LÝ THUYẾT ĐỒ THỊ

**Ma trận đỉnh - cung.**

#define MAX\_VERTICES 100

#define MAX\_EDGES 500

typedef struct {

int n,m;

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

} Graph;

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

int i,j;

G->n = n;

G->m = m;

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

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

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

}

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

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

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

}

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

int e;

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

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

return 1;

return 0;

}

int degree(Graph\* G, int x){

int e, deg = 0;

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

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

deg++;

}

return deg;

}

**Ma trận đỉnh – đỉnh:**

#define MAX\_VERTICES 100

typedef struct {

int n;

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

} Graph;

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

int i,j;

G->n = n;

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;

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

// Đa cung nên +=1 nếu đơn cung thì = 1;

//Nếu đồ thị có hướng thì bỏ đối xứng G->A[y][x] += 1;

}

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

return G->A[x][y] != 0;

}

int degree(Graph\* G, int x){

int y, deg = 0;

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

deg += G->A[x][y];

return deg;

}

**Ma trận đỉnh kề:**

typedef struct

{

int n;

List adj[MAX\_VERTICES]; /\* mang danh sach cac dinh ke \*/

} Graph;

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

{

int i, j;

G->n = n;

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

make\_null(&G->adj[i]);

}

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

{

push\_back(&G->adj[x], y); // y ke với x

push\_back(&G->adj[y], x); // x ke voi y

}

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

{

int j;

for (j = 1; j <= G->adj[x].size; j++)

if (element\_at(&G->adj[x], j) == y)

return 1;

return 0;

}

int degree(Graph \*G, int x)

{

return G->adj[x].size;

}

**Hàm neighbors**

List neighbors(Graph \*G, int x)

{

Graph G;

int y;

List list;

make\_null(&list);

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

if (adjacent(G, x, y))

push\_back(&list, y);

return list;

}

**Danh sách:**

#define MAX\_ELEMENTS 100

typedef int ElementType;

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++;

}

/\* 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 count\_list(List \*L)

{

return L->size;

}

**Stack**

#define MAX\_ELEMENTS 100

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;

}

**Queue**

#define MAX\_ELEMENTS 100

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;

}

**DFS Không có Parent**

**#include "Stack.h" Thêm cái Stack**

**#include "List.h" Thêm cái List**

**// xây dựng đồ thì bằng các loại ma trận**

List neighbors(Graph \*G, int x)

{

int y;

List list;

make\_null(&list);

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

if (adjacent(G, x, y))

push\_back(&list, y);

return list;

}

void DFS(Graph \*G)

{

Stack L;

int mark[MAX\_VERTICES];

make\_null\_stack(&L);

int j;

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

{

mark[j] = 0;

}

push(&L, 1);

mark[1] = 1;

while (!empty(&L))

{

int x = top(&L);

pop(&L);

// if (mark[x] != 0)

// continue;

printf("Duyet %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(&L, y);

}

}

}

}

// int mark[MAX\_VERTICES];

// void DFS(Graph\* G, int x){

// Stack L;

// make\_null\_stack(&L);

// int j;

// push(&L,x);

// mark[x] = 1;

// while(!empty(&L)){

// int x = top(&L);

// pop(&L);

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

// List list = neighbors(G, x);

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

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

// if (mark[y] != 1) {

// mark[y] = 1;

// push(&L, y);

// }

// }

// }

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

// if (mark[j] == 0){

// DFS(G, j);

// }

// }

**DFS Parent**

#define MAX\_VERTICES 100

#define MAX\_ELEMENTS 100

typedef struct

{

    int u;

    int parent;

} ELEMENT\_TYPE;

typedef struct

{

    ELEMENT\_TYPE data[MAX\_ELEMENTS];

    int size;

} Stack;

void make\_null\_stack(Stack \*S)

{

    S->size = 0;

}

void push(Stack \*S, ELEMENT\_TYPE x)

{

    S->data[S->size] = x;

    S->size++;

}

ELEMENT\_TYPE top(Stack \*S)

{

    return S->data[S->size - 1];

}

void pop(Stack \*S)

{

    S->size--;

}

int empty(Stack \*S)

{

    return S->size == 0;

}

typedef int ElementType;

typedef struct

{

    ElementType data[MAX\_ELEMENTS];

    int size;

