Jana Klopsch

U0854469

Analysis 09

1. Who is your programming partner? Which of you submitted the source code of your program?

Savannah Simmons was my programming partner for assignment 09, and I submitted the source code.

2. Evaluate your programming partner.

Savannah Simmons was my programming partner for assignment 09, and she was great to work with! We were very good at thinking through problem solving of the assignment, and had similar thoughts for implementing our ideas. We were great at being engaged while coding, thinking of issues before they would found in testing.

3. Does the straight-line distance (the absolute distance, ignoring any walls) from the start point to the goal point affect the running time of your algorithm?

If the program ignored walls, the running time would be increased for this algorithm (although not significantly). Without walls, every node is placed in the queue, which therefore means every neighbor of every node is placed in the queue. This then means we have a very dense graph, where every node has four edges that need to be checked. By having to examine every node on the graph, the queue could be extremely large by the time the goal node is placed in the queue. However, there is the exception of when the goal node is very close to the start node, in which case it would be seen quickly. Thus, the running time is affected significantly, if it takes a significant amount of time to examine a node. However, because this algorithm implements a Breadth-First Search, it is guaranteed to find the shortest path.

4. Explain the difference between the straight-line distance and the actual solution path length. Give an example of a situation in which they differ greatly. How do each of them affect the running time of your algorithm? Which one is a more accurate indicator of run-time?

The straight-line distance would be if we could traverse the nodes straight from the start node to the goal node. For example, if the start node is in the top left corner of the graph, and the goal node in the bottom right corner of the graph, the straight-line distance would be to go all the way across the top to the right, and then all the way down until we hit the goal node, or all the way down from start, and then all the way across to the right (assuming we still cannot travel diagonally).

The actual solution path could be more of a zig-zag shape, where it might have to travel farther from the goal node first, but then back toward it later. In comparing the straight-line

distance of a maze and the actual solution path distance, the distance for both would be the same for the straight.txt maze in the sample mazes provided, however the straight-line distance for bigMaze.txt is much shorter than the actual solution path.

In comparing the running times of these situations, it seems that solving the actual maze would be faster than solving the straight-line path. However, in looking at the bigMaze.txt, I would expect the running times to be very similar, because although the straight-line path is much shorter, four nodes are placed in the queue at a time, rather than only one or two at a time. Again, by ignoring the walls, the graph becomes very dense, and the queue is filled very quickly, however by acknowledging the walls, the graph is made much easier for finding the shortest path to the goal node.

5. Assuming that the input maze is square (height and width are the same), consider the problem size, N to be the length of one side of the maze. What is the worst-case performance of your algorithm in Big-Oh notation? Your analysis should take in to account the density of the maze (how many wall segments there are in the field). For example, a completely open field with no walls other than the perimeter is not dense at all, as opposed to the example maze given "bigMaze.txt", which is very dense. There is no one correct answer to this since solutions may vary, but you must provide an analysis that shows you are thinking about the problem in a meaningful way related to your solution.

The worst-case scenario for our algorithm would be an empty square maze with the start node in the top left corner of the graph, and the goal node in the bottom right corner of the graph. The performance of this scenario would be $O(N^2)$. With no walls, every node would have to be placed in the queue to check if that node is part of the path from the start to the end node. Because the empty maze is a square grid of length N, there are N^2 nodes, thus the performance would be $O(N^2)$.

For a maze that is “dense”, or a maze that has many walls, the performance is much closer to the number of nodes that are not walls. For example, in a square graph of length N, there are N^2 nodes, say $N^2/2$ of them are walls, we would expect the Big-Oh notation to be $O(N^2/2)$. This behavior would account for every node that is not a wall being checked in the algorithm, with the very last node examined being the goal node. Thus, the Big-Oh notation for our algorithm is directly correlated to the number of empty spaces needed to traverse between the start and goal nodes.

6. How many hours did you spend on this assignment?