BRSU

Homework for Artificial Intelligence for Robotics - Assignment 3

Bastian Lang

April 19, 2015

1 Theoretical Part

1.1 Exercise 1

Does a finite state space always lead to a finite search tree? Can you be more precise about what types of state spaces always lead to finite search trees? (Adapted from Bender, 1996.)

No, a finite search space does not always lead to a finite search tree. If there would be bidirectional connections between two states, then the resulting search tree would be infinite. Say for example state A could reach state B and vice versa. The search tree would have state B as a child of state A and state A as a child of state B. So there is an infinite loop.
If the finite state space does only have unidirectional connections between states and there is no path from any state to itself using however many states in between, then this will always result in a finite search tree.

2 Practical Part

2.1 Task Overview

In this exercise we have been given three text files representing maps. Each map contains characters which can be interpreted as follows:

- * → Dirt

- space → Empty Space
- s → Robot Starting Position
- Any other character → Obstacle

The task was to construct an algorithm that reads the map and uses Breadth-First Search (*BFS*) and Depth-First Search *DFS*) to plan a path for a robot to explore and find each dirt cell. The rules were given with:

- The robot can move from one cell to another at each step.
- The robot can only move to the left, right, up or down cells from the current position.
- The robot does not have previous knowledge about the environment, such as dirt positions or obstacles. It has to "explore".
- The robot cannot move through obstacles and the map is closed.

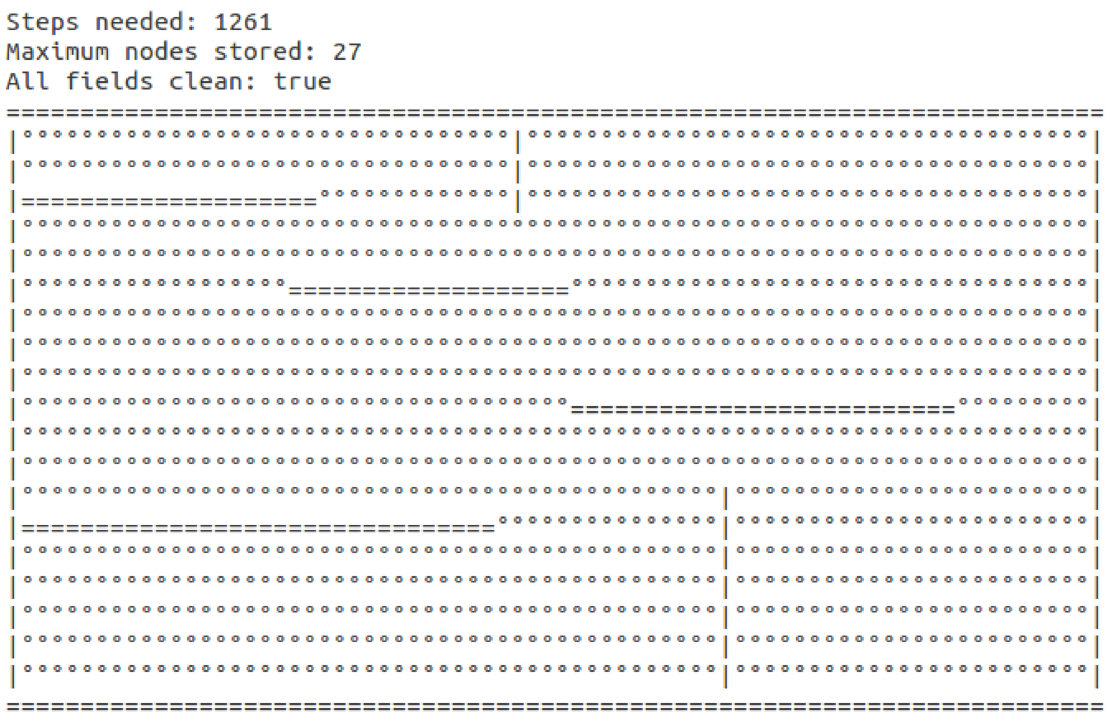
In the end Breath-First Search and Depth-First Search should be compared and for each map should be said which algorithm works better based on observed data.

