Contents

1 Range Queries

1.1 Segment Tree with Lazy Propagation

시간복잡도 : 쿼리마다 $O(\log N)$

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

using namespace std;

struct SegTree{
    vector<int> tree, lazy;
    int base;
    SegTree(int a){
        base=1;
        while(base<a) base<<=1;
        tree.resize(base*2+2);
        lazy.resize(base*2+2);
        base--;
    }
    void propagate(int ns, int nf, int num){
        if(lazy[num]!=0){
            if(ns<nf){
                lazy[num*2]+=lazy[num];
                lazy[num*2+1]+=lazy[num];
            }
            tree[num]+=lazy[num]*(nf-ns+1);
            lazy[num]=0;
        }
    }
    void update(int idx, int val){
        idx+=base;
        tree[idx]=val;
        idx>>=1;
        while(idx!=0){
            tree[idx]=tree[idx*2]+tree[idx*2+1];
            idx>>=1;
        }
    }
    int query(int st, int fn, int ns=1, int nf=-1, int num=1){
        if(nf==-1) nf=base+1;
        propagate(ns,nf,num);
        if(ns>fn || nf<st) return 0;
        if(st<=ns && nf<=fn) return tree[num];
        int mid=(ns+nf)>>1;
        return query(st,fn,ns,mid,num*2)+query(st,fn,mid+1,nf,num*2+1);
    }
    void add_value(int val, int st, int fn, int ns=1, int nf=-1, int num=1){
        if(nf==-1) nf=base+1;
        propagate(ns,nf,num);
        if(ns>fn || nf<st) return;
        if(st<=ns && nf<=fn){
            lazy[num]+=val;
            propagate(ns,nf,num);
            return;
        }
        int mid=(ns+nf)>>1;
        add_value(val,st,fn,ns,mid,num*2);
        add_value(val,st,fn,mid+1,nf,num*2+1);
        tree[num]=tree[num*2]+tree[num*2+1];
    }
};
```

1.2 Persistent Segment Tree

시간복잡도
setup : $O(N)$ 그 이외의 연산 : $O(\log N)$

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

using namespace std;

```

```cpp
struct PST{
    vector<int> tree, left, right;
    vector<int> root;
    int height, t, base;
    //a=minimum # of node at leaf level
    //b=total # of updating operation
    //rn=maximum # of root
    PST(int a, int b, int rn){
        base=height=t=1;
        while(base<a) height++, base=base<<1;
        int tmp=base*2+2+height*b+5;
        tree.resize(tmp);
        left.resize(tmp);
        right.resize(tmp);
        root.resize(rn+2);
        root[0]=setup(1,base);
    }
    int setup(int ns, int nf){
        int k=t++;
        tree[k]=0;
        if(ns<nf){
            int mid=(ns+nf)>>1;
            left[k]=setup(ns,mid);
            right[k]=setup(mid+1,nf);
        }
        return k;
    }
    void update_Kth_tree(int k, int idx, int val){
        if(root[k]==0) root[k]=root[k-1];
        root[k]=make(root[k],idx,val);
    }
    int make(int bef, int idx, int val, int ns=1, int nf=-1){
        if(nf==-1) nf=base;
        if(idx<ns || nf<idx) return bef;
        int k=t++;
        if(ns==nf) tree[k]=tree[bef]+val;
        else{
            int mid=(ns+nf)>>1;
            left[k]=make(left[bef],idx,val,ns,mid);
            right[k]=make(right[bef],idx,val,mid+1,nf);
            tree[k]=tree[left[k]]+tree[right[k]];
        }
        return k;
    }
    //To get the value of Kth segtree -> get_sum(T.root[k],...)
    int get_sum(int num, int st, int fn, int ns=1, int nf=-1){
        if(nf==-1) nf=base;
        if(fn<ns || nf<st) return 0;
        if(st<=ns && nf<=fn) return tree[num];
        int mid=(ns+nf)>>1;
        return get_sum(left[num],st,fn,ns,mid)+get_sum(right[num],st,fn,mid+1,nf);
    }
};
```

2 Dynamic Programming

2.1 Convex Hull Optimization

$dp[i] = \min(dp[j] + a[i]b[j]) \quad (j < i, b[i-1] \geq b[i])$
$dp[i] = \max(dp[j] + a[i]b[j]) \quad (j < i, b[i-1] \leq b[i])$
위의 두 형태의 점화식을 기울기가 $b[j]$, y절편이 $dp[j]$인 직선 j에 대하여 $x = a[i]$의 함수값의 최소/최대를 찾는 것으로 해석하여 최적화하는 기법, 위의 꼴로 점화식을 변환하되, 직선의 x자리에 모든 변수를 묶어서 정리하면 기울기와 y절편을 결정하는데 유용하다.

2.1.1 Convex Hull Trick

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

```cpp

using namespace std;
typedef long long ll;

struct line{
    ll slope, constant;
    line() : slope(0), constant(0){}
    line(ll a, ll b) : slope(a), constant(b){}
};
// x should be monotonic increasing
struct CHT{
    int sz;
    deque<line> L;
    CHT() : sz(0) {}
    double get_point(line L1, line L2){
        return (double)(L2.constant-L1.constant)/(L1.slope-L2.slope);
    }
    bool increase(line L1, line L2, line L3){
        double p1=get_point(L1,L2);
        double p2=get_point(L2,L3);
        return p1<=p2;
    }
    void update(line L3){
        while(sz>1){
            line L2=L.back();
            L.pop_back();
            sz--;
            line L1=L.back();
            if(increase(L1,L2,L3)){
                L.push_back(L2);
                sz++;
                break;
            }
        }
        L.push_back(L3);
        sz++;
    }
    ll query(ll x){
        while(sz>1 && get_point(L[0],L[1])<(double)x){
            L.pop_front();
            sz--;
        }
        line L1=L.front();
        return L1.slope*x+L1.constant;
    }
};
```

시간복잡도 : $O(N)$

2.1.2 Li-Chao Tree

시간복잡도 : $O(NlogX)$
최댓값 Li-Chao Tree, 최솟값은 직선 삽입/값 찾을 때 부호를 반대로 넣으면 된다. LiChaoTree T; T.init(최소x좌표, 최대x좌표); T.insert(0,{기울기,y절편}); T.get(0,x좌표);

