**LÝ THUYẾT ĐỒ THỊ**

[ **KHỞI ĐỘNG** 3](#_Toc44975134)

[ **Bài 3: Các dãy số** 3](#_Toc44975135)

[ **Bài 4: Tam giác** 3](#_Toc44975136)

[ **Bài bổ sung 1 – In số đối xứng** 4](#_Toc44975137)

[ **BTBS 2 – In chữ in thường – in hoa** 5](#_Toc44975138)

[ **BTBS 3 – In số đảo ngược** 5](#_Toc44975139)

[ **BUỔI 1** 6](#_Toc44975140)

[ **Bài 4 – Bậc lớn nhất** 6](#_Toc44975141)

[ **Bài 5 – Danh sách cung:** 7](#_Toc44975142)

[ **Bài 6 – Đọc đồ thị từ tập tin** 8](#_Toc44975143)

[ **Bài 7 – Hệ sinh thái rừng:** 9](#_Toc44975144)

[ **Bài 8 – Bắt tay** 11](#_Toc44975145)

[ **Bài 9 – Gửi nhận mail** 13](#_Toc44975146)

[ **BUỔI 2:** 15](#_Toc44975147)

[ **Bài 1 – Duyệt đồ thị** 16](#_Toc44975148)

[ **Bài 2 – Duyệt đồ thị & Dựng cây duyệt đồ thị** 21](#_Toc44975149)

[ **Bài 3 – Đồ thị liên thông – Qua đảo** 28](#_Toc44975150)

[ **Bài 4 – Tôn ngộ không** 30](#_Toc44975151)

[ **Bài 5: Kiểm tra chu trình – đồ thị vô hướng** 32](#_Toc44975152)

[ **Bài 6: Thuyền trưởng Haddock** 35](#_Toc44975153)

[ **Bài 7: Phân chia đội bóng** 37](#_Toc44975154)

[ **Bài 8 – Kiểm tra tính liên thông mạnh – đếm số BPLT mạnh** 41](#_Toc44975155)

[ **Bài 9 – Come and Go** 45](#_Toc44975156)

[ **BUỔI 3** 50](#_Toc44975157)

[ **Bài 1 – Tìm đường đi ngắn nhất** 50](#_Toc44975158)

[ **Bài 2 – Kiểm tra chu trình và ứng dụng đường đi ngắn nhất.** 52](#_Toc44975159)

[ **Bài 3 - Ứng dụng đường đi ngắn nhất** 54](#_Toc44975160)

[ **Bài 4 – Mê cung trọng số** 54](#_Toc44975161)

[ **Bài 5 – Bellman – Ford pi và pi.** 56](#_Toc44975162)

[ **Bài 6 – Bellman – Ford** 58](#_Toc44975163)

[ **Bài 7 – Floyd – Warshall** 60](#_Toc44975164)

[ **Bài 8 – Tìm số đường đi ngắn nhất (nâng cao)** 62](#_Toc44975165)

[ **BUỔI 4:** 65](#_Toc44975166)

[ **Bài 1 - Xếp hạng đồ thị** 65](#_Toc44975167)

[ **Bài 2 – Cân đá** 69](#_Toc44975168)

[ **Bài 3 – Chia kẹo** 75](#_Toc44975169)

[ **Bài 4 – Tổ chức thi công – Dự án xây nhà** 82](#_Toc44975170)

[ **Bài 5 - Tổ chức thi công – Dự án phần mềm** 87](#_Toc44975171)

[ **BUỔI 5:** 93](#_Toc44975172)

[ **Bài 1 – Tìm cây khung bằng giải thuật Kruskal** 93](#_Toc44975173)

[ **Bài 2 – Tìm cây khung có trọng lượng nhỏ nhất bằng giải thuật Prim** 97](#_Toc44975174)

[ **Bài 3 – Tìm luồng cực đại trong mạng** 101](#_Toc44975175)

[ **BÀI THI THỬ** 105](#_Toc44975176)

[ **Bài 3: Người quen** 105](#_Toc44975177)

[ **Bài 4 – Tìm đường đi của John** 107](#_Toc44975178)

[ **Bài 5 – Quan hệ tiên quyết** 110](#_Toc44975179)

* **KHỞI ĐỘNG**
* **Bài 3: Các dãy số**

|  |  |
| --- | --- |
| *Viết chương trình đọc một dãy số nguyên và in ra dãy số đã sắp xếp.*  *Dữ liệu đầu vào được cho dưới dạng:*  *- Dòng đầu tiên chứa số phần tử n (n <= 100)*  *- Dòng thứ hai chứa n số nguyên (***int***), mỗi số cách nhau 1 khoảng trắng.*  *In dãy số ra màn hình trên cùng 1 dòng, mỗi số cách nhau 1 khoảng trắng.* | *Viết chương trình đọc một ma trận và đếm số phần tử lớn hơn 0 của ma trận này. In kết quả ra màn hình.*  *Dữ liệu đầu vào được cho dưới dạng:*  *- Dòng đầu tiên chứa số hàng m và số cột n, mỗi số cách nhau 1 khoảng trắng, (1 <= m,n <= 100).*  *- m dòng tiếp theo, mỗi dòng chứa các phần tử trên cùng một hàng, mỗi phần tử cách nhau 1 khoảng trắng.*  *Đầu ra: in ra số phần tử lớn hơn 0.* |
| #include <stdio.h>  **int** main()**{**  **int** a[100];  **int** n;    scanf("%d", &n);  **for**(**int** i = 0; i < n; i++)**{**  scanf("%d", &a[i]);  **}**  **int** tg;  **for**(**int** i = 0; i < n - 1; i++)**{**  **for**(**int** j = i + 1; j < n; j++)**{**  if(a[i] > a[j])**{**  // Hoan vi 2 so a[i] va a[j]  tg = a[i];  a[i] = a[j];  a[j] = tg;  **}**  **}**  **}**  **for**(**int** i = 0; i < n; i++)**{**  pr**int**f("%d ", a[i]);  **}**    return 0;  **}** | #include <stdio.h>  **int** main() **{**  **int** A[100][100], m, n;  //Đọc số hàng m và số cột n  scanf("%d%d", &m, &n);  **int** i, j;  //Đọc ma trận  **for** (i = 0; i < m; i++)  **for** (j = 0; j < n; j++)  scanf("%d", &A[i][j]);  //Đếm số phần tử > 0  **int** kq = 0;  **for** (i = 0; i < m; i++)  **for** (j = 0; j < n; j++)  **if** (A[i][j] > 0)  kq++;    //In số phần tử > 0 ra màn hình  pr**int**f("%d\n", kq);  return 0;  **}** |

* **Bài 4: Tam giác**

|  |  |  |
| --- | --- | --- |
| *Viết chương trình in ra màn hình hình tam giác vuông cân cạnh n với n nhập từ bàn phím (n > 0) theo dạng:*  *Ví dụ: n = 4*  *\* \*\* \*\*\* \*\*\*\** | *Viết chương trình in ra màn hình hình tam giác vuông cân cạnh n với n nhập từ bàn phím (n > 0) theo dạng:*  *Ví dụ: n = 4*  *\*  \*\*  \*\*\* \*\*\*\** | *Viết chương trình in ra màn hình hình tam giác vuông cân cạnh n với n nhập từ bàn phím (n > 0) theo dạng:*  *Ví dụ: n = 4*  *\*  \*\*\*  \*\*\*\*\**  *\*\*\*\*\*\*\** |
| #include<stdio.h>  **int** main()  **{**  **int** n,i,j;  scanf("%d",&n);  **for**(i = 1; i <= n; i++) **{**  **for**(j = 1; j <= i; j++)  pr**int**f("\*");  pr**int**f("\n");  **}**  return 0;  **}** | #include <stdio.h>  **int** main() **{**  **int** n, i, j;  scanf("%d", &n);  **for** (i = 1; i <= n; i++) **{**  **for** (j = 1; j <= n-i; j++)  pr**int**f(" ");  **for** (j = 1; j <= i; j++)  pr**int**f("\*");  pr**int**f("\n");  **}**  return 0;  **}** | #include <stdio.h>  **int** main() **{**  **int** n, i, j;  scanf("%d", &n);  **for** (i = 1; i <= n; i++) **{**  **for** (j = 1; j <= n-i; j++)  pr**int**f(" ");  **for** (j = 1; j <= 2\*i-1; j++)  pr**int**f("\*");  pr**int**f("\n");  **}**  return 0;  **}** |

* **Bài bổ sung 1 – In số đối xứng**

|  |
| --- |
| **Đầu vào:**   * Nhập vào số nguyên dương có 3 chữ số   **Đầu ra:**   * Kiểm tra số đó có đối xứng không |
| #include <stdio.h>  #include <stdlib.h>  #include <conio.h>  using namespace std;  **int** main()  **{**  **int** n, a, b, c;  do  **{**  pr**int**f("Nhap vao mot so duong co ba chu so: ");  scanf("%d", &n);  if(n > 999 || n < 0)  **{**  pr**int**f("Vui long nhap lai!!");  getch();  system("cls");  **}**  else  **{**  a = n / 100;  b = n % 100 / 10;  c = n % 10;  if(a == c)  **{**  pr**int**f("So %d%d%d la so doi xung!!", a, b, c);  **}**  else  **{**  pr**int**f("So %d%d%d khong phai la so doi xung!!", a, b, c);  **}**  **}**  **}**  while(n > 999 || n < 0);  getch();  return 0;  **}** |

* **BTBS 2 – In chữ in thường – in hoa**

|  |
| --- |
| Viết chương trình in cho phép nhập vào 1 chữ cái HOA, in ra màn hình chữ in thường của nó.  Viết chương trình in cho phép nhập vào 1 chữ cái THƯỜNG, in ra màn hình chữ in hoa của nó. |
| #include <stdio.h>  **int** main()  **{**  char s[100];  **int** i;  pr**int**f("\nNhap vao mot chuoi : ");  gets(s);  i = 0;  while (s[i] != 0)  **{**  if (isalpha(s[i])) //Hoa sang thuong  if (s[i] < 97)  s[i] += 32;  else // thuong sang hoa  s[i] -= 32;  else  pr**int**f("\n ky tu ban nhap vao khong dung");  i++;  **}**  pr**int**f("\nChuoi bien thanh : %s", s);  return 0;  **}** |

* **BTBS 3 – In số đảo ngược**

|  |
| --- |
| **Input:**  Nhập vào một mảng số nguyên n.  **Output:**  Đảo ngược các số lại |
| #include <stdio.h>  //in so dao nguoc  **int** main() **{**  **int** A[100] ;  **int** i, j, n;  scanf("%d",&n);  **for**(i=1; i<=n; i++)  scanf("%d",&A[i]);  **for**(i = n; i > 0; i--)  printf("%d ", A[i]);  return 0;  **}** |

* **BUỔI 1**
* **Bài 4 – Bậc lớn nhất**

|  |
| --- |
| Cho một đồ thị vô hướng, không khuyên, không đa cung có n đỉnh và m cung.  Viết chương trình tính và in ra màn hình đỉnh có bậc lớn nhất và bậc tương ứng của nó. Nếu có nhiều đỉnh có bậc bằng nhau thì in ra đỉnh có số thứ tự nhỏ nhất.  **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 đỉnh có bậc lớn nhất và bậc của nó.  Xem thêm ví dụ bên dưới.  Trong ví dụ đầu tiên ta có:   * Bậc của đỉnh 1 là 1, * Bậc của đỉnh 2 là 3 * Bậc của đỉnh 3 là 2 * Bậc của đỉnh 4 là 2   Vậy đỉnh có bậc lớn nhất là đỉnh 2 và bậc của nó là 3. Vì thế ta in ra:  2 3 |
| #include<stdio.h>  #define MAX\_VERTICES 20  #define MAX\_EDGES 20  **typedef** struct**{**  **int** n, m;  **int** A[MAX\_VERTICES][MAX\_EDGES];  **}**Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = m;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->m; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** e, **int** x, **int** y)**{**  G->A[x][e] = 1;  G->A[y][e] = 1;  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** deg = 0;  **int** j;  **for**(j = 1; j <= G->m; j++)  if(G->A[x][j] == 1)  deg++;  return deg;  **}**  **int** main()**{**  Graph G;  **int** i, j, u, v, n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for**(j = 1; j <= G.m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, j, u, v);  **}**  **int** max = degree(&G, 1);  **int** t = 1;  **for**(i = 2; i <= G.n; i++)  if(max < degree(&G, i))**{**  max = degree(&G, i);  t = i;  **}**  pr**int**f("%d %d", t, max);  return 0;  **}** |

* **Bài 5 – Danh sách cung:**

|  |  |
| --- | --- |
| Cho cấu trúc dữ liệu đồ thị được cài đặt bằng phương pháp "Danh sách cung" như sau:  **typedef** struct  **{**  **int** x, y; **}** Edge; **typedef** struct **{**  **int** n, m;  Edge edges[MAX\_EDGES]; **}** 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\* G, **int** n)**{**  G->n = n;  G->m = 0; **}**  Viết hàm **add\_edge(Graph\* G, int x, int y)** để thêm cung (x, y) vào đồ thị G.  void add\_edge(Graph\* G, **int** x, **int** y) **{** **}** | Cho cấu trúc dữ liệu đồ thị được cài đặt bằng phương pháp "Danh sách cung" như sau:  **typedef** struct  **{**  **int** x, y; **}** Edge; **typedef** struct **{**  **int** n, m;  Edge edges[MAX\_EDGES]; **}** 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\* G, **int** n)**{**  G->n = n;  G->m = 0; **}**  Viết hàm **add\_edge(Graph\* G, int x, int y)** để thêm cung (x, y) vào đồ thị G.  void add\_edge(Graph\* G, **int** x, **int** y) **{** **}**  Chú ý:  - Nếu cung (x, y) không hợp lệ (vd: x < 1, y > n, ...) thì bỏ qua không làm gì cả. |
| void add\_edge(Graph\* G, **int** x, **int** y)**{**  G->edges[G->m].x = x;  G->edges[G->m].y = y;  G->m++;  **}** | void add\_edge(Graph\* G, **int** x, **int** y)**{**  if(x < 1 || x > G->n || y > G->n || y < 1)**{**  **}**else**{**  G->edges[G->m].x = x;  G->edges[G->m].y = y;  G->m++;  **}**  **}** |

* **Bài 6 – Đọc đồ thị từ tập tin**

|  |
| --- |
| *Hãy viết chương trình đọc đồ thị từ tập tin và hiển thị ma trận kề của đồ thị này.*  *Giả sử đồ thị được cho là đơn đồ thị vô hướng.*  **Đầu vào (Input):**  Dữ liệu đầu vào được nhập từ tập tin **dt1.txt** 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 Ma trận kề (0/1) của đồ thị |
| #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y] = 1;  G->A[y][x] = 1;  **}**  **int** main()**{**  Graph G;  freopen("dt1.txt", "r", stdin);  **int** u, v, j, i, n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  **for**(j = 1; j <= m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**  **for**(i = 1; i <= G.n; i++)**{**  **for**(j = 1; j <= G.n; j++)  pr**int**f("%d ", G.A[i][j]);  pr**int**f("\n");  **}**  return 0;  **}** |

* **Bài 7 – Hệ sinh thái rừng:**

Trong hệ sinh thái rừng gồm có các loài động vật sau đây: gấu trúc, diều hâu, chim cú, sóc, quạ, thú có túi, chuột chù, chuột, chim gõ kiến. Những nhà nghiên cứu về hệ sinh thái này nhận thấy có sự tương tác của các loài vật này về nguồn thức ăn. Ví dụ: Gấu trúc và Chim cú cạnh tranh với nhau vì chúng có cùng nguồn thức ăn. Sự cạnh tranh được các nhà nghiên cứu tổng hợp lại như sau:

- (Gấu trúc và Chim cú)

- (Gấu trúc và Diều hâu)

- (Diều hâu và Chim cú)

- (Gấu trúc và Sóc)

- (Sóc và Quạ)

- (Diều hâu và Quạ)

- (Chim cú và Quạ)

- (Thú có túi và Sóc)

- (Thú có túi và Chuột chù)

- (Thú có túi và Chim gõ kiến)

- (Sóc và Chim gõ kiến)

- (Chim gõ kiến và Chuột chù)

- (Chuột chù và Chuột).