2.2 Approach

I started by Dividing the task into a set of small problems. These are

- read a map file and store the map in an appropriate data structure
- localize the start position of the robot
- compute the four possible next positions (four neighbours)
- filter visited positions and positions already in list
- choose next action using breadth-first approach
- choose next action using depth-first approach
- create algorithm that uses breadth-first or depth-first
- keep track of maximum nodes stored and number of steps taken
- print out a map and results

Plain naive *BFS* or *DFS* does not care if a node already has been visited. So for this task it was important to not expand nodes that already have been expanded or have already been stored for future expanding. When expanding a node and adding its children to the queue or stack (depending on BSF or DFS), I only stored those nodes that have not been expanded before (I keep track of those nodes in a separate list) and those nodes who are not already stored in the queue or stack.
The robot cannot move through obstacles, therefore a neighbour representing an obstacle will not be added to the explore-list.



```
Steps needed: 1261
Maximum nodes stored: 27
All fields clean: true
==========================================================================
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|====================°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°===================°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°==========================°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|================================°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°|
==========================================================================
```

Figure 2.1: Result of map1 using BFS

2.3 Results

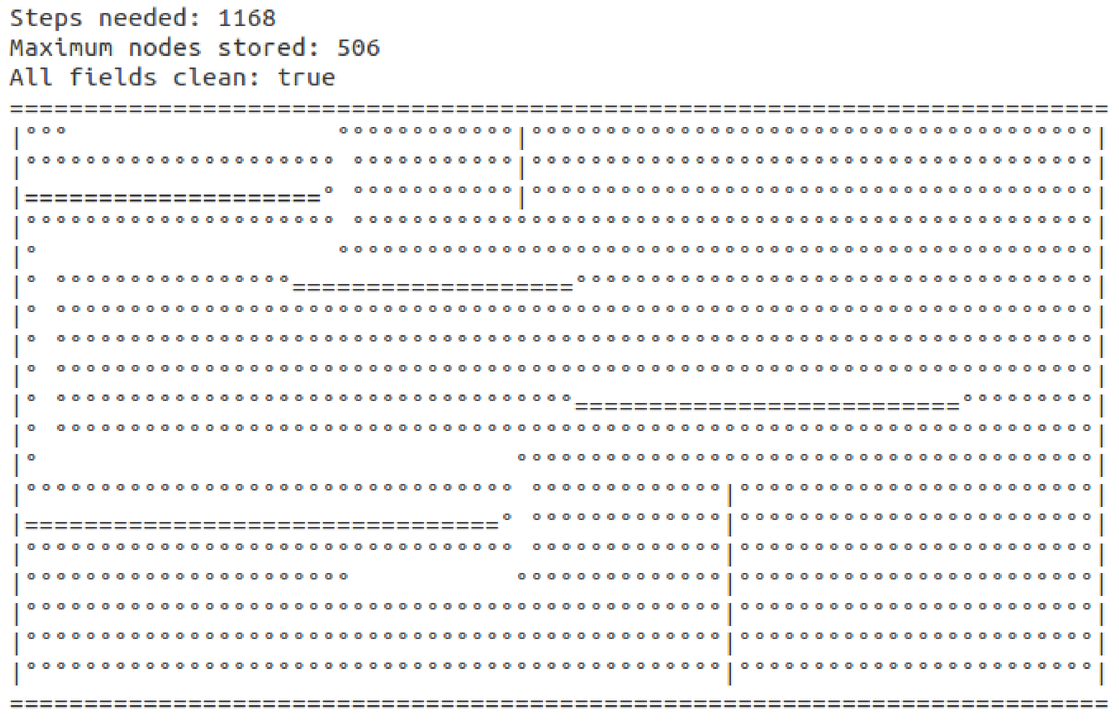
General remark: In my implementation I marked the positions the robot has visited so far using small circles. Files containing every single step can be found in the provided resource/results folder.

2.3.1 Map 1

For map 1 using BFS the robot needs to explore 1261 positions until the map is clean. It stores 27 nodes at once in its exploration map (and 1261 of course in the list of visited nodes)(see figure 2.1).

Using DFS the robot needs to explore 1168 positions and stores 506 nodes at once (see figure 2.2).

