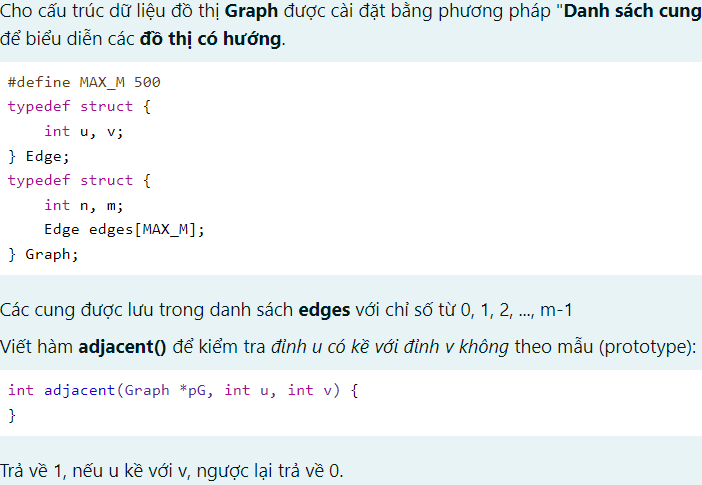
**LÝ THUYẾT ĐỒ THỊ**

|  |  |
| --- | --- |
| **// List**  typedef struct {  int data[100];  int size;  }List;  void makenullList(List \*pL){  pL->size = 0;  }  int empty\_List(List \*pL){  return pL->size == 0;  }  void push\_back(List \*pL, int x){  pL->data[pL->size++] = x;  }  int element\_at(List \*pL, int i){  return pL->data[i-1];  }  **// Queue**  typedef struct {  int data[100];  int front, rear;  }Queue;  void make\_null\_Queue(Queue \*pQ){  pQ->front = 0;  pQ->rear = -1;  }  int empty\_Queue(Queue \*pQ){  return ( pQ->front > pQ->rear);  }  void enQueue(Queue \*pQ, int x){  pQ->data[++pQ->rear] = x;  }  void deQueue(Queue \*pQ){  pQ->front++;  }  int Front(Queue \*pQ){  return pQ->data[pQ->front];  **}**  // **DFS**  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);  }  }  **/ Kiểm tra đồ thị vô hướng phân đôi**  #define NO\_COLOR 0 #define BLUE 1  #define RED 2 int color[100], 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(){  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;  }  **// Chuyển Mê Cung Số => MT đỉnh - đỉnh**  Graph G;  init\_graph(&G, n\*m);  int di[] = {-1, 1, 0, 0};  int dj[] = {0, 0, -1, 1};  int u, v, k, ii, jj;  for(i = 0; i < m; i++)  for(j = 0; j < n; j++){  u = (i\*n + j) + 1;  for(k = 0; k <= 3; k++){  ii = i + di[k];  jj = j + dj[k];  if( ii >= 0 && ii < m && jj >= 0 && jj < n){  v = ii\*n + jj + 1;  add\_edge(&G, u, v, A[(v-1)/n][(v-1)%n]);  }  }  }  **// Bellman\_Ford**  int pi[MAXN], p[MAXN], negative\_cycle = 0;  void BellmanFord(Graph \*pG, int s){  int it, k, u, v, w;  for(u = 1; u <= pG->n; u++){  pi[u] = oo;  p[u] = 0;  }  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;  }  }  }  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]){  negative\_cycle = 1;  return;  }  }  }  }  for(v = 1; v <= pG->n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  pi[u][u] = 0;  }  for(u = 1; u <= pG->n; u++){  for(v = 1; v <= pG->n; v++){  if(pG->W[u][v] != NOEDGE){  pi[u][v] = pG->W[u][v];  next[u][v] = v;  }  }  }  for(k = 1; k <= pG->n; k++){  for(u = 1; u <= pG->n; u++){  for(v = 1; v <= pG->n; v++){  if(pi[u][k] != oo && pi[k][v] != oo &&  pi[u][k] + pi[k][v] < pi[u][v]) {  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  }  }  }  }  for(u = 1; u <= pG->n; u++){  if(pi[u][u] < 0){  negative\_cycle = 1;  break;  }  }  } | **// Stack**  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->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);  }  // **BFS**  void BFS(Graph \*pG, int s){  Queue Q;  make\_null\_Queue(&Q);  (&Q, s);  while( !empty(&Q) ){  int u = front(&Q);  deQueue(&Q);  if( mark[u] != 0)  continue;  printf("%d\n", u);  mark[u] = 1;  (int v = 1; v <= pG->n; v++){  if( adjacent(pG, u, v) )  enQueue(&Q, v);  }  }  }  **// Kiểm tra đồ thị chứa chu trình**  #define White 0  #define Gray 1  #define Black 2  int color[Max\_M];  int has\_circle, v;  void DFS(Graph \*pG, int u){  color[u] = Gray;  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;  }  **// SCC ( Tarjan)**  int num[100], min\_num[100], on\_stack[100], k;  Stack S;  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]);  }  }else;  }  if(num[u] == min\_num[u]){  int w;  do{  w = top(&S);  pop(&S);  on\_stack[w] = 0;  }while(w!=u);  }  }  int main(){  for( int i = 1; i <= n; i++){  num[i] = -1;  min\_num[i] = 0;  on\_stack[i] = 0;  }  k = 1;  Makenull\_Stack( &S);  }  // **MooreDijkstra**  #include “limits.h”  #define oo INT\_MAX  void MooreDijkstra(Graph \*pG, int s){  int u, v, it;  for(u = 1; u <= pG->n; u++){  pi[u] = oo;  mark[u] = 0;  }  p[s] = -1;  pi[s] = 0;  for(it = 1; it < pG->n; it++){  int j, min\_pi = oo;  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){  int p = pi[u] + pG->W[u][v];  if( p < pi[v]){  pi[v] = p;  p[v] = u;  }  }  }  }  **// Đường đi ngắn nhất từ s đến t**  #include <stdio.h>  #define Max 100  typedef struct{  int n,m;  int A[Max][Max];  }Graph;  void init\_graph(Graph \*pG, int n){  int i,j;  pG->n=n;  for(i=1;i<=pG->n;i++){  for(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;  }  int adjacent(Graph \*pG, int u, int v){  if((pG->A[u][v]!=0) && (pG->A[v][u]!=0)) return 1;  return 0;  }  int main(){  Graph G;  int i,m,n,u,v;  freopen("test.txt","r",stdin);  scanf("%d%d",&n,&m);  init\_graph(&G,n);  for(i=1;i<=m;i++){  scanf("%d%d",&u,&v);  add\_edge(&G,u,v);  }  return 0;  }  **// Floy\_Warshall**  int pi[MAXN][MAXN], next[MAXN][MAXN],  negative\_cycle = 0;  void FloyWarshall(Graph \*pG){  int u, v, k;  for(u = 1; u <= pG->n; u++){  **Danh Sách Cung (DSC)**  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++;  }  } |

