

2 diametre\_bfs  
3 diametre\_dfs  
4 SSSP  
5 spfa  
6 Floyd\_Warshall  
7 SCC  
8 mst-kruskal  
9 mst-kruskal  
10 mst-prim

//SPOJ

//diamètre

//Longest path in an undirected tree:

//Given an undirected tree, we need to find the longest path of this tree where a path is defined as a sequence of nodes.

#define Graph vector<vector<int > >

Graph g**;**

void addBiEdge**(**int v**,** int w**)**

**{**

g**[**v**].**pb**(**w**);** // Add w to v’s list.

g**[**w**].**pb**(**v**);** // Since the graph is undirected

**}**

// method returns farthest node and its distance from node u

//<node, distance>

pair**<**int**,** int**>** bfs**(**int u**)**

**{**

// mark all distance with -1

vector**<**int**>** dis**(** sz**(**g**)** **,** **-**1 **);**

queue**<**int**>** q**;**

q**.**push**(**u**);**

// distance of u from u will be 0

dis**[**u**]** **=** 0**;**

**while** **(!**q**.**empty**())**

**{**

int t **=** q**.**front**();** q**.**pop**();**

// loop for all adjacent nodes of node-t

**for** **(**auto v **:** g**[**t**])**

**{**

// push node into queue only if

// it is not visited already

**if** **(**dis**[**v**]** **==** **-**1**)**

**{**

q**.**push**(**v**);**

// make distance of v, one more

// than distance of t

dis**[**v**]** **=** dis**[**t**]** **+** 1**;**

**}**

**}**

**}**

// get farthest node distance and its index

auto it **=** max\_element**(**all**(**dis**));**

int maxDis **=** **\***it **;**

int idx **=** distance**(**dis**.**begin**(),** it**);**

**return** mp**(**idx**,** maxDis**);**

**}**

// method prints longest path of given tree

void LongestPathLength**()**

**{**

// first bfs to find one end point of longest path

auto t1 **=** bfs**(**0**);**

// second bfs to find actual longest path

auto t2 **=** bfs**(**t1**.**first**);**

cout **<<** "Longest path is from " **<<** t1**.**first **<<** " to " **<<** t2**.**first **<<** " of length " **<<** t2**.**second**;**

//cout<<t2.second;

**}**

// Driver code pour calibrage

//int main\_drive(){

int main**(){**

// Create a graph given in the example

g**.**resize**(**10**);**

addBiEdge**(**0**,** 1**);**

addBiEdge**(**1**,** 2**);**

addBiEdge**(**2**,** 3**);**

addBiEdge**(**2**,** 9**);**

addBiEdge**(**2**,** 4**);**

addBiEdge**(**4**,** 5**);**

addBiEdge**(**1**,** 6**);**

addBiEdge**(**6**,** 7**);**

addBiEdge**(**6**,** 8**);**

LongestPathLength**();**

//rep="Longest path is from 5 to 7 of length 5"

**return** 0**;**

**}**

//int main(){

//int n,a,b;

//cin>>n;

//rep(i,n-1){

//cin>>a>>b;

//addBiEdge(a,b);

//}

//

//

// longestPathLength();

//

//}

3 diametre\_dfs

// lpl using 1 dfs recursive time: O( n log n ) ; space O( log( n ) )

#define Graph vector<vector<int > >

Graph g**;**

void addBiEdge**(**int v**,** int w**)**

**{**

g**[**v**].**pb**(**w**);** // Add w to v’s list.

g**[**w**].**pb**(**v**);** // Since the graph is undirected

**}**

// method returns farthest node and its distance from $node

vector**<**int**>** h**;**//height: doit etre init à -1, ça sera fait dans le main plus tard

vector**<**int**>** d**;**//Diameter: les diamtres des sous arbres

//< height , Diameter >

pair**<**int**,**int**>** lpl**(**int node**)** **{**

**if** **(~**h**[**node**]** **||** **!**h**[**node**]** **){**// ni blanc, ni gris ;/ ne serait-il pas plus rapide de faire h[node]>0

