Assignment 3 Write up

Lauren Mitchell

The purpose of the assignment is to compare and contrast A\* algorithm and Dijkstra’s algorithm. It was not until this assignment that I realized the similarities between the two. In fact, they are almost the same algorithm, where Dijkstra is just a special case of A\*.

In A\*, the total cost F(n) is calculated from the real cost, g(n), and the heuristics, h(n). Dijkstra’s algorithm does the same calculation for total cost, except the total cost equals the real cost and the heuristic is zero. A\* algorithm is a more efficient algorithm because it prioritizes the nodes from the heuristic where Dijkstra will explore all possible paths.

After running both algorithms, it shows that A\* is more efficient. Although, they do find the same shortest same with the same cost, A\* evaluates only the nodes it needs to where as Dijkstra will explore all the nodes until it has found the final path. Because we are giving the heuristic in A\*, we are able to eliminate a portion of the nodes that we do not need to explore.

Procedure for A\*

In my A\* algorithm, I passed in graph, start, and end as my parameters. I used a Priority Queue to keep track of my nodes. In my Graph class, I have an a dictionary represented by graph\_dictionary to keep track of all the nodes in my dictionary. I initialized an empty dictionary for previous which represents the previous node(s) visited and a cost dictionary to keep track of the cost so far. There is no node before previous and the initial cost at the start is 0. I begin by looping through all the nodes in graph\_dictionary and get the neighbors or the adjacent nodes to the current node that it is at. I get the weight to the neighbor weight and add it to the current cost, giving us a new updated cost called newCost. We then check if the newCost is less than the cost of the nextNode. If it is, then we update that cost to equal the newCost. We then want to incorporate the Heuristic value for that node. With our new updated cost, with the Heuristics included, we all add that back into the priority queue.

After running the algorithm, my solution is:

cost: 26

Number of nodes evaluated: 14

Procedure for Dijkstra

I begin by creating an empty list called solved to store the solved nodes. I have it hard coded for the startV to be S and the endV to be F. I add the first vertex to the list solved. I then begin a while loop for when endV is not solved. I go into my solved list, look at the top of the list and find the neighboring nodes for the current node. If those neighboring nodes are not solved, we look at the cost and update the cost into newCost. (similar to A\*). If that newCost is less than the minCost, we update the solvedV value to the be the next node, update the minCost to be the newCost. We then loop through all that until we have reached the final node, ‘F’.

After running the algorithm, my solution is:

Cost: 26

Number of nodes evaluated: 20

A Picture of My Graph:

