**Duyệt đồ thị**

|  |  |  |
| --- | --- | --- |
| Câu 1: Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.  Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.  Quy ước:  - Các đỉnh kề của 1 đỉnh được liệt kê theo thứ tự tăng dần | | |
| #include<stdio.h>  #define MAX\_ELEMENTS 100  #define MAX\_VERTEXES 100  typedef int ElementType;  typedef struct {  int data[MAX\_ELEMENTS];  int front,rear;  }Queue;  void make\_null\_queue(Queue\* Q)  {  Q->front = 0;  Q->rear = -1;  }  void push(Queue\* Q,int x)  {  Q->rear++;  Q->data[Q->rear] = x;  }  int top(Queue\* Q)  {  return Q->data[Q->front];  }  void pop(Queue\* Q)  {  Q->front++;    }  int empty(Queue\* Q)  {  return Q->front > Q->rear;  }  //List  typedef struct {  ElementType data[MAX\_ELEMENTS];  int size;  }List;  void make\_null(List\* L)  {  L->size = 0;  }  int count\_list(List \*L)  {  return L->size;  }  // | //Graph  typedef struct{  int A[100][100];  int n;  }Graph;  void add\_edge(Graph\* G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  }  //  void 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)  {  G->n = n;  for(int i =1; i<n;i++)  for(int j =1; j<n;j++)  G->A[i][j] = 0;  }  int adjacent(Graph\* G,int x,int y)  {  return G->A[x][y] == 1;  }  List neighbors(Graph\* G, int x){  List L;  make\_null(&L);  for(int y =1; y <= G->n; y++)  {  if(adjacent(G,x,y) && y != x)  {  push\_back(&L,y);  }  }  return L;    } | void breath\_first\_search(Graph\* G)  {  Queue frontier;  int mark[MAX\_VERTEXES];  make\_null\_queue(&frontier);    int j;  for(j=1; j <=G->n;j++)  mark[j] = 0;  push(&frontier,1);  mark[1] = 1;  while(!empty(&frontier))  {  int x = top(&frontier); pop(&frontier);  printf("%d\n",x);  List list = neighbors(G,x);  for(j = 1; j<=list.size;j++)  {  int y = element\_at(&list,j);  if(mark[y] ==0)  {  mark[y] = 1;  push(&frontier,y);  }  }  //Kiem tra neu chua lien thong  //Kiem tra dinh nao chua duoc duyet  if(empty(&frontier))  {  for(int i= 1; i <= G->n; i++)  {  if(mark[i] == 0){  mark[i] = 1;  push(&frontier,i);  }  }  }    }    }  int main()  {  Graph G;  int n, m, u, v, e;  // freopen("dothi.txt", "r", stdin);  scanf("%d%d", &n, &m);  init\_graph(&G, n);    for (e = 0; e < m; e++) {  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  }  breath\_first\_search(&G);    return 0;  } |

|  |  |  |
| --- | --- | --- |
| Câu 2: Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều sâu (sử dụng NGĂN XẾP) bắt đầu từ đỉnh 1.  Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.  Quy ước:  Các đỉnh kề của 1 đỉnh được liệt kê theo thứ tự tăng dần. | | |
| #include <stdio.h>  #define MAX\_VERTEXES 100  #define MAX\_ELEMENTS 100  #define MAX\_NODES 100  /\* Khai bao Stack\*/  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;  }  // khi bao do thi  typedef struct {  int n, m;  int A[MAX\_VERTEXES][MAX\_VERTEXES];  }Graph;  // khoi tao do thi  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;  }  // them cung  void add\_edge (Graph \*G, int x, int y) {  G->A[x][y] = 1;  G->A[y][x] = 1;  }  // kiem tra co ke  int adjacent (Graph \*G, int x, int y) {  return G->A[x][y] != 0;  } | /\* Khoi tao mark, chua dinh nào duoc xét \*/  int j;  for (j = 1; j <= G->n; j++)  //mark[j] = 0;  /\* Ðua 1 vào frontier \*/  push(&frontier, x);  // khai bao danh sach  typedef int ElementType;  typedef struct {  ElementType data [MAX\_ELEMENTS];  int size;  }List;  / tao rong  void make\_null(List \* L) {  L->size = 0;  }  // them phan tu  void push\_back(List \*L, ElementType x) {  L->data[L->size] = x;  L->size++;  }  // lay phan tu tai vi tri i, phan tu dat o bat ky dau  ElementType element\_at(List \* L, int i) {  return L->data[i-1];  }  // tra ve so phan tu danh sach  int count\_list(List \*L) {  return L->size;  }  // tim cac dinh ke  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];  /\* Duyet do thi theo chieu sau \*/  void depth\_first\_search(Graph\* G, int x) {  Stack frontier;  //int mark[MAX\_VERTEXES];  make\_null\_stack(&frontier); | /\* Khoi tao mark, chua dinh nào duoc xét \*/  int j;  for (j = 1; j <= G->n; j++)  //mark[j] = 0;  /\* Ðua 1 vào frontier \*/  push(&frontier, x);  /\* Vong lap chinh dung de duyet \*/  while (!empty(&frontier)) {  /\* Lay phan tu dau tiên trong frontier ra \*/  int x = top(&frontier); pop(&frontier);  if (mark[x] != 0)  continue;  printf("%d\n", x);  mark[x] = 1; //Ðánh d?u nó dã duy?t  /\* L?y các d?nh k? c?a nó \*/  List list = neighbors(G, x);  /\* Xét các d?nh k? c?a nó \*/  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;  // freopen("dothi.txt", "r", stdin);  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  //depth\_first\_search(&G);  }  depth\_first\_search(&G, 1);  for (w = 1; w <= n; w++)  if (mark[w] == 0)  depth\_first\_search(&G, w);  return 0;  } |

|  |  |  |  |
| --- | --- | --- | --- |
| Câu 3: Cho một đồ thị vô hướng đơn. Hãy in ra thứ tự của các đỉnh khi duyệt đồ thị theo chiều sâu (sử dụng ĐỆ QUY) bắt đầu từ đỉnh 1.  Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.  Quy ước:  Các đỉnh kề của 1 đỉnh được liệt kê theo thứ tự tăng dần. | | | |
| #include <stdio.h>  typedef int ElementType;  int mark[100];  //List  typedef struct {  int data[100];  int size;  }List;  void make\_null(List\* L)  {  L->size = 0;  }  void push\_back(List\*L,int x){  L->data[L->size] =x;  L->size++;  }  typedef struct{  int A[100][100];  int n;  }Graph;  void init\_graph(Graph\* G, int n)  {  G->n = n;  for(int i = 1; i <= n; i ++)  for(int k = 1; k <= n; k ++)  G->A[i][k] = 0;  } | List neighbors(Graph G,int dinh){  List list;  make\_null(&list);  for(int i =1; i <= G.n; i++)  {  if(i!=dinh && G.A[dinh][i] && G.A[i][dinh])  {  push\_back(&list,i);  }  }  return list;  }  void add\_edge(Graph \* G, int u, int v)  {  G->A[u][v] = G->A[v][u] = 1;  }  void depth\_first\_search(Graph \* G,int dinh){  if(mark[dinh] == 1)  return;  printf("%d\n",dinh);  mark[dinh] = 1;  List nei = neighbors(\*G,dinh);  for(int i = 0; i < nei.size; i++)  {  depth\_first\_search(G,nei.data[i]);  }    } | int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  }  //Khoi tao mark  for (e = 1; e <= n; e++) {  mark[e] = 0;  }  depth\_first\_search(&G,1);  //Neu do thi chua lien thong  for(int i = 1; i <= G.n;i++)  {  if(mark[i] == 0)  {  depth\_first\_search(&G,i);  break;  }  }  return 0;  } |