**return** mp**(**h**[**node**],**d**[**node**]);**

**}**

**if** **(**sz**(**g**[**node**])** **==** 0**)** **return** mp**(**0**,**0**);**

// cerr<<node<<endl;

h**[**node**]=**0**;**//coloration gris (en cours)

int Diameter **=** 0**,** h1**=-**1**,** h2**=-**1**;**

**for** **(**auto v **:** g**[**node**])**

**{**

//cerr<<v<<" "<<!~h[node]<<endl;

// push node into queue only if

// it is not visited already

**if** **(!~**h**[**v**])**

**{**

auto p **=** lpl**(**v**);**

**if(**p**.**fi **>=** h1**){**h2 **=** h1**;**h1 **=** p**.**fi**;}**

**else** **if(**p**.**fi **>** h2**){**h2 **=** p**.**fi**;}**

Diameter **=** max**(**Diameter**,**p**.**se**);**

**}**

**}**

**return** mp**(**1**+** max**(**h1**,**h2**),**

max**(** Diameter**,** h1 **+** h2 **+** 2**)**

**);**

**}**

int main**(){**

//freopen("a.txt", "r", stdin);

ios**::**sync\_with\_stdio**(**0**);**

cin**.**tie**(**0**);** cout**.**tie**(**0**);**

int n**,**a**,**b**;**

cin**>>**n**;**

g**.**resize**(**n**);**

rep**(**i**,**n**-**1**){**

cin**>>**a**>>**b**;**

addBiEdge**(**a**-**1**,**b**-**1**);**

**}**

d**.**resize**(**n**);**

h**.**resize**(**n**,-**1**);**

auto p **=** lpl**(**0**);**

cout**<<**p**.**se**;**

**}**

4 SSSP

const double eps **=** 1e-9**;**

int Dyjkestra\_infinite **=** std**::**numeric\_limits**<**int**>::**max**();**

#define Dj\_val\_min 0

#define nax 10000

#define null -1

int to**[**nax**];**

int nxt**[**nax**];**

int from**[**nax**];**

//int p[nax][nax];//poid

int cost**[**nax**];**//poid

ull dist**[**nax**];**

int edge\_size **=** 0**;**

void AddEdge**(**int de**,** int vers**,**int poid**){**

to**[**edge\_size**]** **=** vers**;**

nxt**[**edge\_size**]** **=** from**[**de**];**

cost**[**edge\_size**]=**poid**;**

from**[**de**]** **=** edge\_size**;**

edge\_size**++;**

**}**

int d\_cmp**(**double a**,**double b**){**

**if(**fabs**(**a**-**b**)<**eps**)**

**return** 0**;**

**return** **((**a**>**b**)<<**1**)-**1**;**

**}**

void dijkstra\_init**(){**

memset**(**from**,** **-**1**,** nax**\*(sizeof** from**[**0**]));**

edge\_size**=**0**;**

**}**

double dijkstra**(**int src**,** int snk**){**

priority\_queue**<**pair**<**ull**,**int**>,**vector**<**pair**<**ull**,**int**>>,** greater**<**pair**<**ull**,**int**>>>** q**;**

//priority\_queue< pair<ull,int> > q;

q**.**push**({**Dj\_val\_min**,**src**});**

memset**(**dist**,** Dyjkestra\_infinite**,** nax**\*(sizeof** dist**[**0**]));**//initialisé à l'inf

**while(**q**.**size**()){**

double d**=**q**.**top**().**first**;**

int cur**=**q**.**top**().**second**;**

q**.**pop**();**

**if(**d**>**dist**[**cur**])**

**continue;**

**if(**snk**==**cur**)**

**return** d**;**

**for(**int j**=**from**[**cur**];~**j**;**j**=**nxt**[**j**]){**

int t**=**to**[**j**];**

ll dd**=**d**+**cost**[**j**];**//produit

**if(**dist**[**t**]>**dd**){**//inferieur

dist**[**t**]=**dd**;**

q**.**push**({**dd**,**t**});**

**}**

**}**

**}**

**return** **-**1**;**

**}**

void \_main**();**

int main**()** **{**

int ttt**;**

cin **>>** ttt**;**

**while** **(**ttt**--)** **{**

\_main**();**

**}**

**return** 0**;**

**}**

void \_main**(){**

//init

dijkstra\_init**();**

int v**,**k**,**a**,**b**,**c**;**

cin**>>**v**>>**k**;**

/\*

int noeud[v];//ptr to dernier element lié à

to[k];

nxt[k];

cost[k];

