CSE 150 Homework 1 Report

Eli Eshel-Krohn A11283400
Joshua Anthony A14281769
Andrew Hwang A11570188

Problem Descriptions

Problem 1: Finding a Fixed Food Dot using Depth First Search

The first search algorithm we implemented was Depth First Search, which always expands the deepest unexpanded node. The data structure used in this search algorithm is a stack, which is a last-in-first-out data structure. The push and pop operations add and remove items to the top of the stack. Depth first search is complete only in finite spaces, as it fails for infinite depth spaces and spaces with loops. We modified the algorithm so that it would keep track of the already expanded nodes so that it would only expand each node once, making the algorithm complete. DFS is not optimal because it will stop at the first solution in finds, which may not have the lowest cost, as it expands the deepest nodes first. The time complexity is $O(b^m)$ where b is the branching factor and m is the maximum depth, which is poor is m is much larger than the depth of the solution. DFS is linear in depth as the space complexity is O(bm).

Problem 2: Breadth First Search

The Breadth First Search uses a queue in order to expand a layer of nodes at a time. The queue is a first in first out data structure where the push and pop operations add items to the end of the queue and remove items froms the front of the queue respectively. Breadth first search is complete in finite spaces and optimal if the cost of each step is equal. The space complexity is $O(b^d)$, where b is the branching factor and d is the depth of the solution.

Problem 3: Varying the Cost Function

The Uniform Cost function utilized a Priority Queue in order to find the lowest cost path to the goal. Every time a successor is found, the cost of the path so far is added to the cost of the action just taken and pushed onto the priority queue. The priority queue finds the successors of the node at the end of the lowest cost path. Uniform Cost is complete if all of the costs are positive. The lowest cost path found by UF is the optimal path. The time and space complexity of this algorithm are both O(b^(1+[C*/ε])).

Problem 4: A* search

Similar to Uniform Cost Search, A* search also uses a Priority Queue. However, A* uses an admissible heuristic to estimate the total cost of the path to the goal. A* is complete, even for infinite graphs unless the goal is infinitely deep. A* finds the optimal solution if the heuristic is

consistent, because this means that f(n) is nondecreasing along any path. A* search takes exponential time and uses $O(b^m)$ space, where b is the branching factor and m is the maximum depth, because it potentially keeps all nodes in memory.

Problem 5 Finding All the Corners

The finding all corners problem asks for the shortest route to visit all 4 corners in the maze. In order to find the optimal solution bfs was used with our implementation of the corners problem. In addition to the triple that returns the successor's current position, action, and cost; we also stored a tuple of 4 flags, one for each corner in the maze. The isGoal function returned true if all of the flags were marked true. Whenever the successors of a node were retrieved a check on whether or not the successors vertex was a corner was used, marking the flag true if it was. The breadth first search found the first instance of all of the corners being marked therefore finding the optimal solution.

Problem 6: Corners Problem: Heuristic

In order to solve the Corners Problem, which utilizes A* search to collect the food in all four corners of the maze, a heuristic which takes into account the mazedistance to the next corner is used. The purpose of this heuristic is to reduce the number of nodes expanded during the search by simplifying problem. The heuristic that we have implemented reduces the problem to searching for a single corner. The heuristic is implemented by adding an unvisited corner into a list and returning a value based on the mazedistance between the current position and unvisited corner. Once a corner is visited the heuristic returns a value based on the next unvisited corner until all of the corners have been reached. This heuristic is admissible because mazedistance returns the true cost to get to the current corner that it is searching for which makes h(n) = h*(n), therefore h(n) <= h*(n) is true. In order to prove consistency the equation h(n) = c(n,a,n') + h(n') is taken into consideration. This equation holds true because the cost of each step is 1 and the heuristic for h(n') is smaller than h(n) by 1. These values are true because mazedistance returns the true cost of reaching a corner from a position, which will go down by 1 for each action taken. Since h(n) = c(n,a,n') + h(n') is true the equation h(n) <= c(n,a,n') + h(n') also holds true, therefore the heuristic is consistent.

