**Danh sách duyệt - ma trận kề**

**ĐƠN ĐỒ THỊ VÔ HƯỚNG**

**//khoi tao**

void init(Graph \*G, int n){

G->n=n;

G->m=0;

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

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

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

}

**//them cung**

void add(Graph \*G, int u, int v){

G->A[u][v]=1;

G->A[v][u]=1;

G->m++;

}

**DO THI VO HUONG ( DA CUNG VA KHUYEN)**

**//khoi tao**

void khoitao(Graph \*G, int n){

G->n=n;

G->m=0;

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

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

G->A[i][l]=0;

}

**//them cung**

void add\_cung(Graph \*G, int u, int v){

G->A[u][v]++;

if(u!=v)

G->A[v][u]++;

G->m++;

}

**//tinh bac**

int deg(Graph \*G, int u){

int dem=0;

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

dem+=G->A[u][i];

return dem + G->A[u][u];

}

**//dinh ke tang dan**

void printNeighbours(Graph\* G, int vertex) {

printf("neighbours(%d) =", vertex);

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

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

printf(" %d", i);

}

}

printf("\n");

}

**//DO THI CO HUONG (DA CUNG VA KHUYEN)**

**//khoi tao**

void khoi(Graph \*G, int n){

G->n=n;

G->m=0;

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

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

G->A[i][l]=0;

}

**//them**

void them(Graph \*G, int u, int v){

G->A[u][v]++;

G->m++;//bo dong nay van duoc

}

**//tinh bac**

int deg\_ch(Graph \*G, int u){

int bac=0;

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

bac += G->A[u][v] + G->A[v][u];

}

return bac;

}

**//tinh bac indeg**

int indegree(Graph \*pG, int u) {

//dòng này có vẫn được dùng để kt mà ch trình kt rùi nên khỏi

/\* if (u < 1 || u > pG->n) {

// Tr? v? -1 n?u d?nh không h?p l?

return -1;

}\*/

int indeg = 0;

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

indeg += pG->A[v][u];

}

return indeg;

}

**//tinh bac out**

int outdegree(Graph \*pG, int u) {

/\* if (u < 1 || u > pG->n) {

// Tr? v? -1 n?u d?nh không h?p l?

return -1;

}

\*/

int outdeg = 0;

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

outdeg += pG->A[u][v];

}

return outdeg;

}

**//dinh ke**

void print\_Neighbours(Graph\* G, int vertex) {

printf("neighbours(%d) =", vertex);

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

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

printf(" %d", i);

}

}

printf("\n");

}

**//DON DO THI CO HUONG**

void dttch(Graph \*G, int u, int v){

G->A[u][v]=1;

G->m++;

}

**int main(){**

Graph G;

int n,m,u,v;

// nhap dinh va cung

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

init(&G, n);

//doc m cung va them vao dt

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

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

them(&G, u, v);

}

//in ma tran

printf("Ma Tran Ke:\n");

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

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

printf("%d ", G.A[u][v]);

}

printf("\n");

}

//tin bac

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

printf("bac(%d): %d\n", k, deg\_ch(&G, k));

}

//dinh ke

// In các d?nh k? c?a t?ng d?nh

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

printNeighbours(&G, i);

}

return 0;

}

**Tính liên thông cua đồ thị**

Cho một đồ thị **vô hướng đơn**. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị **theo chiều sâu** (sử dụng NGĂN XẾP) bắt đầu từ đỉnh 1.

// Kiểm tra ngăn xếp có rỗng hay không

bool emptyStack(Stack \*s){

return s->size==0;

}

// Thêm 1 phần tử vào ngăn xếp

void pushStack(Stack \*s, int element){

s->dat#include <stdio.h>

#include <stdbool.h>

#define maxv 50

#define idx 100

int n,m;//Biến toàn cục lưu n đỉnh và m cung

// Khai báo cấu trúc đồ thị

typedef int Vertices;

typedef struct{

int matrix[maxv][maxv];

Vertices n;//Đỉnh n

}Graph;

// Khai báo cấu trúc danh sách - LIST

typedef struct{

int data[idx];

int size;

}List;

// Khai báo cấu trúc ngắn xếp - STACK

typedef struct{

int data[idx];

int size;

