Finding the shortest path

Introduction

For this last assignment we will revisit solving mazes. In the previous maze assignment you had to write maze solvers that walked through the maze in search of the exit. As long as they found the exit the length of the path didn't really matter that much. This time you will need to find the shortest path from the start node to the exit node. And to make it more difficult you will also have to deal with mazes that contain loops.

The assignment consists of two parts. First you will need to generate a tree with special properties from the maze, and secondly you will need to traverse this tree and mark the shortest path between start and exit in the maze.

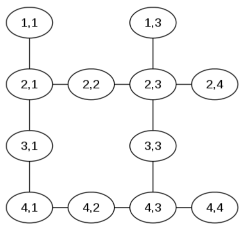
Maze as a graph

To illustrate the approach we will use a tiny maze as our example:

```
6,6
######
# # ##
#   S#
# # ##
# E  #
######
```

The maze contains dead ends and a loop. A wall following algorithm will not take the shortest path to the exit. If you keep your right hand to the wall you will take the long way around the loop examining two dead ends along the way. And if you keep your left hand to the wall you will take the dead end in it lower right corner before arriving at the exit. So we need a different approach.

It helps to think of the maze as a graph. The nodes are the locations and the edges connect locations that are directly adjacent to each other. If the maze contains loops the graph will also contain these loops. If the maze does not contain loops it called a perfect maze, and the graph will be a tree. The graph of our example maze is:

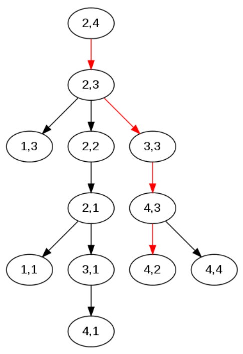


The graph representation of our example maze.

For your algorithm you will not actually have to generate a graph to find the shortest path. It is just shown here to clarify the process. You can create the minimum spanning tree used in the next section by directly traversing the ASCII map of the maze.

Minimum spanning tree

Now with this graph view in mind you can traverse the maze from the start node and create a minimum spanning tree. A minimum spanning tree is a sub graph that connects the nodes of the graph (in our case the locations in the maze) with a minimum number of edges. The spanning tree for our graph will look like this:



Minimum spanning tree with shortest path

Normally a spanning tree is created for a graph with weighted edges where you would minimise the total weight of the edges. Since our edges all have weight 1 the construction is simpler, but you need to carefully select the traversal method to traverse the maze (or its graph equivalent) so that the spanning tree that is generated consists of the shortest paths from the root node to any other node in the graph.

Finding the shortest path

Since the minimum spanning tree consists of the shortest paths from the start node to any other node, finding the shortest path from the start node to the exit node is just a matter of searching the tree from the root node until you find your exit node.

The path in the tree from the root (start node) to exit node is thus the shortest path in the maze. The shortest path is shown in red in the minimum spanning tree above.

The output of your assignment should be the shortest path displayed in the ASCII maze in this way:

```
######
# # ##
#  .S#
# #.##
# E. #
######
```

The implementation

For this assignment we provide the following files source files: `main.c`, `maze.h`, `maze.c` and `tree.h`. The files `main.c` and `tree.h` are incomplete, but the maze implementation can be used directly. These files are provided to save you time, but feel free to ignore them and use your own code. The requirements are that you use a minimum spanning tree with the described shortest path properties to find the shortest path and print the ASCII maze in the format shown above.

Report

For this assignment the report counts for 50% of the grade. Your report should contain the following sections:

- Introduction of the assignment (formal, no copy paste).
- Discussion of your implementation: what data structures did you use, which graph and tree traversal methods?
- Explanation of the design choices you made. For example: why did you choose these traversal methods?
- Testing: what mazes did you use, how did you perform the testing?

- Conclusions: What problems did you encounter, and how did you solve them.

Not counting your figures (probably mazes and trees) the report should be around 2 pages long.

Tips

- For this last assignment you may need to use some of the data structures developed in the previous assignments (stacks, queues, sets). Make sure these support data structures work by testing them before you use them for this assignment.
- The template tar file also contains some test mazes. Use these to test your code.
- You don't need to print the minimum spanning tree, but it can be very useful for debugging. The graph and the tree in this document are generated using the Graphviz software. To show how simple it is to visualise these, the "dot" code for the tree picture is shown here:

```
digraph BST {
node [fontname="Arial"]
   "2,4" -> "2,3" [color = red]
   "2,3" -> "1,3"
   "2,3" -> "2,2"
   "2,2" -> "2,1"
   "2,1" -> "1,1"
   "2,1" -> "3,1"
   "3,1" -> "4,1"
   "2,3" -> "3,3" [color = red]
   "3,3" -> "4,3" [color = red]
   "4,3" -> "4,2" [color = red]
   "4,3" -> "4,4"
}
```

This code is easily generated from your tree data structure with a simple traversal. The command to create a `png` file from this is: `dot -Tpng tree.dot > tree.png`

Bonus

The method described here will only find a shortest path. If the exit is moved one position to the left (at location 4,1) there will be two shortest paths in the maze. Can you extend your code to give all shortest paths in the maze?

References

- http://en.wikipedia.org/wiki/Tree_traversal
- http://en.wikipedia.org/wiki/Maze
- http://en.wikipedia.org/wiki/Maze_solving_algorithm
- http://www.graphviz.org/