Competitive Programming Library

Rathijit Paul

September 27, 2019

Contents

1 Data Structure

1.1 Fenwick Tree (1D)

```
struct BIT{
    vll bit1;
    vll bit2;
    ll N;

    BIT(ll n)
    {
        N=n;
        bit1.assign(n+1,0);
        bit2.assign(n+1,0);
    }

    BIT(vll v) : BIT(v.size())
    {
        for(ll i=0;i<v.size();i++)
            add(i+1, i+1, v[i]);
    }

    ll sum1(ll idx)
    {
        ll ret=0;
        for (; idx > 0; idx -= idx & -idx)
            ret += bit1[idx];
        return ret;
    }

    ll sum2(ll idx)
    {
        ll ret=0;
        for (; idx > 0; idx -= idx & -idx)
            ret += bit2[idx];
        return ret;
    }

    ll prefix_sum(ll idx)
    {
        return sum1(idx)*idx -  sum2(idx);
    }

    ll sum(ll l, ll r)
    {
```

```
        return prefix_sum(r) - prefix_sum(l-1);
    }

    void add1(ll idx, ll val)
    {
        for (; idx <= N; idx += idx & -idx)
            bit1[idx] += val;
    }

    void add2(ll idx, ll val)
    {
        for (; idx <= N; idx += idx & -idx)
            bit2[idx] += val;
    }

    void add(ll l, ll r, ll val)
    {
        add1(l,val);
        add1(r+1,-val);
        add2(l,val*(l-1));
        add2(r+1,-val*r);
    }
};


int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    cin>> n;
    vll v(n);
    for(i=0;i<n;i++) cin>> v[i];

    BIT bit(v);

    cin>> m;
    while(m--){
        cin>> t;
        if(t==1){
            cin>> a >> b;
            cout<< "Sum:␣" << bit.sum(a,b) << "\n";
        }
        else{
            cin>> a >> b >> c;
            bit.add(a,b,c);
```

```
        }
    }
    return 0;
}
```

1.2 Fenwick Tree (2D)

```
struct BIT_2D {
    vvl bit;
    ll n, m;

    BIT_2D(ll N, ll M)
    {
        n=N,m=M;
        vll v;
        v.assign(m,0);
        bit.assign(n,v);
    }

    BIT_2D(vvl val) : BIT_2D(val.size(), val[0].size())
    {
        for(ll i=0;i<val.size();i++)
            for(ll j=0;j<val[i].size();j++)
                add(i,j,val[i][j]);
    }

    ll sum(ll x, ll y)
    {
        ll ret = 0;
        for (ll i = x; i >= 0; i = (i & (i + 1)) - 1)
            for (ll j = y; j >= 0; j = (j & (j + 1)) - 1)
                ret += bit[i][j];
        return ret;
    }

    void add(ll x, ll y, ll delta)
    {
        for (ll i = x; i < n; i = i | (i + 1))
            for (ll j = y; j < m; j = j | (j + 1))
                bit[i][j] += delta;
    }
};

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    cin>> n >> m >> t;
    vvl val(n);
```

```
    for(i=0;i<n;i++){
        vll v(m);
        for(j=0;j<m;j++)
            cin>> v[j];
        val[i]=v;
    }

    BIT_2D bit(val);

    while(t--){
        cin>> cs;
        if(cs==1){
            cin>> a >> b >> c;
            a--,b--;
            bit.add(a,b,c);
        }
        else{
            cin>> a >> b >> i >> j;
            a--,b--,i--,j--;

            ll x=0,res;

            res=bit.sum(i,j);
            if(b-1>=0)
                res-=bit.sum(i,b-1);
            if(a-1>=0)
                res-=bit.sum(a-1,j);
            if(a-1>=0 and b-1>=0)
                res+=bit.sum(a-1,b-1);

            cout<< res << "\n";
        }
    }

    return 0;
}
```

1.3 Segment Tree (1D)

```
class Segment_Tree{
    ll N;
    struct Node{
        ll prop,sum;

        Node() {}

        Node(ll p, ll s)
        {
            prop=p,sum=s;
        }
    };

    vector<Node>val;

public:

    Segment_Tree(ll n)
    {
        Node nd(0,0);
        N=(n+5)*4;
        val.assign(N,nd);
    }

    Segment_Tree(vll arr) : Segment_Tree(arr.size())
    {
        for(ll i=0;i<arr.size();i++)
            update(1,1,N,i+1,i+1,arr[i]);
    }

    void update(ll id, ll b, ll e, ll i, ll j, ll x)
    {
        if (i > e || j < b)
            return;
        if (b >= i && e <= j)
        {
            val[id].sum += ((e - b + 1) * x);
            val[id].prop += x;
            return;
        }

        ll Left = id << 1;
        ll Right = (id << 1) + 1;
```

```
        ll mid = (b + e) >> 1;

        update(Left, b, mid, i, j, x);
        update(Right, mid + 1, e, i, j, x);
        val[id].sum = val[Left].sum + val[Right].sum + (e - b + 1)*val[id].pr
    }

    ll query(ll id, ll b, ll e, ll i, ll j, ll carry = 0)
    {
        if (i > e || j < b)
            return 0;

        if (b >= i and e <= j)
            return val[id].sum + carry * (e - b + 1);

        ll Left = id << 1;
        ll Right = (id << 1) + 1;
        ll mid = (b + e) >> 1;

        ll p1 = query(Left, b, mid, i, j, carry + val[id].prop);
        ll p2 = query(Right, mid + 1, e, i, j, carry + val[id].prop);

        return p1 + p2;
    }

};
```

1.4 Segment Tree (2D)

```
ll tree[4*MX][4*MX],val[MX][MX];

void buildY(ll nodex, ll lx, ll rx, ll nodey, ll ly, ll ry, ll n, ll m)
{
    if(ly==ry){
        if(lx==rx)
            tree[nodex][nodey]=val[lx][ly];
        else
            tree[nodex][nodey]=tree[nodex*2][nodey]+tree[nodex*2+1][nodey];
    }
    else{
        ll midy=(ly+ry)/2;
        buildY(nodex,lx,rx,nodey*2,ly,midy,n,m);
        buildY(nodex,lx,rx,nodey*2+1,midy+1,ry,n,m);
        tree[nodex][nodey]=tree[nodex][nodey*2]+tree[nodex][nodey*2+1];
    }
}

void buildX(ll nodex, ll lx, ll rx, ll n, ll m)
{
    if(lx!=rx){
        ll midx=(lx+rx)/2;
        buildX(nodex*2,lx,midx,n,m);
        buildX(nodex*2+1,midx+1,rx,n,m);
    }
    buildY(nodex,lx,rx,1,1,m,n,m);
}

ll sumY(ll nodex, ll nodey, ll tly, ll try1, ll ly, ll ry, ll n, ll m)
{
    if (ly > ry)
        return 0;
    if (ly==tly and try1==ry)
        return tree[nodex][nodey];

    ll midy = (tly + try1)/2;//if(!midy)midy++;
    return sumY(nodex, nodey*2, tly, midy, ly, min(ry, midy), n , m) + sumY(n
}

ll sumX(ll nodex, ll tlx, ll trx, ll lx, ll rx, ll ly, ll ry, ll n, ll m)
{
    if(lx>rx)
        return 0;
```

```
    if(lx==tlx and rx==trx)
        return sumY(nodex, 1, 1, m, ly, ry, n, m);

    ll midx = (tlx+trx)/2;//if(!midx)midx++;
    return sumX(nodex*2,tlx,midx,lx,min(rx,midx),ly,ry,n,m) + sumX(nodex*2+1,
}

void updateY(ll nodex, ll lx, ll rx, ll nodey, ll ly, ll ry, ll x, ll y, ll n
{
    if (ly == ry) {
        if (lx == rx)
            tree[nodex][nodey] = new_val;
        else
            tree[nodex][nodey] = tree[nodex*2][nodey] + tree[nodex*2+1][nodey
    } else {
        ll my = (ly + ry) / 2;
        if (y <= my)
            updateY(nodex, lx, rx, nodey*2, ly, my, x, y, new_val, n, m);
        else
            updateY(nodex, lx, rx, nodey*2+1, my+1, ry, x, y, new_val, n, m);
        tree[nodex][nodey] = tree[nodex][nodey*2] + tree[nodex][nodey*2+1];
    }
}

void updateX(ll nodex, ll lx, ll rx, ll x, ll y, ll new_val, ll n, ll m)
{
    if (lx != rx) {
        ll mx = (lx + rx) / 2;
        if (x <= mx)
            updateX(nodex*2, lx, mx, x, y, new_val, n, m);
        else
            updateX(nodex*2+1, mx+1, rx, x, y, new_val, n, m);
    }
    updateY(nodex, lx, rx, 1, 1, m, x, y, new_val, n, m);
}

int main()
{
    ll n,m,t,i,j,k,a,b,c,d,cs=1;

    cin>> n >> m;
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
            cin>> val[i][j];

    buildX(1,1,n,n,m);
```

```
    cin>> t;
    while(t--){
        cin>> cs;
        if(!cs){
            cin>> a >> b >> c >> d;
            cout<< sumX(1,1,n,a,b,c,d,n,m) << endl;
        }
        else{
            cin>> a >> b >> c;
            updateX(1,1,n,a,b,c,n,m);
        }
    }

    return 0;
}
```

1.5 Maximum Bracket Sequence Using Segment Tree

```
/** Maximum correct bracket subsequence in a Range **/

struct info{
    ll maxBrac,unFB,unSB;
};

string str;
info tree[MX*3];

void init(ll node, ll b, ll e)
{
    stack<char>Q;

    for(ll i=b;i<=e;i++){
        if(str[i]=='('
            Q.push(str[i]);
        else{
            if(Q.empty())
                tree[node].unSB++;
            else{
                Q.pop();
                tree[node].maxBrac+=2;
            }
        }
    }
    tree[node].unFB=Q.size();

    if (b == e) {
        return;
    }
    ll Left = node * 2;
    ll Right = node * 2 + 1;
    ll mid = (b + e) / 2;
    init(Left, b, mid);
    init(Right, mid + 1, e);
}

info query(ll node, ll b, ll e, ll i, ll j)
{
    info tmp;
    tmp.maxBrac=tmp.unFB=tmp.unSB=0;

    if (i > e || j < b)
        return tmp;
```

```
    if (b >= i && e <= j)
        return tree[node];

    ll Left = node * 2;
    ll Right = node * 2 + 1;
    ll mid = (b + e) / 2;
    info p1 = query(Left, b, mid, i, j);
    info p2 = query(Right, mid + 1, e, i, j);

    tmp.maxBrac = p1.maxBrac+p2.maxBrac;
    tmp.maxBrac += min(p1.unFB,p2.unSB)*2;
    tmp.unFB = p1.unFB+p2.unFB-min(p1.unFB,p2.unSB);
    tmp.unSB = p1.unSB+p2.unSB-min(p1.unFB,p2.unSB);

    return tmp;
}

int main()
{
    fast_io;

    ll n,m,t,i,j,k,a,b,c,cs=1;

    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> str;
    cin>> m;

    init(1,0,str.size()-1);

    while(m--){
        cin>> a >> b;

        info res=query(1,0,str.size()-1,--a,--b);
        cout<< res.maxBrac << endl;
    }

    return 0;
}
```

1.6 MO's Algorithm

```
const ll block=320;
struct Query
{
    ll l,r,id;

    Query() {}
    Query(ll a, ll b, ll c)
    {
        l=a, r=b, id=c;
    }
    bool operator< (const Query &rhs)
    {
        ll block_a = l/block, block_b = rhs.l/block;
        if(block_a==block_b)
            return (r<rhs.r)^(block_a&1);
        return block_a<block_b;
    }
};

ll l=0,r=-1,res=0;
ll cnt[MX];
vll val;

void add(ll x)
{
    if(!cnt[val[x]])
        res++;
    cnt[val[x]]++;
}

void remove(ll x)
{
    if(cnt[val[x]]==1)
        res--;
    cnt[val[x]]--;
}

int main()
{

    FIO;
//    IN;
//    OUT;
```

```
    ll N;
    cin>> N;
    val.resize(N);
    for(ll i=0; i<N; i++)
        cin>> val[i];

    ll Q;
    cin>> Q;
    ll ans[Q];
    vector<Query>qq(Q);
    for(ll i=0; i<Q; i++)
        cin>> qq[i].l >> qq[i].r, qq[i].id=i, qq[i].l--, qq[i].r--;

    sort(all(qq));
    for(ll i=0; i<Q; i++)
    {
        while(l > qq[i].l) add(--l);
        while(r < qq[i].r) add(++r);
        while(l < qq[i].l) remove(l++);
        while(r > qq[i].r) remove(r--);
        ans[qq[i].id]=res;
    }

    for(ll i=0;i<Q;i++) cout<< ans[i] << "\n";

}
```