Các nhà nghiên cứu cần biết giữa hai loài x và y thì chúng có chung những đối thủ cạnh tranh nào? Hãy viết chương trình để giúp các nhà nghiên cứu tìm ra đối thủ cạnh tranh chung của hai loài x và y.

|  |  |
| --- | --- |
| **Đầu vào (Input):**  - Dòng đầu tiên chứa hai số nguyên n và m tương ứng là số loài động vật trong hệ sinh thái rừng và số cặp loài u - v cạnh tranh với nhau vì chúng có cùng nguồn thức ăn  - m dòng kế tiếp, mỗi dòng chứa một cặp số nguyên u v cách nhau một khoảng trắng. Loài u v cạnh tranh với nhau vì chúng có cùng nguồn thức ăn  - Dòng cuối cùng chứa 2 số nguyên x y là hai loài mà các nhà nghiên cứu cần tìm những đối thủ cạnh tranh chung của chúng.  **Đầu ra (Output):**  *- In ra màn danh sách những đối thủ cạnh tranh chung của hai loài theo thứ tự từ nhỏ đến lớn.*  *- Nếu hai loài không có chung đối thủ cạnh tranh thì in ra "****KHONG CHUNG DOI THU****"* | **Đầu vào (Input):**  - Dòng đầu tiên chứa số nguyên n tương ứng là số loài động vật trong hệ sinh thái rừng  - n dòng kế tiếp, mỗi dòng chứa n số nguyên w cách nhau một khoảng trắng.   * w=0: nói rằng động vật (dòng) thứ i  và động (cột) thứ j không là đối thủ cạnh tranh. * w=1: nói rằng động vật (dòng) thứ i  và động (cột) thứ j là đối thủ cạnh tranh.   - Dòng cuối cùng chứa 2 số nguyên x y là hai loài mà các nhà nghiên cứu cần tìm những đối thủ cạnh tranh chung của chúng.  **Đầu ra (Output):**  *- In ra màn số đối thủ cạnh tranh chung của hai loài.* |
| #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->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)**{**  if(G->A[x][y] == 1)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  pr**int**f("%d ", i);  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** deg = 0;  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  deg++;  return deg;  **}**  void doithu(Graph \*G, **int** x, **int** y)**{**  **int** i, j = 0;  **for**(i = 1; i <= G->n; i++)**{**  if(G->A[x][i] == 1 && G->A[y][i] == 1)**{**  pr**int**f("%d ", i);  j++;  **}**  **}**  if(j == 0)  pr**int**f("KHONG CHUNG DOI THU");  **}**  **int** main()  **{**  Graph G;  **int** i, u, v, n, m, x, y;  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);  **}**  scanf("%d%d", &x, &y);    doithu(&G, x, y);  return 0;  **}** | #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->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)**{**  if(G->A[x][y] == 1)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  pr**int**f("%d ", i);  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** deg = 0;  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  deg++;  return deg;  **}**  void doithu(Graph \*G, **int** x, **int** y)**{**  **int** i, j = 0;  **for**(i = 1; i <= G->n; i++)**{**  **int** a = G->A[x][i];  **int** b = G->A[i][x];  **int** c = G->A[y][i];  **int** d = G->A[i][y];  if((a == 1 && c == 1) || (a == 1 && d == 1) || (b == 1 && c == 1) || (b == 1 && d == 1))**{**  j++;  **}**  **}**  pr**int**f("%d", j);  **}**  **int** main()**{**  Graph G;  **int** i, j, u, n, x, y;  scanf("%d", &n);  init\_graph(&G, n);  **for**(i = 1; i <= n; i++)**{**  **for**(j = 1; j <= n; j++)**{**  scanf("%d", &u);  G.A[i][j] = u;  **}**  **}**  scanf("%d%d", &x, &y);  doithu(&G, x, y);    return 0;  **}** |

* **Bài 8 – Bắt tay**

|  |  |
| --- | --- |
| **Dự tiệc**  Trong một buổi tiệc có **n** người tham dự, người ta quan sát được có **m** lần bắt tay.  Hỏi mỗi người đã bắt tay bao nhiêu lần. Dĩ nhiên, một người không thể tự bắt tay với chính mình và ta được biết rằng đôi khi do quên hai người có thể bắt tay với nhau nhiều lần.  **Đầu vào**  Dữ liệu đầu vào được đọc từ bàn phím (stdin) với định dạng sau:   * Dòng đầu tiên gồm hai số nguyên n và m cách nhau một khoảng trắng với n là số người và m là tổng số lần bắt tay quan sát được. * m dòng kế tiếp, mỗi dòng chứa 2 số nguyên a b cách nhau một khoảng trắng ý nói rằng người a và người b đã bắt tay với nhau.   **Đầu ra**   * In ra màn hình (stdout) số lần bắt tay của từng người theo *thứ tự từ 1 đến n, mỗi người 1 dòng.* * Xem chi tiết trong phần ví dụ. Trong ví dụ đầu tiên, người số 1 đã bắt tay 5 lần, người số 2 đã bắt tay 4 lần và người thứ 3 đã bắt tay 5 lần. | Trong một buổi tiệc có **n** người tham dự, người ta quan sát được có **m** lần bắt tay.  Dĩ nhiên, một người không thể tự bắt tay với chính mình và ta được biết rằng đôi khi do quên hai người có thể bắt tay với nhau nhiều lần.  Hỏi người bắt tay nhiều lần nhất đã bắt tay bao nhiêu lần trong bưa tiệc này.  **Đầu vào**  Dữ liệu đầu vào được đọc từ bàn phím (stdin) với định dạng sau:   * Dòng đầu tiên gồm hai số nguyên **n** và **m** cách nhau một khoảng trắng với:   + **n** là số người và   + **m** là tổng số lần bắt tay quan sát được. * **m** dòng kế tiếp, mỗi dòng chứa 2 số nguyên **a b** cách nhau một khoảng trắng ý nói rằng người **a** và người **b** đã bắt tay với nhau.   **Đầu ra**   * In ra màn hình (stdout) số lần bắt tay nhiều nhất của người bắt tay *nhiều lần nhất.* * Xem chi tiết trong phần ví dụ. Trong ví dụ đầu tiên, người số 1 đã bắt tay 5 lần, người số 2 đã bắt tay 4 lần và người thứ 3 đã bắt tay 5 lần. Vậy người 1 và người 3 là người có số lần bắt tay nhiều nhất và *số lần bắt tay nhiều nhất* là 5. |
| #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  **int** adjacent(Graph \*G, **int** x, **int** y)**{**  if(G->A[x][y] > 0)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) > 0)  pr**int**f("%d ", i);  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** deg = 0;  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  deg += G->A[x][i];  return deg;  **}**  **int** main()**{**  Graph G;  **int** i, u, v, n, m;  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);  **}**    **for**(i = 1; i <= G.n; i++)  pr**int**f("%d\n", degree(&G, i));    return 0;  **}** | #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  **int** adjacent(Graph \*G, **int** x, **int** y)**{**  if(G->A[x][y] > 0)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) > 0)  pr**int**f("%d ", i);  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** deg = 0;  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) == 1)  deg += G->A[x][i];  return deg;  **}**  **int** main()**{**  Graph G;  **int** i, u, v, n, m;  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);  **}**    **int** max = degree(&G, 1);  **for**(i = 2; i <= G.n; i++)  if(max < degree(&G, i))  max = degree(&G, i);    pr**int**f("%d", max);    return 0;  **}** |

* **Bài 9 – Gửi nhận mail**

|  |  |
| --- | --- |
| Trong một lớp học có **n** sinh viên (giả sử được đánh số từ 1 đến **n**). Mỗi sinh viên đều được cấp một tài khoản nên có thể sử dụng tài khoản này để đăng nhập hệ thống email của Google để gởi email cho nhau.  Giả sử không có ai tự gởi email cho chính mình cả và người A có thể gởi cho người B nhiều email khác nhau.  Hãy tìm người đã nhận nhiều email nhất.  **Đầu vào**  Dữ liệu đầu vào được đọc từ bàn phím (stdin) với định dạng sau:   * Dòng đầu tiên là số nguyên n tương ứng với số sinh viên trong lớp. * n dòng kế tiếp, mỗi dòng chứa n số nguyên w cách nhau một khoảng trắng nói rằng sinh viên (dòng) thứ i  đã gởi cho sinh viên (cột) thứ j: w emails.   **Đầu ra**  In ra màn hình (stdout) người đã nhân nhiều email nhất và số email người đó đã nhận được theo định dạng sau:  x has received y email(s).  Với x là người đã nhận nhiều email nhất và y là số email mà x đã nhận. Nếu có nhiều người có số email nhận được bằng nhau thì chọn người có số thứ tự nhỏ nhất. | **Đầu vào**  Dữ liệu đầu vào được đọc từ bàn phím (stdin) với định dạng sau:   * Dòng đầu tiên gồm hai số nguyên n và m cách nhau một khoảng trắng với n là số người và m là tổng số lần gởi email đi. * m dòng kế tiếp, mỗi dòng chứa 2 số nguyên a b cách nhau một khoảng trắng nói rằng người a đã gởi 1 email cho người b.   **Đầu ra**   * In ra màn hình (stdout) người đã gởi email nhiều nhất và số email người đó đã gởi đi theo định dạng sau:   x has sent y email(s).  Với x là người đã gởi email nhiều nhất và y là số email mà x đã gởi. Nếu có nhiều người có số email gởi đi bằng nhau thì chọn người có số thứ tự nhỏ nhất. |
| #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  **int** adjacent(Graph \*G, **int** x, **int** y)**{**  if(G->A[x][y] > 0)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) > 0)  pr**int**f("%d ", i);  **}**  **int** nhan\_mail(Graph \*G, **int** x)**{**  **int** i, tong = 0;  **for**(i = 1; i <= G->n; i++)  tong += G->A[i][x];  return tong;  **}**  **int** main()**{**  Graph G;  **int** i, j, u, n;  scanf("%d", &n);  init\_graph(&G, n);    **for**(i = 1; i <= n; i++)**{**  **for**(j = 1; j <= n; j++)**{**  scanf("%d", &u);  G.A[i][j] = u;  **}**  **}**    **int** max = nhan\_mail(&G, 1);  **int** t = 1;  **for**(i = 1; i <= G.n; i++)**{**  if(max < nhan\_mail(&G, i))**{**  max = nhan\_mail(&G, i);  t = i;  **}**  **}**  **for**(i = 1; i <= G.n; i++)  if(max == nhan\_mail(&G, i) && t > i)**{**  t = i;  **}**    pr**int**f("%d has received %d email(s).", t, max);  return 0;  **}** | #include<stdio.h>  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  **}**  **int** adjacent(Graph \*G, **int** x, **int** y)**{**  if(G->A[x][y] > 0)  return 1;  return 0;  **}**  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  if(adjacent(G, x, i) > 0)  pr**int**f("%d ", i);  **}**  **int** gui\_mail(Graph \*G, **int** x)**{**  **int** i, tong = 0;  **for**(i = 1; i <= G->n; i++)  tong += G->A[x][i];  return tong;  **}**  **int** main()**{**  Graph G;  **int** i, u, v, m, n;  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);  **}**    **int** max = gui\_mail(&G, 1);  **int** t = 1;  **for**(i = 1; i <= G.n; i++)**{**  if(max < gui\_mail(&G, i))**{**  max = gui\_mail(&G, i);  t = i;  **}**  **}**  **for**(i = 1; i <= G.n; i++)  if(max == gui\_mail(&G, i) && t > i)**{**  t = i;  **}**    pr**int**f("%d has sent %d email(s).", t, max);  return 0;  **}** |

* **BUỔI 2:**
* **Bài 1 – Duyệt đồ thị**

|  |  |  |
| --- | --- | --- |
| 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*   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng. | 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*.   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng. | 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*.   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra các đỉnh theo thứ tự duyệt, mỗi đỉnh trên 1 dòng. |
| #include<stdio.h>  #define MAX\_N 50  **typedef** struct**{**  **int** size;  **int** data[MAX\_N];  **}**List;  void makenull\_list(List \*L)**{**  L->size = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->size] = x;  L->size++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->size;  **}**  **int** empty\_list(List \*L)**{**  return L->size == 0;  **}**  **typedef** struct**{**  **int** front, rear;  **int** data[MAX\_N];  **}**Queue;  void makenull\_queue(Queue \*Q)**{**  Q->rear = -1;  Q->front = 0;  **}**  **int** empty\_queue(Queue \*Q)**{**  return Q->front > Q->rear;  **}**  **int** top\_queue(Queue \*Q)**{**  return Q->data[Q->front];  **}**  void pop\_queue(Queue \*Q)**{**  Q->front++;  **}**  void push\_queue(Queue \*Q, **int** x)**{**  Q->rear++;  Q->data[Q->rear] = x;  **}**  **typedef** struct**{**  **int** top\_idx;  **int** data[MAX\_N];  **}**Stack;  void makenull\_stack(Stack \*S)**{**  S->top\_idx = 0;  **}**  **int** empty\_stack(Stack \*S)**{**  return S->top\_idx == 0;  **}**  void push\_stack(Stack \*S, **int** x)**{**  S->data[S->top\_idx] = x;  S->top\_idx++;  **}**  **int** top\_stack(Stack \*S)**{**  return S->data[S->top\_idx - 1];  **}**  void pop\_stack(Stack \*S)**{**  S->top\_idx--;  **}**  **typedef** struct**{**  **int** n;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** i, deg = 0;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  deg++;  return deg;  **}**  List neighbors(Graph \*G, **int** x)**{**  List L;  makenull\_list(&L);  **int** i;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  push\_back(&L, i);    return L;  **}**  **int** mark[MAX\_N];  void breath\_first\_search(Graph \*G, **int** k)**{**  Queue Q;  makenull\_queue(&Q);  push\_queue(&Q, k);  mark[k] = 1;  while(!empty\_queue(&Q))**{**  **int** x = top\_queue(&Q);  pop\_queue(&Q);  pr**int**f("%d\n", x);  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  if(mark[y] == 0)**{**  push\_queue(&Q, y);  mark[y] = 1;  **}**  **}**  **}**  **}**  void duyetsau\_dequy(Graph \*G, **int** k)**{**  if(mark[k] == 0)**{**  pr**int**f("%d\n", k);  mark[k] = 1;  List L = neighbors(G, k);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  duyetsau\_dequy(G, y);  **}**  **}**  **}**  void depth\_first\_search(Graph \*G, **int** k)**{**  Stack S;  makenull\_stack(&S);  push\_stack(&S, k);  while(!empty\_stack(&S))**{**  **int** x = top\_stack(&S);  pop\_stack(&S);  if(mark[x] == 0)**{**  pr**int**f("%d\n", x);  mark[x] = 1;  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)  push\_stack(&S, element\_at(&L, i));  **}**  **}**  **}**  **int** main()**{**  Graph G;  //freopen("dothi.txt", "r", stdin);  **int** n, m, i, u, v;  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);  **}**    **for**(i = 1; i <= G.n; i++)  mark[i] = 0;  **for**(i = 1; i <= G.n; i++)  if(mark[i] == 0)**{**  breath\_first\_search(&G, i);  **}**    return 0;  **}** | #include<stdio.h>  #define MAX\_N 50  **typedef** struct**{**  **int** size;  **int** data[MAX\_N];  **}**List;  void makenull\_list(List \*L)**{**  L->size = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->size] = x;  L->size++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->size;  **}**  **int** empty\_list(List \*L)**{**  return L->size == 0;  **}**  **typedef** struct**{**  **int** front, rear;  **int** data[MAX\_N];  **}**Queue;  void makenull\_queue(Queue \*Q)**{**  Q->rear = -1;  Q->front = 0;  **}**  **int** empty\_queue(Queue \*Q)**{**  return Q->front > Q->rear;  **}**  **int** top\_queue(Queue \*Q)**{**  return Q->data[Q->front];  **}**  void pop\_queue(Queue \*Q)**{**  Q->front++;  **}**  void push\_queue(Queue \*Q, **int** x)**{**  Q->rear++;  Q->data[Q->rear] = x;  **}**  **typedef** struct**{**  **int** top\_idx;  **int** data[MAX\_N];  **}**Stack;  void makenull\_stack(Stack \*S)**{**  S->top\_idx = 0;  **}**  **int** empty\_stack(Stack \*S)**{**  return S->top\_idx == 0;  **}**  void push\_stack(Stack \*S, **int** x)**{**  S->data[S->top\_idx] = x;  S->top\_idx++;  **}**  **int** top\_stack(Stack \*S)**{**  return S->data[S->top\_idx - 1];  **}**  void pop\_stack(Stack \*S)**{**  S->top\_idx--;  **}**  **typedef** struct**{**  **int** n;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** i, deg = 0;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  deg++;  return deg;  **}**  List neighbors(Graph \*G, **int** x)**{**  List L;  makenull\_list(&L);  **int** i;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  push\_back(&L, i);    return L;  **}**  **int** mark[MAX\_N];  void breath\_first\_search(Graph \*G, **int** k)**{**  Queue Q;  makenull\_queue(&Q);  push\_queue(&Q, k);  mark[k] = 1;  while(!empty\_queue(&Q))**{**  **int** x = top\_queue(&Q);  pop\_queue(&Q);  pr**int**f("%d\n", x);  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  if(mark[y] == 0)**{**  push\_queue(&Q, y);  mark[y] = 1;  **}**  **}**  **}**  **}**  void duyetsau\_dequy(Graph \*G, **int** k)**{**  if(mark[k] == 0)**{**  pr**int**f("%d\n", k);  mark[k] = 1;  List L = neighbors(G, k);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  duyetsau\_dequy(G, y);  **}**  **}**  **}**  void depth\_first\_search(Graph \*G, **int** k)**{**  Stack S;  makenull\_stack(&S);  push\_stack(&S, k);  while(!empty\_stack(&S))**{**  **int** x = top\_stack(&S);  pop\_stack(&S);  if(mark[x] == 0)**{**  pr**int**f("%d\n", x);  mark[x] = 1;  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)  push\_stack(&S, element\_at(&L, i));  **}**  **}**  **}**  **int** main()**{**  Graph G;  //freopen("dothi.txt", "r", stdin);  **int** n, m, i, u, v;  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);  **}**    **for**(i = 1; i <= G.n; i++)  mark[i] = 0;  //**int** t = 0;  **for**(i = 1; i <= G.n; i++)  if(mark[i] == 0)**{**  duyetsau\_dequy(&G, i);  **}**  return 0;  **}** | #include<stdio.h>  #define MAX\_Vertices 25  #define MAX\_Length 25  #define MAX\_Element 40 //**for** Stack  **typedef** struct  **{**  **int** A[MAX\_Vertices][MAX\_Vertices];  **int** n; //so luong dinh  **}** Graph;  //Ham khoi tao do thi Dinh-Dinh  void init\_Graph(Graph \*G, **int** n)  **{**  **int** i, j;  G->n = n; //so luong dinh =6, nhap vao;  **for**(i=1; i<=n; i++)//co the su sung n hoac G->n  **for**(j=1; j<=n; j++)//Nhu tren  G->A[i][j]=0;  **}**  //Ham them cung vao do thi Dinh-dinh  void add\_edge(Graph \*G, **int** x, **int** y)  **{**  G->A[x][y]=1;  G->A[y][x]=1;  **}**  //Ham kiem tra dinh ke nhau (lang gieng) lien thuoc  **int** adjacent(Graph \*G, **int** x, **int** y) //Kieu du lieu tra loi Yes No  **{**  return (G->A[x][y]!=0);  **}**  //Ham tinh bac cua mot dinh  **int** degree(Graph \*G,**int** x)  **{**  **int** deg=0, i;  **for**(i=1; i<= G->n; i++)  if(G->A[i][x]==1) //co the thay the bang hâm adjacent  //if(adjacent(G, i ,x))  deg++;  return deg;  **}**  //Kiem tra cac dinh lien ke, can cai dat danh sach  //Khai bao cau truc danh sach List  **typedef** struct  **{**  **int** data[MAX\_Length];  **int** size;  **}** List;  //Ham khoi tao danh sach rong  void make\_null(List \*list)  **{**  list->size = 0;  **}**  //Them mot phan tu (dinh) vao cuoi danh sach  void push\_back(List \*list, **int** x)  **{**  list->data[list->size] = x; //cho phan tu dau tien = 0 va tang no len sau  list->size++;//tang phan tu len  **}**  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  **int** element\_at(List \*list, **int** i)  **{**  return list->data[i-1]; //do mang bat dau tu vi tri khong, cho nen phai -1  **}**  //Tim lang gieng cua dinh x  List neighbors(Graph \*G, **int** x) //in ra mot danh sach, nen dung kieu List.  **{**  List L; //can khoi tao mot danh sach khi tim  make\_null(&L);  **int** i;  **for**(i=1; i<=G->n; i++)//giong nhu ham Degree  **{**  //duyet qua tat ca cac dinh, neu dinh nao di toi duoc dinh x thi them vao List  if(G->A[i][x] == 1)  //if(adjacent(G, i, x))  push\_back(&L, i); //dua mot dinh vao danh sach  **}**  return L;  **}**  //Cai dat ngan xep de tinh DFS  //Khai bao cau truc cua ngan xep  **typedef** struct  **{**  **int** data[MAX\_Element];  **int** size; //bien de lay phan tu cuoi cung trong ngan xep ra  **}** Stack;  //Khoi tao ngan xep rong  void make\_null\_stack(Stack \*stack)  **{**  stack->size = 0;  **}**  //Them mot phan tu vao trong ngan xep  void push\_stack(Stack \*stack, **int** x) //x: dinh them vao Stack  **{**  stack->data[stack->size] = x; //them x vao data, va chi so cua man tai Ngan xep link excel.  stack->size ++; //tang size len khi them  **}**  //Lay mot phan tu trong Stack  **int** top(Stack \*stack)  **{**  return stack->data[stack->size-1]; //so sanh vao excel, size & idex, size - 1 = idex  **}**  //Xoa mot phan tu thuoc Stack  void pop(Stack \*stack)  **{**  stack->size --;  **}**  //Kiem tra Stack rong  **int** empty(Stack \*stack)  **{**  return stack->size == 0;  **}**  //Thuat toan DFS duyet theo chieu sau  List depth\_first\_search(Graph \*G, **int** x)  **{**  Stack S;  make\_null\_stack(&S); //Khoi tao ham rong  push\_stack(&S, x); //dua dinh x vao  List list\_dfs;  make\_null(&list\_dfs);  **int** mark[MAX\_Vertices]; //mangx danh dau xem mot dinh da duoc duyet chua  //Khoi tao tat ca cac dinh chua duoc duyet  **int** i;  **for**(i=1; i<=G->n; i++)  mark[i]= 0; //Khi mark =0 thi dinh chua duoc duyet, nguoc lai  while(!empty(&S))  **{**  **int** u = top(&S); //Lay mot dinh ra  pop(&S); //Xoa dinh do khoi ngan xep khi da lay  if(mark[u] == 1)  continue;  // pr**int**f("Duyet: %d\n", u);  push\_back(&list\_dfs, u); //dua u vao danh sach thay vi duyet u  mark[u]= 1;  List list = neighbors(G, u); //Ham neighbors su dung kieu du du lieu List nen phai goi List ra  **for**(i=1; i<=list.size; i++) //lay phan tu trong list  **{**  **int** v = element\_at(&list, i);  if(mark[v] == 0)  push\_stack(&S, v); //Dua v vao ngan xep  **}**  **}**  return list\_dfs;  **}**  **int** main()  **{**  Graph G;  //freopen("DFS.txt", "r", stdin); bo qua dong nay khi nop bai  **int** n, m;  scanf("%d%d", &n, &m);  init\_Graph(&G, n);  **int** e, u, v;  **for**(e=1; e<=m; e++)  **{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v); //them cung  **}**  **int** mark\_dfs[MAX\_Vertices]; //kiem tra duyet cua tat ca do thi  **int** i, j;  **for**(i=1; i<=G.n; i++)  **{**  mark\_dfs[i] = 0;  **}**  **for**(i=1; i<=G.n; i++)  **{**  if(mark\_dfs[i] == 0) //neu chua duoc duyet  **{**  List dfs = depth\_first\_search(&G, i);  **for**(j=1; j<=dfs.size; j++) //tien hanh in danh sach  **{**  **int** k = element\_at(&dfs, j);  pr**int**f("%d\n", k);  mark\_dfs[k] = 1;  **}** //2 dong lap **for** khong su dung cung mot bien lap  **}**  **}**  return 0;  **}** |