|  |  |  |
| --- | --- | --- |
| Câu 4: Có n hòn đảo và m cây cầu. Mỗi cây cầu bắt qua 2 hòn đảo. Một hôm chúa đảo tự hỏi là với các cây cầu hiện tại thì đứng ở một hòn đảo bất kỳ có thể nào đi đến được tất cả các hòn đảo khác hay không.  Hãy giúp chúa đảo viết chương trình kiểm tra.  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 đảo và số cây cầu.  - m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng có 1 cây cầu bắt qua hai hòn đảo u và v.  Đầu ra (Output):  Nếu có thể đi được in ra màn hình YES, ngược lại in ra NO. | | |
| #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);  }  1; | //Khoi tao mark  for(int i =1; i<= n ; i++)  {  mark[i] = 0;  }  int dinhBatDau =  DUNG\_CAY\_THEO\_CHIEU\_SAU(G,dinhBatDau);  //Neu chua lien thong  for(int i = 1; i <= G.n; i++)  {  if(mark[i] == 0 )  {  printf("NO");  return 0;  }  }  printf("YES");  return 0;  } |

|  |  |  |
| --- | --- | --- |
| Câu 6: Viết chương trình kiểm tra xem một đồ thị có hướng có liên thông mạnh 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 2 số nguyên u, v mô tả cung (u, v).  Chú ý: đồ thị không chứa đa cung.  Đầu ra (Output):  In ra màn hình strong connected nếu đồ thị đã cho liên thông mạnh, ngược lại in ra unconnected | | |
| #include<stdio.h>  int mark[100];  typedef struct{  int A[100][100];  int n;  }Graph;  typedef struct{  int data[100];  int size;  }List;  void DFS(Graph G,int n)  {  int i;  if(mark[n] == 1)  return;  mark[n] = 1;  //Get neighbors  List nei;  nei.size = 0;  for(i =1; i <= G.n; i ++ )  {  if(G.A[n][i] == 1 && i!=n)  { | nei.data[nei.size++] = i;  }  }    for(i =0; i < nei.size; i ++ )  {  DFS(G,nei.data[i]);  }  }  int main()  {  int i,k,n,m,u,v;  Graph G;    scanf("%d%d",&n,&m);  G.n= n;  for(i = 1; i <= n; i++ )  for(k = 1; k <= n; k++ )  G.A[i][k] = 0;    for(i = 1; i <= m; i ++ )  {    scanf("%d%d",&u,&v);  G.A[u][v] = 1;    } | for(int i = 1; i <=n ; i++)  {  for(k = 1; k <= n; k++ )  mark[k] = 0;    DFS(G,i);    for(k = 1; k <= n; k++ )  if (mark[k] == 0)  {  printf("unconnected");  return 0;  }  }  printf("strong connected");      return 0 ;  } |

|  |  |
| --- | --- |
| Câu 7: Viết chương trình đếm số bộ phận liên thông mạnh (BPLTM) của một đồ thị có hướng.  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).  Chú ý: đồ thị không chứa đa cung.  Đầu ra (Output):  In ra màn hình một con số nguyên duy nhất chỉ số BPTLM của đồ thị. | |
| #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  typedef int Vertices;  int n,m;//Bien toan cuc luu so dinh va so cung  //Khai bao cau truc do thi - GRAPH  typedef struct{  int matrix[maxv][maxv];  Vertices n;//  }Graph;  //Khai bao cau truc ngan xep - STACK  typedef struct{  int data[maxv];  int size;  }Stack;  //Lam rong  void makenullStack(Stack \*s){  s->size=0;  }  //Them phan tu vao ngap xep  void pushStack(Stack \*s, int element){  s->data[s->size] = element;  s->size++;  }  //Lay phan tu cua ngan xep  int getStack(Stack \*s){  return s->data[s->size-1];  }  void delSize(Stack \*s){  s->size--;  }  //Khai bao cau truc danh sach - LIST  typedef struct{  int data[idx];  int size;  }List;  //Lam rong  void makenullList(List \*l){  l->size=0;  }  //Them phan tu vao danh sach  void pushList(List \*l, int element){  l->data[l->size] = element;  l->size++;  }  //Truy cap phan tu trong danh sach  int getList(List \*l, int index){  return l->data[index];  }  //Khoi tao  void initGraph(Graph \*g){  g->n=n;  for(int i=1;i<=n;i++){  for(int j=1;j<=n;j++){  g->matrix[i][j]=0;  }  }  }  //Them cung  void addEdges(Graph \*g, int x, int y){  g->matrix[x][y]=1;  }  //In do thi  void printGraph(Graph g){  for(int i=1;i<=n;i++){  for(int j=1;j<=n;j++){  printf("%d ",g.matrix[i][j]);  }  printf("\n");  }  }  //Tim dinh lang gieng  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=n;i++){  if(g->matrix[x][i]==1){  pushList(&L,i);  }  }  return L;  } | //Cac bien mang toan cuc de thuc hien viec kiem tra  int count,k=1;  int num[maxv],min\_num[maxv],on\_stack[maxv];  //Ham tim gia tri nho nhat  int min(int a, int b){  return (a<b) ? a : b;  }  void strong\_connect(Graph \*g, int x){  //Cho num va min num bang chi so k  num[x] = min\_num[x] = k;  k++;  Stack s;  makenullStack(&s);  //Dua dinh dau tien vao stack  pushStack(&s,x);  //Danh dau dinh do da dua vao stack bang bien mang on\_stack  on\_stack[x]=1;  //Tim dinh lang gieng cua dinh do  List L = neighbors(g,x);  //Xet cac dinh ke cua x  for(int j=0;j<L.size;j++){  int v = getList(&L,j);  //Neu dinh ke x chua co trong stack  if(num[v]==-1){  //Goi de qui duyet dinh ke cua x  strong\_connect(g,v);  //Khi dut lui cap nhat lai min\_num cho dinh x(min\_num cha < min\_num con)  min\_num[x] = min(min\_num[x],min\_num[v]);  }  //Neu dinh v ke x da co trong stack  else if(on\_stack[v]==1){  //Cap nhat lai min num  min\_num[x] = min(min\_num[x],num[v]);  }  //Sau khi duyet xong 1 dinh ma vong lap for o tren dung lai tuc la ta da co 1 bo phan lien thong manh  //Neu do thi lien thong manh thi ket thuc luon chuong trinh vi do thi lien thong thi chi co 1 thanh phan lien thong  if(num[x]==min\_num[x]){  count++;//Bien count de diem so bo phan lien thong manh cua do thi  int value;  do{  //Lay dinh o dau stack ra  value = getStack(&s);  delSize(&s);  //danh dau dinh da lay ra khoi stack  on\_stack[x]=0;  }while(value!=x);//Lap den khi lay duoc dinh x(Dinh cat)  }  }  }  int main(){  Graph g;  int u,v;  scanf("%d%d",&n,&m);  initGraph(&g);  for(int i=1;i<=m;i++){  scanf("%d%d",&u,&v);  addEdges(&g,u,v);  }  count=0;//Khoi tao bien dem ban dau = 0 - day la bien toan cuc  for(int i=1;i<=n;i++){  num[i] = -1;  min\_num[i] = 0;  }  for(int i=1;i<=n;i++){  if(num[i]==-1){  strong\_connect(&g,i);  }  }  printf("%d",count);  return 0;  } |

