LTDt Tài liệu cuối kỳ

Lý thuyết đồ thị (Trường Đại học Cần Thơ)

Danh Sách Cung (DSC)

```
#include <stdio.h>

#define MAX_M 500
typedef struct {
    int u, v;
} Edge;
typedef struct {
    int n, m;
    Edge edges[MAX_M];
} Graph;
```

Init_graph():

```
void init_graph(Graph *pG, int n){
   pG->n= n;
   pG->m = 0;
}
```

Add edge():

```
void add_edge(Graph *pG, int u, int v) {
   pG->edges[pG->m].u = u;
   pG->edges[pG->m].v = v;
   pG->m++;
}
```

Add edge() nâng cao:

```
void add_edge(Graph *pG, int u, int v) {
  if((u >= 1) & (u <= pG->n )& (v >= 1 ) & (v <=pG->n ) || u==v) {
      pG->edges[pG->m].u = u;
      pG->edges[pG->m].v = v;
      pG->m++;
  }
}
**đơn đồ thị có hướng**
void add_edge(Graph *pG, int u, int v) {
   int T=1;
   if(pG->m > 0){
   for (int i = 0; i < pG->m ; i++){
      if((u == pG -> edges[i].u) && (v == pG -> edges[i].v) ){
        T= 0;
        break;
       }
   }
      if(T== 1){
         pG->edges[pG->m].u = u;
         pG->edges[pG->m].v = v;
```

```
            pG->m++;
        }
    }
    else{
        pG->edges[pG->m].u = u;
        pG->edges[pG->m].v = v;
        pG->m++;
    }
}
**đơn đồ thị vô hướng**
void add_edge(Graph *pG, int u, int v) {
    if(u == v ){
        return;
    }
    for(int i = 0; i < pG->m ;i++){
        if((pG->edges[i].u == u && pG->edges[i].v == v) || (pG->edges[i].u == v && pG->edges[i].v
== u)){
            return;
        }
    }
    pG->edges [pG->m].u = u;
    pG->edges [pG->m].v = v;
    pG->m++;
}
```

Adjacent():

```
**đồ thị có hướng**
int adjacent(Graph *pG, int u, int v) {
    int i;
    for(i = 0 ; i < pG->m ; i++){
        if((u == pG->edges[i].u && v == pG->edges[i].v))
        return 1;
    }
    return 0;
}
**đồ thị vô hướng**
int adjacent(Graph *pG, int u, int v) {
        int e;
        for (e = 0; e < pG->m; e++){
            if ((pG->edges[e].u == u && pG->edges[e].v == v) ||(pG->edges[e].u == v && pG-
>edges[e].v == u)){
```

```
            return 1;
        }
    }
  return 0;
}
```

Degree():

```
int degree(Graph *pG, int u){
  int e, deg_u = 0;
//Duyệt qua từng cung 0, 1, 2, ..., m - 1
  for (e = 0; e < pG->m; e++) {
//Nếu cung có dạng (u, -)
    if (pG->edges[e].u == u)
    deg_u++;
//Nếu cung có dạng (-, u)
    if (pG->edges[e].v == u)
    deg_u++;

  }
return deg_u;
}
**Hàm main()**
int main() {
      Graph G;
      int n, m, e, u, v;
      //Đọc số đỉnh và số cung & khởi tạo đồ thị
      scanf("%d%d", &n, &m);
      init_graph(&G, n);
      //Đọc m cung và thêm vào đồ thị
      for (e = 0; e < m; e++) {
            scanf("%d%d", &u, &v);
            add_edge(&G, u, v);
      }
      for (int u = 1; u <= n; u++)
        printf("deg(%d) = %d\n", u, degree(&G, u));

      return 0;
}
```

Ma Trận kề(MTK)

```
#include <stdio.h>
#define MAX_N 100
```

```
typedef struct {
	int n, m;
	int A[MAX_N][MAX_N];
} Graph;
```

Init_graph() Add edge() Degree() Neighbour():

đơn đồ thị vô hướng

```
void init_graph(Graph *G, int n) {
   G->n = n;
   G->m = 0;
   int u,v;
   for (u = 1; u <= n; u++)
      for (v = 1; v <= n; v++)
         scanf("%d",&G->A[u][v]);
}
void add_edge(Graph *G, int u, int v){
        G->A[u][v] = 1;
        G->A[v][u] = 1;
        G->m++;
};
int deg(Graph* G, int x){
int de=0;
int i;
for(i=1;i<=G->n;i++)
if(G->A[x][i]==1)
de++;
return de;
}
```

đơn đồ thị có hướng

```
void add_edge(Graph *pG, int u, int v) {
   pG->A[u][v]=1;
}
```

