Akash Kumar

Dr. Burns

CSC 575-01

2/15/2021

BFS Maze Analysis

Setup:

To output statistics on the time required to solve the maze using Breadth First Search (BFS), I used Python's *time* module. Specifically, I used the *time*() function in that module to measure the start and end times. The start time is recorded immediately before the call agent.bfs(maze, goal), while the end time is recorded immediately after the call returns. Subtracting the start time from the end time will give us the time elapsed for the BFS function. This time is measured in seconds. To convert to microseconds (µs), I multiplied the time by 10^6. To see the effect of differently sized mazes on the performance of the BFS function, I called the BFS function on 5 different mazes. Lastly, to account for hardware and environmental factors, I timed each BFS function call multiple times using a for loop and used the average of those times as the time elapsed for solving each maze.

Here are the results from running the code on 5 different mazes:

Grid 1:
Time taken to find the solution path using BFS: 951.654 µs.
Total number of nodes explored: 133
Length of the solution path: 35

Grid 2:
Time taken to find the solution path using BFS: 52.003 µs.
Total number of nodes explored: 7
Length of the solution path: 4

Grid 3:
Time taken to find the solution path using BFS: 131.471 µs.
Total number of nodes explored: 19
Length of the solution path: 10

Grid 4:
Time taken to find the solution path using BFS: 219.830 µs.
Total number of nodes explored: 31
Length of the solution path: 12

Grid 5:
Time taken to find the solution path using BFS: 17448.688 µs.
Total number of nodes explored: 2185
Length of the solution path: 105

Time and Space Analysis of BFS:
Since the agent always starts from the top left-most location of the grid (1,1) and since all other maze locations have at most 3 possible directions to choose from, let the branching factor of the BFS tree be 3.

The root (start node) of the tree is at level 0.
Let b be the branching factor. Then, b = 3.
Let d be the depth of the optimal solution. This is equivalent to the length of the optimal path or the level of the goal node in the tree.

The time complexity of BFS is $O(b^d)$ (exponential growth).
The space complexity of BFS is $O(b^d)$ (exponential growth).

Since the big-O time and space complexity of BFS is exponential, as the depth d of the BFS tree increases, the time and space required will increase exponentially. To measure this exponential growth empirically, let's look at 2 ratios:

1) Time elapsed / d
2) Space used / d

If the time and space complexity of BFS is linear, the 2 ratios will remain relatively constant as d increses. If the complexities are exponential, then the 2 ratios should increase exponentially (since the derivative of an exponential is an exponential).

Here is the table of the ratios for the 5 different mazes:

d (length of solution path)	Time elapsed (in µs) / d	Space used (# of explored nodes) / d
4	13.00	1.75
10	13.15	1.90
12	18.32	2.58
35	27.19	3.80
105	166.18	20.81

From the above table, we see that the ratios do in fact increase as d increases.
To determine if the ratios are increasing linearly or exponentially, I divide the ratios by d (derivative of the derivative) to get new ratios. If the new ratios remain relatively constant as d

increases, this means the time and space complexities are quadratic at worst. If the new ratios are increasing, then this suggests that the time and space complexities may be exponential.

Let's look at d=10 and d=105 to compare the ratios:
For d=10,

(time elapsed)/d^2 = 13.15/10 = 1.32

(space used)/d^2 = 1.90/10 = 0.19

For d=105,

(time elapsed)/d^2 = 166.18/105 = 1.58

(space used)/d^2 = 20.81/105 = 0.20

The new ratios increase as d increases, so the time and space complexities of BFS are worse than quadratic time. Since big-O is about the worst case, this is in agreement with the exponential big-O time and space complexities of BFS.