Comparing both results the DFS completes cleaning the map about 100 steps faster, but needs to store roughly 480 nodes more at the same time, which is about 20 times as much.



```
Steps needed: 1168
Maximum nodes stored: 506
All fields clean: true
==========================================================================
|ooo                  oooooooooooo|oooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooo ooooooooooo|oooooooooooooooooooooooooooooooooooooo|
|====================o ooooooooooo|oooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooo oooooooooooooooooooooooooooooooooooooooooooooooooo|
|o                    ooooooooooooooooooooooooooooooooooooooooooooooooooo|
|o oooooooooooooooo===================ooooooooooooooooooooooooooooooooooo|
|o oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|o oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|o oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|o ooooooooooooooooooooooooooooooooooo==========================ooooooooo|
|o oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|o                                ooooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooo ooooooooooooo|oooooooooooooooooooooooo|
|================================o ooooooooooooo|oooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooo ooooooooooooo|oooooooooooooooooooooooo|
|oooooooooooooooooooooo           oooooooooooooo|oooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooo|
==========================================================================
```

Figure 2.2: Result of map1 using DFS

2.3.2 Map 2

For map 2 BFS takes 1073 steps and has to store 20 nodes at the same time (see figure 2.3).

DFS needs 1078 steps, but has to store 466 nodes (see figure 2.4).

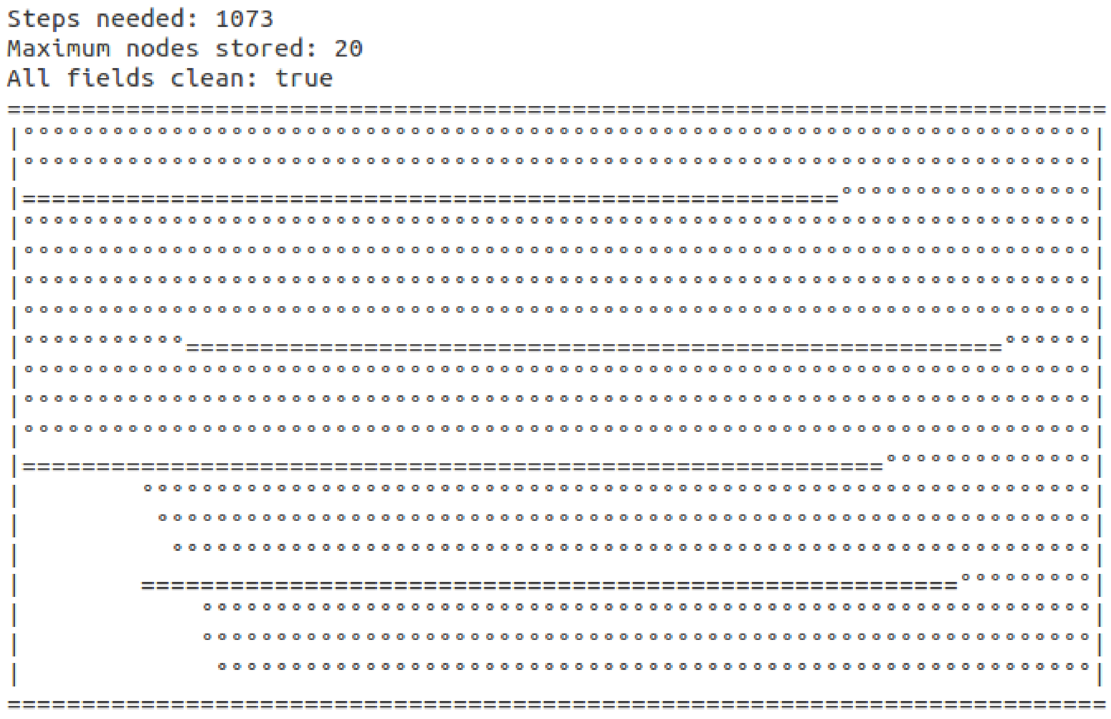
The two algorithms need about the same number of steps, but DFS needs to store about 23 times more nodes.

2.3.3 Map 3

For map 3 BFS takes 2460 steps and has to store 44 nodes at the same time. It is not able to reach and clean all fields in the map (see figure 2.5).

DFS cannot reach all of the fields as well and therefore also needs 2460 steps, but has to store up to 878 nodes at once (see figure 2.6).

This time again DFS needs to store about 20 times the amount of nodes than BFS. Both algorithms fail to clean the whole map.



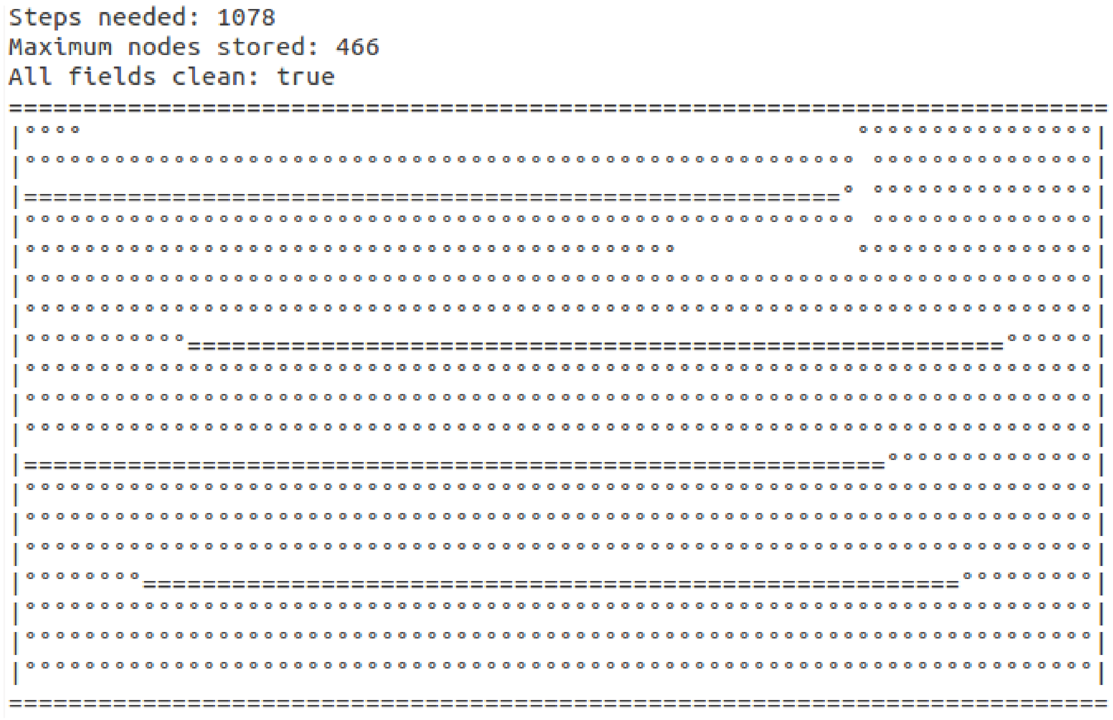
```
Steps needed: 1073
Maximum nodes stored: 20
All fields clean: true
==========================================================================
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|=======================================================ooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|ooooooooooo=======================================================oooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|==========================================================oooooooooooooo|
|        oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|         ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|          oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|        =======================================================ooooooooo|
|            oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|            oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|             ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|                                                                        |
==========================================================================
```

Figure 2.3: Result of map2 using BFS

2.3.4 Conclusion

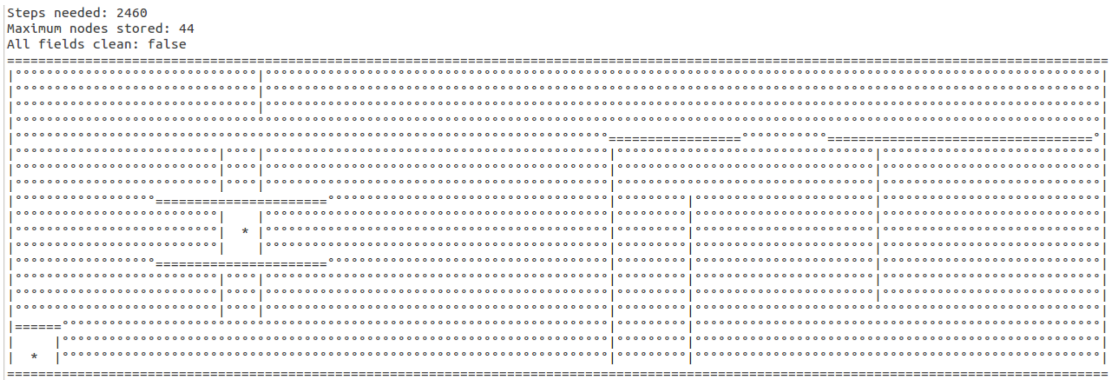
Both algorithms are complete. They find a solution if one exists. Both are not optimal because they do not find the optimal solution.
The time complexity of DFS for the first map is about 8% better. But for all the maps BFS has the better space complexity.