|  |  |
| --- | --- |
| **Danh Sách Cung (DSC)** | **Ma Trận Kề** |
| //ds duyet  #include<stdio.h>  #define MAX 500  typedef struct{  int u,v;  }Edge;  typedef struct{  int n,m;  Edge A[MAX];  }Graph;  //khoi tao  void init(Graph \*G, int n){  G->n=n;  G->m=0;  }  //them cung  void add(Graph \*G, int u, int v){  G->A[G->m].u=u;  G->A[G->m].v=v;  G->m++;  }  //ke  int adj(Graph \*G, int u, int v){  for(int i=0;i<G->m;i++)  if((G->A[i].u==u && G->A[i].v==v) ||  (G->A[i].u==v && G->A[i].v==u))  return 1;  return 0;    }  //bac  int bac(Graph \*G, int x){  int count=0;  for(int i=0; i<G->m;i++){  if(G->A[i].u==x)  count++;  if(G->A[i].v==x)  count++;  }  return count;  }  int main(){  Graph G;  int n,m,u,v;  freopen("dothi.txt", "r", stdin);  scanf("%d %d", &n, &m);  init(&G, n);  printf("n = %d, m = %d\n", G.n, G.m);  //them cung  for(int i=0;i<m;i++){  scanf("%d %d", &u, &v);  add(&G, u,v);  }  //in cung  for(int j=0;j<G.m;j++){  printf("%d %d\n", G.A[j].u, G.A[j].v);  }  //ke  printf("Danh sach ke:\n");  for (int j = 1; j <= n; j++) {  printf("Ke voi dinh %d: ", j);    for (int k = 1; k <= n; k++) {  if (j != k && adj(&G, j, k)) {  printf("%d ", k);  }  }  printf("\n");  }  //bac  for(int h=1;h<=G.n;h++){  printf("bac(%d) la: %d\n", h, bac(&G,h));  }  }  **Ma trận đỉnh - cung**  #include <stdio.h>  #define max\_n 100  #define max\_m 500  typedef struct {  int n, m;  int A[max\_n][max\_m]; //ma tran lien thuoc  } Graph;  //khoi tao  void khoitao(Graph \*G, int n, int m) {  G->n = n;  G->m = m;  for (int i = 1; i <= n; i++) {  for (int j = 1; j <= m; j++) {  G->A[i][j] = 0;  }  }  }  //them cung  void add(Graph \*G, int u, int v, int e) {  G->A[u][e]++;  G->A[v][e]++;  }  //kt ke  int adj(Graph \*G, int u, int v) {  for (int i = 1; i <= G->m; i++) {  if (G->A[u][i] > 0 && G->A[v][i] >0)  return 1;  return 0;  }  }  //tinh bac  int degree(Graph \*G, int u) {  int dem = 0;  for (int i = 1; i <= G->m; i++) {  //if (G->A[u][i] == 1)  dem+=G->A[u][i];  }  return dem ;  }  int main() {  Graph G;  int n, m,u, v;  freopen("dothi.txt","r", stdin);  //printf("Nhap so dinh n va so cung m: ");  scanf("%d%d", &n, &m);  khoitao(&G, n, m);  //Thêm các cung  for (int i = 1; i <= m; i++) {  // int ;  // printf("Nhap cung thu %d (u, v): ", i);  scanf("%d%d", &u, &v);  add(&G, u, v, i);  }  // In ma tr?n k?  printf("\nMa tran ke:\n");  for (int i = 1; i <= n; i++) {  for (int j = 1; j <= m; j++) {  printf("%d ", G.A[i][j]);  }  printf("\n");  }  // In các d?nh k?  printf("\nCac dinh ke:\n");  for (int i = 1; i <= n; i++) {  printf("Cac dinh ke cua dinh %d: ", i);  for (int j = 1; j <= m; j++) {  if (G.A[i][j] > 0) {  // Tìm d?nh k? b?ng cách l?y ch? s? u ho?c v c?a cung  int u = 0, v = 0;  for (int k = 1; k <= n; k++) {  if (G.A[k][j] == 1 && u == 0)  u = k;  else if (G.A[k][j] == 1 && v == 0)  v = k;  }  if (u == v && u == i) {  printf("%d ", i); // In ra d?nh t? k?  } else {  printf("%d ", (u == i) ? v : u); // In ra d?nh k?  }  }  }  printf("\n");  }  // Tính b?c c?a m?t d?nh  for (int i = 1; i <= G.n; i++) {  printf("bac(%d): %d\n", i,degree(&G,i));  }  return 0;  } | #include<stdio.h>  #define max 100  typedef struct{  int n,m;  int A[max][max];  }Graph;  //DON DO THI VO HUONG  //khoi tao  void khoitao(Graph \*G, int n){  G->n=n;  G->m=0;  for(int i=0;i<n;i++){  for(int j=0;j<n;j++){  G->A[i][j]=0;  }  }  }  //them cung  void add\_vh(Graph \*G, int u, int v){  G->A[u][v]=1;  G->A[v][u]=1;  G->m++;  }  //DO THI VO HUONG ->CUNG, KHUYEN  //them cung  void add\_vh\_cung(Graph \*G, int u, int v){  G->A[u][v]++;  if(u!=v)  G->A[v][u]++;  G->m++;  }  //tinh bac  int bac\_vh\_cunh(Graph \*G, int u){  int dem=0;  for(int i=1;i<=G->n;i++){  dem = dem+ G->A[u][i];  }  return dem + G->A[u][u];  }  //tinh so cung  int edgeCount(Graph\* G) {  int count = 0;  // Duy?t qua ma tr?n d?nh - d?nh và d?m s? cung  for (int i = 0; i < G->n; i++) {  for (int j = i; j < G->n; j++) { // B?t d?u t? i d? tránh tính cung trùng l?p  count += G->A[i][j]; // Tang count lên b?ng giá tr? c?a m?i ph?n t? trong ma tr?n  }  }  // Tr? v? count  return count;  }  //dinh ke theo chieu tang dan  void ke\_vh\_cung(Graph \*G, int u){  printf("ke %d: ", u);  for(int i=1;i<=G->n;i++){  if(G->A[u][i]>=1){  printf("%d ", i);  }  }  printf("\n");  }  //DO THI CO HUONG  //them cung  void add\_ch(Graph \*G, int u, int v){  G->A[u][v]=1;  G->m++;  }  //DO THI CO HUONG-> CUNG, KHUYEN  //them cung  void add\_ch\_cung(Graph \*G, int u, int v){  G->A[u][v]++;  G->m++;  }  //tinh bac tong trong + ngoai  int bac\_ch\_cung(Graph \*G, int u){  int bac=0;  for(int i=1;i<=G->n;i++){  bac+= G->A[u][i]+ G->A[i][u];  }  return bac;  }  //bac trong ->A  int bac\_trong(Graph \*G, int u){  int bac=0;  for(int i=1;i<=G->n;i++){  bac+=G->A[i][u];  }  return bac;  }  //bac ngoai A->  int bac\_ngoai(Graph \*G, int u){  int bac=0;  for(int i=1;i<=G->n;i++){  bac+=G->A[u][i];  }  return bac;  }  //dinh ke  void ke\_ch\_cung(Graph \*G, int u){  printf("ke %d: ",u);  for(int i=1;i<=G->n;i++){  if(G->A[u][i]>=1){  printf("%d ", i);  }  }  printf("\n");  }  int main(){  Graph G;  // int n,m,u,v;  freopen("dothi.txt", "r", stdin);  /\* scanf("%d %d", &n, &m);  khoitao(&G, n);  //doc cung  for(int i=1;i<=m;i++){  scanf("%d %d", &u,&v);  add\_vh\_cung(&G,u,v);  }  //in ma tran  printf("ma tran ke:\n");  for(int k=1;k<=n;k++){  for(int l=1;l<=n;l++){  printf("%d ", G.A[k][l]);  }  printf("\n");  }  //in bac  for(int b=1;b<=n;b++){  printf("bac tong(%d)= %d\n", b, bac\_vh\_cunh(&G,b));  }  printf("\*\*\*\*\*\*\*\*\n");  //bac trong  for(int b=1;b<=n;b++){  printf("bac trong(%d)= %d\n", b, bac\_trong(&G,b));  }  printf("\*\*\*\*\*\*\*\*\n");  //bac ngoai  for(int b=1;b<=n;b++){  printf("bac ngoai(%d)= %d\n", b, bac\_ngoai(&G,b));  }  //ke  for(int i=1;i<=n;i++){  ke\_vh\_cung(&G, i);  }\*/  //dem so cung  scanf("%d", &G.n);  int i, j;  for (i = 1; i <= G.n; i++)  for (j = 1; j <= G.n; j++) {  scanf("%d", &G.A[i][j]);  }  printf("so cung do thi: %d\n", edgeCount(&G));  return 0;  } |