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

using namespace std;
typedef long long ll;
typedef pair<ll, ll> Line;

struct LiChaoTree{
    ll f(Line l, ll x){
        return l.first * x + l.second;
    }
    struct Node{
        int lnode, rnode;
        ll xl, xr;
        Line l;
    };
    vector<Node> nodes;
    void init(ll xmin, ll xmax){
        nodes.push_back({-1,-1,xmin,xmax,{0,-1e18}});
    }
    void insert(int n, Line newline){
        ll xl = nodes[n].xl, xr = nodes[n].xr;
        ll xm = (xl + xr) >> 1;
        Line llow = nodes[n].l, lhigh = newline;
        if( f(llow, xl) >= f(lhigh,xl) ) swap(llow, lhigh);
        if( f(llow, xr) <= f(lhigh, xr) ){
            nodes[n].l = lhigh;
            return;
        }
        else if( f(llow, xm) <= f(lhigh, xm) ){
            nodes[n].l = lhigh;
            if( nodes[n].rnode == -1 ){
                nodes[n].rnode = nodes.size();
                nodes.push_back({-1,-1,xm+1,xr,{0,-1e18}});
            }
            insert(nodes[n].rnode, llow);
        }
        else{
            nodes[n].l = llow;
            if( nodes[n].lnode == -1 ){
                nodes[n].lnode = nodes.size();
                nodes.push_back({-1,-1,xl,xm,{0,-1e18}});
            }
            insert(nodes[n].lnode, lhigh);
        }
    }
    ll get(int n, ll xq){
        if( n == -1 ) return -1e18;
        ll xl = nodes[n].xl, xr = nodes[n].xr;
        ll xm = (xl + xr) >> 1;
        if( xq <= xm ) return max(f(nodes[n].l, xq), get(nodes[n].lnode, xq));
        else return max(f(nodes[n].l, xq), get(nodes[n].rnode, xq));
    }
};
```

2.2 Knuth Optimization

1. $dp[i][j] = min(dp[i][k] + dp[k+1][j]) + cost[i][j] \qquad (i \leq k < j)$
2. $cost[b][c] \leq cost[a][d] \qquad (a \leq b \leq c \leq d)$ $\Leftarrow$ Monotonicity (단조성)
3. $cost[a][c] + cost[b][d] \leq cost[a][d] + cost[b][c] \qquad (a \leq b \leq c \leq d)$ $\Leftarrow$ Quadrangle Inequality (사각 부등식)

1, 2, 3이 성립할 때, 다음이 성립한다.
$pos[i][j-1] \leq pos[i][j] \leq pos[i+1][j] \qquad (pos[i][j] = dp[i][j]$를 최소화 하는 k의 위치)
따라서 k의 위치를 $1 \sim N$까지 순회할 필요없이 위의 부등식 범위만 고려하여 순회함으로써 시간복잡도를 $O(N^3)$에서 $O(N^2)$으로 줄일 수 있다.
2015년도 인터넷 예선 F번 Merging Files

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

int tc, n;
int m[5005];
int dp[5005][5005], pos[5005][5005];
int sum[5005];
const int INF=1e9;

int main()
{
    for(scanf("%d",&tc) ; tc>0 ; tc--){
        scanf("%d",&n);
        for(int i=1 ; i<=n ; i++){
            scanf("%d",&m[i]);
            pos[i][i]=i;
            sum[i]=sum[i-1]+m[i];
        }
        for(int len=2 ; len<=n ; len++){
            for(int i=1 ; i<=n-len+1 ; i++){
                dp[i][i+len-1]=INF;
                int s=pos[i][i+len-2], f=pos[i+1][i+len-1];
                for(int j=s ; j<=f ; j++){
                    if(j<n && dp[i][i+len-1]>dp[i][j]+dp[j+1][i+len-1]){
                        pos[i][i+len-1]=j;
                        dp[i][i+len-1]=dp[i][j]+dp[j+1][i+len-1];
                    }
                }
                dp[i][i+len-1]+=sum[i+len-1]-sum[i-1];
            }
        }
        printf("%d\n",dp[1][n]);
    }
    return 0;
}
```

2.3 Lowest Common Ancestor

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

using namespace std;

int n;
int a,b;
vector<int> v[100005];
int q[100005][2];
bool visit[100005]={1};
int anc[100005][20];
int lvl[100005];

void init(){
    int s=0,e=1;
    q[0][0]=1;
    while(s<e){
        int idx=q[s][0];
        lvl[idx]=q[s++][1];
        visit[idx]=1;
        for(auto x:v[idx]){
            if(!visit[x]){
                anc[x][0]=idx;
                q[e][0]=x;
                q[e++][1]=lvl[idx]+1;
            }
        }
    }
    for(int i=1; i<20; i++){
        for(int j=1; j<=n; j++){
            anc[j][i]=anc[anc[j][i-1]][i-1];
        }
    }
}
int lca(int x, int y){
    int temp=lvl[x]-lvl[y];
    if(temp<0){
        temp=-temp;
        swap(x,y);
    }
    for(int i=0; temp; i++){
        if(temp&(1<<i)){
            x=anc[x][i];
            temp-=1<<i;
        }
    }
    if(x==y) return x;
    for(int i=19; x!=y && i>=0; i--){
        if(anc[x][i] && anc[y][i] && anc[x][i]!=anc[y][i]){
            x=anc[x][i];
            y=anc[y][i];
        }
    }
    return anc[x][0];
}

```

