**Assignment 1**

**1.Dr Strange v/s COVID-19**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Scanner;

public class probB {

private class Vertex {

HashMap<String, Integer> nbrs = new HashMap<>();

HashMap<String,Integer> patrolAt=new HashMap<>();

}

HashMap<String, Vertex> vtces = new HashMap<>();

public void addVertex(String vname) {

Vertex vtx = new Vertex();

vtces.put(vname, vtx);

}

public void addEdge(String vname1, String vname2, int cost) {

Vertex vtx1 = vtces.get(vname1);

Vertex vtx2 = vtces.get(vname2);

if (vtx1 == null || vtx2 == null || vtx1.nbrs.containsKey(vname2))

return;

vtx1.nbrs.put(vname2, cost);

vtx2.nbrs.put(vname1, cost);

}

public void display() {

System.out.println("---------------------------");

ArrayList<String> keys = new ArrayList<>(vtces.keySet());

for (String key : keys) {

Vertex vtx = vtces.get(key);

System.out.println(key + " : " + vtx.patrolAt);

}

}

public void addTime(String vname,int k,int[] tij) {

if(k==0){

return;

}

else{

Vertex time = vtces.get(vname);

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

int timestamp=tij[i];

time.patrolAt.put(Integer.toString(i + 1), timestamp);

}

}

}

public class DijkstraPair implements Comparable<DijkstraPair> {

String vname;

String psf;

int cost;

public int compareTo(DijkstraPair o) {

return o.cost - this.cost;

}

}

public HashMap<String, Integer> dijkstra(String src,int n) {

HashMap<String, Integer> ans = new HashMap<>();

HashMap<String, DijkstraPair> map = new HashMap<>();

int flag=0;

HeapGeneric<DijkstraPair> heap = new HeapGeneric<>();

for (String key : vtces.keySet()) {

DijkstraPair np = new DijkstraPair();

np.vname = key;

np.psf = "";

np.cost = Integer.MAX\_VALUE;

if (key.equals(src)) {

np.cost = 0;

np.psf = key;

}

if (key.equals(src)) {

//Vertex vxt=vtces.get(src);

if (vtces.get(src).patrolAt.containsKey("1") && vtces.get(src).patrolAt.get("1") == 0){

ArrayList<Integer> delays = new ArrayList<>(vtces.get(src).patrolAt.values());

for (int time : delays) {

if (time == np.cost) {

np.cost = np.cost + 1;

}

else{

break;

}

}

}

}

heap.add(np);

map.put(key, np);

}

while (!heap.isEmpty()) {

DijkstraPair rp = heap.remove();

map.remove(rp.vname);

//System.out.println("NAme of removed pair"+rp.vname+ " cost of removed pair "+rp.cost);

ans.put(rp.vname, rp.cost);

for (String nbr : vtces.get(rp.vname).nbrs.keySet()) {

if (map.containsKey(nbr)) {

int oc = map.get(nbr).cost;

int nc = rp.cost + vtces.get(rp.vname).nbrs.get(nbr);

if(nbr.equals(Integer.toString(n))){

flag=1;

}

//System.out.println(nbr+" "+flag);

if(vtces.get(nbr).patrolAt.containsValue(nc) && flag!=1){

ArrayList<Integer> delays=new ArrayList<>(vtces.get(nbr).patrolAt.values());

for(int time : delays){

if(time==nc){

nc=nc+1;

}

}

}

if (nc < oc) {

DijkstraPair gp = map.get(nbr);

gp.psf = rp.psf + nbr;

gp.cost = nc;

heap.updatePriority(gp);

}

}

}

}

//System.out.println(ans);

flag=0;

return ans;

}

public class HeapGeneric<T extends Comparable<T>> {

ArrayList<T> data = new ArrayList<>();

HashMap<T, Integer> map = new HashMap<>();

public void add(T item) {

data.add(item);

map.put(item, this.data.size() - 1);

upheapify(data.size() - 1);

}

private void upheapify(int ci) {

int pi = (ci - 1) / 2;

if (isLarger(data.get(ci), data.get(pi)) > 0) {

swap(pi, ci);

upheapify(pi);

}

}

private void swap(int i, int j) {

T ith = data.get(i);

T jth = data.get(j);

data.set(i, jth);

data.set(j, ith);

map.put(ith, j);

map.put(jth, i);

}

public void display() {

System.out.println(data);

}

public int size() {

return this.data.size();

}

public boolean isEmpty() {

return this.size() == 0;

}

public T remove() {

swap(0, this.data.size() - 1);

T rv = this.data.remove(this.data.size() - 1);

downheapify(0);

map.remove(rv);

return rv;

}

private void downheapify(int pi) {

int lci = 2 \* pi + 1;

int rci = 2 \* pi + 2;

int mini = pi;

if (lci < this.data.size() && isLarger(data.get(lci), data.get(mini)) > 0) {

mini = lci;

}

if (rci < this.data.size() && isLarger(data.get(rci), data.get(mini)) > 0) {

mini = rci;

}

if (mini != pi) {

swap(mini, pi);

downheapify(mini);

}

}

public T get() {

return this.data.get(0);

}

public int isLarger(T t, T o) {

return t.compareTo(o);

}

public void updatePriority(T pair) {

int index = map.get(pair);

upheapify(index);

}

}

public static void main(String[] args) {

probB graph=new probB();

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int m=sc.nextInt();

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

graph.addVertex(Integer.toString(i+1));

}

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

int ai=sc.nextInt();

int bi=sc.nextInt();

int ci=sc.nextInt();

graph.addEdge(Integer.toString(ai),Integer.toString(bi),ci);

}

int[] k=new int[n];

for(int j=0;j<n;j++){

k[j]=sc.nextInt();

int[] tij=new int[k[j]];

for(int b=0;b<k[j];b++){

tij[b]=sc.nextInt();

}

graph.addTime(Integer.toString(j+1),k[j],tij);

}

if(graph.dijkstra("1",n).get(Integer.toString(n))==2147483647)

System.out.println(-1);

else

System.out.println(graph.dijkstra("1",n).get(Integer.toString(n)));

}

}