Alicia Sang
4.3.2025
Data Structures and Algorithms

Lab 4 Reflection

1. When comparing DFS and BFS on the same maze,BFS consistently finds the shortest path to the goal; because BFS explores the maze in and guarantees that the first path found is the shortest. DFS, on the other hand, often produces longer and less direct paths as it explores deeply in one direction before backtracking. In mazes with relatively few blocked cells, DFS can sometimes reach the goal quickly by chance, but the path taken is rarely the best and shortest path. When there are many blocked cells, DFS is more likely to get caught in inefficient paths or in dead ends, while BFS is more reliable in exploring all reachable areas. In terms of exploration effort, BFS tends to push more cells in open mazes, whereas DFS often pushes more cells in dense mazes because of its repeated backtracking.
2. With relatively few blocked cells, both DFS and BFS usually find a path to the goal, but BFS still returns a shorter, more efficient route. DFS may complete faster in terms of computation but typically produces a longer and less direct path. When many cells are blocked, BFS maintains its ability to find the shortest path if one exists, while DFS may waste time exploring long dead ends. DFS's performance in these cases is highly variable and often worse than BFS in both time and accuracy.
3. Search direction significantly affects DFS because it prioritizes depth. The order in which the algorithm explores neighbors can lead DFS down completely different routes, thus resulting in major differences in path length and outcome. Randomized order in particular introduces inconsistency and often results in long windy paths. BFS, on the other hand, is much less sensitive to search direction; while direction order may slightly influence which shortest path BFS finds, the overall path length and the number of cells are largely unaffected, thus making BFS more consistent across different search orders.
4. For the exact same maze, DFS produces a wide variety of different paths depending on search order; a change in direction priority can alter the entire first path, which can be a long detour, especially in large mazes. BFS, on the other hand, consistently explores all four directions, regardless of which one is first. Maze size makes these differences more prominent; DFS may go down a long, long detour depending on which direction it explores first, whereas BFS will explore all directions pretty evenly regardless of order.
5. Stack is O(1), because both push() and pop() operate at the end of the list, which is efficient and runs in constant time. Queue is O(1) and O(n), because push() is O(1), as it appends to the end; however, pop() is O(n), since all elements must be shifted one position left, which becomes inefficient with large queues. Doubly-linked LinkedList is O(1) because push()and pop() operate in O(1) time, because nodes are directly linked so no shifting is required.