```
int main(){
    scanf("%d",&n);
    for(int x,y,i=0; i<n-1; i++){
        scanf("%d%d",&x,&y);
        v[x].push_back(y);
        v[y].push_back(x);
    }
    init();
    int m;
    scanf("%d",&m);
    while(m--){
        int x,y;
        scanf("%d%d",&x,&y);
        printf("%d\n",lca(x,y));
    }
    return 0;
}
```

2.4 Berlekamp-Massey

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

using namespace std;

const int mod = 998244353;
using ll = long long;
ll ipow(ll x, ll p){
        ll ret = 1, piv = x;
        while(p){
                if(p & 1) ret = ret * piv % mod;
                piv = piv * piv % mod;
                p >>= 1;
        }
        return ret;
}
vector<int> berlekamp_massey(vector<int>& x){
        vector<int> ls, cur;
        int lf, ld;
        for(int i=0; i<x.size(); i++){
                ll t = 0;
                for(int j=0; j<cur.size(); j++){
                        t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
                }
                if((t - x[i]) % mod == 0) continue;
                if(cur.empty()){
                        cur.resize(i+1);
                        lf = i;
                        ld = (t - x[i]) % mod;
                        continue;
                }
                ll k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
                vector<int> c(i-lf-1);
                c.push_back(k);
                for(auto &j : ls) c.push_back(-j * k % mod);
                if(c.size() < cur.size()) c.resize(cur.size());
                for(int j=0; j<cur.size(); j++){
                        c[j] = (c[j] + cur[j]) % mod;
                }
                if(i-lf+(int)ls.size()>=(int)cur.size()){
                        tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
                }
                cur = c;
        }
        for(auto &i : cur) i = (i % mod + mod) % mod;
        return cur;
}
int get_nth(vector<int>& rec, vector<int>& dp, ll n){
        int m = rec.size();
        vector<int> s(m), t(m);
        s[0] = 1;
        if(m != 1) t[1] = 1;
```

```
	else t[0] = rec[0];
	auto mul = [&rec](vector<int>& v, vector<int>& w){
		int m = v.size();
		vector<int> t(2 * m);
		for(int j=0; j<m; j++){
			for(int k=0; k<m; k++){
				t[j+k] += 1ll * v[j] * w[k] % mod;
				if(t[j+k] >= mod) t[j+k] -= mod;
			}
		}
		for(int j=2*m-1; j>=m; j--){
			for(int k=1; k<=m; k++){
				t[j-k] += 1ll * t[j] * rec[k-1] % mod;
				if(t[j-k] >= mod) t[j-k] -= mod;
			}
		}
		t.resize(m);
		return t;
	};
	while(n){
		if(n & 1) s = mul(s, t);
		t = mul(t, t);
		n >>= 1;
	}
	ll ret = 0;
	for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
	return ret % mod;
}
int guess_nth_term(vector<int> x, ll n){
	if(n < x.size()) return x[n];
	vector<int> v = berlekamp_massey(x);
	if(v.empty()) return 0;
	return get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
	// smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
	vector<int> rnd1, rnd2;
	mt19937 rng(0x14004);
	auto randint = [&rng](int lb, int ub){
		return uniform_int_distribution<int>(lb, ub)(rng);
	};
	for(int i=0; i<n; i++){
		rnd1.push_back(randint(1, mod - 1));
		rnd2.push_back(randint(1, mod - 1));
	}
	vector<int> gobs;
	for(int i=0; i<2*n+2; i++){
		int tmp = 0;
		for(int j=0; j<n; j++){
			tmp += 1ll * rnd2[j] * rnd1[j] % mod;
			if(tmp >= mod) tmp -= mod;
		}
		gobs.push_back(tmp);
		vector<int> nxt(n);
		for(auto &i : M){
			nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
			if(nxt[i.x] >= mod) nxt[i.x] -= mod;
		}
		rnd1 = nxt;
	}
	auto sol = berlekamp_massey(gobs);
	reverse(sol.begin(), sol.end());
	return sol;
}
ll det(int n, vector<elem> M){
	vector<int> rnd;
	mt19937 rng(0x14004);
	auto randint = [&rng](int lb, int ub){
		return uniform_int_distribution<int>(lb, ub)(rng);
	};
	for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
	for(auto &i : M){
```

```
                i.v = 1ll * i.v * rnd[i.y] % mod;
        }
        auto sol = get_min_poly(n, M)[0];
        if(n % 2 == 0) sol = mod - sol;
        for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
        return sol;
}
int main()
{
        cout<<guess_nth_term({0,1,1,2,3,5,8},10);
        return 0;
}
```

3 Divide & Conquer

3.1 Repeated matrix squaring

시간복잡도 : $O(D^3 \log N)$ $(D =$행렬크기$)$

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

using namespace std;
typedef vector<vector<int> > matrix;

int d, t;
const int MOD=31991;

matrix multiply(matrix A, matrix B)
{
    matrix C(d,vector<int>(d,0));
    for(int i=0 ; i<d ; i++)
        for(int j=0 ; j<d ; j++)
            for(int k=0 ; k<d ; k++) (C[i][j]+=A[i][k]*B[k][j])%=MOD;
    return C;
}

matrix get_pow(int p, matrix M)
{
    if(p==1) return M;
    matrix ret=get_pow(p/2,M);
    ret=multiply(ret,ret);
    if(p%2==1) ret=multiply(ret,M);
    return ret;
}

int main()
{
    scanf("%d %d",&d,&t);
    matrix base(d,vector<int>(d,0));
    for(int i=1 ; i<d ; i++) base[i-1][i]=1;
    for(int i=0 ; i<d ; i++) base[d-1][i]=1;
    matrix res=base;
    int sum=0;
    if(t>1){
        res=get_pow(t-1,base);
        for(int i=0 ; i<d ; i++) (sum+=res[i][d-1])%=MOD;
    }
    else sum=1;
    printf("%d",sum);
    return 0;
}
```

4 Graph

4.1 Dijkstra

시간복잡도 : $O(E \log V)$

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

using namespace std;
typedef pair<int,int> P;

int n, st;
vector<P> lst[20002];
priority_queue<P, vector<P>, greater<P> > PQ;
int dist[20002];
bool visit[20002];
const int INF=1e9;

int main()
{
    int m, a, b, c, v;
    scanf("%d %d",&n,&m);
    scanf("%d",&st);
    for(int i=0 ; i<m ; i++){
        scanf("%d %d %d",&a,&b,&c);
        lst[a].push_back(P(b,c));
    }
    for(int i=1 ; i<=n ; i++) dist[i]=INF;
    dist[st]=0;
    PQ.push(P(0,st));
    while(!PQ.empty()){
        do{
            v=PQ.top().second;
            PQ.pop();
        }while(!PQ.empty() && visit[v]);
        visit[v]=true;
        for(int i=0 ; i<lst[v].size() ; i++){
            if(dist[lst[v][i].first]>dist[v]+lst[v][i].second){
                dist[lst[v][i].first]=dist[v]+lst[v][i].second;
                PQ.push(P(dist[lst[v][i].first],lst[v][i].first));
            }
        }
    }
    for(int i=1 ; i<=n ; i++){
        if(dist[i]==INF) printf("INF\n");
        else printf("%d\n",dist[i]);
    }
    return 0;
}
```

4.2 Strongly Connected Component

시간복잡도 : $O(N)$

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

using namespace std;

int n, m;
vector<int> lst[10002];
vector<vector<int> > res;
int dfsn[10002], cnt, scc;
stack<int> S;
bool finished[10002];

int dfs(int v)
{
    dfsn[v]=++cnt;
    S.push(v);
    int minv=dfsn[v];
    for(int i=0 ; i<lst[v].size() ; i++){
        int nxt=lst[v][i];
        if(dfsn[nxt]==0) minv=min(minv,dfs(nxt));
        else if(!finished[nxt]) minv=min(minv,dfsn[nxt]);
    }
    if(dfsn[v]==minv){
        vector<int> res1;
        while(1){
            int c=S.top();
            S.pop();
            finished[c]=true;
            res1.push_back(c);
            if(c==v) break;
        }
        scc++;
        sort(res1.begin(),res1.end());
        res.push_back(res1);
    }
    return minv;
}

