Hệ sinh thái

#include<stdio.h>

#define MAX\_ELEMENTS 100

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

typedef int ElementType;

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

int empty\_list(List l){

return count\_list(&l)==0;

}

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

return l;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++)

printf("%d ",element\_at(&l,i));

}

List competition(List l0,List l1){

int i,j,x;

List l2;

make\_null(&l2);

for(i=1;i<=l0.size;i++){

for(j=1;j<=l1.size;j++){

if(element\_at(&l1,j)==element\_at(&l0,i)){

x=element\_at(&l0,i);

push\_back(&l2,x);

}

}

}

return l2;

}

void swap(int\* x,int\* y){

int temp;

temp=\*x;

\*x=\*y;

\*y=temp;

}

void bubbleSort(List\* l){

int i,j;

for(i=0;i<l->size-1;i++){

for(j=l->size-1;j>=i+1;j--){

if(l->data[j]<l->data[j-1])

swap(&(l->data[j]),&(l->data[j-1]));

}

}

}

int main(){

Graph G;

int n, m, e, x, y;

// freopen("HeSinhThaiRung.txt", "r", stdin); //Khi n?p bài nh? b? dòng này

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

//Ð?c d? th?

for(e=1;e<=m;e++){

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

add\_edge(&G,e,x,y);

}

//Ð?c 2 loài x và y

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

//

List l0;

make\_null(&l0);

l0=neighbors(&G,x);

List l1;

make\_null(&l1);

l1=neighbors(&G,y);

List l2;

make\_null(&l2);

l2= competition(l0,l1);

if(!empty\_list(l2)){

bubbleSort(&l2);

print\_List(l2);

}

else

printf("KHONG CHUNG DOI THU ");

return 0;

}

Shake hand

void solveShakeHand(Graph g){

int i,j;

int count;

for( i=1;i<=g.n;i++){

count=0;

for(j=1;j<=g.n;j++)

{

count+=g.A[i][j];

}

printf("%d\n",count);

}

}

Send Email

void solve\_SendMail(Graph g){

int x,max=0,sum;

int i,j;

for(i=1;i<=g.n;i++){

sum=0;

for(j=1;j<=g.n;j++){

sum+=g.A[j][i];

}

if(max<sum){

max=sum;

x=i;

}

}

printf("%d has received %d email(s).",x,max);

}

Duyet rộng

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct{

int data[MAX\_ELEMENTS];

int front, rear;

}Queue;

//

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void sortList(List\* l)

{

int i, j;

for (i = 0; i < l->size-1; i++)

// Last i elements are already in place

for (j = 0; j < l->size-i-1; j++)

if (l->data[j] > l->data[j+1])

swap(&l->data[j], &l->data[j+1]);

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void make\_null\_queue(Queue\* q){

q->front=0;

q->rear=-1;

}

void push(Queue\* q,int x){

q->rear++;

q->data[q->rear]=x;

}

int top(Queue\* q){

return q->data[q->front];

}

void pop(Queue\* q){

q->front++;

}

int empty(Queue\* q){

return q->front > q->rear;

}

//khai bao do thi dinh cung

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

sortList(&l);

return l;

}

// depth first search

void depth\_first\_search(Graph\* g){

Queue frontier;

int mark[MAX\_VERTEXES],k;

make\_null\_queue(&frontier);

//

int j;

for(j=1;j<=g->n;j++){

mark[j] = 0;

}

for(k=1;k<=g->n;k++){

if(mark[k]==0){

push(&frontier,k);

mark[k]=1;

while(!empty(&frontier)){

int x=top(&frontier);pop(&frontier);

printf("%d\n",x);

List list = neighbors(g,x);

for(j=1;j<=list.size;j++){

int y=element\_at(&list,j);

if(mark[y]==0){

mark[y]=1;

push(&frontier,y);

}

}

}

make\_null\_queue(&frontier);

}

}

}

int main(){

//freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,e,u, v);

}

//printMatrix(G);

depth\_first\_search(&G);

return 0;

}

Duyệt sâu hàng đợi

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct{

int data[MAX\_ELEMENTS];

int front, rear;

}Queue;

//

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void sortList(List\* l)

{

int i, j;

for (i = 0; i < l->size-1; i++)

// Last i elements are already in place

for (j = 0; j < l->size-i-1; j++)

if (l->data[j] > l->data[j+1])

swap(&l->data[j], &l->data[j+1]);

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void make\_null\_queue(Queue\* q){

q->front=0;

q->rear=-1;

}

void push(Queue\* q,int x){

q->rear++;

q->data[q->rear]=x;

}

int top(Queue\* q){

return q->data[q->front];

}

void pop(Queue\* q){

q->front++;

}

int empty(Queue\* q){

return q->front > q->rear;

}

//khai bao do thi dinh cung

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

sortList(&l);

return l;

}

// depth first search

int mark[MAX\_VERTEXES];

void traversal(Graph g,int x){

if(mark[x]==1){

return;

}

printf("%d \n",x);

mark[x]=1;