* **Bài 2 – Duyệt đồ thị & Dựng cây duyệt đồ thị**

|  |  |  |
| --- | --- | --- |
| 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*   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra cây duyệt đồ thị theo định dạng:  1 <đỉnh cha của 1> 2 <đỉnh cha của 2>  **Quy ước:** Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 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*   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra cây duyệt đồ thị theo định dạng:  1 <đỉnh cha của 1> 2 <đỉnh cha của 2>  **Quy ước:** Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 0. | Cho một đồ thị **vô hướng đơn**. Hãy dựng (các) cây DUYỆT ĐỒ THỊ khi duyệt đồ thị theo **chiều sâu** (dùng *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*.   **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 nói rằng mô tả cung (u, v).  **Đầu ra (Output):**  In ra cây duyệt đồ thị theo định dạng:  1 <đỉnh cha của 1>  **Quy ước:** Nếu 1 đỉnh không có đỉnh cha (nó là đỉnh gốc của cây) thì đỉnh cha của nó là 0. |
| #include<stdio.h>  #define MAX\_N 50  **typedef** struct**{**  **int** front, rear;  **int** data[MAX\_N];  **}**Queue;  void makenull\_queue(Queue \*Q)**{**  Q->front = 0;  Q->rear = -1;  **}**  **int** empty\_queue(Queue \*Q)**{**  return Q->front > Q->rear;  **}**  **int** top\_queue(Queue \*Q)**{**  return Q->data[Q->front];  **}**  void pop\_queue(Queue \*Q)**{**  Q->front++;  **}**  void push\_queue(Queue \*Q, **int** x)**{**  Q->rear++;  Q->data[Q->rear] = x;  **}**  **typedef** struct**{**  **int** top\_idx;  **int** data[MAX\_N];  **}**Stack;  void makenull\_stack(Stack \*S)**{**  S->top\_idx = 0;  **}**  **int** empty\_stack(Stack \*S)**{**  return S->top\_idx == 0;  **}**  void push\_stack(Stack \*S, **int** x)**{**  S->data[S->top\_idx] = x;  S->top\_idx++;  **}**  void pop\_stack(Stack \*S)**{**  S->top\_idx--;  **}**  **int** top\_stack(Stack \*S)**{**  return S->data[S->top\_idx - 1];  **}**  **typedef** struct**{**  **int** size;  **int** data[MAX\_N];  **}**List;  void makenull\_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** p)**{**  return L->data[p - 1];  **}**  **typedef** struct**{**  **int** n;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  List neighbors(Graph \*G, **int** x)**{**  List L;  makenull\_list(&L);  **int** i;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  push\_back(&L, i);  return L;  **}**  **int** mark[MAX\_N];  **int** parent[MAX\_N];  void breath\_first\_search(Graph \*G, **int** k)**{**  Queue Q;  makenull\_queue(&Q);  push\_queue(&Q, k);  mark[k] = 1;  parent[k] = 0;  while(!empty\_queue(&Q))**{**  **int** x = top\_queue(&Q);  pop\_queue(&Q);  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  if(mark[y] == 0)**{**  push\_queue(&Q, y);  mark[y] = 1;  parent[y] = x;  **}**  **}**  **}**  **}**  void depth\_first\_search(Graph \*G, **int** k)**{**  Stack S;  makenull\_stack(&S);  push\_stack(&S, k);    while(!empty\_stack(&S))**{**  **int** x = top\_stack(&S);  pop\_stack(&S);  if(mark[x] == 0)**{**  mark[x] = 1;  List L = neighbors(G, x);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  if(mark[y] == 0)**{**  push\_stack(&S, y);  parent[y] = x;  **}**  **}**  **}**  **}**  **}**  void duyetsau\_dequy(Graph \*G, **int** k)**{**  if(mark[k] == 0)**{**  pr**int**f("%d\n", k);  mark[k] = 1;  List L = neighbors(G, k);  **int** i;  **for**(i = 1; i <= L.size; i++)**{**  **int** y = element\_at(&L, i);  duyetsau\_dequy(G, y);  **}**  **}**  **}**  **int** main()**{**  Graph G;  //freopen("dothi.txt", "r", stdin);  **int** i, n, m, u, v;  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);  **}**  **for**(i = 1; i <= G.n; i++)**{**  parent[i]= 0;  mark[i] = 0;  **}**      **for**(i = 1; i <= G.n; i++)  if(mark[i] == 0)  breath\_first\_search(&G, i);    **for**(i = 1; i <= G.n; i++)  pr**int**f("%d %d\n", i, parent[i]);  return 0;  **}** | /\* Cay duyet do thi  Cap nhat parent cua dinh  \*/  #include<stdio.h>  #define MAX\_Vertices 25  #define MAX\_Length 25  #define MAX\_Element 40 //**for** Stack  **typedef** struct  **{**  **int** A[MAX\_Vertices][MAX\_Vertices];  **int** n; //so luong dinh  **}** Graph;  //Ham khoi tao do thi Dinh-Dinh  void init\_Graph(Graph \*G, **int** n)  **{**  **int** i, j;  G->n = n; //so luong dinh =6, nhap vao;  **for**(i=1; i<=n; i++)//co the su sung n hoac G->n  **for**(j=1; j<=n; j++)//Nhu tren  G->A[i][j]=0;  **}**  //Ham them cung vao do thi Dinh-dinh  void add\_edge(Graph \*G, **int** x, **int** y)  **{**  G->A[x][y]=1;  G->A[y][x]=1;  **}**  //Ham kiem tra dinh ke nhau (lang gieng) lien thuoc  **int** adjacent(Graph \*G, **int** x, **int** y) //Kieu du lieu tra loi Yes No  **{**  return (G->A[x][y]!=0);  **}**  //Ham tinh bac cua mot dinh  **int** degree(Graph \*G,**int** x)  **{**  **int** deg=0, i;  **for**(i=1; i<= G->n; i++)  if(G->A[i][x]==1) //co the thay the bang hâm adjacent  //if(adjacent(G, i ,x))  deg++;  return deg;  **}**  //Kiem tra cac dinh lien ke, can cai dat danh sach  //Khai bao cau truc danh sach List  **typedef** struct  **{**  **int** data[MAX\_Length];  **int** size;  **}** List;  //Ham khoi tao danh sach rong  void make\_null(List \*list)  **{**  list->size = 0;  **}**  //Them mot phan tu (dinh) vao cuoi danh sach  void push\_back(List \*list, **int** x)  **{**  list->data[list->size] = x; //cho phan tu dau tien = 0 va tang no len sau  list->size++;//tang phan tu len  **}**  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  **int** element\_at(List \*list, **int** i)  **{**  return list->data[i-1]; //do mang bat dau tu vi tri khong, cho nen phai -1  **}**  //Tim lang gieng cua dinh x  List neighbors(Graph \*G, **int** x) //in ra mot danh sach, nen dung kieu List.  **{**  List L; //can khoi tao mot danh sach khi tim  make\_null(&L);  **int** i;  **for**(i=1; i<=G->n; i++)//giong nhu ham Degree  **{**  //duyet qua tat ca cac dinh, neu dinh nao di toi duoc dinh x thi them vao List  if(G->A[i][x] == 1)  //if(adjacent(G, i, x))  push\_back(&L, i); //dua mot dinh vao danh sach  **}**  return L;  **}**  //Cai dat ngan xep de tinh DFS  //Khai bao cau truc cua ngan xep  **typedef** struct  **{**  **int** data[MAX\_Element];  **int** size; //bien de lay phan tu cuoi cung trong ngan xep ra  **}** Stack;  //Khoi tao ngan xep rong  void make\_null\_stack(Stack \*stack)  **{**  stack->size = 0;  **}**  //Them mot phan tu vao trong ngan xep  void push\_stack(Stack \*stack, **int** x) //x: dinh them vao Stack  **{**  stack->data[stack->size] = x; //them x vao data, va chi so cua man tai Ngan xep link excel.  stack->size ++; //tang size len khi them  **}**  //Lay mot phan tu trong Stack  **int** top(Stack \*stack)  **{**  return stack->data[stack->size-1]; //so sanh vao excel, size & idex, size - 1 = idex  **}**  //Xoa mot phan tu thuoc Stack  void pop(Stack \*stack)  **{**  stack->size --;  **}**  //Kiem tra Stack rong  **int** empty(Stack \*stack)  **{**  return stack->size == 0;  **}**  //Thuat toan DFS duyet theo chieu sau  List depth\_first\_search(Graph \*G, **int** x, **int** parent[]) //khai bao them parent  **{**  Stack S;  make\_null\_stack(&S); //Khoi tao ham rong  push\_stack(&S, x); //dua dinh x vao  parent[x] = 0; //dinh bat dau, cha = 0  List list\_dfs;  make\_null(&list\_dfs);  **int** mark[MAX\_Vertices]; //mangx danh dau xem mot dinh da duoc duyet chua  //Khoi tao tat ca cac dinh chua duoc duyet  **int** i;  **for**(i=1; i<=G->n; i++)  mark[i]= 0; //Khi mark =0 thi dinh chua duoc duyet, nguoc lai  while(!empty(&S))  **{**  **int** u = top(&S); //Lay mot dinh ra  pop(&S); //Xoa dinh do khoi ngan xep khi da lay  if(mark[u] == 1)  continue;  // pr**int**f("Duyet: %d\n", u);  push\_back(&list\_dfs, u); //dua u vao danh sach thay vi duyet u  mark[u]= 1;  List list = neighbors(G, u); //Ham neighbors su dung kieu du du lieu List nen phai goi List ra  **for**(i=1; i<=list.size; i++) //lay phan tu trong list  **{**  **int** v = element\_at(&list, i);  if(mark[v] == 0)**{**  push\_stack(&S, v); //Dua v vao ngan xep  parent[v] = u;  **}**  **}**  **}**  return list\_dfs;  **}**  **int** main()  **{**  Graph G;  //freopen("Cayduyetdothi.txt", "r", stdin);  **int** n, m;  **int** i, j;  scanf("%d%d", &n, &m);  init\_Graph(&G, n);  **int** e, u, v;  **for**(e=1; e<=m; e++)  **{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v); //them cung  **}**  //Khai bao mang luu dinh parent cua mot dinh v trong qua trinh duyet  **int** parent[MAX\_Vertices];  **for**(i=1; i<= G.n; i++)**{**  parent[i]= -1; //ban dau chua co dinh cha, cho = -1    **}**      **int** mark\_dfs[MAX\_Vertices]; //kiem tra duyet cua tat ca do thi  **for**(i=1; i<=G.n; i++)  **{**  mark\_dfs[i] = 0;  **}**  **for**(i=1; i<=G.n; i++)  **{**  if(mark\_dfs[i] == 0) //neu chua duoc duyet  **{**  List dfs = depth\_first\_search(&G, i, parent);  **for**(j=1; j<=dfs.size; j++) //tien hanh in danh sach  **{**  **int** k = element\_at(&dfs, j);  // pr**int**f("%d\n", k); khong in ra su dung parent  mark\_dfs[k] = 1;  **}** //2 dong lap **for** khong su dung cung mot bien lap  **}**  **}**  **for**(u=1; u<= G.n; u++)  pr**int**f("%d %d\n", u, parent[u] );  return 0;  **}** | //Duyet do thi theo chieu sau - De quy - Tree  // TREE  // Do thi khong lien thong  #include<stdio.h>  #define MAX\_Vertices 25 //Cap phat so luong dinh  #define MAX\_Length 25  **typedef** struct  **{**  **int** A[MAX\_Vertices][MAX\_Vertices];  **int** n; //so luong dinh  **}** Graph; //cai dat cau truc  //Ham khoi tao do thi Dinh-Dinh  void init\_Graph(Graph \*G, **int** n)  **{**  **int** i, j;  G->n = n; //so luong dinh =6, nhap vao;  **for**(i=1; i<=n; i++)//co the su sung n hoac G->n  **for**(j=1; j<=n; j++)//Nhu tren  G->A[i][j]=0;  **}**  //Ham them cung vao do thi Dinh-dinh  void add\_edge(Graph \*G, **int** x, **int** y)  **{**  G->A[x][y]=1;  G->A[y][x]=1;  **}**  //Ham kiem tra dinh ke nhau (lang gieng) lien thuoc  **int** adjacent(Graph \*G, **int** x, **int** y) //Kieu du lieu tra loi Yes No  **{**  return (G->A[x][y]!=0);  **}**  //Ham tinh bac cua mot dinh  **int** degree(Graph \*G,**int** x)**{**  **int** deg=0, i;  **for**(i=1; i<= G->n; i++)  if(G->A[i][x]==1) //co the thay the bang hâm adjacent  //if(adjacent(G, i ,x))  deg++;  return deg;  **}**  //Kiem tra cac dinh lien ke, can cai dat danh sach  //Khai bao cau truc danh sach List lang gieng cua dinh x  **typedef** struct**{**  **int** data[MAX\_Length];  **int** size;  **}**List;  //Ham khoi tao danh sach rong  void make\_null(List \*list)**{**  list->size = 0;  **}**  //Them mot phan tu (dinh) vao cuoi danh sach  void push\_back(List \*list, **int** x)**{**  list->data[list->size] = x; //cho phan tu dau tien = 0 va tang no len sau  list->size++;//tang phan tu len  **}**  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  **int** element\_at(List \*list, **int** i)**{**  return list->data[i-1]; //do mang bat dau tu vi tri khong, cho nen phai -1  **}**  //Tim lang gieng cua dinh x  List neighbors(Graph \*G, **int** x)**{**  List L;  make\_null(&L);  **int** i;  **for**(i=1; i<=G->n; i++)  **{**  if(G->A[i][x] == 1)  push\_back(&L, i);  **}**  return L;  **}**  //Khai bao mang de su dung De Quy  **int** mark[MAX\_Vertices];  //Ho tro dung cay  **int** parent[MAX\_Vertices];  //Duyet do thi theo chieu sau - De Quy  void DFS\_Recursion(Graph \*G, **int** u, **int** p)**{** // p : parent  if(mark[u] == 1)  return;  //Duyet u  // pr**int**f("Duyet: %d\n", u);  parent[u] = p; //truyen vao cha cua u se la p  mark[u] = 1;  List list = neighbors(G, u);  **int** i;  **for**(i=1; i<=list.size; i++)**{**  **int** v = element\_at(&list, i);  DFS\_Recursion(G, v, u);  **}**  **}**  **int** main()  **{**  Graph G;  //freopen("DFS\_DeQuy.txt", "r", stdin);  **int** n, m;  scanf("%d%d",&n ,&m);  init\_Graph(&G, n);  **int** u, v, i;  //Doc cung tu tap tin  **for**(i=1; i<=m; i++)**{**  scanf("%d%d",&u ,&v);  add\_edge(&G, u, v);  **}**  **for**(i=1; i<=n; i++)**{**  mark[i]= 0; //danh dau tat ca cac dinh chua duoc duyet  parent[i] = -1 ; //tat ca cac dinh ban dau deu khong co CHA  **}**  //Do thi khong lien thong  **for**(i=1; i<=n; i++)  if(mark[i] == 0)  DFS\_Recursion(&G, i, 0);    //DFS\_Recursion(&G, 1, 0); //cha cua dinh 1 - dinh bat dau = 0  //in cha cua cac dinh ra  **for**(i=1; i<=n; i++)  pr**int**f("%d %d\n", i, parent[i]);  return 0;  **}** |

