```
//Cau truc danh sach list
typedef struct  {
        int data[Max_Length];
        int size;
}List;
void make_null(List *list){
        list->size=0;
}
//Them phan tu vao cuoi danh sach
void push_back(List *list,int x){
        list->data[list->size] = x;
        list->size++;
}
//Lay mot phan tu (dinh) tai vi tri i
int element_at(List *list,int i){
        return list->data[i-1];
}
void copy_list(List *L1,List *L2){
        int i,x;
        make_null_list(L1);
        for(i=1;i<=L2->size;i++){

        x=element_at(L2,i);

        push_back(L1,x);
        }
}
//Hang doi
typedef struct {
   int data[100];
   int front, rear;
}Queue;

void make_null_queue(Queue *pQ){
   pQ->front = 0;
   pQ->rear= -1;
}

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

int front(Queue *pQ){
   return pQ->data[pQ->front];
}

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

int empty(Queue *pQ){
   return pQ->front > pQ->rear;
}

//NGĂN XẾP
typedef struct {
   int data[100];
   int top_idx;
}Stack;

void make_null_stack(Stack *pS){
        pS->top_idx = -1;
}

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

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

void init_graph(Graph *pG, int n){
pG-> n = n; pG->m = 0;
   for(int i=1; i<=pG->n; i++){
      for(int j=1; j<=pG->n; j++){
         pG->A[i][j] = 0;
      }
   }
}

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

int adjacent(Graph *pG, int u, int v){
   return pG->A[u][v];
}

int deg(Graph *pG, int u){
   int count=0;
   for(int i=1; i<=pG->n; i++){
      if(pG->A[u][i] == 1){
         count++;
      }
   }
   return count;
}

*//GIONG
int n, m, u, v;
   Graph G;
   scanf("%d%d", &n, &m);
   init_graph(&G, n);
   for(int i=1; i<=m; i++){
      scanf("%d%d", &u, &v);
      add_edge(&G, u, v);
```

Duyệt theo chiều rộng từ 1 đỉnh

```
// KHAI BÁO GRAHP A[][]
// KHAI BÁO QUEUE
int mark[100];

void BFS(Graph *pG, int s){
  Queue Q;
  make_null_queue(&Q);

  enqueue(&Q, s);

  while(!empty(&Q)){
    int u = front(&Q);
    dequeue(&Q);
    if(mark[u] !=0) continue;

    printf("%d\n", u);
    mark[u] = 1;

    for(int v=1; v<=pG->n; v++){
      if(adjacent(pG, u, v))
        enqueue(&Q, v);
    }
  }
}

int main(){
//Giong
  for(int i=1; i<=n; i++){
    mark[i] = 0;
  }

  BFS(&G, 1);
  return 0;
}
```

Đồthị có hướng xóa dòng
pG->A[v][u]=1;
từ đỉnh s khai báo biến s đưa vào
BFS(&G, S)

Duyệt theo chiều rộng toàn bộ đồ thi

Thêm dòng:

```
for(int i=1; i<=n; i++){
    if(mark[i] == 0){
      BFS(&G, i);
    }
  }
```

DUYỆT THEO CHIỀU SÂU TỪ 1 ĐỈNH

1 Khai báo Graph
2 Khai báo ngăn xếp

```
int mark[100];

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

int main(){
    //GIONG
    for(j=1;j<=G.n;j++){
        mark[j]=0;
    }
    DFS(&G,1);
    return 0;
}
```

CÁC PHẦN KHÁC GIỐNG DUYỆT THEO CHIỀU RỘNG

KIỂM TRA ĐỒ THỊ VÔ HƯỚNG LIÊN THÔNG

Ý tưởng: Nếu duyệt toàn bộ đồ thị mà còn đỉnh có mark[] = 0 nghĩa là đồ thi không có tính liên thông

1 Khai báo Graph
2 Khai báo Queue
3 Khai báo hàm connect

```
int connect(Graph *pG){
    int i;
    for(i=1;i<=pG->n;i++){
        if(mark[i]==0)
        return 0;
    }
    return 1;
}
```

Int mark[100];
4. Khai báo hàm BFS chú ý bỏ printf trong hàm.
5.

```
int main(){
    //Giong
    for(j=1;j<=G.n;j++){
        mark[j]=0;
    }
    BFS(&G,1);
    if(connect(&G)){

    printf("CONNECTED");
    } else
printf("DISCONNECTED");
    return 0;
}
```

BỘ PHẬN LIÊN THÔNG

Ý tưởng duyệt theo chiều rộng kiểm tra sao mỗi lần duyệt nếu còn đỉnh có mark[] = 0 thì đó là một bộ phận liên thông

1. **Khai báo Graph**
2. **Khai báo Queue**
3. **BFS chú ý bỏ printf("%d", u);**
4. **Mark[100]**

```
int main(){
    //GIONG
    for(j=1;j<=G.n;j++){
        mark[j]=0;
    }
    for(j=1;j<=G.n;j++){
        if(mark[j]==0){

    BFS(&G,j);

    bplt++;
        }
    }
    printf("BPLT: %d\n",bplt);
    return 0;
}
```

ĐẾM SỐ ĐỈNH CỦA BỘ PHẬN LIÊN THÔNG

```
//Các bước trên giống tìm số bplt
int mark[100];
int nb_u;
void BFS(Graph *pG, int s){
    // xem thuật toán BFS
    if(mark[u] !=0) continue;
        nb_u++;
    //xem thuật toán BFS
}

int main(){
    //GIONG
    Int max = 0;
    for(int i=1; i<=n; i++){
        mark[i] = 0;
    }

    nb_u = 0;

    BFS(&G, 1);
    printf("%d", nb_u);
    return 0;
}
```

TÌM BỘ PHẬN LIÊN THÔNG NHIỀU ĐỈNH NHẤT

```
//Các bước trên giống tìm số đỉnh bplt
int main(){
    //gIONG

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

    for(int i=1;i<=n; i++){
      if(mark[i]==0){
      nb_u = 0;
      BFS(&G, i);
    }
     if(nb_u>max){
       max = nb_u;
        }
    }
    printf("%d", max);
    return 0;
}
```

ỨNG DỤNG LIÊN THÔNG

Qua đảo, Tôn Ngộ Không → Kiểm tra đồ thị vô hướng liên thông

KIỂM TRA ĐỒ THỊ CHỨ CHU TRÌNH

Ý tưởng:
Duyệt theo chiều sâu
Chưa duyệt : WHITE
Đang duyệt: GRAY
Đã duyệt: BLACK

- Khởi tạo tất cả các đỉnh đều là màu trắng
- Hàm đệ qui DFS gồm các bước:

+ Nếu v có màu trắng -> gọi đệ qui duyệt v
+ Nếu đỉnh kề v nào đó có màu xám -> có chu trình -> has_circle = 1.
+ Nếu đỉnh v màu đen duyệt xong r bỏ qua

~~KIỂM TRA ĐƠN ĐỒ THỊ CÓ HƯỚNG CÓ CHU TRÌNH~~

- **#define WHITE 0**
- **#define GRAY 1**
- **#define BLACK 2**
- Khởi tạo graph
- Khai báo stack
- Int color[100]
- Int has_circle;

```
void DFS(Graph *pG,int u){
        int v;
        color[u]=2;
//      printf("%d\n",u);
//      mark[u]=1;
        for(v=1;v<=pG->n;v++)

        if(adjacent(pG,u,v)){

        if(color[v] == 0)

        DFS(pG,v);
                        else
if(color[v] == 2)

        has_circle = 1;
                }
        color[u]=1;

}
int main(){
        //GIỐNG
        for(j=1;j<=G.n;j++){
                mark[j]=0;
                color[j]=0;
        }
        has_circle=0;
        for(j=1;j<=G.n;j++){
                if(color[u] == 0)

        DFS(&G,j);
        }
        if(has_circle==1)
printf("CIRCLED");
        else printf("NO CIRCLE");
        return 0;
}
```