đồ thị vô hướng (có thể chứa đa cung và chứa khuyên)

```
void add_edge(Graph *G, int u, int v){
        if(G->A[u][v]!=0 || G->A[v][u]!=0){
                G->A[u][v]= 2 ;
                G->A[v][u]= 2;
                G->m++;
        }
        else{
        G->A[u][v]=1;
        G->A[v][u]=1;
```

```
        G->m++;
    }
}
int degree(Graph *pG, int u) {
    int v, deg_u = 0;
    for(v=0; v<= pG->n; v++){
        deg_u += pG->A[u][v];
    }
    return deg_u += pG->A[u][u];
}
void neighbours(Graph *pG, int u){
  int v, j;
    for(v=0; v<=pG->n ;v++)
        for(j=1; j<=pG->A[u][v] ;j++)
            printf("%d ", v);
    printf("\n") ;
}
```
đồ thị có hướng (có thể chứa đa cung và chứa khuyên)
```
void add_edge(Graph *pG, int u, int v) {
    pG->A[u][v] += 1;
    pG->m++;
}
int degree(Graph *pG, int u) {
    int v, deg_u = 0;
    for(v=0; v<= pG->n; v++){
        deg_u += pG->A[u][v]+pG->A[v][u];
    }
    return deg_u;
}
void neighbours(Graph *pG, int u){
  int v, j;
    for(v=0; v<=pG->n ;v++)
        for(j=1; j<=pG->A[u][v] ;j++)
            printf("%d ", v);
    printf("\n") ;
}
```
Hàm main()
```
int main() {
        Graph G;
        int n, m, e, u, v;
```

```
        //Đọc số đỉnh và số cung & khởi tạo đồ thị
        scanf("%d%d", &n, &m);
        init_graph(&G, n);
        //Đọc m cung và thêm vào đồ thị
        for (e = 0; e < m; e++) {
                scanf("%d%d", &u, &v);
                add_edge(&G, u, v);
        }
   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");
   }
   for(i=1;i<=n;i++){
        printf("neighbours(%d) = ",i);
        neighbours(&G,i);
   }
        return 0;
}
```

Đọc đồ thị từ tập tin:

```
#include<stdio.h>
#define MAX_N 100
typedef struct{
int n; // Voi n la so dinh cua ma tran
int A[MAX_N][MAX_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_edge(Graph *G, int x, int y){
G->A[x][y]=1;
G->A[y][x]=1;
}
int main(){
int e,n,m,u,v,i,j;
Graph G;
```

```
freopen("dt1.txt","r",stdin);
scanf("%d%d",&n,&m);
init_graph(&G,n);
for(e=1;e<=n;e++){
scanf("%d%d",&u,&v);
add_edge(&G,u,v);
}
for (i=1;i<=G.n;i++){
for (j=1;j<=G.n;j++){
printf("%d ",G.A[i][j]);
}
printf("\n");
}
return 0;
}
```

BFS

Duyệt theo chiều rộng từ đỉnh 1: (DSC)

```
#include <stdio.h>
#define MAX 500
typedef struct{
   int Data[MAX];
   int front, rear;
}Queue;

void makeNullQueue(Queue *Q){
   Q->rear = -1;
   Q->front = 0;
}

void enqueue(Queue *Q, int x){
   Q->rear++;
   Q->Data[Q->rear]=x;
}

void dequeue (Queue *Q){
   Q->front++;
}

int empty(Queue *Q){
   return Q->front > Q->rear;
```

```
}

int front(Queue *Q){
   return Q->Data[Q->front];
}

int Mark[MAX];
void BFS(Graph *G, int s){
   Queue Q;
   makeNullQueue(&Q);
   enqueue(&Q,s);
   while(!empty(&Q)){
      int v,u=front(&Q);
      dequeue(&Q);
      if(Mark[u])
          continue;
      printf ("%d\n",u);
      Mark[u]=1;
      for(v = 1 ;v <= G->n ; v++)
         if(adjacent(G, u,v))
            enqueue(&Q,v);
   }
}

int main(){
   Graph G;
   int n,m,u,v,i;
   //freopen("dt.txt", "r", stdin);
   scanf("%d %d",&n,&m);
//Khởi tạo đồ thị
   init_graph(&G, n);
//Thêm cung vào đồ thị
   for(i=0 ; i < m ; i++){
   scanf ("%d %d",&u,&v);
   add_edge(&G, u,  v);
   }
   for(i=1 ; i<=G.n ; i++){
      Mark[i]=0;
      BFS(&G,1);
   }
```

```
}
```

Duyệt theo chiều rộng từ đỉnh y: (DSC)

```
#include <stdio.h>
#define MAX 500
int main(){
   Graph G;
   int n,m,u,v,i;
   //freopen("dt.txt", "r", stdin);
   scanf("%d %d",&n,&m);
//Khởi tạo đồ thị
   init_graph(&G, n);
//Thêm cung vào đồ thị
   for(i=0 ; i < m ; i++){
   scanf ("%d %d",&u,&v);
   add_edge(&G, u,  v);
   }
   for(i=1 ; i<=G.n ; i++)
   Mark[i]=0;
   int y;
   scanf("%d",&y);
   BFS(&G,y);
}
```

DFS

Duyệt theo chiều sâu từ đỉnh 1: (DSC)

```
#include<stdio.h>
#define MAX 500
#define MAX_SIZE 100
typedef int ElementType;
typedef struct{
        ElementType data[MAX_SIZE];
        int top_idx;
}Stack;

void makeNullStack(Stack *pS){
        pS->top_idx=-1;
}
void push(Stack *pS, ElementType u){
        pS->top_idx++;
        pS->data[pS->top_idx]=u;
```

```
}

ElementType top(Stack *pS){
	return pS->data[pS->top_idx];
}

void pop(Stack *pS){
	pS->top_idx--;
}

int empty(Stack *pS){
	return pS->top_idx == -1;
}
int mark[MAX];
void DFS(Graph *pG, int s) {
	Stack S;
	makeNullStack(&S);
	push(&S,s);
	while(!empty(&S)) {
		int u = top(&S); pop(&S);
		if(mark[u]!=0)
			continue;
		printf("%d\n",u);
		mark[u]=1;
		int v;
		for(v=pG->n;v>=1;v--)
			if(adjacent(pG,u,v))
				push(&S,v);
	}
}

int main() {
	Graph G;
	int u, v, n, m, i;
	scanf("%d%d", &n, &m);
	initGraph(&G, n);
	for(i = 0; i < m; i++) {
	scanf("%d%d", &u, &v);
	addEdge(&G, u, v);
	}
```

```
        for(i = 1; i <= G.n; i++)
        mark[i] = 0;
        DFS(&G, 1);
        return 0;
}
```