* **Bài 3 – Đồ thị liên thông – Qua đảo**

|  |
| --- |
| 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>  #define MAX\_ELEMENT 100  **typedef** struct**{**  **int** last;  **int** data[MAX\_ELEMENT];  **}**List;  void makenull\_list(List \*L)**{**  L->last = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->last] = x;  L->last++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->last;  **}**  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  **int** mark[MAX\_VERTICES];  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **for**(i = 1; i <= G->n; i++)  mark[i] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y] = 1;  G->A[y][x] = 1;  **}**  List neighbors(Graph \*G, **int** x)**{**  **int** i;  List list;  makenull\_list(&list);  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] == 1)  push\_back(&list, i);  return list;  **}**  void duyetsau\_dequy(Graph \*G, **int** x)**{**  **int** i;  if(mark[x] == 0)**{**  mark[x] = 1;  List L = neighbors(G, x);  **for**(i = 1; i <= L.last; i++)**{**  duyetsau\_dequy(G, element\_at(&L, i));  **}**  **}**    **}**  **int** main()**{**  Graph G;  // freopen("Test.txt", "r", stdin);  **int** u, v, j ,i, n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    **for**(j = 1; j <= m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**  **int** t = 0;  duyetsau\_dequy(&G, 1);  **for**(i = 1; i <= G.n; i++)  if(mark[i] == 0)**{**  t = 1;  pr**int**f("NO");  break;  **}**  if(t == 0)  pr**int**f("YES");  return 0;  **}** |

* **Bài 4 – Tôn ngộ không**

|  |
| --- |
| Tôn Ngộ Không là một trong các nhân vật chính của truyện Tây du ký. Khi còn ở Hoa Quả Sơn, Tôn Ngộ Không là vua của loài khỉ. Hoa Quả Sơn có rất nhiều cây ăn trái, nên loài khỉ rất thích. Do đặc tính của mình, khỉ không thích đi bộ mà chỉ thích chuyền từ cây này sang một cây khác. Tuy nhiên, nếu khoảng cách giữa hai cây quá xa thi chúng không thể chuyền qua lại được.  Đường đường là vua của loài khỉ, Tôn Ngộ Không muốn vạch ra một kế hoạch hái trái cây trên tất cả các cây có trên Hoa Quả Sơn mà không cần phải nhảy xuống đất. Tôn Ngộ Không dự định sẽ bắt đầu leo lên một cây, hái trái của cây này, sau đó chuyền qua một cây kế tiếp hái trái của này và tiếp tục như thế cho đến khi tất cả các cây đều được hái trái. Một cây có thể được chuyền qua chuyền lại nhiều lần.  Hãy giúp Tôn Ngộ Không kiểm tra xem kế hoạch này có thể thực hiện được không.  **Đầu vào (Input):**  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 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>  #define MAX\_ELEMENT 100  **typedef** struct**{**  **int** last;  **int** data[MAX\_ELEMENT];  **}**List;  void makenull\_list(List \*L)**{**  L->last = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->last] = x;  L->last++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->last;  **}**  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  **int** mark[MAX\_VERTICES];  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **for**(i = 1; i <= G->n; i++)  mark[i] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y] = 1;  G->A[y][x] = 1;  **}**  List neighbors(Graph \*G, **int** x)**{**  **int** i;  List list;  makenull\_list(&list);  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] == 1)  push\_back(&list, i);  return list;  **}**  void duyetsau\_dequy(Graph \*G, **int** x)**{**  **int** i;  if(mark[x] == 0)**{**  mark[x] = 1;  List L = neighbors(G, x);  **for**(i = 1; i <= L.last; i++)**{**  duyetsau\_dequy(G, element\_at(&L, i));  **}**  **}**  **}**  **int** main()**{**  Graph G;  // freopen("Test.txt", "r", stdin);  **int** u, v, j ,i, n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    **for**(j = 1; j <= m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**  **int** t = 0;  duyetsau\_dequy(&G, 1);  **for**(i = 1; i <= G.n; i++)  if(mark[i] == 0)**{**  t = 1;  pr**int**f("KHONG");  break;  **}**  if(t == 0)  pr**int**f("DUOC");  return 0;  **}** |

* **Bài 5: Kiểm tra chu trình – đồ thị vô hướng**

|  |
| --- |
| 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>  #define MAX\_ELEMENT 100  **typedef** struct**{**  **int** last;  **int** data[MAX\_ELEMENT];  **}**List;  void makenull\_list(List \*L)**{**  L->last = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->last] = x;  L->last++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->last;  **}**  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  **int** mark[MAX\_VERTICES];  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **for**(i = 1; i <= G->n; i++)  mark[i] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y] = 1;  G->A[y][x] = 1;  **}**  List neighbors(Graph \*G, **int** x)**{**  **int** i;  List list;  makenull\_list(&list);  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] == 1)  push\_back(&list, i);  return list;  **}**  #define white 0  #define black 1  #define gray 2  **int** color[MAX\_VERTICES];  **int** cycle;  void dfs(Graph \*G, **int** x, **int** parent)**{**  **int** i;  color[x] = gray;  List L = neighbors(G, x);  **for**(i = 1; i <= L.last; i++)**{**  **int** y = element\_at(&L, i);  if(y == parent)  continue;  if(color[y] == gray)**{**  cycle = 1;  return;  **}**  if(color[y] == white)  dfs(G, y, x);  **}**  color[x] = black;  **}**  **int** contains\_cycle(Graph \*G)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  color[i] = white;  cycle = 0;  **for**(i = 1; i <= G->n; i++)  if(color[i] == white)  dfs(G, i, 0);  return cycle;  **}**  **int** main()**{**  Graph G;  //freopen("Test.txt", "r", stdin);  **int** u, v, j , n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    **for**(j = 1; j <= m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**    if(contains\_cycle(&G) == 0)**{**  pr**int**f("NO");  **}**else  pr**int**f("YES");  return 0;  **}** |

* **Bài 6: Thuyền trưởng Haddock**

|  |
| --- |
| Thuyền trưởng Haddock (truyện T**int**in) là một người luôn say xỉn. Vì thế đôi khi T**int**in không biết ông ta đang say hay tỉnh. Một ngày nọ, T**int**in hỏi ông ta về cách uống. Haddock nói như thế này: Có nhiều loại thức uống (soda, wine, water, …), tuy nhiên Haddock lại tuân theo quy tắc “*để uống một loại thức uống nào đó cần phải uống tất cả các loại thức uống tiên quyết của nó*”. Ví dụ: để uống rượu (wine), Haddock cần phải uống soda và nước (water) trước. Vì thế muốn say cũng không phải dễ !  Cho danh sách các thức uống và các thức uống tiên quyết của nó. Hãy xét xem Haddock có thể nào say không ? Để làm cho Haddock say, ông ta phải uống hết tất cả các thức uống.  **Đầu vào (Input):**  Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:   * Dòng đầu tiên chứa 2 số nguyên **n** và **m**, tương ứng là số thức uống và số điều kiện tiên quyết . * **m** dòng tiếp theo mỗi dòng chứa 2 số nguyên **u** **v** nói rằng thức uống **u** là tiên quyết của thức uống **v**.   **Đầu ra (Output):**   * Nếu Haddock có thể say in ra YES, ngược lại in ra NO. |
| #include<stdio.h>  #define MAX\_ELEMENT 100  **typedef** struct**{**  **int** last;  **int** data[MAX\_ELEMENT];  **}**List;  void makenull\_list(List \*L)**{**  L->last = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->last] = x;  L->last++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->last;  **}**  #define MAX\_VERTICES 20  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  G->A[y][x]++;  **}**  List neighbors(Graph \*G, **int** x)**{**  **int** i;  List list;  makenull\_list(&list);  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] == 1)  push\_back(&list, i);  return list;  **}**  #define white 0  #define black 1  #define gray 2  **int** color[MAX\_VERTICES];  **int** cycle;  void dfs(Graph \*G, **int** x, **int** parent)**{**  **int** i, j;  color[x] = gray;  List L = neighbors(G, x);  **for**(i = 1; i <= L.last; i++)**{**  **int** y = element\_at(&L, i);  if(y == parent)  continue;  if(color[y] == gray)**{**  cycle = 1;  return;  **}**  if(color[y] == white)  dfs(G, y, x);  **}**  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  if(G->A[i][j] > 1)**{**  cycle = 1;  return;  **}**  color[x] = black;  **}**  **int** contains\_cycle(Graph \*G)**{**  **int** i;  **for**(i = 1; i <= G->n; i++)  color[i] = white;  cycle = 0;  **for**(i = 1; i <= G->n; i++)  if(color[i] == white)  dfs(G, i, 0);  return cycle;  **}**  **int** main()**{**  Graph G;  //freopen("dothi\_haddock.txt", "r", stdin);  **int** u, v, j, n, m;  scanf("%d%d", &n, &m);  init\_graph(&G, n);    **for**(j = 1; j <= m; j++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**    if(contains\_cycle(&G) == 1)  pr**int**f("NO");  else  pr**int**f("YES");    return 0;  **}** |

* **Bài 7: Phân chia đội bóng**

|  |
| --- |
| David là huấn luyện viên của một đội bóng gồm **N** thành viên. David muốn chia đội bóng thành hai nhóm. Để tăng tính đa dạng của các thành viên trong nhóm, David quyết định không xếp hai thành viên đã từng thi đấu với nhau vào chung một nhóm. Bạn hãy lập trình giúp David phân chia đội bóng.  **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 thành viên và số cặp thành viên đã từng thi đấu với nhau. * M dòng tiếp theo mỗi dòng chứa 2 số nguyên **u** **v** nói rằng 2 thành viên **u** và **v** đã từng thi đấu chung với nhau.   **Đầu ra (Output):**   * Nếu phân chia được, hãy in ra các thành viên của mỗi nhóm. Nhóm của thành viên 1 sẽ được in trước, nhóm còn lại in ra sau. Các thành viên trong nhóm được in ra theo thứ tự tăng dần và in trên 1 dòng. Hai thành viên cách nhau 1 khoảng trắng. * Nếu không thể phân chia, in ra **IMPOSSIBLE** |
| #include<stdio.h>  // CTDL Danh Sach  #define MAX\_ELEMENTS 100  **typedef** **int** ElementType;  **typedef** struct**{**  ElementType data[MAX\_ELEMENTS];  **int** size;  **}**List;  void make\_null(List\* L)**{**  L->size=0;  **}**  void push\_back(List\* L, ElementType x)**{**  L->data[L->size]=x;  L->size++;  **}**  ElementType element\_at(List\* L, **int** i)**{**  return L->data[i-1];  **}**  **int** count\_list(List\* L)**{**  return L->size;  **}**  #define MAX\_VERTICES 100  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph\* G, **int** n)**{**  **int** i,j;  G->n=n;  **for**(i=1;i<=n;i++)  **for**(j=1;j<=n;j++)  G->A[i][j]=0;  **}**  void add\_edge(Graph\* G, **int** x, **int** y)**{**  /\* vo huong  G->A[x][y]=1;  G->A[y][x]=1;  \*/  // da cung  G->A[x][y]+=1;  G->A[y][x]+=1;  /\* co huong  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)**{**  /\* don cung  **int** y, deg=0;  **for**(y=1;y<=G->n;y++)  if(G->A[x][y]>0)  deg++;  return deg;  \*/  // da cung  **int** y, deg=0;  **for**(y=1; y<=G->n;y++)  deg+=G->A[x][y];  return deg;  **}**  List neighbors(Graph\* G, **int** x)**{**  **int** y;  List list;  make\_null(&list);  **for**(y=1; y<=G->n;y++)  if(adjacent(G,x,y))  push\_back(&list,y);  return list;  **}**  #define MAX\_VERTICES 100  **int** color[MAX\_VERTICES];  **int** fail;  void colorize(Graph\* G, **int** x, **int** c)**{**  if(color[x]==-1)**{**  color[x]=c;  List list = neighbors(G,x);  **int** j;  **for**(j=1; j<=list.size;j++)**{**  **int** y = element\_at(&list,j);  colorize(G,y,!c);  **}**  **}**  else  if(color[x]!=c)  fail=1;  **}**  **int** is\_bigraph(Graph\* G)**{**  **int** j;  **for**(j=1; j<=G->n; j++)  color[j]=-1;  fail=0;  colorize(G,1,0);  return !fail;  **}**  **int** main()**{**  Graph G;  **int** n, m, v, x, y;  scanf("%d%d",&n,&m);  init\_graph(&G,n);  **for**(v=1;v<=m;v++)**{**  scanf("%d%d",&x,&y);  add\_edge(&G,x,y);  **}**  **int** kq = is\_bigraph(&G);  if(kq==1)**{**  //pr**int**f("Doi 1: ");  **for**(v=1; v<=m+1; v++)  if(color[v]==0)  pr**int**f("%d ",v);    pr**int**f("\n");    //pr**int**f("Doi 2: ");  **for**(v=1; v<=m+1; v++)  if(color[v]==1)  pr**int**f("%d ",v);  **}**  else  pr**int**f("IMPOSSIBLE");  return 0;  **}** |

* **Bài 8 – Kiểm tra tính liên thông mạnh – đếm số BPLT mạnh**

|  |  |
| --- | --- |
| 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**. | 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. |
| #define maxe 100  #define maxn 100  #define maxm 100  #include <stdio.h>  **int** mark[maxn], num[maxn], min\_num[maxn], on\_stack[maxn];  **int** idx=1;  **int** cnt =0;  **typedef** struct**{**  **int** data[maxe];  **int** size;  **}**Stack;  Stack S;  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** data[maxn];  **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++;  **}**  **int** element\_at(List\* L, **int** p)**{**  return L->data[p-1];  **}**  **int** size(List\* L)**{**  return L->size;  **}**  **typedef** struct**{**  **int** n,m;  **int** a[maxn][maxm];  **}**Graph;  void init\_graph(Graph\* pG, **int** n, **int** m)**{**  **int** i,j;  pG->m = m;  pG->n = n;  **for** (i=1; i<=n; i++)  **for** (j=1; j<=m; j++)  pG->a[i][j]=0;  **}**  void add\_edge(Graph\* pG,**int** e, **int** x, **int** y)**{**  pG->a[x][e]=1;  pG->a[y][e]=-1;  **}**  **int** degree(Graph\* pG, **int** x)**{**  **int** i,deg=0;  **for** (i=1; i<=pG->m; i++)  deg= deg + pG->a[x][i];  return deg;  **}**  **int** adjacent(Graph\* pG, **int** x, **int** y)**{**  **int** i,j;  **for** (i=1; i<=pG->m; i++)  **for** (j=1; j<=pG->m; j++)  if ((pG->a[x][i]==1)&&(pG->a[y][i]==-1)&&(pG->a[x][j]==1)&&(pG->a[y][j]==-1))  return 1;  return 0;  **}**  List neighbors(Graph\* G,**int** x)**{**  **int** i;  List list;  make\_null(&list);  **for** (i=1; i<= G->n; i++)  if (i!=x)  if (adjacent(G,x,i)) push\_back(&list,i);  return list;  **}**  void traversal(Graph\* G, **int** x)**{**  **int** i,y;  if (mark[x]==1)  return;  pr**int**f("%d \n",x);  mark[x] = 1;  List list = neighbors(G,x);  **for** (i=1 ; i<=list.size ; i++)**{**  y = element\_at(&list,i);  traversal(G,y);  **}**  **}**  **int** min(**int** a, **int** b)**{**  if (a>b) return b;  else return a;  **}**  void strong\_connect(Graph\* G, **int** x)**{**  **int** i;  num[x] = min\_num[x] = idx; idx ++;  push(&S, x);  on\_stack[x] = 1;  List list = neighbors(G,x);  **for** (i = 1; i<= list.size; i++)**{**  **int** y = element\_at(&list,i);  if (num[y] < 0)**{**  strong\_connect(G,y);  min\_num[x] = min(min\_num[x], min\_num[y]);  **}**else if (on\_stack[y])  min\_num[x] = min(min\_num[x], num[y]);  **}**  if (num[x] == min\_num[x])**{**  cnt ++;  **int** w;  do**{**  w = top(&S); pop(&S);  on\_stack[w]= 0;  **}** while (w!= x);  **}**  **}**  **int** main()**{**  //freopen("dt.txt","r",stdin);  Graph G;  List list;  make\_null(&list);  **int** i,n,m,x,y;  scanf("%d%d",&n,&m);  init\_graph(&G,n,m);    **for** (i=1;i<=m;i++)**{**  scanf("%d%d",&x,&y);  add\_edge(&G,i,x,y);  **}**  **for** (i =1; i<=n ; i++)**{**  num[i]= -1;  on\_stack[i]=0;**}**  idx = 1;  make\_null\_stack(&S);  **for** (i =1; i<=n ; i++)**{**  if (num[i]==-1)  strong\_connect(&G, i);  **}**  if (cnt ==1 ) pr**int**f("strong connected");  else pr**int**f("unconnected");  **}** | #define maxe 100  #define maxn 100  #define maxm 100  #include <stdio.h>  **int** mark[maxn], num[maxn], min\_num[maxn], on\_stack[maxn];  **int** idx=1;  **int** cnt =0;  **typedef** struct**{**  **int** data[maxe];  **int** size;  **}**Stack;  Stack S;  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** data[maxn];  **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++;  **}**  **int** element\_at(List\* L, **int** p)**{**  return L->data[p-1];  **}**  **int** size(List\* L)**{**  return L->size;  **}**  **typedef** struct**{**  **int** n,m;  **int** a[maxn][maxm];  **}**Graph;  void init\_graph(Graph\* pG, **int** n, **int** m)**{**  **int** i,j;  pG->m = m;  pG->n = n;  **for** (i=1; i<=n; i++)  **for** (j=1; j<=m; j++)  pG->a[i][j]=0;  **}**  void add\_edge(Graph\* pG,**int** e, **int** x, **int** y)**{**  pG->a[x][e]=1;  pG->a[y][e]=-1;  **}**  **int** degree(Graph\* pG, **int** x)**{**  **int** i,deg=0;  **for** (i=1; i<=pG->m; i++)  deg= deg + pG->a[x][i];  return deg;  **}**  **int** adjacent(Graph\* pG, **int** x, **int** y)**{**  **int** i,j;  **for** (i=1; i<=pG->m; i++)  **for** (j=1; j<=pG->m; j++)  if ((pG->a[x][i]==1)&&(pG->a[y][i]==-1)&&(pG->a[x][j]==1)&&(pG->a[y][j]==-1))  return 1;  return 0;  **}**  List neighbors(Graph\* G,**int** x)**{**  **int** i;  List list;  make\_null(&list);  **for** (i=1; i<= G->n; i++)  if (i!=x)  if (adjacent(G,x,i)) push\_back(&list,i);  return list;  **}**  void traversal(Graph\* G, **int** x)**{**  **int** i,y;  if (mark[x]==1)  return;  pr**int**f("%d \n",x);  mark[x] = 1;  List list = neighbors(G,x);  **for** (i=1 ; i<=list.size ; i++)**{**  y = element\_at(&list,i);  traversal(G,y);  **}**  **}**  **int** min(**int** a, **int** b)**{**  if (a>b) return b;  else return a;  **}**  void strong\_connect(Graph\* G, **int** x)**{**  **int** i;  num[x] = min\_num[x] = idx; idx ++;  push(&S, x);  on\_stack[x] = 1;  List list = neighbors(G,x);  **for** (i = 1; i<= list.size; i++)**{**  **int** y = element\_at(&list,i);  if (num[y] < 0)**{**  strong\_connect(G,y);  min\_num[x] = min(min\_num[x], min\_num[y]);  **}**else if (on\_stack[y])  min\_num[x] = min(min\_num[x], num[y]);  **}**  if (num[x] == min\_num[x])**{**  cnt ++;  **int** w;  do**{**  w = top(&S); pop(&S);  on\_stack[w]= 0;  **}** while (w!= x);  **}**  **}**  **int** main()**{**  //freopen("dt.txt","r",stdin);  Graph G;  List list;  make\_null(&list);  **int** i,n,m,x,y;  scanf("%d%d",&n,&m);  init\_graph(&G,n,m);    **for** (i=1;i<=m;i++)**{**  scanf("%d%d",&x,&y);  add\_edge(&G,i,x,y);  **}**  **for** (i =1; i<=n ; i++)**{**  num[i]= -1;  on\_stack[i]=0;**}**  idx = 1;  make\_null\_stack(&S);  **for** (i =1; i<=n ; i++)**{**  if (num[i]==-1)  strong\_connect(&G, i);  **}**  pr**int**f("%d",cnt);  **}** |

