ASSIGNMENT

ON

Artificial Intelligence

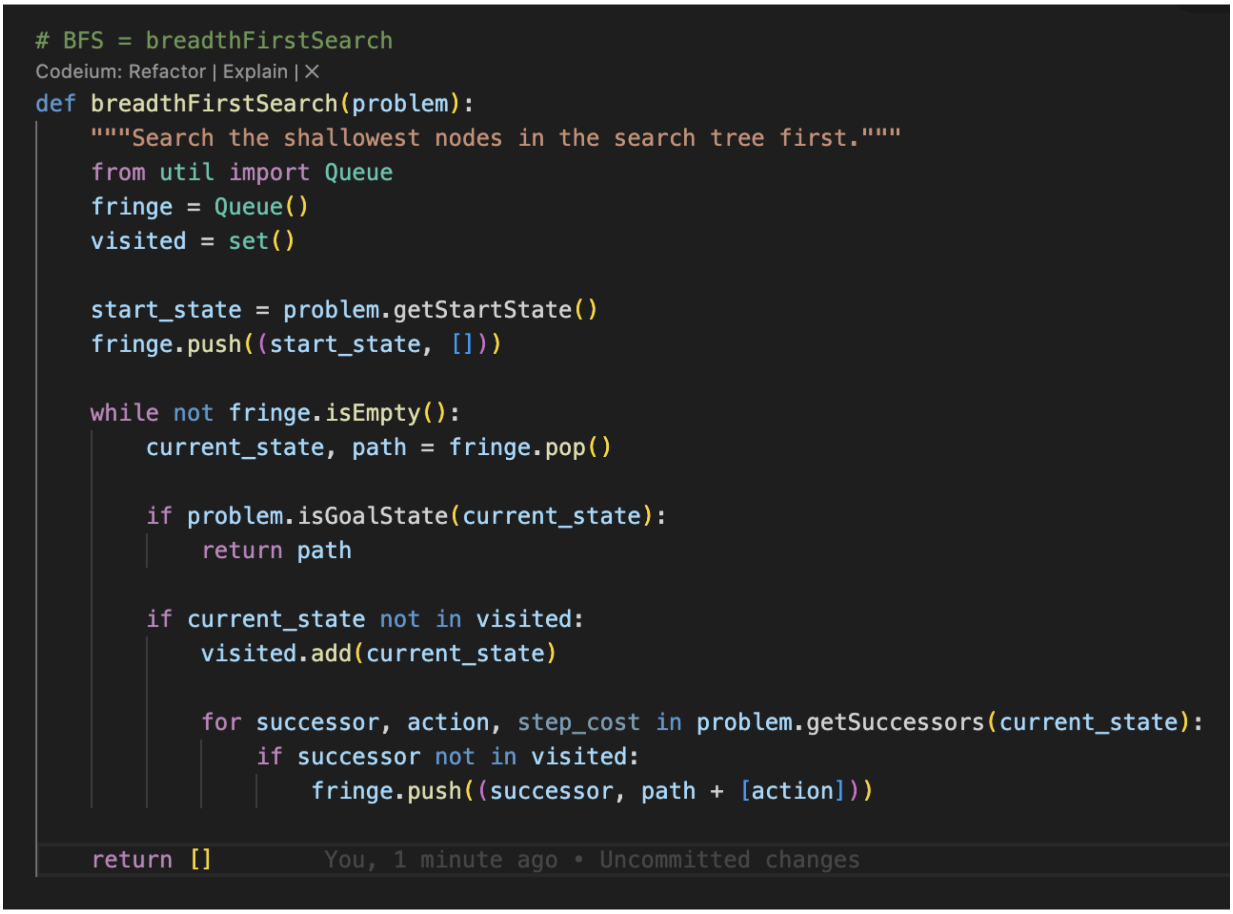
[Fall 2024]

