|  |  |
| --- | --- |
| Buổi 5 | **Tìm kiếm trên đồ thị bằng thuật toán**  **Depth First Search – DFS** |

1. **Duyệt đồ thị bằng thuật toán DFS**

Thuật toán (sử dụng stack)

1. Lấy s là một đỉnh của đồ thị (đỉnh bắt đầu)
2. Bỏ s 🡪 Stack
3. Lặp khi Stack ≠ Ø
   1. Lấy p 🡨 Stack
   2. Duyệt p
   3. Đặt các đỉnh kề của p chưa từng có trong stack 🡪 Stack

Cài đặt thuật toán (sử dụng đệ qui) :

Sử dụng một array visited[] để đánh dấu đỉnh đã được xét hay chưa, với :

visited[i] = true , đã xét i và visited[i] = false , chưa xét i

khởi tạo visited[i] = false , mọi i = 0 … <n

DFS(s)

{

Duyệt đỉnh s ;

visited[s] = true ; //đánh dấu đã xét đỉnh v

for(u ∈ Kề(s))

if (chưa xét u) DFS(u) ;

}

1. **Áp dụng**

**Bài 1** : Tìm các đỉnh liên thông với đỉnh x theo thuật toán DFS

|  |  |  |
| --- | --- | --- |
| Đồ thị | AdjList.txt | Output |
|  | 8  1  0 4  3  2 6  1  3 7  7 | Nhập đỉnh xuất phát x : ? (giả sử nhập x = 2)  Các đỉnh liên thông với 2 : 2 - 3 - 6 - 7 - |

*Hướng dẫn* :

* Giải thuật DFS đệ qui thường phải có nơi gọi hàm 🡪 trong AdjList.cs viết 2 phương thức :

TraverseDFS(int s) : chuẩn bị môi trường ban đầu và gọi DFS(int s)

public void TraverseDFS(int s)

{

S là đỉnh xuất phát

Khởi tạo và gán giá trị ban đầu cho visited[]

Nếu (chưa đánh dấu đỉnh s)

DFS(s);

}

public void DFS(int s)

{

Đánh dấu đỉnh s;

Duyệt đỉnh s;

Xét từng đỉnh u kề với s

Nếu (u chưa được đánh dấu) DFS(u);

}

**Bài 2** : Tìm đường đi từ đỉnh x đến y.

|  |  |  |
| --- | --- | --- |
| Đồ thị | Input : AdjList.txt | Output : xuất lên màn hình |
|  | 8  1  0 4  3  2 6  1  3 7  7 | Nhập đỉnh xuất phát x : ? và đỉnh đến y = ?  (giả sử nhập x = 2 , y = 7)  Đường đi từ 2 đến 7 : 2 🡪 3 🡪 6 🡪 7 |

*Hướng dẫn* :

* Tương tự như giải thuật BFS, khai báo thêm trong class AdjList.cs các thành phần sau :

bool[] visited; // Dùng đánh dấu đỉnh đã đi qua

int[] pre; // Dùng lưu trữ đỉnh nằm trước trên đường đi

* Trong class AdjList.cs, viết 3 phương thức :

// Tạo môi trường ban đấu và gọi DFS

public void DFSRouteXtoY(int x, int y)

{

Nếu (x hoặc y là đỉnh cô lập)

Xuất : " Không có đường đi";

return;

Khởi tạo và gán giá trị ban đầu cho visited[]

Khởi tạo và gán giá trị ban đầu cho pre[] : pre[i] = -1,∀i

Chọn x là đỉnh xuất phát, gọi : DFS\_XtoY(x);

// Xuất đường đi

Nếu (pre[y] == -1)

Xuất : " Không có đường đi tới y";

Ngược lại : xuất đường đi, gọi :

RouteXY(x, y);

}

// Hàm Đệ qui DFS

public void DFS\_XtoY(int s)

{

Đánh dấu đỉnh s

Xét từng đỉnh u kề s (int u in v[s])

Nếu (u chưa được đánh dấu)

{

Cho s đứng trước u : pre[u] = s;

Gọi : DFS\_XtoY(u);

}

}

// Xuất đường đi : tương tự như BFS, sử dụng pre[]

public void RouteXY(int x, int y)

{

}

**Bài 3** : Xét tính liên thông của đồ thị, nếu đồ thị không liên thông thì cho biết đồ thị có bao nhiêu thành phần liên thông (TPLT) và xuất các thành phần liên thông lên màn hình

|  |  |  |
| --- | --- | --- |
| Đồ thị | Input : AdjList2.txt | Output : xuất lên màn hình |
|  | 8  1  0 4  3  2 5 6  1  3  3 7  7 | Đồ thị có 2 thành phần liên thông  TPLT 1 : 0 1 4  TPLT 2 : 2 3 5 6 7 |

*Hướng dẫn* :

* Trong class AdjList.cs, để đơn giản cho cài đặt 🡪 thêm 2 thành phần như sau :

