Q1. Use an array to store the vertex and their corresponding edge, visit the array according to the edge starting from the first vertex until arrive the target or visited to the end. If arrive target then it is success. If cannot arrive target then it is fail.

Q2

Use a matrix. Build all possible road which exclude the banned roads. Access the city from the one with most adjacent city to the one with the least. Check if the city and all the cities it not adjacent to have common adjacent city. If all have common adjacent city then all cities can be accessed with 2 edges. If no then check next city according to the order.

Q3

Prepare a ‘total cost and previous node’ table. Store each node’s edge first. Start access from first node. Read the node’s edges and update the table one by one if the node’s cost plus the edge’s cost is smaller than the one stored in the table. Then store the unvisited nodes ,which are the node’s edge’s target, into a next-visit list with ascending order. After finish reading all edges in the node, apply the same process on the first node in the next-visit list. After the repeating process, if the final node haven’t been update then it fail. If the final node has been updated,

use a stack to store the ‘previous’ in the ‘total cost and previous node’ table and output it in first to last order.

Q4

The vertex must be direct to or be directed from the final vertex. Record the first directed from. If the first vertex is directed from, start from final vertex, first vertex, and continuous. If directed from is not at first, start from first vertex to the previous vertex of first directed from, which must be direct to, then go to final vertex, then go back to first directed from and continuous. If there is no directed from, all vertex are direct to, so can start from first vertex and continuously go till the last.

Q1

#include <stdio.h>

int main(){

int n,t;

scanf("%d",&n);

scanf("%d",&t);

int array[n];

for (int i=0;i<n-1;i++){

int input;

scanf("%d",&input);

array[i]=(input+i);

}

int choosing=0;

while ((choosing!=(t-1))&&(choosing!=(n-1))){

choosing=array[choosing];

// printf("%d\n",array[choosing]);

}

if (choosing==(t-1)) printf("YES"); else printf("NO");

return 0;

}

Q2

#include <stdio.h>

struct cellT{

int city;

int totalban;

};

int checkrelative(int a,int b,int\*\* array,int n){

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

if ((array[a][i]==0)&&(array[b][i]==0)){

return i;

}

}

return -1;

}

void checktotalroad(int a,int\*\* array,int n){

printf("%d\n",n-1);

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

{ if ((array[a][i]==0)&&(a!=i)) printf("%d %d\n",a+1,i+1);

if (array[a][i]!=0) printf("%d %d\n",i+1,checkrelative(a,i,array,n)+1);

}

}

int main(){

int n,m;

scanf("%d",&n);

scanf("%d",&m);

int \*\*array=(int\*\*)malloc(n \* sizeof(int\*));

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

array[i] = (int\*)malloc(n \* sizeof(int));

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

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

array[i][j]=0;

}

}

struct cellT access[n];

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

access[i].city=i;

access[i].totalban=0;

}

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

int a,b;

scanf("%d",&a);

scanf("%d",&b);

a--;b--;

array[a][b]=1;

array[b][a]=1;

access[a].totalban++;

access[b].totalban++;

}

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

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

struct cellT temp;

if (access[j].totalban<access[i].totalban)

{

temp=access[j];

access[j]=access[i];

access[i]=temp;

}

}

}

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

printf("%d %d\n",access[i].city,access[i].totalban);

}\*/

//start finding station

int count=0;

int working;

int fail=1;

while ((count<n)&&(fail==1)){

fail=0;

working=access[count].city;

//printf("doing too good with %d\n",working);

//run for each city

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

if (array[working][j]==1)

{

printf("city %d not accessable by %d\n",j,working);

if (checkrelative(working,j,array,n)==-1) fail=1;} //check all inaccessible, if no relative then this city fail

}

//printf("fail here is %d",fail);

count++;

}

if (fail==0)

checktotalroad(working,array,n);

return 0;}

Q3

#include <stdio.h>

typedef struct cellT{

int target;

int cost;

} cellT;

typedef struct listT{

cellT cell;

struct listT\* next;

} listT;

typedef struct resultT{

int cost;

int previous;

} resultT;

void addcell(listT\* list[],int target,int cost,int listnum){

/////////////////////////////////

if (list[listnum]==NULL){

list[listnum]=(listT\*)malloc(sizeof(listT));

list[listnum]->cell.target=target;

list[listnum]->cell.cost=cost;

list[listnum]->next=NULL;

//if (list!=NULL) printf("done!\n");

return 0;

}

listT\* newlist=list[listnum];//go through list

while (newlist->next!=NULL)

{

//printf("going \n");

newlist=newlist->next;

}

newlist->next=(listT\*)malloc(sizeof(listT));

cellT newcell;

newcell.target=target;

newcell.cost=cost;

newlist->next->cell=newcell;

newlist->next->next=NULL;

};

cellT readcurrentlist(listT\* list[],int listnum)

{

//if (list==NULL) exit(0);//excluded this condition in code

cellT newcell=list[listnum]->cell;

list[listnum]=list[listnum]->next;

return newcell;