1.7 MO's Algorithm On Tree

```
            /**
                Count on a tree II
                LCA,MO,DFS
                how many different integers that represent the weight of node
            **/

ll block=200;
struct Query
{
    ll l,r,id;
    ll type;

    Query() {}
    Query(ll a, ll b, ll c)
    {
        l=a, r=b, id=c;
    }
    bool operator< (const Query &rhs)
    {
        ll block_a = l/block, block_b = rhs.l/block;
        if(block_a==block_b)
            return (r<rhs.r)^(block_a&1);
        return block_a<block_b;
    }
};

ll l=0,r=-1,res=0;
ll cur=-1;
ll val[3*MX],cnt[MX],node[MX],vcnt[MX];
bool vis[MX];
vll G[MX];
pll somoy[MX];
ll level[MX],sparse[MX][22],par[MX];

inline void BFS(ll s)
{
    memset(vis, false, sizeof vis);

    queue<ll>Q;
    Q.push(s);
    par[s]=s;
    vis[s]=true;

    while(!Q.empty()){
```

```
        ll u=Q.front();
        Q.pop();

        for(auto v:G[u]){
            if(vis[v])
                continue;
            par[v]=u;
            vis[v]=true;
            Q.push(v);
            level[v]=level[u]+1;
        }
    }
}

inline void LCA_InIt(ll N, ll root)
{
    BFS(root);
    memset(sparse, -1, sizeof sparse);

    for(ll i=1;i<=N;i++)
        sparse[i][0]=par[i];

    for(ll j=1;(1<<j)<=N;j++)
        for(ll i=1;i<=N;i++)
            if(sparse[i][j-1]!=-1)
                sparse[i][j]=sparse[sparse[i][j-1]][j-1];
}

inline ll LCA_query(ll N, ll u, ll v)
{
    if(level[u]<level[v])
        swap(u,v);

    ll log=1;
    while(true){
        ll next=log+1;
        if((1<<next)>level[u])
            break;
        log++;
    }

    for(ll i=log;i>=0;i--)
        if(level[u]-(1<<i)>=level[v])
            u=sparse[u][i];

    if(u==v)
```

```
        return u;

    for (ll i = log; i >= 0; i--)
        if (sparse[u][i] != -1 && sparse[u][i] != sparse[v][i])
           u = sparse[u][i], v = sparse[v][i];

    return par[u];
}

void DFS(ll u)
{
    somoy[u].ff=++cur;
    val[cur]=u;
    vis[u]=true;

    for(auto v:G[u]){
        if(!vis[v])
            DFS(v);
    }
    somoy[u].ss=++cur;
    val[cur]=u;
}

inline void add(ll x)
{
    cnt[val[x]]++;
    if(cnt[val[x]]==2){
        vcnt[node[val[x]]]--;
        if(!vcnt[node[val[x]]]) res--;
    }
    else{
        vcnt[node[val[x]]]++;
        if(vcnt[node[val[x]]]==1) res++;
    }
}

inline void remove(ll x)
{
    cnt[val[x]]--;
    if(cnt[val[x]]==1){
        vcnt[node[val[x]]]++;
        if(vcnt[node[val[x]]]==1) res++;
    }
    else{
        vcnt[node[val[x]]]--;
        if(!vcnt[node[val[x]]]) res--;
```

```
    }
}

int main()
{
//    IN;
//    OUT;

    ll N,Q;
    while(scanf("%lld %lld",&N,&Q)!=EOF){
        memset(cnt,0,sizeof cnt);
        memset(vcnt,0,sizeof vcnt);
        memset(sparse,-1,sizeof sparse);
        for(ll i=0;i<MX;i++) G[i].clear();
        cur=-1,res=0;
        l=0,r=-1;

        ordered_set ost;
        for(ll i=1; i<=N; i++)
            scanf("%lld",&node[i]),ost.insert(node[i]);
        for(ll i=1;i<=N;i++) node[i]=ost.order_of_key(node[i]);

        for(ll i=0;i<N-1;i++){
            ll u,v;
            scanf("%lld %lld",&u,&v);
            G[u].push_back(v);
            G[v].push_back(u);
        }

        block=sqrt(N)+1;
        DFS(1);
        LCA_InIt(N,1);

        ll ans[Q];
        vector<Query>qq(Q);
        for(ll i=0; i<Q; i++){
            ll u,v;
            qq[i].id=i;
            scanf("%lld %lld",&u,&v);

            if(somoy[u].ff>somoy[v].ff) swap(u,v);
            ll lca=LCA_query(N,u,v);
            if(lca==u)
                qq[i].l=somoy[u].ff,qq[i].r=somoy[v].ff,qq[i].type=0;
            else
                qq[i].l=somoy[u].ss,qq[i].r=somoy[v].ff,qq[i].type=lca;
```

```
        }

        sort(all(qq));
        for(ll i=0; i<Q; i++)
        {
            while(l > qq[i].l) add(--l);
            while(r < qq[i].r) add(++r);
            while(l < qq[i].l) remove(l++);
            while(r > qq[i].r) remove(r--);

            ans[qq[i].id]=res;
            if(qq[i].type and !vcnt[node[qq[i].type]]) ans[qq[i].id]++;
        }

        for(ll i=0;i<Q;i++) printf("%lld\n",ans[i]);
    }

}
```

2 Graph

2.1 Dijkstra

```
struct comp{
    bool operator() (const pii(ll,ll) &a, const pii(ll,ll) &b)
    {
        return a.sd > b.sd;
    }
};

priority_queue<pii(ll,ll), vctr(pii(ll,ll)), comp>Q;
vctr(pii(ll,ll)) edg[MX];
ll dis[MX];
bool vis[MX];

int main()
{
    ll n,m,a,b,c,t,i,j,k,beg;
    string s;

    cin>> n >> m;
    while(m--){
        cin>> a >> b >> c;
        edg[a].pb(mkpr(b,c));
        edg[b].pb(mkpr(a,c));
    }

    beg=1;
    loop(i,n)
        dis[i]=INF,vis[i]=false;
    dis[beg]=0;
    Q.in(mkpr(beg,0));

    while(!Q.empty()){
        ll a=Q.top().ft;
        Q.out();
        if(vis[a])
            continue;
        k=edg[a].size();
        loop(i,k){
            b=edg[a][i-1].ft;
            c=edg[a][i-1].sd;

            if(!vis[b] && dis[b]>dis[a]+c){
```

```
                dis[b]=dis[a]+c;
                Q.in(mkpr(b,dis[b]));
            }
        }
        vis[a]=true;
    }

    cout<< dis[n] << endl;

    loop(i,n)
        edg[i].clear();
    while(!Q.empty())Q.out();

    return 0;
}
```

2.2 Minimum Spanning Tree-Krushkal

```
struct edge{
    ll u,v,w;

    bool operator < (const edge& a) const
    {
        return w < a.w;
    }
};

vctr(edge) ed;
ll par[MX];

ll find(ll n)
{
    if(n==par[n])
        return par[n];
    return par[n]=find(par[n]);
}

ll MST(ll n)
{
    ll i,j;

    sort(ed.begin(),ed.end());
    loop(i,n)
        par[i]=i;

    ll cnt=0,s=0,m=(ll)ed.size();
    for(i=0;i<m;i++){
        ll u=find(ed[i].u);
        ll v=find(ed[i].v);

        if(u!=v){
            par[u]=v;
            cnt++;
            s+=ed[i].w;
            if(cnt==n-1)
                break;
        }
    }

    return s;
```

```
}

int main()
{
    ll n,m,a,b,c,t,i,j,k;
    string s;

    cin>> n >> m;

    while(m--){
        cin>> a >> b >> c;

        edge tmp;
        tmp.u=a,tmp.v=b,tmp.w=c;
        ed.pb(tmp);
    }

    cout<< MST(n);

    return 0;
}
```

2.3 Strongly Connected Component

```
#define WHITE 1
#define GRAY 2
vector<ll>edge[MX],revEdge[MX],components[MX];
ll color[MX],mark;
stack<ll>Q;
bool visited[MX];

void DFS1(ll node)
{
    color[node]=GRAY;
    for(auto u: edge[node])
        if(color[u]==WHITE)
            DFS1(u);

    Q.push(node);
}

void DFS2(ll node, ll mark)
{
    components[mark].pb(node);
    visited[node]=true;

    cout<< node << endl;

    for(auto u: revEdge[node])
        if(!visited[u])
            DFS2(u,mark);
}

void findSSC(ll nodes)
{
    ll i;

    while(!Q.empty())
        Q.pop();
    for(i=0;i<MX;i++)
        visited[i]=false,color[i]=WHITE;
    for(i=0;i<MX;i++)
        components[i].clear();


    for(i=1;i<=nodes;i++)
```

```
        if(color[i]==WHITE)
            DFS1(i);

    while(!Q.empty()){
        ll u=Q.top();
        Q.pop();
        if(!visited[u]){
            mark++;
            DFS2(u,mark);
        }
    }

}


int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);

    cin>> n >> m;
    for(i=0;i<m;i++){
        cin>> a >> b;
        a++,b++;
        edge[a].pb(a);
        revEdge[a].pb(b);
    }

    mark=0;
    findSSC(n);

    for(i=1;i<=mark;i++){
        for(auto x:components[i])
            cout<< x << "␣";
        cout<< components[i].size() << "\n";
    }
    cout<< endl;

    return 0;
}
```

2.4 Topological Sort

```
vctr(ll) edge[MX];
ll inDegree[MX];

bool  compare(ll a, ll b)
{
    return inDegree[a] < inDegree[b];
}

vctr(ll) TopSort(ll n)
{
    vctr(ll) node,ans;

    for(ll i=1;i<=n;i++)
        node.pb(i);
    sort(node.begin(),node.end(),compare);

    queue<ll>q;
    for(ll i=0;i<n;i++){
        if(inDegree[node[i]])
            break;
        q.push(node[i]);
    }

    while(!q.empty()){
        ll u=q.front();
        q.pop();
        ans.pb(u);

        ll x=edge[u].size();
        for(ll i=0;i<x;i++){
            ll v=edge[u][i];

            inDegree[v]--;
            if(!inDegree[v])
                q.push(v);
        }
    }

    return ans;
}

int main()
{
```

```
    ll n,m,a,b,c,t,i,j,k;
    string s;

    cin>> n >> m;

    for(i=0;i<m;i++){
        cin>> a >> b;
        edge[a].pb(b);
        inDegree[b]++;
    }

    vctr(ll) ans=TopSort(n);

    for(i=0;i<ans.size();i++)
        cout<< ans[i] << '␣';

    return 0;
}
```

2.5 Heavy Light Decomposition

```
class Segment_Tree{
    ll N;
    struct Node{
        ll prop,sum;

        Node() {}

        Node(ll p, ll s)
        {
            prop=p,sum=s;
        }
    };

    vector<Node>val;

public:
    Segment_Tree(ll n)
    {
        Node nd(0,0);
        N=(n+5)*4;
        val.assign(N,nd);
    }

    Segment_Tree(vll arr) : Segment_Tree(arr.size())
    {
        for(ll i=0;i<arr.size();i++)
            update(1,1,N,i+1,i+1,arr[i]);
    }

    void update(ll id, ll b, ll e, ll i, ll j, ll x)
    {
        if (i > e || j < b)
            return;
        if (b >= i && e <= j)
        {
            val[id].sum = x;
            return;
        }

        ll Left = id << 1;
        ll Right = (id << 1) + 1;
        ll mid = (b + e) >> 1;
```

```
        update(Left, b, mid, i, j, x);
        update(Right, mid + 1, e, i, j, x);
        val[id].sum = max( val[Left].sum , val[Right].sum );
    }

    ll query(ll id, ll b, ll e, ll i, ll j)
    {
        if (i > e || j < b)
            return -INF;

        if (b >= i and e <= j)
            return val[id].sum;

        ll Left = id << 1;
        ll Right = (id << 1) + 1;
        ll mid = (b + e) >> 1;

        ll p1 = query(Left, b, mid, i, j);
        ll p2 = query(Right, mid + 1, e, i, j);

        return max( p1 , p2 );
    }

};

vector<pair<pll,ll> > G[MX];
vll root;
bool vis[MX];
ll par[MX],level[MX],sparse[MX][22];
ll heavy[MX],subsize[MX];
ll chain_heads[MX*2];
ll base_array[MX*2];
ll edge_counted;
ll chain_size;
Segment_Tree seg(MX);

struct treeNode{
    ll par;
    ll depth;
    ll subtree;
    ll pos_seg;
    ll chain;
}node[MX];

struct Edge{
    ll weight;
```

```
    ll deep_node;
}edge[MX];


void DFS(ll u, ll p, ll lvl)
{
    vis[u]=true;
    par[u]=p,level[u]=lvl;
    subsize[u]=1,heavy[u]=-1;
    ll mx=0;

    for(auto x:G[u]){
        ll v=x.ff.ff;
        if(!vis[v]){
            DFS(v,u,lvl+1);
            subsize[u]+=subsize[v];
            if(subsize[v]>mx) mx=subsize[v],heavy[u]=v;
        }
    }

    node[u].par=p;
    node[u].depth=lvl;
    node[u].subtree=subsize[u];
}

void LCA_Build(ll N)
{
    memset(vis,false,sizeof vis);
    for(auto r:root) if(!vis[r]) DFS(r,-1,0);
    memset(sparse, -1, sizeof sparse);

    for(ll i=1;i<=N;i++) sparse[i][0]=par[i];

    for(ll log=1;(1<<log)<=N;log++)
        for(ll i=1;i<=N;i++)
            if(sparse[i][log-1]!=-1)
                sparse[i][log]=sparse[sparse[i][log-1]][log-1];
}

ll LCA_Query(ll N, ll u, ll v)
{
    if(level[u]<level[v]) swap(u,v);

    ll log=1;
    while(true){
        ll next=log+1;
```

```
        if((1<<next)>level[u]) break;
        log++;
    }

    for(ll i=log;i>=0;i--)
        if(level[u]-(1<<i)>=level[v])
            u=sparse[u][i];

    if(u==v) return u;

    for (ll i=log;i>=0;i--)
        if (sparse[u][i]!=-1 and sparse[u][i]!=sparse[v][i])
           u=sparse[u][i],v=sparse[v][i];

    return par[u];
}

ll kth_Parent(ll N, ll u, ll K)
{
    if(level[u]<K) return -1;
    if(!K) return u;

    ll x;
    for(ll i=0;(1<<i)<=N;i++){
        if(sparse[u][i]!=-1 and (1<<i)>K)
            break;
        x=i;
    }

    return kth_Parent(N,sparse[u][x],K-(1<<x));
}


void HLD(ll cur_node, ll cost)
{
    vis[cur_node]=true;
    if (chain_heads[chain_size]==-1) chain_heads[chain_size] = cur_node;

    node[cur_node].chain=chain_size;
    node[cur_node].pos_seg=edge_counted;
    base_array[edge_counted]=cost;

    ll cc;
    for(auto x:G[cur_node]){
        if(x.ff.ff==heavy[cur_node]){
            cc=x.ff.ss;
```

```
            edge[x.ss].deep_node=heavy[cur_node];
            edge[x.ss].weight=x.ff.ss;
        }
    }

    if(heavy[cur_node]!=-1) ++edge_counted,HLD(heavy[cur_node],cc);

    for(auto x:G[cur_node]){
        ll v=x.ff.ff;
        if(!vis[v] and v!=heavy[cur_node]){
            ++edge_counted,++chain_size;
            HLD(v,x.ff.ss);
            edge[x.ss].deep_node=v;
            edge[x.ss].weight=x.ff.ss;
        }
    }
}

ll crawl_tree(ll u, ll v)
{
    ll chain_u,chain_v=node[v].chain,ans=-INF;
    if(level[u]<level[v]) swap(u,v);

    while(true){
        chain_u=node[u].chain;

        if(chain_u==chain_v){
//            cout<< u << " " << v << endl;
//            cout<< node[v].pos_seg+1 << " " << node[u].pos_seg << endl;
//            cout<< endl;
            ll cur=seg.query(1,1,edge_counted,node[v].pos_seg+1,node[u].pos_s
            if(u!=v) ans=max(ans,cur);
            break;
        }

        ll cur=seg.query(1,1,edge_counted,node[chain_heads[chain_u]].pos_seg,
        ans=max(ans,cur);
        u=node[chain_heads[chain_u]].par;
    }

    return ans;
}

void change(ll edge_no, ll val)
{
    ll pos=node[edge[edge_no].deep_node].pos_seg;
```

```
    ll cur=edge[edge_no].weight;

    seg.update(1,1,edge_counted,pos,pos,val);
        edge[edge_no].weight = val;
}

ll max_edge(ll u, ll v, ll N)
{
    ll lca=LCA_Query(N,u,v);

    return max(crawl_tree(u,lca),crawl_tree(v,lca));
}

void setup(ll N)
{
    LCA_Build(N);

    memset(vis,false,sizeof vis);
    for(auto r:root) HLD(r,-1);

    for(ll i=1;i<=edge_counted;i++) seg.update(1,1,edge_counted,i,i,base_arra
}

void reset()
{
    for(ll i=0;i<MX;i++) G[i].clear();
    memset(chain_heads,-1,sizeof chain_heads);
    root.clear();
    edge_counted=1,chain_size=1;
}

int main()
{

//    FIO;
//    IN;
//    OUT;

    ll T;
    scanf("%lld",&T);
    while(T--){
        ll N;
        scanf("%lld",&N);

        reset();
        root.push_back(1);
```

```
        for(ll i=1;i<=N-1;i++){
            ll u,v,w;
            scanf("%lld␣%lld␣%lld",&u,&v,&w);
            G[u].push_back({{v,w},i});
            G[v].push_back({{u,w},i});
        }

        setup(N);

        char str[10];
        while(true){
            scanf("%s",str);
            if(str[0]=='D') break;

            if(str[0]=='Q'){
                ll u,v;
                scanf("%lld␣%lld",&u,&v);
                ll res=max_edge(u,v,N);
                if(res==-INF) res=0;
                assert(res>=0);
                printf("%lld\n",res);
            }
            else{
                ll id,w;
                scanf("%lld␣%lld",&id,&w);
                if(id>=1 and id<=N-1) change(id,w);
            }
        }
    }

}
```

2.6 Maximum Flow

```
// Adjacency list implementation of FIFO push relabel maximum flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It solves
// random problems with 10000 vertices and 1000000 edges in a few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

struct Edge {
    ll from, to, cap, flow, index;
    Edge(ll from, ll to, ll cap, ll flow, ll index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
    ll N;
    vector<vector<Edge> > G;
    vector<ll> excess;
    vector<ll> dist, active, count;
    queue<ll> Q;

    PushRelabel(ll N) : N(N), G(N), excess(N), dist(N), active(N), count(2*N)

    void AddEdge(ll from, ll to, ll cap) {
        G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
        if (from == to) G[from].back().index++;
        G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
    }

    void Enqueue(ll v) {
        if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }
```

```
}

void Push(Edge &e) {
    ll amt = ll(min(excess[e.from], ll(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
}

void Gap(ll k) {
    for (ll v = 0; v < N; v++) {
        if (dist[v] < k) continue;
        count[dist[v]]--;
        dist[v] = max(dist[v], N+1);
        count[dist[v]]++;
        Enqueue(v);
    }
}

void Relabel(ll v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (ll i = 0; i < G[v].size(); i++)
        if (G[v][i].cap - G[v][i].flow > 0)
    dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
}

void Discharge(ll v) {
    for (ll i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);
    if (excess[v] > 0) {
        if (count[dist[v]] == 1)
        Gap(dist[v]);
            else
        Relabel(v);
    }
}

ll GetMaxFlow(ll s, ll t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
```

```
        active[s] = active[t] = true;
        for (ll i = 0; i < G[s].size(); i++) {
            excess[s] += G[s][i].cap;
            Push(G[s][i]);
        }

        while (!Q.empty()) {
            ll v = Q.front();
            Q.pop();
            active[v] = false;
            Discharge(v);
        }

        ll totflow = 0;
        for (ll i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
        return totflow;
    }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)


int main()
{

    ll n,m;

    cin>> n >> m;
    PushRelabel pr(n);
    for(ll i=0;i<m;i++){
        ll a,b,c;
        cin>> a >> b >> c;
        if(a==b)
            continue;
        pr.AddEdge(a-1,b-1,c);
        pr.AddEdge(b-1,a-1,c);
    }
    cout<< pr.GetMaxFlow(0,n-1);

    return 0;
}
```

2.7 Maximum Bipartite Matching

```
// Adjacency list implementation of FIFO push relabel maximum flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It solves
// random problems with 10000 vertices and 1000000 edges in a few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

struct Edge {
    ll from, to, cap, flow, index;
    Edge(ll from, ll to, ll cap, ll flow, ll index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
    ll N;
    vector<vector<Edge> > G;
    vector<ll> excess;
    vector<ll> dist, active, count;
    queue<ll> Q;

    PushRelabel(ll N) : N(N), G(N), excess(N), dist(N), active(N), count(2*N)

    void AddEdge(ll from, ll to, ll cap) {
        G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
        if (from == to) G[from].back().index++;
        G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
    }

    void Enqueue(ll v) {
        if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }
```

```
}

void Push(Edge &e) {
    ll amt = ll(min(excess[e.from], ll(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
}

void Gap(ll k) {
    for (ll v = 0; v < N; v++) {
        if (dist[v] < k) continue;
        count[dist[v]]--;
        dist[v] = max(dist[v], N+1);
        count[dist[v]]++;
        Enqueue(v);
    }
}

void Relabel(ll v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (ll i = 0; i < G[v].size(); i++)
        if (G[v][i].cap - G[v][i].flow > 0)
    dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
}

void Discharge(ll v) {
    for (ll i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);
    if (excess[v] > 0) {
        if (count[dist[v]] == 1)
        Gap(dist[v]);
            else
        Relabel(v);
    }
}

ll GetMaxFlow(ll s, ll t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
```

```
        active[s] = active[t] = true;
        for (ll i = 0; i < G[s].size(); i++) {
            excess[s] += G[s][i].cap;
            Push(G[s][i]);
        }

        while (!Q.empty()) {
            ll v = Q.front();
            Q.pop();
            active[v] = false;
            Discharge(v);
        }

        ll totflow = 0;
        for (ll i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
        return totflow;
    }
};

// BEGIN CUT
// The following code solves SPOJ problem #4110: Fast Maximum Flow (FASTFLOW)


struct info{
    ll age,height,divocee;
};

int main()
{

    fast_io;
    ll n,m,t,i,j,k,a,b,c,cs=1;
    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> t;

    for(cs=1;cs<=t;cs++){
        cin>> n >> m;
        PushRelabel pr(n+m+2);

        vector<info>man(n),woman(m);

        for(i=0;i<n;i++)
            cin>> man[i].height >> man[i].age >> man[i].divocee;
        for(i=0;i<m;i++)
```

```
            cin>> woman[i].height >> woman[i].age >> woman[i].divocee;

        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                if(fabs(man[i].height-woman[j].height)<=12 and fabs(man[i].ag
                    pr.AddEdge(i+1,j+1+n,1);
                    //pr.AddEdge(j+1+n,i+1,1);
                }
            }
        }

        for(i=0;i<n;i++){
            pr.AddEdge(0,i+1,1);
            pr.AddEdge(i+1,0,1);
        }
        for(i=0;i<m;i++){
            pr.AddEdge(i+1+n,n+m+1,1);
            pr.AddEdge(n+m+1,i+1+n,1);
        }

        /*for(i=0;i<m;i++){
            cin>> a >> b >> c;
            if(a==b)
                continue;
            pr.AddEdge(a-1,b-1,c);
            pr.AddEdge(b-1,a-1,c);
        }*/

        cout<< "Case␣" << cs << ":␣";
        cout<< pr.GetMaxFlow(0,n+m+1) << "\n";

    }

    return 0;
}
```