int adjacent(Graph \*pG, int u, int v) {

int i;

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

if ((pG->edges[i].u == u && pG->edges[i].v == v)) {

return 1; // Đỉnh u kề với đỉnh v

}

}

return 0; // Không có kề

}

**Câu 2: Viết chương trình đọc một đồ thị vô hướng liên thông và tìm cây khung có trọng số nhỏ nhất bằng [thuật toán Prim](https://else.ctu.edu.vn/mod/page/view.php?id=8078" \o "Thuật toán Prim).**

- Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

- Đầu ra (Output)

In ra màn hình theo định dạng sau:

#include <stdio.h>

#include <limits.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 = INT\_MAX;

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 = INT\_MAX, 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 n, m, u, v, w, i;

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

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);

}

int sum\_w = prim(G, T);

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

bubble\_sort(edges, dem);

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

printf("%d %d %d\n", edges[i].u, edges[i].v, edges[i].w);

return 0;

}

**Câu 4: Cho cấu trúc dữ liệu đồ thị Graph được cài đặt bằng phương pháp "Ma trận kề" dùng để lưu trữ các đồ thị có hướng (có thể chứa đa cung và chứa khuyên).**

#define MAX\_N 100typedef struct {

int n, m;

int A[MAX\_N][MAX\_N];

} Graph;

Viết hàm int indegree(Graph \*pG, int u) để tính bậc vào của đỉnh theo mẫu (prototype):

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

**int bac=0;**

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

**bac+= pG->A[i][u];**

**}**

**return bac;**

**}**

**Câu 5: Cho cấu trúc dữ liệu đồ thị Graph được cài đặt bằng phương pháp "Danh sách cung" như sau:**

typedef struct  {  
 int u, v;  
} Edge;  
typedef struct {  
 int n, m;  
 Edge edges[MAX\_M];  
} Graph;

Các cung được lưu trong danh sách edges với chỉ số từ 0, 1, 2, ..., m-1

Hàm khởi tạo đồ thị:

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

pG->n = n;

pG->m = 0;

}

Viết hàm add\_edge(Graph \*pG, int u, int v) để thêm cung (u, v) vào đồ thị pG theo mẫu (prototype):

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

**//Đưa cung (u, v) vào edges**

**pG->edges[pG->m].u = u;**

**pG->edges[pG->m].v = v;**

**//Tăng số cung lên 1**

**pG->m++;**

**}**

**Câu 6: Viết chương trình tìm luồng cực đại trên mạng bằng thuật toán Ford - Fulkerson (duyệt theo chiều rộng).**

- Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

- Đầu ra (Output)