} List;

typedef struct

{

    int n;

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

} Graph;

//=======================

Thêm danh sách ở đây.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

    int i, j;

    G->n = n;

    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;

    G->A[y][x] += 1;

}

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

{

    return G->A[x][y] != 0;

}

int degree(Graph \*G, int x)

{

    int y, deg = 0;

    for (y = 1; y <= G->n; y++)

        deg += G->A[x][y];

    return deg;

}

List neighbors(Graph \*G, int x)

{

    int y;

    List list;

    make\_null(&list);

    for (y = 1; y <= G->n; y++)

        if (adjacent(G, x, y))

            push\_back(&list, y);

    return list;

}

int mark[MAX\_VERTICES];

int parent[MAX\_VERTICES];

void DFS(Graph \*G, int x)

{

    Stack L;

    int v;

    make\_null\_stack(&L);

    ELEMENT\_TYPE pr;

    pr.u = x;

    pr.parent = 0;

    push(&L, pr);

    while (!empty(&L))

    {

        pr = top(&L);

        pop(&L);

        int u = pr.u, p = pr.parent;

        if (mark[u] != 0)

            continue;

        //      printf("Duyet: %d\n", u);

        mark[u] = 1;

        parent[u] = p;

        int v;

        for (v = 1; v <= G->n; v++)

        {

            if (adjacent(G, u, v))

            {

                pr.u = v;

                pr.parent = u;

                push(&L, pr);

            }

        }

    }

    for (v = 1; v <= G->n; v++)

        if (mark[v] == 0)

        {

            DFS(G, v);

        }

}

int main()

{

Đọc file như bt!!!

    for (u = 1; u <= G.n; u++)

        mark[u] = 0;

    DFS(&G, 1);

    for (u = 1; u <= G.n; u++)

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

    return 0;

}

**BFS Không có parent**

#define MAX\_VERTICES 100

**Xây dựng đồ thị như bình thường!!**

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

int y;

List list;

make\_null(&list);

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

if (adjacent(G, x, y))

push\_back(&list, y);

return list;

}

**Thêm List!!!!**

**Thêm Queue!!!**

int mark[MAX\_VERTICES];

void BFS(Graph\* G, int x) {

Queue L;

make\_null\_queue(&L);

int j;

push(&L, x);

mark[x] = 1;

while (!empty(&L))

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

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] != 1) {

mark[y] = 1;

push(&L, y);

}

}

}

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

if (mark[j] == 0)

{

BFS(G, j);

}

}

**BFS Parent**

**#include "Queue.h"**

**#include "List.h"**

#define MAX\_VERTICES 100

List neighbors(Graph \*G, int x)

{

int y;

List list;

make\_null(&list);

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

if (adjacent(G, x, y))

push\_back(&list, y);

return list;

}

int mark[MAX\_VERTICES];

int parent[MAX\_VERTICES];

void BFS(Graph \*G, int x)

{

Queue L;

make\_null\_queue(&L);

int j;

int pr = 0;

push(&L, x);

mark[x] = 1;

parent[x] = pr;

while (!empty(&L))

{

int x = top(&L);

pop(&L);

pr = x;

List list = neighbors(G, x);

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

{

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

if (mark[y] != 1)

{

mark[y] = 1;

parent[y] = pr;

push(&L, y);

}

}

}

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

/\* N?u d?nh j chua du?c duy?t, duy?t n� \*/

if (mark[j] == 0)

{

BFS(G, j);

}

}

**DFS Đệ quy**

int mark[MAX\_VERTICES];

void traversal(Graph\* G, int x) {

if (mark[x] == 1)

return;

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

mark[x] = 1;

List list = neighbors(G, x);

int j;

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

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

traversal(G, y);

}

}

void DFS(Graph\* G) {

int j;

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

mark[j] = 0;

traversal(G, 1);

}

**DFS Đệ quy Parent**

int mark[MAX\_VERTICES];

int parent[MAX\_VERTICES];

void traversal(Graph \*G, int x, int p)

{

if (mark[x] == 1)

return;

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

parent[x] = p;

mark[x] = 1;

List list = neighbors(G, x);

int j;

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

{

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

traversal(G, y, x);

}

}

void DFS(Graph \*G)

{

int j;

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

mark[j] = 0;

traversal(G, 1, 0);