* **Bài 9 – Come and Go**

|  |
| --- |
| Trong một thành phố có **N** địa điểm được nối với nhau bằng **M** con đường 1 chiều và 2 chiều. Yêu cầu tối thiểu của một thành phố là **từ địa điểm này bạn phải có thể đi đến một địa điểm khác bất kỳ**.  Hãy viết chương trình kiểm tra xem các con đường của thành phố có thoả mãn yêu cầu tối thiểu này không. Nếu có in ra OKIE, ngược lại in ra NO.  (nguồn: UVA Online Judge, Problem 11838)  Dữ liệu đầu vào có dạng như sau:  4 5 1 2 1 1 3 2 2 4 1 3 4 1 4 1 2  Trong ví dụ này, có 4 địa điểm và 5 con đường, mỗi con đường có dạng a b p,  trong đó a, b là các địa điểm; và nếu p = 1, con đường đang xét là đường 1 chiều, ngược lại nó là đường 2 chiều.  **Đầu vào (Input):**  Dữ liệu đầu vào được nhập từ bàn phím (stdin) với định dạng:  - Dòng đầu tiên chứa 2 số nguyên N và M, tương ứng là số địa điểm và số con đường.  - M dòng tiếp theo mỗi dòng chứa 3 số nguyên a, b, p. Nếu p = 1, con đường (a, b) là con đường 1 chiều, ngược lại nếu p = 2, con đường (a, b) là con đường 2 chiều.  **Đầu ra (Output):**  In ra màn hình OKIE nếu các con đường của thành phố có thoả mãn yêu cầu, ngược lại in ra NO.  Xem thêm ví dụ bên dưới.  **Gợi ý:**   * Xây dựng đồ thị có hướng từ dữ liệu các con đường và các địa điểm   + Địa điểm ~ đỉnh   + Đường 1 chiều ~ cung   + Đường 2 chiều ~ 2 cung * Áp dụng giải thuật kiểm tra đồ thị có liên thông mạnh hay không. |
| #include <stdio.h>  #define INFINITY 9999  #define MAXN 500  #define NO\_EDGE 0  **typedef** struct **{**  **int** n;  **int** m;  **int** L[MAXN][MAXN];  **}** Graph;  void init\_graph(Graph\* G, **int** n, **int** m) **{**  G->n = n;  G->m = m;  **int** i, j;  **for** (i = 1; i <= n; i++) **{**  **for** (j = 1; j <= m; j++) **{**  G->L[i][j] = NO\_EDGE;  **}**  **}**  **}**  #include <stdio.h>  #define MAX\_ELEMENTS 100  // 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;  **}**  // LIST  #define MAX\_ELEMENTS 100  **typedef** **int** ElementType;  **typedef** struct **{**  ElementType data[MAX\_ELEMENTS];  **int** size;  **}** List;  void make\_null(List\* L) **{**  L->size = 0;  **}**  void push\_back(List\* L, ElementType x) **{**  L->data[L->size] = x;  L->size++;  **}**  ElementType element\_at(List\* L, **int** i) **{**  return L->data[i-1];  **}**  **int** count\_list(List\* L) **{**  return L->size;  **}**  **int** empty\_list(List\* L) **{**  return L->size == 0;  **}**  **int** adjacent(Graph\* G, **int** x, **int** y) **{**  **int** e;  **for** (e = 1; e <= G->m; e++)  if (G->L[x][e] == 1 && G->L[y][e] == 1)  return 1;  return 0;  **}**  **int** degree(Graph\* G, **int** x) **{**  **int** e, deg = 0;  **for** (e = 1; e <= G->m; e++)  if (G->L[x][e] == 1)  deg++;  return deg;  **}**  void pr**int**\_list(List L) **{**  **int** i;  **for** (i = 1; i <= L.size; i++) **{**  pr**int**f("%d ", element\_at(&L, i));  **}**  **}**  List neighbors(Graph\* G, **int** x) /\* Tim cac dinh ke cua dinh x \*/  **{**  //Graph G;  **int** y;  List list;  make\_null(&list);  **for**(y = 1; y <= G->n; y++)  if(adjacent(G,x,y))  push\_back(&list,y);  return list;  **}**  void add\_edge(Graph\* G, **int** x, **int** y) **{**  G->L[x][y] = 1;  **}**  **int** min(**int** a, **int** b) **{**  if (a <= b) return a;  else return b;  **}**  #define MAX\_VERTICES 1000  **int** on\_stack[MAX\_VERTICES];  **int** num[MAX\_VERTICES];  **int** min\_num[MAX\_VERTICES];  **int** k, count\_pop = 0;  Stack S;  void strong\_connect(Graph\* G, **int** x) **{**  num[x] = k;  min\_num[x] = k;  k++;  push(&S, x);  on\_stack[x] = 1;    List list = neighbors(G, x);    **int** j;  **for** (j = 1; j <= list.size; j++) **{**  **int** y = element\_at(&list, j);  if (num[y] < 0) **{**  strong\_connect(G, y);  min\_num[x] = min(min\_num[x], min\_num[y]);  **}**  else if (on\_stack[y]) **{**  min\_num[x] = min(min\_num[x], num[y]);  **}**  **}**  if (num[x] == min\_num[x]) **{**  **int** w;  do **{**  w = top(&S);  pop(&S);  count\_pop++;  // on\_stack[w] == 0;  **}** while (w != x);  **}**  **}**  **int** main() **{**  // freopen("dothi.txt", "r", stdin);  Graph G;  **int** n, m;  **int** e, u, v, t;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);    **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &t);  if (t == 1) **{**  add\_edge(&G, u, v);  **}**  else if (t == 2) **{**  add\_edge(&G, u, v);  add\_edge(&G, v, u);  **}**  **}**  // Khoi tao num[] va on\_stack[]  **for** (v = 1; v <= n; v++) **{**  num[v] = -1;  on\_stack[v] = 0;  **}**  // Kiem tra dinh treo  **int** dk = 1;  **for** (v = 1; v <= n; v++) **{**  List list = neighbors(&G, v);  if (empty\_list(&list)) **{**  pr**int**f("NO");  dk = 0;  break;  **}**  **}**  make\_null\_stack(&S);  k = 1;  **for** (v = 1; v <= G.n; v++) **{**  if (num[v] == -1) **{**  strong\_connect(&G, v);  **}**  **}**  if (dk == 1) **{**  if (count\_pop == n) **{**  pr**int**f("OKIE");  **}**  else **{**  pr**int**f("NO");  **}**  **}**      return 0;  **}** |

* **BUỔI 3**
* **Bài 1 – Tìm 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 từ 1 đến n, in ra -1. |
| #include <stdio.h>  #define MAX\_VERTICES 50  #define MAX\_EDGES 50  #define NO\_EDGE -1  **typedef** struct **{**  **int** n, m;  **int** A[MAX\_VERTICES][MAX\_EDGES];  **}** Graph;  void init\_graph(Graph\* G, **int** n, **int** m) **{**  **int** i, j;  G->n = n;  G->m = m;  **for** (i = 1; i <= n; i++)  **for** (j = 1; j <= m; j++)  G->A[i][j] = NO\_EDGE;  **}**  void add\_edge(Graph\* G, **int** x, **int** y, **int** w) **{**  G->A[x][y] = w;  **}**  #define INFINITY 9999999  **int** mark[MAX\_VERTICES];  **int** pi[MAX\_VERTICES];  **int** p[MAX\_VERTICES];  void Dijkstra(Graph\* G, **int** s) **{**  **int** i, j, it;  **for** (i = 1; i <= G->n; i++) **{**  pi[i] = INFINITY;  mark[i] = 0;  **}**  pi[s] = 0;  p[s] = -1;  **for** (it = 1; it < G->n; it++) **{**  **int** min\_pi = INFINITY;  **for** (j = 1; j <= G->n; j++)  if (mark[j] == 0 && pi[j] < min\_pi) **{**  min\_pi = pi[j];  i = j;  **}**  mark[i] = 1;    **for** (j = 1; j <= G->n; j++)  if (G->A[i][j] != NO\_EDGE && mark[j] == 0) **{**  if (pi[i] + G->A[i][j] < pi[j]) **{**  pi[j] = pi[i] + G->A[i][j];  p[j] = i;  **}**  **}**  **}**  **}**  **int** main() **{**  Graph G;  //freopen("Test.txt", "r", stdin);  **int** n, m, u, v, e, w;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G,u,v,w);  **}**    Dijkstra(&G, 1);  if(pi[n]>0)**{**  pr**int**f("%d", pi[n]);  **}**  else pr**int**f("-1");  return 0;  **}** |

* **Bài 2 – Kiểm tra chu trình và ứng dụng đường đi ngắn nhất.**

|  |  |
| --- | --- |
| 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** | "*Ngưu Lang là vị thần chăn trâu của Ngọc Hoàng Thượng đế, vì say mê một tiên nữ phụ trách việc dệt vải tên là Chức Nữ nên bỏ bễ việc chăn trâu, để trâu đi nghênh ngang vào điện Ngọc Hư. Chức Nữ cũng vì mê tiếng tiêu của Ngưu Lang nên trễ nải việc dệt vải. Ngọc Hoàng giận dữ, bắt cả hai phải ở cách xa nhau, người đầu sông Ngân, kẻ cuối sông*.  *Mỗi năm một lần, Sau đó, Ngọc Hoàng thương tình nên ra ơn cho hai người mỗi năm được gặp nhau vào ngày 7 tháng Bảy âm lịch. Khi tiễn biệt nhau, Ngưu Lang và Chức Nữ khóc sướt mướt. Nước mắt của họ rơi xuống trần hóa thành cơn mưa và được người dưới trần gian đặt tên là mưa ngâu.*" (Theo wikipedia.com)  Để gặp được nhau vào ngày 7/7, Ngưu Lang và Chức Nữ phải nhờ đến bầy quạ đen bắt cầu (gọi là Ô kiều) cho mình đi qua để gặp nhau.  Sông Ngân Hà có n ngôi sao, giả sử được đánh số từ 1 đến n. Ngưu Lang ở tại ngôi sao Ngưu (Altair), được đánh số 1, còn Chức Nữ ở tại ngôi sao Chức Nữ (Vega) được đánh số n. Để bắt được một cây cầu từ ngôi sao này sang ngôi sao kia cần một số lượng quạ nào đó. Một khi con quạ ở cây cầu nào thì phải ở đó cho đến khi Ngưu Lang và Chức Nữ gặp được nhau.  Quạ thì càng ngày càng hiếm, nên Ngưu Lang và Chức Nữ phải tính toán sao cho số lượng quạ ít nhất có thể.  Hãy giúp Ngưu Lang và Chức Nữ viết chương trình tính xem cần phải nhờ đến ít nhất bao nhiêu con quạ để bắt cầu cho họ gặp nhau.  **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.  **Đầu ra (Output):**  In ra màn hình số lượng quạ cần thiết. |
| #include <stdio.h>  #define MAXN 1000  #define INFINITY 9999999  **int** pi[MAXN];  **int** p[MAXN];  **typedef** struct **{**  **int** u, v;  **int** w;  **}** Edge;  **typedef** struct **{**  **int** n, m;  Edge edges[MAXN];  **}** Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = 0;  **}**  void add\_edge(Graph \*G, **int** u, **int** v, **int** w)**{**  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->edges[G->m].w = w;  G->m++;  **}**    void BellmanFord(Graph \*G, **int** s)**{**  **int** i, it,k;  **for**(i=1; i<= G->n; i++)**{**  pi[i] = INFINITY;  **}**  pi[s] = 0;  p[s] = -1;  **for**(it = 1; it < G->n; it++)**{**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pi[v] = pi[u] + w;  p[v] = u;  **}**  **}**  **}**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pr**int**f("negative cycle"); break;  **}** else **{**  pr**int**f("ok"); break;  **}**  **}**  **}**  **int** main()**{**  // freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, e, w;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G,u,v,w);  **}**  BellmanFord(&G, 1);  // pr**int**f("%d", pi[4]);  /\* **for**(i=1; i<=G.n;i++)  pr**int**f("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);\*/  **}** | #include <stdio.h>  #define MAXN 1000  #define INFINITY 9999999  **int** pi[MAXN];  **int** p[MAXN];  **typedef** struct **{**  **int** u, v;  **int** w;  **}** Edge;  **typedef** struct **{**  **int** n, m;  Edge edges[MAXN];  **}** Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = 0;  **}**  void add\_edge(Graph \*G, **int** u, **int** v, **int** w)**{**  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->edges[G->m].w = w;  G->m++;  **}**    void BellmanFord(Graph \*G, **int** s)**{**  **int** i, it,k;  **for**(i=1; i<= G->n; i++)**{**  pi[i] = INFINITY;  **}**  pi[s] = 0;  p[s] = -1;  **for**(it = 1; it < G->n; it++)**{**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pi[v] = pi[u] + w;  p[v] = u;  **}**  **}**  **}**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pr**int**f("negative cycle"); break;  **}** else **{**  // pr**int**f(""); break;  **}**  **}**  **}**  **int** main()**{**  // freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, e,i, w;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G,u,v,w);  **}**  BellmanFord(&G, 1);  pr**int**f("%d", pi[4]);  **for**(i=1; i<=G.n;i++)  // pr**int**f("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);  return 0;  **}** |

* **Bài 3 - Ứng dụng đường đi ngắn nhất**
* **Bài 4 – Mê cung trọng số**

|  |
| --- |
| Cho một mê cung số được biểu diễn bằng một mảng 2 chiều chứa các con số từ 0 đến 9.    Một con robot được đặt tại góc trên bên trái của mê cung và muốn đi đến góc dưới bên phải của mê cung. Con robot có thể đi lên, xuống, qua trái và qua phải 1 ô. Chi phí để đi đến một ô bằng với con số bên trong ô đó.  Hãy tìm cách giúp con robot đi đến ô **góc dưới phải** sao cho tổng chi phí thấp nhất.  Đường đi có chi phí thấp nhất cho ví dụ này là 24.  **Đầu vào (Input):**  Dữ liệu đầu vào được nhập từ bàn phím với định dạng:   * Dòng dầu chứa 2 số nguyên M N (M: số hàng, N: số cột) * M dòng tiếp theo mô tả các số trong mê cung   **Đầu ra (Output):**  In ra màn hình chi phí thấp nhất để con robot đi từ góc trên bên trái về góc dưới bên phải. Ví dụ trên, cần in ra màn hình: 24.  **Gợi ý:**  Mô hình hoá bài toán về đồ thị có hướng   * Đỉnh ~ ô * Cung ~ hai ô cạnh nhau * Trọng số cung (u, v) = giá trị của ô tương ứng với đỉnh v.   Xem tài liệu thực hành để biết cách đặt tên cho các ô. |
| #include<stdio.h>  #define MAX\_VERTICES 100  #define NO\_EDGE 0  #define INFINITY 99999  **int** w[MAX\_VERTICES];  **int** u[MAX\_VERTICES];  **int** mark[MAX\_VERTICES], pi[MAX\_VERTICES],p[MAX\_VERTICES];  **typedef** struct**{**  **int** n;  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}**Graph;  void init\_graph(Graph\* G, **int** n)**{**  **int** i,j;  G->n=n;  **for** (i=1; i<=n; i++)  **for**(j=1; j<=n; j++)  G->A[i][j]=NO\_EDGE;  **}**  void add\_edge(Graph\* G, **int** u,**int** v,**int** w)**{**  G->A[u][v]=w;  **}**  void Dijkstra(Graph\* G,**int** s)**{**  **int** i,j,it;  **for**(i=1;i<=G->n;i++)**{**  pi[i]=INFINITY;  mark[i]=0;  **}**  pi[s]=w[s];  p[s]=-1;    **for**(it=1; it<G->n; it++)**{**  **int** min\_pi=INFINITY;  **for**(j=1; j<=G->n; j++)  if(mark[j]==0 && pi[j]<min\_pi)**{**  min\_pi=pi[j];  i=j;  **}**  mark[i]=1;  **for**(j=1; j<=G->n; j++)  if(G->A[i][j]!=NO\_EDGE && mark[j]==0)**{**  if(pi[i]+G->A[i][j]<pi[j])**{**  pi[j]=pi[i]+G->A[i][j];  p[j]=i;  **}**  **}**  **}**  **}**  **int** main ()**{**  Graph G;  **int** i,j,m,n,u,e,k;  //freopen("mecung.txt", "r", stdin);  scanf("%d%d", &m, &n);    **for**(i=0;i<m;i++)  **for**(j=0;j<n;j++)**{**  u=(i\*n+j)+1;  scanf("%d",&e);  w[u]=e;  **}**  init\_graph(&G,n\*m);  **int** di[]=**{**-1,1,0,0**}**;  **int** dj[]=**{**0,0,-1,1**}**;    **for**(i=0;i<m;i++)  **for**(j=0;j<n;j++)**{**  u=(i\*n+j)+1;  **for**(k=0;k<4;k++)**{**  **int** ii=i+di[k];  **int** jj=j+dj[k];  if(ii>=0 && ii<n && jj>=0 && jj<m)**{**  **int** v=(ii\*n+jj)+1;  add\_edge(&G ,v,u,w[u]);  **}**  **}**  **}**  Dijkstra(&G,1);  pr**int**f("%d",pi[G.n]);  return 0;  **}** |

