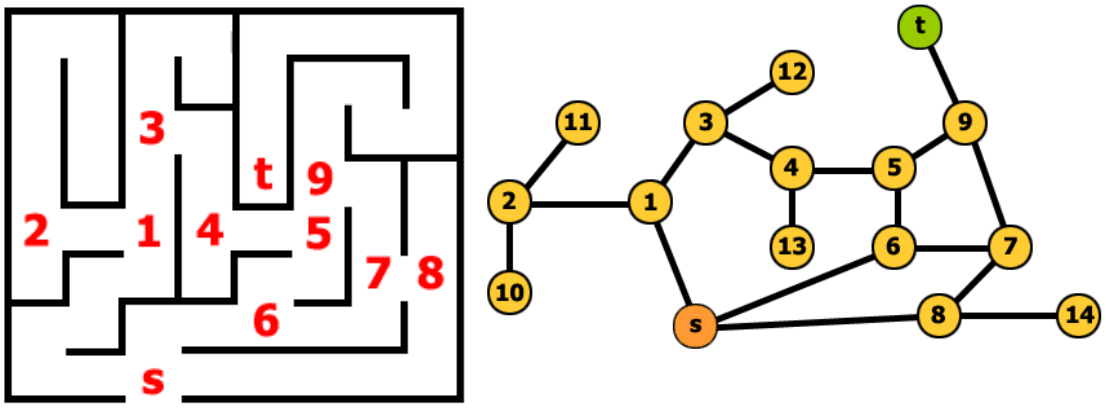
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

B.Sc. Software Development – Artificial Intelligence
Using a Semantic Network to Model a Maze

Overview

Knowledge representation is a key aspect to solving a problem using artificial intelligence. Before we can even begin to develop a solution, the problem must be first abstracted into a ***semantic network*** or tree. The objective of this lab is to create a semantic network representation of a maze. The maze and graph abstraction are shown below:



Implementation

A composite pattern is applied to implement a class *Node* that represents each of the vertices in a graph or a tree. Each *Node* class maintains an *ArrayList* of child nodes, to represent the branches in a tree or edges in a graph. The *Node* class exposes the following methods:

Method Name	Description
getNodeName()/setNodeName(...)	The name of the node or vertex.
Node[] children()	Returns an array of child nodes for a given node.
boolean isLeaf()	A leaf node has no children.
getChildNodeCount()	Returns the number of child nodes.
addChildNode(Node child)	Adds a child node to a given node.
removeChild(Node child)	Removes a child node from a given node.
boolean isVisited()/setVisited(boolean visited)	Marks whether a node has already been visited during a search. For BFS, this should be modified in order to mark nodes as white, grey or black.
isGoalNode()/setGoalNode(boolean goalNode)	Returns whether a given node is a goal node.

Building a Semantic Network of the Maze

Using the *Node* class, a semantic network representing the maze can easily be developed by adding new child nodes to a growing graph, beginning at the starting node "s".

1. **Download** the file ***aiMaze.zip*** from Moodle. The archive contains the classes *Node.java* and *Maze.java*. The latter is a singleton class that represents the semantic network of the maze. The class *Maze.java* is, however, unfinished!
2. **Complete the semantic network** by connecting the remaining node in the constructor of Maze.
3. **Implement the method** *Node getStartNode()* to return the starting node of the semantic network.

Finding an Exit from the Maze

Locating the exit from the maze amounts to searching the semantic network for the goal node "t". The two most common strategies for this are both uninformed or blind searches – depth-first search (DFS) and breadth-first search (BFS). These will be the subjects of our next lab. DFS can be implemented using either a stack or a recursive method call. The pseudocode for a recursive DFS is outlined below.

```
algorithm  dft(x)
   visit(x)
   FOR each y such that (x,y) is an edge DO
     IF y was not visited yet THEN
        dft(y)
```

1. ***Implement*** the above pseudocode to search the Maze for the exit.

Note that a DFS is the equivalent of navigating through the maze using a left-hand or right-hand rule.