COEN 166 Artificial Intelligence

Lab 3

Divya Syal: 1594122, Marianne Yamazaki Dorr: 1606975: , Suvass Ravala: 1593088

Each did 33%

Function 1 getSolutionPath()

```
def getSolutionPath(currentNode, startNode):
    # traverse up from goal node to initial node
    solutionPath = [];

    while (currentNode != startNode):
        solutionPath.append(currentNode.action)
        currentNode = currentNode.parent

    print(solutionPath)
    print(list(reversed(solutionPath)))
    return list(reversed(solutionPath))
```

Comment:

In this function, it takes in the start and end node and reverses it to get the correct path as bfs and A* lead to a reversed path. This function works by running a while loop and appending the current node then increments till it is not the endNode.

Function 2 Breadth-First Search

# paste your code:

```
def breadthFirstSearch(problem):
    """
    Search the shallowest nodes in the search tree first.

    You are not required to implement this, but you may find it useful for Q5.
    """
    "*** YOUR CODE HERE ***"

    visitedStates = []
    startState = problem.getStartState()
    startNode = Node(startState, None, None, 0)
    # queue should contain nodes, not state
    frontier = util.Queue()
    frontier.push(startNode)


   # may enter infinite loop is you do while (not problem.isGoal()):"
    while (not frontier.isEmpty()):
        currentNode = frontier.pop()

        "first check is node.state has been visited"
        if (currentNode.state not in visitedStates):
            # add currentNode to visitedStates
            visitedStates.append(currentNode.state)

            # first check if at goal node
            if(problem.goalTest(currentNode.state)):
                # traverse from goal node to initial node and reverse
                return getSolutionPath(currentNode, startNode)

            # expand this node, generate successors + enter frontier queue
            actionsList = problem.getActions(currentNode.state)

            for action in actionsList:
                nextState = problem.getResult(currentNode.state, action)
                nextCost = problem.getCost(currentNode.state, action) + currentNode.path_cost
                nextNode = Node(nextState, currentNode, action, nextCost)
                frontier.push(nextNode)
```

Comment:

In this function we first decided to create a visitedStates array and create a frontier Queue. After that we got the starting node and the state of it. Then we made sure the frontier was not empty as just checking the goal state would lead to an infinite loop. Then we did a while loop that checked if it was visited or not, if it was not then we would check if it was the goal node, if it was then we would reverse the path. If it was not the goal, then we would run through all the actions and store the next state, cost, and node then push it onto the frontier queue.

In order to code this, we had to create a separate function called getSolutionPath() which would reverse the path as the nodes would be popped out in reverse.

Function 2 aStarSearch

# paste your code (any other function that you defined or modified): def function_name(arguments):

```
    visitedStates = []
    startState = problem.getStartState()
    startNode = Node(startState, None, None, 0)
    Frontier = util.PriorityQueue()
    Frontier.push(startNode, 0)

    while(not Frontier.isEmpty()):
        currentNode = Frontier.pop()

        #Check if we are at the goal node. If so, return the solution path
        if (problem.goalTest(currentNode.state)):
            return getSolutionPath(currentNode, startNode)

        actionsList = problem.getActions(currentNode.state)

        for action in actionsList:
            nextState = problem.getResult(currentNode.state, action)
            nextCost = (problem.getCostOfActions(getSolutionPath(currentNode,
startNode))
                        + heuristic(nextState, problem))
            nextNode = Node(nextState, currentNode, action, nextCost)
            if (nextState not in visitedStates or
problem.getCostOfActions(getSolutionPath(nextNode, startNode)) > nextCost):
                visitedStates.append(nextState)
                Frontier.push(nextNode, nextCost)
```

Comment:

In this search function we started off with a similar approach as the BFS as we had a visited array and a frontier queue. Then we made sure the frontier is not empty and checked if the start node was the goal node. If it was, then we would return the reversed path. If not then we would create a list of actions and for each action, calculate the next state, cost , and node. We would also append the state to the visited array and push it to the frontier queue. The way we calculated the cost is different from BFS as we had to take into account the heuristic.