COT4400 Analysis of Algorithms

Final Project

19 April 2021

By

Param Chokshi

Kejvi Cupa

Lahari Poluparti

Description of Task:

The goal of this project was to find the cheapest solution to the 8-puzzle using Dijkstra's, BFS (Breadth-First-Search on Graphs) and DFS (Depth-First-Search on Graphs) algorithms. We were given the initial state and the goal state. Each algorithm had a predetermined cost of moving the tiles of the 3x3 grid, which were taken into account when calculating the cost of each algorithm and determining the cheapest solution. For Dijkstra's algorithm the cost of moving a tile was the number on the tile while for BFS and DFS it was 1 for moving a tile. For Dijkstra's algorithm the state of each step in the solution can be described as a weighted graph, in which each state is represented by a vertex and each move by an edge. For BFS and DFS the state of each step in the solution can be described as an unweighted graph, in which each state is represented by a vertex and each move by an edge. Each of these algorithms were implemented to find the shortest path between the initial and final state of the provided graphs.

Results

We used two sample problems that these different algorithms were tested on. All of them successfully found the cheapest solution which was the same solution. Based on these tests, the following results were computed. The following test cases were used to test the performance of different algorithms. The test cases are shown in Table 1: The Initial and Goal States of the Test Cases.

Initial State of Test 1	Goal State of Test 1	Initial State of Test 2	Goal State of Test 2
1 3 4 8 0 2 7 6 5	1 2 3 8 0 4 7 6 5	1 3 4 8 0 6 7 5 2	1 2 3 8 0 4 7 6 5

Steps and Cost of using the **Dijkstra's** Algorithm shown in Table 2:

Table 2: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Goal
1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5
Cost of step:	2	4	3	2	11

Steps and Cost of using **BFS** Algorithm shown in Table 3:

Table 3: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Goal
1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5
Cost of step:	1	1	1	1	4

Steps and Cost of using the **DFS** Algorithm shown in Table 4:

Table 4: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Goal
1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5
Cost of step:	1	1	1	1	4

Steps and Cost of using the **Dijkstra's** Algorithm shown in Table 5:

Table 5: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Goal
1 3 4 8 0 6 7 5 2	1 3 4 8 6 0 7 5 2	1 3 4 8 6 2 7 5 0	1 3 4 8 6 2 7 0 5	1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5
Cost of step:	6	2	5	6	2	4	3	2	30

Steps and Cost of using **BFS** Algorithm shown in Table 6:

Table 6: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Goal
1 3 4 8 0 6 7 5 2	1 3 4 8 6 0 7 5 2	1 3 4 8 6 2 7 5 0	1 3 4 8 6 2 7 0 5	1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5
Cost of step:	1	1	1	1	1	1	1	1	8

Steps and Cost of using the **DFS** Algorithm shown in Table 7:

Table 7: Matrix Transformation Steps to calculate the cheapest solution.

Initial State	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Goal
1 3 4 8 0 6 7 5 2	1 3 4 8 6 0 7 5 2	1 3 4 8 6 2 7 5 0	1 3 4 8 6 2 7 0 5	1 3 4 8 0 2 7 6 5	1 3 4 8 2 0 7 6 5	1 3 0 8 2 4 7 6 5	1 0 3 8 2 4 7 6 5	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5

Cost of step:	1	1	1	1	1	1	1	1	8

Discussion:

Dijkstra's Algorithm is used to find the shortest path between two vertices of a weighted graph. It uses a greedy approach to find the cheapest solution. Here since the graph created to represent the states of the puzzle expands infinitely, we only create a vertex once we visit it, rather than creating and storing all vertices in a data container. Here the weight function for calculating the cost of making a move(visiting adjacent vertex) is the number on the moved tile. Similarly, for BFS and DFS each vertex is created once it is visited.

Here each implementation reaches the correct solution using a different approach. Dijkstra's algorithm chooses vertices in a greedy way and keeps track of the distance from the initial vertex. It performs relaxation of the distance each time a better path is found. And once it visits a vertex no further relaxation is required for that particular vertex. Thus, we stop when we reach the goal state. This makes Dijkstra's algorithm the most efficient among the three implementations.

Breadth First Search on unweighted graphs, can be used to trace back the cheapest path for each vertex. The implementation keeps track of the parent vertex or the vertex from which a new vertex is visited. Thus, we can trace back the cheapest path once we reach the solution vertex. Similar to Dijkstra's algorithm, we stop once we reach the solution vertex. Since we are not using a greedy approach, BFS generally takes more time than Dijkstra's algorithm, since it visits all the vertices in a level and then moves on to the next level.

Depth First Search on graphs can be modified to find the cheapest path in an unweighted graph. The DFS tree has back edges, which can be used to find the

cheapest solution. Here each vertex stores the depth of the vertex, which is the path from the initial state, using only back edges. Since DFS makes recursive calls to the adjacent vertices visiting the deepest vertex before moving on to a vertex in the same level, we have to restrict the depth the algorithm will visit as our solution state graph expands infinitely. Thus, for a given depth the DFS will visit all the vertices of the state graph of the given depth. Unlike Dijkstra's algorithm and BFS, DFS does not guarantee the cheapest solution the first time it visits the solution vertex. Thus here, we will visit the whole graph of a given depth, and will update the cheapest solution every time a better path is found. Since DFS visits all the vertices of the graph for a given depth, it runs considerably slower than both Dijkstra's algorithm and BFS.

Conclusion:

For any given problem, we can use different algorithmic approaches to find the solution to the problem. Based on the speed and time requirements, we can select on implementation over other algorithms. Here DFS uses Brute-Force to calculate the shortest path which guarantees the correct answer but is not the most practical option given space constraints. BFS provides a better solution both in terms of space and time complexity, but the worst-case scenario is the same as DFS's. Dijkstra's Algorithm uses a greedy approach to solve the problem more efficiently. Generally greedy algorithms are hard to prove for correctness, but Dijkstra's algorithm does return the correct solution every time. Here other implementations can be also created using Heuristics and optimal strategies.