\*/

rep**(**i**,**k**){**

cin**>>**a**>>**b**>>**c**;**

AddEdge**(**a**-**1**,**b**-**1**,**c**);**

**}**

cin**>>**a**>>**b**;**

ll mn **=** dijkstra**(**a**-**1**,** b**-**1**);**

**if** **(**mn**==-**1**){**

cout **<<**"NO\n"**;**

**}else{**

cout **<<**mn**<<**"\n"**;**

**}**

**}**

5 SPFA

/\*\* // non testé

\* HDU1874 SSSP SPFA solution

\* Time: 0ms.... (?????)

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

#include <vector>

**using** **namespace** std**;**

const int INF **=** 1e9 **+** 7**;**

const int MAXN **=** 205**;**

const int MAXM **=** 1010**;**

struct Edge **{**

int v**,** w**;**

Edge**(**int tv**,** int tw**)**

**{**

v **=** tv**,** w **=** tw**;**

**}**

**};**

int n**,** m**;**

int dist**[**MAXN**];**

bool vis**[**MAXN**];**

vector**<**Edge**>** e**[**MAXN**];**

void init**()**

**{**

**for** **(**int i **=** 0**;** i **<** MAXN**;** i**++)**

**{**

dist**[**i**]** **=** INF**;**

e**[**i**].**clear**();**

**}**

memset**(**vis**,** 0**,** **sizeof** vis**);**

**}**

int main**()**

**{**

**while** **(**scanf**(**"%d%d"**,** **&**n**,** **&**m**)** **!=** EOF**)**

**{**

init**();**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)**

**{**

int u**,** v**,** w**;**

scanf**(**"%d%d%d"**,** **&**u**,** **&**v**,** **&**w**);**

e**[**u**].**push\_back**(**Edge**(**v**,** w**));**

e**[**v**].**push\_back**(**Edge**(**u**,** w**));**

**}**

int start**,** end**;**

scanf**(**"%d%d"**,** **&**start**,** **&**end**);**

// SPFA

queue**<**int**>** Q**;**

Q**.**push**(**start**);**

vis**[**start**]** **=** 1**;**

dist**[**start**]** **=** 0**;** // so important

**while** **(!**Q**.**empty**())**

**{**

int cur **=** Q**.**front**();**

Q**.**pop**();**

vis**[**cur**]** **=** 0**;**

**for** **(**unsigned int i **=** 0**;** i **<** e**[**cur**].**size**();** i**++)**

**{**

int v **=** e**[**cur**][**i**].**v**,** w **=** e**[**cur**][**i**].**w**;**

**if** **(**dist**[**v**]** **>** dist**[**cur**]** **+** w**)** **{**

dist**[**v**]** **=** dist**[**cur**]** **+** w**;**

Q**.**push**(**v**);**

vis**[**v**]** **=** 1**;**

**}**

**}** // for i = 0 -> e[cur].size

**}**

**if** **(**dist**[**end**]** **==** INF**)** **{**

printf**(**"-1\n"**);**

**}** **else** **{**

printf**(**"%d\n"**,** dist**[**end**]);**

**}**

**}**

**return** 0**;**

**}**

6 Floyd\_Warshall

// C++ Program to check if there is a negative weight

// cycle using Floyd Warshall Algorithm

#include<iostream>

**using** **namespace** std**;**

// Number of vertices in the graph

#define V 4

/\* Define Infinite as a large enough value. This

value will be used for vertices not connected

to each other \*/

#define INF 99999

// A function to print the solution matrix

void printSolution**(**int dist**[][**V**]);**

// Returns true if graph has negative weight cycle

// else false.

bool negCyclefloydWarshall**(**int graph**[][**V**])**

**{**

/\* dist[][] will be the output matrix that will

finally have the shortest

distances between every pair of vertices \*/

int dist**[**V**][**V**],** i**,** j**,** k**;**

/\* Initialize the solution matrix same as input

graph matrix. Or we can say the initial values

of shortest distances are based on shortest

paths considering no intermediate vertex. \*/

**for** **(**i **=** 0**;** i **<** V**;** i**++)**

**for** **(**j **=** 0**;** j **<** V**;** j**++)**

dist**[**i**][**j**]** **=** graph**[**i**][**j**];**

/\* Add all vertices one by one to the set of

intermediate vertices.