|  |  |  |
| --- | --- | --- |
| Câu Cho G=<V, E> là một đồ thị vô hướng đơn (không có khuyên, không có đa cung). Hãy viết chương trình kiểm tra xem có chứa chu trình hay không.  Chu trình là một đường đi đơn cung 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 YES nếu đồ thị có chứa chu trình, ngược lại in ra NO | | |
| #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  int n,m;  typedef int Vertices;  typedef struct{  int matrix[maxv][maxv];  Vertices n;  }Graph;  typedef struct{  int data[idx];  int size;  }List;  void makenullList(List \*l){  l->size=0;  }  void pushList(List \*l, int element){  l->data[l->size]=element;  l->size++;  } | int getList(List \*l, int index){  return l->data[index-1];  }  void initGraph(Graph \*g){  g->n=n;  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  g->matrix[i][j]=0;  }  void addEdge(Graph \*g, int x, int y){  g->matrix[x][y]=1;  g->matrix[y][x]=1;  }  bool adjacent(Graph \*g, int x, int y){  return g->matrix[x][y]==1;  }  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=n;i++){  if(adjacent(g,x,i)){  pushList(&L,i);  }  }  return L;  }  int mark[maxv];  int key=0;  void DFS\_Recursion(Graph \*g, int x, int p){  mark[x]=1;  List L = neighbors(g,x); | for(int j=1;j<=L.size;j++){  int v = getList(&L,j);  if(v==p) continue;  if(mark[v]==1){  key=1;  return;  }  DFS\_Recursion(g,v,x);  }  }  bool check(Graph \*g){  for(int i=1;i<=n;i++)  mark[i]=0;  for(int i=1;i<=n;i++){  if(mark[i]==0){  DFS\_Recursion(g,i,0);  }  }  return key;  }  int main(){  Graph g;  scanf("%d%d",&n,&m);  initGraph(&g);  int u,v;  for(int i=1;i<=m;i++){  scanf("%d%d",&u,&v);  addEdge(&g,u,v);  }  if(check(&g)) printf("YES");  else printf("NO");  return 0;  } |

|  |  |
| --- | --- |
| Đường đi ngắn nhất  Cho đồ thị có 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.  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.  - 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 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>  #include <stdlib.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 500  #define INF 999999  typedef struct {  int u, v, w;  } Edge;  typedef struct {  int n, m;  int L[MAX\_VERTICES][MAX\_VERTICES];  } Graph;  void init\_graph(Graph\* G, int n) {  G->n = n;  G->m = 0;  int i, j;  for (i = 1; i <= n; i++) {  for (j = 1; j <= n; j++) {  G->L[i][j] = INF;  }  }  }  void add\_edge(Graph\* G, int u, int v, int w) {  G->L[u][v] = w;  G->m++;  }  int min(int a, int b) {  return a < b ? a : b;  }  int shortest\_path(Graph\* G, int n) {  int d[MAX\_VERTICES];  int visited[MAX\_VERTICES];  int i, j, u, v;  for (i = 1; i <= n; i++) {  d[i] = INF;  visited[i] = 0;  }  d[1] = 0; | if (d[n] == INF)  return -1;  else  return d[n];  }  or (j = 1; j <= n; j++) {  u = -1;  for (i = 1; i <= n; i++) {  if (!visited[i] && (u == -1 || d[i] < d[u]))  u = i;  }  if (d[u] == INF)  break;  visited[u] = 1;  for (v = 1; v <= n; v++) {  if (!visited[v] && G->L[u][v] != INF) {  d[v] = min(d[v], d[u] + G->L[u][v]);  }  }  }  int main() {  //freopen("dothi.txt", "r", stdin);  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%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  int shortest = shortest\_path(&G, n);  printf("%d\n", shortest);  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 s đến t.  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.  - m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.  - dòng cuối cùng chứa 2 số nguyên t và s (đường đi ngắn nhất từ s đến t)  Đầu ra (Output):  In ra màn hình chiều dài của đường đi ngắn nhất từ s đến t. Nếu không có đường đi từ s đến t, in ra -1. | |
| #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 500  #define INF 999999  typedef struct {  int u, v, w;  } Edge;  typedef struct {  int n, m;  int L[MAX\_VERTICES][MAX\_VERTICES];  } Graph;  void init\_graph(Graph\* G, int n) {  G->n = n;  G->m = 0;  int i, j;  for (i = 1; i <= n; i++) {  for (j = 1; j <= n; j++) {  G->L[i][j] = INF;  }  }  }  void add\_edge(Graph\* G, int u, int v, int w) {  G->L[u][v] = w;  G->L[v][u] = w; // Ð? th? vô hu?ng, c?n c?p nh?t c? 2 chi?u c?a ma tr?n tr?ng s?  G->m++;  }  int min(int a, int b) {  return a < b ? a : b;  }  int shortest\_path(Graph\* G, int s, int t) {  int d[MAX\_VERTICES];  int visited[MAX\_VERTICES];  int i, j, u, v;  for (i = 1; i <= G->n; i++) {  d[i] = INF;  visited[i] = 0;  }  d[s] = 0; | for (v = 1; v <= G->n; v++) {  if (!visited[v] && G->L[u][v] != INF) {  d[v] = min(d[v], d[u] + G->L[u][v]);  for (j = 1; j <= G->n; j++) {  u = -1;  for (i = 1; i <= G->n; i++) {  if (!visited[i] && (u == -1 || d[i] < d[u]))  u = i;  }  if (d[u] == INF)  break;  visited[u] = 1;    }  }  }  if (d[t] == INF)  return -1;  else  return d[t];  }  int main() {  //freopen("dothi.txt", "r", stdin);  Graph G;  int n, m, u, v, w, e, s, t;  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);  }  scanf("%d%d", &s, &t);  int shortest = shortest\_path(&G, s, t);  printf("%d\n", shortest);  return 0;  } |

|  |  |
| --- | --- |
| Viết chương trình kiểm tra một đồ thị có hướng (không có khuyên, không có đa cung) xem có chứa chu trình âm hay không.  Chu trình âm là chu trình có tổng trọng số các cung trong chu trình nhỏ hơn 0.  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, w mô tả cung (u, v) có trọng số w.  Đầu ra (Output):  In ra màn hình negative cycle nếu đồ thị có chứa chu trình âm, ngược lại in ra ok  Xem thêm ví dụ bên dưới. | |
| #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 500  #define INF 999999  typedef struct {  int u, v, w;  } Edge;  typedef struct {  int n, m;  Edge edges[MAX\_EDGES];  } Graph;  void init\_graph(Graph\* G, int n) {  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++;  }  int BellmanFord(Graph\* G) {  int d[MAX\_VERTICES];//khai bao 1 mang d co kich thuoc la Max\_V de luu tru duong di ngan nhat xuat phat den cac dinh khac  int i, j;  for (i = 1; i <= G->n; i++) {  d[i] = INF;//khoi tao all cac gia tr? trong mang d ban dau la INF, tuc la vo cung, ngoai tru dinh xuat phat(thuong la 1) co gia tri la 0  }  d[1] = 0;//gan gia tri 0 cho dinh xuat phat trong mang d  for (i = 1; i <= G->n - 1; i++) {//n la dinh  for (j = 0; j < G->m; j++) {//duyet qua all cac cung trong do thi  int u = G->edges[j].u;//lay dinh dau tien cua cung thu j  int v = G->edges[j].v;//lay ra dinh ket thuc cua cung thu j  int w = G->edges[j].w;//lay ra trong so cua cung thu j  if (d[u] != INF && d[u] + w < d[v]) {  d[v] = d[u] + w;  }  }  } | \*/  if (d[u] != INF && d[u] + w < d[v]) {/\*N?u ta v?n có th? c?p nh?t giá tr? d[v] thêm m?t l?n n?a, di?u này có nghia là có chu trình âm trong d? th?.  Trong tru?ng h?p này, ta tr? v? 1 d? cho bi?t r?ng có chu trình âm.  \*/  return 1; // Có chu trình âm  }  }  return 0; // Không có chu trình âm  }  for (j = 0; j < G->m; j++) {//kt xem co chu trinh âm trong do thi hay ko  int u = G->edges[j].u;  int v = G->edges[j].v;  int w = G->edges[j].w;  /\*d[u]: Ðây là giá tr? kho?ng cách t?t nh?t t? d?nh xu?t phát d?n d?nh u  d[v]: Ðây là giá tr? kho?ng cách t?t nh?t t? d?nh xu?t phát d?n d?nh v  Trong dòng l?nh if (d[u] != INF && d[u] + w < d[v]), ta ki?m tra xem có th? c?p nh?t giá tr? d[v] d? d?t du?c du?ng di ng?n nh?t t? d?nh xu?t phát d?n v thông qua u hay không.  int main() {  freopen("dt.txt", "r", stdin);  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%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  if (BellmanFord(&G)) {  printf("negative cycle\n");  } else {  printf("ok\n");  }  return 0;  } |