**int[] index; // Dùng đánh dấu các TPLT**

**int inconnect; // Dùng đếm số TPLT, và thêm propeties**

**public int Inconnect { get => inconnect; set => inconnect = value; }**

* Biến inconnect để lưu số TPLT, ban đầu inconnect = 0 và sẽ tăng 1 khi gọi hàm duyệt DFS
* index[v] : lưu chỉ số thành phần liên thông chứa đỉnh v, mỗi khi gọi hàm duyệt DFS(v) thì chỉ số thành phần liên thông hiện tại đang chứa v 🡪 index[v] = inconnect
* Trong class AdjList.cs, viết 3 phương thức :

public void Connected()

{

// chuẩn bị môi trường ban đầu

// Duyệt từng đỉnh i của đồ thị

// Nếu chưa duyệt đỉnh i

{

// Khởi đầu cho một TPLT mới, xử lý biến inconnect?

// Tìm và đánh dấu các đỉnh cùng TPLT, gọi hàm

DFS\_Connected(i);

}

}

// Lượt duyệt mới vớt đỉnh bắt đầu: s

public void DFS\_Connected(int s)

{

// Là thuật toán DFS và thêm phần xử lý cho index[]

}

// Xuất các thành phần liên thông

public void OutConnected()

{

// Tương tự như BFS

}

**Bài 4** : Kiểm tra đỉnh x có phải là đỉnh khớp .

**Bài 5** : Kiểm tra cạnh (x,y) có phải là cạnh cầu.

**Ứng dụng giải thuật duyệt đồ thị DFS**

**Dự án đường sắt một chiều**

1. **Giới thiệu**

Có ***n*** điểm tập trung dân cư đông đúc. Các điểm này được đánh số từ 1 đến ***n*** (1 ≤ ***n*** ≤ 104). Mạng lưới giao thông công cộng là ***m*** đường xe lửa cao tốc một rail, ***mỗi đường nối một cặp điểm dân cư*** và ***chạy hai chiều*** (0 ≤ ***m*** ≤ 105), và ***mọi cặp điểm đều có thể đi đến được với nhau***. Để tránh sự va chạm giữa các con tàu cao tốc khi chúng có thể đi ngược chiều trên cùng một đường, chính quyền thành phố quyết định sửa lại các con đường đó thành một chiều. Tuy nhiên, sau khi thay đổi thì lại có một vấn đề bất cập sảy ra, đó là: tồn tại các cặp điểm tập trung dân cư không thể đi đến được nhau.

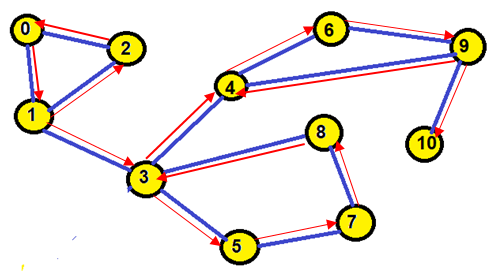
Chính vì vậy, chính quyền lại thêm một quyết định nữa, đó là sẽ xây dựng thêm một số ít nhất các tuyến đường mới để đảm bảo từ một điểm bất kỳ có thể đi tới điểm bất kỳ khác bằng tàu cao tốc.

Sơ đồ mạng lưới giao thông ban đầu :

|  |  |
| --- | --- |
| Sơ đồ | Danh sách đường (OneWay.txt) |
|  | 11 13  0 1  0 2  1 2  1 3  3 4  3 5  3 8  4 6  4 9  5 7  6 9  7 8  9 10 |

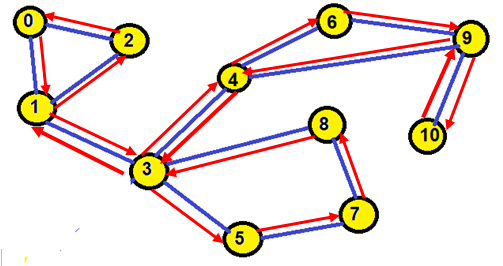
1. **Mô hình thực hiện bài toán**

* Bước 1 : Xem hệ thống đường xe lửa cao tốc trên là một đồ thị vô hướng 11 đỉnh và 13 cạnh và liên thông. Các cạnh đánh số theo chiều thuận (s, u) với s < u. Từ đây ta gọi cạnh theo chiều thuận, ngược lại là cạnh ngược. Ví dụ : (0, 2) cạnh thuận, (2, 0) là cạnh ngược
* Bước 2 : Đầu tiên từ đỉnh 0 duyệt đồ thị theo cách DFS 🡪 kết quả là một tập cạnh đường đi từ 0 đến các đỉnh như sau : (0,1) (1,2) (1,3) (3,4) (3,5) (4,6) (5, 7) (6,9) (7,8) (9,10)
* Bước 3 : Các cạnh trên được sắp một chiều theo chiều thuận, các cạnh còn lại trong tập cạnh của đồ thị được sắp theo chiều ngược : (2,0) (8,3) (9,4) 🡪 ta có kết quả như sau :