* **Bài 5 – Bellman – Ford pi và pi.**

|  |
| --- |
| 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 (-100 < w <= 100).  Viết chương trình tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn 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. * m dòng tiếp theo mỗi dòng chứa 3 số nguyên u, v, w mô tả cung (u, v) có trọng số w.   Dữ liệu được đảm bảo không tồn tại chu trình âm.  **Đầu ra (Output):**  In ra màn hình các giá trị pi và p của các đỉnh theo thứ tự 1, 2, ..., n. |
| #include <stdio.h>  #define MAXN 1000  #define INFINITY 9999999  **int** pi[MAXN];  **int** p[MAXN];  **typedef** struct **{**  **int** u, v;  **int** w;  **}** Edge;  **typedef** struct **{**  **int** n, m;  Edge edges[MAXN];  **}** Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = 0;  **}**  void add\_edge(Graph \*G, **int** u, **int** v, **int** w)**{**  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->edges[G->m].w = w;  G->m++;  **}**    void BellmanFord(Graph \*G, **int** s)**{**  **int** i, it,k;  **for**(i=1; i<= G->n; i++)**{**  pi[i] = INFINITY;  **}**  pi[s] = 0;  p[s] = -1;  **for**(it = 1; it < G->n; it++)**{**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pi[v] = pi[u] + w;  p[v] = u;  **}**  **}**  **}**  **}**  **int** main()**{**  // freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, e, w;  **int** i;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G,u,v,w);  **}**  BellmanFord(&G, 1);  **for**(i=1; i<=G.n;i++)  pr**int**f("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);  **}** |

* **Bài 6 – Bellman – Ford**

|  |
| --- |
| 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>  #define MAXN 1000  #define INFINITY 9999999  **int** pi[MAXN];  **int** p[MAXN];  **typedef** struct **{**  **int** u, v;  **int** w;  **}** Edge;  **typedef** struct **{**  **int** n, m;  Edge edges[MAXN];  **}** Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = 0;  **}**  void add\_edge(Graph \*G, **int** u, **int** v, **int** w)**{**  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->edges[G->m].w = w;  G->m++;  **}**    void BellmanFord(Graph \*G, **int** s)**{**  **int** i, it,k;  **for**(i=1; i<= G->n; i++)**{**  pi[i] = INFINITY;  **}**  pi[s] = 0;  p[s] = -1;  **for**(it = 1; it < G->n; it++)**{**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pi[v] = pi[u] + w;  p[v] = u;  **}**  **}**  **}**  **for**(k=0; k< G->m; k++)**{**  **int** u = G->edges[k].u;  **int** v = G->edges[k].v;  **int** w = G->edges[k].w;  if(pi[u] + w < pi[v])**{**  pi[v] = pi[u] + w;  p[v] = u;  **}**  **}**  **}**  **int** main()**{**  // freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, e, w;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G,u,v,w);  **}**  BellmanFord(&G, 1);  if (pi[n] > 0)  pr**int**f ("NO");  else  pr**int**f ("YES");    return 0;  // pr**int**f("%d", pi[4]);  /\* **for**(i=1; i<=G.n;i++)  pr**int**f("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);\*/  **}** |

* **Bài 7 – Floyd – Warshall**

|  |
| --- |
| 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 (-100 < w <= 100).  Áp dụng giải thuật Floyd - Warshall viết chương trình tìm đường đi ngắn nhất giữa các cặp đỉnh. In chiều dài ngắn nhất giữa các cặp đỉnh ra màn hình theo dạng:  x -> y: chiều dài  ...  **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.   Dữ liệu được đảm bảo không tồn tại chu trình âm.  **Đầu ra (Output):**  In ra màn hình chiều dài đường đi ngắn nhất giữa các cặp đỉnh. Nếu không có đường đi in ra oo (vô cùng).  Liệt kê các cặp theo thứ tự tăng dần của x, và y. |
| #include<stdio.h>  #define MAXN 500  #define NO\_EDGE 951  **typedef** struct**{**  **int** n;  **int** L[MAXN][MAXN];  **}**Graph;  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->L[i][j] = NO\_EDGE;  **}**  void add\_edge(Graph\* G, **int** u, **int** v, **int** w)**{**  G->L[u][v]=w;  **}**  // giai thuat  #define INFINITY 9000  **int** pi[MAXN][MAXN];  **int** next[MAXN][MAXN];  void Floyd\_Warshall(Graph\* G)**{**  **int** u, v, k;  **for**(u=1; u<=G->n; u++)  **for**(v=1; v<=G->n; v++)**{**  pi[u][v] = INFINITY;  next[u][v] = -1;  **}**    **for**(u=1; u<=G->n; u++)  pi[u][u] = 0;    **for**(u=1; u<=G->n; u++)  **for**(v=1; v<=G->n; v++)  if(G->L[u][v] != NO\_EDGE)**{**  pi[u][v] = G->L[u][v];  next[u][v] = v;  **}**    **for**(k=1; k<=G->n; k++)  **for**(u=1; u<=G->n; u++)  **for**(v=1; v<=G->n; v++)  if(pi[u][k] + pi[k][v] < pi[u][v])**{**  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  **}**  **}**  **int** main()**{**  Graph G;  **int** n, m, u, v, w, e;  // freopen("floy1.txt", "r", stdin);  scanf("%d%d",&n,&m);  init\_graph(&G,n);  **for**(e=1; e<=m; e++)**{**  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G,u,v,w);  G.L[u][v]=w;  **}**  Floyd\_Warshall(&G);  **for**(u=1; u<=G.n; u++)  **for**(v=1; v<=G.n; v++)  if(pi[u][v]>=INFINITY-5)**{**  pr**int**f("%d -> %d: oo\n",u,v);  **}**else  pr**int**f("%d -> %d: %d \n",u,v,pi[u][v]);  return 0;  **}** |

* **Bài 8 – Tìm số đường đi ngắn nhất (nâng cao)**

|  |
| --- |
| Cho đồ thị đơn, 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 đếm xem có bao nhiêu 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 và số đường đi có chiều dài bằng với chiều dài dường đi ngắn nhất. Nếu không có đường đi từ 1 đến n, in ra -1 0.  Trong ví dụ đầu tiên có 2 đường đi từ 1 đến 3 là: 1 - 2 - 3 và 1 - 3. Cả hai đường đi này đều ngắn nhất và có chiều dài bằng 6.  Xem thêm ví dụ bên dưới.  **Gợi ý:**  Ngoài pi[v], ta định nghĩa thêm cnt[v]: số đường đi ngắn nhất từ s đến v.  Mỗi lần cập nhật pi[v] ta chú ý cập nhật lại cnt[v]. Nếu đường đi mới qua u tốt hơn thì cnt[v] = cnt[u]. Nếu bằng nhau thì cộng thêm: cnt[v] += cnt[u]. |
| #include<stdio.h>  #define MAX\_N 100  #define NO\_EDGE -1  #define Infinity 99999  typedef struct**{**  **int** size;  **int** data[MAX\_N];  **}**List;  void makenull\_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** p)**{**  return L->data[p - 1];  **}**  typedef struct**{**  **int** n;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = NO\_EDGE;  **}**  void add\_edge(Graph \*G, **int** x, **int** y, **int** w)**{**  G->A[x][y] = w;  **}**  **int** pi[MAX\_N];  **int** p[MAX\_N];  **int** mark[MAX\_N];  **int** soduongdi[MAX\_N];  void Moore\_Dijkstra(Graph \*G, **int** s)**{**  **int** i, j, k;  **for**(i = 1; i <= G->n; i++)**{**  pi[i] = Infinity;  **}**  pi[s] = 0;  p[s] = -1;    **for**(k = 1; k <= G->n; k++)**{**  **int** min\_pi = Infinity;  **for**(j = 1; j <= G->n; j++)**{**  if(mark[j] == 0 && pi[j] < min\_pi)**{**  min\_pi = pi[j];  i = j;  **}**  **}**  mark[i] = 1;  **for**(j = 1; j <= G->n; j++)**{**  if(mark[j] == 0 && G->A[i][j] != NO\_EDGE)**{**  if(pi[i] + G->A[i][j] < pi[j])**{**  pi[j] = pi[i] + G->A[i][j];  p[j] = i;  soduongdi[j] = soduongdi[i];  **}**else if(pi[i] + G->A[i][j] == pi[j])  soduongdi[j]+= soduongdi[i];    **}**  **}**  **}**  **}**  **int** main()**{**  Graph G;  //freopen("t.txt", "r", stdin);  **int** u, v, n, m, i, w;  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);  **}**    **for**(i = 1; i <= G.n; i++)**{**  mark[i] = 0;  soduongdi[i] = 1;  **}**        Moore\_Dijkstra(&G, 1);      if(pi[G.n] == Infinity)**{**  soduongdi[G.n] = 0;  printf("-1 0");  // printf("-1 %d", soduongdi[G.n]);  **}**    else  printf("%d %d", pi[G.n], soduongdi[G.n]);      return 0;  **}** |

* **BUỔI 4:**
* **Bài 1 - Xếp hạng đồ thị**

|  |
| --- |
| 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  Hạng đỉnh 2  ...  Hạng đỉnh n  Xem thêm ví dụ bên dưới. Trong ví dụ đầu tiên ta có: hạng của 1 = 0, hạng của 2 = 2 và hạng của 3 = 1. |
| #include<stdio.h>  #define MAX\_N 50  **typedef** struct**{**  **int** size;  **int** data[MAX\_N];  **}**List;  void makenull\_list(List \*L)**{**  L->size = 0;  **}**  void push\_back(List \*L, **int** x)**{**  L->data[L->size] = x;  L->size++;  **}**  **int** element\_at(List \*L, **int** p)**{**  return L->data[p - 1];  **}**  **int** sizelist(List \*L)**{**  return L->size;  **}**  **int** empty\_list(List \*L)**{**  return L->size == 0;  **}**  **typedef** struct**{**  **int** front, rear;  **int** data[MAX\_N];  **}**Queue;  void makenull\_queue(Queue \*Q)**{**  Q->rear = -1;  Q->front = 0;  **}**  **int** empty\_queue(Queue \*Q)**{**  return Q->front > Q->rear;  **}**  **int** top\_queue(Queue \*Q)**{**  return Q->data[Q->front];  **}**  void pop\_queue(Queue \*Q)**{**  Q->front++;  **}**  void push\_queue(Queue \*Q, **int** x)**{**  Q->rear++;  Q->data[Q->rear] = x;  **}**  **typedef** struct**{**  **int** top\_idx;  **int** data[MAX\_N];  **}**Stack;  void makenull\_stack(Stack \*S)**{**  S->top\_idx = 0;  **}**  **int** empty\_stack(Stack \*S)**{**  return S->top\_idx == 0;  **}**  void push\_stack(Stack \*S, **int** x)**{**  S->data[S->top\_idx] = x;  S->top\_idx++;  **}**  **int** top\_stack(Stack \*S)**{**  return S->data[S->top\_idx - 1];  **}**  void pop\_stack(Stack \*S)**{**  S->top\_idx--;  **}**  void copy\_list(List\* S1, List\* S2)**{**  makenull\_list(S1);  **int** i;  **for**(i=1; i<=S2->size; i++)**{**  **int** a = element\_at(S2, i);  push\_back(S1, a);  **}**  **}**  **typedef** struct**{**  **int** n;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n)**{**  G->n = n;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->n; j++)  G->A[i][j] = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y)**{**  G->A[x][y]++;  //G->A[y][x]++;  **}**  **int** degree(Graph \*G, **int** x)**{**  **int** i, deg = 0;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  deg++;  return deg;  **}**  List neighbors(Graph \*G, **int** x)**{**  List L;  makenull\_list(&L);  **int** i;  **for**(i = 1; i <= G->n; i++)  if(G->A[x][i] > 0)  push\_back(&L, i);    return L;  **}**  **int** rank[MAX\_N];  void ranking(Graph\* G)**{**  **int** d[MAX\_N];  **int** x, u;  **for**(u=1; u<=G->n; u++)  d[u]=0;  **for**(x=1; x<=G->n; x++)  **for**(u=1; u<=G->n; u++)  if(G->A[x][u] !=0)  d[u]++;  List S1, S2;  makenull\_list(&S1);    **for**(u=1; u<=G->n; u++)  if(d[u]==0)  push\_back(&S1, u);  **int** k=0, i;  while (S1.size >0)**{**  makenull\_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); //S1=S2  k++;  **}**  **}**  **int** main()**{**  Graph G;  **int** n, m, u, v, e, i;  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);  **}**  //in ma tran ke  // **for**(i=1; i<=n; i++)**{**  // **for**(j=1; j<=n; j++)  // pr**int**f("%d ", G.A[i][j]);  // pr**int**f("\n");  // **}**    // pr**int**f("\n");  ranking(&G);  **for**(i=1; i<=n;i++)  pr**int**f("%d\n", rank[i]);  return 0;  **}** |

* **Bài 2 – Cân đá**

|  |
| --- |
| Peter rất thích chơi đá. Anh ta thường dùng đá để trang trí sân nhà của mình. Hiện tại Peter có n hòn đá. Dĩ nhiên mỗi hòn đá có một khối lượng nào đó. Peter muốn đặt các hòn đá này dọc theo lối đi từ cổng vào nhà của mình. Peter lại muốn sắp xếp như thế này: hòn đá nặng nhất sẽ đặt ở cạnh cổng rào, kế tiếp là hòn đá nặng thứ 2, ... hòn đá nhẹ nhất sẽ được đặt cạnh nhà. Như vậy nếu đi từ trong nhà ra cổng, ta sẽ gặp các hòn đá có khối lượng tăng dần.  Tuy nhiên, điều khó khăn đối với Peter là anh chỉ có một cây cân đĩa mà không có quả cân nào. Nói cách khác, mỗi lần cân Peter chỉ có thể biết được hòn đá nào nhẹ hơn hòn đá nào chứ không biết nó nặng bao nhiêu kg.  Sau m lần cân, Peter biết được sự khác nhau về cân nặng của m cặp. Với các thông tin này, hãy giúp Peter sắp xếp các viên đá theo thứ tự anh mong muốn.  **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>  #define MAX\_VERTICES 100  #define MAX\_ELEMENTS 100  **typedef** **int** ElementType;  **int** rank[MAX\_VERTICES];  //khai bao cau truc list  **typedef** struct  **{**  ElementType data[MAX\_ELEMENTS];  **int** size;  **}** List;  **typedef** struct  **{**  **int** data[MAX\_ELEMENTS];  **int** front, rear;  **}** Queue;  //khoi bao cau truc di 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;  **}**  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;  **}**  /\* 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[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, x;  make\_null\_list(S1);  **for**(i=1; i<=S2->size; i++)  **{**  x=element\_at(S2,i);  push\_back(S1,x);  **}**  **}**  List topo\_sort(Graph \*G)  **{**  **int** d[MAX\_VERTICES];  **int** x, u;  // **int** k = 1;  **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]++;  Queue Q;  make\_null\_queue(&Q);  **for**(u = 1; u <= G->n; u++)  if(d[u] == 0)  push(&Q, u);  List L;  while(!empty(&Q))  **{**  **int** u = top(&Q);  pop(&Q);  push\_back(&L, u);  **int** v;  **for** (v = 1; v <= G->n; v++)  if(G->A[u][v] != 0)  **{**  d[v]--;  if(d[v] == 0)  push(&Q, v);  **}**  **}**  return L;  **}**  **int** main ()  **{**  //freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  **for** (e =1 ; e <= m; e++)  **{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**  List L = topo\_sort(&G);  **for**(u=1; u<=L.size; u++)  pr**int**f("%d ",element\_at(&L,u));  // return 0;  **}** |

* **Bài 3 – Chia kẹo**

|  |
| --- |
| Cô giáo Trang chuẩn bị kẹo để phát cho các bé mà cô đang giữ. Dĩ nhiên môi bé đều có một tên gọi rất dễ thương ví dụ: Mạnh Phát, Diễm Quỳnh, Đăng Khoa, ... Tuy nhiên, để đơn giản vấn đề ta có thể giả sử các em được đánh số từ 1 đến n.  Cô giáo muốn rằng tất cả các em đều phải có kẹo. Cô lại biết thêm rằng có một số bé có ý muốn hơn bạn mình một chút vì thế các em ấy muốn kẹo của mình nhiều hơn của bạn.  Hãy viết chương trình giúp cô tính xem mỗi em cần được chia ít nhất bao nhiêu kẹo và tổng số kẹo ít nhất mà cô phải chuẩn bị là bao nhiêu.  **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 di 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);  **}**  /\* d[n+1] = 0;  **for** (u = 1; u <= n; u++) **{**  scanf("%d", &d[u]);  do**{**  scanf("%d",&v);  if(x>0) add\_edge(&G, v, u);  **}** while (x>0);  **}**\*/  ranking(&G);  **for**(a=1; a<=n; a++)  **{**  pr**int**f("%d \n",rank[a]);  sum =sum + rank[a];  **}**  pr**int**f("%d",sum);  // pr**int**f("\n");\*/  /\* **for**(u = 1; u <= n; u++)**{**  **for**(v=1;v<=n; v++)  pr**int**f("%d ",G.A[u][v]);  pr**int**f("\n");  **}**\*/  return 0;  **}** |