List list=neighbors(&g,x);

int j;

for(j=1;j<=list.size;j++){

traversal(g,element\_at(&list,j));

}

}

void depth\_first\_search(Graph g){

int i;

for(i=1;i<g.n;i++){

mark[i]=0;

}

for(i=1;i<g.n;i++){

traversal(g,i);

}

}

int main(){

//freopen("depth\_first\_search.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,e,u, v);

}

//printMatrix(G);

depth\_first\_search(G);

return 0;

}

Duyệt sâu hàng đợi

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int size;

}Stack;

//

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void sortList(List\* l)

{

int i, j;

for (i = 0; i < l->size-1; i++)

// Last i elements are already in place

for (j = 0; j < l->size-i-1; j++)

if (l->data[j] > l->data[j+1])

swap(&l->data[j], &l->data[j+1]);

}

/\* Tra ve so phan tu cua danh sach \*/

void make\_null\_stack(Stack\* s){

s->size=0;

}

void push(Stack\* s,int x){

s->data[s->size]=x;

s->size++;

}

int top(Stack\* s){

return s->data[s->size-1];

}

void pop(Stack\* s){

s->size--;

}

int empty(Stack\* s){

return s->size==0;

}

//khai bao do thi dinh cung

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

sortList(&l);

return l;

}

/\* Duyet do thi theo chieu sau \*/

void depth\_first\_search(Graph\* G) {

Stack L;

int mark[MAX\_VERTEXES];

make\_null\_stack(&L);

int k;

int j;

for (j = 1; j <= G->n; j++)

mark[j] = 0;

/\* Ðua 1 vào L, b?t d?u duy?t t? d?nh 1 \*/

for(k=1;k<=G->n;k++){

if(mark[k]==0)

push(&L,k);

/\* Vòng l?p chính dùng d? duy?t \*/

while (!empty(&L)) {

/\* L?y ph?n t? d?u tiên trong L ra \*/

int x = top(&L); pop(&L);

if (mark[x] != 0) // Ðã duy?t r?i, b? qua

continue;

printf("%d\n", x);

mark[x] = 1; //Ðánh d?u nó dã duy?t

/\* L?y các d?nh k? c?a nó \*/

List list = neighbors(G, x);

/\* Xét các d?nh k? c?a nó \*/

for (j = 1; j <= list.size; j++) {

int y = element\_at(&list, j);

push(&L, y);

}

}

}

}

int main(){

//freopen("depth\_first\_search.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,e,u, v);

}

//printMatrix(G);

depth\_first\_search(&G);

return 0;

}

Duyệt đệ quy

int mark[MAX\_VERTEXES];

int parent[MAX\_VERTEXES];

void traversal(Graph g, int x){

if(mark[x]==1){

return ;

}

mark[x]=1;

//printf("duyet %d\n",x);

List l= neighbors(&g,x);

int i;

for(i=1;i<=l.size;i++){

int y= element\_at(&l,i);

if(mark[y]==0){

parent[y]=x;

traversal(g,y);

}

}

}

/\* Duyet do thi theo chieu sau \*/

void depth\_first\_search(Graph\* G) {

int i;

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

mark[i]=0;

parent[i]=0;

}

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

traversal(\*G,i);

}

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

printf("%d %d\n",i,parent[i]);

}

}

Đồ thị liên thông(qua đảo)

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int size;

}Stack;

//

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void sortList(List\* l)

{

int i, j;

for (i = 0; i < l->size-1; i++)

// Last i elements are already in place

for (j = 0; j < l->size-i-1; j++)

if (l->data[j] > l->data[j+1])

swap(&l->data[j], &l->data[j+1]);

}

/\* Tra ve so phan tu cua danh sach \*/

//khai bao do thi dinh cung

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

sortList(&l);

return l;

}

int mark[MAX\_VERTEXES];

int parent[MAX\_VERTEXES];

int count=0;

void traversal(Graph g, int x){

if(mark[x]==1){

return ;

}

mark[x]=1;

count++;

// printf("duyet %d\n",x);

List l= neighbors(&g,x);

int i;

for(i=1;i<=l.size;i++){

int y= element\_at(&l,i);

if(mark[y]==0){

parent[y]=x;

traversal(g,y);

}

}

}

/\* Duyet do thi theo chieu sau \*/

void depth\_first\_search(Graph\* G) {

int i;

int valid=1;

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

mark[i]=0;

parent[i]=0;

}

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

traversal(\*G,i);

if(count!=G->n){

valid=0;

break;

}

count=0;

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

mark[i]=0;

parent[i]=0;

}

}

if(valid){

printf("YES");

}

else{

printf("NO");

}

}

int main(){

//freopen("visit\_island.txt", "r", stdin); //Khi n?p bài nh? b? ḍng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,e,u, v);

}

//printMatrix(G);

depth\_first\_search(&G);

return 0;

}

Kiểm tra chu trình đồ thị vô hướng

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

#define white 0

#define black 1

#define gray 2

typedef int ElementType;

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++){

printf("%d ",element\_at(&l,i));