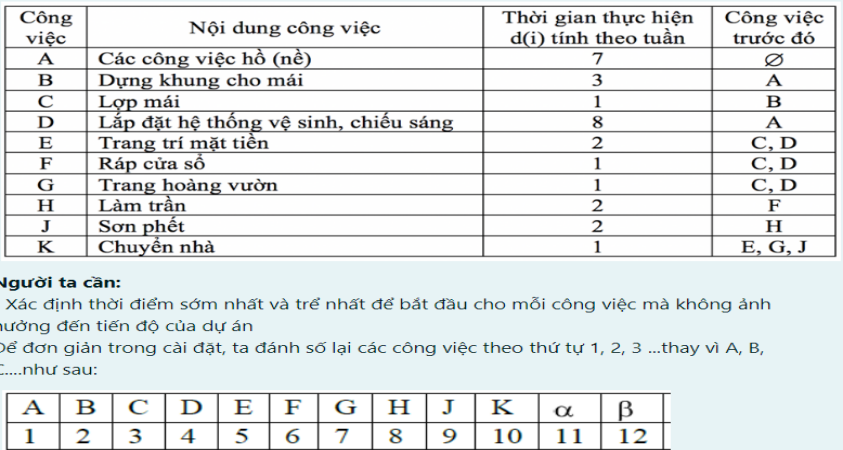
|  |  |
| --- | --- |
| Mê cung số (number maze) | |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #define MAX\_ROWS 100  #define MAX\_COLS 100  // C?u trúc bi?u di?n d? th?  typedef struct {  int rows;  int cols;  int maze[MAX\_ROWS][MAX\_COLS]; // Mê cung s?  } Maze;  // Hàm tìm du?ng di ng?n nh?t t? góc trên bên trái d?n góc du?i bên ph?i  int dijkstra(Maze\* maze) {  int dist[MAX\_ROWS][MAX\_COLS];  int visited[MAX\_ROWS][MAX\_COLS] = {{0}};  // Kh?i t?o kho?ng cách  for (int i = 0; i < maze->rows; i++) {  for (int j = 0; j < maze->cols; j++) {  dist[i][j] = INT\_MAX;  }  }  dist[0][0] = maze->maze[0][0];  for (int count = 0; count < maze->rows \* maze->cols - 1; count++) {  // Tìm ô có kho?ng cách nh? nh?t chua du?c tham  int min\_dist = INT\_MAX;  int u, v;  for (int i = 0; i < maze->rows; i++) {  for (int j = 0; j < maze->cols; j++) {  if (!visited[i][j] && dist[i][j] < min\_dist) {  min\_dist = dist[i][j];  u = i;  v = j;  }  }  } | // Ðánh d?u ô dã du?c tham  visited[u][v] = 1;  // C?p nh?t kho?ng cách d?n các ô k? c?a ô u, v  // Ði lên  if (u > 0 && dist[u][v] + maze->maze[u - 1][v] < dist[u - 1][v]) {  dist[u - 1][v] = dist[u][v] + maze->maze[u - 1][v];  }  // Ði xu?ng  if (u < maze->rows - 1 && dist[u][v] + maze->maze[u + 1][v] < dist[u + 1][v]) {  dist[u + 1][v] = dist[u][v] + maze->maze[u + 1][v];  }  // Ði qua trái  if (v > 0 && dist[u][v] + maze->maze[u][v - 1] < dist[u][v - 1]) {  dist[u][v - 1] = dist[u][v] + maze->maze[u][v - 1];  }  // Ði qua ph?i  if (v < maze->cols - 1 && dist[u][v] + maze->maze[u][v + 1] < dist[u][v + 1]) {  dist[u][v + 1] = dist[u][v] + maze->maze[u][v + 1];  }  }  return dist[maze->rows - 1][maze->cols - 1];  }  int main() {  Maze maze;  scanf("%d%d", &maze.rows, &maze.cols);  // Nh?p mê cung s?  for (int i = 0; i < maze.rows; i++) {  for (int j = 0; j < maze.cols; j++) {  scanf("%d", &maze.maze[i][j]);  }  }  // Tìm du?ng di ng?n nh?t và in ra k?t qu?  int min\_cost = dijkstra(&maze);  printf("%d\n", min\_cost);  return 0;  } |

|  |  |
| --- | --- |
| Áp dụng giải thuật Bellman – Ford kiểm tra xem một đồ thị có hướng có chứa chu trình âm hay không, nếu ta tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại. In kết quả YES (nếu đồ thị có chu trình âm) hoặc NO (trường hợp ngược lại).  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 nói rằng cung (u, v) có trọng số w.  - Dòng cuối cùng chứa đỉnh s.  Đầu ra (Output):  In ra màn hình YES nếu phát hiện chu trình âm, ngược lại in ra NO. | |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 500  #define INF INT\_MAX  // C?u trúc bi?u di?n cung  typedef struct {  int u, v, w; // Ð?nh d?u, d?nh cu?i và tr?ng s? c?a cung  } Edge;  // C?u trúc bi?u di?n d? th?  typedef struct {  int n, m; // S? d?nh và s? cung  Edge edges[MAX\_EDGES]; // Danh sách các cung  } Graph;  // Kh?i t?o d? th?  void init\_graph(Graph \*G, int n) {  G->n = n;  G->m = 0;  }  // Thêm cung vào d? th?  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++;  }  // Thu?t toán Bellman-Ford ki?m tra chu trình âm  int bellman\_ford(Graph \*G, int s) {  int d[MAX\_VERTICES]; // M?ng luu tr? kho?ng cách ng?n nh?t t? d?nh s d?n các d?nh khác  // Kh?i t?o kho?ng cách  for (int i = 1; i <= G->n; i++) {  d[i] = INF;  }  d[s] = 0;  // Duy?t qua t?t c? các cung và c?p nh?t kho?ng cách n?u có du?ng di ng?n hon | for (int i = 1; i <= G->n - 1; i++) {  for (int j = 0; j < G->m; j++) {  int u = G->edges[j].u;  int v = G->edges[j].v;  int w = G->edges[j].w;  if (d[u] != INF && d[u] + w < d[v]) {  d[v] = d[u] + w;  }  }  }    // Ki?m tra xem có chu trình âm không  for (int j = 0; j < G->m; j++) {  int u = G->edges[j].u;  int v = G->edges[j].v;  int w = G->edges[j].w;  if (d[u] != INF && d[u] + w < d[v]) {  return 1; // Có chu trình âm  }  }  return 0; // Không có chu trình âm  }  int main() {  Graph G;  int n, m, u, v, w, s;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    for (int e = 0; e < m; e++) {  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  scanf("%d", &s);  if (bellman\_ford(&G, s)) {  printf("YES\n");  } else {  printf("NO\n");  }  return 0;  } |