* **Bài 4 – Tổ chức thi công – Dự án xây nhà**

|  |
| --- |
| Có một dự án xây nhà với 10 công việc được cho như bảng sau:    **Người ta cần:** - Xác định thời điểm sớm nhất và trể nhất để bắt đầu cho mỗi công việc mà không ảnh hưởng đến tiến độ của dự án Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 ...thay vì A, B, C....như sau:     và lưu vào tập tin theo định dạng như bảng giá trị **đầu vào**  Hãy viết chương trình tìm thời gian sớm nhất hoàn thành dự án và Thời điểm sớm nhất và trể nhất để bắt đầy cho mỗi công việc của dự án mà không ảnh hưởng đến tiến độ của dự án.  **Đầu vào:**  Dòng đầu tiên là số công việc (10), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 7 và danh sách các công việc trước đó rỗng.  Công việc 2 (công việc B) có d[2] = 3 và danh sách công việc trước đó là **{**1**}**.  **Đầu ra:**  Dòng đầu tiên: Thời gian sớm nhất hoàn thành dự án  Mỗi dòng tiếp theo: In ra thời gian sớm nhất và thời trể nhất để bắt đầu cho mỗi công việc (1 => n+2, gồm cả công việc alpha và beta)  **t(u)-T(u)** |
| #include<stdio.h>  #define MAX 100  #define oo 9999999  **typedef** struct  **{**  **int** n,m;  **int** A[MAX][MAX];  **}** Graph;  void init\_graph(Graph \*G, **int** n)  **{**  G->n=n;  G->m=0;  **}**  void add\_edge(Graph \*G, **int** u,**int** v)  **{**  G->A[u][v] += 1;  **}**  **int** adjacent(Graph \*G, **int** x, **int** y)  **{**  return G->A[x][y] != 0 ;  **}**  ///  **typedef** **int** ElementType;  **typedef** struct  **{**  ElementType data[MAX];  **int** size;  **}** List;  void make\_null\_list(List \*L)  **{**  L->size = 0;  **}**  void push\_back(List \*L, **int** x)  **{**  L->data[L->size]=x;  L->size++;  **}**  ElementType element\_at(List \*L, **int** i)  **{**  return L->data[i-1];  **}**  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 \*L1,List \*L2)  **{**  make\_null\_list(L1);  **int** i;  **for**(i=1; i<=L2->size; i++)  **{**  **int** u = element\_at(L2,i);  push\_back(L1,u);  **}**  **}**  ///  **typedef** struct  **{**  **int** front,rear;  **int** data[MAX];  **}** 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];  **}**  **int** pop(Queue \*Q)  **{**  return Q->front++;  **}**  **int** empty(Queue \*Q)  **{**  return Q->front > Q->rear;  **}**  // Ham min  **int** min(**int** a,**int** b)  **{**  if(a<b) return a ;  else return b;  **}**  // Ham max  **int** max(**int** a , **int** b)  **{**  if(a>b) return a ;  else return b;  **}**  ///  **int** d[MAX];  void topo\_sort(Graph \*G,List \*L)  **{**  **int** d[100];  **int** x, u;  Queue Q;  // khoi tao hang doi rong  make\_null\_queue(&Q);  **for** (u = 1; u <= G->n; u++)  d[u] = 0;  **for** (x = 1; x <= G->n; x++)  **for** (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0) // co duong di tu x->u  d[u]++;  **for**(u=1; u<=G->n; u++)  if(d[u]==0) push(&Q,u);  make\_null\_list(L);  while(!empty(&Q))  **{**  **int** u = top(&Q);  pop(&Q);  push\_back(L,u);  List list = neighbors(G,u);  **for**(x=1; x<=list.size; x++)  **{**  **int** v = element\_at(&list,x);  d[v]--;  if(d[v]==0)  push(&Q,v);  **}**  **}**  **}**  **int** main()  **{**  Graph G;  // freopen("dt.txt", "r", stdin);  **int** n, u, x, v, j ;  List L;  scanf("%d", &n);  init\_graph(&G, n+2); // them 2 dinh alpha & beta  d[n+1] = 0; // d[alpha] = 0  **for** (u = 1; u <= n; u++)  **{**  scanf("%d",&d[u]);  do  **{**  scanf("%d", &x);  if (x > 0) add\_edge(&G, x, u);  **}**  while (x > 0);  **}**  //--Kiem tra du lieu doc tu file, xoa sau khi kiem tre ket qua dung  // **for** (i = 1; i <= n; i++) pr**int**f("%d ",d[i]);  // pr**int**f("\n");  // **for** (i = 1; i <= n; i++) **{**  // **for** (j = 1; j <= n; j++)  // pr**int**f("%d ", G.A[i][j]);  // pr**int**f("\n");  // **}**  //------------------------------------------------------------------  //--Noi dinh n+1 voi dinh khong co cung di den no  **for** (u = 1; u <= n; u++)  **{**  **int** deg\_neg = 0;  **for** (x = 1; x <= n; x++)  if (G.A[x][u] > 0) // co duong di tu x->u  deg\_neg++;  if (deg\_neg == 0)  add\_edge(&G, n+1, u);  **}**  //--Noi dinh khong co cung di ra den dinh n+2  **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, n+2);  **}**  //--Kiem tra du lieu doc tu file, xoa sau khi kiem tre ket qua dung  // pr**int**f("\n");  // **for** (i = 1; i <= n+2; i++) **{**  // **for** (j = 1; j <= n+2; j++)  // pr**int**f("%d ", G.A[i][j]);  // pr**int**f("\n");  // **}**  //------------------------------------------------------------------  topo\_sort(&G,&L);  **int** t[100];  //Kiem tra L, sau do xoa  // pr**int**f("\n");  // **for** (i = 1; i <= L.size; i++)  // pr**int**f("%d ",element\_at(&L, i));  // pr**int**f("\n");  //Xac dinh thoi diem som nhat  t[n+1] = 0; // t[alpha] = 0  **for** (j = 2; j <= L.size; j++)  **{**  **int** u = element\_at(&L, j);  t[u] = 0;  **for** (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  **}**  //Xac dinh thoi diem tre nhat  **int** T[100];  T[n+2] = t[n+2];  **for** (j = L.size - 1; j >= 1; j --)  **{**  **int** 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]);  **}**  pr**int**f("%d\n",t[n+2]);  **for**(u=1; u<=G.n; u++)  pr**int**f("%d-%d\n",t[u],T[u]);  **}** |

* **Bài 5 - Tổ chức thi công – Dự án phần mềm**

|  |  |
| --- | --- |
| Việc thực hiện một dự án phát triển phần mềm được bố trí thành các công việc và thời gian thực hiện như sau:    - Anh Tuấn là một thành viên trong nhóm phát triển phần mềm. Anh ta thường hay hỏi mọi thành viên trong nhóm các câu hỏi tương tự như sau: "Nếu công việc E mình bắt đầu làm vào ngày thứ 60 thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?" "Nếu công việc H mình bắt đầu làm vào ngày thứ 50 thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?". Anh ta hỏi mọi người hoài những câu hỏi tương tự như thế làm cho các thành viên trong nhóm bực bội. Biết rằng dựa vào bảng công việc người ta có thể xác định thời điểm sớm nhất và trể nhất để bắt đầu cho mỗi công việc mà không ảnh hưởng đến tiến độ của dự án phần mềm. Hãy viết chương trình để giúp anh Tuấn tự trả lời câu hỏi của mình.  Để đơn giản trong cài đặt, ta đánh số lại các công việc theo thứ tự 1, 2, 3 thay vì A, B, C và lưu vào tập tin theo định dạng như sau:    **Đầu vào:**  Dòng đầu tiên là số công việc (12), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 14 và danh sách các công việc trước đó rỗng. Công việc 2 (công việc B) có d[2] = 12 và danh sách công việc trước đó là **{**1**}**. Dòng cuối cùng: công việc u và thời gian bắt đầu t, hai giá trị u và t tương ứng với câu hỏi của anh Tuấn:  "Nếu công việc u mình bắt đầu làm vào ngày thứ t thì tổng thời gian thực hiện dự án có bị ảnh hưởng không?"  **Đầu ra:**  **Yes**: Nếu ngày bắt đầu thực hiên công việc nằm trong thời điểm sớm nhất và trể nhất để bắt đầu công việc tương ứng.  **No**: Nếu ngày bắt đầu thực hiện công việc KHÔNG nằm trong thời điểm sớm nhất và trể nhất để bắt đầu công việc tương ứng. Ví dụ: Công việc 5, mình có thể bắt đầu làm vào ngày thứ 60 được hay không? => YES (Vì Thời gian sớm nhất và thời gian trể nhất thực hiện công việc 5 là: 56-71, 60 nằm trong khoảng thời gian cho phép) | **Đầu vào:**  Dòng đầu tiên là số công việc (12), các dòng tiếp theo mỗi dòng mô tả một công việc bao gồm d[u]: thời gian hoàn thành công việc u và danh sách các công việc trước đó của u. Danh sách được kết thúc bằng số 0. Ví dụ: công việc 1 (công việc A) có d[1] = 14 và danh sách các công việc trước đó rỗng. Công việc 2 (công việc B) có d[2] = 12 và danh sách công việc trước đó là **{**1**}**.  **Đầu ra:**  Danh sách các công việc then chốt. Mỗi công việc trên một dòng. |
| #include <stdio.h>  #include <stdio.h>  #define MAX\_VERTICES 100  **typedef** struct  **{**  **int** n; /\* n: so dinh \*/  /\* ma tran dinh – dinh \*/  **int** A[MAX\_VERTICES][MAX\_VERTICES];  **}** Graph;  /\* Khoi tao do thi G co n dinh \*/  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 e = (x, y) vao do thi G \*/  void add\_edge(Graph\* G, **int** x, **int** y)  **{**  G->A[x][y] = 1; //y ke voi x  **}**  /\* Kiem tra y co ke voi x khong \*/  **int** adjacent(Graph\* G, **int** x, **int** y)  **{**  return G->A[x][y] != 0;  **}**  /\* Tinh bac cua dinh x: deg(x) \*/  **int** degree(Graph\* G, **int** x)  **{**  **int** y, deg = 0;  **for** (y = 1; y <= G->n; y++)  if (G->A[x][y] > 0)  deg+=G->A[x][y];  return deg;  **}**  /\* KHAI BAO VA DINH NGHIA CTDL DANH SACH \*/  #define MAX\_ELEMENTS 100  **typedef** **int** ElementType;  **typedef** struct  **{**  ElementType data[MAX\_ELEMENTS];  **int** size;  **}** List;  /\* Tao danh sach rong \*/  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;  **}**  /\* Tim cac dinh ke cua dinh x \*/  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 \*L1, List \*L2)  **{**  make\_null\_list(L1);  **int** i;  **for**(i=1; i<=L2->size; i++)  push\_back(L1,element\_at(L2,i));  **}**  #define oo 9999999  **int** min(**int** a, **int** b)  **{**  if(a>=b) return b;  else return a;  **}**  **int** max(**int** a, **int** b)  **{**  if (a>b) return a;  else return b;  **}**  /\* Khai bao Queue \*/  #define MAX\_ELEMENTS 100  **typedef** struct  **{**  **int** data[MAX\_ELEMENTS];  **int** front, rear;  **}** Queue;  void make\_null\_queue(Queue\* Q)  **{**  Q->front = 0;  Q->rear = -1;  **}**  void push(Queue\* Q, **int** x)  **{**  Q->rear++;  Q->data[Q->rear] = x;  **}**  **int** top(Queue\* Q)  **{**  return Q->data[Q->front];  **}**  void pop(Queue\* Q)  **{**  Q->front++;  **}**  **int** empty\_queue(Queue\* Q)  **{**  return Q->front > Q->rear;  **}**  **int** d[MAX\_VERTICES];  void topo\_sort(Graph \*G,List \*L)  **{**  //Tinh d[u]  **int** d[MAX\_VERTICES];  **int** x, u;  **for** (u = 1; u <= G->n; u++)  d[u] = 0;  **for** (x = 1; x <= G->n; x++)  **for** (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0)  d[u]++;  Queue Q;  make\_null\_queue(&Q);  **for**(u=1; u<=G->n; u++)  if(d[u]==0)  push(&Q,u);  make\_null\_list(L);  while(!empty\_queue(&Q))  **{**  **int** u = top(&Q);  pop(&Q);  push\_back(L,u);  List list = neighbors(G,u);  **for**(x=1; x<=list.size; x++)  **{**  **int** v = element\_at(&list,x);  d[v]--;  if(d[v]==0)  push(&Q,v);  **}**  **}**  **}**  **int** main()  **{**  Graph G;  // freopen("dt.txt", "r", stdin);  **int** n, u, x, v, j ;  List L;  scanf("%d", &n);  init\_graph(&G, n+2);  d[n+1] = 0;  **for** (u = 1; u <= n; u++)  **{**  scanf("%d",&d[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++;  if (deg\_neg == 0)  add\_edge(&G, n+1, 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, n+2);  **}**  topo\_sort(&G,&L);  **int** t[MAX\_VERTICES];  t[n+1] = 0;  **for** (j = 2; j <= L.size; j++)  **{**  **int** u = element\_at(&L, j);  t[u] = -1;  **for** (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  **}**  **int** T[MAX\_VERTICES];  T[n+2] = t[n+2];  **for** (j = L.size-1 ; j >= 1; j --)  **{**  **int** 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]);  **}**  //pr**int**f("%d\n",t[G.n]);  **int** a,b;  scanf("%d %d",&a,&b);  /\* **for**(u =1; u<=G.n;u++)**{**  if (t[u]==T[u])  pr**int**f("%d\n",u);  pr**int**f("%d-%d\n",t[u],T[u]);\*/  // **}**  if(b>= t[a] && b<= T[a])  pr**int**f("YES");  else pr**int**f("NO");  return 0;  **}** | #include<stdio.h>  #define MAX 100  #define oo 99999  **typedef** **int** ElementType;  **typedef** struct  **{**  ElementType data[MAX];  **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];  **}**  ///  **typedef** struct  **{**  **int** n,m;  **int** A[MAX][MAX];  **}** Graph;  void init\_graph(Graph \*G, **int** n)  **{**  G->n=n;  G->m=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;  **}**  ///  **typedef** struct  **{**  **int** front,rear;  **int** data[MAX];  **}** 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];  **}**  **int** pop(Queue \*Q)  **{**  return Q->front++;  **}**  **int** empty(Queue \*Q)  **{**  return Q->front > Q->rear;  **}**  // Ham min  **int** min(**int** a,**int** b)  **{**  if(a<b) return a;  else return b;  **}**  **int** max(**int** a , **int** b)  **{**  if(a>b) return a;  else return b;  **}**  **int** d[MAX]; // thoi gian hoan thanh cong viec  void topo\_sort(Graph \*G, List \*L)  **{**  **int** d[MAX];  **int** x,u;  Queue Q;  // khoi tao hang doi rong  make\_null\_queue(&Q);  // khoi tao bac vao cua u = 0  **for**(u=1; u<=G->n; u++)  d[u] = 0;  // duyet qua all dinh , neu co duong di tu x->u thi bac vao u++  **for**(x=1; x<=G->n; x++)  **for**(u=1; u<=G->n; u++)  if(G->A[x][u] > 0)  d[u]++; // tang bac vao cua u  // dinh nao co bac vao = 0 thi push vao Queue  **for**(u=1; u<=G->n; u++)  if(d[u] ==0)  push(&Q,u);  // khoi tao danh sach de luu  make\_null\_list(L);  while(!empty(&Q))  **{**  **int** u = top(&Q);  pop(&Q); // lay u tu trong Queue ra sau do push\_back vao List  push\_back(L,u);  // xet cac dinh ke v cua u  List list = neighbors(G,u);  **for**(x=1; x<=list.size; x++)  **{**  **int** v = element\_at(&list,x);  d[v]--;  if(d[v]==0)  push(&Q,v);  **}**  **}**  **}**  **int** main()  **{**  Graph G;  //freopen("dt.txt", "r", stdin);  **int** n,u,x,v,j;  List L;  scanf("%d",&n);  init\_graph(&G,n+2);  d[n+1] = 0; // thoi gian hoan thanh cong viec cua n+1 = 0  **for**(u=1; u<=n; u++)  **{**  scanf("%d",&d[u]); // thoi gian hoan thanh cong viec u  do  **{**  scanf("%d",&x);  if(x>0) add\_edge(&G,x,u);  **}**  while(x>0);  **}**  // Them cung noi alpha voi cac dinh co bac vao = 0  **for**(u=1; u<=n; u++)  **{**  **int** deg\_neg = 0;  **for**(x=1; x<=n; x++)  if(G.A[x][u] > 0) // neu co duong di tu x->u thi bac vao cua u++  deg\_neg++;  if(deg\_neg == 0)  add\_edge(&G,n+1,u); // alpha->u  **}**  // Them cung noi cac dinh co bac ra = 0 voi beta  **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,n+2);  **}**  topo\_sort(&G,&L);  // Xac dinh thoi diem som nhat  **int** t[100];  t[n+1] = 0; // t[alpha] = 0  **for**(j=2; j<=L.size; j++)  **{**  **int** u = element\_at(&L,j);  t[u] = 0;  **for**(x=1; x<=G.n; x++)  if(G.A[x][u] > 0) // co duong di tu x->u  t[u] = max(t[u],t[x] + d[x]);  **}**  // Xac dinh thoi diem tre nhat  **int** T[100];  T[n+2] = t[n+2]; // T[beta] = t[beta]  **for**(j=L.size-1; j>=1; j--)  **{**  **int** 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]);  **}**  //pr**int**f("%d\n",t[n+2]);  // pr**int**f("\n");  //**for**(u=1;u<=G.n;u++)  // pr**int**f("%d-%d\n",t[u],T[u]);  **for**(u=1; u<=G.n; u++)  if(t[u] == T[u])  **{**  pr**int**f("%d\n",u);  **}**  return 0;  **}** |