Duyệt theo chiều sâu từ đỉnh y: (DSC)

```
int main() {
        Graph G;
        int u, v, n, m, i;
        scanf("%d%d", &n, &m);
        initGraph(&G, n);
        for(i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        addEdge(&G, u, v);
        }
        for(i = 1; i <= G.n; i++)
        mark[i] = 0;
        int s;
   scanf("%d",&s);
        BFS(&G, s);
        return 0;
}
```

cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1. (DSC)

```
#include<stdio.h>
#define MAX_N 500
#define MAX_SIZE 100
int mark[MAX_N];
int parent[MAX_N];
typedef struct{
        int u,p;
}ElementType;
typedef struct{
        ElementType data[MAX_SIZE];
        int front,rear;
}Queue;
void make_null_queue(Queue *pQ){
        pQ->front=0;
        pQ->rear=-1;
}
```

```
void enqueue(Queue *pQ,ElementType u){
	pQ->rear++;
	pQ->data[pQ->rear]=u;
}
int empty(Queue *pQ){
	return pQ->front > pQ->rear;
}
ElementType front(Queue *pQ){
	return pQ->data[pQ->front];
}
void dequeue(Queue *pQ){
	pQ->front++;
}
int mark[MAX_N];
int parent[MAX_N];
void BFS(Graph *pG,int s){
	Queue Q;
	make_null_queue(&Q);
	ElementType pair;
	pair.u=s;
	pair.p=-1;
	enqueue(&Q,pair);
	int v;
	while(!empty(&Q)){
		ElementType pair=front(&Q); dequeue(&Q);
		int u=pair.u,p=pair.p;
		if(mark[u]!=0)
			continue;
		mark[u]=1;
		parent[u]=p;
		for(v=1;v<=pG->n;v++){
			if(adjacent(pG,u,v)){
				ElementType pair;
				pair.u=v; pair.p=u;
				enqueue(&Q,pair);
			}
		}
	}
}
int main(){
```

```
	Graph G;
	int n,m,u,v,e;
	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);
	}
	for(u=1;u<=G.n;u++){
		mark[u]=0;
		parent[u]=-1;
	}
	for(u=1;u<=G.n;u++){
		if(mark[u]==0)
			BFS(&G,u);
	}
	for(u=1;u<=G.n;u++){
		printf("%d %d\n",u,parent[u]);
	}
	return 0;
}
```

cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều sâu bắt đầu từ đỉnh 1. (DSC)

```
#include<stdio.h>
#define MAX_N 500
#define MAX_SIZE 100
int mark[MAX_N];
int parent[MAX_N];
void DFS(Graph *pG,int u,int p){
	if(mark[u]==1){
		return;
	}
	mark[u]=1;
	parent[u]=p;
	int v;
	for(v=1;v<=pG->n;v++){
		if(adjacent(pG,u,v) && mark[v]==0){
			DFS(pG,v,u);
		}
	}
```

```
}
int main(){
	Graph G;
	int n,m,u,v,e;
	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);
	}
	for(u=1;u<=G.n;u++){
		mark[u]=0;
		parent[u]=-1;
	}
	for(u=1;u<=G.n;u++){
		if(mark[u]==0)
			DFS(&G,u,-1);
	}
	for(u=1;u<=G.n;u++){
		printf("%d %d\n",u,parent[u]);
	}
	return 0;
}
```

Liên Thông

Viết chương trình đọc một đồ thị từ bàn phím và kiểm tra xem đồ thị có liên thông không.(DSC)

```
#include<stdio.h>
#define MAX_N 500
#define MAX_SIZE 100
typedef int ElementType;
typedef struct{
	ElementType data[MAX_SIZE];
	int top_idx;
}Stack;
void make_null_stack(Stack *pS){
	pS->top_idx=-1;
}
void push(Stack *pS,ElementType u){
	pS->top_idx++;
```

```
        pS->data[pS->top_idx]=u;
}
ElementType top(Stack *pS){
        return pS->data[pS->top_idx];
}
void pop(Stack *pS){
        pS->top_idx--;
}
int empty(Stack *pS){
        return pS->top_idx==-1;
}
int mark[MAX_N];
void DFS(Graph *pG,int s){
        Stack S;
        make_null_stack(&S);
        push(&S,s);
        int v;
        while(!empty(&S)){
                int u=top(&S);
                pop(&S);
                if(mark[u]!=0)
                        continue;
                mark[u]=1;
                for(v=1;v<=pG->n;v++){
                        if(adjacent(pG,u,v))
                                push(&S,v);
                }
        }
}
int connected(Graph *pG){
        int u;
        for(u=1;u<=pG->n;u++){
                mark[u]=0;
        }
        DFS(pG,1);
        for(u=1;u<=pG->n;u++){
                if(mark[u]==0)
```

```
                return 0;
        }
        return 1;
}
int main(){
        Graph G;
        int n,m,u,v,e;
        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);
        }
        if(connected(&G)){
                printf("CONNECTED");
        }
        else printf("DISCONNECTED");
        return 0;
}
```