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

{

if (mark[j] == 0)

traversal(G, j, 0);

}

}

**Kiểm tra chu Trình đồ thị vô hướng**

#define white 0

#define black 1

#define gray 2

#define MAX\_ELEMENTS 100

#define MAX\_VERTICES 100

int color[MAX\_VERTICES];

int cycle=0;

void chtrinh(Graph\* G,int u,int p){

color[u]=gray;

int j;

List list =neighbors(G,u);

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

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

if(y==p)

continue;

else if(color[y]==gray){

cycle=1;

return;

}

else if(color[y]==white){

p=u;

chtrinh(G,y,p);}

}

color[u]=black;

}

int ktra(Graph\* G){

int j;

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

{

color[j]=white;

}

cycle=0;

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

if(color[j]==white)

chtrinh(G,j,0);

}

return cycle;

}

**Main(**

**if(ktra(&G)==1)**

**printf("YES");**

**else**

**printf("NO"); )**

**Đồ thị phân chia**

int color[MAX\_VERTICES],conflict;

void colorsize(Graph\* G,int u, int c){

if(color[u]==white)

color[u]=c;

List list = neighbors(G,u);

int j;

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

{

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

if(color[y]==white)

colorsize(G,y,!c);

else if(color[y]==c)

conflict=1;;

}

}

int kt\_bigraph(Graph\* G){

int i;

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

color[i]=-1;

}

conflict=0;

colorsize(G,1,0);

return conflict;

}

void in(Graph\* G){

int i,j;

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

if(color[i]==0)

printf("%d ",i);

}

printf("\n");

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

if(color[j]==1)

printf("%d ",j);

}

}

**Liên thông mạnh**

int min(int x,int y){

if(x<y)

return x;

else return y;

}

int idx,on\_stack[MAX\_VERTICES],num[MAX\_VERTICES],min\_num[MAX\_VERTICES],t;

int strong\_connect(Graph\* G,int x){

Stack S;

makenull\_stack(&S);

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

idx++;

push(&S,x);

on\_stack[x]=1;

List L=neighbors(G,x);

int i;

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

int v=element\_at(&L,i);

if( num[v]==-1){

strong\_connect(G,v);

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

}

else if(on\_stack[v]==1)

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

}

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

int u;

do{

u=top(&S);

pop(&S);

on\_stack[u]=0;

t++;

}while(u!=x);

}

return t;

}

int main(){

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

Graph G;

// List list;

int n, m, u, v,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);

}

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

num[e]=-1;

on\_stack[e]=0;

}

idx=1;

// printf("%d \n",strong\_connect(&G,1));

if(strong\_connect(&G,1)==1)

printf("strong connected");

else printf("unconnected");

}

**Come in come out**

int min(int x,int y){

if(x<y)

return x;

else return y;

}

Stack S;

int idx,on\_stack[MAX\_VERTICES],num[MAX\_VERTICES],min\_num[MAX\_VERTICES],t;

int strong\_connect(Graph\* G,int x){

// Stack S;

// makenull\_stack(&S);

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

idx++;

push(&S,x);

on\_stack[x]=1;

List L=neighbors(G,x);

int i;

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

int v=element\_at(&L,i);

if( num[v]==-1){

strong\_connect(G,v);

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

}

else if(on\_stack[v]==1)

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

}

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

if(t==0){

int u;

do{

u=top(&S);

pop(&S);

on\_stack[u]=0;

t++;

}while(u!=x);

}}

return t;

}

int main(){

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

Graph G;

// List list;

makenull\_stack(&S);

int n, m, u, v,e,p=1;

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

init\_graph(&G, n);

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

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

if (p==1)

{

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

}

else{

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

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

}

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

num[e]=-1;

on\_stack[e]=0;

}

idx=1;

// printf("%d\n",strong\_connect(&G,1));

if((strong\_connect(&G,1)==n))

{printf("OKIE");}

else printf("NO");

}

**Dijkstra**

int mark[100],pi[100],p[100];

void dijkstra(Graph\* G, int s){

int v,k,min\_pi,u,j;

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

mark[u]=0;

pi[u]=9999;

}

pi[s]=0;

p[s]=-1;

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

min\_pi=9999;

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

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

min\_pi=pi[j];

u=j;

}

mark[u]=1;