|  |  |  |
| --- | --- | --- |
| Xếp hạng đồ thị  Cô giáo Trang chuẩn bị kẹo để phát cho các bé mà cô đang giữ.  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 bé và số cặp bé mà trong đó có 1 bé muốn có kẹo hơn bạn mình..  - m dòng tiếp theo mỗi dòng chứa 2 số nguyên a, b nói rằng bé a muốn có kẹo nhiều hơn bé b.  Đầu ra (Output):  In ra màn hình số kẹo ít nhất của từng em, mỗi em trên một dòng.  Dòng cuối cùng in tổng số kẹo ít nhất mà cô giáo Trang cần phải chuẩn bị | | |
| #include <stdio.h>  #define MAX\_VERTICES 100  #define MAX\_ELEMENTS 100  typedef int ElementType;  int rank[MAX\_VERTICES];  int d[MAX\_VERTICES];  //khai bao cau truc list  typedef struct {  ElementType data[MAX\_ELEMENTS];  int size;  } List;  //khoi bao cau truc do thi  typedef struct {  int A[MAX\_VERTICES][MAX\_VERTICES];  int n,m;  }Graph;  void make\_null\_list(List\* L) {  L->size = 0;  }  /\* Them mot phan tu vao cuoi danh sach \*/  void push\_back(List\* L, ElementType x) {  L->data[L->size] = x;  L->size++;  }  /\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/  ElementType element\_at(List\* L, int i) {  return L->data[i-1];  }  /\* Tra ve so phan tu cua danh sach \*/  int count\_list(List\* L) {  return L->size;  }  /\* phan do thi \*/  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[y][x]=1;  }  int adjacent(Graph \*G, int x, int y){  return G->A[x][y] != 0;  }  int degree(Graph \*G,int x){  int y,deg=0;  for(y=1; y<= G->n; y++)  deg+= G->A[x][y];  return deg;  }  List neighbors(Graph \*G, int x){  int y;  List list;  make\_null\_list(&list);  for(y=1;y<=G->n;y++)  if(adjacent(G,x,y))  push\_back(&list,y);  return list;  }  void copy\_list(List \*S1, List \*S2){  int i, x;  make\_null\_list(S1);  for(i=1;i<=S2->size;i++){  x=element\_at(S2,i);  push\_back(S1,x);  }  }  int k=1;  List S1, S2;  void ranking(Graph \*G){  int x, u;  for(u = 1; u <= G->n; u++){  d[u] = 0;  // rank[u] = 0;  }  for(x = 1; x <= G->n; x++)  for(u = 1; u <= G->n; u++)  if(G->A[x][u] != 0)  d[u]++;  // d[1]=0; | // List S1, S2;  make\_null\_list(&S1);  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&S1, u);  // int k = 1, i;  int i;  while(S1.size > 0){  make\_null\_list(&S2);  for(i = 1; i <= S1.size; i++){  int u = element\_at(&S1, i);  rank[u] = k;  int v;  for (v = 1; v <= G->n; v++)  if(G->A[u][v] != 0){  d[v]--;  if(d[v] == 0)  push\_back(&S2, v);  }  }  copy\_list(&S1, &S2);  k++;  }  }  int main (){  // freopen("dt.txt", "r", stdin);  Graph G;  int n, m, a, b, e;  int sum=0;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    for (e = 0; e < m; e++) {  scanf("%d%d", &a, &b);  add\_edge(&G, a, b);  }    ranking(&G);  for(a=1;a<=n;a++){  printf("%d \n",rank[a]);  sum =sum + rank[a];  }  printf("%d",sum);  return 0;  } |

|  |  |  |
| --- | --- | --- |
| Viết chương trình xếp hạng cho đồ thị có hướng không chu trình.  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 hạng của các đỉnh theo thứ tự của đỉnh, mỗi đỉnh trên 1 dòng:  Hạng đỉnh 1 | | |
| #include <stdio.h>  #define MAX\_VERTICES 100  #define MAX\_ELEMENTS 100  typedef int ElementType;  int rank[MAX\_VERTICES];  int d[MAX\_VERTICES];  //khai bao cau truc list  typedef struct {  ElementType data[MAX\_ELEMENTS];  int size;  } List;  //khoi bao cau truc do thi  typedef struct {  int A[MAX\_VERTICES][MAX\_VERTICES];  int n,m;  }Graph;  void make\_null\_list(List\* L) {  L->size = 0;  }  /\* Them mot phan tu vao cuoi danh sach \*/  void push\_back(List\* L, ElementType x) {  L->data[L->size] = x;  L->size++;  }  /\* Lay phan tu tai vi tri i, phan tu bat dau o vi tri 1 \*/  ElementType element\_at(List\* L, int i) {  return L->data[i-1];  }  /\* Tra ve so phan tu cua danh sach \*/  int count\_list(List\* L) {  return L->size;  }  /\* phan do thi \*/  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;  } | int i, x;  make\_null\_list(S1);  for(i=1;i<=S2->size;i++){  x=element\_at(S2,i);  push\_back(S1,x);  }  }  int k=0;  List S1, S2;  void ranking(Graph \*G){  int x, u;  for(u = 1; u <= G->n; u++){  d[u] = 0;  // rank[u] = 0;  }  for(x = 1; x <= G->n; x++)  for(u = 1; u <= G->n; u++)  if(G->A[x][u] != 0)  d[u]++;  // d[1]=0;  // List S1, S2;    make\_null\_list(&S1);  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&S1, u);  // int k = 1, i;  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;  }  int degree(Graph \*G,int x){  int y,deg=0;  for(y=1; y<= G->n; y++)  deg+= G->A[x][y];  return deg;  }  List neighbors(Graph \*G, int x){  int y; | List list;  make\_null\_list(&list);  for(y=1;y<=G->n;y++)  if(adjacent(G,x,y))  push\_back(&list,y);  return list;  }  void copy\_list(List \*S1, List \*S2){  int i;  while(S1.size > 0){  make\_null\_list(&S2);  for(i = 1; i <= S1.size; i++){  int u = element\_at(&S1, i);  rank[u] = k;  int v;  for (v = 1; v <= G->n; v++)  if(G->A[u][v] != 0){  d[v]--;  if(d[v] == 0)  push\_back(&S2, v);  }  }  copy\_list(&S1, &S2);  k++;  }  }  int main (){  // freopen("dt.txt", "r", stdin);  Graph G;  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);  add\_edge(&G, u, v);  }  ranking(&G);  for(u=1;u<=n;u++)  printf("%d \n",rank[u]);  return 0;  } |