* **BUỔI 5:**
* **Bài 1 – Tìm cây khung bằng giải thuật Kruskal**

|  |
| --- |
| Viết chương trình tìm cây khung có trọng số nhỏ nhất bằng giải thuật Kruskal.  **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 trọng số của cây khung tìm được và danh sách các cung theo thứ tự tăng dần của trọng số.  Quy ước: các cung được in ra theo định dạng:  **u v w**  với (u < v), mỗi cung trên 1 dòng. Nếu hai cung có trọng số bằng nhau thì cung nào có u nhỏ hơn sẽ được in trước. Nếu có trọng số bằng nhau và u giống nhau thì cung nào có v nhỏ hơn sẽ in trước. |
| #include <stdio.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 100  **int** parent[50];  //**int** root\_u, root\_v, sum\_w;  **typedef** struct  **{**  **int** u, v, w;  **}** Edge;  **typedef** struct  **{**  Edge edge[MAX\_EDGES];  **int** n, m;  **}** Graph;  void init\_graph (Graph \*G, **int** n)  **{**  G->n = n;  G->m = 0;  **}**  void add\_edge(Graph \*G, **int** x, **int** y, **int** w)  **{**  G->edge[G->m].u = x;  G->edge[G->m].v = y;  G->edge[G->m].w = w;  G->m++;  **}**  /\*  **int** findRoot(**int** u)**{**  if(parent[u] == u)  return u;  return findRoot(parent[u]);  **}**  **int** kruskal(Graph \*G, Graph \*T)**{**  **int** u, v, w, e;  **int** root\_u, root\_v, sum\_w;  // sap xep cac cung theo thu tu tang dan trong so  **int** tempu, tempv, tempw;  **int** i, j;  **for** (i = 1; i <= G->m - 2; i++)  **for**( j = i+1; j <= G->m-1; j++)  if(G->edge[i].w > G->edge[j].w)**{**  tempu = G->edge[i].u;  tempv = G->edge[i].v;  tempw = G->edge[i].w;  G->edge[j].u = G->edge[i].v;  G->edge[j].v = G->edge[i].v;  G->edge[j].w = G->edge[i].w;  G->edge[j].w = tempu;  G->edge[j].v = tempv;  G->edge[j].w = tempw;  **}**  //khoi tao dao thi T  init\_graph(T, G->n);  **for**(u = 1; u <= G->n; u++)  parent[u] = u;  **for**(e = 1; e <= G->m; e++)**{**  **int** u = G->edge[e].u;  **int** v = G->edge[e].v;  **int** w = G->edge[e].w;  **int** root\_u = findRoot(u);  **int** root\_v = findRoot(v);  if(root\_u != root\_v)**{**  if(u<v) add\_edge(T, u, v, w);  else add\_edge(T, v, u, w);  //gop 2 BPLT root\_u va root\_v lai  parent[root\_v] = root\_u;  sum\_w += w;  **}**  **}**  return sum\_w;  **}**\*/  void Swap(Edge \*a, Edge \*b)  **{**  Edge temp;  temp = \*a;  \*a = \*b;  \*b = temp;  **}**  void Sort(Graph \*G)  **{**  **int** i, j;  **for**(i = 0; i < (G->m) -1; i++)  **for**(j = i + 1; j < G->m; j++)  if(G->edge[i].w > G->edge[j].w)  Swap(&G->edge[i], &G->edge[j]);  **}**  **int** findRoot(**int** u)  **{**  if(parent[u] == u)  return u;  return findRoot(parent[u]);  **}**  **int** kruskal(Graph \*G, Graph \*T)  **{**  Sort(G);  /\* **int** tempu, tempv, tempw;  **int** i, j;  **for** (i = 1; i <= G->m - 2; i++)  **for**( j = i+1; j <= G->m-1; j++)  if(G->edge[i].w > G->edge[j].w)**{**  tempu = G->edge[i].u;  tempv = G->edge[i].v;  tempw = G->edge[i].w;  G->edge[j].u = G->edge[i].v;  G->edge[j].v = G->edge[i].v;  G->edge[j].w = G->edge[i].w;  G->edge[j].w = tempu;  G->edge[j].v = tempv;  G->edge[j].w = tempw;  **}**\*/  init\_graph(T, G->n);  **int** u, e;  **for**(u = 0; u < G->n; u++)  parent[u] = u;  **int** sum\_w = 0;  **for** (e = 0; e < G->m; e++)  **{**  **int** u = G->edge[e].u;  **int** v = G->edge[e].v;  **int** w = G->edge[e].w;  **int** root\_u = findRoot(u);  **int** root\_v = findRoot(v);  if(root\_u != root\_v)  **{**  add\_edge(T, u, v, w);  parent[root\_v] = root\_u;  sum\_w += w ;  **}**  **}**  return sum\_w;  **}**  **int** main()  **{**  Graph G, T;  // freopen("dt.txt", "r", stdin);  **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);  if(sum\_w > 0)  **{**  pr**int**f("%d\n", sum\_w);  **for** (e = 0; e < T.m; e++)  if(T.edge[e].u > T.edge[e].v)  pr**int**f("%d %d %d\n", T.edge[e].v, T.edge[e].u, T.edge[e].w);  else  pr**int**f("%d %d %d\n", T.edge[e].u, T.edge[e].v, T.edge[e].w);  **}**  **}** |

* **Bài 2 – Tìm cây khung có trọng lượng nhỏ nhất bằng giải thuật Prim**

|  |
| --- |
| Viết chương trình tìm cây khung có trọng số nhỏ nhất bằng giải thuật 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:  - Dòng đầu tiên in trọng số của cây khung tìm được  - n - 1 dòng kế tiếp in ra các cung của cây tìm tìm được theo định dạng: u v w. Cung (u1, v1) sẽ được in ra trước cung (u2, v2) nếu (u1 < u2) hoặc (u1 = u2 và v1 < v2). |
| #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)  **{**  // **int** yes = 0;  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;  // freopen("dt.txt","r",stdin);  **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);  pr**int**f("%d", sum\_w);  bubble\_sort(edges, dem);  **for**(i=0; i<dem; i++)  pr**int**f("\n%d %d %d", edges[i].u, edges[i].v, edges[i].w);  return 0;  **}** |

* **Bài 3 – Tìm luồng cực đại trong mạng**

|  |
| --- |
| Viết chương trình tìm luồng cực đại trên mạng bằng thuật toán Ford - Fullkerson (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:  - Dòng đầu tiên in luồng cực đại theo dạng: **Max flow: s**  - Dòng thứ hai in các đỉnh của X0 theo dạng: **X0: x1 x2 ...**, mỗi đỉnh cách nhau 1 khoảng trắng  - Dòng thứ ba in các đỉnh của Y0 theo dạng: **Y0: y1 y2 ...**, mỗi đỉnh cách nhau 1 khoảng trắng |
| #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;  //freopen("dt.txt", "r", stdin); //Khi n?p bài nh? b? dòng này.  **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);  pr**int**f ("Max flow: %d \n",max\_flow);  pr**int**f("X0: ");  **for** ( e = 1; e <= G.n; e++)  **{**  if (labels[e].dir != 0)  pr**int**f("%d ",e);  **}**  pr**int**f("\nY0: ");  **for** (e = 1; e <= G.n; e++)  **{**  if(labels[e].dir == 0)  pr**int**f("%d ",e);  **}**  return 0;  **}** |

* **BÀI THI THỬ**
* **Bài 3: Người quen**

|  |
| --- |
| Lớp học phần LTDT có n sinh viên (giả sử được đánh số từ 1 đến n). Hai sinh viên trong lớp có thể quen biết nhau, nếu a quen biết b thì b cũng quen biết a và ngược lại; không tính tới việc một sinh viên quen biết với chính mình. Một sinh viên trong lớp có thể quen biết với một hoặc nhiều sinh viên; hoặc không quen biết sinh viên nào trong lớp.  Hãy **viết chương trình** in ra màn hình **số lượng người quen của một sinh viên x nào đó**  **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 lượng sinh viên (n) và số lượng mối quan hệ quen biết (m).  - m dòng tiếp theo mỗi dòng chứa 2 số nguyên u v nói rằng: sinh viên u quen biết với sinh viên v.  - Dòng cuối cùng chứa số nguyên x (sinh viên x cần tìm số lượng người quen trong lớp).  **Đầu ra (Output):**  - In ra màn hình số lượng người quen của x |
| #include<stdio.h>  #define MAX\_N 20  **typedef** struct**{**  **int** n;  **int** m;  **int** A[MAX\_N][MAX\_N];  **}**Graph;  void init\_graph(Graph \*G, **int** n, **int** m)**{**  G->n = n;  G->m = m;  **int** i, j;  **for**(i = 1; i <= G->n; i++)  **for**(j = 1; j <= G->m; 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; //vo huong  **}**  //ke  **int** adj(Graph \*G, **int** x, **int** y)**{**  if(G->A[x][y]==1)  return 1; //xac dinh ke  return 0;  **}**  //Lang gieng  void neighbors(Graph \*G, **int** x)**{**  **int** i;  **for**(i =1; i<=G->n; i++)  if(adj(G, x, i) == 1)  //in gia tri lang gieng ra  pr**int**f("%d ", i);  **}**  //tinh bac  **int** degree(Graph \*G, **int** x)**{**  **int** i;  **int** deg = 0;  **for** (i=1 ;i<=G->n; i++)  if(adj(G, x, i) == 1)  deg++;  return deg;  **}**    void dthu(Graph \*G, **int** x, **int** y)**{**  **int** i, j = 0;  **for**(i = 1; i <= G->n; i++)**{**  **int** a = G->A[x][i];  **int** b = G->A[i][x];  **int** c = G->A[y][i];  **int** d = G->A[i][y];  if((a == 1 && c == 1) || (a == 1 && d == 1) || (b == 1 && c == 1) || (b == 1 && d == 1))**{**  j++;  **}**  **}**  pr**int**f("%d", j);  **}**    **int** main()**{**  Graph G;  **int** n, m, u, v, x, e, i;  **int** nei=0;  //freopen("dt.txt", "r", stdin);//Khi n?p bài nh? xóa dòng này  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for**(e=1; e<=m; e++)**{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**  scanf("%d", &x);  **for**(i =1; i<=G.n; i++)**{**  if(adj(&G, x, i) == 1)  //in gia tri lang gieng ra  nei++;  **}**    pr**int**f("%d:%d", x, nei);  return 0;  **}** |

* **Bài 4 – Tìm đường đi của John**

|  |
| --- |
| John là lập trình viên sống tại thành phố San Francisco. Hiện tại trong nhóm phát triển phần mềm của anh có ký kết hợp đồng phát triển phần mềm với hai công ty. Công ty A thì đặt hàng một phần mềm mô phỏng cho thiết kế kiến trúc nhà như hỗ trợ phối màu và bố trí nội thất trang trí nhà. Công ty B thì đặt hàng phần mềm quản lý nhân sự trong công ty như: hỗ trợ tính lương và điểm danh nhân sự trong công ty của họ. Những người đại diện của công ty A và công ty B trực tiếp làm việc với John thì không nắm rõ về các yêu cầu phát triển hệ thống nên họ thường xuyên thay đổi yêu cầu của mình. Nên hàng tháng anh John phải đến công ty gặp trực tiếp những người đại diện của công ty A và công ty B để triển khai hệ thống đang phát triển và thu thập thêm về yêu cầu phần mềm để tiếp tục phát triển hệ thống.  Biết rằng anh John sống tại thành phố San Francisco có thể đi đến công ty A và công ty B thông qua hệ thống các chuyến bay được cho trong bảng sau đây.    Ví dụ: Giá vé máy bay từ San Francisco đến Detroit và từ Detroit đến San Francisco đều có giá vé $329.  Để đơn giản trong cài đặt, ta đánh số lại các thành phố như sau:    Thông thường thì chuyến công tác của anh John đến gặp khách hàng của công ty A và công B diễn ra 3 ngày:  Ngày 1: Anh John đi từ nhà (tại thành phố San Francisco) đến công ty A (Denver)  Ngày 2: Anh John đi từ công ty A (Denver) đến công ty B (New York)  Ngày 3: Từ công ty B (New York) về nhà (San Francisco)  Hãy giúp John chọn những chuyến bay như thế nào để anh ta đi từ thành phố này đến thành phố kia trong mỗi ngày công tác của mình với chi phí vé máy bay mà anh ta phải trả là thấp nhất.  **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 thành phố và số chuyến bay giữa hai thành phố.  - m dòng tiếp theo mỗi dòng chứa 3 số nguyên u v w nói rằng: chuyến bay từ thành phố u đến thành phố v với giá vé là w  - Dòng cuối cùng chứ 2 số nguyên x y là hai thành phố mà anh John muốn tìm đường đi của một ngày công tác với số tiền vé máy bay là thấp nhất.  **Đầu ra (Output):**  In ra màn hình:  - Dòng đầu tiên: Tổng số tiền vé mà anh John phải chi trả cho chuyến đi công tác từ thành phố x đến thành phố y.  - Mỗi dòng tiếp theo là lịch trình mà anh John đi từ thành phố x sang thành phố y. |
| #include <stdio.h>  #define MAX 100  #define NO\_EDGE 9999999  #define INFINITY 999999  **int** mark[MAX];  **int** pi[MAX];  **int** p[MAX];  **int** path[MAX];  **typedef** struct**{**  **int** n;  **int** A[MAX][MAX];  **}**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]=NO\_EDGE;  **}**  void add\_edge(Graph\* G,**int** x,**int** y,**int** w)**{**  G->A[x][y]=w;  G->A[y][x]=w;  **}**  **int** degree(Graph\* G,**int** x)**{**  **int** y,deg=0;  **for**(y=1;y<=G->n;y++)  if(G->A[x][y]>0)  deg+=G->A[x][y];  return deg;  **}**  **int** adjacent(Graph\* G,**int** x,**int** y)**{**  if(G->A[x][y]!=0)  return 1;  else  return 0;  **}**  void Dijkstra(Graph \*G,**int** s)  **{**  **int** i,j,it;  **for**(i=1;i<=G->n;i++)  **{**  pi[i]=INFINITY;  mark[i]=0;  **}**  pi[s]=0;  p[s]=-1;  **for**(it=1;it<G->n;it++)  **{**  **int** min\_pi =INFINITY;  **for**(j=1;j<=G->n;j++)  **{**  if(mark[j]==0&&pi[j]<min\_pi)  **{**  min\_pi= pi[j];  i=j;  **}**  **}**  mark[i]=1;    **for**(j=1;j<=G->n;j++)  **{**  if(G->A[i][j] != NO\_EDGE && mark[j]==0)  **{**  if(pi[i]+G->A[i][j]<pi[j])  **{**  pi[j]=pi[i]+G->A[i][j];  p[j]=i;  **}**  **}**  **}**  **}**    **}**  **int** main()**{**  freopen("dt.txt", "r", stdin);  Graph G;  **int** n, m, u, v, 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);  Dijkstra(&G, s);    printf("%d\n", pi[t]);  // scanf("%d",&t);  //**for**(t=n; t<=G.n; t++)  //**int** path[MAX];  **int** k=0,i;  while (t != -1)**{**  path[k]=t; k++;  t = p[t];  **}**  **for**(i=k-1; i>=0; i--)  printf("%d ", path[i]);  return 0;  **}** |

* **Bài 5 – Quan hệ tiên quyết**

|  |
| --- |
| Chương trình đào tạo (CTĐT) của một ngành là một danh sách các học phần mà sinh viên cần phải hoàn thành để được tốt nghiệp. Trong một CTĐT, quan hệ tiên quyết giữa các học phần là quan trọng nhất, đảm bảo tính hợp lý về mặt thứ tự của các khối kiến thức phải tích luỹ. Học phần tiên quyết là học phần phải được học trước học phần khác. Mỗi học phần có thể có nhiều học phần tiên quyết hoặc cũng có thể không có học phần tiên quyết nào.  Để đơn giản vấn đề ta giả sử CTĐT gồm n học phần được đánh số 1, 2, ..., n. Một CTĐT được xem là hợp lệ nếu không tồn tại một dãy các học phần u1, u2, ..., uk (k >= 1) sao cho u1 là tiên quyết của u2, u2 là tiên quyết của u3, ..., uk-1là tiên quyết của uk và uk là tiên quyết của u1.  Bài toán đặt ra là làm thế nào để kiểm tra một CTĐT đào tạo bất kỳ có hợp hệ hay không. Hãy viết chương trình đọc vào danh sách các cặp học phần có quan hệ tiên quyết và kiểm tra tính hợp lệ của CTĐT.  **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 học phần và số cặp học phần có quan hệ tiên quyết.  - **m** dòng tiếp theo mỗi dòng chứa 2 số nguyên **x**, **y** nói rằng học phần **x** là học phần tiên quyết của học phần **y**.  **Đầu ra (Output):**  In ra màn hình**HOP LE** nếu CTĐT hợp lệ, ngược lại in ra **KHONG** |
| #include <stdio.h>  #define INFINITY 9999  #define MAXN 500  #define NO\_EDGE 0  **typedef** struct **{**  **int** n; // Dinh  **int** m;  **int** L[MAXN][MAXN];  **}** Graph;  void init\_graph(Graph\* G, **int** n, **int** m) **{**  G->n = n;  G->m = m;  **int** i, j;  **for** (i = 1; i <= n; i++) **{**  **for** (j = 1; j <= m; j++) **{**  G->L[i][j] = NO\_EDGE;  **}**  **}**  **}**  // LIST  #define MAX\_ELEMENTS 100  **typedef** **int** ElementType;  **typedef** struct **{**  ElementType data[MAX\_ELEMENTS];  **int** size;  **}** List;  void make\_null(List\* L) **{**  L->size = 0;  **}**  void push\_back(List\* L, ElementType x) **{**  L->data[L->size] = x;  L->size++;  **}**  ElementType element\_at(List\* L, **int** i) **{**  return L->data[i-1];  **}**  **int** count\_list(List\* L) **{**  return L->size;  **}**  **int** adjeacent(Graph\* G, **int** x, **int** y) **{**  **int** e;  **for** (e = 1; e <= G->m; e++)  if (G->L[x][e] == 1 && G->L[y][e] == 1)  return 1;  return 0;  **}**  **int** degree(Graph\* G, **int** x) **{**  **int** e, deg = 0;  **for** (e = 1; e <= G->m; e++)  if (G->L[x][e] == 1)  deg++;  return deg;  **}**  void print\_list(List L) **{**  **int** i;  **for** (i = 1; i <= L.size; i++) **{**  printf("%d ", element\_at(&L, i));  **}**  **}**  List neighbors(Graph\* G, **int** x) **{**  List ansList;  make\_null(&ansList);  **int** col, temp;  **for**(col = 1; col <= G->n; col++) **{**  if (G->L[x][col] && col != x) **{**  push\_back(&ansList, col);  **}**  **}**  **int** i, j;  **for**(i = 1; i <= count\_list(&ansList); i++) **{**  **for**(j = i+1; j <= count\_list(&ansList)-1; j++) **{**  if(element\_at(&ansList, i) > element\_at(&ansList, j)) **{**  temp = ansList.data[i-1];  ansList.data[i-1] = ansList.data[j-1];  ansList.data[j-1] = temp;  **}**  **}**  **}**  List checkList;  make\_null(&checkList);  **int** check = 1;  **for**(i = 1; i <= count\_list(&ansList); i++) **{**  **for**(j = 1; j <= count\_list(&checkList); j++) **{**  if(element\_at(&ansList, i) == element\_at(&checkList, j)) **{**  check = 0;  **}**  **}**  if(check) **{**  push\_back(&checkList, element\_at(&ansList, i));  **}**  **}**    return checkList;  **}**  void add\_edge(Graph\* G, **int** x, **int** y) **{**  G->L[x][y] = 1;  **}**  #define white 0  #define black 1  #define gray 2  #define MAX\_VERTICES 1000  **int** color[MAX\_VERTICES];  **int** cycle;  // Kiem tra chu trinh doi voi do thi co huong  // Duyet do thi bat dau tu dinh x  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; // Ton tai chu trinh  return;  **}**  if (color[y] == white) **{**  dfs(G, y);  **}**  **}**  color[x] = black;  **}**  // Kiem tra toan bo do thi  **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** e, u, v, m, n;  freopen("thuyentruong.txt","r",stdin);  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  **for** (e = 1; e <= m; e++) **{**  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  **}**    if (contains\_cycle(&G))  printf("KHONG");  else  printf("HOP LE");    return 0;  **}** |