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

```
int color[100];
int has_circle;
void DFS(Graph *pG, int u, int p){
        int v;
        color[u] = 1;
        for(v=1; v<=pG->n; v++){
                if(adjacent(pG, u,
v)){

        if(v==p) continue;

        if(color[v] == 0){

        DFS(pG, v, u);
                                }else
if(color[v] == 1){

        has_circle = 1;
                                }
                }
        }
        color[u] = 2;
}

int main(){
        int i, j, u, v, n, m;
        Graph G;
        scanf("%d%d", &n, &m);
        init_graph(&G, n);
        for(i=1; i<=n; i++){
                scanf("%d%d",
&u, &v);
                add_edge(&G, u,
v);
        }

        for(j =1; j<=n; j++){
          if(color[j]==0){
            DFS(&G, j, -1);
          }
        }

        if(has_circle == 1){
          printf("CIRCLED");
        }else{
          printf("NO CIRCLE");
        }
        return 0;
}
```

KIỂM TRA ĐỒ THỊ PHÂN ĐÔI

Ý tưởng:
Mỗi đỉnh sẽ có 1 trong 3 trạng thái tương ứng với 3 màu:

- Chưa có màu: NO_COLOR.
- Được tô màu: BLUE
- Được tô màu đỏ: RED

Thuật toán duyệt đồ thị đệ qui kết hợp tô màu colorize(u, c) gồm các bước sau

- Tô màu c cho u
- Xát các đỉnh kề v của u, có 3 trường hợp:
  + Nếu v chưa có màu -> gọi đệ quy tô màu ngược lại với c cho nó
  + Nếu v đã có màu và giống với u -> đụng độ không tô được
  + Nếu v đã có màu và khác màu với u -> bỏ qua

Tìm màu ngược lại màu của c.
Gọi S = BLUE + RED
Để tìm màu ngược lại của c, ta lấy S trừ cho c cụ thể:

- Ngược lại của BLUE là S – BLUE = RED
- Ngược lại màu của RED là S – RED = BLUE

Kiểm tra đồ thị vô hướng phân đôi

1. #define NO_COLOR 0
2. #define BLUE 1
3. #define RED 2
4. Khai báo Graph

```
int color[100];
int conflict;
void colorize(Graph *pG, int u,
int c){
  color[u] = c;
  for(int v=1; v<=pG->n; v++){
    if(adjacent(pG, u, v)){
      if(color[v] == NO_COLOR){
        colorize(pG, v, 3-c);
      }else if(color[v] ==
color[u]){
        conflict = 1;
      }
    }
  }
}
int main(){
//GIỐNG
  for(int j=1; j<=n; j++){
    if(color[j] == NO_COLOR){
      colorize(&G, j, NO_COLOR);
    }
  }
  if(conflict==1){
    printf("NO");
  }else{
    printf("YES");
  }
  return 0;
}
```

BÀI TOÁN PHÂN CHIA ĐỘI BÓNG

```
//PHẦN TRÊN GIỐNG
int main(){
  Graph G;
  int n, m, u, v, c[100], r=0;
```

```
  scanf("%d%d", &n, &m);
  init_graph(&G, n);
  for(int i=1; i<=m; i++){
    scanf("%d%d", &u, &v);
    add_edge(&G, u, v);
  }

  for(int i=1; i<=n; i++){
    if(color[i] == NO_COLOR){
      colorize(&G, i, BLUE);
    }
  }

  if(conflict == 1){
    printf("IMPOSSIBLE");
  }else{
    for(int i=1; i<=n; i++){
      if(color[i] == BLUE){
        printf("%d ", i);
      }else{
        c[r] = i;
        r++;
      }
    }
  }
  printf("\n");
  for(int i=0; i<r; i++){
    printf("%d ", c[i]);
  }
}
```

TÍNH LIÊN THÔNG MẠNH ÁP DỤNG THUẬT TOÁN TARJAN

In ra num và min_num

```
//Khởi tạo Graph
//Khai báo stack
Stack S;
int num[100], min_num[100];
int k;
int on_stack[100];
void SCC(Graph *pG, int u){
  num[u] = k;
  min_num[u] = k;
  k++;
  push(&S, u);
  on_stack[u] = 1;

  for(int v=1; v<=pG->n; v++){
    if(adjacent(pG, u, v)){
      if(num[v] < 0){
        SCC(pG, v);
        min_num[u] =
min(min_num[u], min_num[v]);
      }else if(on_stack[v]){
        min_num[u] =
min(min_num[u], num[v]);
      }
    }
  }
  if(num[u] == min_num[u]){
    int w;
    do{
      w = top(&S);
      pop(&S);
      on_stack[w] = 0;
    }while(w!=u);
  }
}


int main(){
      //GIỐNG
      for(j=1;j<=G.n;j++){
            num[j]= -1;
      }
      k=1;
      make_null_stack(&S);
      for(j=1;j<=G.n;j++){
            if(num[j] == -1)

      SCC(&G,j);
      }
      for(i=1;i<=n;i++)
      printf("%d %d\n",num[i],m
in_num[i]);
      return 0;
}
```

Viết chương trình đọc vào một đồ thị có hướng và kiểm tra xem nó có liên thông mạnh không.

```
//Các bước trên giống in num và
min_num
//Khai báo thêm biến toàn cục
count = 0;
Void SCC(){
//
if(num[u] == min_num[u]){
    int w;
    count++;
    do{
      w = top(&S);
      pop(&S);
      on_stack[w] = 0;
    }while(w!=0);
  }
}
int main(){
 //GIỐNG IN NUM VÀ MIN_NUM
  if(count==1)
printf("STRONG CONNECTED");
  else printf("DISCONNECTED");
  return 0;
}
```

Viết chương trình đọc vào một đồ thị có hướng và đếm số bộ phận liên thông mạnh của nó.

```
//Các bước trên giống kiểm tra bộ
phận liên thông mạnh
int num[100], min_num[100];
int k;
Stack S;
int count;
int on_stack[100];
int dem = 0, kq = 0;
void SCC(Graph *pG, int u){
  //GIỐNG BÀI TRÊN

  if(min_num[u] == num[u]){
    dem = 0;
    count++;
    int w;
    do{
      dem++;
      w = top(&S);
      pop(&S);
      on_stack[w] = 0;
    }while(w != u);
    if(dem>kq) kq = dem;
  }
}

int main(){
//giống bài trên
  k=1;
  count=0;
  make_null_stack(&S);
  for(int i=1; i<=n; i++){
    if(num[i] == -1){
      SCC(&G, i);
    }
  }
  printf("%d", count);
  return 0;
}
```

