;
            if (hp[i].parallel(hp[que[ed - 1]])) return false;
            p[ed] = hp[i].crosspoint(hp[que[ed - 1]]);
        }
        while (st < ed && sgn(hp[que[st]].u.cross(hp[que[st]].v, p[ed])) < 0) ed--;
        while (st < ed && sgn(hp[que[st]].u.cross(hp[que[st]].v, p[st + 1])) < 0)
st++;
        if (st + 1 >= ed) return false;
        return true;
    }

    // 得到最后半平面交得到的凸多边形
    // 需要先调用halfplaneinsert() 且返回true
    void getConvex(Polygon &convex) {
        p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
    }
}

```

```

        convex.n = ed - st + 1;
        convex.p.resize(n);
        for (int j = st, i = 0; j <= ed; i++, j++) convex.p[i] = p[j];
    }
};

```

三维几何

```

const double EPS = 1e-8;
const double PI = acos(-1.0);

int sgn(double x) {
    // return (x > EPS) - (x < -EPS);
    if (fabs(x) < EPS) return 0;
    return x < 0 ? -1 : 1;
}

template <typename T>
struct Point3 {
    T x, y, z;
    constexpr Point3(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}

    constexpr Point3 operator+(const Point3& p) const { return
Point3(x+p.x,y+p.y,z+p.z); }
    constexpr Point3 operator-(const Point3& p) const { return Point3(x - p.x, y
- p.y, z - p.z); }
    constexpr Point3 operator*(const T& k) const { return Point3(x * k, y * k, z
* k); }
    constexpr Point3 operator/(const T& k) const { return Point3(x / k, y / k, z
/ k); }
    Point3& operator+=(const Point3& p) { x += p.x; y += p.y; z += p.z; return
*this; }
    Point3& operator-=(const Point3& p) { x -= p.x; y -= p.y; z -= p.z; return
*this; }
    Point3& operator*=(const T& k) { x *= k; y *= k; z *= k; return *this; }
    Point3& operator/=(const T& k) { x /= k; y /= k; z /= k; return *this; }

    // dot product and cross product
    constexpr T operator*(const Point3& p) const { return x * p.x + y * p.y + z *
p.z; }
    constexpr Point3 operator^(const Point3& p) const {
        return Point3(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
    }

    bool operator==(const Point3& p) const {
        return sgn(x - p.x) == 0 && sgn(y - p.y) == 0 && sgn(z - p.z) == 0;
    }
    bool operator!=(const Point3& p) const { return !(*this == p); }
    bool operator<(const Point3& p) const {
        if (sgn(x - p.x) != 0) return x < p.x;
        if (sgn(y - p.y) != 0) return y < p.y;
        return sgn(z - p.z) < 0;
    }

    constexpr T dis2(const Point3& p) const {

```

```

        return (x - p.x) * (x - p.x) + (y - p.y) * (y - p.y) + (z - p.z) * (z -
p.z);
    }

    auto dis(const Point3& p) const {
        if constexpr (std::is_same_v<T, long double>) return sqrtl(dis2(p));
        else return sqrt(dis2(p));
    }

    T len2() const { return x * x + y * y + z * z; }
    auto len() const {
        if constexpr (std::is_same_v<T, long double>) return sqrtl(len2());
        else return sqrt(len2());
    }

    // angle (radians or degrees) <APB
    static double angle(const Point3& a, const Point3& p, Point3& b, bool rad =
true) {
        // PA*PB=|PA||PB|cos(theta)
        double costh = (a - p) * (b - p) / (p.dis(a) * p.dis(b));
        costh = std::clamp(costh, -1.0, 1.0); // avoid precision issues
        if (rad) return acos(costh);
        return acos(costh) * 180.0 / PI; // convert to degrees
    }

    // normalize the vector to length k
    Point3 norm(double k = 1.0) const {
        double l = len();
        if (!sgn(l)) return *this; // origin
        k /= l;
        return Point3(x * k, y * k, z * k);
    }