---> Before start of a iteration, we have shortest

distances between all pairs of vertices such

that the shortest distances consider only the

vertices in set {0, 1, 2, .. k-1} as intermediate

vertices.

----> After the end of a iteration, vertex no. k is

added to the set of intermediate vertices and

the set becomes {0, 1, 2, .. k} \*/

**for** **(**k **=** 0**;** k **<** V**;** k**++)**

**{**

// Pick all vertices as source one by one

**for** **(**i **=** 0**;** i **<** V**;** i**++)**

**{**

// Pick all vertices as destination for the

// above picked source

**for** **(**j **=** 0**;** j **<** V**;** j**++)**

**{**

// If vertex k is on the shortest path from

// i to j, then update the value of dist[i][j]

**if** **(**dist**[**i**][**k**]** **+** dist**[**k**][**j**]** **<** dist**[**i**][**j**])**

dist**[**i**][**j**]** **=** dist**[**i**][**k**]** **+** dist**[**k**][**j**];**

**}**

**}**

**}**

// If distance of any verex from itself

// becomes negative, then there is a negative

// weight cycle.

**for** **(**int i **=** 0**;** i **<** V**;** i**++)**

**if** **(**dist**[**i**][**i**]** **<** 0**)**

**return** **true;**

**return** **false;**

**}**

// driver program

int main**()**

**{**

/\* Let us create the following weighted graph

1

(0)----------->(1)

/|\ |

| |

-1 | | -1

| \|/

(3)<-----------(2)