Viết chương trình đọc vào một đồ thị có hướng và tìm bộ phận liên thông mạnh có nhiều đỉnh nhất.

```
//Giong bài trên
int num[100], min_num[100];
int k;
Stack S;
int on_stack[100];
int dem = 0, kq = 0;
void SCC(Graph *pG, int u){
  num[u] = k;
  min_num[u] = k;
  k++;
  push(&S, u);
  on_stack[u] = 1;

  for(int v=1; v<=pG->n; v++){
    if(adjacent(pG, u, v)){
      if(num[v] < 0){
        SCC(pG, v);
        min_num[u] =
min(min_num[u], min_num[v]);
      }else if(on_stack[v]){
        min_num[u] =
min(min_num[u], num[v]);
      }
    }
  }

  if(min_num[u] == num[u]){
    dem = 0;
    int w;
    do{
      dem++;
      w = top(&S);
      pop(&S);
      on_stack[w] = 0;
    }while(w != u);
```

```
    if(dem>kq) kq = dem;
  }
}

int main(){
  Graph G;
  int n,m, u, v;
  scanf("%d%d", &n, &m);
  init_graph(&G, n);
  for(int i=1; i<=m; i++){
    scanf("%d%d", &u, &v);
    add_edge(&G, u, v);
  }
  for(int i=1; i<=n; i++){
    num[i] = -1;
  }
  k=1;
  make_null_stack(&S);
  for(int i=1; i<=n; i++){
    if(num[i] == -1){
      SCC(&G, i);
    }
  }
  printf("%d", kq);
  return 0;
}
```

Come and Go (nguồn: UVA Online Judge, Problem 11838)

```
int num[100], min_num[100];
int k=0;
Stack S;
int count=0;
int on_stack[100];
void SCC(Graph *pG,int u){
  //GIONG BAI TREN
        if(num[u] == min_num[u]){
                count++;
                int w;
                do{

        w=top(&S);

        pop(&S);

        on_stack[w]=0;
                }while(w != u);
        }

}

int main(){
        int i,j,u,v,n,m,p;
        // GIỐNG
        for(j=1;j<=G.n;j++){
                num[j]= -1;
        }
        k=1;
        make_null_stack(&S);
        for(j=1;j<=G.n;j++){
                if(num[j] == -1)

        SCC(&G,j);
        }
        if(count==1)
printf("OKIE");
        else printf("NO");
        return 0;
}
```

Trust group (nguồn: UVA Online Judge, Problem 11709)

```
int num[100], min_num[100];
int k;
Stack S;
int count;
int on_stack[100];
int dem = 0, kq = 0;
void SCC(Graph *pG, int u){
//GIONG BAI TREN
  if(min_num[u] == num[u]){
    dem = 0;
    count++;
    int w;
    do{
      dem++;
      w = top(&S);
      pop(&S);
      on_stack[w] = 0;
    }while(w != u);
    if(dem>kq) kq = dem;
  }
}

int main(){
//GIONG
  for(int i=1; i<=n; i++){
    num[i] = -1;
  }
  k=1;
  count=0;
  make_null_stack(&S);
  for(int i=1; i<=n; i++){
    if(num[i] == -1){
      SCC(&G, i);
    }
  }
  printf("%d", count);
  return 0;
}
```

THUẬT TOÁN MOORE-DJKTRA

Viết chương trình đọc một đơn đồ thị có hướng, có trọng số không âm từ bàn phím. Cài đặt thuật toán Moore – Dijkstra để tìm (các) đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại. In các thông tin pi[u] và p[u] của các đỉnh ra màn hình.

```
#include <stdio.h>

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 i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                pG->W[i][j] = -1;
                }
        }
}

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

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

void MooreDijkstra(Graph *pG, int s){
        int u, v, it;
        for(u=1; u<=pG->n; u++){
                pi[u] =
99999999;
                mark[u] = 0;
        }

        p[s] = -1;
        pi[s] = 0;

        for(it=1; it<=pG->n; it++){
        int j, min_pi = 9999999;
        for(j=1; j<=pG->n; j++){

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

        min_pi = pi[j];

        u = j;
                        }
                }
        mark[u] = 1;
        for(v=1; v<=pG->n; v++){
                if(pG-
>W[u][v]!=-1 && mark[v] == 0){

        if(pi[u] + pG->W[u][v] <
pi[v]){

                pi[v] = pi[u]+pG-
>W[u][v];

                p[v] = u;

        }
                        }
                }
        }
}

int main(){
```

```
	Graph G;
	int n, m, u, v, s;
	scanf("%d%d", &n, &m);
	init_graph(&G, n);
	for(int i=1; i<=m; i++){

	scanf("%d%d%d", &u, &v,
&s);
		add_edge(&G, u,
v, s);
	}
	MooreDijkstra(&G, 1);
	for(int i=1; i<=n; i++){
		printf("pi[%d]
= %d, p[%d] = %d\n", i, pi[i], i, p[i]);
	}
}
```

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.

```
//PHẦN TRÊN GIỐNG BÀI TRƯỚC
int main(){
	Graph G;
	int n, m, u, v, s;
	scanf("%d%d", &n, &m);
	init_graph(&G, n);
	for(int i=1; i<=m; i++){

	scanf("%d%d%d", &u, &v,
&s);
		add_edge(&G, u,
v, s);
	}
	MooreDijkstra(&G, 1);
	if(pi[n] == 99999){
		printf("-1");
	}else{

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

Viết chương trình đọc một đơn đồ thị có hướng, có trọng số không âm từ bàn phím, tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t (s và t cũng được đọc từ bàn phím).

```
#include <stdio.h>
//PHẦN TRÊN GIỐNG BÀI TRƯỚC
MooreDijkstra()
void Path(Graph *G,int n){
	int path[MAXN];
	int k=0;
	int current = n;
	while(current != -1){
		path[k] =
current;
		k++;

	current=p[current];
	}
	int u;
	for(u=k-1;u>=0;u--){
		printf("%d
",path[u]);
		if(path[u]!=n)
printf("-> ");
	}
}



int main(){
	Graph G;
	int n,m,u,v,w,i,t,h;

	scanf("%d%d",&n,&m);
	init_graph(&G,n);
	for(i=1;i<=m;i++){

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


	add_edge(&G,u,v,w);
	}
	scanf("%d%d",&t,&h);
	MooreDijkstra(&G,t);
//	if(pi[n]==99999) printf("-
1");
//  else printf("pi[%d] = %d",n,pi[n]);
	Path(&G,h);
	return 0;
}
```

Viết chương trình đọc một đơn đồ thị vô hướng, có trọng số không âm từ bàn phím, tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t (s và t cũng được đọc từ bàn phím).

SỬA LẠI HÀM ADD_EDGE

MÊ CUNG SỐ

```
#include <stdio.h>

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

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

void add_edge(Graph *pG, int u, int v,
int w){
	pG->W[u][v] = w;
}
int mark[100];
int pi[100];
int p[100];

void MooreDijkstra(Graph *pG, int s){
	int u, v, it;
	for(u=1; u<=pG->n*pG->m;
u++){
		pi[u] = 99999;
		mark[u] = 0;
	}

	pi[s] = 0;

	//4 o xung quanh cua 1 o
tren, duoi, trai, phai
	int di[] = {-1, 1, 0, 0};
	int dj[] = {0, 0, -1, 1};

	for(it=1; it<=pG->n*pG->m;
it++){
		int t, min_pi =
99999;
		for(t=1; t<=pG-
>n*pG->m; t++){

	if(mark[t] == 0 && pi[t] <
min_pi){

	min_pi = pi[t];

	u = t;
			}
		}

		mark[u] = 1;

		int i = (u-1)/pG-
>n;
		int j = (u-1)%pG-
>n;
		int k, ii, jj;
		for(k =0; k<4;
k++){
			ii = i +
di[k];
			jj = j +
dj[k];

	if(ii>=0 && ii<pG->m &&
jj>=0 && jj< pG->n){

	v = ii*pG->n + jj + 1;

	if(pi[u]+pG->W[ii][jj] <
pi[v]){

		pi[v] = pi[u] +
pG->W[ii][jj];