|  |  |
| --- | --- |
| 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 hòn đó và số lần cân  - m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng hòn đá u nhẹ hơn hòn đá v.  Đầu ra (Output):  In ra màn hình thứ tự của các hòn đá theo khối lượng tăng dần. In các số thứ tự trên cùng một dòng, mỗi số cách nhau một khoảng trắng.  Bạn có thể yên tâm là dữ liệu đầu được giả sử rằng chỉ có một kết quả quả duy nhất.  Xem thêm ví dụ bên dưới. Trong ví dụ đầu tiên ta có: hòn đá 1 nhẹ nhất, kế đến là hòn đá 3 và sau cùng là hòn đá 2. | |
| #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  int n,m; //Bi?n toàn c?c n d?nh và m cung  typedef int Vertices;  //Khai báo c?u trúc d? th?  typedef struct{  int matrix[maxv][maxv];  Vertices n;//Ð?nh n  }Graph;  //Khai báo c?u trúc danh sách  typedef struct{  int data[idx];  int size;  }List;  /\*Các hàm trên danh sách\*/  //Làm r?ng  void makenullList(List \*l){  l->size=0;  }  //Ki?m tra danh sách có r?ng hay không  bool emptyList(List \*l){  return (l->size==0);  }  //Thêm m?t ph?n t? vào danh sách  void pushList(List \*l, int element){  l->data[l->size] = element;  l->size++;  }  //Truy c?p m?t ph?n t? trong danh sách b?t d?u t? v? trí th? 1  int getList(List \*l, int index){  return l->data[index-1];  }  /\*Các hàm thao tác trên d? th?\*/  void initGraph(Graph \*g){  g->n=n;  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  g->matrix[i][j]=0;  }  //Thêm cung e = (x,y) vào d? th? có hu?ng  void addEdge(Graph \*g, int x, int y){  g->matrix[x][y]=1;  }  //Ki?m tra hai d?nh có k? nhau hay không  bool adjacent(Graph \*g, int x, int y){  return g->matrix[x][y]==1;  }  //Tính s? b?c vào c?a 1 d?nh trong d? th? - b?c vào là b?c có mui tên di vào d?nh  int degreeInput(Graph \*g, int x){  int count=0;  for(int i=1;i<=n;i++){  if(adjacent(g,i,x)){  count++;  }  }  return count;  }  //Tìm d?nh láng gi?ng c?a m?t d?nh  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=n;i++){  if(adjacent(g,x,i)){  pushList(&L,i);  }  }  return L;  } | //Sao chép ph?n t? c?a danh sách này sang danh sách khác  void copyList(List \*l1, List \*l2){  makenullList(l1);  for(int i=1;i<=l2->size;i++){  int value = getList(l2,i);  pushList(l1,value);  }  }  //Các bi?n toàn c?c d? ph?c v? cho gi?i thu?t x?p h?ng d? th?  int rank[maxv]; //M?ng rank[] d? luu h?ng c?a t?t c? các d?nh trong d? th?  int d[maxv]; //M?ng d-Degree d? luu s? b?c vào c?a t?t c? các d?nh trong d? th?  List L1,L2; //Danh sách L1 luu nh?ng d?nh có b?c vào b?ng 0(g?c cu), danh sách L2 luu nh?ng d?nh k? v c?a u có b?c vào b?ng 0(g?c m?i)  int k=0; //Bu?c l?p c?a gi?i thu?t, h?ng c?a d?nh s? du?c gán cho bu?c l?p sau khi x?p h?ng 1 d?nh trong d? th? xong  //D?a trên gi?i thu?t ranking d? tìm th? t? topo  List topoSort(Graph \*g){  List toposort;  makenullList(&toposort);  int u;  for(u=1;u<=n;u++) d[u]=0; //Kh?i t?o các d?nh có b?c vào di?u b?ng 0  for(u=1;u<=n;u++) d[u] = degreeInput(g,u); //Tính b?c vào cho các d?nh r?i luu vào m?ng d[]  makenullList(&L1); //Kh?i t?o danh sách L1 luu nh?ng d?nh g?c(d?nh có b?c vào b?ng 0)  for(u=1;u<=n;u++) if(d[u]==0) pushList(&L1,u); //Nh?ng d?nh nào là d?nh g?c s? du?c cho vào danh sách L1(G?c cu)  while(!emptyList(&L1)){ //Duy?t trong khi L1 chua r?ng  makenullList(&L2); //Kh?i t?o danh sách L2 d? ch?a các d?nh k? v c?a u có b?c vào b?ng 0(G?c m?i)  for(int i=1;i<=L1.size;i++){ //Duy?t các d?nh g?c v?a du?c thêm trong danh sách L1  int u = getList(&L1,i); //L?y d?nh g?c ra  rank[u] = k; //X?p h?ng cho d?nh g?c b?ng bu?c l?p k  pushList(&toposort,u); //Ðua d?nh dó vào danh sách topo sau khi dã x?p h?ng xong  List L = neighbors(g,u); //Tìm d?nh k? c?a d?nh v?a du?c x?p h?ng  for(int j=1;j<=L.size;j++){ //Duy?t các d?nh k?  int v = getList(&L,j); //L?y d?nh k?  d[v]--; //Gi?m b?c d?nh k? di 1 tuong duong v?i vi?c xóa d?nh g?c u(g?c m?i) ra kh?i d? th?  if(d[v]==0){ //N?u d?nh k? v là d?nh g?c(g?c m?i)  pushList(&L2,v); //Ðua d?nh g?c m?i vào danh sách L2  }  }  }  copyList(&L1,&L2);//Sao chép các d?nh g?c m?i trong L2 sang L1  k++;//Tang bu?c l?p lên d? x?p h?ng cho các d?nh ti?p theo trong danh sách L1  //Khi nào danh sách L2 r?ng t?c d?nh g?c cu không có d?nh k? thì hi?n nhiên -> Gi?i thu?t k?t thúc -> vì lúc này L1 dã r?ng  }  return toposort;  }  int main(){  Graph g;  int u,v;  scanf("%d%d",&n,&m);  initGraph(&g);  for(int i=1;i<=m;i++){  scanf("%d%d",&u,&v);  addEdge(&g,u,v);  }  List toposort = topoSort(&g);  //In th? t? topo có trong danh sách toposort  for(int u=1;u<=toposort.size;u++){  printf("%d ",getList(&toposort,u));  }  return 0;  } |