}

}

typedef struct {

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=0;

}

}

}

void add\_edge(Graph\* g,int x,int y){

g->A[x][y]=1;

}

int adjacent(Graph\* graph,int x,int y){

return graph->A[x][y];

}

int degree\_out(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

int degree\_in(Graph graph, int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(adjacent(&graph,e,x)){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

List neighbors(Graph g,int x){

int e;

List l;

make\_null(&l);

for(e=1;e<=g.n;e++){

if(adjacent(&g,x,e)){

push\_back(&l,e);

}

}

return l;

}

int color[MAX\_VERTICES];

int cycle;

/\* Duyet do thi theo chieu sau \*/

void depth\_first\_search(Graph\* G,int x, int parent) {

int i;

color[x]=gray;

List list=neighbors(\*G,x);

for(i=1;i<=list.size;i++){

int y= element\_at(&list,i);

if(color[y]==gray){

cycle=1;

return;

}

if(color[y]==white){

depth\_first\_search(G,y,x);

}

}

color[x]=black;

}

int contains\_cycle(Graph\* g){

int i;

cycle=0;

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

color[i]=white;

}

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

if(color[i]==white)

depth\_first\_search(g,i,0);

}

return cycle;

}

int main(){

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

Graph G;

int n, u, v, m, e;

scanf("%d%d", &n, &m);

init\_graph(&G,n);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,u, v);

}

if(contains\_cycle(&G)){

printf("NO");

}

else {

printf("YES");

}

return 0;

}

Phân chia đội bóng

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int size;

}Stack;

//

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void sortList(List\* l)

{

int i, j;

for (i = 0; i < l->size-1; i++)

// Last i elements are already in place

for (j = 0; j < l->size-i-1; j++)

if (l->data[j] > l->data[j+1])

swap(&l->data[j], &l->data[j+1]);

}

/\* Tra ve so phan tu cua danh sach \*/

//khai bao do thi dinh cung

typedef struct

{

int n,m;

int A[MAX\_VERTICES][MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = m;

int i,j;

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

for( j = 1; j <= m ;j++){

graph->A[i][j]=0;

}

}

}

void add\_edge(Graph\* graph,int e,int x,int y){

graph->A[x][e]=1;

graph->A[y][e]=1;

}

int adjacent(Graph\* graph,int x,int y){

int e;

for(e=1;e<=graph->m;e++){

if((graph->A[x][e]==1)&&(graph->A[y][e]==1)){

return 1;

}

}

return 0;

}

int degree(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.m;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

void printMatrix(Graph graph){

int i,j;

for(i=1;i<=graph.n;i++){

for(j=1;j<=graph.m;j++){

printf("%d ",graph.A[i][j]);

}

printf("\n");

}

}

List neighbors(Graph\* G, int x){

List l;

make\_null(&l);

int i,j;

for(i=1;i<=G->m;i++){

if(G->A[x][i]==1){

for(j=1;j<=G->n;j++){

if((j!=x)&&(G->A[j][i]==1)){

push\_back(&l,j);

}

}

}

}

// sortList(&l);

return l;

}

int color[MAX\_VERTEXES];

int fail;

void colorize(Graph graph,int x,int c){

if(color[x]==-1){

color[x]=c;

List list= neighbors(&graph,x);

int y,i;

for(i=1;i<=list.size;i++){

y=element\_at(&list,i);

colorize(graph,y,!c);

}

}

else{

if(color[x]!=c)

fail=1;

}

}

int is\_bigraph(Graph graph){

int i;

for(i=1;i<=graph.n;i++){

color[i]=-1;

}

fail=0;

colorize(graph,1,0);

return !fail;

}

void divide\_footballteam(Graph graph){

if(!is\_bigraph(graph)){

printf("IMPOSSIBLE");

}

else{

int i;

for(i=1;i<=graph.n;i++){

if(color[i]==0)

printf("%d ",i);

}

printf("\n");

for(i=1;i<=graph.n;i++){

if(color[i]==1)

printf("%d ",i);

}

}

}

int main(){

//freopen("depth\_first\_search.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.

Graph G;

int n, m, u, v, e;

scanf("%d%d", &n, &m);

init\_graph(&G, n,m);

for (e = 1; e <= m; e++) {

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

add\_edge(&G,e,u, v);

}

//printMatrix(G);

divide\_footballteam(G);

return 0;

}

Kiểm tra số bộ phận liên thông mạnh(BPLTM)

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

#define white 0

#define black 1

#define gray 2

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int size;

}Stack;

void make\_null\_stack(Stack\* s){

s->size=0;

}

void push(Stack\* s,int x){

s->data[s->size]=x;

s->size++;

}

int top(Stack\* s){

return s->data[s->size-1];

}

void pop(Stack\* s){

s->size--;

}

int empty(Stack\* s){

return s->size==0;

}

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++){

printf("%d ",element\_at(&l,i));

}

}

typedef struct {

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=0;

}

}

}

void add\_edge(Graph\* g,int x,int y){

g->A[x][y]=1;