int main()
{
    scanf("%d %d",&n,&m);
    for(int i=0 ; i<m ; i++){
        int a, b;
        scanf("%d %d",&a,&b);
        lst[a].push_back(b);
    }
    for(int i=1 ; i<=n ; i++) if(dfsn[i]==0) dfs(i);
    sort(res.begin(),res.end());
    printf("%d\n",scc);
    for(int i=0 ; i<scc ; i++){
        for(int j=0 ; j<res[i].size() ; j++) printf("%d ",res[i][j]);
        printf("-1\n");
    }
    return 0;
}
```

4.3 2-SAT

시간복잡도 : $O(N)$
2-SAT에 사용되는 CNF(Closure Normal Form)
$f = (A \lor B) \land (B \lor C) \land (\neg C \lor \neg D)$
위와 같이 하나의 closure에는 최대 2개의 변수 또는 NOT변수들의 OR 연산으로 이루어져 있으며, 각각의 closure는 AND연산을 통해 연결되어 있다. 실제 문제는 위와 같은 형태로 처음부터 나타낼 수는 없으며 아래의 법칙 등을 사용하거나 벤다이어그램을 그려봄으로써 2-SAT에 올바른 CNF를 찾아내야 한다. CNF를 찾아냈다면 이제는 OR 연산을 바탕으로 그래프의 Edge를 만들어야 한다. $A \lor B$ 가 주어지면 $\neg A \to B$ 와 $\neg B \to A$를 모두 추가하는 것을 잊지 말자.

CNF(Closure Normal Form)를 만들 때 사용되는 법칙
$\lor$ 는 덧셈으로, $\land$ 는 곱셈으로 생각하면 산수에 적용되는 교환/결합/분배 법칙과 동일하게 연산할 수 있다.

1. 교환 법칙
$f = A \land B = B \land A$
$f = A \lor B = B \lor A$
2. 결합 법칙
$f = (A \lor B) \lor C = A \lor (B \lor C)$
$f = (A \land B) \land C = A \land (B \land C)$
3. 분배 법칙
$f = A \land (B \lor C) = (A \land B) \lor (A \land C)$
$f = A \lor (B \land C) = (A \lor B) \land (A \lor C)$
4. 동일 법칙
$f = A \land A = A$
$f = A \lor A = A$
5. 흡수 법칙
$f = A \lor (A \land B) = A$
$f = A \land (A \lor B) = A$
6. 드모르간 법칙
$f = \neg(A \land B) = \neg A \lor \neg B$
7. 알아두면 좋은 식

$f = (A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C) = (A \vee A) \wedge (B \vee C)$ 분배법칙의 첫번째는 거꾸로 생각하자.
$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge A) = (A \vee B) \wedge (B \vee C) \wedge (C \vee A)$ 2018 Seoul Regional 기출
$f = (A \wedge B) \vee (C \wedge D) = (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$

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

using namespace std;

int n, m;
vector<int> lst[20002];
int dfsn[20002], cnt, scc_num[20002], scc_cnt;
stack<int> S;

int dfs(int v)
{
    dfsn[v]=++cnt;
    S.push(v);
    int minv=dfsn[v];
    for(int i=0 ; i<lst[v].size() ; i++){
        int nxt=lst[v][i];
        if(dfsn[nxt]==0) minv=min(minv,dfs(nxt));
        else if(scc_num[nxt]==0) minv=min(minv,dfsn[nxt]);
    }
    if(dfsn[v]==minv){
        scc_cnt++;
        while(1){
            int cur=S.top();
            S.pop();
            scc_num[cur]=scc_cnt;
            if(v==cur) break;
        }
    }
    return minv;
}

int main()
{
    scanf("%d %d",&n,&m);
    for(int i=1 ; i<=m ; i++){
        int a, b;
        scanf("%d %d",&a,&b);
        lst[n-a].push_back(b+n);
        lst[n-b].push_back(a+n);
    }
    for(int i=0 ; i<=2*n ; i++) if(dfsn[i]==0 && i!=n) dfs(i);
    for(int i=1 ; i<=n ; i++){
        if(scc_num[n+i]==scc_num[n-i]){
            printf("0");
            return 0;
        }
    }
    printf("1\n");
    for(int i=1 ; i<=n ; i++){
        // SCC_num of not Xn is greater than SCC_num of Xn means
        // the result of topological sort by SCC -> not Xn has higher priority
        // therefore not Xn should be false -> Xn is true
        if(scc_num[n+i]<scc_num[n-i]) printf("1 ");
        else printf("0 ");
    }
    return 0;
}
```

4.4 Finding Diameter of Tree

시간복잡도 : $O(V+E)$

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

using namespace std;
typedef pair<int,int> P;

```

```cpp
int n;
vector<P> lst[10002];
bool check[10002];
int d[10002];
queue<int> Q;

int bfs(int st)
{
    int rd=0, rv=st;
    memset(check,false,sizeof(check));
    memset(d,0,sizeof(d));
    Q.push(st);
    check[st]=true;
    while(!Q.empty()){
        int v=Q.front();
        Q.pop();
        if(rd<d[v]) rd=d[v], rv=v;
        for(int i=0 ; i<lst[v].size() ; i++){
            int nxt=lst[v][i].first;
            if(!check[nxt]){
                check[nxt]=true;
                d[nxt]=d[v]+lst[v][i].second;
                Q.push(nxt);
            }
        }
    }
    return rv;
}

int main()
{
    scanf("%d",&n);
    for(int i=1 ; i<n ; i++){
        int a, b, d;
        scanf("%d %d %d",&a,&b,&d);
        lst[a].push_back(P(b,d));
        lst[b].push_back(P(a,d));
    }
    int v1=bfs(1);
    int v2=bfs(v1);
    printf("%d",d[v2]);
    return 0;
}
```

5 Network Flow

5.1 Dinic

시간복잡도 : $O(V^2E)$

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

using namespace std;

struct Edge{
    int nxt, rev, cap, flw;
    Edge() {}
    Edge(int a, int b, int c) : nxt(a), rev(b), cap(c), flw(0) {}
};

int m, n;
int xy[102][102], num[102][102];
vector<Edge> lst[20005];
int SRC, SNK, res, work[20005], level[20005];
const int INF=1e9;

void add_edge(int v1, int v2, int f)
{
    lst[v1].emplace_back(v2,lst[v2].size(),f);
    lst[v2].emplace_back(v1,lst[v1].size()-1,0);
}
```

```

bool bfs()
{
    queue<int> Q;
    memset(level,-1,sizeof(level));
    level[SRC]=0;
    Q.push(SRC);
    while(!Q.empty()){
        int v=Q.front();
        Q.pop();
        for(int i=0 ; i<lst[v].size() ; i++){
            int nxt=lst[v][i].nxt;
            if(level[nxt]==-1 && lst[v][i].cap-lst[v][i].flw>0){
                Q.push(nxt);
                level[nxt]=level[v]+1;
            }
        }
    }
    return level[SNK]!=-1;
}

int dfs(int v, int fn, int flow)
{
    if(v==fn) return flow;
    for(int &i=work[v] ; i<lst[v].size() ; i++){
        int nxt=lst[v][i].nxt;
        if(level[nxt]==level[v]+1 && lst[v][i].cap-lst[v][i].flw>0){
            int df=dfs(nxt,fn,min(flow,lst[v][i].cap-lst[v][i].flw));
            if(df>0){
                lst[v][i].flw+=df;
                lst[nxt][lst[v][i].rev].flw-=df;
                return df;
            }
        }
    }
    return 0;
}

void dinic()
{
    while(bfs()){
        memset(work,0,sizeof(work));
        while(1){
            int flow=dfs(SRC,SNK,INF);
            if(flow==0) break;
            res+=flow;
        }
    }
}
```

