COMP 380: Artificial Intelligence

Assignment One: Search

Overview: In this assignment you will complete an implementation of uninformed search, in Java. You are given partially completed code and must complete it. The completed code will allow for breadth-first and depth-first search.

You are permitted to work with a partner if you wish. Groups larger than two are not permitted. Each partner must submit the completed code.

1. Provided Code

The code consists of two classes: Path and Graph. You should look them over carefully. There are detailed comments in each.

The `Path` class:

- The Path class is very simple and is really just a wrapper for an ArrayList of Integers. That is, a Path is just a list of nodes where the nodes are indicated by integer ID's.
- The Path class contains two constructors: one for creating an initial Path containing just the start node, and a second constructor that creates a new Path by taking an existing Path and extending it by one node.
- **You must not change the Path class**.

The `Graph` class:

- The Graph class represents the search graph as an adjacency matrix, and provides a search method for searching the graph.
- An adjacency matrix *adjMat* is simply a two-dimensional array, where adjMat[i][j] is true if you can transition from node i to node j (there is an edge connecting them), and is false otherwise.
- The search method takes a start node, goal node, and search method, and attempts to find a path from the start to the goal, using the search method.
- The two possible search methods are depth-first and breadth-first.
- The frontier is simply an ArrayList of Paths.

2. To-Do

You must complete the implementation of the search() method in the Graph class. **That is the only part of the code you should change.**

The search() method must do the following:

- Display (using a print statement) every node that it inspects.
- If it finds the goal, display the path to a goal.
- Return true if the goal is found.
- Return false otherwise.
- Use either depth-first or breadth-first search, depending on the third parameter of the method.

Note: in the lectures we talked about depth-first treating the frontier like a stack, and breadth-first treating it like a queue. In this code, we use an ArrayList regardless of the search method. You can just select Paths from one end of the ArrayList or the other, depending on the search method.

3. Example

Here is the example contained in the provided main() method. We create a Graph with 7 nodes (numbered 0 through 6).

```
Graph g = new Graph(7);
```

We add edges to the Graph:

```
g.add(0, 1);
g.add(0, 2);
g.add(1,5);
g.add(1,6);
g.add(2, 3);
g.add(3, 4);
```

Select a search type:

```
boolean depthFirst = true;
```

Start searching. In this case, 0 is the start node and 4 is the goal node:

```
g.search(0,4, depthFirst);
```

Sample output:

```
Inspect node: 0
Inspect node: 2
Inspect node: 3
Inspect node: 4
```

```
Found goal node!
[0, 2, 3, 4]
```

Or do breadth-first instead:

```
boolean depthFirst = false;
g.search(0,4, depthFirst);
```

Sample output:

```
Inspect node: 0
Inspect node: 1
Inspect node: 2
Inspect node: 5
Inspect node: 6
Inspect node: 3
Inspect node: 4
Found goal node!
[0, 2, 3, 4]
```

Deliverable: Submit your source code Graph.java via Blackboard. Put your name and student number(s) at the top of the file. If you worked with a partner, put both of your names and student numbers there. Each partner must submit the completed code.