|  |  |
| --- | --- |
| //code  #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  #define oo 9999999  typedef int Vertices;  //Khai báo cấu trúc đồ thị  typedef struct{  int matrix[maxv][maxv];  Vertices n;//Đỉnh n  }Graph;  //Khai báo cấu trúc danh sách  typedef struct{  int data[idx];  int size;  }List;  /\*Các hàm trên danh sách\*/  //Làm rỗng  void makenullList(List \*l){  l->size=0;  }  //Kiểm tra danh sách có rỗng hay không  bool emptyList(List \*l){  return (l->size==0);  }  //Thêm một phần tử vào danh sách  void pushList(List \*l, int element){  l->data[l->size] = element;  l->size++;  }  //Truy cập một phần tử trong danh sách bắt đầu từ vị trí thứ 1  int getList(List \*l, int index){  return l->data[index-1];  }  /\*Các hàm thao tác trên đồ thị\*/  void initGraph(Graph \*g, int n){  g->n=n;  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  g->matrix[i][j]=0;  }  //Thêm cung e = (x,y) vào đồ thị có hướng  void addEdge(Graph \*g, int x, int y){  g->matrix[x][y]=1;  }  //Kiểm tra hai đỉnh có kề nhau hay không  bool adjacent(Graph \*g, int x, int y){  return g->matrix[x][y]==1;  }  //Tính số bậc vào của 1 đỉnh trong đồ thị - bậc vào là bậc có mũi tên đi vào đỉnh  int degreeInput(Graph \*g, int x){  int count=0;  for(int i=1;i<=g->n;i++){  if(adjacent(g,i,x)){  count++;  }  }  return count;  }  //Tính số bậc ra của 1 đỉnh trong đồ thị - Bậc ra là số mũi tên đi ra của đỉnh đang xét đến một đỉnh khác  int degreeOutput(Graph \*g, int x){  int count=0;  for(int i=1;i<=g->n;i++){  if(adjacent(g,x,i)){  count++;  }  }  return count;  }  //Tìm đỉnh láng giềng của một đỉnh  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=g->n;i++){  if(adjacent(g,x,i)){  pushList(&L,i);  }  }  return L;  }  //Sao chép phần tử của danh sách này sang danh sách khác  void copyList(List \*l1, List \*l2){  makenullList(l1);  for(int i=1;i<=l2->size;i++){  int value = getList(l2,i);  pushList(l1,value);  }  }  //Các biến toàn cục để phục vụ cho giải thuật xếp hạng đồ thị  int rank[maxv]; //Mảng rank[] để lưu hạng của tất cả các đỉnh trong đồ thị  int d[maxv]; //Mảng d-Degree để lưu số bậc vào của tất cả các đỉnh trong đồ thị  List L1,L2; //Danh sách L1 lưu những đỉnh có bậc vào bằng 0(gốc cũ), danh sách L2 lưu những đỉnh kề v của u có bậc vào bằng 0(gốc mới)  int k=0; //Bước lặp của giải thuật, hạng của đỉnh sẽ được gán cho bước lặp sau khi xếp hạng 1 đỉnh trong đồ thị xong | //Dựa trên giải thuật ranking để tìm thứ tự topo  List topoSort(Graph \*g){  List toposort;  makenullList(&toposort);  int u,n=g->n;  for(u=1;u<=n;u++) d[u]=0; //Khởi tạo các đỉnh có bậc vào điều bằng 0  for(u=1;u<=n;u++) d[u] = degreeInput(g,u); //Tính bậc vào cho các đỉnh rồi lưu vào mảng d[]  makenullList(&L1); //Khởi tạo danh sách L1 lưu những đỉnh gốc(đỉnh có bậc vào bằng 0)  for(u=1;u<=n;u++) if(d[u]==0) pushList(&L1,u); //Những đỉnh nào là đỉnh gốc sẽ được cho vào danh sách L1(Gốc cũ)  while(!emptyList(&L1)){ //Duyệt trong khi L1 chưa rỗng  makenullList(&L2); //Khởi tạo danh sách L2 để chứa các đỉnh kề v của u có bậc vào bằng 0(Gốc mới)  for(int i=1;i<=L1.size;i++){ //Duyệt các đỉnh gốc vừa được thêm trong danh sách L1  int u = getList(&L1,i); //Lấy đỉnh gốc ra  rank[u] = k; //Xếp hạng cho đỉnh gốc bằng bước lặp k  pushList(&toposort,u); //Đưa đỉnh đó vào danh sách topo sau khi đã xếp hạng xong  List L = neighbors(g,u); //Tìm đỉnh kề của đỉnh vừa được xếp hạng  for(int j=1;j<=L.size;j++){ //Duyệt các đỉnh kề  int v = getList(&L,j); //Lấy đỉnh kề  d[v]--; //Giảm bậc đỉnh kề đi 1 tương đương với việc xóa đỉnh gốc u(gốc mới) ra khỏi đồ thị  if(d[v]==0){ //Nếu đỉnh kề v là đỉnh gốc(gốc mới)  pushList(&L2,v); //Đưa đỉnh gốc mới vào danh sách L2  }  }    }  copyList(&L1,&L2);//Sao chép các đỉnh gốc mới trong L2 sang L1  k++;//Tăng bước lặp lên để xếp hạng cho các đỉnh tiếp theo trong danh sách L1  //Khi nào danh sách L2 rỗng tức đỉnh gốc cũ không có đỉnh kề thì hiển nhiên -> Giải thuật kết thúc -> vì lúc này L1 đã rỗng  }  return toposort;  }  //Hàm tìm min  int min(int a, int b){  return (a<b)?a:b;  }  //Hàm tìm max  int max(int a, int b){  return (a>b)?a:b;  }  int main(){  Graph g;  int n, u, x;  scanf("%d", &n);  initGraph(&g, n+2); //Đồ thị lúc này sẽ được thêm 2 đỉnh nữa là đỉnh alpha và beta  int time[100]; //Mảng time lưu thời gian để thực hiện công việc  time[n+1] = 0; // d[alpha] = 0  for (u=1;u<=n;u++) {  scanf("%d",&time[u]); //Nhập thời gian hoàn thành công việc thứ u (u=1,2,3,...)  do{  scanf("%d", &x); //Nhập danh sách công việc trước đó phải làm của công việc u  if(x>0){ //Nếu không phải đỉnh alpha và beta  addEdge(&g, x, u); //Thì thêm công việc trước đó của công việc u - thêm cung vào đồ thị  }  }while(x>0);//Nếu không có công việc trước đó hoặc nhập hết công việc trước đó thì dừng  }    //Thêm đỉnh alpha vào đồ thị  for(int i=1;i<=n;i++){  if(degreeInput(&g,i)==0){  addEdge(&g,n+1,i);//Nối đỉnh alpha vào đỉnh bắt đầu  }  }    //Thêm đỉnh beta vào đỉnh cuối  for(int i=1;i<=n;i++){  if(degreeOutput(&g,i)==0){  addEdge(&g,i,n+2);//Nối đỉnh beta vào đỉnh cuối  }  }  //Muốn tìm thời gian trễ nhất và sớm nhất để bắt đầu công việc thì phải dựa trên thứ tự topo  List L = topoSort(&g);    int t[100];//Mảng lưu thời gian sớm nhất để bắt đầu công việc  int T[100];//Mảng lưu thời gian trễ nhất để bắt đầu công việc    t[n+1] = 0; //Khởi tạo t[alpha] = 0  for (int j=2;j<=L.size;j++){ //Bắt đầu từ đỉnh thứ 2 vì đỉnh thứ 1 đã là đỉnh alpha  int u = getList(&L,j); //Lấy đỉnh trong danh sách topo ra  t[u] = -1; //Gán t[u] bằng số nào đó > 0  for (x=1;x<=g.n;x++){  if (adjacent(&g,x,u)){ // Nếu u kề xx  t[u] = max(t[u], t[x] + time[x]); //Tính thời gian sớm nhất hoàn thành công việc  }  }  }    //Tìm thời điểm trễ nhất  T[n+2] = t[n+2]; //Khởi tạo T[beta] = t[beta]  for (int j=L.size-1;j>=1;j--) { //Bắt đầu bằng đỉnh trước đỉnh cuối vì đỉnh cuối đang là đỉnh beta  int u = getList(&L, j); //Lấy đỉnh trong danh sách topo ra  T[u] = oo; // Gán T[u] bằng vô cực  for (int v = 1; v <= g.n; v++){  if (adjacent(&g,u,v)){ //Nếu v kể u  T[u] = min(T[u], T[v] - time[u]); //Tính thời gian trễ nhất hoàn thành công việc  }  }  }    //In tổng số tuần sớm nhất để hoàn thành công trình -> t[beta];  printf("%d\n",t[n+2]);    //In thời gian sớm nhất và trễ nhất để bắt đầu công việc  for(u=1;u<=g.n;u++) printf("%d-%d\n",t[u],T[u]);    return 0;  } |