-1 \*/

int graph**[**V**][**V**]** **=** **{** **{**0 **,** 1 **,** INF **,** INF**},**

**{**INF **,** 0 **,** **-**1 **,** INF**},**

**{**INF **,** INF **,** 0 **,** **-**1**},**

**{-**1 **,** INF **,** INF **,** 0**}};**

**if** **(**negCyclefloydWarshall**(**graph**))**

cout **<<** "Yes"**;**

**else**

cout **<<** "No"**;**

**return** 0**;**

**}**

7 SCC

#include<memory.h>

struct edge **{**

int e**,** nxt**;**

**};**

int V**,** E**;**

edge e**[**MAXE**],** er**[**MAXE**];**

int sp**[**MAXV**],** spr**[**MAXV**];**

int group\_cnt**,** group\_num**[**MAXV**];**

bool v**[**MAXV**];**

int stk**[**MAXV**];**

void fill\_forward**(**int x**)**

**{**

int i**;**

v**[**x**]=true;**

**for(**i**=**sp**[**x**];** i**;** i**=**e**[**i**].**nxt**)**

**if(!**v**[**e**[**i**].**e**])**

fill\_forward**(**e**[**i**].**e**);**

stk**[++**stk**[**0**]]=**x**;**

**}**

void fill\_backward**(**int x**)**

**{**

int i**;**

v**[**x**]=false;**

group\_num**[**x**]=**group\_cnt**;**

**for(**i**=**spr**[**x**];** i**;** i**=**er**[**i**].**nxt**)**

**if(**v**[**er**[**i**].**e**])**

fill\_backward**(**er**[**i**].**e**);**

**}**

void add\_edge**(**int v1**,** int v2**)** //add edge v1->v2

**{**

e **[++**E**].**e**=**v2**;**

e **[**E**].**nxt**=**sp **[**v1**];**

sp **[**v1**]=**E**;**

// la transposé du graphe pour le retour

er**[** E**].**e**=**v1**;**

er**[**E**].**nxt**=**spr**[**v2**];**

spr**[**v2**]=**E**;**

**}**

void SCC**()**

**{**

int i**;**

stk**[**0**]=**0**;**

memset**(**v**,** **false,** **sizeof(**v**));**

**for(**i**=**1**;** i**<=**V**;** i**++)**

**if(!**v**[**i**])**

fill\_forward**(**i**);**

group\_cnt**=**0**;**

**for(**i**=**stk**[**0**];** i**>=**1**;** i**--)**

**if(**v**[**stk**[**i**]])** **{**

group\_cnt**++;**

fill\_backward**(**stk**[**i**]);**

**}**

**}**

#include <bits/stdc++.h>

**using** **namespace** std**;**

struct edge **{**

int u**,** v**,** w**;**

**};**

const int maxn **=** 10000**;**

int vill**;**

int ans **=** 0**;**

int p**[**maxn**];**

edge e**[**maxn**];**

void init**()**

**{**

ans **=** 0**;**

**for** **(**int i **=** 0**;** i **<** maxn**;** i**++)**

**{**

e**[**i**].**u **=** 0**;**

e**[**i**].**v **=** 0**;**

e**[**i**].**w **=** 1e9**;**

p**[**i**]** **=** i**;**

**}**

**}**

bool cmp**(**const edge a**,** const edge b**)**

**{**

**return** a**.**w **<** b**.**w**;**

**}**

int find**(**int x**)**

**{**

**return** p**[**x**]** **==** x **?** p**[**x**]** **:** p**[**x**]** **=** find**(**p**[**x**]);**

**}**

int main**()**

**{**

**while** **(**scanf**(**"%d"**,** **&**vill**)** **!=** EOF**)**

**{**

**if** **(**vill **==** 0**)** **{**

**break;**

**}**

init**();**

int edgeCount **=** vill **\*** **(**vill **-** 1**)** **/** 2**;**

**for** **(**int i **=** 0**;** i **<** edgeCount**;** i**++)**

**{**

scanf**(**"%d%d%d"**,** **&**e**[**i**].**u**,** **&**e**[**i**].**v**,** **&**e**[**i**].**w**);**

**}**

sort**(**e**,** e**+**edgeCount**,** cmp**);**

**for** **(**int n **=** 0**;** n **<** edgeCount**;** n**++)**

**{**

int px **=** find**(**e**[**n**].**u**),**

py **=** find**(**e**[**n**].**v**);**

**if** **(**px **!=** py**)** **{**

p**[**px**]** **=** py**;**

ans **+=** e**[**n**].**w**;**

**}**

**}**

printf**(**"%d\n"**,** ans**);**

**}** // while

**return** 0**;**

**}**

9 mst-kruskal

/\*\*

\* HDU1233 最小生成树模板题 Kruskal 实现

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

**using** **namespace** std**;**

const int MAXN **=** 105**;**

struct Edge **{**

int u**,** v**,** w**;**

**};**

int n**,** m**;**

int ans **=** 0**,** times **=** 0**;**

int p**[**MAXN**];**

Edge edges**[**MAXN **\*** MAXN**];**

bool cmp**(**const Edge a**,** const Edge b**)**

**{**

**return** a**.**w **<** b**.**w**;**

**}**

void init**()**

**{**

ans **=** 0**;** // 初始化答案

times **=** 0**;** // 初始化合并次数

memset**(**edges**,** 0**,** **sizeof** edges**);**

// 初始化并查集

**for** **(**int i **=** 0**;** i **<** MAXN**;** i**++)**

**{**

p**[**i**]** **=** i**;**

**}**

**}**

/\* 并查集：查找 \*/

int find**(**int x**)**

**{**

**return** x **==** p**[**x**]** **?** x **:** p**[**x**]** **=** find**(**p**[**x**]);**

**}**

int main**()**

**{**

// n 为点数量， m 为边数量

**while** **(**scanf**(**"%d%d"**,** **&**m**,** **&**n**)** **!=** EOF **&&** m **!=** 0**)**

**{**

init**();**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)**

**{**

scanf**(**"%d%d%d"**,** **&**edges**[**i**].**u**,** **&**edges**[**i**].**v**,** **&**edges**[**i**].**w**);**