	}
			}
		}
	}
```

```
}

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

	for(v=0;v<n;v++){

	scanf("%d",&w);

	add_edge(&G,u,v,w);
		}
	}
	MooreDijkstra(&G,1);
	printf("%d",pi[n*m]);
	return 0;
}
```

THUẬT TOÁN BELLMAN – FORD

```
#include <stdio.h>

#define MAXM 500
#define MAXN 100
#define oo 999999
#define NO_EDGE -999999

typedef struct {
  int u, v;
  int w;
} Edge;

typedef struct {
  int n, m;
  Edge edges[MAXM];
} Graph;

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

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

  pG->m++;
}


int pi[MAXN];
int p[MAXN];

int BellmanFord(Graph *pG, int s) {
	int u, v, w, it, k;
	for (u = 1; u <= pG->n; u++)
{
		pi[u] = oo;
	}
	pi[s] = 0;
	p[s] = -1; //trước đỉnh s
không có đỉnh nào cả

	// lặp n-1 lần
	for (it = 1; it < pG->n; it++) {
		// Duyệt qua các
cung và cập nhật (nếu thoả)
		for (k = 0; k < pG-
>m; k++) {
			u =
pG->edges[k].u;
			v = pG-
>edges[k].v;
			w =
pG->edges[k].w;

			if
(pi[u] == oo)    //chưa có đường đi từ
s -> u, bỏ qua cung này

continue;

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

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

	p[v] = u;
			}
		}
	}
	//Làm thêm 1 lần nữa để
kiểm tra chu trình âm (nếu cần thiết)
	for (k = 0; k < pG->m; k++) {
		u = pG-
>edges[k].u;
		v = pG-
>edges[k].v;
		w = pG-
>edges[k].w;

		if (pi[u] == oo)
//chưa có đường đi từ s -> u, bỏ qua
cung này
		  continue;

		if (pi[u] + w <
pi[v]) {
			return
1;
		}
	}
	return 0;
}



int main() {
  Graph G;
  int n, m;
  scanf("%d%d", &n, &m);
  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);
  }
  int s;
  scanf("%d", &s);

  if (BellmanFord(&G, s) == 1)
    printf("YES\n");
  else
    printf("NO\n");


  return 0;
}
```

Viết chương trình đọc vào một đơn đồ thị có hướng, có trọng số, áp dụng thuật toán Bellman – Ford kiểm tra xem nó có chứa chu trình âm hay không khi ta tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại.

```
#include <stdio.h>

#define MAXN 100
#define MAXM 100
typedef struct{
	int u,v;
	int w;
}Edge;

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

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

#define oo 99999

int pi[100];
int p[100];
void BellmanFord(Graph *pG,int s){
	int u,v,w,it,k;
	for(u = 1; u <= pG->n; u++){
		pi[u] = oo;
	}

	pi[s] = 0;
	p[s] = -1;

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

```
                for(k = 0; k < pG->m; k++){
                        u = pG->edges[k].u;
                        v = pG->edges[k].v;
                        w = pG->edges[k].w;

        if(pi[u] == oo) continue;

        if(pi[u] + w < pi[v]){
        pi[v] = pi[u] + w;
        p[v] = u;
                        }
                }
        }
}
void check_cycle(Graph *pG){
        int k,u,v,w,nega=0;
        for(k = 0; k < pG->m; k++){
                u = pG->edges[k].u;
                v = pG->edges[k].v;
                w = pG->edges[k].w;
                if(pi[u] + w < pi[v]){
                        nega = 1;
                        break;
                }
        }
        if(nega==1 && pi[u] != oo) printf("YES");
        else printf("NO");
}
int main(){
        Graph G;
        int i,n,m,u,v,w,x;
        scanf("%d%d",&n,&m);
        init_graph(&G,n);
        for(i=0;i<m;i++){
        scanf("%d%d%d",&u,&v,&w);
        add_edge(&G,u,v,w);
        }
        scanf("%d",&x);
        BellmanFord(&G,x);
        check_cycle(&G);
        return 0;
}
```

THỨ TỰ TOPO

Ý tưởng:
Đỉnh cỏ bậc vào bằng 0 sẽ đứng đầu danh sách.

Gỡ bỏ các đỉnh có bậc vào bằng không với các đỉnh kề của nó. Một số đỉnh kề này sẽ có bậc vào bằng không, ta sẽ thêm nó vào danh sách.
Các biến hỗ trợ
d[u] : lưu bậc vào của đĩnh u, mỗi khi gỡ bỏ cung nối u, ta giảm d[u] đi 1.
Q: Lưu các đĩnh sẽ xét
L : danh sách các đỉnh được sắp xếp

THUẬT TOÁN SẮP XẾP TOPO

```
void topo_sort(Graph *pG, List *pL){
  int d[100];
  for(int u=1; u<=pG->n; u++){
    d[u] = 0;
    for(int x=1; x<=pG->n; x++){
      if(pG->A[x][u] != 0){
        d[u]++;
      }
    }
  }

  Queue Q;
  make_null_queue(&Q);
  for(int u=1; u<=pG->n; u++){
    if(d[u] == 0){
      enqueue(&Q, u);
    }
  }

  make_null_list(pL);
  while(!empty_queue(&Q)){
   int u = front(&Q);
   dequeue(&Q);
   push_back(pL, u);
   for(int v=1; v<=pG->n; v++){
     if(pG->A[u][v] != 0){
       d[v]--;
       if(d[v] == 0){
         enqueue(&Q, v);
       }
     }
   }
  }
}
```

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.

```
//Khai báo Graph
//Khai báo Stack
//Khai báo Queue
//Thuật toán Topo
int main(){
  //Giống

  List L;
  make_null_list(&L);
  topo_sort(&G, &L);
  for(int i=1; i<=L.size; i++){
    printf("%d ", element_at(&L, i));
  }
  return 0;
}
```

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.

//GIỐNG BÀI TRÊN

Bài toán Xếp đá

//GIỐNG BÀI TRÊN

THUẬT TOÁN XẾP HẠNG ĐỒ THỊ

Ý tưởng:
-Đỉnh có bậc vào bằng 0 (gốc cũ) sẽ có hạng = 0;
-Gỡ bỏ các cung nối các đỉnh hạng 0 với các đỉnh kề của nó
-Sau khi gở bỏ các cung, các đỉnh có bậc vào bằng 0 (gốc mới) sẽ có hạng bằng 1
-Gỡ bỏ các cung nói các đỉnh có hạng bằng 1 với các đỉnh kề của nó.
Các biến hổ trợ
D[u] : bậc vào của đỉnh u, mỗi khi gỡ bỏ cung nói đến u. ta giảm d[u] đi 1.
R[u] : Lưu hạng của đỉnh u.
S1: danh sách lưu cá đỉnh đang xác định hạng (các gốc cũ)
S2: lưu các đỉn sắp sửa xem xét (có d[u] == 0, các gốc mới).

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.