    // IO
    friend std::istream& operator>>(std::istream& is, Point3& p) {
        return is >> p.x >> p.y >> p.z;
    }
    friend std::ostream& operator<<(std::ostream& os, const Point3& p) {
        return os << p.x << ' ' << p.y << ' ' << p.z;
    }
};

using Point = Point3<double>;
// using Point = Point3<long double>;
// using Point = Point3<long long>;

```

博弈论

SG函数

SG(x)， x 是游戏的状态， $SG=0$ ，先手必败，否则先手必胜

设后继状态为 a_1, a_2, \dots, a_p ， $SG(x)=\text{mex}(SG(a_1), SG(a_2), \dots)$

一个游戏的SG函数值等于各个游戏SG函数值的nim和（异或和）

dp

数位dp

给定区间[l,r]，问区间满足条件的数有多少个，cal(r)-cal(l-1)

windy数（不含前导0且相邻两个数字之差至少为2），求a到b中有多少个windy数

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
//pos:当前位置 limit填数的限制 pre上一个数 lead前面是否是前导0
vector<vector<int>> dp(15,vector<int>(10,-1));
vector<int> v;
int dfs(int pos,int pre,bool lead,bool limit){
    if(pos<0) return 1;
    if(!limit&&!lead&&dp[pos][pre] != -1){
        return dp[pos][pre];
    }
    int ans=0,up=limit?v[pos]:9;
    for(int i=0;i<=up;i++){
        if(lead){
            ans+=dfs(pos-1,i,i==0,limit&&i==up);
        }else{
            if(abs(i-pre)<2) continue;
            ans+=dfs(pos-1,i,0,limit&&i==up);
        }
    }
    return (!lead&&!limit)?dp[pos][pre]=ans:ans;
}
//计算
int cal(int x){
    v.clear();
    if(x==0) return 1;
    while(x){
        v.push_back(x%10);
        x/=10;
    }
    return dfs(v.size()-1,0,1,1);
}
void solve(){
    int a,b;
    cin>>a>>b;
    cout<<cal(b)-cal(a-1)<<"\n";
}
signed main(){
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}
```

随机算法

模拟退火

T : 温度

ΔT : 温度变化率, 每次温度等于上一次 $T * \Delta T$, 一般取0.95-0.99, 模拟徐徐降温

x : 当前选择的解

Δx : 解变动量

x_1 : 当前的目标解, 等于 $x + \Delta x$

Δf : 当前解的函数值与目标解函数值的差值, 等于 $f(x) - f(x_1)$

每次的 Δx 在一个大小与 T 成正比的值域内随机取值。如果 $f(x_1) < f(x)$, 那么接受目标解 $x = x_1$, 如果 $f(x_1) > f(x)$, 则以一定的概率接受, 概率是 $e^{\frac{-\Delta f}{T}}$, 直到 T 趋近于0, 循环结束

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define double long double
struct edge{
    int x,y,w;
};

void solve(){
    int n;
    cin>>n;
    vector<edge> v(n);
    double xans=0,yans=0;
    for(int i=0;i<n;i++){
        cin>>v[i].x>>v[i].y>>v[i].w;
        xans+=v[i].x;
        yans+=v[i].y;
    }
    double ans=1e18;//全局能量最小值
    auto energy=[&](double x,double y){
        double res=0;
        for(int i=0;i<n;i++){
            pair<double,double> p={v[i].x-x,v[i].y-y};
            res+=sqrtl(p.first*p.first+p.second*p.second)*v[i].w;
        }
        return res;
    }; //计算能量, 能量越低越符合要求
    auto sa=[&](){
        double t=3000,down=0.997;//初始温度, 降温系数
        double x=xans,y=yans;//当前值
        while(t>1e-14){ //继续降温
            double xtmp=x+(rand()*2-RAND_MAX)*t;//新的值
            double ytmp=y+(rand()*2-RAND_MAX)*t;
            double newenergy=energy(xtmp,ytmp); //新的能量
            double delta=newenergy-ans;//新能量和全局最优能量的差值
            if(delta<0){ //如果更优, 接受, 修改全局最优能量, 全局最优答案和当前值
                xans=x=xtmp;
```

