**//Matrix Exponential**

//The code for Max Expo

//ll matRow,matCol; // the per matrix row and column number

/\*YOU MUST PREPARE THE BASE MATRIX FIRST\*/

typedef long long int ll;

struct matrix

{

ll arr[5][5];

}; // matExpo object

ll row,col; // the o(n^3) where row = n;

ll MOD; // the number through which we will mod if given

matrix baseMatrix; // the main matrix which will be returned

matrix special; // the special value matrix for any problem

matrix multiplication(matrix A, matrix B) // this function will multiply two matrix and return 1

{

matrix C;

memset(C.arr,0,sizeof(C.arr)); // initialization

//looping

for(int i=1; i<=row; i++) // for the answer row

{

for(int j=1; j<=col; j++) //for the answer col

{

for(int k=1; k<=row; k++) // for the general looping

{

C.arr[i][j] = (C.arr[i][j]%MOD + ((A.arr[i][k]%MOD) \* (B.arr[k][j]%MOD))%MOD)%MOD; /\* This needs to

be modified if there is any kind or different operation rather than just

multiplication\*/

}

}

}

return C; // return of the answer matrix

}

matrix bigMod(matrix A, ll power, ll mod) // MOD = defined mod

{

// cout<<" power " << power << endl;

if(power == 1)

{

return A;

} // mod = parameter mod

if(power%2 == 0)

{

matrix C = bigMod(A,power/2,mod);

return multiplication(C,C);

}

else if(power%2 == 1)

{

matrix C = bigMod(A,power-1,mod);

return multiplication(C,A);

}

}

void matExpoInitialization(ll r,ll c, ll mod) // all the variables of matExpo will be initialized here

{

row = r; // row initialized

col = c; // col initialized

//base matrix initialization in main

//special matrix update in main

int v=1;

for(ll i=1; i<=mod; i++)

{

v \*= 10;

}

MOD = v;

}

//Gaussian Elimination

//This equation solves linear Equations

//after making them matrix

//Ax = B, x = (A^-1)B

//we have a structure named matrix

//MX\_GAUSS is the dimension of structure matrix member

//boolean valid solution it says either solution is possible or not

//it will solve for n by n linearly independent matrix

struct matrix

{

int row,col;

double arr[MX\_GAUSS+2][MX\_GAUSS+2];

};

bool validSolution;

double GaussianElimination(struct matrix A, struct matrix B) // we are sending Ax = B, A and B

{

// n is the row number now

int n = A.row;

for(int i=1;i<=n;i++)

{

//it says either solution is possible or not

validSolution = false;

//Adjust valid row

for(int j=i;j<=n;j++) // including this row with all other rows

{

if(fabs(A.arr[j][i])>EPS) // if it's not zero

{

swap(A.arr[j],A.arr[i]); //then swap may be the same row

swap(B.arr[j],B.arr[i]);

validSolution = true;

break;

}

}

//sanity check

if(!validSolution)

{

//There is no solution

break;

}

//now to make the down rows same column values zero

for(int j = i+1;j<=n;j++)

{

double multipliyer = A.arr[j][i] / A.arr[i][i];

A.arr[j][i] = 0;

for(int k = i+1; k<=n; k++)

{

A.arr[j][k] = A.arr[j][k] - multipliyer \* A.arr[i][k];

}

B.arr[j][1] = B.arr[j][1] - multipliyer\*B.arr[i][1];

}

//Now to make the up rows same column values zero

for(int j = i-1; j>=1; j--) // Going upward

{

double multipliyer = A.arr[j][i] / A.arr[i][i];

A.arr[j][i] = 0;

for(int k= i+1; k <= n; k++)

{

A.arr[j][k] = A.arr[j][k] - multipliyer \* A.arr[i][k];

}

B.arr[j][1] = B.arr[j][1] - multipliyer\*B.arr[i][1];

}

}

/\*just for simple print the matrix sanity check

just comment out it if you want to use it

\*/

/\*for(int i=1;i<=n;i++)

{

cout<<A.arr[i][i]<<" " << B.arr[i][1]<<endl;

//printf("%lf \n",B.arr[i][1]);

}\*/

/\*EXTRA CALCULATION

VARIES WITH PROBLEMS\*/

ans = B.arr[1][1] / A.arr[1][1];