//GIỐNG BÀI TRÊN

```
int r[MAXN];
void rank(Graph *pG){
  int d[MAXN];

  for(int u=1; u<=pG->n; u++){
    d[u] = 0;
    for(int x=1; x<=pG->n; x++){
      if(pG->A[x][u] != 0){
        d[u]++;
      }
    }
  }

  List S1, S2;
  make_null_list(&S1);
  for(int u=1; u<=pG->n; u++){
    if(d[u] == 0){
      push_back(&S1, u);
    }
  }

  int k = 0;
  while(S1.size > 0){
    make_null_list(&S2);
    for(int i=1; i<=S1.size; i++){
      int u = element_at(&S1, i);
      r[u] = k;

      for(int 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;
  scanf("%d%d", &n, &m);
  init_graph(&G, n);
  for(int i=1; i<=m; i++){
    scanf("%d%d", &u, &v);
    add_edge(&G, u, v);
  }

  rank(&G);
  for(int i=1; i<=n; i++){
    printf("r[%d] = %d\n", i, r[i]);
  }
}
```

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.

```
int r[MAXN];
void rank(Graph *pG){
  int d[MAXN];

  for(int u=1; u<=pG->n; u++){
    d[u] = 0;
    for(int x=1; x<=pG->n; x++){
      if(pG->A[x][u] != 0){
        d[u]++;
      }
    }
  }

  List S1, S2;
  make_null_list(&S1);
  for(int u=1; u<=pG->n; u++){
    if(d[u] == 0){
      push_back(&S1, u);
    }
  }

  int k = 0;
  while(S1.size > 0){
    make_null_list(&S2);
    for(int i=1; i<=S1.size; i++){
      int u = element_at(&S1, i);
      r[u] = k;

      for(int 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;
  scanf("%d%d", &n, &m);
  init_graph(&G, n);
  for(int i=1; i<=m; i++){
    scanf("%d%d", &u, &v);
    add_edge(&G, u, v);
  }

  rank(&G);
  for(int i=1; i<=n; i++){
    printf("r[%d] = %d\n", i, r[i]);
  }
}
```

BÀI TOÁN CHIA KẸO

```
//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);
        }

        rank(&G);
        int tong=0;
        for(int i=1;i<=G.n;i++){

        printf("%d\n",r[i]+1);

        tong=tong+r[i]+1;
        }
        printf("%d",tong);
        return 0;
}
```

BÀI TOÁN PHÂN CHIA ĐỘI BÓNG

```
//GIỐNG BÀI TRÊN
int main(){
  Graph G;
  int n, m, u, v;
  scanf("%d%d", &n, &m);
  init_graph(&G, n);
  for(int i=1; i<=m; i++){
    scanf("%d%d", &u, &v);
    add_edge(&G, u, v);
  }

  rank(&G);
  for(int i=1; i<=n; i++){
    printf("%d ", r[i]+1);
  }

}
```

Cho một dự án gồm n công việc. Mỗi công việc u có một thời gian hoàn thành d[u] và một danh sách các công việc phải hoàn thành trước khi thực hiện u. Hãy tính thời gian

sớm nhất và thời gian trễ nhất để bắt đầu các công việc.

```
//GIỐNG BÀI TRÊN
int d[MAXN]; //lưu thời gian hoàn thành công việc
void topo_sort(Graph *pG, List *pL){
  int d1[100];
  for(int u=1; u<=pG->n; u++){
    d1[u] = 0;
    for(int x=1; x<=pG->n; x++){
      if(pG->A[x][u] != 0){
        d1[u]++;
      }
    }
  }

  Queue Q;
  make_null_queue(&Q);

  for(int u=1; u<=pG->n; u++){
    if(d1[u] == 0){
      enqueue(&Q, u);
    }
  }

  make_null_list(pL);
  while(!empty_queue(&Q)){
    int u = front(&Q);
    dequeue(&Q);
    push_back(pL, u);

    for(int  v=1; v<=pG->n; v++){
      if(pG->A[u][v] != 0){
        d1[v]--;
        if(d1[v]==0){
          enqueue(&Q, v);
        }
      }
    }
  }
}

int max(int a, int b){
  if(a>b) return a;
  return b;
}

int min(int a, int b){
  if(a<b) return a;
  return b;
}

int main(){
  Graph G;
  int n, u, x, v, j;
  scanf("%d", &n);
  init_graph(&G, n+2);
  int alpha = n+1, beta = n+2;
  d[alpha] = 0;
  for(u=1; u<=n; u++){
    scanf("%d", &d[u]); //thời gian hoàn thành công việc u
    do{
      scanf("%d", &x);
      if(x>0){
        add_edge(&G, x, u);
      }
    }while(x > 0);
  }

  for(u=1; u<=n; u++){
    int deg_neg = 0;
    for(x=1; x<=n; x++){
      if(G.A[x][u] > 0){
        deg_neg++;      //deg_neg là bậc vào của u
      }
    }

    if(deg_neg == 0){
      add_edge(&G, alpha , u);
    }
  }

  for(u=1; u<=n; u++){
    int deg_pos = 0;
    for(v=1; v<=n; v++){
      if(G.A[u][v] > 0){
        deg_pos++;
      }
    }
    if(deg_pos==0){
      add_edge(&G, u, beta);
    }
  }

  List L;
  topo_sort(&G, &L);

  int t[100];
  t[alpha] = 0;

  for(j=2; j<=L.size; j++){
    int u = element_at(&L, j);
    t[u] = -99999;
    for(int x=1; x<=G.n; x++){
      if(G.A[x][u] > 0){
        t[u] = max(t[u], t[x]+d[x]);
      }
    }
  }

  int T[100];
  T[beta] = t[beta];
  for(j = L.size -1; j>=1; j--){
    int u = element_at(&L, j);
    T[u] = 99999;
    for(v=1; v<=G.n; v++){
      if(G.A[u][v] > 0){
        T[u] = min(T[u], T[v] - d[u]);
      }
    }
  }
  for(int i=1;i<=n;i++){

    printf("%d %d\n",t[i],T[i]);
    }
    return 0;
}
```

Quản lý dự án phần mềm

```
//GIỐNG BÀI TRÊN
int d[MAXN];
void topo_sort(Graph *G,List *L){
    int d1[MAXN];
    int u,x,v;
    for(u=1;u<=G->n;u++){
        d1[u]=d[u];
    }
    for(u=1;u<=G->n;u++){
        d1[u]=0;
        for(x=1;x<=G->n;x++)
            if(G->A[x][u] != 0)

    d1[u]++;
    }

    Queue Q;
    make_null_queue(&Q);

    for(u=1;u<=G->n;u++)
        if(d1[u]==0)

    enqueue(&Q,u);
    make_null_list(L);

    while(!empty_queue(&Q)){
        u = front(&Q);
        dequeue(&Q);
        push_back(L,u);

        for(v=1;v<=G->n;v++)
            if(G->A[u][v] != 0){

    d1[v]--;

    if(d1[v]==0)

        enqueue(&Q,v);
            }
    }
}

int min(int a,int b){
    if (a<b) return a;
    return b;
}

int max(int a,int b){
    if(a>b) return a;
    return b;
```

```
}
int main(){
	Graph G;
	int n,i,j,u,v,m,x;
	scanf("%d",&n);
	for(v=1;v<=n;v++){

	scanf("%d",&d[v]);
	}
	init_graph(&G,n+2);
	int alpha = n+1, beta = n+2;
	d[alpha] = 0;
	scanf("%d",&m);
	for(i=1;i<=m;i++){

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

	add_edge(&G,u,v);
	}

	for(u = 1 ; u <= n ; u++){
		int deg_neg = 0;
		for(x = 1; x<= n ;
x++)

	if(G.A[x][u] > 0)

	deg_neg++;
		if(deg_neg == 0)

	add_edge(&G,alpha,u);

	}

	for(u = 1 ; u <= n ; u++){
		int deg_pos = 0;
		for(v = 1; v<= n ;
v++)

	if(G.A[u][v] > 0)

	deg_pos++;
		if(deg_pos == 0)

	add_edge(&G,u,beta);
	}
	List L;
	make_null_list(&L);
	topo_sort(&G,&L);
	int t[MAXN];
	t[alpha] = 0;
	for(j=2;j<=L.size;j++){
		u =
element_at(&L,j);
		t[u] = -oo;

	for(x=1;x<=G.n;x++)

	if(G.A[x][u] > 0)

	t[u] = max(t[u],t[x]+d[x]);
	}

	int T[MAXN];
		T[beta] = t[beta];

	for(j=L.size-1;j>=1;j--){
		u =
element_at(&L,j);
		T[u] = oo;

	for(v=1;v<=G.n;v++)

	if(G.A[u][v] > 0)

	T[u] = min(T[u],T[v] - d[u]);
	}
	printf("%d",t[n]+d[n]);
	return 0;
}
```