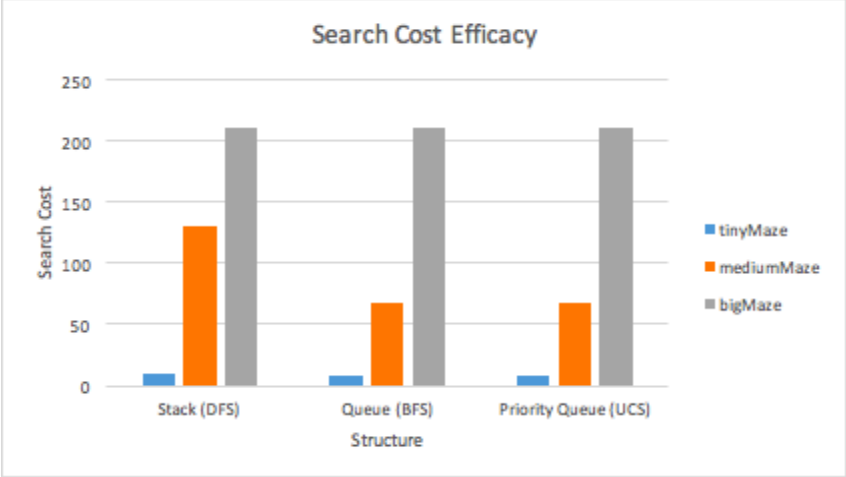
Problem 7: Eating All The Dots

Eating all the pacman food in as few steps as possible is a complicated search problem that, with the null heuristic, will expand 10s of thousands of nodes during the A* search. In order to reduce this to less than 9000 nodes, a heuristic that returned the maze distance to the next food location was used. When there is no more food available the heuristic returns 0 as this is the goal state. This heuristic is admissible because it uses the maze distance to the food, which is equal to the cost to get to the food as the cost of each step is 1. This heuristic is consistent as it satisfies the equation h(n) <= c(n,a,n') + h(n'). This is because the maze distance is the true cost of moving to the food, thus h(n) = c(n,a,n') + h(n').

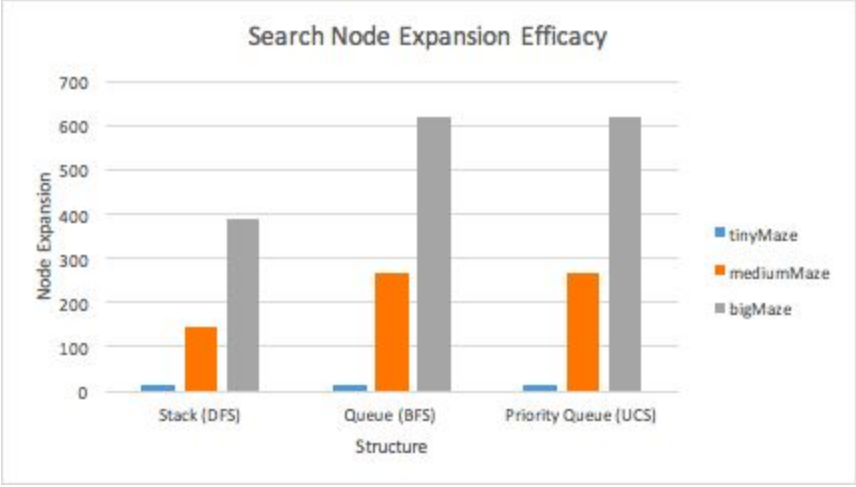
Problem 8: Suboptimal Search

This search was implemented using a greedy algorithm. It called on UniformCostSearch which used a priorityqueue (but could have been done using any optimal algorithm) to find the path between nodes and only focused on getting to the closest node with a dot inside of it. The goal was realized by checking if there were no longer any food dots on the map. Similarly to with problem 3, which also uses a priority queue, it runs with a O(b^(1+[C*/ε])) time and space complexity. It is important to realize that this method is not optimal due to its nature as a greedy algorithm. It only focuses on getting the shortest path between each dot found. In some cases the path is not linear and branching can lead to choosing a less optimal path, since algorithm does not look beyond a single step. For example, say I have a graph where the node can go east, west, and north as the next step. It may actually be optimal in the long term to go north, but all are seen as equal costs, so the choice is arbitrary.