```

        yans=y=ytmp;
        ans=newenergy;
    }else if(exp(-delta/t)*RAND_MAX>rand()){//否则, 以一定概率接受
        x=xtmp;//修改当前值
        y=ytmp;
    }
    t*=down;//降温
}
}//模拟退火
xans/=n;
yans/=n;
ans=energy(xans,yans);//初始的一个答案
while((double)clock()/CLOCKS_PER_SEC<0.8) sa();//一直多次模拟退火, 直到快超时
cout<<fixed<<setprecision(3)<<xans<<" "<<yans<<"\n";
}

signed main(){
    srand(time(0));
    cin.tie(nullptr)->sync_with_stdio(0);
    int t=1;
    //cin>>t;
    while(t--) solve();
    return 0;
}

```

平板电视

```
#include<bits/extc++.h>
using namespace __gnu_pbds;
```

拉链法哈希

```
cc_hash_table <int,int> f;
```

红黑树

```
__gnu_pbds::tree<Key, Mapped, Cmp_Fn = std::less<Key>, Tag = rb_tree_tag,
    Node_Update = null_tree_node_update,
    Allocator = std::allocator<char> >
//如果要用order_of_key或find_by_order, node_update需要使用
tree_order_statistics_node_update
```

```
__gnu_pbds::tree<std::pair<int, int>, __gnu_pbds::null_type,
    std::less<std::pair<int, int> >, __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update>
trr;
```

insert(x): 向树中插入一个元素 x, 返回 std::pair<point_iterator, bool>。

erase(x): 从树中删除一个元素/迭代器 x, 返回一个 bool 表明是否删除成功。

order_of_key(x): 返回 x 以 Cmp_Fn 比较的排名, 0开始。

find_by_order(x): 返回 Cmp_Fn 比较的排名所对应元素的迭代器。

lower_bound(x): 以 Cmp_Fn 比较做 lower_bound, 返回迭代器。
upper_bound(x): 以 Cmp_Fn 比较做 upper_bound, 返回迭代器。
join(x): 将 x 树并入当前树, 前提是两棵树的类型一样, x 树被删除。
split(x,b): 以 Cmp_Fn 比较, 小于等于 x 的属于当前树, 其余的属于 b 树。
empty(): 返回是否为空。
size(): 返回大小。

元素不可重, 可以使用pair<int,int>加入时间戳来使元素可重

```
//插入
tr.insert({x,++cnt});
//删除
auto it=tr.lower_bound({x,0});
if(it!=tr.end()) tr.erase(it);
//查找元素排名
tr.order_of_key({x,0})+1;
//根据排名查找元素
auto it=tr.find_by_order(x-1);
if(it==tr.end()) continue;
it->first;
//前驱
auto it=tr.lower_bound({x,0});
if(it==tr.begin()) continue;
it--;
it->first;
//后继
auto it=tr.upper_bound({x,INF});
if(it==tr.end()) continue;
it->first;
```

快读快写

```
void get(int &x){
    x=0;
    char ch=getchar();
    while(ch<'0' || ch>'9'){
        ch=getchar();
    }
    while(ch>='0' && ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
}
void print(__int128 x){
    if(x==0){
        putchar('0');
        return;
    }
    stack<char> st;
    while(x){
        st.push(x%10+'0');
        x/=10;
    }
    while(!st.empty()){

    }
}
```

```
    putchar(st.top());
    st.pop();
}
}
```

火车头

```
#define fastcall __attribute__((optimize("-O3")))
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