In ra màn hình theo định dạng sau: Max flow: f với f là giá trị luồng cực đại

|  |  |
| --- | --- |
| #include <stdio.h>  #include <stdlib.h>  typedef int ElementType;  typedef struct Node {  ElementType Data;  struct Node \*Next;  } Node;  typedef struct {  Node \*Front, \*Rear;  } Queue;  void make\_null\_queue (Queue \*pQ){  Node \*Header = (Node\*)malloc(sizeof(Node));  Header->Next = NULL;  pQ->Front = Header;  pQ->Rear = Header;  }  void enqueue (Queue \*pQ, ElementType x){  pQ->Rear->Next = (Node\*)malloc (sizeof (Node));  pQ->Rear = pQ->Rear->Next;  pQ->Rear->Data = x;  pQ->Rear->Next = NULL;  }  ElementType dequeue (Queue \*pQ){  Node \*Temp;  Temp = pQ->Front;  pQ->Front = pQ->Front->Next;  free(Temp);  return pQ->Front->Data;  }  #define MAX\_N 100  #define NO\_EDGE 0  #define oo 1e9  typedef struct {  int d, p, s;  } label;  typedef struct {  int n;  int C[MAX\_N][MAX\_N], F[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  for(int u = 1; u <= n; u++){  for(int v = 1; v <= n; v++){  G->C[u][v] = NO\_EDGE;  }  }  }  void read\_graph(Graph \*G){  int n, m;  scanf("%d %d", &n, &m);  init\_graph(G, n);  for(int i = 0; i <= m; i++){  int u, v, c;  scanf("%d %d %d", &u, &v, &c);  G->C[u][v] = c;  }  }  label lb[MAX\_N];  void reset(Graph \*G){  for(int u = 1; u <= G->n; u++){  for(int v = 1; v <= G->n; v++){  G->F[u][v] = 0;  }  }  }  int min(int a, int b){  return a < b ? a : b;  } | int edmonds\_karp(Graph \*G, int s, int t){  for(int u = 1; u <= G->n; u++)  lb[u].d = 0;  lb[s].d = 1;  lb[s].p = s;  lb[s].s = oo;  Queue Q;  make\_null\_queue(&Q);  enqueue(&Q, s);  while(Q.Front != Q.Rear) {  int u = dequeue(&Q);  for(int v = 1; v <= G->n; v++){  if(G->C[u][v] != NO\_EDGE && !lb[v].d && G->F[u][v] < G->C[u][v]){  lb[v].d = 1;  lb[v].p = u;  lb[v].s = min(lb[u].s, G->C[u][v] - G->F[u][v]);  enqueue(&Q, v);  }  }  for(int x = 1; x <= G->n; x++){  if(G->C[x][u] != NO\_EDGE && !lb[x].d && G->F[x][u]){  lb[x].d = 1;  lb[x].p = u;  lb[x].s = min(lb[u].s, G->F[x][u]);  enqueue(&Q, x);  }  }  if (lb[t].d) return 1;  }  return 0;  }  int ford\_fullkerson(Graph \*G, int s, int t){  reset(G);  int max\_flow = 0;  while(edmonds\_karp(G, s, t)){  int sigma = lb[t].s, u = t;  while(u != s){  int p = lb[u].p;  if(lb[u].d > 0)  G->F[p][u] += sigma;  if(lb[u].d < 0)  G->F[u][p] -= sigma;  u = p;  }  max\_flow += sigma;  }  return max\_flow;  }  int main(){  Graph G;  read\_graph(&G);  printf("Max flow: %d\n", ford\_fullkerson(&G, 1, G.n));  printf("X0: ");  for(int u = 1; u <= G.n; u++){  if(lb[u].d)  printf("%d ", u);  }  printf("\nY0: ");  for(int u = 1; u <= G.n; u++){  if(!lb[u].d)  printf("%d ", u);  }  } |

**Câu 7: 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.**

- Đầu vào

Dữ liệu đầu vào được nhập từ bàn phím với định dạng

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.

- Đầu ra

In ra màn hình tổng trọng số của các cung bị xoá.

#include <stdio.h>

#include <stdlib.h>

// Định nghĩa cấu trúc Edge để biểu diễn cung trong đồ thị

typedef struct {

int u, v, w; // Đỉnh u và v kết nối bởi cung có trọng số w

} Edge;

// Hàm so sánh hai cung dựa trên trọng số

int compare(const void \*a, const void \*b) {

return ((Edge \*)a)->w - ((Edge \*)b)->w;

}

// Hàm tìm đỉnh gốc của tập hợp chứa đỉnh u

int find(int parent[], int u) {

if (parent[u] != u) {

parent[u] = find(parent, parent[u]);

}

return parent[u];

}

// Hàm kết hợp hai tập hợp chứa u và v

void unionSet(int parent[], int rank[], int u, int v) {

int rootU = find(parent, u);

int rootV = find(parent, v);

if (rootU != rootV) {

if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

} else if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

} else {

parent[rootV] = rootU;

rank[rootU]++;

}

}

}

// Hàm tính tổng trọng số của các cung bị xoá

int maxRemovedEdgesWeight(Edge edges[], int n, int m) {

int \*parent = (int \*)malloc((n + 1) \* sizeof(int));

int \*rank = (int \*)calloc((n + 1), sizeof(int));

int minSpanningTreeWeight = 0;

// Khởi tạo parent ban đầu

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

parent[i] = i;

}

// Sắp xếp các cung theo trọng số tăng dần

qsort(edges, m, sizeof(Edge), compare);

// Tìm cây khung nhỏ nhất và tính tổng trọng số của cây khung nhỏ nhất

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

Edge edge = edges[i];

int rootU = find(parent, edge.u);

int rootV = find(parent, edge.v);

if (rootU != rootV) {

minSpanningTreeWeight += edge.w;

unionSet(parent, rank, rootU, rootV);

}

}

// Tính tổng trọng số của các cung bị xoá

int removedEdgesWeight = 0;

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

removedEdgesWeight += edges[i].w;

}

removedEdgesWeight -= minSpanningTreeWeight;

free(parent);

free(rank);

return removedEdgesWeight;

}

int main() {

int n, m;

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

// Khởi tạo mảng lưu trữ các cung

Edge \*edges = (Edge \*)malloc(m \* sizeof(Edge));

// Đọc các cung và trọng số từ bàn phím

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

scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);

}

// Tính tổng trọng số của các cung bị xoá

int totalRemovedEdgesWeight = maxRemovedEdgesWeight(edges, n, m);

// In kết quả ra màn hình

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

// Giải phóng bộ nhớ

free(edges);

return 0;

}

Câu 8: 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.

- Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

- Đầu ra (Output)

In các đỉnh ra màn hình theo thứ tự topo. In các đỉnh trên một dòng, cách nhau 1 khoảng trắng.

**Gợi ý**

Ngoài giải thuật sắp xếp topo dựa trên phương pháp duyệt theo chiều rộng, ta cũng có thể sắp xếp topo dựa trên duyệt theo chiều sâu.

|  |  |
| --- | --- |
| #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTEXES 100  typedef struct {  int data[MAX\_VERTEXES];  int size;  } List;  typedef struct {  int n;  int A[MAX\_VERTEXES][MAX\_VERTEXES];  } Graph;  void make\_null\_list(List \*list) {  list->size = 0;  }  void push\_back(List \*list, int x) {  list->data[list->size] = x;  list->size++;  }  int empty\_list(List list) {  return list.size == 0;  }  void init\_graph(Graph \*G, int n) {  G->n = n;  int i, j;  for (i = 1; i <= n; i++)  for (j = 1; j <= n; j++)  G->A[i][j] = 0;  }  void add\_edge(Graph \*G, int x, int y) {  G->A[x][y] = 1;  } | int mark[MAX\_VERTEXES];  void dfs(Graph \*G, int u, List \*L) {  mark[u] = 1;  int v;  for (v = 1; v <= G->n; v++)  if (G->A[u][v] && mark[v] == 0)  dfs(G, v, L);  push\_back(L, u);  }  void topo\_sort(Graph \*G, List \*L) {  make\_null\_list(L);  int u;  for (u = 1; u <= G->n; u++)  mark[u] = 0;  for (u = 1; u <= G->n; u++)  if (mark[u] == 0)  dfs(G, u, L);  }  int main() {  //freopen("input.txt", "r", stdin);  Graph G;  int n, m, u, v;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (int i = 0; i < m; i++) {  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  }  List L;  topo\_sort(&G, &L);  for (int i = L.size - 1; i >= 0; i--)  printf("%d ", L.data[i]);  return 0;  } |

Câu 9: Ngưu lang chức nữ

- Đầu vào (Input):

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số ngôi sao và số cặp sao có thể bắt cầu.

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u v q nói rằng để bắt 1 cây cầu bắt qua hai ngôi sao u và v cần phải tốn q con quạ (0<q<100)(0<𝑞<100).

- Đầu ra (Output):

In ra màn hình số lượng quạ cần thiết.

|  |  |
| --- | --- |
| #include <stdio.h>  #define MAX\_N 100  #define NO\_EDGE -1  #define oo 99999  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  for(int i = 1; i <= n; i++){  for(int j = 1; j <= n; j++){  G->A[i][j] = NO\_EDGE;  }  }  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  void read\_graph(Graph \*G) {  int n, m;  scanf("%d %d", &n, &m);  init\_graph(G, n);  for(int i = 0; i < m; i++){  int u, v, w;  scanf("%d %d %d", &u, &v, &w);  add\_edge(G, u, v, w);  }  } | int mark[MAX\_N], p[MAX\_N], pi[MAX\_N];  void dijkstra(Graph \*G, int u){  for(int i = 1; i <= G->n; i++){  mark[i] = 0;  pi[i] = oo;  }  pi[u] = 0;  p[u] = -1;  for (int i = 1; i <= G->n - 1; i++) {  for (int v = 1; v <= G->n; v++){  if (G->A[u][v] != NO\_EDGE && !mark[v] && pi[u] + G->A[u][v] < pi[v]) {  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  mark[u] = 1;  int min\_pi = oo;  for (int j = 1; j <= G->n; j++){  if (!mark[j] && pi[j] < min\_pi) {  min\_pi = pi[j];  u = j;  }  }  }  }  int main(){  Graph G;  read\_graph(&G);  dijkstra(&G, 1);  printf("%d", pi[G.n]);  } |

**Câu 10: Viết chương trình bằng ngôn ngữ C, cho phép người dùng nhập vào một đồ thị vô hướng và in ma trận kề của nó ra màn hình. Biểu diễn đồ thị bằng phương pháp "Ma trận kề".**

- Đầu vào

Dữ liệu đầu vào được đọc từ dòng nhập chuẩn (stdin, bàn phím) theo định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m cách nhau một khoảng trắng, n: số đỉnh, m: số cung

m dòng tiếp theo, mỗi dòng chứa 2 số nguyên u v cách nhau 1 khoảng trắng mô tả cung (u, v).