}

int adjacent(Graph\* graph,int x,int y){

return graph->A[x][y];

}

int degree\_out(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

int degree\_in(Graph graph, int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(adjacent(&graph,e,x)){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

List neighbors(Graph g,int x){

int e;

List l;

make\_null(&l);

for(e=1;e<=g.n;e++){

if(adjacent(&g,x,e)){

push\_back(&l,e);

}

}

return l;

}

int num[MAX\_VERTICES];

int min\_num[MAX\_VERTICES];

int onstack[MAX\_VERTICES];

int k ;

Stack s;

int count;

/\* Duyet do thi theo chieu sau \*/

int min(int a,int b){

return a>b?b:a;

}

void strong\_connect(Graph g,int x){

num[x]= min\_num[x]=k;

k++;

push(&s,x);

onstack[x] = 1;

List list =neighbors(g,x);

int j;

for(j=1;j<=list.size;j++){

int y=element\_at(&list,j);

if(num[y]<0){

strong\_connect(g,y);

min\_num[x]=min(min\_num[x],min\_num[y]);

}

else if(onstack[y]){

min\_num[x]=min(min\_num[x],num[y]);

}

}

//printf("min\_num[%d] = %d \n",x,min\_num[x]);

if(num[x]==min\_num[x]){

//printf("%d la dinh khop.\n",x);

count++;

int w;

do{

w=top(&s);

pop(&s);

onstack[w]=0;

}while(w!=x);

}

}

int isStrongConnect(Graph g){

int i;

for(i=1;i<=g.n;i++){

num[i]=-1;

}

k=1;

count=0;

strong\_connect(g,1);

if(count==1)

return 1;

else

return 0;

}

int main(){

// freopen("strong\_connected.txt","r",stdin);

Graph G;

int n, u, v, m, e;

scanf("%d%d", &n, &m);

init\_graph(&G,n);

for(e=1;e<=m;e++){

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

add\_edge(&G,u,v);

}

//print\_graph(G);

isStrongConnect(G);

printf("%d",count);

return 0;

}

Come and go

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

#define white 0

#define black 1

#define gray 2

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int size;

}Stack;

void make\_null\_stack(Stack\* s){

s->size=0;

}

void push(Stack\* s,int x){

s->data[s->size]=x;

s->size++;

}

int top(Stack\* s){

return s->data[s->size-1];

}

void pop(Stack\* s){

s->size--;

}

int empty(Stack\* s){

return s->size==0;

}

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++){

printf("%d ",element\_at(&l,i));

}

}

typedef struct {

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=0;

}

}

}

void add\_edge(Graph\* g,int x,int y,int w){

if(w==1)

g->A[x][y]=1;

else

{

g->A[x][y]=1;

g->A[y][x]=1;

}

}

int adjacent(Graph\* graph,int x,int y){

return graph->A[x][y];

}

int degree\_out(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

int degree\_in(Graph graph, int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(adjacent(&graph,e,x)){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

List neighbors(Graph g,int x){

int e;

List l;

make\_null(&l);

for(e=1;e<=g.n;e++){

if(adjacent(&g,x,e)){

push\_back(&l,e);

}

}

return l;

}

int num[MAX\_VERTICES];

int min\_num[MAX\_VERTICES];

int onstack[MAX\_VERTICES];

int k ;

Stack s;

int count;

/\* Duyet do thi theo chieu sau \*/

int min(int a,int b){

return a>b?b:a;

}

void strong\_connect(Graph g,int x){

num[x]= min\_num[x]=k;

k++;

push(&s,x);

onstack[x] = 1;

List list =neighbors(g,x);

int j;

for(j=1;j<=list.size;j++){

int y=element\_at(&list,j);

if(num[y]<0){

strong\_connect(g,y);

min\_num[x]=min(min\_num[x],min\_num[y]);

}

else if(onstack[y]){

min\_num[x]=min(min\_num[x],num[y]);

}

}

//printf("min\_num[%d] = %d \n",x,min\_num[x]);

if(num[x]==min\_num[x]){

//printf("%d la dinh khop.\n",x);

count++;

int w;

do{

w=top(&s);

pop(&s);

onstack[w]=0;

}while(w!=x);

}

}

int isStrongConnect(Graph g){

int i;

for(i=1;i<=g.n;i++){

num[i]=-1;

}

k=1;

count=0;

strong\_connect(g,1);

int valid=1;

for(i=1;i<=g.n;i++){

if(num[i]<0){

valid=0;

break;

}

}

if(count==1&&valid)

return 1;

return 0;

}

int main(){

//freopen("strong\_connected.txt","r",stdin);

Graph G;

int n, u, v, m, e,w;

scanf("%d%d", &n, &m);

init\_graph(&G,n);

for(e=1;e<=m;e++){

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

add\_edge(&G,u,v,w);

}

//print\_graph(G);

if(isStrongConnect(G)){

printf("OKIE");

}

else

printf("NO");

return 0;

