Project 1 Part 1 Q1-4 (Search in Pacman) David Hoang, 913611912

CSC 665 Spring 2018

Depth First Search

Depth first search uses a stack for its implementation. A list called “explored” is used to store states so that it can be checked whether they were previously expanded. To begin with, the initial start state is put in to a tuple together with an empty “path” list. This is pushed to the stack (the fringe).

It enters a while loop that checks whether the fringe is empty, and since it is not, the node is popped off the stack. It is checked whether it is a goal state, then checked whether the state had already been explored. If both conditions are false, the position of the node is recorded in the “explored” list, and the node’s successors are fetched using the getSuccessors() function. Then the successors are pushed as a tuple of the successors’ position and path that had to be taken to get that position.

This process repeats, with the topmost node in the stack always getting popped off first, recording the path as it explores the maze, then eventually returns the path to the goal.

Breadth First Search

The same as depth first search, but uses a queue for its data structure.

Uniform Cost Search

The same as depth first search, but uses a priority queue as its data structure. The way that nodes are pushed to the priority queue are also different, since it uses the PriorityQueue’s update function instead. It puts nodes in to the queue based on cost (priority).

A* Search

The same as uniform cost search, but the manhattanDistance function is used as the heuristic. The heuristic function’s result is added to the cost, then the node is put in to the queue based on that sum.

Comparison using mediumMaze

Depth first search turns out to find the goal the fastest. However, it is not the most optimal solution. Breadth first, uniform cost search, and A* search all found the optimal solution in mediumMaze. Breadth first and uniform cost search turn out to be the same in number of nodes expanded, but A* is slightly faster at finding the path to the goal.

Comparison using bigMaze

Similar to the mediumMaze, depth first search finds the goal the fastest. Because there is only one solution, all the searches end up finding the optimal solution. Breadth first and uniform cost search turn out to be the same in number of nodes expanded, but A* is slightly faster at finding the path to the goal.

Thoughts

All the problems in this assignment turned out to be mostly the same, just a different data structure used and some added priority functions for UCS and A* search.

I had trouble with the auto-grader for some time, despite having run each of my implementations in the Pacman game and having it work correctly. For all the tests in the auto-grader I was missing the last node. This was because the auto-grader checks all nodes popped, but I had implemented it so that it would return the path to the goal without needing to pop the last node. So, it boiled down to reordering the loop to first pop the node, then check if that node was a goal, rather than checking if the successor was a goal and returning the path from there.