Viết chương trình đọc một đồ thị từ bàn phím và đếm số bộ phận liên thông của đồ thị.(DSC)

```
#include<stdio.h>
#define MAX_N 500
#define MAX_SIZE 100
……giống bài trên
int main(){
        Graph G;
        int n,m,u,v,e;
        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);
        }
        for(u=1;u<=G.n;u++){
                mark[u]=0;
        }
```

```
	int cnt=0;
	for(u=1;u<=G.n;u++){
		if(mark[u]==0){
			DFS(&G,u);
			cnt++;
		}
	}
	printf("%d",cnt);
	return 0;
}
```

Viết chương trình đọc một đồ thị từ bàn phím và đếm số đỉnh của bộ phận liên thông của đỉnh 1.(DSC)

```
…giống ở trên
int mark[MAX_N];
int nb_u;
void DFS(Graph *pG,int s){
	Stack S;
	make_null_stack(&S);
	push(&S,s);
	int v;
	nb_u=0;
	while(!empty(&S)){
		int u=top(&S);
		pop(&S);
		if(mark[u]!=0)
			continue;
		mark[u]=1;
		nb_u++;
		for(v=1;v<=pG->n;v++){
			if(adjacent(pG,u,v))
				push(&S,v);
		}
	}
}
int main(){
	Graph G;
	int n,m,u,v,e;
```

```
	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);
	}
	for(u=1;u<=G.n;u++){
		mark[u]=0;
	}
	nb_u=0;
	DFS(&G,1);
	printf("%d",nb_u);
	return 0;
}
```

Viết chương trình đọc một đồ thị từ bàn phím và đếm số đỉnh của bộ phận liên thông của đỉnh S.(DSC)

```
int main(){
	Graph G;
	int n,m,u,v,e;
	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);
	}
	for(u=1;u<=G.n;u++){
		mark[u]=0;
	}
	int s;
	scanf("%d",&s);
	nb_u=0;
	DFS(&G,s);
	printf("%d",nb_u);
	return 0;
}
```

Viết chương trình đọc một đồ thị từ bàn phím, tìm bộ phận liên thông có nhiều đỉnh nhất.(DSC)

```
int main(){
	Graph G;
	int n,m,u,v,e;
	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);
	}
	for(u=1;u<=G.n;u++){
		mark[u]=0;
	}
	nb_u=0;
	int max_cnt=0;
	for(u=1;u<=G.n;u++){
		if(mark[u]==0){
			DFS(&G,u);
			if(nb_u>max_cnt){
				max_cnt=nb_u;
			}
		}
	}
	printf("%d",max_cnt);
	return 0;
}
```

Chu Trình

chương trình đọc vào một đơn đồ thị và kiểm tra xem nó có chứa chu trình hay không.

Giống duyệt cây theo chiều rộng từ đỉnh 1

```
#define WHITE 0
#define GRAY 1
#define BLACK 2
int color[MAX_N];
int has_circle;
void DFS(Graph *pG,int u){
	color[u]=GRAY;
	int v;
	for(v=1;v<=pG->n;v++){
```

```
            if(adjacent(pG,u,v)){
                if(color[v]==WHITE)
                    DFS(pG,v);
                else if(color[v]==GRAY)
                    has_circle=1;
            }
        }
        color[u]=BLACK;
}
int main(){
        Graph G;
        int n,m,u,v,e;
        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);
        }
        for(u=1;u<=G.n;u++){
            color[u]=WHITE;
        }
        has_circle=0;
        for(u=1;u<=G.n;u++){
            if(color[u]==WHITE)
                DFS(&G,u);
        }
        if(has_circle==1){
            printf("CIRCLED");
        }
        else printf("NO CIRCLE");
        return 0;
}
```

Moore – dijkstra

Moore – dijkstra(pi and p)

****có hướng****

```
#include <stdio.h>
#define MAXN 1000
```

```
#define INFINITY 9999999
int pi[MAXN];
int p[MAXN];
typedef struct {
int u, v;
int w;
} Edge;
typedef struct {
int n, m;
Edge edges[MAXN];
} Graph;
void init_graph(Graph *G, int n, int m){
G->n = n;
G->m = 0;
}
void add_edge(Graph *G, int u, int v, int w){
G->edges[G->m].u = u;
G->edges[G->m].v = v;
G->edges[G->m].w = w;
G->m++;
}
void BellmanFord(Graph *G, int s){
int i, it,k;
for(i=1; i<= G->n; i++){
pi[i] = INFINITY;
}
pi[s] = 0;
p[s] = -1;
for(it = 1; it < G->n; it++){
for(k=0; k< G->m; k++){
int u = G->edges[k].u;
int v = G->edges[k].v;
int w = G->edges[k].w;
if(pi[u] + w < pi[v]){
pi[v] = pi[u] + w;
p[v] = u;
}
```

```
}
}
}
int main(){
// freopen("dt.txt", "r", stdin);
Graph G;
int n, m, u, v, e, w;
int i;
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);
}
BellmanFord(&G, 1);
for(i=1; i<=G.n;i++)
printf("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);
}
```
vô hướng
```
#include <stdio.h>
#define MAX_VERTICES 50
#define MAX_EDGES 50
#define NO_EDGE -1
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] = NO_EDGE;
}
void add_edge(Graph* G, int x, int y, int w) {
G->A[x][y] = w;
```

```
G->A[y][x] = w;
}
#define INFINITY 9999999
int mark[MAX_VERTICES];
int pi[MAX_VERTICES];
int p[MAX_VERTICES];
void Dijkstra(Graph* G, int s) {
int i, j, it;
for (i = 1; i <= G->n; i++) {
pi[i] = INFINITY;
mark[i] = 0;
}
pi[s] = 0;
p[s] = -1;
for (it = 1; it < G->n; it++) {
int min_pi = INFINITY;
for (j = 1; j <= G->n; j++)
if (mark[j] == 0 && pi[j] < min_pi) {
min_pi = pi[j];
i = j;
}
mark[i] = 1;
for (j = 1; j <= G->n; j++)
if (G->A[i][j] != NO_EDGE && mark[j] == 0) {
if (pi[i] + G->A[i][j] < pi[j]) {
pi[j] = pi[i] + G->A[i][j];
p[j] = i;
}
}
}
}
int main() {
// freopen("dt.txt", "r", stdin);
Graph G;
int n, m, u, v, e, w;
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);
}
Dijkstra(&G, 1);
for(int i=1; i<=G.n;i++)
printf("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);
}
```