}

TÌM ĐƯỜNG ĐI NGẮN NHẤT

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

#define NO\_EDGE 0

#define INFINITY 100

typedef int ElementType;

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++){

printf("%d ",element\_at(&l,i));

}

}

typedef struct {

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=NO\_EDGE;

}

}

}

void add\_edge(Graph\* g,int x,int y,int W){

g->A[x][y]=W;

}

int adjacent(Graph\* graph,int x,int y){

return graph->A[x][y];

}

int degree\_out(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

int degree\_in(Graph graph, int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(adjacent(&graph,e,x)){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

List neighbors(Graph g,int x){

int e;

List l;

make\_null(&l);

for(e=1;e<=g.n;e++){

if(adjacent(&g,x,e)){

push\_back(&l,e);

}

}

return l;

}

int pi[MAX\_VERTICES];

int mark[MAX\_VERTICES];

int p[MAX\_VERTICES];

void Dijkstra(Graph\* graph,int s){

int i,j,it;

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

pi[i]=INFINITY;

mark[i]=0;

}

pi[s]=0;

p[s]=-1;

for(it=1;it<=graph->n;it++){

int min\_pi=INFINITY;

for(j=1;j<=graph->n;j++){

if(mark[j]==0&&(pi[j]<=min\_pi)){

min\_pi=pi[j];

i=j;

}

}

mark[i]=1;

for(j=1;j<=graph->n;j++){

if((graph->A[i][j]!=NO\_EDGE)&&mark[j]==0){

if(pi[i]+graph->A[i][j]<pi[j]){

pi[j]=pi[i]+graph->A[i][j];

p[j]=i;

}

}

}

}

}

void shortest\_road(Graph\* g,int u){

int path[MAX\_VERTICES];

int k=0;

int current =u;

while(current != -1){

path[k]=current;

k++;

current=p[current];

}

int i;

for(i=k-1;i>=0;i--){

printf("%d ",path[i]);

}

}

int main(){

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

Graph G;

int n, u, v, m, e,w;

scanf("%d%d", &n, &m);

init\_graph(&G,n);

for (e = 1; e <= m; e++) {

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

add\_edge(&G, u, v, w);

}

//print\_graph(G);

// for(e=1;e<=n;e++){

// printf("%d %d\n",degree\_in(G,e),degree\_out(G,e));

// }

Dijkstra(&G,1);

//shortest\_road(&G,6);

if(pi[n]!=INFINITY){

printf("%d",pi[n]);

}

else {

printf("-1");

}

return 0;

}

Bellman-Ford pi and p

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define INFINITY 999

typedef struct {

int x,y,w;

}Edge;

typedef struct

{

int n,m;

Edge edges[MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = 0;

}

void add\_edge(Graph\* graph,int x,int y,int w){

graph->edges[graph->m].x=x;

graph->edges[graph->m].y=y;

graph->edges[graph->m].w=w;

graph->m++;

}

void print\_EDGES\_Graph(Graph graph){

int i;

for(i=0;i<graph.m;i++){

printf("%d: (%d,%d)=%d\n",i,graph.edges[i].x,graph.edges[i].y,graph.edges[i].w);

}

}

int pi[MAX\_VERTICES];

int p[MAX\_VERTICES];

void BellmanFord(Graph\* g,int s){

int i,j,it;

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

pi[i]=INFINITY;

}

pi[s]=0;

p[s]=-1;

for(it=1;it<g->n;it++){

for(j=0;j<g->m;j++){

int u=g->edges[j].x;

int v=g->edges[j].y;

int w=g->edges[j].w;

if(pi[u]+w < pi[v]){

pi[v]=pi[u]+w;

p[v]=u;

}

}

}

// for(j=0;j<g->m;j++){

// int u=g->edges[j].x;

// int v=g->edges[j].y;

// int w=g->edges[j].w;

// if(pi[u]+w < pi[v]){

// printf("YES");

// return;

// }

// }

// printf("NO");

}

void floyd\_Warshall(Graph\* g){

int j;

BellmanFord(g,1);

for(j=1;j<=g->n;j++){

printf("pi[%d] = %d",j,pi[j]);

printf(", p[%d] = %d\n",j,p[j]);

}

}

int main(){

Graph g;

int n,m,u,v,e,w;

//freopen("Bellman\_Ford.txt","r",stdin);

scanf("%d%d",&n,&m);

init\_graph(&g,n,m);

for(e=1;e<=m;e++){

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

add\_edge(&g,u,v,w);

}

//print\_EDGES\_Graph(g);

floyd\_Warshall(&g);

return 0;

}

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define INFINITY 999

typedef struct {

int x,y,w;

}Edge;

typedef struct

{

int n,m;

Edge edges[MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = 0;

}

void add\_edge(Graph\* graph,int x,int y,int w){

graph->edges[graph->m].x=x;

graph->edges[graph->m].y=y;

graph->edges[graph->m].w=w;

graph->m++;

}

void print\_EDGES\_Graph(Graph graph){

int i;

for(i=0;i<graph.m;i++){

printf("%d: (%d,%d)=%d\n",i,graph.edges[i].x,graph.edges[i].y,graph.edges[i].w);