}Stack;

/\*Các hàm của ngăn xếp\*/

// Làm rỗng ngăn xếp

void makenullStack(Stack \*s){

s->size=0;

}a[s->size]=element;

s->size++;

}

//Truy cập phần tử trong ngăn xếp

int getStack(Stack \*s){

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

}

// Xóa vị trí phần tử trong ngăn xếp đi(Vị trị này thực tế vẫn còn nhưng chúng ta không truy cập đến nữa)

void delSize(Stack \*s){

s->size--;

}

/\*Các hàm trên danh sách\*/

// Làm rỗng danh sách

void makenullList(List \*l){

l->size=0;

}

// Kiểm tra danh sách có rỗng hay không

bool emptyList(List \*l){

return l->size==0;

}

// Thêm 1 phần tử vào danh sách

void pushList(List \*l, int element){

l->data[l->size]=element;

l->size++;

}

// Truy cập đến phần tử có vị trí index trong danh sách

int getList(List \*l, int index){

return l->data[index-1];

}

/\*Các hàm trên đồ thị\*/

// Khởi tạo đồ thị

void initGraph(Graph \*g){

g->n=n;

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

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

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

}

// Thêm cung e = (x,y) vào đồ thị

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

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

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

}

// Kiểm tra 2 đỉnh có kề nhau hay không

bool adjacent(Graph \*g, int x, int y){

return g->matrix[x][y]==1;

}

// Tìm tất cả đỉnh láng giềng(đỉnh kề) của đỉnh x

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

List L;

makenullList(&L);

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

if(adjacent(g,x,i)){

pushList(&L,i);

}

}

return L;

}

int mark[maxv];//Mảng toàn cục mark để đánh dấu 1 đỉnh đã được duyệt hay chưa

// Giải thuật duyệt sâu sử dụng ngăn xếp - DFS - STACK

List DFS\_Stack(Graph \*g, int x){

List rs;//Danh sách lưu các đỉnh đã được duyệt

makenullList(&rs);

Stack s;

makenullStack(&s);

pushStack(&s,x);

while(!emptyStack(&s)){ //Lập trong khi ngăn xếp chưa rỗng

int u = getStack(&s);

delSize(&s);

if(mark[u]==1) continue; //Nếu đỉnh u đã được duyệt thì bỏ qua các câu lệnh phía sau

pushList(&rs,u); //Nếu u chưa duyệt thay vì in ra thì chúng ta sẽ đưa vào danh sách rs(result) -> danh sách kết quả

mark[u]=1; //Đánh dấu u đã duyệt

List L = neighbors(g,u); //Tìm đỉnh láng giềng của u đưa vào danh sách L

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

int v = getList(&L,j); //Lấy đỉnh láng giềng u trong danh sách L ra -> Lấy đỉnh v

if(mark[v]==0){ //Nếu v chưa được duyệt

pushStack(&s,v); //Đưa vào ngăp xếp duyệt v

}

}

}

return rs;

}

int main(){

Graph g;

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

initGraph(&g);

int u,v;

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

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

addEdge(&g,u,v);

}

// Khởi tạo toàn bộ phần tử trong mảng mark bằng 0 vì chưa có đỉnh nào được duyệt

for(int i=1;i<=n;i++) mark[i]=0;

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

if(mark[i]==0){ //Nếu đỉnh chưa được duyệt

List L = DFS\_Stack(&g,i); //Gọi hàm và duyệt

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

int v = getList(&L,j);

printf("%d\n",v); //in

mark[v]=1; //Đánh dấu đã duyệt

}

}

}

return 0;

}

Cho một đồ thị **vô hướng đơn**. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị **theo chiều sâu** (sử dụng ĐỆ QUY) bắt đầu từ đỉnh 1.