**Duyệt đồ thị tiếp theo**

|  |  |
| --- | --- |
| Cho một đồ thị vô hướng đơn. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều rộng bắt đầu từ đỉnh 1.  Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.  Quy ước:  Các đỉnh kề của 1 đỉnh được liệt kê theo thứ tự tăng dần | |
| #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  int n,m;  typedef int Vertices;  typedef struct{  int matrix[maxv][maxv];  Vertices n;  }Graph;  //khai bao cau truc - LIST  typedef struct{  int data[idx];  int size;  }List;  //khai bao cau truc - QUEUE  typedef struct{  int front;  int rear;  int data[maxv];  }Queue;  //hang doi - Queue  //lam rong han doi  void makenullQueue(Queue \*q){  q->front=0;  q->rear=-1;  }  // kt queue co rong ko  bool emptyQueue(Queue \*q){  return q->front > q->rear;  }  // them phtu vao hang doi  void pushQueue(Queue \*q, int element){  q->rear++;  q->data[q->rear]=element;  }  //truy cap phtu trong hang doi  int getQueue(Queue \*q){  return q->data[q->front];  }  // xoa vi tri phtu trong hang doi di ko can dung nua  void plusFront(Queue \*q){  q->front++;  }  //lam rong ds  void makenullList(List \*l){  l->size=0;  }  //kt co rong ko  bool emptyList(List \*l){  return l->size==0;  }  // them phtu vao ds  void pushList(List \*l, int element){  l->data[l->size]=element;  l->size++;  }  // truy cap phtu trong ds bat dau tu vi tri 1(mang bat dau bang 0 nên index phai tru di 1)  int getList(List \*l, int index){  return l->data[index-1];  }  //do thi  // khoi tao do thi  void initGraph(Graph \*g){  g->n=n;  for(int i=1;i<=n;i++){  for(int j=1;j<=n;j++){  g->matrix[i][j]=0;  }  }  }  // tim all cac dinh ke cua dinh x | }  //them cung e = (x,y) vao do thi  void addEdge(Graph \*g, int x, int y){  g->matrix[x][y]=1;  g->matrix[y][x]=1;  }  // kt dinh co ke hay ko  bool adjacent(Graph \*g, int x, int y){  return g->matrix[x][y]==1;  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=n;i++){  if(adjacent(g,x,i)){  pushList(&L,i);  }  }  return L;  }  int mark[maxv];//mang toan cuc mark da danh dau 1 da duoc duyet hay chua  int parent[maxv]; //mang toan cuc parent da luu dinh cha cua toan bo dinh trong do thi  //BFS - QUEUE  List BFS(Graph \*g, int x){  List rs; //Danh sach luu cac dinh da duoc duyet  makenullList(&rs);  Queue q;  makenullQueue(&q);  pushQueue(&q,x);  parent[x]=0; //neu dinh dang xet la dinh goc thi dinh cha cua dinh goc la 0  while(!emptyQueue(&q)){ //lap trong khi hang doi chua rong  int u = getQueue(&q);  plusFront(&q);  if(mark[u]==1) continue; //neu dinh u da duoc duyet thi bo qua cac lenh phia sau  pushList(&rs,u); //neu u chua duyet thay vi in ra thi chung ta se dua vao ds rs(result) -> danh sach ket qua  mark[u]=1; //danh dau u da duyet  List L = neighbors(g,u); //tim dinh ke cua u dua vao ds L  for(int j=1;j<=L.size;j++){  int v = getList(&L,j); //lay dinh ke u trong danh sach L ra -> Lay dinh v  if(mark[v]==0){ //Neu v chua duoc duyet  pushQueue(&q,v); //dua vao hang doi duyet v  if(parent[v]==-1){ //neu dinh v chua co dinh cha  parent[v]=u; //thi dinh cha cua dinh v la u  }  }  }  }  return rs;  }  int main(){  Graph g;  scanf("%d%d",&n,&m);  initGraph(&g);  int u,v;  for(int e=1;e<=m;e++){  scanf("%d%d",&u,&v);  addEdge(&g,u,v);  }  // Kh?i t?o toàn b? ph?n t? trong m?ng mark b?ng 0 vì chua có d?nh nào du?c duy?t  for(int i=1;i<=n;i++){  mark[i]=0;  parent[i]=-1;  }    for(int i=1;i<=n;i++){  if(mark[i]==0){ //N?u d?nh chua du?c duy?t  List L = BFS(&g,i); //G?i hàm và duy?t  for(int j=1;j<=L.size;j++){  int v = getList(&L,j);  // printf("%d\n",v);  //Ð? yêu c?u ch? in d?nh cha c?a các d?nh ra thôi nên không c?n in th? t? duy?t c?a d? th?  mark[v]=1;//Ðánh d?u dã duy?t  }  }  }  //In các d?nh cha c?a các d?nh trong d? th?  for(int i=1;i<=n;i++)  printf("%d %d\n",i,parent[i]); //Ð?nh i có d?nh cha là parent[i]  return 0;  } |

|  |  |
| --- | --- |
| Cho một đồ thị vô hướng đơn. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo chiều sâu (dùng NGĂN XẾP) bắt đầu từ đỉnh 1.  Nếu đồ thị không liên thông, sau khi duyệt xong lần 1, tìm đỉnh có chỉ số nhỏ nhất chưa duyệt mà duyệt nó, và cứ tiếp tục như thế cho đến khi tất cả các đỉnh đều được duyệt.  Quy ước:  Các đỉnh kề của 1 đỉnh được liệt kê theo thứ tự tăng dần | |
| #include <stdio.h>  #include <stdbool.h>  #define maxv 50  #define idx 100  int n,m;  typedef int Vertices;  typedef struct{  int matrix[maxv][maxv];  Vertices n;//Ð?nh n  }Graph;  // khai bao cau truc danh dach list  typedef struct{  int data[idx];  int size;  }List;  //khai bao cau truc ngan xep stack  typedef struct{  int data[idx];  int size;  }Stack;  //ngan xep  // khoi tao stack rong  void makenullStack(Stack \*s){  s->size=0;  }  // kiem tra stack rong  bool emptyStack(Stack \*s){  return s->size==0;  }  // them 1 phtu vao stack  void pushStack(Stack \*s, int element){  s->data[s->size]=element;  s->size++;  }  //truy cap phan tu trong stack  int getStack(Stack \*s){  return s->data[s->size-1];  }  // xoa vi tri phan tu trong stack, ko can toi nua  void delSize(Stack \*s){  s->size--;  }  // lam rong danh ds  void makenullList(List \*l){  l->size=0;  }  // kt ds rong ko  bool emptyList(List \*l){  return l->size==0;  }  //them 1 phtu vao ds  void pushList(List \*l, int element){  l->data[l->size]=element;  l->size++;  }  // truy cap den phtu co vi tri index trong danh sach  int getList(List \*l, int index){  return l->data[index-1];  }  //do thi  //khoi tao  void initGraph(Graph \*g){  g->n=n;  for(int i=1;i<=n;i++)  for(int j=1;j<=n;j++)  g->matrix[i][j]=0;  }  // them cung e = (x,y) vao do thi  void addEdge(Graph \*g, int x, int y){  g->matrix[x][y]=1;  g->matrix[y][x]=1;  } | //kt 2 dinh co ke ko  bool adjacent(Graph \*g, int x, int y){  return g->matrix[x][y]==1;  }  //tim all cac dinh ke cua dinh x  List neighbors(Graph \*g, int x){  List L;  makenullList(&L);  for(int i=1;i<=n;i++){  if(adjacent(g,x,i)){  pushList(&L,i);  }  }  return L;  }  int mark[maxv];//mang toan cuc mark da danh dau 1 dinh da duoc duyet hay chua  int parent[maxv]; //mang toan parent da luu dinh cha cua toan bodinh trong do thi  // DFS - STACK  List DFS\_Stack(Graph \*g, int x){  List rs;//Danh sach luu cac dinh da duoc duyet  makenullList(&rs);  Stack s;  makenullStack(&s);  pushStack(&s,x);  parent[x]=0; //dinh dang xet là dinh goc thi dinh cha la 0  while(!emptyStack(&s)){ //stack chua rong  int u = getStack(&s);  delSize(&s);  if(mark[u]==1) continue; //neu dinh u da duyet thi bo qua all cac cau lenh phia sau  pushList(&rs,u); //neu u chua duyet thay vi in ra thi chung ta se dua vao ds rs(result) -> ds ket qua  mark[u]=1; //danh dau u da duyet  List L = neighbors(g,u); //tim dinh ke cua dinh u dua vao ds L  for(int j=1;j<=L.size;j++){  int v = getList(&L,j); //lay dinh ke u trong danh sach L ra -> lay dinh v  if(mark[v]==0){ //neu v chua duoc duyet  pushStack(&s,v); //dua vao stack da duyet  parent[v]=u; //vi day la Stack nen s? khac Queue  }  }  }  return rs;  }  int main(){  Graph g;  scanf("%d%d",&n,&m);  initGraph(&g);  int u,v;  for(int i=1;i<=m;i++){  scanf("%d%d",&u,&v);  addEdge(&g,u,v);  }  // khoi tao toan bo phtu mark bang 0 vi chua co dinh nao duoc duyet  for(int i=1;i<=n;i++){  mark[i]=0;  parent[i]=-1;  }    for(int i=1;i<=n;i++){  if(mark[i]==0){ //Neu dinh chua duoc duyet  List L = DFS\_Stack(&g,i); //goi ham va duyet  for(int j=1;j<=L.size;j++){  int v = getList(&L,j);  mark[v]=1; //danh dau da duyet  }  }  }  //In các dinh cha cua cac dinh trong do thi  for(int i=1;i<=n;i++)  printf("%d %d\n",i,parent[i]); //dinh i co dinh cha la parent[i]  return 0;  } |