5.2 Max-flow min-cut Theorem

같은 컴포넌트에 존재하는 임의의 두 정점 u와 v를 서로 다른 컴포넌트로 분리하기위해 제거해야 하는 간선의 가중치 값의 합의 최솟값은 u와 v를 각각 source와 sink로 하는 유량그래프에서 최대유량과 같다. 따라서 적절하게 유량그래프를 만든 뒤 Edmonds Karp나 Dinic을 사용하여 최대 유량을 구하면 된다.

5.3 Konig's Theorem

이분 그래프에서의 Minimum Vertex Cover는 Maximum Bipartite Matching과 같으며 실제 조건을 만족하는 정점 집합은 다음과 같다. 이분 그래프 상의 왼쪽 정점 집합을 L, 오른쪽 정점 집합을 R이라고 하자. 이때 다음과 같은 집합 X를 정의한다. $X = \{L$에 매칭되지 않은 정점들과, 그 정점으로부터 alternating path를 통해 도달할 수 있는 L, R의 모든 정점$\}$ 이렇게 세개의 집합을 정의하면 우리가 구하고자 하는 Vertex Cover집합 C는 다음과 같다. $C = (L - X) \cup (R \cap X)$

5.4 Hopcroft-Karp

시간복잡도 : $O(E\sqrt{V})$

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

using namespace std;

int n;
vector<int> lst[10005];
int G1[10005], G2[10005], dist[10005];
bool used[10005];
const int INF=1e9;
// size of lst, G1, dist and used is MAXN
// size of G2 is MAXM
// vertex number should be 1~n in G1 and 1~m in G2
// only add edges which is from G1 to G2

void bfs()
{
    queue<int> Q;
    for(int i=0 ; i<n ; i++){
        if(!used[i]) Q.push(i), dist[i]=0;
        else dist[i]=INF;
    }
    while(!Q.empty()){
        int v=Q.front();
        Q.pop();
        for(int i=0 ; i<lst[v].size() ; i++){
            int nxt=lst[v][i];
            if(G2[nxt]!=-1 && dist[G2[nxt]]==INF){
                dist[G2[nxt]]=dist[v]+1;
                Q.push(G2[nxt]);
            }
        }
    }
}

bool dfs(int v1)
{
    for(int i=0 ; i<lst[v1].size() ; i++){
        int v2=lst[v1][i];
        if(G2[v2]==-1 || (dist[G2[v2]]==dist[v1]+1 && dfs(G2[v2]))){
            used[v1]=true;
            G1[v1]=v2;
            G2[v2]=v1;
            return true;
        }
    }
    return false;
}

int Hopcroft_Karp()
{
    int matched=0;
    memset(used,false,sizeof(used));
    memset(G1,-1,sizeof(G1));
    memset(G2,-1,sizeof(G2));
    while(1){
        bfs();
        int flow=0;
        for(int i=0 ; i<n ; i++){
            if(!used[i] && dfs(i)) flow++;
        }
        if(flow==0) break;
        matched+=flow;
    }
    return matched;
}
```

5.5 Minimum Cost Maximum Flow (SPFA)

시간복잡도 : $O(VEf)$ (f =최대 유량, 플로우 그래프 특성 상 실제로는 $O(VEf)$ 보다는 빠르게 작동)
u에서 v로 가는 간선의 cost가 w면 역방향 간선의 cost는 $-w$이다.

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

using namespace std;

int n, m;
vector<int> lst[805];
int SRC, SNK, cst[805][805];
int cap[805][805], flw[805][805];
int rcost, rflow;
const int INF=1e9;

void MCMF()
{
    while(1){
        int prev[805], dist[805];
        bool InQ[805]={0};
        memset(prev,-1,sizeof(prev));
        for(int i=0 ; i<=SNK ; i++) dist[i]=INF;
        queue<int> Q;
        dist[SRC]=0;
        Q.push(SRC);
        InQ[SRC]=true;
        while(!Q.empty()){
            int v=Q.front();
            Q.pop();
            InQ[v]=false;
            for(int i=0 ; i<lst[v].size() ; i++){
                int nxt=lst[v][i];
                if(dist[nxt]>dist[v]+cst[v][nxt] && cap[v][nxt]-flw[v][nxt]>0){
                    prev[nxt]=v;
                    dist[nxt]=dist[v]+cst[v][nxt];
                    if(!InQ[nxt]) Q.push(nxt), InQ[nxt]=true;
                }
            }
        }
        if(prev[SNK]==-1) break;
        int v=SNK, flow=INF;
        while(v!=SRC){
            flow=min(flow,cap[prev[v]][v]-flw[prev[v]][v]);
            v=prev[v];
        }
        v=SNK;
        while(v!=SRC){
            flw[prev[v]][v]+=flow;
            flw[v][prev[v]]-=flow;
            rcost+=flow*cst[prev[v]][v];
            v=prev[v];
        }
        rflow+=flow;
    }
}
```

6 Graph Modeling Method

6.1 L-R maxflow

L-R max flow는 Flow Graph 중 각 edge의 유량에 대한 하한선과 상한선이 존재하는 경우에 대한 최대 유량을 찾는 방법이다. 모델링 방법은 기존의 Flow Graph에서 새로운 Source와 새로운 Sink를 추가하는 것이다. 새로운 Source와 Sink에 edge를 연결하는 방법은 다음과 같다. 정점 v1에서 v2로 하한유량 L과 상한유량 R을 갖는 edge가 있다고 가정하자. 새로운 Source에서 v2로 연결되는 capacity L짜리 edge를 연결하고, v1에서 새로운 Sink로 capacity가 L인 edge를 새롭게 연결하면 된다. 이후 마지막으로 기존의 Sink에서 기존의 Source로 가는 capacity가 무한대인 edge를 추가한 뒤, 기존의 그래프의 edge 중 상한과 하한을 갖는 edge들의 capacity를 R-L로 수정한다. 이렇게하면 L-R flow를 구하기 위한 새로운 그래프가 완성된다. 이제 조건을 만족하는 정답이 존재하는지 판별하는

방법은 새로운 Source에서 새로운 Sink로 들어오는 유량이 L_{tot}인지 확인하면되고, 만약 존재성이 밝혀진다면, 현재의 Flow Graph의 상태를 유지한채 기존의 Source에서 Sink로 가는 Maximum Flow를 구하여 L_{tot}에 더해주면 구하고자 하는 정답이된다.

