CS451 Assignment 1 Report

Can Yılankıran S011483

In this assignment, the goal was successfully implementing some search techniques for N-Puzzle problem. Used search techniques to solve N-Puzzle problem are as follows: Breadth first search, Depth first search, Uniform cost search and A* search (including heuristic). Also, N is equal to 8 in this assignment. I had some issues while implementing Uniform cost search, my output does not really match to example scenarios' output on the assignment sheet.

Algorithms

Firstly, I want to start with, Breadth first search because I am familiar with this search method since I took CS201. Shortly, I can explain this search method like, the goal is traversing the tree from left to right. So, we can consider the tree horizontally divided into the layers and we start with root layer then continue with second layer.

Let's continue with Depth first search. DFS is exactly opposite to the BFS. The goal is start with a node (root node) and travers as long as possible in a vertical way. Then it will backtrack and continue with another possible vertical way.

Thirdly, Uniform cost search uses a distancing mechanism to find the shortest path to the goal node from the start node (root node). For each path from the start node to the goal node, a path cost will be calculated according to the nodes, and the lowest one will be the solution.

Lastly, in A* search is like the Uniform cost search, but it includes some heuristic function for efficiency. We can describe cost structure of A* as follows:

$$f(n) = g(n) + h(n)$$

In this equation $g(n)$ is the path cost from start node (root node) to the node n. h(n) is the heuristic function and in this assignment, I was expected to use Manhattan Distance as heuristic function. It calculates the cheapest path from node n to goal node. So, A* star algorithm chooses the minimum value coming from this equation.

Implementation Details

Firstly, I want to mention that carefully read the assignment sheet and I know that I wasn't supposed to add new methods while completing the code. But while writing all algorithms I needed to check that the new node is already visited or not. I tried the way that:

```
for child in self.graph.reveal_neighbors(state):
    if child not in self.visited:

    ……..
```

But I get compile errors. So, to manage this error, I added a check_value_exists method in all search algorithms. It takes 2 parameters, first one is the dictionary (self.visited) and the second one is the node(for example: child).

I used these 2 lines of code to check that the current state is equal to target state or not.

```
if state.is_equal(target):
  return True, self.counter, state.step
```

Let's continue with the implementation details of BFS. In BFS the list() object must act like a queue in order to successfully implement the logic of BFS(FIFO). So, I used pop() function to get the last element in the list which is the first node got in to the list. Then, I used list.insert(0,child) function to add the element to the beginning of the list. By these two methods I obtained the logic of BFS(FIFO).

Secondly, while implementing the DPS I used the opposite way of BFS. Because DPS logic is LIFO So, I used pop() function to get the last element in the list which is the first node got in to the list. Then, I used list.append(child) function to add the element the to end of list. By doing that, I obtained a list acting like a stack.

In UCS, I had some issues and I does not get the true output of visited nodes. Priority queue helped me to manage the queue according to a priority (in this case it is path cost). I tried the way that adding child.step + state.step to calculate the total cost and push them into to the Priority queue. But surely, I miss some really important point about UCS.

At last, in A* search with the power of Priority queue I was able to implement this search algorithm. For every child, node I added the cost from start node to current node which is g(n) with Manhattan Distance (the cheapest path from node n to goal node). Then I pushed those child nodes to the Priority queue with their costs.

Results

As expected, the most efficient algorithm, considering the count of visited nodes an elapsed time, is A* algorithm. The worst one is the DFS algorithm according to the visited nodes and elapsed time. But those results can change when the target state is closer or far to the start(root) node. For example, if the goal state is very far from the root(start) node DPS algorithm will perform better than the BFS. Vice versa, BFS will perform better than DFS when the target state is close to the root(start) node.

Conclusion

I have learned how to implement those search algorithms in Python and the important things need to be considered while implementation procedure. Such as, list() object acting like a stack or queue while implementing DFS and BFS, in order to obtain FIFO and LIFO mechanism. I can really say something about the UCS even though I found a match when I run the code, I am pretty sure that I missed a crucial detail about UCS. For A* algorithm I learned what is a Manhattan Distance and how powerful using a heuristic while implementing such search algorithms.