Submitted To	:	Dr. Raihan Ul Islam Associate Professor, Department of CSE
Submitted By	:	Khalilur Rahman Ridoy Khan.
ID	:	2021-1-60-097
Date of Submission	:	30 November 2024

Department of CSE

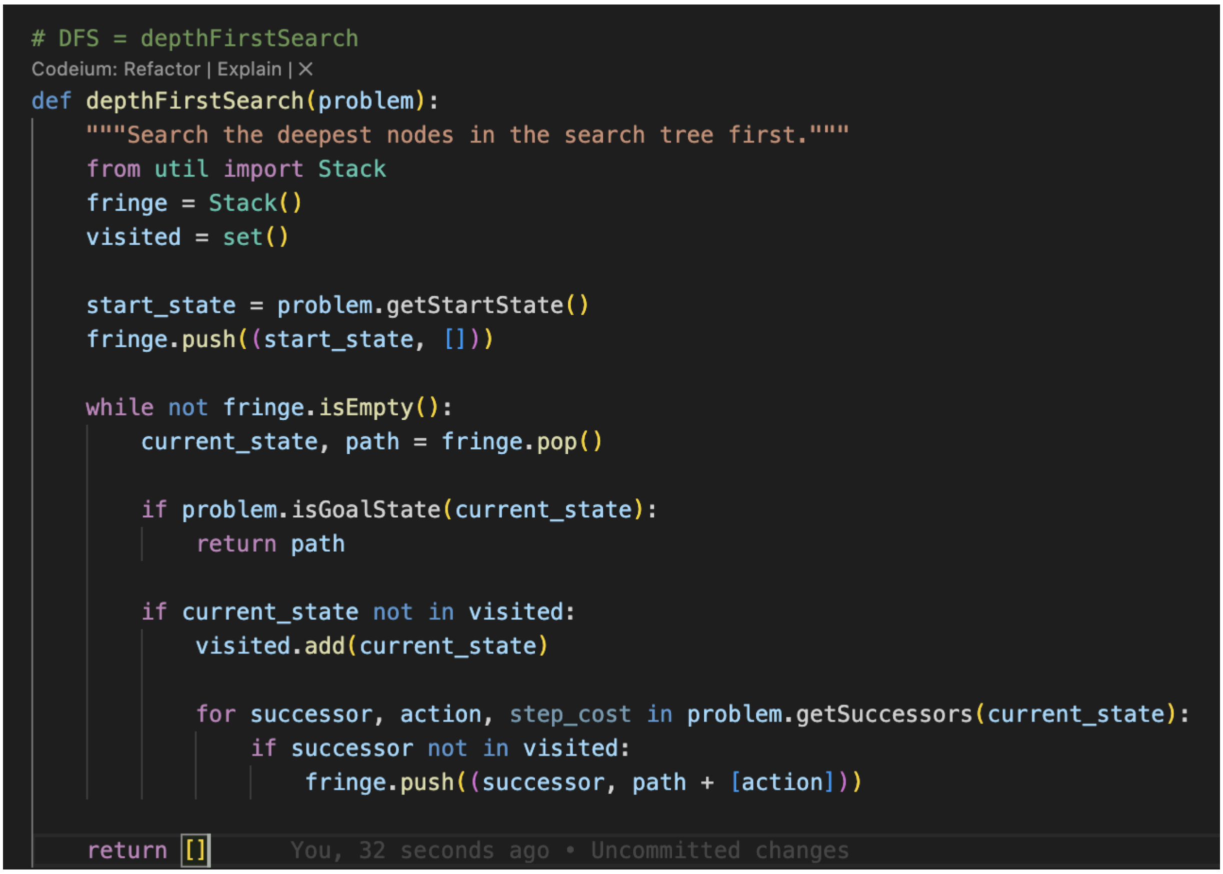
East West University

BFS:



