Brian Burton
CS 2420
03/25/17

# Assignment 9 Analysis Document

Answer the following questions:

1. 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? In other words, does how close the Start and Goal are affect how fast your algorithm finds a shortest "Manhattan" distance?

Our program executes so fast that it really does not matter where the start and goal values are placed. Not only that, but our algorithm has to check every single vertex no matter where the start and goal values are placed. It needs to check all of them in order to find the shortest path. Therefore it is pretty close to the same amount of run time no matter what. The thing that will change the run time is how big the maze or file is and where the wall are placed.

2. 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 path is a path that ignores all variables and only looks at how far the start and goal values are. The actual solution path finds the shortest path while not ignoring all of the variables. If the shortest path has to go all the way around the maze because of the way that the walls are placed then it will certainly take more time to compute. The straight-line distance can affect the run time since it has to determine a straight path, while the actual solution path affects it since it has to look at all of the nodes. The actual solution is a more accurate indicator of run time because it takes longer to compute.

3. 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).

   Create a test example to verify your thinking. If possible, augment your code to keep a counter of how often a node is looked at (i.e., every time your code checks to see if a node has already been visited, it is being "looked at").

The worst case performance or Big – Oh notation is when N is one hundred. If there are no walls then that would be quite the easy maze, however, if the walls made you zig zag through the maze then that would be the worst solution. This is because you had to go through the entire maze.

4. Hypothesize what affect using depth first search would have on the path planning? Provide an example maze for the purpose of this discussion. Would depth first solve the above example in a faster or slower manner?

   Provide enough discussion to show us you have seriously considered this as well as all the other questions.

I assume that if we were to use depth first search for our maze program then we would have to look at all of the nodes in addition to determine which path was the shortest by comparing each of them.

5. One more thought problem (don't spend more that 15 minutes on this). Say you created an entire matrix of nodes representing the maze and did not store any edges. To know if an edge exists for any node at (R,C) you would have to check (R-1,C), (R+1,C), (R,C+1), and (R,C-1). Can you think of an other algorithm to find the shortest path from a given start (R1,C1) to a goal (R2,C2).

I think you could just go through each possible way to go through the maze, and then compare them against each other to see which maze has the least amount of edges. It would defiantly take more steps but I believe that it would be a possible solution.

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

Caleb and I spent every night from 9pm until midnight Monday through Friday on this problem. Then we worked after class on Tuesday from 11am until 2pm. That makes a grand total of 18 hours spent on this project.