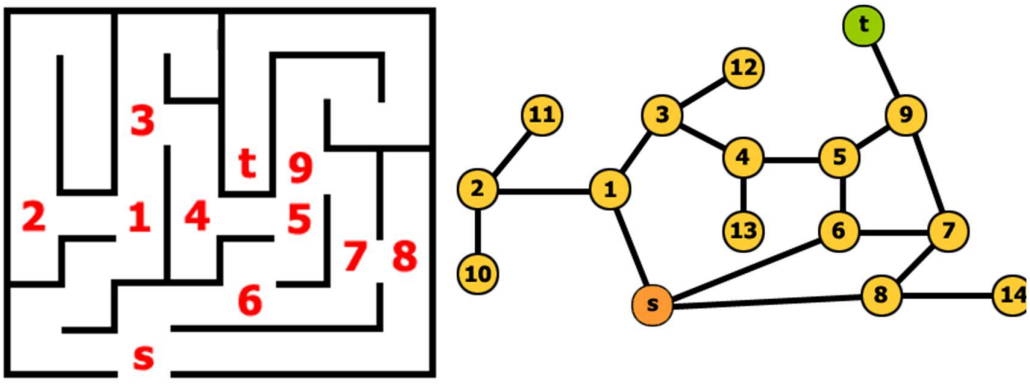
Department of Computer Science & Applied Physics

Using a Semantic Network to Model a Maze

Overview

Knowledge representation is a key aspect of 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<E>* that represents each of the vertices in a graph or a tree. Each *Node<E>* class maintains an *ArrayList* of child nodes to represent the branches in a tree or edges in a graph.

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 ***ai_UninformedSearchSource.zip*** from Moodle. The archive contains, among others, the classes *Node.java* and *Maze.java*. The latter is a singleton class that represents the semantic network of the maze.
2. **Check whether the semantic network** is complete, i.e., whether all the nodes are instantiated in the constructor of *Maze* and properly interconnected.
3. **Call the method** *Node<String> getSource()* to return the starting node of the semantic network if it is not already done in one of the source files.

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 if it is not already implemented in one of the provided source files.

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