COEN 166 Artificial Intelligence

Lab Assignment #3: Search I

Gagan Gupta **SCU#00001478479**

Problem 1 Breadth-First Search:

Function 1:

```
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.
    """
    from util import Queue
    x = Queue()
    visited = []
    start = problem.getStartState()
    x.push((start,[]))

    while not (x.isEmpty()):
        popped = x.pop()
        visited.append(popped[0])
        if problem.goalTest(popped[0]):
            return popped[1]
        futureActions = problem.getActions(popped[0])
        if len(futureActions)!=0:
            for action in futureActions:
                tempState = problem.getResult(popped[0], action)
                tempPath = popped[1] + [action]
                if tempState not in visited and tempState not in (state[0] for state in x.list):
```

Comment: BFS for a Pacman Maze in which we use the queue from util. The item we push onto the queue contains the current state and the actions taken to get there. We first push the starting state and go into the while loop where we loop until the queue is empty or the goal state is reached. The first action is to pop the bottom element (FIFO) from the queue and adds that state to already visited. From that state we also get a list of the possible actions to take. If there are actions to take, we go through each action, find the state that results from that action, and add that action to the path popped from the queue. If that state hasn't been visited and isn't already in the queue, we push the resulting state and concatenated path onto the queue. When we hit the goal state in the search, we return action list that was popped from the goal state. Pacman then follows this list of actions to reach the goal state.

Problem 2 Depth-First Search:

Function 1:

```
def depthFirstSearch(problem):

    from util import Stack
    x = Stack()
    visited = []
    start = problem.getStartState()
    x.push((start,[]))

    while not (x.isEmpty()):
        popped = x.pop()
        visited.append(popped[0])
        if problem.goalTest(popped[0]):
            return popped[1]
        futureActions = problem.getActions(popped[0])
        if len(futureActions)!=0:
            for action in futureActions:
                tempState = problem.getResult(popped[0], action)
                tempPath = popped[1] + [action]
                if tempState not in visited and tempState not in (state[0] for state in x.list):
                    x.push((tempState,tempPath))
```

Comment: DFS for a Pacman Maze in which we use the stack from util. The item we push onto the stack contains the current state and the actions taken to get there. We first push the starting state and go into the while loop where we loop until the stack is empty or the goal state is reached. The first action is to pop the top element (LIFO) from the stack and adds that state to already visited. From that state we also get a list of the possible actions to take. If there are actions to take, we go through each action, find the state that results from that action, and add that action to the path popped from the stack. If that state hasn't been visited and isn't already in the stack, we push the resulting state and concatenated path onto the stack. When we hit the goal state in the search, we return action list that was popped from the goal state. Pacman then follows this list of actions to reach the goal state.