7 Geometry

7.1 CCW

```
//ccw->1, cw->-1
int ccw(int a1, int b1, int a2, int b2, int a3, int b3)
{
        int ret=(a2-a1)*(b3-b2)-(a3-a2)*(b2-b1);
        if(ret==0) return 0;
        if(ret>0) return 1
        if(ret<0) return -1;
}
```

7.2 Graham Scan

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

using namespace std;
typedef long long ll;

struct A{
    ll x, y, p, q;
};

bool cmp1(const A &a, const A &b) {return a.y<b.y || (a.y==b.y && a.x<b.x);}
bool cmp2(const A &a, const A &b) {return a.p*b.q-a.q*b.p>0 || (a.p*b.q-a.q*b.p==0 && a.p*a.p+
  a.q*a.q<b.p*b.p+b.q*b.q);}
bool ccw(ll x1, ll y1, ll x2, ll y2) {return x1*y2-y1*x2>0;}

int n;
A m[100002];
stack<int> S;

int main()
{
    scanf("%d",&n);
    for(int i=1 ; i<=n ; i++) scanf("%lld %lld",&m[i].x,&m[i].y);
    sort(m+1,m+1+n,cmp1);
    for(int i=1 ; i<=n ; i++) m[i].p=m[i].x-m[1].x, m[i].q=m[i].y-m[1].y;
    sort(m+2,m+1+n,cmp2);
    S.push(1); S.push(2);
    for(int i=3 ; i<=n ; i++){
        while(S.size()!=1){
            int first=S.top();
            S.pop();
            int second=S.top();
            if(ccw(m[first].p-m[second].p,m[first].q-m[second].q,m[i].p-m[first].p,m[i].q-m[
              first].q)){
                S.push(first);
                break;
            }
        }
        S.push(i);
    }
    printf("%d",S.size());
    return 0;
}
```

7.3 Rotating Calipers

```
#include<bits/stdc++.h>
#define MAX 100000
```

```
#define INF 10000
using namespace std;
using ll = long long;

struct cor {
	ll x, y;
	cor(ll _x = 0, ll _y = 0) :x(_x), y(_y) {}
	bool operator==(cor a) { return x == a.x && y == a.y; }
	cor operator-(cor a) { return cor(x - a.x, y - a.y); }
	bool operator<(cor a) {
		if (y != a.y) return y < a.y;
		return x < a.x;
	}
	friend istream& operator>>(istream& stream, cor& a) { return stream >> a.x >> a.y; }
	friend ostream& operator<<(ostream& stream, cor& a) { return stream << a.x << ' ' << a
	  .y; }
	friend ll outpro(cor a, cor b) { return a.x * b.y - a.y * b.x; }
	friend ll ccw(cor a, cor b, cor c) {
		ll tmp = outpro(b - a, c - a);
		if (tmp > 0) return 1;
		if (tmp < 0) return -1;
		return 0;
	}
	friend ll dist(cor a, cor b) { return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y -
	   b.y); }
};
struct convex {
	ll sz;
	vector<cor> v;
	cor& operator[](ll a) { return v[(a + sz) % sz]; }
	void updt(cor& a) {
		convex& h = *this;
		while (sz >= 2) {
			ll tmp = ccw(h[-2], h[-1], a);
			if (tmp > 0) break;
			v.pop_back();
			--sz;
		}
		v.push_back(a);
		++sz;
	}
	double operator()() {
		convex& h = *this;
		ll l = 0, r = 0, m = 0;
		for (ll i = 0; i < sz; ++i) {
			if (h[m] < h[i]) m = i;
		}
		r = m;
		ll ret = dist(h[l], h[r]);
		while (l < m || r < sz) {
			ll tmp = outpro(h[l + 1] - h[l], h[r + 1] - h[r]);
			if (tmp > 0) ret = max(ret, dist(h[l], h[++r]));
			if (tmp < 0) ret = max(ret, dist(h[++l], h[r]));
			if (!tmp) ret = max(ret, dist(h[++l], h[++r]));
		}
		return sqrt(ret);
	}
};

ll c;
vector<cor> v;
cor bot(INF, INF);
convex hull;

int main() {
	cin >> c;
	while (c--) {
		cor x;
		cin >> x;
		if (x < bot) bot = x;
		v.push_back(x);
	}
	v.erase(unique(v.begin(), v.end()), v.end());
```

```
        c = v.size();
        sort(v.begin(), v.end(), [](cor& a, cor& b) {
                ll tmp = ccw(bot, a, b);
                if (!tmp) return a < b;
                return tmp > 0;
                });
        for (cor& x : v) {
                hull.updt(x);
        }
        cout << fixed;
        cout.precision(12);
        cout << hull();
        return 0;
}
```

8 Math

8.1 Euler's totient function

n이하의 자연수 중 n과 서로소인 수의 개수를 구하는 함수 $\phi(n) = n\prod_{p|n}(1-\frac{1}{p})$

8.2 Fermat's little Theorem

p가 소수이고 a가 p의 배수가 아니면 $a^{p-1} \equiv 1 \quad (mod\ p)$ 이다.
즉, a^{p-1}를 p로 나눈 나머지는 1이다.

8.3 Euler's Theorem

a와 n이 서로소인 자연수일때 다음 합동식이 성립한다.
$a^{\phi(n)} \equiv 1 \quad (mod\ n)$

8.4 Chinese Remainder Theorem

$x \equiv a_1 \pmod{n_1}$, $x \equiv a_2 \pmod{n_2}$, ... , $x \equiv a_k \pmod{n_k}$ 이고 $n_1, n_2, ..., n_k$는 pair-wise 서로소 관계일 때, 다음과 같은 x가 유일하게 존재한다.

$$x \equiv \sum_{i=1}^{k} a_i \times \left(\frac{N}{n_i}\right)^{\phi(n_i)} \pmod{N} \qquad \left(N = \prod_{i=1}^{k} n_i\right)$$

8.5 Landau's Theorem

n개의 팀이 토너먼트 경기를 할 때, 각 팀이 승리한 횟수를 오름차순으로 정렬한 결과를 $(s_1, s_2, ..., s_n)$ 라고 하면 $(s_1, s_2, ..., s_n)$ 이 유효하기 위한 필요충분조건은 다음과 같다.

1. $0 \le s_1 \le s_2 \le ... \le s_n$
2. $s_1 + s_2 + ... + s_i \ge \binom{i}{2} \quad (i = 1, 2, ..., n-1)$
3. $s_1 + s_2 + ... + s_n = \binom{n}{2}$

8.6 Fibonacci Sequence

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$\sum_{i=1}^{n} F_i = F_{n+2} - 1$ $\qquad$ $\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$

$\sum_{i=1}^{n} F_{2i-1} = F_{2n}$ $\qquad$ $\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$

$gcd(F_n, F_m) = F_{gcd(n,m)}$

$$F_{2n-1} = F_n^2 + F_{n-1}^2 \qquad F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n$$

8.7 Extended Euclidean Algorithm

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

using namespace std;

tuple<int, int, int> extended_euclidean(int a, int b) {
    if (b == 0)
        return make_tuple(a, 1, 0);
    int g, x, y;
    tie(g, x, y) = extended_euclidean(b, a%b);
    return make_tuple(g, y, x-(a/b)*y);
}
```