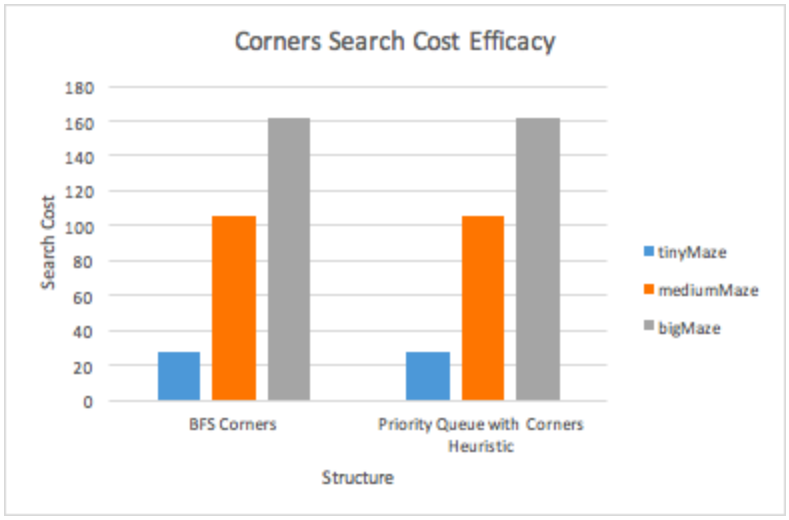
Analysis



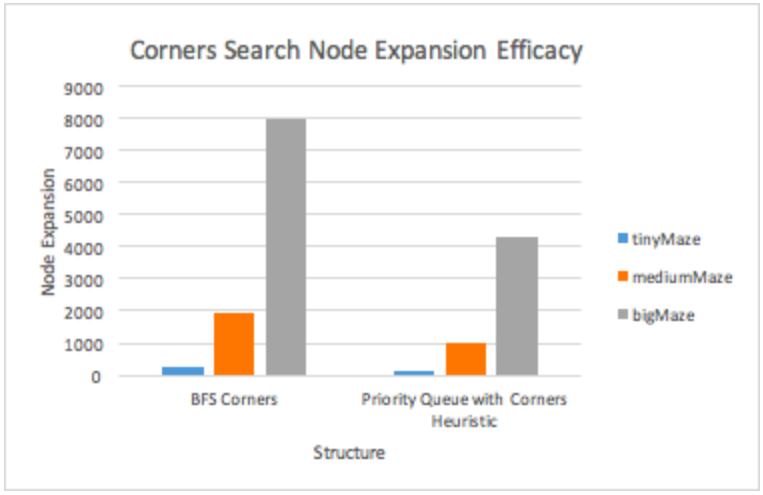
Compares the search cost optimality of the three data structures used in the first four standard search problems



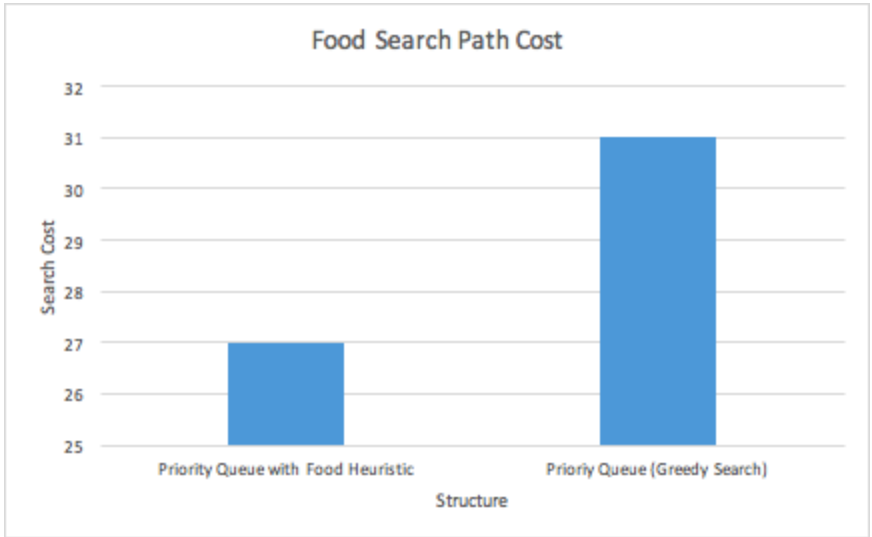
Compares the node expansion optimality of the three data structures used in the first four standard search problems



Compares the search cost optimality of the two data structures used in the corners search problems



Compares the node expansion optimality of the two data structures used in the corners search problems



Compares the search cost for a tinySearch of the algorithms written to eat all of the dots

The evidence shown by this data points to several conclusions. We see that bfs is better than dfs at finding the shorter path and priority queue is better still in that it finds the shorter weighted path. We also see that using an admissible and consistent heuristic should be more efficient than using no heuristic. It also beats out greedy search.

To conclude, certain algorithms are more efficient and a good heuristic can seriously improve an algorithm.

Contributions

Josh

I worked on problem 7. I also contributed to solving problems 1-5 and writing the report.

Andrew

I worked on implementing the corner's heuristic for problem 6. I also helped with problems 1-5 and writing the report on the parts that I worked on.

Eli

I worked on problems 1, 2, 5, and 8 as well as the report.