|  |  |
| --- | --- |
| Buổi 4 | **Ứng dụng thuật toán**  **Breadth First Search – BFS** |

**Bài 1** : Đỉnh khớp

Đỉnh 𝑥 là đỉnh khớp nếu xóa các cạnh kề của đỉnh 𝑥 thì số miền liên thông của đồ thị tăng lên ít nhất là 2. Hãy kiểm tra xem đỉnh 𝑥 có phải là đỉnh khớp của đồ thị 𝐺 không.

|  |  |  |
| --- | --- | --- |
| Đồ thị | Input : CutBridge.txt | Output : xuất lên màn hình |
|  | 8  1 4  0 4  3  2 6  0 1  3 7  6 | Nhập đỉnh cần xét x = ?  (trả lời)  X là đỉnh khớp hay không? |

*Hướng dẫn* :

* Trong class AdjList.cs viết phương thức bỏ các cạnh kề với x :

public void RemoveEdgeX(int x)

{

Duyệt từng đỉnh i của đồ thị

Nếu i = x : xóa hết các phần tử trong dslk v[i]

Ngược lại : xóa phần tử x trong v[i]

}

* Trong Program, sử dụng phương thức trên :

// Tạo đồ thị ban đầu g

string filePath = "../../../TextFile/CutBridge.txt";

AdjList g = new AdjList();

g.FileToAdjList(filePath); g.Output();

// Gọi phương thức liên thông : g.Connected();

// Gọi int gInconnect1 = số TPLT của g

// Nhập đỉnh x

// Gọi phương thức bỏ các cạnh kề x : g.RemoveEdgeX(x);

// Gọi phương thức liên thông của g (sau khi đã bỏ các cạnh kề của x)

Gọi int gInconnect2 = số TPLT của g (sau khi đã bỏ các cạnh kề của x)

// Nếu gInconnect2 > gInconnect1 + 1 : đỉnh x là đỉnh khớp,

ngược lại : x không phải là đỉnh khớp

**Bài 2** : cạnh cầu

Cạnh cầu của đồ thị là cạnh nếu xóa đi thì số miền liên thông của đồ thị tăng lên. Hãy kiểm tra xem cạnh (𝑥, 𝑦) có phải là cầu của đồ thị 𝐺 không

|  |  |  |
| --- | --- | --- |
| Đồ thị | Input : CutBridge.txt | Output : xuất lên màn hình |
|  | 8  1 4  0 4  3  2 6  0 1  3 7  6 | Nhập cạnh (x, y) = ?  (trả lời)  Cạnh (x,y) là cạnh cầu hay không? |

*Hướng dẫn* :

* Trong class AdjList.cs viết phương thức bỏ cạnh (x,y)

public void RemoveEdgeXY(int x, int y)

{

// Xóa y trong v[x]

// Xóa x trong v[y]

}

* Trong Program.cs, tương tự như bài 1 : Tính số TPLT của g trước khi bỏ cạnh (x,y) 🡪 bỏ cạnh (x,y) trong g 🡪 tính số TPLT của g 🡪 so sánh 2 số TPLT trước, sau khi bỏ (x,y) 🡪 kết luận?

**Bài 3 :** Đồ thị phân đôi (đồ thị hai phía – Bipartitle graph)

|  |  |  |
| --- | --- | --- |
| Đồ thị | Input : Bipartite.txt | Output : xuất lên màn hình |
|  | 6  1 3 5  0 2 4  1 3 5  0 2 4  1 3 5  0 2 4 | Đồ thị là đồ thị phân đôi? |

Một [đồ thị đơn](https://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93_th%E1%BB%8B_%C4%91%C6%A1n&action=edit&redlink=1" \o "Đồ thị đơn (trang không tồn tại)) [vô hướng](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_v%C3%B4_h%C6%B0%E1%BB%9Bng" \o "Đồ thị vô hướng) �:=(�,�)G(,V,E) được gọi là **hai phía** mà tập đỉnh của nó có thể chia thành hai tập con �X và �Y rời nhau sao cho bất kì cạnh nào của đồ thị cũng nối một đỉnh của �X với một đỉnh thuộc �Y. Khi đó người ta còn ký hiệu là: �:=(�⋃�,�)G = (X ∪ Y, E) và gọi một tập (chẳng hạn �X) là **tập các đỉnh trái** và tập còn lại (chẳng hạn �Y) là **tập các đỉnh phải** của đồ thị hai phía �

Nếu |�1|=|�2||V1| = |V2| thì �G được gọi là đồ thị hai phía **cân bằng**.

