|  |
| --- |
| 4-22-2023 |

|  |
| --- |
| Anushka Chinoy, Kiara Vasquez, Medhi Welsh-Brito |
| Solving the 8-Puzzle with Different Graph Traversal Algorithms |
| COT 4400 Spring 2023 |

**Abstract**

The 8-Puzzle problem is a classic problem in computer science, where there is a 3x3 grid with eight tiles numbered from one through eight, and a blank tile. Given an initial state, the goal of the problem is to rearrange the tiles until the goal state is achieved: the tiles arranged in numerical order around the edge of the grid. Solving the problem can be done by approaching it as a search problem, where each possible arrangement represents a vertex in a graph. Different graph traversal algorithms, such as Breadth-First Search, Depth-First Search, and Dijkstra’s algorithm can be utilized to find the shortest path to the goal state. This 8-Puzzle problem can be used as a benchmark for these algorithms for the purpose of evaluating each of their performances.

**Introduction**

The goal of the 8-Puzzle is to find the cheapest path to the goal state, given an initial state. With that in mind, the 8-Puzzle can be solved through the implementation of different graph traversal algorithms, where each of the possible moves generate states that are treated as vertices in an undirected graph. Three graph traversal algorithms, Breadth-First Search (BFS), Depth-First Search (DFS), and Dijkstra’s algorithm, were implemented for the purpose of exploring how each of them perform when solving the puzzle. BFS starts at the root or initial state and moves “down” through all future states by exploring all the vertices in the same level before moving to the next. Similarly, DFS starts at the root; however, it then visits each vertex to the furthest depth, before backtracking and starting again. Lastly, Dijkstra’s algorithm traditionally works by setting all the costs of the vertices to infinity. It then visits the vertex with the smallest distance and calculates the cost by adding the weight of the vertex and the cost of the edge. If the calculated cost is smaller than the original cost, then it is updated. This process is then repeated for the entirety of the graph and will result in traversal using the minimum distance to each vertex. Each of the algorithms implemented were tested with different initial states provided in a text file to see the difference in costs and paths taken to reach the goal state.

**Comparison of Solutions**

*Breadth-First Search (BFS)*

The implementation of the Breadth-First Search (BFS) algorithm utilizes a queue to explore each of the possible moves that could lead to the goal state. Starting from the initial state, BFS enqueues all the next states in the order of moving the empty tile left, up, right, and down until the goal state has been reached. As the algorithm explores each of the possible states in that level, it then moves down and explores the next level of possible moves, which have all the immediately possible states. The cost of moving the empty tile is 1, which results in the cost of the shortest path being the number of moves made to reach the goal state. It guarantees that the optimal solution is found, but it requires a significant amount of memory and time for larger search spaces.

*Depth-First Search (DFS)*

The Depth-First Search (DFS) algorithm uses a stack to explore the search space as it searches for the shortest path. The DFS algorithm explores the search space by traversing as far as possible down each path before backtracking, adding all next states to a stack and removing them one by one in backwards order (down, right, up, left). When there are no more unexplored states, it stops searching in that direction and moves back to the previous state, continuing its search from there as removing from the stack would result in. It continues to do so until the goal state has been found or until the stack is empty. Similar to BFS, the cost of moving the empty tile when generating possible moves from a given state is 1. DFS has a lower memory requirement than BFS, but it does not guarantee that the optimal solution is found.

*Dijkstra’s Algorithm*

Dijkstra's algorithm is a graph search algorithm that is used to find the shortest path between nodes in a weighted graph. The implementation of the algorithm utilizes a priority queue, so that it explores all possible paths to the goal state in increasing order. Children or next states are generated by moving the empty tile with neighboring tiles in the order of left, up, right, and down. The cost of moving a tile is equal to the value of the tile moved. Therefore, the cost of the shortest path is the sum of the value of the tiles moved to reach the goal state. This algorithm guarantees that the optimal solution is found, but it is more computationally expensive than BFS and DFS.

**Comparison of Results**

The three algorithms were run for multiple initial states, both solvable and unsolvable. The unsolvable states were correctly determined to be unsolvable and ended the execution there. The solvable ones were called for each algorithm and the path of traversal as well as the total cost were printed.

Table 1: BFS vs. DFS vs. Dijkstra Cost

|  |  |  |  |
| --- | --- | --- | --- |
| Initial State | BFS | DFS | Dijkstra |
| {1, 3, 4,  8, 0, 2,  7, 6, 5} | 4 | 28 | 11 |
| {1, 3, 4,  8, 0, 6,  7, 5, 2} | 8 | 52,634 | 30 |
| {1, 4, 8,  3, 0, 6,  7, 5, 2} | 18 | 2,798 | 70 |

As shown in Table 1, BFS consistently had lower costs than both DFS and Dijkstra’s algorithm, though the comparison to DFS holds more significance as the cost of a move in BFS and DFS is 1 while the cost of a move in Dijkstra’s is the value of the tile being swapped with the empty tile. This is because both BFS and DFS are uninformed search algorithms and operate identically on unweighted vs. weighted graphs. BFS always finds the shortest path by exploring the shallowest nodes first, and this is apparent from the dramatically lower cost in each example initial state. DFS explores deeply into one path at a time and is not always going to return the shortest path, and in the 8-puzzle it never led to the optimal solution. The cost of DFS was a strong outlier for the latter two examples shown in the table and although it has advantages for memory concerns, it had the worst performance on the 8-puzzle.

Like BFS, Dijkstra’s algorithm always returns the shortest path to any node though it specifically operates on weighted graphs. Thus, the cost for each move was given by the value of the tile being swapped and the total cost reflected higher costs than BFS, though close in range, and the nodes were explored by the lowest cumulative cost first. The speed of each computation can be visualized from the cost results as BFS completed within a second, DFS completed within a second up to a minute for the second example, and Dijkstra completed within a second.

**Conclusion**

The 8-puzzle problem can be solved using a variety of search algorithms, including BFS, DFS, and Dijkstra's Algorithm. The choice of algorithm depends on the size of the search space and the computational resources available. BFS and Dijkstra's Algorithm guarantee that the optimal solution is found, but they require a significant amount of memory and computation. DFS has a lower memory requirement but does not guarantee the optimal solution. In the case of the 8-puzzle problem, BFS found the optimal solution in the least number of steps, Dijkstra’s algorithm found the optimal solution in the least cost, while DFS found a longer path that was not optimal.

**Work Cited**

T. Brown, “Solving 8 Puzzle: Exploring Search Options - JavaScript in Plain English,” *Medium*,

Jan. 07, 2022. [Online]. Available: <https://javascript.plainenglish.io/solving-8-puzzle->

exploring-search-options-2e446e29d21