}

}

int pi[MAX\_VERTICES];

int p[MAX\_VERTICES];

void BellmanFord(Graph\* g,int s){

int i,j,it;

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

pi[i]=INFINITY;

}

pi[s]=0;

p[s]=-1;

for(it=1;it<g->n;it++){

for(j=0;j<g->m;j++){

int u=g->edges[j].x;

int v=g->edges[j].y;

int w=g->edges[j].w;

if(pi[u]+w < pi[v]){

pi[v]=pi[u]+w;

p[v]=u;

}

}

}

// for(j=0;j<g->m;j++){

// int u=g->edges[j].x;

// int v=g->edges[j].y;

// int w=g->edges[j].w;

// if(pi[u]+w < pi[v]){

// printf("YES");

// return;

// }

// }

// printf("NO");

}

void floyd\_Warshall(Graph\* g){

int j;

BellmanFord(g,1);

for(j=1;j<=g->n;j++){

printf("pi[%d] = %d",j,pi[j]);

printf(", p[%d] = %d\n",j,p[j]);

}

}

int main(){

Graph g;

int n,m,u,v,e,w;

//freopen("Bellman\_Ford.txt","r",stdin);

scanf("%d%d",&n,&m);

init\_graph(&g,n,m);

for(e=1;e<=m;e++){

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

add\_edge(&g,u,v,w);

}

//print\_EDGES\_Graph(g);

floyd\_Warshall(&g);

return 0;

}

Bellman\_Ford

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define INFINITY 999

typedef struct {

int x,y,w;

}Edge;

typedef struct

{

int n,m;

Edge edges[MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = 0;

}

void add\_edge(Graph\* graph,int x,int y,int w){

graph->edges[graph->m].x=x;

graph->edges[graph->m].y=y;

graph->edges[graph->m].w=w;

graph->m++;

}

void print\_EDGES\_Graph(Graph graph){

int i;

for(i=0;i<graph.m;i++){

printf("%d: (%d,%d)=%d\n",i,graph.edges[i].x,graph.edges[i].y,graph.edges[i].w);

}

}

int pi[MAX\_VERTICES];

int p[MAX\_VERTICES];

void BellmanFord(Graph\* g,int s){

int i,j,it;

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

pi[i]=INFINITY;

}

pi[s]=0;

p[s]=-1;

for(it=1;it<g->n;it++){

for(j=0;j<g->m;j++){

int u=g->edges[j].x;

int v=g->edges[j].y;

int w=g->edges[j].w;

if(pi[u]+w < pi[v]){

pi[v]=pi[u]+w;

p[v]=u;

}

}

}

for(j=0;j<g->m;j++){

int u=g->edges[j].x;

int v=g->edges[j].y;

int w=g->edges[j].w;

if(pi[u]+w < pi[v]){

printf("YES");

return;

}

}

printf("NO");

}

int main(){

Graph g;

int n,m,u,v,e,w,s;

//freopen("Bellman\_Ford.txt","r",stdin);

scanf("%d%d",&n,&m);

init\_graph(&g,n,m);

for(e=1;e<=m;e++){

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

add\_edge(&g,u,v,w);

}

scanf("%d",&s);

//print\_EDGES\_Graph(g);

BellmanFord(&g,s);

return 0;

}

Floyd-Warshall

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define INFINITY 999

typedef struct {

int x,y,w;

}Edge;

typedef struct

{

int n,m;

Edge edges[MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n,int m){

graph->n = n;

graph->m = 0;

}

void add\_edge(Graph\* graph,int x,int y,int w){

graph->edges[graph->m].x=x;

graph->edges[graph->m].y=y;

graph->edges[graph->m].w=w;

graph->m++;

}

void print\_EDGES\_Graph(Graph graph){

int i;

for(i=0;i<graph.m;i++){

printf("%d: (%d,%d)=%d\n",i,graph.edges[i].x,graph.edges[i].y,graph.edges[i].w);

}

}

int pi[MAX\_VERTICES];

int p[MAX\_VERTICES];

void BellmanFord(Graph\* g,int s){

int i,j,it;

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

pi[i]=INFINITY;

}

pi[s]=0;

p[s]=-1;

for(it=1;it<g->n;it++){

for(j=0;j<g->m;j++){

int u=g->edges[j].x;

int v=g->edges[j].y;

int w=g->edges[j].w;

if(pi[u]+w < pi[v]){

pi[v]=pi[u]+w;

p[v]=u;

}

}

}

}

// for(j=0;j<g->m;j++){

// int u=g->edges[j].x;

// int v=g->edges[j].y;

// int w=g->edges[j].w;

// if(pi[u]+w < pi[v]){

// printf("YES");

// return;

// }

// }

// printf("NO");

//}

void floyd\_Warshall(Graph\* g){

int i,j;

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

BellmanFord(g,i);

for(j=1;j<=g->n;j++){

if(pi[j]<100)

printf("%d -> %d: %d\n",i,j,pi[j]);

else {

printf("%d -> %d: oo\n",i,j);

}

}

}

}

int main(){

Graph g;

int n,m,u,v,e,w;

//freopen("Bellman\_Ford.txt","r",stdin);

scanf("%d%d",&n,&m);

init\_graph(&g,n,m);

for(e=1;e<=m;e++){

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

add\_edge(&g,u,v,w);

}

//print\_EDGES\_Graph(g);

floyd\_Warshall(&g);

return 0;

}

#define INFINITY 9999999

int pi[MAXN][MAXN];

int next[MAXN][MAXN];

void Floyd\_Warshall(Graph\* G) {

int u, v, k;

for (u = 1; u <= G->n; u++)

for (v = 1; v <= G->n; v++) {

pi[u][v] = INFINITY;

next[u][v] = -1;

}

for (u = 1; u <= G->n; u++)

p[u][u] = 0;

for (u = 1; u <= G->n; u++)

for (v = 1; v <= G->n; v++)

if (G->L[u][v] != NO\_EDGE) {

pi[u][v] = G->L[u][v];

next[u][v] = v;

}

for (k = 1; k <= G->n; k++)

for (u = 1; u <= G->n; u++)

for (v = 1; v <= G->n; v++)

if (pi[u][k] + p[k][v] < pi[u][v]) {

pi[u][k] = pi[u][k] + pi[k][v];

next[u][v] = next[u][k];

}

}

Advance short road

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

#define NO\_EDGE 0

#define INFINITY 100

typedef int ElementType;

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

/\* Tra ve so phan tu cua danh sach \*/

int count\_list(List\* L) {

return L->size;

}

void print\_List(List l){

int i;

for(i=1;i<=l.size;i++){

printf("%d ",element\_at(&l,i));

}

}

typedef struct {

int n,m;

int A[MAX\_VERTICES][MAX\_VERTICES];

}Graph;

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=NO\_EDGE;

}

}

}

void add\_edge(Graph\* g,int x,int y,int W){

g->A[x][y]=W;

g->A[y][x]=W;

}

int adjacent(Graph\* graph,int x,int y){

return graph->A[x][y];

}

int degree\_out(Graph graph,int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(graph.A[x][e]==1){

count++;

}

}

return count;

}

int degree\_in(Graph graph, int x){

int count=0;

int e;

for(e=1;e<=graph.n;e++){

if(adjacent(&graph,e,x)){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

List neighbors(Graph g,int x){

int e;

List l;

make\_null(&l);

for(e=1;e<=g.n;e++){

if(adjacent(&g,x,e)){

push\_back(&l,e);

}

}

return l;

}

int pi[MAX\_VERTICES];

int mark[MAX\_VERTICES];

int p[MAX\_VERTICES];

int count=0;

void Dijkstra(Graph\* graph,int s){

int i,j,it;

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

pi[i]=INFINITY;

mark[i]=0;

}

int min =INFINITY;

pi[s]=0;

p[s]=-1;

for(it=1;it<=graph->n;it++){

int min\_pi=INFINITY;

for(j=1;j<=graph->n;j++){

if(mark[j]==0&&(pi[j]<=min\_pi)){

min\_pi=pi[j];

i=j;

}

}

mark[i]=1;

for(j=1;j<=graph->n;j++){

if((graph->A[i][j]!=NO\_EDGE)&&mark[j]==0){

if(pi[i]+graph->A[i][j]<=pi[j]){

pi[j]=pi[i]+graph->A[i][j];

p[j]=i;

if((j==graph->n)){

if (pi[j]<min){

min=pi[j];

count++;

}

else

if(pi[j]==min){

count++;

}

}

}

}

}

}

}

void shortest\_road(Graph\* g,int u){

int path[MAX\_VERTICES];

int k=0;

int current =u;

while(current != -1){

path[k]=current;

k++;

current=p[current];

}

int i;

for(i=k-1;i>=0;i--){

printf("%d ",path[i]);

}

}

void shortest\_road\_advance(Graph\* g){

if(pi[g->n]!=INFINITY){

printf("%d %d",pi[g->n],count);

}

else

{

printf("-1 0");

}

}

int main(){

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

Graph G;

int n, u, v, m, e,w;

scanf("%d%d", &n, &m);

init\_graph(&G,n);

for (e = 1; e <= m; e++) {

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

add\_edge(&G, u, v, w);

}

//print\_graph(G);

// for(e=1;e<=n;e++){

// printf("%d %d\n",degree\_in(G,e),degree\_out(G,e));

// }

Dijkstra(&G,1);

//shortest\_road(&G,6);

shortest\_road\_advance(&G);

return 0;

}

Xếp hạng đồ thị

#include<stdio.h>

#define max\_vertices 100

#define MAX\_ELEMENTS 100

typedef int ElementType;

typedef struct {

int n,m;

int A[max\_vertices][max\_vertices];

}Graph;

typedef struct{

ElementType data[MAX\_ELEMENTS];

int size;

} List;

/\* Tao danh sach rong \*/

void make\_null(List\* L) {

L->size = 0;

}

/\* Them mot phan tu vao cuoi danh sach \*/

void push\_back(List\* L, ElementType x) {

L->data[L->size] = x;

L->size++;

}

/\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/

ElementType element\_at(List\* L, int i) {

return L->data[i-1];

}

int sizeOfList(List l){

return l.size;

}

void init\_graph(Graph\* g,int n){

g-> n=n;

int i,j;

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

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

g->A[i][j]=0;

}

}

}

void add\_edge(Graph\* g,int x,int y){

g->A[x][y]=1;

}

int degree\_come(Graph\* g,int vertice){

int i;

int count=0;

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

if(g->A[i][vertice]){

count++;

}

}

return count;

}

void print\_graph(Graph g){

int i,j;

for(i=1;i<=g.n;i++){

for(j=1;j<=g.n;j++)

printf("%d ",g.A[i][j]);

printf("\n");

}

}

void copy\_list(List\* s1,List\* s2){

make\_null(s1);

int i;

for(i=0;i<s2->size;i++){

push\_back(s1,s2->data[i]);

}

make\_null(s2);

}

int rank[max\_vertices];

void ranking(Graph\* g){

int d[max\_vertices];

int x,u,i;

for(u=1;u<=g->n;u++){

d[u]=0;

}

for(x=1;x<=g->n;x++){

for(u=1;u<=g->n;u++)

if(g->A[x][u]!=0)

d[u]++;

}

List s1,s2;

make\_null(&s1);

for(i=1;i<=g->n;i++)

if(d[i]==0)

push\_back(&s1,i);

int k=0;

while(s1.size>0){

make\_null(&s2);

for(u=1;u<=s1.size;u++){

int i = element\_at(&s1,u);

rank[i]=k;

int v;

for(v=1;v<=g->n;v++)

if(g->A[i][v]!=0){

d[v]--;

if(d[v]==0)

push\_back(&s2,v);

}

}

copy\_list(&s1,&s2);

k++;

}

}

int main(){

Graph g;

int n,i,m;

int x,y;

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

scanf("%d%d",&n,&m);

init\_graph(&g,n);

for(i=1;i<=m;i++){

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

add\_edge(&g,x,y);

}

//print\_graph(g);

ranking(&g);

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

printf("%d ",rank[i]);

}

return 0;

}

Chia kẹo thêm cung yx

Kruskal

#include<stdio.h>

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

#define INFINITY 999

typedef struct {

int x,y,w;

}Edge;

typedef struct

{

int n,m;

Edge edges[MAX\_EDGES];

}Graph;

void init\_graph(Graph\* graph,int n){

graph->n = n;

graph->m = 0;

}

void add\_edge(Graph\* graph,int x,int y,int w){

graph->edges[graph->m].x=x;

graph->edges[graph->m].y=y;

graph->edges[graph->m].w=w;

graph->m++;

}

void print\_EDGES\_Graph(Graph graph){

int i;

for(i=0;i<graph.m;i++){

printf("%d: (%d,%d)=%d\n",i,graph.edges[i].x,graph.edges[i].y,graph.edges[i].w);

}

}

int parent[MAX\_VERTICES];

int findRoot(int u){

if(parent[u] == u)

return u;

return findRoot(parent[u]);

}

void swap(Edge\* a,Edge\* b){

Edge temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void sortEdges(Graph\* g){

int i,j;

for(i=0; i<(g->m)-1; i++){

for(j=(g->m)-1; j>=i+1; j--){

if((g->edges[j].w) < (g->edges[j-1].w)){

swap(&g->edges[j],&g->edges[j-1]);

}

}

}

}

int Kruskal(Graph\* g,Graph\* t){

sortEdges(g);

init\_graph(t,g->n);

int i,sum = 0;

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

parent[i] = i;

}

for(i=0;i<=g->m-1;i++){

int u = g->edges[i].x;

int v = g->edges[i].y;

int w = g->edges[i].w;

int root\_u = findRoot(u);

int root\_v = findRoot(v);

if(root\_u != root\_v){

add\_edge(t,u,v,w);

parent[root\_v] = root\_u;

sum+=w;

}

}

return sum;

}

int main(){

Graph g;

int n,m,u,v,e,w,s;

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

scanf("%d%d",&n,&m);

init\_graph(&g,n);

for(e=1;e<=m;e++){

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

add\_edge(&g,u,v,w);

}

scanf("%d",&s);

// print\_EDGES\_Graph(g);

Graph t;

int sum = Kruskal(&g,&t);

printf("%d\n",sum);

// print\_EDGES\_Graph(t);

int i;

for(i=1;i<=t.m;i++){

if(t.edges[i-1].x<t.edges[i-1].y)

printf("%d %d %d\n",t.edges[i-1].x,t.edges[i-1].y,t.edges[i-1].w);

else{

printf("%d %d %d\n",t.edges[i-1].y,t.edges[i-1].x,t.edges[i-1].w);

}

}

return 0;

}