Hướng dẫn :

Trong program.cs, viết

case 6:

{ // Kiểm tra đồ thị 2 phía

string filePath = "../../../TextFile/Bipartite.txt";

AdjList g = new AdjList();

g.FileToAdjList(filePath); g.Output();

if (g.IsBipartite(0))

Console.WriteLine(" Đồ thị 2 phía");

else

Console.WriteLine(" Không phải đồ thị 2 phía");

break;

}

Trong class AdjList.cs, viết phương thức:

// Đồ thị 2 phía dùng BFS. Gọi hàm : IsBipartite(0)

public bool IsBipartite(int x)

{

// Khởi tạo và gán giá trị color[i] = -1, i = 0..<n

// Trước khi xét, đặt color[x] = 1

// Khởi tạo Queue<int> q = new Queue<int>();

// bỏ x vào q

// Lặp khi q khác rỗng

{

//Lấy s <- q;

//Nếu s thuộc v[s] -> return false;

// Duyệt (u in v[s])

{

Nếu(color[u] == -1)

{

// gán : color[u] = 1 - color[s];

// Bỏ u -> q;

}

Ngược lại Nếu(color[u] == color[s])

return false;

}

}

return true;

}

**Bài 4** : Bài toán đi trên lưới

Cho lưới hình chữ nhật có kích thước 𝑛 × 𝑚 gồm các số 0, 1. Các dòng được đánh số từ 1 đến 𝑛 từ trên xuống dưới, các cột được đánh số từ 1 đến 𝑚 từ trái sang phải. Ô trong bảng có giá trị 0 gọi là ô nước, ô có giá trị 1 gọi là đất. Từ một ô tại vị trí (𝑖,𝑗) là đất có thể đi đến 1 trong 4 ô xung quanh nếu các ô đó cùng là ô đất. Cho hai ô 𝑠(𝑖1,𝑗1) và 𝑡(𝑖2,𝑗2) thuộc ô có giá trị 1. Hãy một tìm đường đi ngắn nhất từ 𝑠(𝑖1,𝑗1) đến 𝑡(𝑖2,𝑗2)

|  |  |  |
| --- | --- | --- |
| Input : Grid.txt | Mô tả | Output |
| 6 8  2 1 4 6  0 1 1 1 1 0 0 0  0 1 0 0 1 0 0 0  0 1 1 0 1 0 1 0  0 0 1 1 1 1 1 0  0 0 0 0 0 0 1 0  0 0 0 0 0 0 0 0 | Số dòng, số cột ma trận  Tọa độ ô bắt đầu (3,2) và ô cuối (5,7)  Ma trận | 8 (đường đi có 8 cạnh)  2 1 (tọa độ các ô đi qua)  2 2  3 2  3 3  3 4  3 5  3 6  4 6 |

*Nhận xét* :

- Bài toán qui về tìm đường đi trên đồ thị, với mỗi ô là một đỉnh 🡪 số đỉnh của đồ thị = row(6)\*col(8)

- Đánh số đỉnh của đồ thị : đỉnh 0 (tương ứng ô đầu (0,0) , . . . , đỉnh 47 (tương ứng ô cuối (5,7)

- Cách qui đổi ô (x,y) của ma trận sang số thứ tự đỉnh trong đồ thị : **y + x\*col**

- Để xây dựng các cạnh kề của đỉnh k trong đồ thị như sau : ô (i,j) tương ứng đỉnh k = j + i\*col

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | k - col |  | | k -1 | k = j + i\*8 | k +1 | |  | k + col |  | | Mỗi một ô (= 1) có tối đa 4 ô kề được xác định như hình bên  Nếu ô kề = 1 🡪 có cạnh nối ô đó với k  Lưu ý các ô biên có số ô kề < 4 |

- Từ ma trận 🡪 xây dựng đồ thị danh sách kề và tìm đường đi từ ô (2,1) đến ô (4,6) tương ứng với từ đỉnh x = (1 + 2\*col) đến đỉnh y = (6 + 4\*col)

*Hướng dẫn* : từ nhận xét trên, trong class AdjList.cs viết 2 phương thức sau :

public void GridToAdjList(string filePath)

{

Xử lý đọc file Grid.txt 🡪 2 đỉnh x, y và ma trận a[]

Ma trận a[] 🡪 đồ thị DS kề

// Tìm và xuất đường đi theo tọa độ

PathOnGrid(x, y, col);

}

// Tìm và xuất đường đi từ đỉnh x đến y nhưng hiển thị trên màn hình theo tọa độ ô tương ứng. Tham số col phục vụ cho việc chuyển đổi đỉnh đồ thị 🡪 tọa độ trong ma trận a[]

public void PathOnGrid(int x, int y, int col)

{

// Bài toán tìm đường đi trên đồ thị DS kề tư đỉnh x 🡪 y

// số ô đã đi qua = số phần tử trong stack

// Qui đổi đỉnh k sang ô (x,y) theo công thức : (k / col, k % col)

// Nên sử dụng Tuple<int, int> để mô tả cho cặp tọa độ.

}

**Bài tập thêm** : tìm đường đi từ cửa vào đến cửa ra trong mê cung như sau :

Cửa vào

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Cửa ra

Trình bày kết quả lên màn hình :

|  |  |
| --- | --- |
| Xuất ma trận mê cung | Xuất đường đi |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0  0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0  0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 0 1 0  0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0  0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0  0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0  0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0  0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0  0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0  0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0  0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 0 1 0  0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0  0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0  0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0  0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0  0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0  0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0  0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0  0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0  0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |