struct **UnionFind**

{

vector<int> data;

void init(int n) { data.assign(n+1, -1); }

bool unionSet(int x, int y) {

x = root(x); y = root(y); if(x != y) { if(data[y] < data[x]) swap(x, y);

data[x] += data[y]; data[y] = x; }

return x != y;

}

bool findSet(int x, int y) { return root(x) == root(y); }

int root(int x) { return data[x] < 0 ? x : data[x] = root(data[x]); }

int size(int x) { return -data[root(x)]; }

};

struct **SegmentTree** {

vector<int> Tree;

void init(int Arr[],int N) { Tree.assign(6\*N,0); Build(Arr,0,N-1,0); }

void Build(int Arr[],int L,int R,int POS) {

if(L==R) { Tree[POS]=Arr[L]; return; }

int Mid=(L+R)/2;

Build(Arr,L,Mid,2\*POS+1);

Build(Arr,Mid+1,R,2\*POS+2);

Tree[POS]=max(Tree[POS\*2+1],Tree[POS\*2+2]);

}

// Query(0,N-1,L-1,R-1,0)

int Query(int L,int R,int QL,int QR,int POS) {

if(QL<=L && QR>=R) return Tree[POS];

if (QL>R || QR<L) return 0;

int Mid=(L+R)/2;

return max( Query( L, Mid , QL,QR , 2\*POS+1),

Query(Mid+1 ,R , QL,QR , 2\*POS+2) );

}

void Update(int L,int R,int QL,int QR,int POS,int VALUE) {

if(L>R || QL>R || QR<L) { return; }

if(L==R) { Tree[POS]+=VALUE; return; }

int Mid=(L+R)/2;

Update(L,Mid,QL,QR,2\*POS+1,VALUE); Update(Mid+1,R,QL,QR,2\*POS+2,VALUE);

Tree[POS]=max(Tree[POS\*2+1],Tree[POS\*2+2]);

}

};

Graph Algorithms :

struct Node{

int node, cost;

// i.e Node o is higher up in priority in the queue

bool operator<(Node const &o) const {

return cost > o.cost;

}

};

vector<Node> AdjList[MAXN];

int dist[MAXN];

void ShortestPath(int src){

Node start,curr,next;

start.node = src;

start.cost = 0;

memset(dist, -1, sizeof(dist)); // Initialize distance

priority\_queue<Node> q; // Initialize queue

q.push(start);

while(!q.empty()){

curr = q.top();

q.pop();

//cout << curr.node << " " << curr.cost << ", dist : " << dist[curr.node] << endl;

if(dist[curr.node] == -1){

dist[curr.node] = curr.cost;

for(int i = 0; i < AdjList[curr.node].size(); i++){

if(dist[AdjList[curr.node][i].node] != -1) continue;

next.node = AdjList[curr.node][i].node;

next.cost = curr.cost + AdjList[curr.node][i].cost;

q.push(next);

}

}

}

}

scanf("%d %d %d",&x,&y,&v);

AdjList[x].push\_back({y,v});

AdjList[y].push\_back({x,v});

ShortestPath(src);

const int N=2;

const int MOD=1e9+7;

void Copy(ll A[N][N],ll B[N][N]){ for(int i=0;i<N;i++) for(int j=0;j<N;j++) A[i][j]=B[i][j];}

void Mul(ll A[N][N],ll B[N][N]){ ll C[N][N];

for(int i=0;i<N;i++) { for(int j=0;j<N;j++){ C[i][j]=0; for(int k=0;k<N;k++) {

C[i][j]=(C[i][j]+(ll)A[i][k]\*(ll)B[k][j])%MOD; }}}

Copy(A,C); }

 void Power(ll A[N][N],int [Exp](http://www.opengroup.org/onlinepubs/009695399/functions/exp.html)){

ll Ans[2][2]={{1,0},{0,1}};

  while([Exp](http://www.opengroup.org/onlinepubs/009695399/functions/exp.html)) {

if(Exp&1) {

Mul(Ans,A);

}

Mul(A,A);

[Exp](http://www.opengroup.org/onlinepubs/009695399/functions/exp.html)=Exp>>1;

}

Copy(A,Ans);

}

ll Solve(int n)

{

if(n==0) return 1;

if(n==1) return 1;

ll A[2][2]={{1,1},{1,0}};

Power(A,n);

return (A[0][1]+MOD)%MOD;

}