Moore – dijkstra(Chiều dài)

****có hướng****

```
#include <stdio.h>
#define NO_EDGE -1
typedef struct {
        int n, m;
        int W[100][100];
}Graph;

void init_graph(Graph* pG, int n) {
        pG->n = n;
        pG->m = 0;
        for (int u = 1; u <= n; u++) {
                for (int v = 1; v <= n; v++) {
                        pG->W[u][v] = NO_EDGE;
                }
        }
}

void add_edge(Graph* pG, int u, int v, int w) {
        pG->W[u][v] = w;
        pG->m++;
}

#define oo 999999
int mark[100], p[100], pi[100];

void Dijkstra(Graph* pG, int s) {
        for (int u = 1; u <= pG->n; u++) {
```

```
            mark[u] = 0;
            pi[u] = oo;
        }
        pi[s] = 0;
        p[s] = -1;
        int u;
        for (int i = 1; i < pG->n; i++) {
            int minpi = oo;
            for (int v = 1; v <= pG->n; v++) {
                if (!mark[v] && pi[v] < minpi) {
                    minpi = pi[v];
                    u = v;
                }
            }
            mark[u] = 1;
            for (int v = 1; v <= pG->n; v++) {
                if (!mark[v] && pG->W[u][v] != NO_EDGE) {
                    if (pi[u] + pG->W[u][v] < pi[v]) {
                        pi[v] = pi[u] + pG->W[u][v];
                        p[v] = u;
                    }
                }
            }
        }
}

int main() {
        int n, m;
        scanf("%d %d", &n, &m);
        Graph G;
        init_graph(&G, n);
        for (int e = 0; e < m; e++) {
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            add_edge(&G, u, v, w);
        }
        Dijkstra(&G, 1);
```

```
	if (pi[n] < oo)
		printf("%d", pi[n]);
	else
		printf("-1");
	return 0;
}
```
vô hướng
```
#include <stdio.h>
#define NO_EDGE -1
typedef struct {
	int n, m;
	int W[100][100];
}Graph;

void init_graph(Graph* pG, int n) {
	pG->n = n;
	pG->m = 0;
	for (int u = 1; u <= n; u++) {
		for (int v = 1; v <= n; v++) {
			pG->W[u][v] = NO_EDGE;
		}
	}
}

void add_edge(Graph* pG, int u, int v, int w) {
	pG->W[u][v] = w;
	pG->W[v][u] = w;
	pG->m++;
}

#define oo 999999
int mark[100], p[100], pi[100];

void Dijkstra(Graph* pG, int s) {
	for (int u = 1; u <= pG->n; u++) {
		mark[u] = 0;
		pi[u] = oo;
```

```
        }
        pi[s] = 0;
        p[s] = -1;
        int u;
        for (int i = 1; i < pG->n; i++) {
                int minpi = oo;
                for (int v = 1; v <= pG->n; v++) {
                        if (!mark[v] && pi[v] < minpi) {
                                minpi = pi[v];
                                u = v;
                        }
                }
                mark[u] = 1;
                for (int v = 1; v <= pG->n; v++) {
                        if (!mark[v] && pG->W[u][v] != NO_EDGE) {
                                if (pi[u] + pG->W[u][v] < pi[v]) {
                                        pi[v] = pi[u] + pG->W[u][v];
                                        p[v] = u;
                                }
                        }
                }
        }
}

int main() {
        int n, m;
        scanf("%d %d", &n, &m);
        Graph G;
        init_graph(&G, n);
        for (int e = 0; e < m; e++) {
                int u, v, w;
                scanf("%d %d %d", &u, &v, &w);
                add_edge(&G, u, v, w);
        }
        Dijkstra(&G, 1);
        if (pi[n] < oo)
                printf("%d", pi[n]);
```

```
	else
		printf("-1");
	return 0;
}
```

Moore – dijkstra(đường đi)