}

Example : Spoj Going to School

#include <bits/stdc++.h>

using namespace std;

#define MX\_GAUSS 17

#define EPS 1e-9

vector<int>graph[MX\_GAUSS+1],weight[MX\_GAUSS+1];

//This equation solves linear Equations

//after making them matrix

//Ax = B, x = (A^-1)B

//we have a structure named matrix

//MX\_GAUSS is the dimension of structure matrix member

//boolean valid solution it says either solution is possible or not

//it will solve for n by n linearly independent matrix

struct matrix

{

int row,col;

double arr[MX\_GAUSS+2][MX\_GAUSS+2];

};

bool validSolution;

double GaussianElimination(struct matrix A, struct matrix B,int st) // we are sending Ax = B, A and B

{

// n is the row number now

int n = A.row;

for(int i=1;i<=n;i++)

{

//it says either solution is possible or not

validSolution = false;

//Adjust valid row

for(int j=i;j<=n;j++) // including this row with all other rows

{

//cout<<"j " << j << " "<<A.arr[j][i]<<endl;

if(fabs(A.arr[j][i])>EPS) // if it's not zero

{

swap(A.arr[j],A.arr[i]); //then swap may be the same row

swap(B.arr[j],B.arr[i]);

validSolution = true;

break;

}

}

//sanity check

if(!validSolution)

{

//There is no solution

//cout<<"damn"<<endl;

break;

}

//now to make the down rows same column values zero

for(int j = i+1;j<=n;j++)

{

double multipliyer = A.arr[j][i] / A.arr[i][i];

A.arr[j][i] = 0;

for(int k = i+1; k<=n; k++)

{

A.arr[j][k] = A.arr[j][k] - multipliyer \* A.arr[i][k];

}

B.arr[j][1] = B.arr[j][1] - multipliyer\*B.arr[i][1];

}

//Now to make the up rows same column values zero

for(int j = i-1; j>=1; j--) // Going upward

{

double multipliyer = A.arr[j][i] / A.arr[i][i];

A.arr[j][i] = 0;

for(int k= i+1; k <= n; k++)

{

A.arr[j][k] = A.arr[j][k] - multipliyer \* A.arr[i][k];

}

B.arr[j][1] = B.arr[j][1] - multipliyer\*B.arr[i][1];

}

}

/\*just for simple print the matrix sanity check

just comment out it if you want to use it

\*/

/\*for(int i=1;i<=n;i++)

{

cout<<A.arr[i][i]<<" " << B.arr[i][1]<<endl;

//printf("%lf \n",B.arr[i][1]);

}

/\*EXTRA CALCULATION

VARIES WITH PROBLEMS\*/

double ans = B.arr[st][1] / A.arr[st][st];

return ans;

}

int main(void) {

int T,t;

scanf("%d",&T);

for(t=1;t<=T;t++) {

int n,st,en;

scanf("%d %d %d",&n,&st,&en);

for(int i=0;i<MX\_GAUSS;i++) {

graph[i].clear();

weight[i].clear();

}

for(int i=1;i<=n-1;i++) {

int a,b,c;

scanf("%d %d %d",&a,&b,&c);

graph[a].push\_back(b);

graph[b].push\_back(a);

weight[a].push\_back(c);

weight[b].push\_back(c);

}

struct matrix A,B;

A.row=n-1;

A.col=A.row-1;

for(int i=1;i<=n;i++){

if(i == en) continue;

if(i != en){

int base,target;

if(i>en) base=i-1;

else base=i;

memset(A.arr[base],0,sizeof(A.arr[base]));

double sum=0.0;

for(int j=0;j<graph[i].size();j++) {

sum += (double)weight[i][j];

}

for(int j=0;j<graph[i].size();j++) {

if(graph[i][j]>en) {

target=graph[i][j]-1;

}

else {

target=graph[i][j];

if(target==en) {

continue;

}

}

A.arr[base][target] = (1.0\*weight[i][j])/sum;

}

A.arr[base][base] = -1.0;

B.arr[base][1] = -1.0;

}

}

if(st==en){

double ans=0;

printf("%0.5lf\n",ans);

continue;

}

if(en<st) {

st--;

}

double ans = GaussianElimination(A,B,st);

printf("%0.5lf\n",ans);

}

return 0;

}