8.8 Finding Modular inverse in $O(N)$

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

using namespace std;

void modularInverse(int n, int prime)
{
        int dp[n + 1];
        dp[0] = dp[1] = 1;
        for (int i = 2; i <= n; i++)
                dp[i] = dp[prime % i] * (prime - prime / i) % prime;
}
```

8.9 Fast Fourier Transform

시간복잡도 : $O(N \log N)$

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

using namespace std;
typedef complex<double> base;
const double PI=3.14159265358979323846;

void FFT(vector<base> &a, bool invert)
{
    int N=a.size();
    for(int i=1, j=0 ; i<N ; i++){
        int bit=N>>1;
        for(; j>=bit ; bit>>=1) j-=bit;
        j+=bit;
        if(i<j) swap(a[i],a[j]);
    }
    for(int len=2 ; len<=N ; len<<=1){
        double ang=2*PI/len*(invert ? -1 : 1);
        base wlen(cos(ang),sin(ang));
        for(int i=0 ; i<N ; i+=len){
            base w(1);
            for(int j=0 ; j<len/2 ; j++){
                base u=a[i+j], v=a[i+j+len/2]*w;
                a[i+j]=u+v;
                a[i+j+len/2]=u-v;
                w*=wlen;
            }
        }
    }
    if(invert)
        for(int i=0 ; i<N ; i++) a[i]/=N;
}
//when result of A(x)*B(x) is needed, resize a and b by na+nb-1 before using function
//size of both a and b is equal to length of equation A(x) and B(x)
```

```
void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res)
{
    vector<base> fa(a.begin(),a.end());
    vector<base> fb(b.begin(),b.end());
    int N=1, na=a.size(), nb=b.size();
    while(N<max(na,nb)) N<<=1;
    fa.resize(N), fb.resize(N);
    FFT(fa,false), FFT(fb,false);
    for(int i=0 ; i<N ; i++) fa[i]*=fb[i];
    FFT(fa,true);
    res.resize(N);
    for(int i=0 ; i<N ; i++) res[i]=(int)(fa[i].real()+(fa[i].real()>0 ? 0.5 : -0.5));
}
```

9 String

9.1 KMP

시간복잡도 : $O(N + M)$

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

using namespace std;

int nh, ns;
char h[1000005], s[1000005]; // h=total string, s=target string
int fail[1000005]; // size=length of s
vector<int> res;

void KMP()
{
    nh=strlen(h);
    ns=strlen(s);
    for(int i=1, j=0 ; i<ns ; i++){
        fail[i]=0;
        while(j>0 && s[i]!=s[j]) j=fail[j-1];
        if(s[i]==s[j]) fail[i]=++j;
    }
    for(int i=0, j=0 ; i<nh ; i++){
        while(j>0 && h[i]!=s[j]) j=fail[j-1];
        if(h[i]==s[j]){
            if(j==ns-1){
                res.push_back(i-ns+1);
                j=fail[j];
            }
            else j++;
        }
    }
}
```

9.2 Trie

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

using namespace std;

struct Trie{
    Trie *nxt[10];
    bool epoint, nexist;
    Trie(){
        memset(nxt,0,sizeof(nxt));
        epoint=nexist=false;
    }
    ~Trie(){
        for(int i=0 ; i<10 ; i++) if(nxt[i]) delete nxt[i];
    }
    void update(const char *key){
        if(*key==0) epoint=true;
        else{
```

```cpp
            int idx=*key-'0';
            if(!nxt[idx]) nxt[idx]=new Trie;
            nexist=true;
            nxt[idx]->update(key+1);
        }
    }
    bool consistent(){
        if(epoint && nexist) return false;
        for(int i=0 ; i<10 ; i++){
            if(nxt[i] && !nxt[i]->consistent()) return false;
        }
        return true;
    }
};

int t, n;
char s[12];

int main()
{
    for(scanf("%d",&t) ; t>0 ; t--){
        scanf("%d",&n);
        Trie *root=new Trie;
        for(int i=0 ; i<n ; i++){
            scanf("%s",s);
            root->update(s);
        }
        printf("%s\n",root->consistent() ? "YES" : "NO");
        delete root;
    }
    return 0;
}
```

10 Tree Decomposition Method

10.1 Heavy Light Decomposition

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

using namespace std;
typedef pair<int,int> P;

struct Edge{
    int v, cost;
    Edge() {}
    Edge(int a, int b) : v(a), cost(b) {}
};

struct SegmentTree{
    vector<int> tree;
    int base;
    SegmentTree() {}
    void setup(int a){
        base=1;
        while(base<a) base<<=1;
        tree.resize(base*2+2);
        base--;
    }
    void update(int idx, int val){
        idx+=base;
        tree[idx]=val;
        idx=idx>>1;
        while(idx!=0){
            tree[idx]=max(tree[idx*2],tree[idx*2+1]);
            idx=idx>>1;
        }
    }
    int get_value(int num, int st, int fn, int ns=1, int nf=-1){
        if(nf==-1) nf=base+1;
        if(ns>fn || nf<st) return 0;
        if(st<=ns && nf<=fn) return tree[num];
        int mid=(ns+nf)>>1;
        return max(get_value(num*2,st,fn,ns,mid),get_value(num*2+1,st,fn,mid+1,nf));
    }
};

struct HeavyLightDecomposition{
    vector<vector<Edge> > lst;
    vector<int> par, sub, depth;
    vector<int> tail, dfsn, chain;
    vector<SegmentTree> HLDSegTree;
    int dn;
    HeavyLightDecomposition(int a){
        lst.resize(a+2);

        sub.resize(a+2);

        par.resize(a+2);
        depth.resize(a+2);
        dn=0;
        tail.resize(a+2);
        dfsn.resize(a+2);
        chain.resize(a+2);

