2)

If the node in the middle (on top of the triangle) is labeled 1 the DFS will get stuck in a loop where the node labeled 1 always will be explored before the node labeled 2. To fix this we can either change the labeling so the node on top will have a higher label than 2. Or change the prioritizing order to avoid getting stuck in a loop. It can be that it prioritizes unexplored nodes, and if both nodes are unexplored, it chooses the lowest. Or you could modify it to be a iterative depending DFS so it starts with doing a DFS with the limit of one, than two and so on. This would allow it not to get stuck in a loop.

4 a)

Paths:

Iteration 1: 43, 44

Iteration 2: 43, 44, 34

Iteration 3: 43, 44, 54

Iteration 4: 43, 42

Iteration 5: 43, 42, 32

**Start**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |

**Iteration 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 42 | 1 | 5 | 6 |
| 44 | 1 | 3 | 4 |
| 53 | 1 | 5 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 44 | 1 | 3 | 4 |

**Iteration 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 42 | 1 | 5 | 6 |
| 44 | 1 | 3 | 4 |
| 53 | 1 | 5 | 6 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 44 | 1 | 3 | 4 |
| 34 | 2 | 2 | 4 |

**Iteration 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 42 | 1 | 5 | 6 |
| 44 | 1 | 3 | 4 |
| 53 | 1 | 5 | 6 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 44 | 1 | 3 | 4 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |

**Iteration 4**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 42 | 1 | 5 | 6 |
| 44 | 1 | 3 | 4 |
| 53 | 1 | 5 | 6 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |
| 64 | 3 | 5 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 44 | 1 | 3 | 4 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |
| 42 | 1 | 5 | 6 |

**Iteration 5**

|  |  |  |  |
| --- | --- | --- | --- |
| Open cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 42 | 1 | 5 | 6 |
| 44 | 1 | 3 | 4 |
| 53 | 1 | 5 | 6 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |
| 64 | 3 | 5 | 8 |
| 32 | 2 | 4 | 6 |
| 41 | 2 | 6 | 8 |
| 52 | 2 | 6 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| Closed Cell | G | H | F |
| 43 | 0 | 4 | 4 |
| 44 | 1 | 3 | 4 |
| 34 | 2 | 2 | 4 |
| 54 | 2 | 4 | 6 |
| 42 | 1 | 5 | 6 |
| 32 | 2 | 4 | 6 |

**b) A\***

A\* solved this problem with a total of 71 operations, making it the second most efficient method among the three. A\* combines the features of lowest-cost first search and best-first search, making it an informed search algorithm that utilizes heuristics. It determines the next node to explore by combining two values: the distance from the start node to the current node and the estimated distance from the current node to the goal node. The algorithm selects the node with the lowest value, indicating a shorter path from the start node to the goal node. A\* has the advantage of always finding the shortest path. However, it is complex and susceptible to obstacles in the graph. This drawback becomes apparent when solving this problem since the grid contains "walls." The heuristic does not account for these obstacles, resulting in some "unnecessary" exploration by A\*, leading to a few extra operations compared to best-first search. Nevertheless, A\* outperforms breadth-first search (BFS) due to its informed search approach.

**b) BFS**

BFS is utilized to solve this problem, requiring 364 operations and making it the least efficient method among the three. BFS is an uninformed search algorithm that systematically explores all surrounding nodes until it encounters the goal node. It starts by exploring all neighboring nodes at a depth of one, followed by depth two, depth three, and so on. The advantage of BFS is that it always finds the shortest solution if one exists. However, the algorithm requires a significant amount of memory to store all paths up to a certain depth. This drawback becomes evident while solving this problem, as BFS visually expands across the grid in a systematic manner, both towards the goal node and areas where there is no goal. As a result, a large number of operations (364) are needed to reach the goal, making BFS significantly less efficient than both A\* and best-first search for solving this problem.

**b) Best-first search**

Best-first search is employed to solve this problem with only 48 operations, making it the most efficient method among the three. Best-first search is an informed search algorithm that, similar to A\*, utilizes a priority queue based on heuristics to determine which node to explore next. However, unlike A\*, best-first search does not consider the distance from the start node when selecting the next node. It focuses solely on the estimated distance from the current node to the goal. The advantage of best-first search is its higher efficiency compared to BFS. However, it can still be susceptible to obstacles in the graph, similar to A\*. The reason why A\* is more affected by this disadvantage than best-first search in this case is that A\* takes the distance from the start node to the current node into account. Consequently, some explorations in the "wrong direction" receive relatively low rankings since those nodes are closer to the start node. Best-first search, on the other hand, only considers the estimated distance to the goal node, resulting in slightly better performance than A\* in this particular case.

**C)**

**A picture containing square, colorfulness, rectangle, pixel

Description automatically generated**

**A picture containing square, screenshot, rectangle, pixel

Description automatically generated**

In this case the best-first search starts on a path that gets it closer to the goal, so much that all the nodes inside the “maze” is closer to the goal than any other nodes surrounding the starting node. Therefore it only explores the path within the maze. Breadth-first does not get stuck in the maze and finds the shortest path.