- Đầu ra

In ma trận kề của đồ thị đã nhập, các phần tử cách nhau 1 khoảng trắng.

|  |  |
| --- | --- |
| #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;  for (int u = 1; u <= n; u++)  for (int v = 1; v <= n; v++)  pG->A[u][v] = 0;  }  void add\_edge(Graph \*pG, int u, int v) {  pG->A[u][v] += 1;  if (u != v)  pG->A[v][u] += 1;  } | /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");  }    return 0;  } |

**Câu 11: 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.**

- Luồng hợp lệ phải thoả mãn các yêu cầu sau:

0 <= F[u][v] <= C[u][v], với mọi cung (u, v)

Tổng luồng đi ra khỏi đỉnh s = Tổng luồng đi vào đỉnh t

Tổng luồng đi vào đỉnh u = Tổng luồng đi ra khỏi u (u khác s và u khác t)

- Luồng tầm thường (trivial flow) là luồng có F[u][v] = 0 với mọi cung (u,v). Dĩ nhiên luồng tầm thường là luồng hợp lệ.

- Giả sử người ta đã khởi tạo giá trị cho các luồng F[u][v] trên mạng, hãy viết chương trình kiểm tra xem luồng khởi tạo có hợp lệ không.

- Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u v c f mô tả cung (u, v) có khả năng thông qua c và luồng khởi tạo f.

- Đầu ra (Output):

Nếu luồng khởi tạo của mạng hợp lệ, in ra màn hình YES, ngược lại in ra NO.

|  |  |
| --- | --- |
| #include <stdio.h>  #include <stdbool.h>  #define MAX\_VERTICES 100  int main() {  int n, m;  // printf("Nhap so dinh va so cung: ");  scanf("%d %d", &n, &m);  // Khởi tạo ma trận C và F  int C[MAX\_VERTICES][MAX\_VERTICES] = {0}; // Ma trận khả năng thông qua  int F[MAX\_VERTICES][MAX\_VERTICES] = {0}; // Ma trận luồng khởi tạo  //printf("Nhap cac cung (u, v, c, f):\n");  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 tính hợp lệ của luồng khởi tạo  bool valid = true;  // Kiểm tra luồng qua từng cung  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 = false;  break;  }  }  } | // Kiểm tra tổng luồng đi ra khỏi s và tổng luồng đi vào t  int totalOutFlowS = 0, totalInFlowT = 0;  for (int i = 1; i <= n; i++) {  totalOutFlowS += F[1][i];  totalInFlowT += F[i][n];  }  if (totalOutFlowS != totalInFlowT) {  valid = false;  }  // Kiểm tra tổng luồng đi vào mỗi đỉnh (trừ s và t) bằng tổng luồng đi ra  for (int i = 2; i < n; i++) {  int totalInFlow = 0, totalOutFlow = 0;  for (int j = 1; j <= n; j++) {  totalInFlow += F[j][i];  totalOutFlow += F[i][j];  }  if (totalInFlow != totalOutFlow) {  valid = false;  break;  }  }  // In kết quả  if (valid) {  printf("YES\n");  } else {  printf("NO\n");  }  return 0;  } |

Câu 12: Viết chương trình đọc vào một đơn đồ thị vô hướng và kiểm tra xem nó có chứa chu trình hay không.

Nhắc lại: Chu trình là một đường đi có đỉnh đầu trùng với đỉnh cuối.

- Đầu vào (Input):

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số đỉnh và số cung.

m dòng tiếp theo mỗi dòng chứa 2 số nguyên u, v mô tả cung (u, v).

- Đầu ra (Output):

In ra màn hình CIRCLED nếu đồ thị có chứa chu trình, ngược lại in ra NO CIRCLE

Xem thêm ví dụ bên dưới.

|  |  |
| --- | --- |
| #include <stdio.h>  /\* Khai báo CTDL Graph\*/  #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;  for (int u = 1 ; u <= n; u++)  for (int v = 1 ; v <= n; v++)  pG->A[u][v] = 0;  }  void add\_edge(Graph \*pG, int u, int v) {  pG->A[u][v] += 1;  if (u != v)  pG->A[v][u] += 1;  if (pG->A[u][v] > 1)  printf("da cung (%d, %d)\n", u, v);  if (u == v)  printf("khuyen %d\n", u);  pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  #define WHITE 0  #define GRAY 1  #define BLACK 2  int color[MAX\_N];//Lưu trạng thái của các đỉnh  int has\_circle;//Đồ thị chứa trình hay không  void DFS(Graph \*pG, int u, int p) {  //1. Tô màu dang duyệt cho u  color[u] = GRAY;  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++)  if (adjacent(pG, u, v)) {  if (v == p) //2a. Nếu v == p  continue; //bỏ qua | int main() {  //1. Khai báo đồ thị G  Graph G;  //2. Đọc dữ liệu và dựng đồ thị  int n, m, u, v;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (int e = 0; e < m; e++) {  scanf("%d%d", &u, &v);  if (color[v] == WHITE) //2a. Nếu v chưa duyệt  DFS(pG, v, u); //gọi đệ quy duyệt nó  else if (color[v] == GRAY) //2b. v đang duyệt  has\_circle = 1; //chứa chu trình  }  //3. Tô màu duyệt xong cho u  color[u] = BLACK;  }    add\_edge(&G, u, v);  }  for (int u = 1; u <= G.n; u++)  color[u] = WHITE;  //2. Khởi tạo biến has\_circle  has\_circle = 0;  //3. Duyệt toàn bộ đồ thị để kiểm tra chu trình  for (int u = 1; u <= G.n; u++)  if (color[u] == WHITE) //u chưa duyệt  DFS(&G, u, -1); //gọi DFS(&G, u) để duyệt từ u  //4. Kiểm tra has\_circle  if (has\_circle)  printf("CIRCLED\n");  else  printf("NO CIRCLE\n");    return 0;  } |