#include <stdio.h>

typedef int ElementType;

int mark[100];

//List

typedef struct {

int data[100];

int size;

}List;

void make\_null(List\* L)

{

L->size = 0;

}

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

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

L->size++;

}

typedef struct{

int A[100][100];

int n;

}Graph;

void init\_graph(Graph\* G, int n)

{

G->n = n;

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

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

G->A[i][k] = 0;

}

List neighbors(Graph G,int dinh){

List list;

make\_null(&list);

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

{

if(i!=dinh && G.A[dinh][i] && G.A[i][dinh])

{

push\_back(&list,i);

}

}

return list;

}

void add\_edge(Graph \* G, int u, int v)

{

G->A[u][v] = G->A[v][u] = 1;

}

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

if(mark[dinh] == 1)

return;

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

mark[dinh] = 1;

List nei = neighbors(\*G,dinh);

for(int i = 0; i < nei.size; i++)

{

depth\_first\_search(G,nei.data[i]);

}

}

int main(){

Graph G;

int n, m, u, v, e;

freopen("dothi.txt", "r", stdin);

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

init\_graph(&G, n);

for (e = 0; e < m; e++) {

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

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

}

//Khoi tao mark

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

mark[e] = 0;

}

//

depth\_first\_search(&G,1);

//Neu do thi chua lien thong

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

{

if(mark[i] == 0)

{

depth\_first\_search(&G,i);

break;

}

}

return 0;

}

Cho một đồ thị **vô hướng đơn**. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.

Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.

#include<stdio.h>

#define MAX\_ELEMENTS 100

#define MAX\_VERTEXES 100

typedef int ElementType;

typedef struct {

int data[MAX\_ELEMENTS];

int front,rear;

}Queue;

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;

}

//List

typedef struct {

ElementType data[MAX\_ELEMENTS];

int size;

}List;

void make\_null(List\* L)

{

L->size = 0;

}

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

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

L->size++;

}

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

{

return L->data[i-1];

}

int count\_list(List \*L)

{

return L->size;

}

//

//Graph

typedef struct{

int A[100][100];

int n;

}Graph;

void add\_edge(Graph\* G, int u, int v){

G->A[u][v] = 1;

G->A[v][u] = 1;

}

//

void init\_graph(Graph\* G,int n)

{

G->n = n;

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

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

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

}

int adjacent(Graph\* G,int x,int y)

{

return G->A[x][y] == 1;

}

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

List L;

make\_null(&L);

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

{

if(adjacent(G,x,y) && y != x)

{

push\_back(&L,y);

}

}

return L;

}

void breath\_first\_search(Graph\* G)

{

Queue frontier;

int mark[MAX\_VERTEXES];

make\_null\_queue(&frontier);

int j;

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

mark[j] = 0;

push(&frontier,1);

mark[1] = 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);

}

}

//Kiem tra neu chua lien thong

//Kiem tra dinh nao chua duoc duyet

if(empty(&frontier))

{

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

{

if(mark[i] == 0){

mark[i] = 1;

push(&frontier,i);

}

}

}

}

}

int main()

{

Graph G;

int n, m, u, v, e;

freopen("dothi.txt", "r", stdin);

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

init\_graph(&G, n);

for (e = 0; e < m; e++) {

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

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

}

breath\_first\_search(&G);

return 0;

}

Bài 1

#include<stdio.h>

#define max\_E 100 //sl phtu trong mang

#define max\_V 100//sl dinh trong do thi

typedef int ElememtType;

typedef struct{

int data[max\_E];

int front, rear;

}Queue;

//khoitao

void make\_null(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

//them phtu

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

Q->rear++;

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

}

//tra ve phtu dau

int top(Queue\* Q){

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

}

//xoa phtu dau

void pop(Queue\* Q){

Q->front++;

}

//kt rong

int empty(Queue\* Q){

return Q->front > Q->rear;

}

//danh sach

typedef struct{

ElememtType data[max\_E];

int size;//sl phtu trong ds

}List;

//khotao ds rong

void makenull\_list(List\* L){

L->size=0;

}

//them phtu vao ds

void push\_list(List\* L, ElememtType x){

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

L->size++;

}

//tra ve phtu dau tien

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

return L->data[i-1];

}

//dem sl phtu

int count(List\* L){

return L->size;

}

//do thi

typedef struct{

int A[100][100];

int n;

}Graph;

//them cung

void add(Graph\* G, int u, int v){

G->A[u][v]=1;

G->A[v][u]=1;

}

//khoitao

void init(Graph\* G, int n){

G->n=n;

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

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

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

}

}

}

//kt ke

int adj(Graph\* G, int u, int v){

return G->A[u][v]=1;