**}**

// 给边按照权值从小到大排序

sort**(**edges**,** edges **+** m**,** cmp**);**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)**

**{**

int x **=** edges**[**i**].**u**,** y **=** edges**[**i**].**v**;**

// 并查集的合并操作

// 查询当前边起点和终点所在集合

int px **=** find**(**x**),**

py **=** find**(**y**);**

**if** **(**px **==** py**)** **{**

**continue;** // 如果 px 和 py 在一个集合中，则不做任何操作

**}** **else** **{**

p**[**px**]** **=** py**;** // 合并

ans **+=** edges**[**i**].**w**;** // 更新答案

times**++;** // 合并次数+1

**}**

**}**

// 没有找到最小生成树

**if** **(**times **!=** n**-**1**)** **{**

printf**(**"?\n"**);**

**}** **else** **{**

printf**(**"%d\n"**,** ans**);**

**}**

**}**

**return** 0**;**

**}**

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

**using** **namespace** std**;**

const int INF **=** 1e9 **+** 7**;**

const int MAXN **=** 105**;**

struct Edge **{**

int v**,** w**;**

Edge**(**int tv**,** int tw**)**

**{**

v **=** tv**,** w **=** tw**;**

**}**

**};**

int n**,** m**;**

int ans **=** 0**;**

int dist**[**MAXN**];**

bool vis**[**MAXN**];**

vector**<**Edge**>** e**[**MAXN**];**

void init**()**

**{**

ans **=** 0**;** // 初始化答案

memset**(**vis**,** 0**,** **sizeof** vis**);**

**for** **(**int i **=** 0**;** i **<** MAXN**;** i**++)**

**{**

dist**[**i**]** **=** INF**;**

e**[**i**].**clear**();**

**}**

**}**

int main**()**

**{**

// n 为点数量， m 为边数量

**while** **(**scanf**(**"%d%d"**,** **&**m**,** **&**n**)** **!=** EOF **&&** m **!=** 0**)**

**{**

init**();**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)**

**{**

int u**,** v**,** w**;**

scanf**(**"%d%d%d"**,** **&**u**,** **&**v**,** **&**w**);**

e**[**u**].**push\_back**(**Edge**(**v**,** w**));**

e**[**v**].**push\_back**(**Edge**(**u**,** w**));**

**}**

/\* Prim \*/

bool flag **=** **true;** // 如果 flag = false

dist**[**1**]** **=** 0**,** vis**[**1**]** **=** 1**;** // 因为题目的标号从 1

// 遍历起点的所有边，更新 dist[v] = w

**for** **(**unsigned int i **=** 0**;** i **<** e**[**1**].**size**();** i**++)**

**{**

int v **=** e**[**1**][**i**].**v**,** w **=** e**[**1**][**i**].**w**;**

dist**[**v**]** **=** w**;**

**}**

/\*

，vis[x] = 1 一定成立，那么 flag 会被重置为 false

\*/

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

**{**

int cur**,** w **=** INF**;** // cur

// 和 dijkstra 有些异曲同工之妙

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

**{**

**if** **(!**vis**[**j**]** **&&** dist**[**j**]** **<** w**)** **{**

cur **=** j**,** w **=** dist**[**j**];**

**}**

**}**

**if** **(**w **==** INF**)** **{**

flag **=** **false;**

**break;**

**}**

ans **+=** w**;**

vis**[**cur**]** **=** 1**;** // 标记下一个点已访问

// 对于选定的点，遍历它的所有出边，进行松弛操作

**for** **(**unsigned i **=** 0**;** i **<** e**[**cur**].**size**();** i**++)**

**{**

int v **=** e**[**cur**][**i**].**v**,** w **=** e**[**cur**][**i**].**w**;**

**if** **(!**vis**[**v**]** **&&** dist**[**v**]** **>** w**)** **{**

dist**[**v**]** **=** w**;**

**}**

**}**

**}** /\* end prim \*/

**if** **(!**flag**)** **{**

printf**(**"?\n"**);**

**}** **else** **{**

printf**(**"%d\n"**,** ans**);**

**}**

**}**

**return** 0**;**

**}**