Câu 13: Viết chương trình đọc một đơn đồ thị vô hướng, có trọng số không âm từ bàn phím và in ra chiều dài đường đi ngắn nhất từ đỉnh 1 đến đỉnh n.

- Đầu vào (Input)

Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m (1≤n<100;0≤m<500)(1≤𝑛<100;0≤𝑚<500)

m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w (0≤w≤100)(0≤𝑤≤100).

- Đầu ra (Output)

In ra màn hình chiều dài của đường đi ngắn nhất từ 1 đến n. Nếu không có đường đi từ 1 đến n, in ra -1

|  |  |
| --- | --- |
| #include <stdio.h>  #define MAXN 100  #define oo 999999  #define NO\_EDGE -1  typedef struct {  int n, m;  int W[MAXN][MAXN];  } 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++;  }  int mark[MAXN];  int pi[MAXN];  int p[MAXN];  void MooreDijkstra(Graph \*pG, int s) {  int u, v, it;  for (u = 1; u <= pG->n; u++) {  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0; //chiều dài đường đi ngắn nhất từ s đến chính nó bằng 0  p[s] = -1; //trước đỉnh s không có đỉnh nào cả  //Lặp n-1 lần  for (it = 1; it < pG->n; it++) {  //1. Tìm u có mark[u] == 0 và có pi[u] nhỏ nhất  int j, min\_pi = oo;  for (j = 1; j <= pG->n; j++)  if (mark[j] == 0 && pi[j] < min\_pi) {  min\_pi = pi[j];  u = j;  } | //2. Đánh dấu u đã xét  mark[u] = 1;  //3. Cập nhật pi và p của các đỉnh kề của v (nếu thoả)  for (v = 1; v <= pG->n; v++)  if (pG->W[u][v] != NO\_EDGE && 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;  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);  }    MooreDijkstra(&G, 1);  if (pi[n] < oo)  printf("%d\n", pi[n]);  else  printf("-1\n");  return 0;  } |

Câu 14: Tôn ngộ không

Giả sử số lượng cây ăn trái ở Hoa Quả Sơn là n cây và được đánh số từ 1 đến n.

- Dữ liệu đầu vào được nhập từ bàn phím với định dạng:

Dòng đầu tiên chứa 2 số nguyên n và m, tương ứng là số cây và số cặp cây có thể chuyền qua lại.

m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v, cách nhau 1 khoảng trắng, nói rằng có thể chuyền từ cây u sang cây v hoặc chuyền từ cây v sang cây u.

- Đầu ra (Output):

Nếu kế hoạch của Tôn Ngộ Không có thể thực hiện được DUOC, ngược lại in ra KHONG.

|  |  |
| --- | --- |
| #include<stdio.h>  int mark[100];  typedef struct{  int A[100][100];  int n,m;  }Graph;  typedef struct{  int data[100];  int size;  }List;  void init\_graph(Graph \* G,int n)  {  G->n = n;  for(int i = 1 ; i <= n; i++ )  for(int k = 1 ; k <= n; k++ )  G->A[i][k] = 0;  }  List neighbors(Graph G, int dinh){  List L;  L.size = 0;  for(int i = 1 ; i <= G.n; i++ )  {  if(G.A[dinh][i] == 1 && i != dinh)  {  L.data[L.size] = i;  L.size++;  }  }  return L;  }  void add\_edge(Graph \* G, int u, int v)  {  G->A[u][v] = G->A[v][u] = 1;  } | void DUNG\_CAY\_THEO\_CHIEU\_SAU(Graph G,int dinhBatDau)  {  if(mark[dinhBatDau] == 1)  return;  mark[dinhBatDau] = 1;    List nei = neighbors(G,dinhBatDau);  for(int i = 0; i < nei.size; i++)  {  DUNG\_CAY\_THEO\_CHIEU\_SAU(G,nei.data[i]);    }    }  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);  }  //Khoi tao mark  for(int i =1; i<= n ; i++)  {  mark[i] = 0;  }  int dinhBatDau = 1;  DUNG\_CAY\_THEO\_CHIEU\_SAU(G,dinhBatDau);  //Neu chua lien thong  for(int i = 1; i <= G.n; i++)  {  if(mark[i] == 0 )  {  printf("KHONG");  return 0;  }  }  printf("DUOC");  return 0;  } |