}

//tao 1 ds chua cac dinh ke cua x

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

List L;

makenull\_list(&L);//khoitao ds rong

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

if(adj(G,x,i) && i != x)

push\_list(&L,i);

}

return L;

}

//duyet theo chieu rong

void bfs(Graph\* G){

Queue fon;//tao 1 hàng doi de luu cac dinh can duyet

int mark[max\_V];//tao 1 mang de danh dau cac dinh da duyet

make\_null(&fon);//khoi tao hang doi

int j;

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

mark[j]=0;//danh dau chua dinh nao duoc duyet

push(&fon,1);//dua dinh j vao hang doi

mark[1]=1;//danh dau da duyet

while(!empty(&fon)){//dk lap la ko rong

//lay dinh dau tien ra-xoa dinh dau

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

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

List list = neigh(G,x);//tim dinh ke cua dinh x

for(j=1;j<=list.size;j++){//duyet qua cac dinh ke

int y=element\_at(&list,j);//lay dinh ke x ra

if(mark[y]==0){//neu chua duoc duyet

mark[y]=1;//danh dau da duyet

push(&fon,y);//dua vao hang doi

}

}

//kt xem dinh nao chua duoc duyet

if(empty(&fon)){

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

if(mark[i]==0){

mark[i]=1;

push(&fon,i);

}

}

}

}

}

int main(){

Graph G;

int n,m,u,v;

freopen("dothi.txt","r",stdin);

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

init(&G,n);

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

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

add(&G,u,v);

}

bfs(&G);

return 0;

}

Bài tập 3- chuẩn bị thực hành

**Câu 3: in ra ma trận với file đọc vào là (u,v,w)**

#include <stdio.h>

#define MAX\_VERTICES 100

#define No\_EDGE 0

typedef struct {

int n, m;

int W[MAX\_VERTICES][MAX\_VERTICES];

} Graph;

// Kh?i t?o d? th?

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

G->n = n;

G->m = 0;

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

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

G->W[i][j] = No\_EDGE;

}

}

}

// Thêm cung vào d? th?

void add\_edge(Graph \*G, int u, int v, int w) {

if (u > G->n || v > G->n || u < 1 || v < 1) {

printf("Khong hop le\n");

} else {

G->W[u][v] = w;

G->W[v][u] = w; // Vì d? th? vô hu?ng nên c?n c?p nh?t c? 2 chi?u

G->m++;

}

}

/\*

// In ma tr?n tr?ng s? c?a d? th?

void print\_weight\_matrix(Graph G) {

// printf("Ma tran trong so cua do thi:\n");

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

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

printf("%d ", G.W[i][j]);

}

printf("\n");

}

}

\*/

int main() {

int n, m, u, v, w;

Graph G;

// M? file và d?c d? th? t? file

freopen("dothi.txt", "r", stdin);

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

// Kh?i t?o d? th?

init\_graph(&G, n);

// Thêm cung vào d? th?

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

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

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

}

// In ma tr?n tr?ng s? c?a d? th?

//print\_weight\_matrix(G);

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

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

printf("%d ", G.W[i][j]);

}

printf("\n");

}

return 0;

}

**Câu: In ra co stt - LIST**

#include <stdio.h>

typedef struct{

int data[100];

int size;

}List;

void make\_null(List\* L){

L->size=0;

}

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

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

L->size++;

}

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

return L->data[i-1];

}

/\* Vi?t mã l?nh c?a b?n ? dây \*/

//Ð?nh nghia hàm void print\_List(List\* L)

void print\_List(List\* L) {

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

printf("%d %d",i+1, L->data[i]);

printf("\n");

}

}

/\* H?t ph?n mã l?nh c?a b?n \*/

int main() {

List L;

int n, i, x;

scanf("%d", &n);

make\_null(&L);

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

scanf("%d", &x);

push\_back(&L, x);

}

print\_List(&L);

return 0;

}

**Câu 5: in ra có sô thứ tự - HÀNG ĐỢI**

#include <stdio.h>

typedef struct{

int data[100];

int front, rear;

}Queue;

void make\_null(Queue\* Q){

Q->front=0;

Q->rear=-1;

}

void enqueue(Queue\* Q, int x){

Q->rear++;

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

}

int front(Queue\* Q){

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

}

void dequeue(Queue\* Q){

Q->front++;

}

int empty(Queue\* Q){

return Q->front > Q->rear;

}

/\* Vi?t mã l?nh c?a b?n ? dây \*/

//Ð?nh nghia hàm void print\_Queue(Queue\* Q)

void print\_Queue(Queue\* Q) {

int count = 1; // S? th? t? ban d?u là 1

while (!empty(Q)) {

printf("%d %d\n", count, front(Q)); // In s? th? t? và ph?n t? tuong ?ng

dequeue(Q);

count++; // Tang s? th? t? lên cho ph?n t? ti?p theo

}

}

/\* H?t ph?n mã l?nh c?a b?n \*/

int main() {

Queue Q;

int n, i, x;

scanf("%d", &n);

make\_null(&Q);

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

scanf("%d", &x);

enqueue(&Q, x);

}

print\_Queue(&Q);

return 0;

}

**Câu 6: in ra theo stt - STACK**

#include <stdio.h>

typedef struct{

int data[100];

int top\_idx;

}Stack;

void make\_null(Stack\* S){

S->top\_idx=-1;

}

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

S->top\_idx++;

S->data[S->top\_idx]=x;

}

int top(Stack\* S){

return S->data[S->top\_idx];

}

void pop(Stack\* S){

S->top\_idx--;

}

int empty(Stack\* S){

return S->top\_idx ==-1;

}

/\* Vi?t mã l?nh c?a b?n ? dây \*/

//Ð?nh nghia hàm void print\_Stack(Stack\* S)

void print\_Stack(Stack\* S) {

int i = 1; // S? th? t? ban d?u là 1

while (!empty(S)) {

printf("%d %d\n", i++, top(S)); // In s? th? t? và ph?n t? tuong ?ng

pop(S);

}

}

/\* H?t ph?n mã l?nh c?a b?n \*/

int main() {

Stack S;

int n, i, x;

scanf("%d", &n);

make\_null(&S);

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

scanf("%d", &x);

push(&S, x);

}

print\_Stack(&S);

return 0;

}

Bài tập 4 - về nhà

Câu 1:Hãy viết chương trình đọc đồ thị từ tập tin và in ra tổng chi phí của 1 cung ,cụ thể nào đó và tổng chi phí của đồ thị này.

#include <stdio.h>

#define MAX\_VERTICES 100

typedef struct {

int v; // Ð?nh k?t thúc c?a cung

int d; // Kho?ng cách

int c; // Chi phí m?i mét

} Edge;

typedef struct {

int n;

int m;

Edge edges[MAX\_VERTICES \* (MAX\_VERTICES - 1) / 2];

} Graph;

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

G->n = n;

G->m = 0;

}

void add\_edge(Graph\* G, int u, int v, int d, int c) {

G->edges[G->m].v = v;

G->edges[G->m].d = d;

G->edges[G->m].c = c;

G->m++;

}

// Tính chi phí c?a m?t cung c? th?

int edge\_cost(Graph G, int u, int v) {

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

if (G.edges[i].v == v && i == u) {

return G.edges[i].d \* G.edges[i].c;

}

}

return -1; // Tr? v? -1 n?u cung không t?n t?i

}

int main() {

FILE \*file;

Graph G;

int n, m, u, v, d, c;

long long total\_cost = 0;

file = fopen("dothi.txt", "r");

if (file == NULL) {

printf("Khong the mo file.");

return 1;

}

fscanf(file, "%d%d", &n, &m);

init\_graph(&G, n);

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

fscanf(file, "%d%d%d%d", &u, &v, &d, &c);

add\_edge(&G, u - 1, v - 1, d, c);

}

fclose(file);

int u\_input, v\_input;

printf("Nhap vao dinh bat dau va dinh ket thuc cua cung can tinh chi phi: ");

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

int cost = edge\_cost(G, u\_input - 1, v\_input - 1);

if (cost != -1) {

printf("Chi phi cua cung tu dinh %d den dinh %d la: %d\n", u\_input, v\_input, cost);

} else {

printf("Khong tim thay cung tu dinh %d den dinh %d trong do thi.\n", u\_input, v\_input);

}

return 0;

}