**BÀI TẬP THỰC HÀNH**

**NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**TUẦN 1 – DFS, BFS**

**(cài đặt BFS, bài tập về nhà DFS)**

Nội dung: Cài đặt thuật toán tìm kiếm Depth-First Search và Breadth-First Search trong không gian tìm kiếm hữu hạn.

Mô tả bài toán: cho không gian tìm kiếm hữu hạn các state, giả sử mỗi state là một con số integer, sử dụng DFS hoặc BFS để tìm đường đi đến goal state với initial state cho trước.

Input: initial state

Output: chuỗi các state để đến được goal

Hướng dẫn:

Bước 1: Cài đặt cấu trúc dữ liệu Node

1. Tạo class Node, cài đặt cấu trúc dữ liệu Node như sau:

**public** **class** Node {

**int** state; //

**boolean** visited;

List<Node> neighbours;

Node parent;

}

Mỗi một Node đại diện cho một state, biến visited để đánh dấu Node này đã từng được duyệt hay chưa, danh sách neighbours là danh sách kề của Node, tức là các Node con của Node đang xét, Node parent là Node cha của node đang xét.

1. Viết constructor cho Node:

Node(**int** state)

{

**this**.state=state;

**this**.neighbours=**new** ArrayList<>();

**this**.parent= **null;**

}

1. Viết hàm addNeighbours cho cấu trúc dữ liệu Node như sau:

**public** **void** addNeighbours(Node neighbourNode)

{

**this**.neighbours.add(neighbourNode);

}

Hàm này thực hiện nhiệm vụ thêm Node neighbour vào danh sách kề của Node đang xét. Chú ý, chúng ta không new một Node neighbour mới, mà chỉ add địa chỉ của neighbour **đang tồn tại** vào danh sách kề mà thôi.

1. Viết hàm getNeighbours cho cấu trúc dữ liệu Node như sau:

**public** List getNeighbours() {

**return** neighbours;}

Hàm này trả về danh sách toàn bộ các Node con của Node đang xét.

Bước 2: Cài đặt class BFS

Tạo class BFS, không có thuộc tính nào, chỉ có hàm bfsUsingQueue như sau:

**public** **class** BFS {

**public** **void** bfsUsingQueue(Node initial, int goal)

{

//Tự cài đặt theo slide

}

}

Gợi ý: trong hàm BFS chỉ khó ở 2 điểm sau:

* Khi thêm các Node vào stack, ta cần lưu vết lại Node cha của các Node được thêm vào. Đây là thời điểm chúng ta dùng đến thuộc tính parent của Node.
* Khi tìm được Node chứa goal đúng như nhu cầu, ta cần thực hiện backtrack để có được chuỗi đường đi hợp lý. Sử dụng đoạn code sau cho backtracking:

**if** (node.state==goal)

{

String s="";

**while** (node!=initial)

{

s=node.state+" "+s;

node=node.parent;

}

System.***out***.println(initial.state+ " "+s);

**return**;

}

(Bài tập về nhà: cài đặt DFS y như BFS, chỉ khác là dùng Stack thay cho Queue)

Bước 4: Tạo class Main, viết hàm main như sau:

**public** **class** Main {

**public** **static** **void** main(String arg[])

{

Node node40 =**new** Node(40);

Node node10 =**new** Node(10);

Node node20 =**new** Node(20);

Node node30 =**new** Node(30);

Node node60 =**new** Node(60);

Node node50 =**new** Node(50);

Node node70 =**new** Node(70);

node40.addNeighbours(node10);

node40.addNeighbours(node20);

node10.addNeighbours(node30);

node20.addNeighbours(node10);

node20.addNeighbours(node30);

node20.addNeighbours(node60);

node20.addNeighbours(node50);

node30.addNeighbours(node60);

node60.addNeighbours(node70);

node50.addNeighbours(node70);

BFS bfsExample = **new** BFS();

System.***out***.println("The BFS traversal of the graph using queue ");

bfsExample.bfsUsingQueue(node40,70);

}

}

Hàm main yêu cầu tìm đường đi từ Node 40 đến Node 70 của đồ thị sau, sử dụng DFS và BFS.