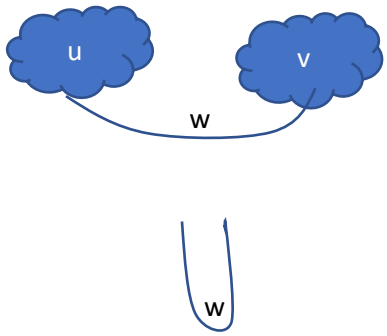
CÂY KHUNG NHỎ NHẤT VÀ LUỒNG CỰC ĐẠI TRÊN MẠNG

Thuật toán Kruskal

-Sắp xếp các cây thei thứ tự trọng số tăng dần

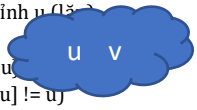
-khởi tạo cây T gồm các đỉnh G và không chứ cung nào

-Thêm các cung e vào cây T mà không tạo chu trình



Viết chương trình đọc đồ thị vô hướng liên thông và tìm cây khung có trọng số nhỏ nhất bằng thuật toán Kruskal.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct{
   int u, v;
   int w;
}Edge;

typedef struct{
   int n, m;
   Edge edges[500];
}Graph;

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

int min(int u, int v){
   return u<v?u:v;
}

int max(int u, int v){
   return u>v?u:v;
}

void add_edge(Graph *pG, int u, int v,
int w){
   pG->edges[pG->m].u = min(u, v);
   pG->edges[pG->m].v = max(u, v);
   pG->edges[pG->m].w = w;

   pG->m++;
}

int parent[100];

// int findRoot(int u){
//    if(parent[u] == u)
//       return u;
//    return findRoot(parent[u]);
// }

//Tìm gốc của đỉnh u (lặ
int findRoot(int u
   while(parent[u] != u)
      u = parent[u];
   return u;
}

int cmpfunc(const void *a, const void
*b) {
   return ((const Edge*)a)->w - ((const
Edge*)b)->w;
}

//Thuật toán Kruskal tìm cây khung
nhỏ nhất
int Kruskal(Graph *pG, Graph *pT){
   //1. Sắp xếp các cung của G theo thứ
tự trọng số tăng dần
   qsort(pG->edges, pG->m,
sizeof(Edge), cmpfunc);

   //2. Khởi tạo pT không chứ cung
nào, khởi tạo bộ quản lý các BPLT
   init_graph(pT, pG->n);
   for(int u=1; u<=pG->n; u++)
      parent[u] = u;//Mỗi đỉnh u là một
bộ phận liên thông
```



```
   int sum_w = 0; // Tổng trọng số các cung của cây

   //3. Duyệt qua các cung của G (đã sắp xếp)
   for(int e = 0; e < pG->m; e++){
      int u = pG->edges[e].u;
      int v = pG->edges[e].v;
      int w = pG->edges[e].w;
      int root_u = findRoot(u); //Tìm BPLT của u
      int root_v = findRoot(v); //Tìm BPLT của v
      if(root_u != root_v){ // u và v ở 2 bộ phận liên thông khác nhau
         //Thêm cung (u, v; w) vào cây pT
         add_edge(pT, u, v, w);
         //Gộp 2 BPLT root_u và root_v lại
         parent[root_v] = root_u;
         sum_w += w;
      }

   }
    return sum_w;
}


//Sử dụng thuật toán Kruskal
int main() {
   Graph G, T;
   int n, m, u, v, w, e;

   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);
   }

   int sum_w = Kruskal(&G, &T); //Gọi hàm Kruskal
   printf("%d\n", sum_w);

   for (e = 0; e < T.m; e++)
      printf("%d %d %d\n", T.edges[e].u, T.edges[e].v,
         T.edges[e].w);

   return 0;
}
```

Giải thuật Prim

1Khởi tạo
Pi[u] =oo, p[u] = -1
Mark[u] = 0, pi[s] = 0

2Lặp n-1 lần
Chọn u trong số các đỉnh chưa xét (mark[u] = 0) mà có pi[u] nhỏ nhất
Cập nhật mark[u] = 1

```
For(v là các đỉnh kề chưa xét của u )
If(pi[v] > trọng số cung (u, v))
   If(pi[v] > trọng số cung (u, v))
            Pi[v] = trọng số cung (u, v)
            P[v] = u;
3.Dựng cây (dựa vào các p[u] tìm được ở bước 2 )
For( u =1; u<=n; u++)
     If(p[u] != -1) thêm cung (p[u], u ) vào pT.
```

Viết chương trình đọc một đồ thị vô hướng liên thông và tìm cây khung có trọng số nhỏ nhất bằng thuật toán Prim.

```
#include<stdio.h>
#define MAX_LENGTH 100
#define MAX_VERTICES 100
#define MAX_EDGES 500
typedef struct{
            int u,v,w;
} Edge;
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=0; j<n; j++)
                                    G->A[i][j]=0;
}
void add_edge(Graph *G,int x, int y, int w){
            G->A[x][y]+=w;
            G->A[y][x]+=w;
}
void swap(Edge *a, Edge *b){
            Edge t;
            t=*a;
            *a=*b;
            *b=t;
}
int nho_hon(Edge a, Edge b){
            if((a.u<b.u) || ((a.u==b.u) && (a.v<b.v)) )
                        return 1;
            return 0;
}
void bubble_sort(Edge e[], int n){
            int i,j;
            for(i=0; i<=n-1; i++)
                        for(j=n-1; j>i; j--)
                                    if(nho_hon(e[j], e[j-1]))

                                    swap(&e[j], &e[j-1]);
}
typedef struct{
            int data[MAX_LENGTH];
            int size;
}List;
void make_null_list(List* L){
            L->size = 0;
}
int empty_list(List L){
            return 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];
}
int distancefrom(int u, List L, Graph G){
            int min_dist = 9999;
            int min_v = -1;
            int i;
            for(i=1; i<=L.size; i++){
                        int v = element_at(&L,i);
            if((G.A[u][v]!=0) && (min_dist>G.A[u][v])){

            min_dist=G.A[u][v];
                        min_v=v;
                        }
            }
            return min_v;
}
int check(List L, int x){
            int i;
            for(i=1; i<=L.size; i++)
            if(x==element_at(&L,i))
                        return 1;
            return 0;
}
Edge edges[100];
int dem=0;
int mark[100];
int prim(Graph G, Graph T){
            init_graph(&T, G.n);
            List L;
            make_null_list(&L);
            int u,i,sum_w=0;
            for(i=1; i<G.n; i++)
                        mark[i] = 0;
            push_back(&L,1);
            mark[1] = 1;
            for(i=1; i<G.n; i++){
                        int min_dist=9999, min_u, min_v;
            for(u=1; u<=G.n; u++)
                        if(mark[u]==0){
```

