Implementation of Different Algorithms on 8-Puzzle

This article tries to explain and show the differences between different techniques in tree search while solving 8-Puzzle. In this project, techniques are implemented and tested in Python.

There are 4 different algorithms: BFS(Breadth-First Search), DFS(Depth-First Search), UCS(Unit Cost Search), A\* search. BFS is searching for the shallowest node while DFS is looking for the deepest node by expanding most-left node always. UCS is a bit more evolved, looking for minimum cost node. A\* is the newer technique for tree search that measures the manhattan-distance, which is total of cost and the estimated distance between result and current node.

All of the algorithms are implemented to a one-dimension array. To avoid the infinity tree, a dictionary(list of visited nodes) is hold and all of the nodes are checked if visited or not. Visited nodes have not been extended. To simulate BFS and DFS algorithms, just an array is implemented and managed add and delete operations based on FIFO(queue)(for BFS) and LIFO(stack)(for DFS). In UCS and A\*, priority queue is hold. In each step, the priority queue is pulling the node with minimum value of cost(for UCS) and manhattan-distance(for A\*) from array and evaluating the node.

Test is made by checking 2 cases. Which are:

Case 1:

0 1 2 8 7 6

3 4 5 to 5 4 3

6 7 8 2 1 0

Case 2:

1 2 3 2 8 1

8 0 4 to 0 4 3

7 6 5 7 6 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case 1 | BFS | DFS | UCS | A\* |
| Num. Of visited Nodes | 178224 | 89463 | 179829 | 657 |
| Depth of Graph | 28 | 84098 | 28 | 28 |
| Runtime(sec) | 18.24 | 17.43 | 20.31 | 0.08 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case 2 | BFS | DFS | UCS | A\* |
| Num. Of visited Nodes | 385 | 156037 | 468 | 25 |
| Depth of Graph | 9 | 102589 | 9 | 9 |
| Runtime(sec) | 0.02 | 51.09 | 0.03 | 0.0(TLTM) |

TLTM=Too Little to Measure

Hereby, you can see that DFS is better in some cases from BFS and BFS is better in some cases from DFS. BFS and UCS’s values are close to each other. UCS is working like BFS in this case, because the cost of the nodes are step values of these nodes. And we can see A\* search is highly effective based on other algorithms because it doesn’t check all nodes until it finds a solution, just trying to take the shortest way to the solution by checking estimated distance from result.

Based on results, it can be clearly understood that using a more efficient algorithm saves a lot of time. It is not so crucial in cases like 8-puzzle, but timing and reaction is so important in continous, multi agent environments. These algorithms let humanity make AI’s which can respond to instant changes.

Additional note: Implemented DFS is working less efficient in the manner of time. I saw some implementations which making operations at the end of the array and works in a very short time, but I wanted to stick to the concept of stack, so that I used insert() method to make operations at the top of the stack.