****có hướng****

```
#include <stdio.h>
#define NO_EDGE -1
typedef struct {
	int n, m;
	int W[100][100];
}Graph;

void init_graph(Graph* pG, int n) {
	pG->n = n;
	pG->m = 0;
	for (int u = 1; u <= n; u++) {
		for (int v = 1; v <= n; v++) {
			pG->W[u][v] = NO_EDGE;
		}
	}
}

void add_edge(Graph* pG, int u, int v, int w) {
	pG->W[u][v] = w;
	//pG->W[v][u] = w;
	pG->m++;
}

#define oo 999999
int mark[100], p[100], pi[100];

void Dijkstra(Graph* pG, int s) {
	for (int u = 1; u <= pG->n; u++) {
		mark[u] = 0;
		pi[u] = oo;
	}
```

```
	pi[s] = 0;
	p[s] = -1;
	int u;
	for (int i = 1; i < pG->n; i++) {
		int minpi = oo;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pi[v] < minpi) {
				minpi = pi[v];
				u = v;
			}
		}
		mark[u] = 1;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pG->W[u][v] != NO_EDGE) {
				if (pi[u] + pG->W[u][v] < pi[v]) {
					pi[v] = pi[u] + pG->W[u][v];
					p[v] = u;
				}
			}
		}
	}
}

int main() {
	int n, m, s, t;
	scanf("%d %d", &n, &m);
	Graph G;
	init_graph(&G, n);
	for (int e = 0; e < m; e++) {
		int u, v, w;
		scanf("%d %d %d", &u, &v, &w);
		add_edge(&G, u, v, w);
	}
	scanf("%d %d", &s, &t);
	Dijkstra(&G, s);
	int path[100];
	int cnt = 0;
```

```
	while (t != s) {
		path[cnt] = t;
		t = p[t];
		cnt++;
	}
	path[cnt] = s;
	for (int i = cnt; i > 0; i--)
		printf("%d -> ", path[i]);
	printf("%d", path[0]);
	return 0;
}
**vô hướng**
#include <stdio.h>

#define NO_EDGE -1

typedef struct {
	int n, m;
	int W[100][100];
}Graph;

void init_graph(Graph* pG, int n) {
	pG->n = n;
	pG->m = 0;
	for (int u = 1; u <= n; u++) {
		for (int v = 1; v <= n; v++) {
			pG->W[u][v] = NO_EDGE;
		}
	}
}

void add_edge(Graph* pG, int u, int v, int w) {
	pG->W[u][v] = w;
	pG->W[v][u] = w;
	pG->m++;
}
```

```
#define oo 999999
int mark[100], p[100], pi[100];

void Dijkstra(Graph* pG, int s) {
	for (int u = 1; u <= pG->n; u++) {
		mark[u] = 0;
		pi[u] = oo;
	}
	pi[s] = 0;
	p[s] = -1;
	int u;
	for (int i = 1; i < pG->n; i++) {
		int minpi = oo;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pi[v] < minpi) {
				minpi = pi[v];
				u = v;
			}
		}
		mark[u] = 1;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pG->W[u][v] != NO_EDGE) {
				if (pi[u] + pG->W[u][v] < pi[v]) {
					pi[v] = pi[u] + pG->W[u][v];
					p[v] = u;
				}
			}
		}
	}
}

int main() {
	int n, m, s, t;
	scanf("%d %d", &n, &m);
	Graph G;
	init_graph(&G, n);
	for (int e = 0; e < m; e++) {
```

```
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            add_edge(&G, u, v, w);
        }
        scanf("%d %d", &s, &t);
        Dijkstra(&G, s);
        int path[100];
        int cnt = 0;
        while (t != s) {
            path[cnt] = t;
            t = p[t];
            cnt++;
        }
        path[cnt] = s;
        for (int i = cnt; i > 0; i--)
            printf("%d -> ", path[i]);
        printf("%d", path[0]);
        return 0;
}
```

Viết chương trình đọc một đơn đồ thị có hướng, có trọng số không âm **từ bàn phím và in ra chiều dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh n.**

```
#include <stdio.h>
#define NO_EDGE -1
typedef struct {
    int n, m;
    int W[100][100];
}Graph;

void init_graph(Graph* pG, int n) {
    pG->n = n;
    pG->m = 0;
    for (int u = 1; u <= n; u++) {
        for (int v = 1; v <= n; v++) {
            pG->W[u][v] = NO_EDGE;
        }
    }
}
```

```
void add_edge(Graph* pG, int u, int v, int w) {
	pG->W[u][v] = w;
	pG->m++;
}

#define oo 999999
int mark[100], p[100], pi[100];

void Dijkstra(Graph* pG, int s) {
	for (int u = 1; u <= pG->n; u++) {
		mark[u] = 0;
		pi[u] = oo;
	}
	pi[s] = 0;
	p[s] = -1;
	int u;
	for (int i = 1; i < pG->n; i++) {
		int minpi = oo;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pi[v] < minpi) {
				minpi = pi[v];
				u = v;
			}
		}
		mark[u] = 1;
		for (int v = 1; v <= pG->n; v++) {
			if (!mark[v] && pG->W[u][v] != NO_EDGE) {
				if (pi[u] + pG->W[u][v] < pi[v]) {
					pi[v] = pi[u] + pG->W[u][v];
					p[v] = u;
				}
			}
		}
	}
}
```

```
int main() {
	int n, m;
	scanf("%d %d", &n, &m);
	Graph G;
	init_graph(&G, n);
	for (int e = 0; e < m; e++) {
		int u, v, w;
		scanf("%d %d %d", &u, &v, &w);
		add_edge(&G, u, v, w);
	}
	Dijkstra(&G, 1);
	if (pi[n] < oo)
		printf("%d", pi[n]);
	else
		printf("-1");
	return 0;
}
```

Cho đồ thị vô hướng G = <V, E> có n đỉnh và m cung (n < 100, m < 500). Mỗi cung được gán một trọng số w (0 < w <= 100).

Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến n.

```
#include <stdio.h>

#define MAX_VERTICES 50

#define MAX_EDGES 50

#define NO_EDGE -1

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] = NO_EDGE;
}
void add_edge(Graph* G, int x, int y, int w) {
G->A[x][y] = w;
G->A[y][x] = w;
}
#define INFINITY 9999999
int mark[MAX_VERTICES];
int pi[MAX_VERTICES];
int p[MAX_VERTICES];
void Dijkstra(Graph* G, int s) {
int i, j, it;
for (i = 1; i <= G->n; i++) {
pi[i] = INFINITY;
mark[i] = 0;
}
pi[s] = 0;
p[s] = -1;
for (it = 1; it < G->n; it++) {
```

```
int min_pi = INFINITY;
for (j = 1; j <= G->n; j++)
if (mark[j] == 0 && pi[j] < min_pi) {
min_pi = pi[j];
i = j;
}
mark[i] = 1;
for (j = 1; j <= G->n; j++)
if (G->A[i][j] != NO_EDGE && mark[j] == 0) {
if (pi[i] + G->A[i][j] < pi[j]) {
pi[j] = pi[i] + G->A[i][j];
p[j] = i;
}
}
}
}
int main() {
// freopen("dt.txt", "r", stdin);
Graph G;
int n, m, u, v, e, w;
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);
}
Dijkstra(&G, 1);
if(pi[n]>0){
printf("%d", pi[n]);
}
else printf("-1");
return 0;
}
```

Topo

Thứ tự topo theo chiều rộng

Viết chương trình đọc vào một *đồ thị có hướng không chu trình* G. Áp dụng thuật toán sắp xếp topo theo phương pháp duyệt theo chiều rộng để sắp xếp các đỉnh của G. In các đỉnh ra màn hình theo thứ tự topo.

```
#include<stdio.h>
#define MAX_N 100
typedef struct{
	int n,m;
	int A[MAX_N][MAX_N];
}Graph;
void init_graph(Graph *pG,int n){
	pG->n=n;
	pG->m=0;
	int u,v;
	for(u=1;u<=n;u++){
		for(v=1;v<=n;v++)
			pG->A[u][v]=0;
	}
}
void add_edge(Graph *pG,int u,int v) {
```

```
    pG->A[u][v]+=1;
}
#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++;
}

ElementType element_at(List* L, int i) {
        return L->data[i-1];
}

int count_list(List* L) {
        return L->size;
}

int empty_list(List* L) {
        return L->size == 0;
}
typedef struct{
        int data[MAX_N];
        int front,rear;
}Queue;
void make_null_queue(Queue *pQ){
        pQ->front=0;
        pQ->rear=-1;
}
```

```
void enqueue(Queue *pQ,int u){
	pQ->rear++;
	pQ->data[pQ->rear]=u;
}
int empty_queue(Queue *pQ){
	return pQ->front > pQ->rear;
}
int front(Queue *pQ){
	return pQ->data[pQ->front];
}
void dequeue(Queue *pQ){
	pQ->front++;
}
void topo_sort(Graph *pG,List *pL){
	int d[MAX_N];
	int u,x;
	for(u=1;u<=pG->n;u++){
		d[u]=0;
		for(x=1;x<=pG->n;x++)
			if(pG->A[x][u]!=0)
				d[u]++;
	}
	Queue Q;
	make_null_queue(&Q);
	for(u=1;u<=pG->n;u++)
		if(d[u]==0)
			enqueue(&Q,u);
	make_null(pL);
	while(!empty_queue(&Q)){
		int u=front(&Q);
		dequeue(&Q);
		push_back(pL,u);
		int v;
		for(v=1;v<=pG->n;v++)
			if(pG->A[u][v]!=0){
				d[v]--;
				if(d[v]==0)
```

```
                    enqueue(&Q,v);
                }
        }
}
int main(){
        Graph G;
        int n,m,u,e,v,i;
        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);
        }
        List L;
        topo_sort(&G,&L);
        for(i=1;i<=L.size;i++){
                printf("%d ",element_at(&L,i));
        }
        return 0;
}
```

Thứ tự topo theo chiều rộng hoặc sâu

Viết chương trình đọc vào một *đồ thị có hướng không chu trình* G, in các đỉnh của G ra màn hình theo thứ tự topo. Nếu có nhiều thứ tự topo, in một thứ tự bất kỳ.

```
#include<stdio.h>
#define MAX_N 100
typedef struct{
        int n,m;
        int A[MAX_N][MAX_N];
}Graph;
void init_graph(Graph *pG,int n){
        pG->n=n;
        pG->m=0;
        int u,v;
        for(u=1;u<=n;u++){
                for(v=1;v<=n;v++)
                        pG->A[u][v]=0;
        }
```

```
}
void add_edge(Graph *pG,int u,int v) {
   pG->A[u][v]+=1;
}
#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++;
}

ElementType element_at(List* L, int i) {
        return L->data[i-1];
}

int count_list(List* L) {
        return L->size;
}

int empty_list(List* L) {
        return L->size == 0;
}
typedef struct{
        int data[MAX_N];
        int front,rear;
}Queue;
void make_null_queue(Queue *pQ){
        pQ->front=0;
```

```
	pQ->rear=-1;
}
void enqueue(Queue *pQ,int u){
	pQ->rear++;
	pQ->data[pQ->rear]=u;
}
int empty_queue(Queue *pQ){
	return pQ->front > pQ->rear;
}
int front(Queue *pQ){
	return pQ->data[pQ->front];
}
void dequeue(Queue *pQ){
	pQ->front++;
}
int mark[MAX_N];
void DFS(Graph *pG,int u,List *pL){
	mark[u]=1;
	int v;
	for(v=1;v<=pG->n;v++){
		if(pG->A[u][v]>0 && mark[v]==0){
			DFS(pG,v,pL);
		}
	}
	push_back(pL,u);
}
void topo_sort(Graph *pG,List *pL){
	int u;
	for(u=1;u<=pG->n;u++){
		mark[u]=0;
	}
	make_null(pL);
	for(u=1;u<=pG->n;u++){
		if(mark[u]==0)
			DFS(pG,u,pL);
	}
}
```

```
int main(){
	Graph G;
	int n,m,u,e,v,i;
	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);
	}
	List L;
	topo_sort(&G,&L);
	for(i=L.size;i>=1;i--){
		printf("%d ",element_at(&L,i));
	}
	return 0;
}
```