3 String

3.1 Hashing

```
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
typedef long long int ll;
typedef pair < int,int > PII;
typedef pair < ll,ll > PLL;
#define F first
#define S second
ostream& operator<<(ostream & os, PLL h)
{
        return os << "(␣" << h.F << ",␣" << h.S << "␣)" << endl;

}
PLL operator+ (PLL a, ll x)     {return {a.F + x, a.S + x} ;}
PLL operator- (PLL a, ll x)     {return {a.F - x, a.S - x} ;}
PLL operator* (PLL a, ll x)     {return {a.F * x, a.S * x} ;}
PLL operator+(PLL x, PLL y) { return {x.F + y.F,x.S + y.S} ;}
PLL operator-(PLL x,PLL y) { return {x.F - y.F, x.S - y.S} ;}
PLL operator*(PLL x,PLL y) { return {x.F * y.F , x.S * y.S} ;}
PLL operator%(PLL x,PLL y) { return {x.F % y.F, x.S % y.S} ;}

PLL base = {37,41};

PLL M = {1000000021, 1e9 + 9 };
int const MX = 2e6 + 10;

PLL P[MX];
PLL h[MX] ;

map < PLL, int > mp;

PLL Hash(string &s)
{
    PLL hh = {0,0};

    for(int i = 0; i < s.size(); i++)
    {
        hh = (hh * base + (s[i] - 'a' + 1)) % M ;
    }
    //cout << hh << endl;
```

```
    return hh ;
}

PLL sub(int l,int r)
{
    return ( (h[r] - (h[l-1]*P[r-l+1])) % M + M ) % M ;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    P[0] = {1,1};

    for(int i = 1; i < MX; i++)
        P[i] = (P[i-1] * base) % M;


    int n,k;
    cin >> n >> k;
    string s;
    cin >> s;
    s += s;


    h[0] = {0,0} ;

    for(int i = 0; i < s.size(); i++){
        h[i+1] = (h[i] * base +  (ll)(s[i] - 'a' + 1))% M ;
    }

    int g;
    cin >> g;

    for(int i = 0; i < g; i++)
    {

        string s;
        cin >> s;
        //cout << Hash(s) << endl;

        mp[Hash(s)] = i+1;
    }
    vector < int > ans;
    vector < bool > vis(g+1,0);
    bool f = 0;
```

```
    for(int i = 1; i <= k && !f ; i++){
        ans.clear();
        int cnt = 0;
        for(int j = i,cnt = 1; j + k - 1 <= s.size() and cnt <= n; j += k, cn
            PLL tempHash = sub(j,j+k-1);
            //cout << tempHash << endl;

            if(mp.count(tempHash) and !vis[mp[tempHash]]){
                ans.pb(mp[tempHash]);
                vis[mp[tempHash]] = 1;
            }

        }
        if(ans.size() == n){
            f = 1;
            break;
        }
        for(int i =0 ; i < ans.size(); i++)
            vis[ans[i]] = 0;
    }

    if(!f){
        return cout << "NO", 0 ;
    }
    cout << "YES" << '\n';

    for(auto x: ans)
        cout << x << "␣" ;
    cout << endl;

    return 0;
}
```

3.2 Knuth-Moriss-Pitt

```
vll pref_func(string str)
{
    ll n=str.size();
    vll v(n);
    v[0]=0;
    for(ll i=1,j=0;i<n;i++){
        while(j>0 and str[i]!=str[j])
            j=v[j-1];
        if(str[i]==str[j])
            j++;
        v[i]=j;
    }

    return v;
}



int main()
{
    ll n,m,t,i,j,k,a,b,c,cs=1;

    string text,str;

    cin>> text >> str;
    vll pf=pref_func(str);
    vll res;

    for(i=0,j=0;i<text.size();i++){
        while(j>0 and text[i]!=str[j])
                j=pf[j-1];
        if(str[j]==text[i])
            j++;
        if(j==str.size()){
            res.pb(i-j);
            j=pf[j-1];
        }
    }
    if(j==str.size())
        res.pb(i-j);

    cout<< res.size() << endl;
    for(i=0;i<res.size();i++)
```

```
        cout<< res[i]+1 << '␣';
    cout<< endl;


    return 0;
}
```

3.3 Z Function

```
vll z_func(string str)
{
    ll n=str.size();
    vll z(n);
    z[0]=0;
    for(ll i=1,l=0,r=0;i<n;i++){
        if(i<=r)
            z[i]=min(r-i+1,z[i-l]);
        while(i+z[i]<n and str[z[i]]==str[i+z[i]])
            z[i]++;
        if(i+z[i]-1>r)
            l=i,r=i+z[i]-1;
    }

    return z;
}

int main()
{

    fast_io;
    ll n,m,t,i,j,k,a,b,c,cs=1;
    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);
    string str;

    cin>> str;
    vll zf=z_func(str);

    for(i=0;i<str.size();i++)
        cout<< zf[i] << '␣';
    cout<< endl;
    return 0;
}
```

3.4 Xor Maximization Using Trie Tree

```
const ll K=2;

struct Node{
    ll next[K];
    bool leaf=false;

    Node()
    {
        memset(next, -1, sizeof next);
    }
};

vector<Node>tr(1);

void add_String(string str)
{
    ll v=0;
    for(auto ch: str)
    {
        ll c=ch-'0';
        if(tr[v].next[c]==-1){
            tr[v].next[c]=tr.size();
            tr.emplace_back();
        }
        v=tr[v].next[c];
    }
    tr[v].leaf=true;
}

ll find_max_xor(string s)
{
    ll root=0;
    ll val=0;

    for(ll i=0;i<s.size();i++){
        if(s[i]=='0'){
            if(tr[root].next[1]!=-1){
                val=val*2+1;
                root=tr[root].next[1];
            }
            else if(tr[root].next[0]!=-1){
                val=val*2;
```

```
                root=tr[root].next[0];
            }
            else
                val=val*2;
        }
        else{
            if(tr[root].next[0]!=-1){
                val=val*2+1;
                root=tr[root].next[0];
            }
            else if(tr[root].next[1]!=-1){
                val=val*2;
                root=tr[root].next[1];
            }
            else
                val=val*2;
        }
    }

    return val;
}

string make(ll n)
{
    string str;

    while(n){
        str+=n%2+48;
        n/=2;
    }
    for(ll i=str.size();i<32;i++)
        str+=48;
    reverse(str.begin(),str.end());

    //cout<< str << endl;

    return str;
}

int main()
{

    fast_io;
    ll n,m,t,i,j,k,a,b,c,cs=1;
    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);
```

```
    cin>> n;
    for(i=0;i<n;i++){
        cin>> a;
        string str=make(a);
        add_String(str);
    }

    cin>> k;
    string str=make(k);
    cout<< find_max_xor(str);
    return 0;
}
```

3.5 Aho Corasik

```
ll leaf_node[MX];
string text;

const ll K = 26;

struct Vertex {
    ll next[K];
    bool leaf = false;
    ll p = -1;
    char pch;
    ll link = -1;
    ll go[K];
    vll pos;
    vll exit_link;
    ll cnt=0;
    bool check=false;

    Vertex(ll p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s, ll pos) {
    ll v = 0;
    for (char ch : s) {
        ll c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].pos.push_back(pos);
    t[v].leaf = true;
    leaf_node[pos]=v;
}

ll go(ll v, char ch);

ll get_link(ll v) {
```

```
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

ll go(ll v, char ch) {
    ll c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }

    return t[v].go[c];
}

void DFS(ll pos, ll node)
{
    if(pos==text.size()) return;

    ll c = text[pos] - 'a';
    if(t[node].next[c]!=-1){
        t[t[node].next[c]].cnt++;
        DFS(pos+1,t[node].next[c]);
    }
    else if(node) DFS(pos,get_link(node));

}

ll calc_count(ll node)
{
    if(t[node].check) return t[node].cnt;
    t[node].check=true;

    for(auto x:t[node].exit_link) t[node].cnt+=calc_count(x);

    return t[node].cnt;
}

int main()
{
```

```
//    FIO;
//    IN;
//    OUT;

    ll T;
    cin>> T;
    for(ll cs=1;cs<=T;cs++){
        ll n;
        cin>> n >> text;

        t.clear();
        t.push_back(Vertex());

        for(ll i=0;i<n;i++){
            string s;
            cin>> s;
            add_string(s,i);
        }

        for(ll i=0;i<t.size();i++){
            t[get_link(i)].exit_link.push_back(i);
        }

        DFS(0,0);

        cout<< "Case␣" << cs << ":\n";
        for(ll i=0;i<n;i++){
            cout<< calc_count(leaf_node[i]) << "\n";
        }
    }
}
```

3.6 Suffix Array Emax

```
vll sort_cyclic_shifts(string const& s)
{
    ll n=s.size();
    const ll alp=256;

    vll p(n),c(n),cnt(max(alp,n),0);
    ll i,j,k,h;
    for(i=0;i<n;i++)
        cnt[s[i]]++;
    for(i=1;i<alp;i++)
        cnt[i]+=cnt[i-1];
    for(i=0;i<n;i++)
        p[--cnt[s[i]]]=i;

    ll classes=1;
    c[p[0]]=0;
    for(i=1;i<n;i++){
        if(s[p[i]]!=s[p[i-1]])
            classes++;
        c[p[i]]=classes-1;
    }

    vll pn(n),cn(n);
    for(h=0;(1<<h)<n;h++){
        for(i=0;i<n;i++){
            pn[i]=p[i]-(1<<h);
            if(pn[i]<0)
                pn[i]+=n;
        }

        fill(cnt.begin(),cnt.begin()+classes,0);
        for(i=0;i<n;i++)
            cnt[c[pn[i]]]++;
        for (i=1;i<classes;i++)
            cnt[i]+=cnt[i-1];
        for (i=n-1;i>=0;i--)
            p[--cnt[c[pn[i]]]]=pn[i];

        cn[p[0]]=0;
        classes=1;
        for(i=1;i<n;i++){
            pll cur={c[p[i]],c[(p[i]+(1<<h))%n]};
```

```
            pll prev={c[p[i-1]],c[(p[i-1]+(1<<h))%n]};

            if(cur!=prev)
                classes++;
            cn[p[i]]=classes-1;
        }

        c.swap(cn);
    }

    return p;
}

vll build_suffix_array(string s)
{
    s+='$';
    vll res=sort_cyclic_shifts(s);
    res.erase(res.begin());
    return res;
}

vll lcp_construction(string const& s, vll const& p) {
    ll n = s.size();
    vll rank(n, 0);
    for (ll i = 0; i < n; i++)
        rank[p[i]] = i;

    ll k = 0;
    vll lcp(n-1, 0);
    for (ll i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        ll j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}

ll uniqueSubstrings(string str)
{
```

```
    ll n=str.size();
    vll sa=build_suffix_array(str);
    vll lcp=lcp_construction(str,sa);

    ll res=n-sa[0];
    for(ll i=1;i<n;i++)
        res+=(n-sa[i])-lcp[i-1];
    res++;

    return res;
}

int main()
{
    ll n,m,t,i,j,k,a,b,c,cs=1;

    string str;

    cin>> str;

    vll res=build_suffix_array(str);
    for(i=0;i<res.size();i++)
        cout<< res[i] << '␣';
    cout<< endl;

    cout<< uniqueSubstrings(str) << endl;

    return 0;
}
```

3.7 Suffix Array $O(n \log n)$

```
#include<bits/stdc++.h>
using namespace std;


string str;
ll N, m, SA [MX], LCP [MX];
ll x [MX], y [MX], w [MX], c [MX];

inline bool cmp (const ll a, const ll b, const ll l) { return (y [a] == y [b]

void Sort () {
    for (ll i = 0; i < m; ++i) w [i] = 0;
    for (ll i = 0; i < N; ++i) ++w [x [y [i]]];
    for (ll i = 0; i < m - 1; ++i) w [i + 1] += w [i];
    for (ll i = N - 1; i >= 0; --i) SA [--w [x [y [i]]]] = y [i];
}

void DA () {
    ++N;
    for (ll i = 0; i < N; ++i) x [i] = str [i], y[i] = i;
    Sort ();
    for (ll i, j = 1, p = 1; p < N; j <<= 1, m = p) {
        for (p = 0, i = N - j; i < N; i++) y [p++] = i;
        for (ll k = 0; k < N; ++k) if (SA [k] >= j) y [p++] = SA [k] - j;
        Sort ();
        for (swap (x, y), p = 1, x [SA [0]] = 0, i = 1; i < N; ++i) x [SA [i]]
    }
    for (ll i = 1; i < N; ++i) SA [i - 1] = SA [i]; --N;
}

void kasaiLCP () {
    for (ll i = 0; i < N; ++i) c [SA [i]] = i;
    LCP [0] = 0;
    for (ll i = 0, h = 0; i < N; ++i) if (c[i] > 0) {
            ll j = SA [c [i] - 1];
            while (i + h < N && j + h < N && str [i + h] == str [j + h]) ++h;
            LCP [c [i]] = h;
            if (h > 0) --h;
        }
}

void suffixArray () {
    m = 256;
    N = str.size();
```

```
    DA ();
    kasaiLCP ();
}


int main()
{
    ll n,m,t,i,j,k,a,b,c,cs=1;

    cin>> str;
    suffixArray();

    for(i=0;i<str.size();i++)
        cout<< SA[i] << ' ';


    return 0;
}
```