```
                int v =
distancefrom(u,L,G);

        if(v!=-1 &&
G.A[u][v]<min_dist){

                min_dist =
G.A[u][v];

                min_u = u;

                min_v = v;

                edges[dem].u =
v;

                edges[dem].v =
u;

                edges[dem].w =
min_dist;

                dem++;

        }
                }
        push_back(&L, min_u);
        mark[min_u]=1;
        add_edge(&T, min_u, min_v,
min_dist);
        sum_w += min_dist;
        }
        return sum_w;
}
int main(){
Graph G,T;
int e,n,m,u,v,w,i;
scanf("%d%d", &n, &m);
init_graph(&G,n);
for(e=0; e<m; e++){
        scanf("%d%d%d", &u, &v,
&w);
        add_edge(&G,u,v,w);
}
int sum_w = prim(G,T);
printf("%d", sum_w);
bubble_sort(edges, dem);
for(i=0; i<dem; i++)
        printf("\n%d %d %d",
edges[i].u, edges[i].v, edges[i].w);
return 0;
}
```

Thuật toán Ford – Fulkerson

Viết chương trình tìm luồng cực đại trên mạng bằng thuật toán Ford - Fulkerson (duyệt theo chiều rộng).

```
#include <stdio.h>
#define MAXN 100
#define NO_EDGE 0
#define INF 999999
typedef struct {
        int C[MAXN][MAXN];
        int F[MAXN][MAXN];
        int n;
}Graph;
void init_graph(Graph *G,int n){
        G->n=n;
}
typedef struct {
        int dir;
        int pre;
        int sigma;
} Label;
Label labels[MAXN];
void init_flow(Graph *G) {
        int u, v;
        for (u = 1; u <=G->n; u++)
                for (v = 1; v <= G-
>n; v++)
                        G-
>F[u][v] = 0;
}
typedef struct {
        int data[MAXN];
        int front, rear;
}Queue;
void make_null_queue (Queue* Q) {
        Q->front = 0;
        Q->rear = -1;
}
void enqueue (Queue *Q, int x) {
        Q->rear++;
        Q->data[Q->rear] = x;
}
int top(Queue *Q) {
        return Q->data[Q->front];
}
void dequeue(Queue * Q){
        Q->front++;
}
int empty (Queue* Q) {
        return Q->front>Q->rear;
}
int min (int a, int b) {
        return a < b ? a : b;
}
int FordFullkerson (Graph* G, int s, int
t) {
        init_flow(G);
        int u,v,sum_flow = 0;
        Queue Q;
  do {
                for (u = 1; u <= G-
>n; u++)
                labels[u].dir = 0;
                labels[s].dir = 1;
                labels[s].pre = s;
                labels[s].sigma =
INF;

        make_null_queue(&Q);
                enqueue(&Q,s);
                int found = 0;

        while(!empty(&Q)) {
                        int u =
top (&Q);

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

        if (labels[v].dir == 0 && G-
>C[u][v] != NO_EDGE && G->F[u][v] <
G-> C[u][v] ) {

                labels[v].dir = 1;

        labels[v].pre = u;

        labels[v].sigma =
min(labels[u].sigma, G->C[u][v] - G-
>F[u][v]);

        enqueue(&Q,v);

        }

        if (labels[v].dir == 0 && G-
>C[v][u] != NO_EDGE && G->F[v][u] >
0) {

        labels[v].dir = -1;

        labels[v].pre = u;

        labels[v].sigma =
min(labels[u].sigma, G->F[u][v]);

        enqueue(&Q,v);

        }
                        }

        if(labels[t].dir != 0) {

        found = 1;

        break;
                        }
                }
                if (found == 1) {
                        int x =
t;
                        int
sigma = labels[t].sigma;

        sum_flow += sigma;

        while(x!=s) {

        int u = labels[x].pre;

        if (labels[x].dir>0)
```

```
                        G->F[u][x] +=
sigma;

                else

                        G->F[x][u] -=
sigma;

                x = u;
                                }
                        } else break;
                } while(1);
                return sum_flow;
}
int main() {
Graph G;
int n, m, u, v, c ,e;
scanf("%d%d", &n, &m);
init_graph(&G, n);
for (e = 0; e < m; e++) {
                scanf("%d%d%d", &u, &v,
&c);
                G.C[u][v] = c;
}
int max_flow =
FordFullkerson(&G,1,n);
                printf ("Max flow: %d
\n",max_flow);
                printf("X0: ");
for ( e = 1; e <= G.n; e++) {
if (labels[e].dir != 0)
                printf("%d ",e);
}
printf("\nY0: ");
for (e = 1; e <= G.n; e++) {
                if(labels[e].dir == 0)
                        printf("%d ",e);
}
return 0;
}
```

Cho mạng được biểu diễn bằng đồ thị n đỉnh, m cung. Đỉnh phát s = 1 và đỉnh thu t = n. Mỗi cung (u, v) có khả năng thông qua là C[u][v] và luồng đi qua nó là F[u][v]. Để tìm luồng lớn nhất trên mạng bằng giải thuật Ford - Fulkerson, ta phải khởi tạo một **luồng hợp lệ** nào đó trên mạng và sau đó tìm cách tăng luồng.

```
#include <stdio.h>

#define MAX_N 1000

int n, m;
int C[MAX_N+1][MAX_N+1]; // Ma
trận chứa thông tin về khả năng thông
qua
int F[MAX_N+1][MAX_N+1]; // Ma
trận chứa thông tin về luồng

int main() {
  scanf("%d%d", &n, &m);

  // Đọc thông tin về mạng từ input
  for (int i = 0; i < m; i++) {
    int u, v, c, f;
    scanf("%d%d%d%d", &u, &v, &c,
&f);
    C[u][v] = c;
    F[u][v] = f;
  }

  // Kiểm tra điều kiện 1
  int valid_flow = 1;
  for (int u = 1; u <= n; u++) {
    for (int v = 1; v <= n; v++) {
      if (F[u][v] < 0 || F[u][v] >
C[u][v]) {
        valid_flow = 0;
        break;
      }
    }
    if (!valid_flow) break;
  }

  // Kiểm tra điều kiện 2
  if (valid_flow) {
    int s_out = 0, t_in = 0;
    for (int i = 1; i <= n; i++) {
      s_out += F[1][i];
      t_in += F[i][n];
    }
    if (s_out != t_in) valid_flow = 0;
  }

  // Kiểm tra điều kiện 3
  if (valid_flow) {
    for (int u = 2; u < n; u++) {
      int in_u = 0, out_u = 0;
      for (int i = 1; i <= n; i++) {
        in_u += F[i][u];
        out_u += F[u][i];
      }
      if (in_u != out_u) {
        valid_flow = 0;
        break;
      }
    }
  }