Viết chương trình đọc vào một *đơn đồ thị có hướng không chu trình*, xếp hạng các đỉnh và in hạng của các đỉnh ra màn hình.

```
#include<stdio.h>
#define MAX_N 100
typedef struct{
	int n,m;
	int A[MAX_N][MAX_N];
}Graph;
void init_graph(Graph *pG,int n){
	pG->n=n;
	pG->m=0;
	int u,v;
	for(u=1;u<=n;u++){
		for(v=1;v<=n;v++)
			pG->A[u][v]=0;
	}
}
void add_edge(Graph *pG,int u,int v) {
   pG->A[u][v]+=1;
}
#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++;
}

ElementType element_at(List* L, int i) {
	return L->data[i-1];
}

int count_list(List* L) {
	return L->size;
}

int empty_list(List* L) {
	return L->size == 0;
}
void copy_list(List *pS1,List *pS2){
	int i,x;
	make_null(pS1);
	for(i=1;i<=pS2->size;i++){
		x=element_at(pS2,i);
		push_back(pS1,x);
	}
}
int r[MAX_N];
void rank(Graph *pG){
	int d[MAX_N];
```

```
	int u,v;
	for(u=1;u<=pG->n;u++){
		d[u]=0;
		for(v=1;v<=pG->n;v++){
			if(pG->A[v][u]!=0)
				d[u]++;
		}
	}
	List S1,S2;
	make_null(&S1);
	for(u=1;u<=pG->n;u++)
		if(d[u]==0)
			push_back(&S1,u);
	int i,k=0;
	while(S1.size>0){
		make_null(&S2);
		for(i=1;i<=S1.size;i++){
			int u=element_at(&S1,i);
			r[u]=k;
			for(v=1;v<=pG->n;v++)
				if(pG->A[u][v]!=0){
					d[v]--;
					if(d[v]==0)
						push_back(&S2,v);
				}
		}
		copy_list(&S1,&S2);
		k++;
	}
}
int main(){
	Graph G;
	int n,m,u,v,e;
	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);
        }
        rank(&G);
        for(u=1;u<=n;u++){
            printf("r[%d] = %d\n",u,r[u]);
        }
        return 0;
}
```

Viết chương trình đọc vào một *đa đồ thị có hướng không chu trình*, xếp hạng các đỉnh và in hạng của các đỉnh ra màn hình.

```
#include<stdio.h>
#define MAX_N 100
typedef struct{
        int n,m;
        int A[MAX_N][MAX_N];
}Graph;
void init_graph(Graph *pG,int n){
        pG->n=n;
        pG->m=0;
        int u,v;
        for(u=1;u<=n;u++){
            for(v=1;v<=n;v++)
                pG->A[u][v]=0;
        }
}
void add_edge(Graph *pG,int u,int v) {
   pG->A[u][v]+=1;
}
#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++;
}

ElementType element_at(List* L, int i) {
	return L->data[i-1];
}

int count_list(List* L) {
	return L->size;
}

int empty_list(List* L) {
	return L->size == 0;
}
void copy_list(List *pS1,List *pS2){
	int i,x;
	make_null(pS1);
	for(i=1;i<=pS2->size;i++){
		x=element_at(pS2,i);
		push_back(pS1,x);
	}
}
int r[MAX_N];
void rank(Graph *pG){
	int d[MAX_N];
	int u,v;
	for(u=1;u<=pG->n;u++){
		d[u]=0;
		for(v=1;v<=pG->n;v++){
			if(pG->A[v][u]!=0)
				d[u]++;
		}
	}
```

```
    List S1,S2;
    make_null(&S1);
    for(u=1;u<=pG->n;u++)
        if(d[u]==0)
            push_back(&S1,u);
    int i,k=0;
    while(S1.size>0){
        make_null(&S2);
        for(i=1;i<=S1.size;i++){
            int u=element_at(&S1,i);
            r[u]=k;
            for(v=1;v<=pG->n;v++)
                if(pG->A[u][v]!=0){
                    d[v]--;
                    if(d[v]==0)
                        push_back(&S2,v);
                }
        }
        copy_list(&S1,&S2);
        k++;
    }
}
int main(){
    Graph G;
    int n,m,u,v,e;
    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);
    }
    rank(&G);
    for(u=1;u<=n;u++){
        printf("r[%d] = %d\n",u,r[u]);
    }
    return 0;
}
```