}

cellT readcurrentnonarraylist(listT\*\* list)

{

//if (list==NULL) exit(0);//excluded this condition in code

cellT newcell=(\*list)->cell;

\*list=(\*list)->next;

return newcell;

}

int insert(int target,int cost,listT\*\* list)

{/////////////////////////////////////problem:in node0, both 2 insert think the 'nextlist' ptr is NULL

if (\*list==NULL){

// printf("insert in new list\n");

(\*list)=(listT\*)malloc(sizeof(listT));

(\*list)->cell.target=target;

(\*list)->cell.cost=cost;

(\*list)->next=NULL;

return 0;

}

//printf("insert in not new list\n");

listT\* newlist=(\*list);//go through list

while ((newlist->next!=NULL)&&(newlist->next->cell.cost<=cost))

{newlist=newlist->next;

//printf("stuck here!");

}

//no greater, put at last

if (newlist->next==NULL)

{

newlist->next=(listT\*)malloc(sizeof(listT));

cellT newcell;

newcell.target=target;

newcell.cost=cost;

newlist->next->cell=newcell;

newlist->next->next=NULL;

return 0;

}

//face greater value, put in front

listT\* newlist2;

newlist2=(listT\*)malloc(sizeof(listT));

newlist2->cell.cost=cost;

newlist2->cell.target=target;

newlist2->next=newlist->next;

newlist->next=newlist2;

}

void recursion(listT\* node[],resultT result[],int currentnode,int marked[],listT\* list){

cellT currentcell;

while(node[currentnode]!=NULL){

// read whole list////////////////

//printf("read adj from node!\n");

currentcell=readcurrentlist(node,currentnode);//take first cellT from nodelist, 1.repeat update result array 2. save to 'nextlist' for next access

if ((result[currentnode].cost+currentcell.cost)<result[currentcell.target].cost)

{

result[currentcell.target].cost=(result[currentnode].cost+currentcell.cost);

result[currentcell.target].previous=currentnode;

}

if (marked[currentcell.target]==0){

//printf("inserting node's unmarked adj %d into list\n",currentcell.target);

insert(currentcell.target,currentcell.cost,&list);//insert into 'nextlist' with order if not visited

marked[currentcell.target]=1;

//debugging

listT\* newlist;

newlist=list;

while (newlist!=NULL)

{

// printf("\n debug list contain %d %d \n",newlist->cell.target,newlist->cell.cost);

newlist=newlist->next;

}

}

}

marked[currentnode]=1;

//printf("done with node %d!\n\n",currentnode);

if (list!=NULL)

{

//printf("list is not empty! use new node!\n");

currentcell=readcurrentnonarraylist(&list);//read and delete from 'nextlist'//havent make when 'nextlist' is empty

//printf("original node is %d\n",currentnode);

currentnode=currentcell.target;

//printf("new node is %d\n",currentnode);

recursion(node,result,currentnode,marked,list);//go other node according to 'nextlist'

}

}

int main(){

int n,m;

scanf("%d",&n);

scanf("%d",&m);

listT\* node[n];//a2

int marked[n];//a1

listT\* list;//b

resultT result[n];//c

//initalize

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

marked[i]=0;

result[i].cost=10000001;

result[i].previous=-1;

}

result[0].cost=0;

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

node[i]=NULL;

}

list=NULL;

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

int a,b,c;

scanf("%d",&a);

scanf("%d",&b);

scanf("%d",&c);

a--;b--;

addcell(node,a,c,b);//pass by reference!

addcell(node,b,c,a);

// if (node[a]!=NULL) printf("%d good!\n",a);

//if (node[b]!=NULL) printf("%d good!\n",b);

}

recursion(node,result,0,marked,list);

//printf("after recur\n");

if ((result[n-1].cost==10000001)||(result[n-1].previous==-1)) printf("-1");

else {

int\* stack=(int\*)malloc(sizeof(int));

int amount=0;

int output=(n-1);

while (output!=-1){

stack=(int\*)realloc(stack,sizeof(int)\*(amount+10));

stack[amount]=output;

output=result[output].previous;

amount++;

}

//printf("ok here");

for (int i=amount-1;i>=0;i--){printf("%d ",stack[i]+1);}

}

return 0;

}

Q4

#include <stdio.h>

int main(){

int t;

scanf("%d",&t);

for (int i=0;i<t;i++)

{

int n,firstin=-1;

scanf("%d",&n);

int array[n];

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

{

scanf("%d",&array[j]);

if ((array[j]==1)&&(firstin==-1)) firstin=j;

}

if (array[n-1]==0) for (int k=0;k<n+1;k++) printf("%d ",k+1);

if (array[n-1]==1) {if (firstin==0) {printf("%d ",n+1);for (int k=0;k<n;k++) printf("%d ",k+1);}

if (firstin!=0) {for (int k=0;k<firstin;k++) printf("%d ",k+1);printf("%d ",n+1);for (int k=firstin;k<n;k++) printf("%d ",k+1);}

}

printf("\n");

}

return 0;}