Adrian Bollerslev

Assignment 09 Pacman Analysis

When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 10% of your assignment 9 grade. Ensure that your analysis document addresses the following.

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

Joshua Shipley, u919708

2. Evaluate your programming partner.

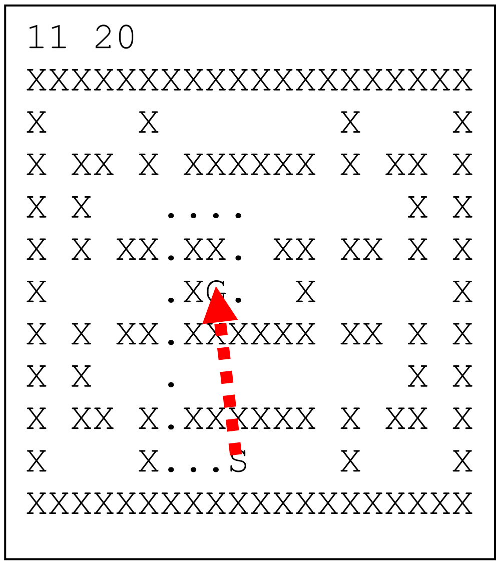
Good, I enjoy working with him. I think we divide the work evenly. Last assignment I did a little less work while this assignment he did a little less. He has a good knowledge of the concepts.

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?

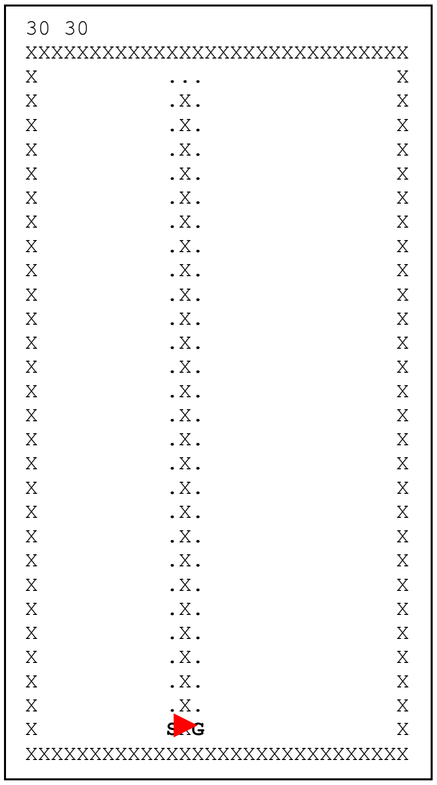
The straight-line distance doesn't directly affect the runtime of our algorithm but the shortest possible path does. However, the distance of a straight-line has a slight correlation with the distance of the shortest possible path, so in that sense it indirectly affects our runtime. When finding the shortest path, the while loop of the algorithm stops when the shortest path is found and therefore a shorter path directly affects runtime. Once we have found the shortest path to the ending cell we begin moving backwards (and putting down '.'s). A shorter path will again speed up runtime as the for loop in this method is directly correlated to the shortest path length. The absolute distance is not a reliable measure of runtime. We can see this by looking at the provided maze "turn.txt" in which the absolute distance is very small but a large wall makes the path quite long. Due to the long path this maze takes long to complete despite having a very short absolute distance. In summary, only because the absolute straight-line distance generally correlates with the shortest path does it affect the runtime. It is not an accurate measure because sometimes the maze will place start close to end but still require a long 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 is the distance from S to G assuming there are no walls and that we can move diagonally. It is the actual shortest possible distance, irrespective of the guidelines we have followed to solve the mazes.



```
11 20
XXXXXXXXXXXXXXXXXXXX
X    X        X    X
X XX X XXXXXX X XX X
X X   ....       X X
X X XX.XX. XX XX X X
X     .XG.  X      X
X X XX.XXXXXX XX X X
X X   .          X X
X XX X.XXXXXX X XX X
X    X...S    X    X
XXXXXXXXXXXXXXXXXXXX
```



```
30 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X          ...               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          .X.               X
X          S.G               X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Pictured:

Left: classicSol.txt

Right: turnSol.txt

Here are two examples demonstrating an "average" maze and an "outlier" maze. The shortest path is demonstrated by a '.' while the absolute straight-line distance is shown through a red arrow. As discussed above, the key affect the absolute distance has on runtime lies in its correlation with the shortest possible path. The classicSol.txt pictured on the left shows a fairly typical maze. The shortest possible path clearly has a correlation with the absolute distance. The turnSol.txt pictured on the right shows a rather extreme example. This absolute distance is incredibly short with only one "character" between the start and finish. However, due to the way the walls are setup, the shortest possible path is quite long. Due to the long shortest possible path, and, the large amount of white space to "check", this maze will take quite long for its grid size. While in the majority of cases the absolute straight-line distance can be used to "estimate" the runtime, it is clearly not a reliable measure. The algorithm is directly affected by the actual shortest path, the size of the grid, and the number of white spaces it must "check". These are only three of the many factors which could be used to estimate runtime much more consistently than the straight-line distance. In short, the straight-line distance is not a reliable or accurate measure to estimate the runtime.

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 algorithm we wrote uses Breadth-First Search. As the name implies, this algorithm searches all the possible paths of a distance then increases the distance by 1. It puts the starting node in the queue then puts all the edges of the starting node in the queue. It then puts all of their edges in the queue…etc.

This can be written as:
(Node1+Node1Edges)+(Node2+Node2Edges)+(Node3+Node3edges)=
NodeN+NodeNedges

Due to the way, this algorithm is written each Node is visited twice, once as the current explored Node and once as an unexplored "edge Node". A dense graph will mean that we will not have to check each node twice while a more open graph will ensure we do so. If we think of N as the length of the side then N^2 is the total amount of nodes. I believe the algorithm will best case be O(N^2) runtime because that means each node was check once and it was a very dense graph. I believe worst case O(N^4) because each node and all its vertices checked.

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

11

Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.

Upload your solution (.pdf only) to the assignment 9 webpage by 11:59pm on November 2.