For my architecture and the given maps neither the time advantage for map 1 of DFS nor the space advantage in all maps for BFS are significant enough to clearly favour one algorithm over the other as for both are complete. But thinking about way bigger problems the BFS approach is the one to go with.
Both algorithms have to look for the child nodes of every node in their list of nodes to expand. So having a list 20 times bigger will result in much more computation time for DFS.



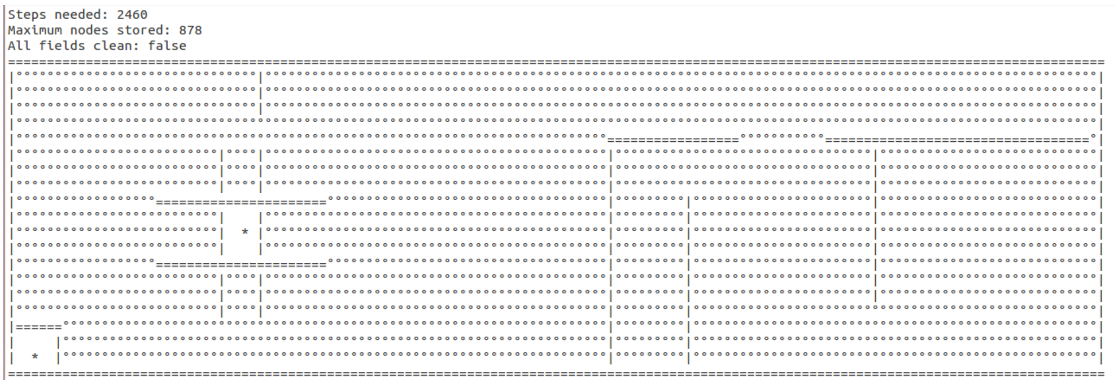
```
Steps needed: 1078
Maximum nodes stored: 466
All fields clean: true
==========================================================================
|oooo                                                    oooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooo ooooooooooooooo|
|=======================================================o ooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooo ooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooo            oooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|ooooooooooo=======================================================oooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|==========================================================oooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooo=======================================================ooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
==========================================================================
```

Figure 2.4: Result of map2 using DFS



```
Steps needed: 2460
Maximum nodes stored: 44
All fields clean: false
=============================================================================================================================================
|ooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo=================ooooooooooo==================================o|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooooooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooo======================oooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|    |oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|  * |oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|    |oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooo======================oooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|ooooooooooooooooooooooo|oooooooooooooooooooooooooooo|
|oooooooooooooooooooooooooo|oooo|oooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|oooooooooooooooooooooooooooooooooooooooooooooooooooo|
|======oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|oooooooooooooooooooooooooooooooooooooooooooooooooooo|
|     |oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|oooooooooooooooooooooooooooooooooooooooooooooooooooo|
|  *  |oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo|ooooooooo|oooooooooooooooooooooooooooooooooooooooooooooooooooo|
=============================================================================================================================================
```

Figure 2.5: Result of map3 using BFS



```
Steps needed: 2460
Maximum nodes stored: 878
All fields clean: false
=============================================================================================================================================
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°=================°°°°°°°°°°°==================================°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°======================°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|    |°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|  * |°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|    |°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°======================°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|======°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|     |°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
|  *  |°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|°°°°°°°°°|°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°|
=============================================================================================================================================
```

Figure 2.6: Result of map3 using DFS