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

if(G->A[u][v]!=0&&mark[v]==0)

if(pi[u]+G->A[u][v]<pi[v]){

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

p[v]=u;

}

}

}

**int mark[100],pi[100],p[100];**

**void dijkstra(Graph\* G, int s){**

**int v,k,min\_pi,u,j;**

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

**mark[u]=0;**

**pi[u]=9999;**

**}**

**pi[s]=0;**

**p[s]=-1;**

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

**min\_pi=9999;**

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

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

**min\_pi=pi[j];**

**u=j;**

**}**

**mark[u]=1;**

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

**if(G->A[u][v]!=0&&mark[v]==0)**

**if(pi[u]+G->A[u][v]<pi[v]){**

**pi[v]=pi[u]+G->A[u][v];**

**p[v]=u;**

**}**

**}**

**}**

int main(){

Graph G;

freopen(“dothi.txt”, “r”, stdin);

int n, m, u, v, 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\_adge(&G,u,v,w);

}

dijkstra(&G,1);

if(pi[n]==9999)

printf(“-1”);

else

printf(“%d”,pi[n]);

return 0;

}

**Bellman – Ford**

#include <stdio.h>

typedef struct

{

int u, v;

int w;

} edge;

typedef struct

{

edge A[100];

int m, n;

} Graph;

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

{

G->n = n;

G->m = 0;

}

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

{

G->A[G->m].u = u;

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

G->A[G->m].w = w;

G->m++;

}

int p[100], pi[100];

int Bellman(Graph \*G, int s)

{

int u, v, w, it, k, t = 0;

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

{

pi[u] = 9999;

}

pi[s] = 0;

p[s] = -1;

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

{

for (k = 0; k < G->m; k++)

{

u = G->A[k].u;

v = G->A[k].v;

w = G->A[k].w;

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

{

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

p[v] = u;

}

}

}

for (k = 0; k < G->m; k++)

{

u = G->A[k].u;

v = G->A[k].v;

w = G->A[k].w;

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

t = 1;

}

return t;

}

int main()

{

Graph G;

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

freopen("dothi.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\_adge(&G, u, v, w);

}

// scanf("%d",&s);

Bellman(&G,1);

// if (Bellman(&G, 1 == 1))

// printf("negative cycle");

// else

// printf("ok");

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

{

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

}

return 0;

}

**Dijkstra có đầu và cuối**

int mark[100],pi[100],p[100];

void dijkstra(Graph\* G, int s,int t){

int v,k,min\_pi,u,j;

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

mark[u]=0;

pi[u]=9999;

}

pi[s]=0;

p[s]=-1;

for(k=1;k<=t;k++){

min\_pi=9999;

for(j=1;j<=t;j++)

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

min\_pi=pi[j];

u=j;

}

mark[u]=1;

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

if(G->A[u][v]!=0&&mark[v]==0)

if(pi[u]+G->A[u][v]<pi[v]){

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

p[v]=u;

}

}

}

int main(){

scanf("%d%d",&s,&t);

if (s<t){

dijkstra(&G,s,t);

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

}

else {

dijkstra(&G,t,s);

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

}

return 0;

}

**Dijkstra Đếm đường đi từ s ->n**

int mark[100],pi[100],p[100];

int t[100];

void dijkstra(Graph\* G, int s){

int v,k,min,u,j;

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

mark[u]=0;

pi[u]=9999;

t[u]=0;

}

pi[s]=0;

p[s]=-1;

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

min=9999;

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

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

min=pi[j];

u=j;

}

mark[u]=1;

t[u]=1;

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

if(G->A[u][v]!=0&&mark[v]==0){

if(pi[u]+G->A[u][v]<pi[v]){

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

t[v]=t[u];

p[v]=u;

}

else if (pi[u]+G->A[u][v]==pi[v])

t[v]+=t[u];

}

}

}

int main(){

Graph G;

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

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

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

init\_graph(&G, n);

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

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

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

}

dijkstra(&G,1);

if(pi[n]==9999)

printf("-1 0");

else

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

return 0;

}

**Liên thông in đỉnh chẵn lẻ**

List neighbors(Graph \*G, int u)

{

List list;

make\_list(&list);

int v;

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

if (adjacent(G, u, v))

push\_back(&list, v);

return list;