4 Math

4.1 Number Theory

```
#define MX 10000005
#define mod 1000000007
#define INF 10000000000000

bool marked[MX];
vll primes;

void sieve()
{
    int i,j;

    marked[0]=marked[1]=true;
    for(i=2;i*i<=MX-1;i++){
        if(marked[i]==false){
            for(j=i*i;j<=MX-1;j+=i){
                marked[j]=true;
            }
        }
    }

    loop(i,MX-1)
        if(!marked[i]){
            primes.pb(i);
            //cout<< i << ' ';
        }
}

vll primeFactors(ll N)
{
        vll factors;
        ll pf_id=0,pf=primes[pf_id];
        while(pf*pf<=N){
        while(N%pf==0){
            N/=pf;
            factors.pb(pf);
        }
        pf=primes[++pf_id];
        }
        if(N!=1)
        factors.pb(N);
```

```
    return factors;
}

ll numPF(ll N)
{
        ll pf_id=0,pf=primes[pf_id],ans=0;
        while(pf*pf<=N){
        while(N%pf==0){
            N/=pf;
            ans++;
        }
        pf=primes[++pf_id];
        }
        if(N!=1)
        ans++;

    return ans;
}

ll numDiv(ll N)
{
        ll pf_id=0,pf=primes[pf_id],ans=1;
        while(pf*pf<=N){
        ll power=0;
        while(N%pf==0){
            N/=pf;
            power++;
        }
        ans*=(power+1);
        pf=primes[++pf_id];
        }
        if(N!=1)
        ans*=2;

    return ans;
}

ll sumDiv(ll N)
{
        ll pf_id=0,pf=primes[pf_id],ans=1;
        while(pf*pf<=N){
        ll power=0;
        while(N%pf==0){
            N/=pf;
            power++;
        }
```

```
        ans*=((ll)pow((double)pf,power+1.0)-1)/(pf-1);
        pf=primes[++pf_id];
        }
        if(N!=1)
        ans*=((ll)pow((double)N,2.0)-1)/(N-1);;

    return ans;
}

int main()
{
    ll n,m,a,b,c,t,i,j,k;
    string s;

    sieve();

    return 0;
}
```

4.2 Rho Pollard Sieve

```
/*
_sieve _prime_factorize
Program to print prime factorisation of a number in the range [1, 1e18].
Input: an integer denoting the value of n
Output: prime factors of n in ascending order, separated by '*' character
Sample Input: 546534813485312
Sample Output: 5*1373*89533*13216567543
Time Complexity: O(1e7)
Space Complexity: O(1e6)
Stack Overflow Problem Link: https://stackoverflow.com/questions/50251565
*/

#include <iostream>
#include <vector>

#include <algorithm> // __gcd()
#include <cstring> // memset()
#include <cassert> // assert()

using namespace std;

#define int long long
typedef long double dbl;

const int N=2e6+10;

int np, prime[N];
bool isp[N];
void sieve(int N) {
    memset(isp, true, sizeof isp);
    isp[0] = isp[1] = false;
    for(int i=2; i<N; i++) if(isp[i]) {
        prime[++np]=i;
        for(int j=2*i; j<N; j+=i) {
            isp[j]=false;
        }
    }
}

inline int mul(int a, int b, int m) {
        a%=m; if(a<0) a+=m;
        b%=m; if(b<0) b+=m;
        int q = ((dbl)a * (dbl)b) / (dbl)m;
        int r = a*b - q*m;
```

```
        return (r<0 ? r+m:r);
}
inline int pwr(int a, int n, int m) {
    int ans(1);
    while(n) {
        if(n & 1) ans = mul(ans, a, m);
        if(n >>= 1) a = mul(a, a, m);
    }
    return ans%m;
}
int myrand(int n) {
        return rand()%n*rand()%n*rand()%n;
}
bool ispmiller(int p) { // O(30*logp)
        if(p<2) return false;
        if(p==2) return true;
        if(p%2==0) return false;
        int s=p-1; s>>=__builtin_ctzll(s);
        for(int i=0; i<60; i++) {
                int val=pwr(myrand(p-1)+1,s,p), temp=s;
                while(temp!=p-1 and 1<val and val<p-1) {
                        val=mul(val,val,p);
                        temp<<=1;
                }
                if(val!=p-1 and temp%2==0) return false;
        }
        return true;
}
inline int pollardrho(int n) { // O(n^0.25)
        if(n==1) return 1;
        if(n%2==0) return 2;
        int c=myrand(n-1)+1, x=myrand(n-2)+2, y=x;
        int d=1; while(d==1) {
                x=mul(x,x,n)+c; if(x>=n) x-=n;
                y=mul(y,y,n)+c; if(y>=n) y-=n;
                y=mul(y,y,n)+c; if(y>=n) y-=n;
                d=__gcd(abs(x-y),n);
                if(d==n) return (ispmiller(n) ? n:pollardrho(n));
        }
        return d;
}

#undef int
int main() {
#define int long long
```

```
    int n; cin >> n;
    if(ispmiller(n)) {
        cout << n << '\n'; // input n is prime, output as it is
        return 0;
    }

    vector<int> factors; // holds the prime factorisation of input n

    sieve(1e6);
    for(int i=1; i<np and prime[i]*prime[i]<=n; i++) {
        if(n%prime[i]==0) { // n is divisible by prime[i] (<= 1e6)
                while(n%prime[i]==0) {
                        n /= prime[i];
                        factors.push_back(prime[i]);
                }
        }
    }

    if(ispmiller(n)) {
        factors.push_back(n);
    }
    else if(n>1) { // n still has some prime factors > 1e6
        int x = pollardrho(n);
        assert(x > 1e6);
        factors.push_back(x);
        factors.push_back(n/x);
    }

    // Print the factorisation
    for(int i=0; i<(int)factors.size()-1; i++) cout << factors[i] << '*';
        cout << (factors.empty() ? 1 : factors.back()) << '\n';
    return 0;
}
```

4.3 Big Integer

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<ll,ll> lll;
typedef pair<ll,int> lli;
typedef pair<int,int> ii;

#define EL printf("\n")
#define OK printf("OK")
#define pb push_back
#define mp make_pair
#define ep emplace_back
#define X  first
#define Y  second
#define fillchar(a,x) memset(a, x, sizeof(a))
#define FOR(i,l,r) for (int i=l;i<=r;i++)
#define FORD(i,r,l) for (int i=r;i>=l;i--)

const int base = 1e9;
typedef vector<int> BigInt;

void Set(BigInt &a) {
    while (a.size() > 1 && a.back() == 0) a.pop_back();
}

void Print(BigInt a) {
    Set(a);
    printf("%d", (a.size() == 0) ? 0 : a.back());
    FORD(i,a.size()-2,0) printf("%09d", a[i]); EL;
}



BigInt Integer(string s) {
    BigInt ans;
    if (s[0] == '-') return ans;
    if (s.size() == 0) {ans.pb(0); return ans;}
    while (s.size()%9 != 0) s = '0'+s;
    for (int i=0;i<s.size();i+=9) {
        int v = 0;
        for (int j=i;j<i+9;j++) v = v*10+(s[j]-'0');
        ans.insert(ans.begin(),v);
```

```
    }
    Set(ans);
    return ans;
}

BigInt Integer(char c[]) {
    string s = "";
    FOR(i,0,strlen(c)-1) s = s + c[i];
    return Integer(s);
}

BigInt Integer(ll x) {
    string s = "";
    while (x > 0) s = char(x%10+'0') + s, x /= 10;
    return Integer(s);
}

BigInt Integer(int x) {
    return Integer((ll) x);
}




void operator >> (istream &in, BigInt &a) {
    string s;
    getline(cin, s);
    a = Integer(s);
}

void operator << (ostream &out, BigInt a) {
    Print(a);
}




bool operator < (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (a.size() != b.size()) return (a.size() < b.size());
    FORD(i,a.size()-1,0)
        if (a[i] != b[i]) return (a[i] < b[i]);
    return false;
}
```

```
bool operator > (BigInt a, BigInt b) {
    return (b < a);
}

bool operator == (BigInt a, BigInt b) {
    return (!(a < b) && !(b < a));
}

bool operator <= (BigInt a, BigInt b) {
    return (a < b || a == b);
}

bool operator >= (BigInt a, BigInt b) {
    return (b < a || b == a);
}

bool operator < (BigInt a, int b) {
    return (a < Integer(b));
}

bool operator > (BigInt a, int b) {
    return (a > Integer(b));
}

bool operator == (BigInt a, int b) {
    return (a == Integer(b));
}

bool operator >= (BigInt a, int b) {
    return (a >= Integer(b));
}

bool operator <= (BigInt a, int b) {
    return (a <= Integer(b));
}



BigInt max(BigInt a, BigInt b) {
    if (a > b) return a;
    return b;
}

BigInt min(BigInt a, BigInt b) {
    if (a < b) return a;
    return b;
```

```
}



BigInt operator + (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    int carry = 0;
    FOR(i,0,max(a.size(), b.size())-1) {
        if (i < a.size()) carry += a[i];
        if (i < b.size()) carry += b[i];
        ans.pb(carry%base);
        carry /= base;
    }
    if (carry) ans.pb(carry);
    Set(ans);
    return ans;
}

BigInt operator + (BigInt a, int b) {
    return a + Integer(b);
}

BigInt operator ++ (BigInt &a) { // ++a
    a = a + 1;
    return a;
}

void operator += (BigInt &a, BigInt b) {
    a = a + b;
}

void operator += (BigInt &a, int b) {
    a = a + b;
}



BigInt operator - (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    int carry = 0;
```

```
    FOR(i,0,a.size()-1) {
        carry += a[i] - (i < b.size() ? b[i] : 0);
        if (carry < 0) ans.pb(carry+base), carry = -1;
        else ans.pb(carry), carry = 0;
    }
    Set(ans);
    return ans;
}

BigInt operator - (BigInt a, int b) {
    return a - Integer(b);
}

void operator -- (BigInt &a) { // --a
    a = a - 1;
}

void operator -= (BigInt &a, BigInt b) {
    a = a + b;
}

void operator -= (BigInt &a, int b) {
    a = a - b;
}




BigInt operator * (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    BigInt ans;
    ans.assign(a.size()+b.size(), 0);
    FOR(i,0,a.size()-1) {
        ll carry = 0ll;
        for (int j=0;j<b.size() || carry > 0;j++) {
            ll s = ans[i+j] + carry + (ll)a[i]*(j<b.size()?(ll)b[j]:0ll);
            ans[i+j] = s%base;
            carry = s/base;
        }
    }
    Set(ans);
    return ans;
}

BigInt operator * (BigInt a, int b) {
```

```
    return a * Integer(b);
}

void operator *= (BigInt &a, BigInt b) {
    a = a * b;
}

void operator *= (BigInt &a, int b) {
    a = a * b;
}



BigInt operator / (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (b == Integer(0)) return Integer("-1");
    BigInt ans, cur;
    FORD(i,a.size()-1,0) {
        cur.insert(cur.begin(), a[i]);
        int x = 0, L = 0, R = base;
        while (L <= R) {
            int mid = (L+R)>>1;
            if (b*Integer(mid) > cur) {
                x = mid;
                R = mid-1;
            }
            else
                L = mid+1;
        }
        cur = cur - Integer(x-1)*b;
        ans.insert(ans.begin(),x-1);
    }
    Set(ans);
    return ans;
}

BigInt operator / (BigInt a, int b) {
    Set(a);
    BigInt ans;
    ll cur = 0ll;
    FORD(i,a.size()-1,0) {
        cur = (cur*(ll)base + (ll)a[i]);
        ans.insert(ans.begin(),cur/b);
        cur %= b;
    }
```

```
    Set(ans);
    return ans;
}

void operator /= (BigInt &a, BigInt b) {
    a = a / b;
}

void operator /= (BigInt &a, int b) {
    a = a / b;
}



BigInt operator % (BigInt a, BigInt b) {
    Set(a);
    Set(b);
    if (b == Integer(0)) return Integer("-1");
    BigInt ans;
    FORD(i,a.size()-1,0) {
        ans.insert(ans.begin(), a[i]);
        int x = 0, L = 0, R = base;
        while (L <= R) {
            int mid = (L+R)>>1;
            if (b*Integer(mid) > ans) {
                x = mid;
                R = mid-1;
            }
            else
                L = mid+1;
        }
        ans = ans - Integer(x-1)*b;
    }
    Set(ans);
    return ans;
}

int operator % (BigInt a, int b) {
    Set(a);
    if (b == 0) return -1;
    int ans = 0;
    FORD(i,a.size()-1,0)
        ans = (ans*(base%b) + a[i]%b)%b;
    return ans;
}
```

