Sadhika Huria
CMPT310
Assignment 1

Question:

Algorithm part:
This part worth 10 points. Answer the following question in a text file named WeightedDFS.txt:
As was explained in class, the vanilla DFS follows blindly the leftmost branch as it goes down the search tree, counting each step as one unit of cost. Suppose the cost per action is not one any more but is a positive number. How would you modify your DFS implementation/algorithm to take advantage of the variations of the costs for actions at each node to find a shortest path (lowest cost) for the DFS solution. Remember the solution would still be considered DFS, but a more optimum solution than the vanilla version. Provide your answer in the form of pseudo-code, similar to the BestFirstSearch algorithm or BFS algorithm given in class and in the search ppt. Your pseudo-code should be detailed enough that is implementable without further explanation needed by a software engineer.

Solution:

Modify DFS to use a priority stack instead. This is where nodes are pushed based on their path cost. By doing this, we can still prioritize depth-first exploration and avoid taking unnecessarily expensive paths.

Pseudocode:

```
function WeightedDFS(problem):
    frontier <- priority stack (min-heap ordered by path cost)
    frontier.push(Node(problem.initial), priority=0)
    explored <- empty dictionary (state -> min cost encountered)

    while frontier is not empty:
        node <- frontier.pop()

        if problem.goal_test(node.state):
            return solution(node)

        if node.state in explored and node.path_cost >= explored[node.state]:
            continue  # Skip if we've already seen a better path to this state

        explored[node.state] = node.path_cost  # Store the best cost encountered

        for each action in problem.actions(node.state):
            child <- child_node(problem, node, action)
            frontier.push(child, priority=child.path_cost)

    return failure
```

Explanation:

1. Uses a priority stack – Instead of a regular stack, we maintain a min-heap where nodes with lower cost are explored first.
2. Keeps track of visited states with cost – We store the lowest cost encountered for each state to avoid revisiting worse paths.
3. Maintains DFS behavior – Since we still explore nodes depth-first, this remains a DFS variant rather than turning into UCS.
4. Optimizes path selection – Among multiple paths, it prioritizes the one with the lower cost while still expanding deeper first.