}

/\*Khoi tao bien\*/

int on\_stack[MAX\_VERTICES], num[MAX\_VERTICES], min\_num[MAX\_VERTICES];

int k = 0;

int Check[MAX\_VERTICES];

Stack S;

Stack check;

int cntLT = -1;

List PA[MAX\_VERTICES];

/\*Ket thuc khoi tao\*/

/\*Min\*/

int min(int a, int b)

{

return a < b ? a : b;

}

/\*End Min\*/

/\* Duyệt đồ thị bắt đầu từ đỉnh x \*/

void strong\_connect(Graph \*G, int x)

{

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

k++;

push(&S, x);

on\_stack[x] = 1;

List list = neighbors(G, x);

int j;

/\* Xét các đỉnh kề của nó \*/

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 (on\_stack[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);

// push(&check, x);

cntLT++;

make\_list(&PA[cntLT]);

int w;

do

{

w = top(&S);

// printf("%d ", w);

push\_back(&PA[cntLT], w);

pop(&S);

on\_stack[w] = 0;

} while (w != x);

// printf("\n");

}

}

void Tarjan(Graph \*G)

{

int v;

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

{

num[v] = -1;

on\_stack[v] = 0;

}

k = 1;

make\_stack(&S);

make\_stack(&check);

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

{

if (num[v] == -1)

strong\_connect(G, v);

}

}

int Xuli(Graph \*G, int n)

{

int i;

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

{

int j;

int flag = 1;

for (j = 1; j <= PA[i].size; j++)

{

if (element\_at(&PA[i], j) % 2 != 0) // chan thi khac 0 le thi bang 0

{

flag = 0;

break;

}

}

if (flag == 1)

{

push(&check, i);

}

}

if (empty\_stack(&check))

return 0;

return 1;

}

int main(int argc, char const \*argv[])

{

Graph G;

int n, m, e, u, v;

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);

}

Tarjan(&G);

if (Xuli(&G, cntLT))

printf("Co bo phan lien thong toan dinh chan!!\n");

else

printf("Khong co bo phan lien thong toan dinh chan!!\n");

while (!empty\_stack(&check))

{

int i = top(&check);

pop(&check);

int j;

for (j = 1; j <= PA[i].size; j++)

{

printf("%d ", element\_at(&PA[i], j));

}

printf("\n");

}

printf("Cac bo phan lien thong la:\n");

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

{

int j;

for (j = 1; j <= PA[i].size; j++)

{

printf("%d ", element\_at(&PA[i], j));

}

printf("\n");

}

return 0;

}

**Số đỉnh lớn nhất của bộ phận liên thông**

**Sử dụng cái trên thay đổi xử lí**

void Xuli(Graph \*G)

{

int max = 0;

if (cntLT + 1 == 1)

{

for (int i = 1; i <= PA[cntLT].size; i++)

{

printf("%d ", element\_at(&PA[cntLT], i));

}

}

else

{

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

{

if (max < PA[i].size)

{

max = PA[i].size;

}

}

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

{

if (max == PA[i].size)

{

for (int j = 1; j <= PA[i].size; j++)

{

printf("%d ", element\_at(&PA[i], j));

}

printf("\n");

}

}

}

}

**Đường đi nhắn nhất bến xe**

#include <stdio.h>

typedef struct

{

int A[100][100];

int n;

} Graph;

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

{

int i, j;

G->n = n;

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

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

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

}

void add\_adge(Graph \*G, int x, int y, int w)

{

G->A[x][y] = w;

G->A[y][x] = w;

}

int mark[100], pi[100], p[100];

void dijkstra(Graph \*G, int s)

{

int v, k, min\_pi, u, j;

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

{

mark[u] = 0;

pi[u] = 9999;

}

pi[s] = 0;

p[s] = -1;

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

{

min\_pi = 9999;

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

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

{

min\_pi = pi[j];

u = j;

}

mark[u] = 1;

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

if (G->A[u][v] != 0 && mark[v] == 0)

if (pi[u] + G->A[u][v] < pi[v])

{

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

p[v] = u;

}

}

}

int main()

{

Graph G;

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

int n, m, u, v, 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\_adge(&G, u, v, w);

}

int s;

scanf("%d", &s);

dijkstra(&G, s);

printf("%0.2f", (float)(pi[n] \* (60 / 10)));

return 0;

}