```
void operator %= (BigInt &a, BigInt b) {
    a = a % b;
}

void operator %= (BigInt &a, int b) {
    a = a % Integer(b);
}

BigInt gcd(BigInt a, BigInt b) {
    Set(a);
    Set(b);
    while (b > Integer(0)) {
        BigInt r = a%b;
        a = b;
        b = r;
    }
    Set(a);
    return a;
}

BigInt lcm(BigInt a, BigInt b) {
    return (a*b/gcd(a,b));
}


BigInt sqrt(BigInt a) {
    BigInt x0 = a, x1 = (a+1)/2;
    while (x1 < x0) {
        x0 = x1;
        x1 = (x1+a/x1)/2;
    }
    return x0;
}


BigInt pow(BigInt a, BigInt b) {
    if (b == Integer(0)) return Integer(1);
    BigInt tmp = pow(a, b/2);
    if (b%2 == 0) return tmp * tmp;
    return tmp * tmp * a;
}


BigInt pow(BigInt a, int b) {
    return pow(a,(Integer(b)));
}
```

```
int log(int n, BigInt a) { // log_n(a)
    Set(a);
    int ans = 0;
    while (a > Integer(1)) {
        ans++;
        a /= n;
    }
    return ans;
}


int main()
{
    BigInt B;  cin >> B;
    BigInt A = Integer("123456789");
    BigInt C = Integer(12345678911);
    int x; x = 123456789;

    if (B <= A) cout << A - B;
    else {
        cout << "-";
        cout << B - A;
    }

    cout << A + B; Print(A + x);
    cout << A * B; Print(A * x);
    cout << A / B; Print(A / x);
    cout << A % B; printf("%d\n", A % x);

    C = ++A; ++B; C += B + x;
    Print(A); Print(B); Print(C);

    cout << max(A,B);
    cout << min(A,B);

    cout << gcd(A,B);
    cout << lcm(A,B);

    cout << sqrt(A);
    printf("%d␣%d␣%d\n", log(2,A), log(10,B), log(5,C));

    A = Integer(16); x = 12;
    cout << pow(A,B);
    cout << pow(A,x);
```

```
    return 0;
}
```

4.4 Gaussian Elimination

```
#include<bits/stdc++.h>
using namespace std;

#define fast_io ios_base::sync_with_stdio(0); //cin.tie(0);
#define ll long long
#define ld long double
#define pb push_back
#define ins insert
#define in push
#define out pop
#define loop(i,n) for(i=1;i<=n;i++)
#define loon(i,n) for(i=n;i>0;i--)
#define vctr(x) vector< x >
#define pii(x,y) pair< x,y >
#define mkpr(x,y) make_pair(x,y)
#define ft first
#define sd second
#define MX 1005
#define mod 1000000007
#define INF 10000000000000

double mat[MX][MX],solution[MX];
ll N;

void swap_row(ll i, ll j)
{
        for (ll k=0; k<=N; k++)
        {
                double temp = mat[i][k];
                mat[i][k] = mat[j][k];
                mat[j][k] = temp;
        }
}

void print()
{
        for (ll i=0; i<N; i++, printf("\n")){
                for (ll j=0; j<=N; j++)
                        printf("%f␣", mat[i][j]);
            printf("\n");
    }
}

ll forwardElim()
```

```
{
        for (ll k=0; k<N; k++)
        {
                ll i_max = k;

                for (ll i = k+1; i < N; i++)
                        if (fabs(mat[i][k]) > fabs(mat[i_max][k]))
                                i_max = i;

                if (mat[i_max][k]==0){
                    //cout<< i_max << endl;
                        return k;
                }

                if (i_max != k)
                        swap_row(k, i_max);

                for (ll i=k+1; i<N; i++)
                {
                        double f = mat[i][k]/mat[k][k];

                        for (ll j=k+1; j<=N; j++)
                                mat[i][j] -= mat[k][j]*f;

                        mat[i][k] = 0;
                }
        }


        return -1;
}

void backSub()
{
        for (ll i = N-1; i >= 0; i--)
        {
                solution[i] = mat[i][N];

                for (int j=i+1; j<N; j++)
                {
                        solution[i] -= mat[i][j]*solution[j];
                }

                solution[i] = solution[i]/mat[i][i];
        }
```

```
        /*printf("\nSolution for the system:\n");
        for (int i=0; i<N; i++)
                printf("%f\n", solution[i]);*/
}

void gaussianElimination()
{
        ll singular_flag = forwardElim();

        if (singular_flag != -1)
        {
                printf("Singular␣Matrix.\n");

                if (mat[singular_flag][N])
                        printf("Inconsistent␣System.\n");
                else
                        printf("May␣have␣infinitely␣many␣"
                                "solutions.\n");

                return;
        }

        backSub();
}

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> N;

    for(i=0;i<N;i++)
        for(j=0;j<=N;j++)
            cin>> mat[i][j];

    gaussianElimination();

    print();
    for(i=0;i<N;i++)
        cout<< fixed << setprecision(10) << solution[i] << endl;

    return 0;
```

```
}
```

4.5 Lagrange Interpolation

```
/*
    Complexity O(n^2) for pre-computing
    O(n) per query
*/

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

// x and y are vectors of the points
// n is the number of points; degree of poly is n-1
// returns intermediate co-efficients
vector<double> lagrange_interpolate(vector<double> x, vector<double> y, int n
    vector<double> ret = y;
    for(int i=0; i<n; ++i) {
        for(int j=0; j<n; ++j) {
            if(i == j) continue;
            ret[i] /= (x[i] - x[j]);
        }
    }
    return ret;
}

int main() {
    // Number of points
    // therefor polynomial order is n-1
    int n;
    cin >> n;

    vector<double> x(n), y(n);
    for(int i=0; i<n; ++i) {
        cin >> x[i] >> y[i];
    }

    vector<double> coef = lagrange_interpolate(x, y, n);

    // number of queries. How many x 's are there you want to know the y
    int q;
    cin >> q;

    while(q--) {
        double xx;
        cin >> xx;

        double mul = 1.0;
```

```
        for(int i=0; i<n; ++i) mul *= (xx - x[i]);

        bool input_point = false;
        double sum = 0;
        for(int i=0; i<n; ++i) {
            double div = (xx - x[i]);
            if(div == 0) {
                // This means this is one of the input points
                // Answer it immediately
                cout << y[i] << "\n";

                input_point = true;
                break;
            }
            else {
                sum += (coef[i] / div);
            }
        }
        if(input_point) continue;

        double res = mul * sum;
        cout << res << "\n";
    }

    return 0;
}
```

4.6 Newton Interpolation

```
/*
    Complexity O(n^2) for pre-computing
    O(n) per query
*/

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

vector<double> divided_diff(vector<double> x, vector<double> y, int n) {
    vector<double> ret(n, 0);
    ret[0] = y[0];

    vector<double> last = y;
    for(int i=1; i<n; ++i) {
        vector<double> temp;
        for(int j=0; j+1<(int) last.size(); ++j) {
            double diff = last[j+1] - last[j];
            diff /= (x[i+j] - x[j]);
            temp.push_back(diff);
        }

        last = temp;
        ret[i] = last[0];
    }

    return ret;
}

int main() {
    // Number of points
    // therefor polynomial order is n-1
    int n;
    cin >> n;

    vector<double> x(n), y(n);
    for(int i=0; i<n; ++i) {
        cin >> x[i] >> y[i];
    }

    vector<double> coef = divided_diff(x, y, n);

    // number of queries. How many x 's are there you want to know the y
    int q;
    cin >> q;
```

```
    while(q--) {
        double xx;
        cin >> xx;

        double mul = 1.0;
        double res = coef[0];
        for(int i=1; i<(int) coef.size(); ++i) {
            mul *= (xx - x[i-1]);
            res += (mul * coef[i]);
        }
        cout << res << "\n";
    }

    return 0;
}

/*

4
5 12
6 13
9 14
11 16

1
10

*/

/*

5
5 150
7 392
11 1452
13 2366
21 9702

2
6
9

*/
```

4.7 Fast Fourier Transformation (Complex)

```
#define CD complex<double>
#define MX 10000005
#define mod 1000000007
#define INF 10000000000000
#define EXP 0.000000001

const double PI = acos(-1.0);

ll rev(ll id, ll level)
{
    ll res=0;
    for(ll i=0;i<level;i++){
        if(id & (1<<i))
            res|=1<<(level-i-1);
    }
    return res;
}

void FFT(vector<CD>&PoC, bool inverse)
{
    ll len=PoC.size();
    ll level=0;

    while((1<<level)<len)
        level++;

    for(ll i=0;i<len;i++){
        if(i<rev(i,level))
            swap(PoC[i],PoC[rev(i,level)]);
    }

    for(ll l=2;l<=len;l<<=1){
         double ang=(2*PI*(inverse?-1:1))/l;
         CD wN(cos(ang),sin(ang));
         for(ll i=0;i<len;i+=l){
             CD w(1);
             for(ll j=0;j<l/2;j++){
                CD u=PoC[i+j],v=PoC[i+j+l/2]*w;
                PoC[i+j]=u+v;
                PoC[i+j+l/2]=u-v;
                w*=wN;
             }
         }
```

```
    }

    if(inverse){
        for(CD &x:PoC)
            x/=len;
    }
}

vector<ll> multiplyTP(vector<ll>&A,vector<ll>&B)
{
    vector<CD>polA(A.begin(),A.end()),polB(B.begin(),B.end());
    ll n=1;
    while(n<A.size()+B.size())
        n*=2;
    polA.resize(n);
    polB.resize(n);

    FFT(polA,false);
    FFT(polB,false);

    for(ll i=0;i<n;i++)
        polA[i]*=polB[i];
    FFT(polA,true);

    vector<ll>ans(n);
    for(ll i=0;i<n;i++)
        ans[i]=round(polA[i].real());

    return ans;
}

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    cin>> n >> m;

    vector<ll>A(n),B(m),C;

    for(i=0;i<n;i++)
        cin>> A[i];
    for(i=0;i<m;i++)
        cin>> B[i];

    C=multiplyTP(A,B);
```

```
    for(i=0;i<C.size();i++)
        cout<< C[i] << '␣';
    cout<< endl;

    return 0;
}
```

4.8 Fast Fourier Transformation (NTT)

```
#define MX 10000005
#define mod 1000000007
#define INF 10000000000000
#define EXP 0.000000001

const double PI = acos(-1.0);

const ll root_mod = 7340033;
const ll root = 5;
const ll root_1 = 4404020;
const ll root_pw = 1 << 20;

ll rev(ll id, ll level)
{
    ll res=0;
    for(ll i=0;i<level;i++){
        if(id & (1<<i))
            res|=1<<(level-i-1);
    }
    return res;
}

ll modPow(ll x, ll n, ll mood)
{
    if(n==0)
        return 1%mood;
    ll t=modPow(x,n/2,mood);

    t*=t;
    t%=mood;
    if(n%2)
        t*=x;
    t%=mood;

    return t;
}

void FFT(vector<ll>&PoC, bool inverse)
{
    ll len=PoC.size();
    ll level=0;

    while((1<<level)<len)
```

```
        level++;

    for(ll i=0;i<len;i++){
        if(i<rev(i,level))
            swap(PoC[i],PoC[rev(i,level)]);
    }

    for(ll l=2;l<=len;l<<=1){
         ll wN=inverse?root_1:root;
         for(ll i=l;i<root_pw;i<<=1)
            wN=(wN*wN)%root_mod;

         for(ll i=0;i<len;i+=l){
             ll w=1;
             for(ll j=0;j<l/2;j++){
                ll u=PoC[i+j],v=(PoC[i+j+l/2]*w)%root_mod;
                PoC[i+j]=(u+v)%root_mod;
                PoC[i+j+l/2]=(u-v+root_mod)%root_mod;
                w*=wN;
                w%=root_mod;
             }
         }
    }

    if(inverse){
        ll inv=modPow(len,root_mod-2,root_mod);
        for(ll &x:PoC)
            x*=inv,x%=root_mod;
    }
}

vector<ll> multiplyTP(vector<ll>&A,vector<ll>&B)
{
    vector<ll>polA(A.begin(),A.end()),polB(B.begin(),B.end());
    ll n=1;
    while(n<A.size()+B.size())
        n*=2;
    polA.resize(n);
    polB.resize(n);

    FFT(polA,false);
    FFT(polB,false);

    for(ll i=0;i<n;i++)
        polA[i]*=polB[i],polA[i]%=root_mod;
    FFT(polA,true);
```

```
    vector<ll>ans(n);
    for(ll i=0;i<n;i++)
        ans[i]=polA[i];

    return ans;
}

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    cin>> n >> m;

    vector<ll>A(n),B(m),C;

    for(i=0;i<n;i++)
        cin>> A[i];
    for(i=0;i<m;i++)
        cin>> B[i];

    C=multiplyTP(A,B);

    for(i=0;i<C.size();i++)
        cout<< C[i] << '␣';
    cout<< endl;

    return 0;
}
```

4.9 Big Integer Multiplication Using FFT

```
#define EXP 0.000000001

const double PI = acos(-1.0);

const ll root_mod = 7340033;
const ll root = 5;
const ll root_1 = 4404020;
const ll root_pw = 1 << 20;



ll rev(ll id, ll level)
{
    ll res=0;
    for(ll i=0;i<level;i++){
        if(id & (1<<i))
            res|=1<<(level-i-1);
    }
    return res;
}

ll modPow(ll x, ll n, ll mood)
{
    if(n==0)
        return 1%mood;
    ll t=modPow(x,n/2,mood);

    t*=t;
    t%=mood;
    if(n%2)
        t*=x;
    t%=mood;

    return t;
}

void FFT(vector<ll>&PoC, bool inverse)
{
    ll len=PoC.size();
    ll level=0;

    while((1<<level)<len)
        level++;
```

```
    for(ll i=0;i<len;i++){
        if(i<rev(i,level))
            swap(PoC[i],PoC[rev(i,level)]);
    }

    for(ll l=2;l<=len;l<<=1){
         ll wN=inverse?root_1:root;
         for(ll i=l;i<root_pw;i<<=1)
            wN=(wN*wN)%root_mod;

         for(ll i=0;i<len;i+=l){
             ll w=1;
             for(ll j=0;j<l/2;j++){
                ll u=PoC[i+j],v=(PoC[i+j+l/2]*w)%root_mod;
                PoC[i+j]=(u+v)%root_mod;
                PoC[i+j+l/2]=(u-v+root_mod)%root_mod;
                w*=wN;
                w%=root_mod;
             }
         }
    }

    if(inverse){
        ll inv=modPow(len,root_mod-2,root_mod);
        for(ll &x:PoC)
            x*=inv,x%=root_mod;
    }
}

vector<ll> multiplyTP(vector<ll>&A,vector<ll>&B)
{
    vector<ll>polA(A.begin(),A.end()),polB(B.begin(),B.end());
    ll n=1;
    while(n<A.size()+B.size())
        n*=2;
    polA.resize(n);
    polB.resize(n);

    FFT(polA,false);
    FFT(polB,false);

    for(ll i=0;i<n;i++)
        polA[i]*=polB[i],polA[i]%=root_mod;
    FFT(polA,true);
```

```
    vector<ll>ans(n);
    for(ll i=0;i<n;i++)
        ans[i]=polA[i];

    return ans;
}

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> t;
    while(t--){
        string str1,str2,str3;
        cin>> str1 >> str2;

        vector<ll>A(str1.size()),B(str2.size()),C;

        for(i=str1.size()-1,j=0;i>=0;i--,j++)
            A[j]=str1[i]-48;
        for(i=str2.size()-1,j=0;i>=0;i--,j++)
            B[j]=str2[i]-48;

        C=multiplyTP(A,B);

        ll carry=0,mul=10;
        for(i=0;i<C.size();i++){
            ll digit=C[i]+carry;
            //ll digit=C[i];
            //cout<< C[i] << ' ';
            str3+=digit%mul+48;
            carry=digit/mul;

        }
        reverse(str3.begin(),str3.end());

        //cout<< str3 << endl;

        bool ok=false;
        for(i=0;i<str3.size();i++){
            if(str3[i]!='0')
                ok=true;
```

```
            if(ok)
                cout<< str3[i];
        }
        if(!ok)
            cout<< 0;
        cout<< endl;
    }

    return 0;
}
```

5 Geometry

5.1 Geometry Library

```
                                /*Geom_Library*/


// C++ routines for computational geometry.
double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c
); }
  PT operator / (double c)     const { return PT(x/c,   y/c
); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
```

```
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
```

```
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
```

```
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
```

```
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << "␣"
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << "␣"
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;
```

```
// expected: 1 0 1
cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << "␣"
     << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << "␣"
     << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 0 0 1
cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << "␣"
     << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << "␣"
     << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 1 1 1 0
cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << "␣"
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << "␣"
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << "␣"
     << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

// expected: (1,2)
cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl

// expected: (1,1)
cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

vector<PT> v;
v.push_back(PT(0,0));
v.push_back(PT(5,0));
v.push_back(PT(5,5));
v.push_back(PT(0,5));

// expected: 1 1 1 0 0
cerr << PointInPolygon(v, PT(2,2)) << "␣"
     << PointInPolygon(v, PT(2,0)) << "␣"
     << PointInPolygon(v, PT(0,2)) << "␣"
     << PointInPolygon(v, PT(5,2)) << "␣"
     << PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1
cerr << PointOnPolygon(v, PT(2,2)) << "␣"
     << PointOnPolygon(v, PT(2,0)) << "␣"
     << PointOnPolygon(v, PT(0,2)) << "␣"
     << PointOnPolygon(v, PT(5,2)) << "␣"
     << PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)
//           (5,4) (4,5)
//           blank line
```

```
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

5.2 Convex Hull

```
struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c
); }
  PT operator / (double c)     const { return PT(x/c,   y/c
); }
};

bool cmp(PT a, PT b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cw(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
}

bool ccw(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
}

void convex_hull(vector<PT>& a) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), &cmp);
    PT p1 = a[0], p2 = a.back();
    vector<PT> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[up.size()-1], a[
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2)) {
            while(down.size() >= 2 && !ccw(down[down.size()-2], down[down.siz
                down.pop_back();
```

```
            down.push_back(a[i]);
        }
    }

    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}

inline double dist(const PT &a, const PT &b) {
        return hypot((double)(a.x-b.x), (double)(a.y-b.y));
}

int main()
{
    fast_io;
    ll n,m,t,i,j,k,a,b,c,cs=1;
    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> n;
    vector<PT>points(n);

    for(i=0;i<n;i++)
        cin>> points[i].x >> points[i].y;

    convex_hull(points);
    vector<PT>res=points;
    double convex_area=ComputeArea(res);

    cout<< res.size() << endl;
    cout<< convex_area << endl;
    for(i=0;i<res.size();i++)
        cout<< res[i].x << ' ' << res[i].y << endl;

    return 0;
}
```

6 Dynamic Programming

6.1 Bit Mask DP Sample

```
ll weight[MX][MX],dp[1<<(MX-1)],n;

ll setBit(ll N, ll pos){ return N = N | (1<<pos); }
ll resetBit(ll N, ll pos){ return N = N & ~(1<<pos); }
bool checkBit(ll N, ll pos){ return (bool)(N & (1<<pos)); }

ll bitMaskDP(ll mask)
{
    if(mask==(1<<n)-1)
        return 0;
    if(dp[mask]!=-1)
        return dp[mask];

    ll mn=1<<30;
    for(ll i=0;i<n;i++){
        if(!checkBit(mask,i)){
            ll price=weight[i][i];

            for(ll j=0;j<n;j++){
                if(i!=j and checkBit(mask,j))
                    price+=weight[i][j];
            }
            ll value=price+bitMaskDP(setBit(mask,i));
            mn=min(mn,value);
        }
    }

    dp[mask]=mn;
    return dp[mask];
}

int main()
{

    ll m,t,i,j,k,a,b,c,cs=1;

    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);

    cin>> t;
    while(t--){
```

```
        cin>> n;
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                cin>> weight[i][j];

        for(i=0;i<1<<(MX-1);i++)
            dp[i]=-1;

        cout<< "Case␣" << cs++ << ":␣" << bitMaskDP(0) << endl;
    }

    return 0;
}
```

6.2 Digit DP Sample

```
ll dp[MX][MX][3],total;
bool vis[MX][MX][3];
string str;

ll digitDP(ll pos, ll prev, ll ok)
{
    if(pos>=(ll)str.size()){
        return 1;
    }
    if(vis[pos][prev][ok])
        return dp[pos][prev][ok];
    vis[pos][prev][ok]=true;

    ll low=0,up=9;
    if(prev==total)
        up=0;
    else if(ok==1)
        up=str[pos]-48;
    if(!pos)
        low=1;

    ll ans=0;

    if(ok==1){
        for(ll i=low;i<=up;i++){
            ll x=0;
            if(i)
                x=1;
            if(i==up){
                ans+=digitDP(pos+1,prev+x,ok);
            }
            else{
                ans+=digitDP(pos+1,prev+x,3-ok);
            }
        }
    }
    else{
        for(ll i=low;i<=up;i++){
            ll x=0;
            if(i)
                x=1;
            ans+=digitDP(pos+1,prev+x,ok);
        }
```

```
    }

    //cout<< ans << endl;
    dp[pos][prev][ok]=ans;

    return dp[pos][prev][ok];
}

int main()
{

    ll n,m,t,i,j,k,a,b,c,cs=1;

    //freopen(input.txt, r, stdin);
    //freopen(output.txt, w, stdout);

    cin>> t;

    while(t--){
        string str1, str2;

        cin>> n >> m;
        n--;

        ll ans=0;
        total=3;

        if(n){
            memset(vis, false, sizeof vis);
            while(n){
                str1+=(char)(n%10+48);
                n/=10;
            }
            reverse(str1.begin(),str1.end());

            str="";
            ll tmp=0;

            for(i=1;i<=str1.size()-1;i++){
                str+='9';
                tmp+=digitDP(0,0,1);
                memset(vis, false, sizeof vis);
            }

            //cout<< tmp << ' ';
```

```
            str=str1;
            tmp+=digitDP(0,0,1);

            //cout<< tmp << endl;
            ans-=tmp;
        }

        if(m){
            memset(vis, false, sizeof vis);
            while(m){
                str2+=(char)(m%10+48);
                m/=10;
            }
            reverse(str2.begin(),str2.end());
            str=str2;

            str="";
            ll tmp=0;

            for(i=1;i<=str2.size()-1;i++){
                str+='9';
                tmp+=digitDP(0,0,1);
                memset(vis, false, sizeof vis);
            }

            //cout<< tmp << ' ';

            str=str2;
            tmp+=digitDP(0,0,1);
            //cout<< tmp << endl;
            ans+=tmp;
        }

        cout<< ans << endl;

    }
    return 0;
}
```

6.3 Sum Over Subsets DP (IUT IUPC Code)

```
            /* Power Puff Girls OF IUT IUPC 2019 */
            /* SOS DP */
            /* Sum of Over All Super-Set of a Sub-Set */
            /* Inclusion/Exclusion */
            /* Time: O((2^N)*N) */
            /* Memory: O((2^N)*N) */


ll N,v;
ll m1,m2,m3,m4;
ll cnt[1<<19];
ll F[5][1<<19];
ll res[1<<19];

ll bigMod(ll x, ll n)
{
    if(!n)
        return 1LL;
    ll temp=bigMod(x,n/2);
    temp=(temp*temp)%mod;
    if(n%2)
        temp=(temp*x)%mod;
    return temp;
}

void countF(ll id)
{
    ll dp[(1<<N)][N+1];
    memset(dp,0,sizeof dp);
    for(ll mask = (1<<N)-1; mask >= 0; --mask){
        dp[mask][0] = cnt[mask];
        for(ll i = 0;i < N; ++i){
            if(!(mask & (1<<i)))
                dp[mask][i+1] = (dp[mask][i] + dp[mask^(1<<i)][i])%mod;
            else
                dp[mask][i+1] = dp[mask][i];
        }
        F[id][mask] = dp[mask][N];
    }

    for(ll mask = 0; mask<(1<<N); mask++)
        res[mask]=(res[mask]*F[id][mask])%mod;
}
```

```
void countRes()
{
    ll dp[(1<<N)][N+1];
    memset(dp,0,sizeof dp);

    for(ll mask = (1<<N)-1; mask >= 0; --mask){
        dp[mask][0] = 0;
        for(ll i = 0;i < N; ++i){
            if(!(mask & (1<<i)))
                dp[mask][i+1] = (dp[mask][i] + dp[mask^(1<<i)][i] + res[mask^
            else
                dp[mask][i+1] = dp[mask][i];
        }
        res[mask] = (res[mask] - dp[mask][N] + mod)%mod;
    }
}

int main()
{

    fast_io;

    cin>> N;

    for(ll i=0; i<(1<<19); i++)
        res[i]=1;

    cin>> m1;
    memset(cnt,0,sizeof cnt);
    for(ll i=0; i<m1; i++)
        cin>> v,cnt[v]++;
    countF(0);

    cin>> m2;
    memset(cnt,0,sizeof cnt);
    for(ll i=0; i<m2; i++)
        cin>> v,cnt[v]++;
    countF(1);

    cin>> m3;
    memset(cnt,0,sizeof cnt);
    for(ll i=0; i<m3; i++)
        cin>> v,cnt[v]++;
    countF(2);

    cin>> m4;
```

```
    memset(cnt,0,sizeof cnt);
    for(ll i=0; i<m4; i++)
        cin>> v,cnt[v]++;
    countF(3);

    countRes();

    ll mul=(((m1*m2)%mod)*((m3*m4)%mod))%mod;
    mul=bigMod(mul,mod-2);
    for(ll i=0; i<(1<<N); i++)
    {
        res[i]=(res[i]*mul)%mod;
        cout<< res[i] << "\n";
    }
    return 0;
}
```

6.4 Sum Over Submasks DP

```
                /**
                        Minimum sum of K Submask of a Mask
                        where All On Bits of the Mask exist
                        at least once in the K Submasks
                **/


ll submask[(1<<16)][16];

int main()
{

    FIO;
//    IN;
//    OUT;

    ll N,K;
    cin>> N >> K;
    vector<pair<pll,ll> >inp(N);
    for(ll i=0;i<N;i++) cin>> inp[i].ff.ff >> inp[i].ff.ss >> inp[i].ss;


    for(ll mask=0;mask<(1<<N);mask++){
        ll sum=0;
        ll h=0,w=0;
        ll cnt=0;
        for(ll i=0;i<N;i++){
            if((1<<i)&mask) {
                sum+=inp[i].ff.ff*inp[i].ff.ss*inp[i].ss;
                h=max(h,inp[i].ff.ff);
                w=max(w,inp[i].ff.ss);
                cnt+=inp[i].ss;
            }
        }
        submask[mask][1]=h*w*cnt-sum;
    }

    for(ll k=2;k<=K;k++){
        for (ll mask=0; mask<(1<<N); ++mask){
            submask[mask][k]=INF;
            for (ll s=mask; s; s=(s-1)&mask){
                submask[mask][k]=min(submask[mask][k],submask[s][1]+submask[m
                submask[mask][k]=min(submask[mask][k],submask[s][k-1]+submask
            }
```

```
            }
        }

        cout<< submask[(1<<N)-1][K];

    }
```

6.5 Tree DP Sample

```
                        /**---------End of Template---------**/

                /** Maximum multiplication of sizes of connected components o


vll G[MX];
ll sz[MX];
BigInt dp[MX][MX];
bool vis[MX];
BigInt temp[MX][MX];

void DFS(ll u, ll p)
{
    sz[u]=1;
    for(ll i=0;i<MX;i++) temp[u][i]=Integer(1);

    for(auto v:G[u]){
        if(v==p) continue;
        DFS(v,u);
        for(ll i=sz[u];i>=1;i--){
            for(ll j=sz[v];j>=0;j--){
                temp[u][i+j]=max(temp[u][i]*dp[v][j],temp[u][i+j]);
            }
        }
        sz[u]+=sz[v];
    }
    for(ll i=1;i<=sz[u];i++) dp[u][i]=temp[u][i],temp[u][0]=max(temp[u][i]*i,
    dp[u][0]=temp[u][0];
}

int main()
{

    FIO;

    ll N;
    cin>> N;
    for(ll i=1;i<N;i++){
        ll u,v;
        cin>> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
```

```
    DFS(1,-1);

    cout<< dp[1][0];
}
```

7 Miscellaneous

7.1 Common Template Code

```
#include<bits/stdc++.h>
using namespace std;

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds;
using namespace __gnu_cxx;

#define  FIO        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
#define  IN         freopen("input.txt","r",stdin);
#define  OUT        freopen("output.txt","w",stdout);
#define  debug(x)   cout<< #x << " --> " << x << "\n";
#define  NL         printf("\n");
#define  case(x)    printf("Case %lld: ",x);
#define  readI(x)   scanf("%d",&x);
#define  readL(x)   scanf("%lld",&x);
#define  writeI(x)  printf("%d",x);
#define  writeL(X)  printf("%lld",x);
#define  all(v)     v.begin(),v.end()

#define  ll     long long
#define  ld     long double
#define  pb     push_back
#define  pii    pair< int,int >
#define  pll    pair< ll,ll >
#define  vii    vector< int >
#define  vll    vector< ll >
#define  vss    vector< string >
#define  vdd    vector< double >
#define  vpi    vector< pii >
#define  vpl    vector< pll >
#define  vvi    vector< vii >
#define  vvl    vector< vll >
#define  PQ     priority_queue
#define  ff     first
#define  ss     second
#define  MX     100005
#define  mod    1000000007
#define  INF    10000000000000000
#define  EPS    1e-12
```

```
/* Special functions:
                find_by_order(k) --> returns iterator to the kth largest elem
                order_of_key(val) --> returns the number of items in a set th
*/

typedef tree<
ll,                                     // type long long
null_type,
less<ll>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

struct compare
{
    bool operator() (ll a, ll b)
    {
        return a>b;
    }
};

const ll p1=131,p2=137;
const ll mod1=1000000009;
const ll mod2=1000000007;

bool marked[MX];
vll primes;


inline ll bigMod(ll x, ll n)
{
    ll res=1;
    while(n){
        if(n&1) res=(res*x)%mod;
        x=(x*x)%mod;
        n=n>>1;
    }
    return res;
}

void sieve()
{
    marked[0]=marked[1]=true;
    for(ll i=2;i*i<MX;i++){
        if(marked[i]==false){
```

```
            for(ll j=i*i;j<MX;j+=i){
                marked[j]=true;
            }
        }
    }

    for(ll i=2;i<MX;i++)
        if(!marked[i]){
            primes.push_back(i);
        }
}

vpl primeFactors(ll N)
{
        vpl factors;
        ll pf_id=0,pf=primes[pf_id];
        while(pf*pf<=N){
        ll cnt=0;
                while(N%pf==0) N/=pf,cnt++;
        if(cnt) factors.push_back({pf,cnt});

                pf=primes[++pf_id];
        }

        if(N!=1) factors.push_back({N,1});
    return factors;
}


struct pair_hash {
    template <class T1, class T2>
    size_t operator () (const pair<T1,T2> &p) const {
        auto h1 = hash<T1>{}(p.first);
        auto h2 = hash<T2>{}(p.second);

        // Mainly for demonstration purposes, i.e. works but is overly simple
        // In the real world, use sth. like boost.hash_combine
        return h1 ^ h2;
    }
};

struct chash {
        static uint64_t splitmix64(uint64_t x) {
                // http://xorshift.di.unimi.it/splitmix64.c
                x += 0x9e3779b97f4a7c15;
                x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
```

```
                x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
                return x ^ (x >> 31);
        }
        // Note: when casting y to unsigned x, x will be least unsigned int c
        // to y mod 2^64.
        size_t operator()(uint64_t x) const {
                static const uint64_t FIXED_RANDOM = chrono::steady_clock::no
                return splitmix64(x + FIXED_RANDOM);
        }
};
gp_hash_table<ll, ll, chash>var;


                        /**---------End of Template---------**/



int main()
{

    FIO;
    IN;
    OUT;

}
```

7.2 GP Hash Table Samples

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

// For integer
gp_hash_table<int, int> table;

// Custom hash function approach is better
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().co
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;

const ll TIME = chrono::high_resolution_clock::now().time_since_epoch().count
const ll SEED = (ll)(new ll);
const ll RANDOM = TIME ^ SEED;
const ll MOD = (int)1e9+7;
const ll MUL = (int)1e6+3;
 struct chash{
    ll operator()(ll x) const { return std::hash<ll>{}((x ^ RANDOM) % MOD * M
};
gp_hash_table<ll, int, chash> table;

unsigned hash_f(unsigned x) {
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = (x >> 16) ^ x;
    return x;
}
struct chash {
    int operator()(ll x) const { return hash_f(x); }
};
gp_hash_table<ll, int, chash> table[N][N];
// so table[i][j][k] is storing an integer for corresponding k as hash
unsigned hash_combine(unsigned a, unsigned b) { return a * 31 + b; }

// For pairs
// The better the hash function, the less collisions
// Note that hash function should not be costly
struct chash {
    int operator()(pii x) const { return x.first* 31 + x.second; }
};
gp_hash_table<pii, int, chash> table;
```

```
// Another recommended hash function by neal on CF
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<ll,int,custom_hash> safe_gp_hash_table;
unordered_map<ll,int,custom_hash> safe_umap;

typedef gp_hash_table<int, int, hash<int>,
equal_to<int>, direct_mod_range_hashing<int>, linear_probe_fn<>,
hash_standard_resize_policy<hash_prime_size_policy,
hash_load_check_resize_trigger<true>, true>>
gp;
gp Tree;
// Now Tree can probably be used for fenwick, indices can be long long
// S is an offset to handle negative value
// If values can be >= -1e9, S=1e9+1
// maxfen is the MAXN in fenwick, this case it was 2e9+2;
// Note that it was okay to declare gp in integer as the values were
// still in the range of int.
void add(long long p, int v) {
    for (p += S; p < maxfen; p += p & -p)
        Tree[p] += v;
}
int sum(int p) {
    int ans = 0;
    for (p += S; p; p ^= p & -p)
        ans += Tree[p];
    return ans;
}
```

7.3 Matrix Exponentiation

```
ll mod;

class Matrix{

public:
    vvl mat;
    ll r,c;

    Matrix(ll r, ll c)
    {
        this->r=r,this->c=c;
        vll v(c); for(ll i=0;i<r;i++) mat.push_back(v);
        for(ll i=0;i<r;i++)
            for(ll j=0;j<c;j++)
                mat[i][j]=0;
    }

    Matrix(vvl ip)
    {
        r=ip.size(),c=ip[0].size();
        for(ll i=0;i<r;i++)
            for(ll j=0;j<c;j++)
                mat[i][j]=ip[i][j];
    }

    Matrix operator*(Matrix rhs)
    {
        Matrix temp(r,rhs.c);
        for(ll i=0;i<r;i++){
            for(ll j=0;j<c;j++){
                temp.mat[i][j]=0;
                for(ll k=0;k<c;k++)
                    temp.mat[i][j]=(temp.mat[i][j]+mat[i][k]*rhs.mat[k][j])%m
            }
        }

        return temp;
    }
};

Matrix matPow(Matrix m, ll p)
```

```
{
    if(!p){
        Matrix temp(m.r,m.c);
        for(ll i=0;i<m.r;i++) temp.mat[i][i]=1;
        return temp;
    }

    Matrix temp=matPow(m,p/2);
    temp=temp*temp;
    if(p%2) temp=temp*m;

    return temp;
}

/**

F(n) = a1*F(n-1) + a2*F(n-2) + a3*F(n-3) + .... + akF(n-k);

k*k Matrix:
|0 0 0 . . . a1|
|1 0 0 . . . a2|
|0 1 0 . . . a3|
.  . . . . . . .
.  . . . . . . .
.  . . . . . . .
|0 0 . . . . ak|

**/


int main()
{

//    FIO;
//    IN;
//    OUT;

    ll T;
    cin>> T;
    for(ll cs=1;cs<=T;cs++){
        ll a,b,n,m;
        cin>> a >> b >> n >> m;

        Matrix mat(2,2);
        mat.mat[0][1]=mat.mat[1][0]=mat.mat[1][1]=1;
        mod=pow(10,m);
```

```
        if(n>1) mat=matPow(mat,n-1);

        ll res=(a*mat.mat[0][1]+b*mat.mat[1][1])%mod;
        if(n==0) res=a%mod;
        else if(n==1) res=b%mod;
        cout<< "Case␣" << cs << ":␣" << res << "\n";
    }

}
```