* Bước 4 :
  + Nếu đồ thị liên thông thì kết quả trên là kết quả cuối cùng
  + Nếu đồ thị không liên thông 🡪 nhận xét các cạnh đặc biệt sau :

Cạnh (1,3) , (3,4) ,(9,10) : là các cạnh cầu, đi qua nhưng không có đường về 🡪 các cạnh cầu phải có đường 2 chiều đến và đi 🡪 thêm cạnh ngược (3,1) , (4,3) , (10,9) 🡪 làm thêm 3 con đường này. Kết quả, ta có sơ đồ hệ thống đường xe lửa như sau :



Mô hình

OneWay.txt

Cây DFS

Bổ xung cạnh

Qui định chiều

Duyệt đồ thị

Theo thuật toán DFS

Đồ thị

(Danh sách kề)

1. **Cài đặt**

Bước 1 : Đọc file OneWay.txt và chuyển đồ thị danh sách cạnh 🡪 đồ thị danh sách kề (đã làm trong buổi thực hành 1). Trong Program.cs có hàm chuyển đồ thị danh sách cạnh sang đồ thị danh sách kề như sau :

static AdjList EdgeListToAdjList(string filePath)

{

AdjList g = new AdjList();

StreamReader sr = new StreamReader(filePath);

string[] s = sr.ReadLine().Split();

g.N = int.Parse(s[0]);

int m = int.Parse(s[1]);

g.V = new LinkedList<int>[g.N];

for (int i = 0; i < g.N; i++)

g.V[i] = new LinkedList<int>();

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

{

s = sr.ReadLine().Split();

g.V[int.Parse(s[0])].AddLast(int.Parse(s[1]));

g.V[int.Parse(s[1])].AddLast(int.Parse(s[0]));

Tuple<int, int> e = new Tuple<int, int>(int.Parse(s[0]), int.Parse(s[1]));

EdgeSet.Add(e);

}

sr.Close();

// xuất tập cạnh

Console.WriteLine();

foreach (Tuple<int, int> e in EdgeSet)

Console.WriteLine(" (" + e.Item1 + ", " + e.Item2 + ")");

return g;

}

Trong Program.cs có sử dụng biến toàn cục static List<Tuple<int, int>> EdgeSet; lưu trữ danh sách cạnh của đồ thị. Trong quá trình đọc file 🡪 xây dựng ds cạnh EdgeSet

Bước 2 : Từ đỉnh đầu tiên (đỉnh 0) 🡪 Duyệt đồ thị theo giải thuật DFS để xây dựng tập cạnh trong đường đi DFS. Ở bước 4, ta có xét đến cạnh cầu nên trong AdjList.cs viết phương thức duyệt DFS xây dựng đường đi và tính số thành phần liên thông (TPLT).

Khai báo thêm các thành phần :

bool[] visited; // Dùng đánh dấu đỉnh đã đi qua

int inconnect; // Dùng đếm số TPLT

int[] pre; // Dùng lưu trữ đường đi

List<Tuple<int, int>> DFSEdge; // Danh sách cạnh đi qua khi duyệt DFS

Nên viết riêng phương thức xuất tập cạnh DFSEge :

// Xuất cây DFSEdge

public void OutDFSEdge()

{

Console.WriteLine(" Các đường một chiều trong DFSEdge : ");

foreach (Tuple<int, int> e in DFSEdge)

Console.WriteLine(" (" + e.Item1 + ", " + e.Item2 + ")");

}

Bước 3 : Đưa các cạnh còn lại trong EdgeSet chuyển thành cạnh ngược đưa vào trong DFSEdge. Trong AdjList.cs viết phương thức :

// add các cạnh còn lại trong tập cạnh đồ thị và đảo chiều vào cây DFS

public void AddToDFSEdge(List<Tuple<int, int>> eSet)

{ // khởi tạo danh sách cạnh DFS trung gian : x

List<Tuple<int, int>> x = new List<Tuple<int, int>>();

x = DFSEdge;

// Duyệt từng cạnh trong x

// Nếu e chưa có trong x

{ // khởi tạo cạnh ed ngược với e

Tuple<int, int> ed = new Tuple<int, int>(e.Item2, e.Item1);

// Đưa ed vào DFSEdge

}

}

Bước 4 : tìm các cạnh cầu (x,y) trong đồ thị và thêm cạnh ngược (y,x) vào DFSEge

Trong AdjList.cs, bài 3 đã có các phương thức sau để sử dụng cho việc tìm cạnh cầu

public void Connected()

{

}

// Lượt duyệt mới vớt đỉnh bắt đầu: s

public void DFS\_Connected(int s)

{

}

Vấn đề tìm cạnh cầu đã làm trong bài 4.

Bước 5 : Xuất tập các đường (một chiều)