  // In kết quả
  if (valid_flow) {
    printf("YES\n");
  } else {
    printf("NO\n");
  }
  return 0;
}
```

Cho đồ thị G = <V, E> vô hướng, liên thông và có trọng số. Viết chương trình tìm cách xoá một số cung của G sao cho G vẫn còn liên thông và tổng trọng số của các cung bị xoá là lớn nhất.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Edge {
  int u;
  int v;
  int w;
} Edge;

int cmp(const void* a, const void* b) {
  return (*(Edge*)a).w -
(*(Edge*)b).w;
}

int find(int u, int parent[]) {
  if (parent[u] == u) {
    return u;
  }
  return parent[u] = find(parent[u],
parent);
}

void merge(int u, int v, int parent[]) {
  parent[find(u, parent)] = find(v,
parent);
}

int is_connected(int n, int parent[]) {
  int root = find(0, parent);
  for (int i = 1; i < n; i++) {
    if (find(i, parent) != root) {
      return 0;
    }
  }
  return 1;
}

int main() {
  int n, m;
  scanf("%d %d", &n, &m);

  Edge edges[m];
  for (int i = 0; i < m; i++) {
    scanf("%d %d %d", &edges[i].u,
&edges[i].v, &edges[i].w);
    edges[i].u--;
    edges[i].v--;
  }
```

```
  qsort(edges, m, sizeof(Edge), cmp);

  int parent[n];
  for (int i = 0; i < n; i++) {
    parent[i] = i;
  }

  int sum_weight = 0;
  for (int i = 0; i < m; i++) {
    int u = edges[i].u;
    int v = edges[i].v;
    int w = edges[i].w;
    if (find(u, parent) != find(v,
parent)) {
      merge(u, v, parent);
    } else if (is_connected(n, parent))
{
      sum_weight += w;
    }
  }

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

  return 0;
}
```

Microsoft Excel là chương trình xử lý bảng tính nằm trong bộ Microsoft Office của hãng phần mềm Microsoft. Excel được thiết kế để giúp ghi lại, trình bày các thông tin xử lý dưới dạng bảng, thực hiện tính toán và xây dựng các số liệu thống kê trực quan.

```
#include <stdio.h>
#include <stdbool.h>

#define MAXN 1000

int n, m;
int adj[MAXN][MAXN];
bool visited[MAXN];
bool onStack[MAXN];

bool hasCycle(int u) {
  visited[u] = true;
  onStack[u] = true;
  for (int v = 1; v <= n; v++) {
    if (adj[u][v]) {
      if (!visited[v]) {
        if (hasCycle(v)) {
          return true;
        }
      } else if (onStack[v]) {
        return true;
      }
    }
  }
  onStack[u] = false;
  return false;
}

bool isCircularReference() {
  for (int i = 1; i <= n; i++) {
    visited[i] = false;
    onStack[i] = false;
  }
  for (int i = 1; i <= n; i++) {
    if (!visited[i]) {
      if (hasCycle(i)) {
        return true;
      }
    }
  }
  return false;
}

int main() {
  scanf("%d %d", &n, &m);
  for (int i = 0; i < m; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    adj[u][v] = 1;
  }
  if (isCircularReference()) {
    printf("CIRCULAR
REFERENCE\n");
  } else {
    printf("OK\n");
  }
  return 0;
}
```

Thuyền trưởng Haddock (truyện Tintin)

```
#include<stdio.h>
#define MAX_ELEMENTS 100
#define MAX_VERTICES 100

typedef struct {
 int n;
 int
A[MAX_VERTICES][MAX_VERTICES];
} Graph;

typedef int ElementType;
typedef struct {
 ElementType data[MAX_ELEMENTS];
 int size;
} List;

void make_null_list(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];
}

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;
}

int adjacent(Graph* G, int x, int y) {
        return G->A[x][y] != 0;
}

List neighbors(Graph* G, int x) {
        int y;
        List list;
        make_null_list(&list);
        for (y = 1; y <= G->n; y++)
                if (adjacent(G, x,
y))

        push_back(&list, y);
        return list;
}
#define white 0
#define black 1
#define gray 2

int color[MAX_VERTICES];
int cycle;
void dfs(Graph* G, int x) {
        color[x] = gray;
        int j;
        List list = neighbors(G, x);
        for (j = 1; j <= list.size; j++) {
                int y =
element_at(&list, j);
                if (color[y] ==
gray) {
                        cycle =
1;

        return;
                }
                if (color[y] ==
white) {
                        dfs(G,
y);
                }
        }
        color[x] = black;
}

int contains_cycle(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) {
                        dfs(G,
j);
                }
        }
return cycle;
}
int main() {
        Graph G;
        int n, m, e, u, v;
        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;
}
```

DUYỆT THEO CHIỀU SÂU BẰNG ĐỆ QUY

```
#include <stdio.h>
#define MAX_VERTEXES 100
#define MAX_ELEMENTS 100
#define MAX_NODES 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;
}

typedef struct {
        int n, m;
        int
A[MAX_VERTEXES][MAX_VERTEXES];
}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 adjacent (Graph *G, int x, int y) {
        return G->A[x][y] != 0;
}

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;
}

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_VERTEXES];

void depth_first_search(Graph* G, int
x) {
        Stack frontier;
        make_null_stack(&frontier);
        int j ;
        for (j = 1; j <= G->n; j++)
        push(&frontier, x);
        while (!empty(&frontier)) {
                int x =
top(&frontier); pop(&frontier);
                if (mark[x] != 0)
                continue;
                printf("%d\n",
x);
                mark[x] = 1;
                List list =
neighbors(G, x);
                for (j = 1; j <=
list.size; j++) {
                        int y =
element_at(&list, j);

        push(&frontier, y);
                }
        }
}
int main () {
        Graph G;
        int n, m, u, v, w, 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);
        }
        depth_first_search(&G, 1);
        for (w = 1; w <= n; w++)
                if (mark[w] == 0)

        depth_first_search(&G, w);
        return 0;
}
```

PHẦN LÝ THUYẾT

THUẬT TOÁN TARJAN :

-Thứ tự các đỉnh kề của 1 được sắp xếp từ bé đến lớn

-Khi duyệt xong 1 đỉnh v, quay về đỉnh cha u (đỉnh trước), cập nhật lại **Min_num[u] = min (min_num[u], min_num[v])**

-Khi xét 1 đỉnh kề v của u mà v đang có mặt trong Stack, cập nhật lại :

Min_num[u] = min(min_num[u], num[v]).

Đồ thị phân đôi :

Ý tưởng:

Mỗi đỉnh sẽ có 1 trong 3 trạng thái tương ứng với 3 màu:

- Chưa có màu: NO_COLOR.
- Được tô màu: BLUE

- Được tô màu đỏ: RED

Thuật toán duyệt đồ thị đệ qui kết hợp tô màu colorize(u, c) gồm các bước sau

- Tô màu c cho u
- Xát các đỉnh kề v của u, có 3 trường hợp:
  + Nếu v chưa có màu -> gọi đệ quy tô màu ngược lại với c cho nó
  + Nếu v đã có màu và giống với u -> đụng độ không tô được
  + Nếu v đã có màu và khác màu với u -> bỏ qua

Tìm màu ngược lại màu của c.
Gọi S = BLUE + RED
Để tìm màu ngược lại của c, ta lấy S trừ cho c cụ thể:

- Ngược lại của BLUE là S – BLUE = RED
- Ngược lại màu của RED là S – RED = BLUE

Thuật toán MooreDijkstra

Pi[v] = min(pi[v], pi[u] + L(u,v));

Quản lý dự án :

t[alpha] = 0
T[beta] = t[beta]
t[u] = max{t[x] + d[x]}
T[u] = min{T[v]} – d[u]

Thuật toán Prim

Pi[v] = min(pi[v], L(u,v));

Thuật toán Chiuliu-Edmond

Pha co :
-Gọi đồ thị gốc là Go, t = 0
-Cho dòng lặp chạy
+ Xây dựng đồ thị xấp xỉ Ht từ Gt
+ Kiểm tra điều kiện dừng nếu Ht không có chu trình -> thoát vòng lặp chuyển sang pha giản
+ Ht có chu trình -> co Gt thành Gt+1
+ tang t = t + 1

Pha giản :
+ Mở đỉnh
+ Xóa cung
+ điều chỉnh trọng số