        HLDSegTree.resize(a+2);
    }
    void add_edge(int v1, int v2, int cost){
        lst[v1].emplace_back(v2,cost);
        lst[v2].emplace_back(v1,cost);
    }
    void get_subsize(int cur, int prev);
    void HLD(int cur, int prev, int c_num);
    void construct_segtree(int a);
    void update_segtree(int v1, int v2, int cost);
    int get_max(int x, int y);
};
//get_subsize(1,0)
void HeavyLightDecomposition::get_subsize(int cur, int prev)
{
    sub[cur]=1;
    for(int i=0 ; i<lst[cur].size() ; i++){
        int nxt=lst[cur][i].v;
        if(nxt!=prev){
            HeavyLightDecomposition::get_subsize(nxt,cur);
            sub[cur]+=sub[nxt];
        }
    }
}
//chain number = initial vertex of dfsn
//tail = terminal vertex of dfsn
//dfsn is continuous in chain
//HLD(1,0,1)
void HeavyLightDecomposition::HLD(int cur, int prev, int c_num)
{
    dfsn[cur]=++dn;
    chain[dn]=c_num;
    tail[c_num]=dn;
    depth[dfsn[cur]]=depth[dfsn[prev]]+1;
    par[dfsn[cur]]=dfsn[prev];
    int idx=-1;
    for(int i=0 ; i<lst[cur].size() ; i++){
        int nxt=lst[cur][i].v;
        if(nxt!=prev && (idx==-1 || sub[nxt]>sub[idx])) idx=nxt;
    }
    if(idx!=-1) HeavyLightDecomposition::HLD(idx,cur,c_num);
    for(int i=0 ; i<lst[cur].size() ; i++){
        int nxt=lst[cur][i].v;
        if(nxt!=prev && nxt!=idx)
            HeavyLightDecomposition::HLD(nxt,cur,dn+1);
    }
}

void HeavyLightDecomposition::construct_segtree(int a)
```

```cpp
{
    for(int i=1 ; i<=a ; i++){
        if(chain[dfsn[i]]==dfsn[i]) HLDSegTree[dfsn[i]].setup(tail[dfsn[i]]-dfsn[i]+1);
    }
    for(int i=1 ; i<=a ; i++){
        for(int j=0 ; j<lst[i].size() ; j++){
            int v1=i;
            int v2=lst[i][j].v;
            int cost=lst[i][j].cost;
            v1=dfsn[v1], v2=dfsn[v2];
            if(par[v2]!=v1) continue;
            int seg_num=chain[v2];
            if(chain[v1]!=chain[v2]) HLDSegTree[seg_num].update(1,cost);
            else HLDSegTree[seg_num].update(v2-chain[v2]+1,cost);
        }
    }
}

void HeavyLightDecomposition::update_segtree(int v1, int v2, int cost)
{
    v1=dfsn[v1], v2=dfsn[v2];
    if(depth[v1]>depth[v2]) swap(v1,v2);
    int seg_num=chain[v2];
    if(chain[v1]==chain[v2]) HLDSegTree[seg_num].update(v2-chain[v2]+1,cost);
    else HLDSegTree[seg_num].update(1,cost);
}

int HeavyLightDecomposition::get_max(int x, int y)
{
    int ret=0;
    x=dfsn[x];
    y=dfsn[y];
    while(chain[x]!=chain[y]){
        int x1=chain[x];
        int y1=chain[y];
        if(depth[x1]>depth[y1]){
            ret=max(ret,HLDSegTree[chain[x]].get_value(1,1,x-chain[x]+1));
            x=par[x1];
        }
        else{
            ret=max(ret,HLDSegTree[chain[y]].get_value(1,1,y-chain[y]+1));
            y=par[y1];
        }
    }
    if(depth[x]>depth[y]) swap(x,y);
    //x=LCA
    if(x!=y) ret=max(ret,HLDSegTree[chain[x]].get_value(1,x-chain[x]+2,y-chain[x]+1));
    return ret;
}

int main()
{
    int n, m;
    scanf("%d",&n);
    HeavyLightDecomposition T(n);
    vector<P> elist(n);
    for(int i=1 ; i<n ; i++){
        int v1, v2, cost;
        scanf("%d %d %d",&v1,&v2,&cost);
        elist[i]=P(v1,v2);
        T.add_edge(v1,v2,cost);
    }
    T.get_subsize(1,0);
    T.HLD(1,0,1);
    T.construct_segtree(n);
    scanf("%d",&m);
    for(int i=0 ; i<m ; i++){
        int t, x, y;
        scanf("%d %d %d",&t,&x,&y);
        if(t==1) T.update_segtree(elist[x].first,elist[x].second,y);
        else printf("%d\n",T.get_max(x,y));
    }
    return 0;
}
```

10.2 Centroid Decomposition

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

using namespace std;
typedef vector<vector<int> > graph;

struct CentroidDecomposition{
    graph CentTree; // result of Centroid Tree
    vector<bool> visit;
    vector<int> sub; // size of subtree
    int tot;
    CentroidDecomposition(int N){
        CentTree.resize(N+5);
        visit.resize(N+5);
        sub.resize(N+5);
    }
    void get_sz(int cur, int prev, graph &tree){
        sub[cur]=1;
        for(int i=0 ; i<tree[cur].size() ; i++){
            int nxt=tree[cur][i];
            if(!visit[nxt] && nxt!=prev){
                get_sz(nxt,cur,tree);
                sub[cur]+=sub[nxt];
            }
        }
    }
    int get_cen(int cur, int prev, graph &tree){
        for(int i=0 ; i<tree[cur].size() ; i++){
            int nxt=tree[cur][i];
            if(!visit[nxt] && nxt!=prev && sub[nxt]>tot/2)
                return get_cen(nxt,cur,tree);
        }
        return cur;
    }
    int decomp(int cur, int prev, graph &tree){
        get_sz(cur,prev,tree);
        tot=sub[cur];
        int cen=get_cen(cur,prev,tree);
        visit[cen]=true;
        for(int i=0 ; i<tree[cen].size() ; i++){
            int nxt=tree[cen][i];
            int nxtcen;
            if(!visit[nxt] && nxt!=prev){
                nxtcen=decomp(nxt,cen,tree);
                CentTree[cen].push_back(nxtcen);
                CentTree[nxtcen].push_back(cen);
            }
        }
        return cen;
    }
};
```