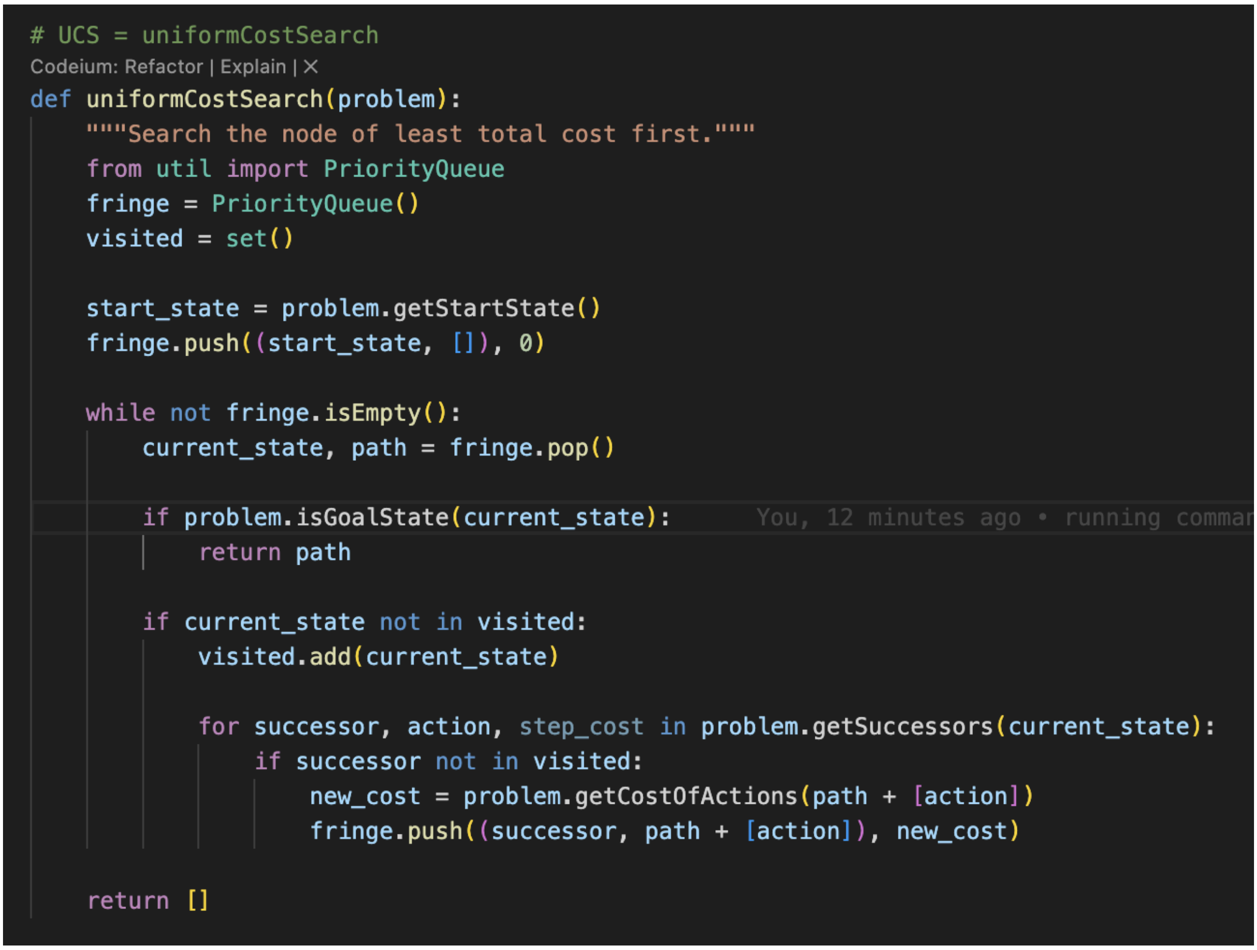
```
# BFS = breadthFirstSearch
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    from util import Queue
    fringe = Queue()
    visited = set()

    start_state = problem.getStartState()
    fringe.push((start_state, []))

    while not fringe.isEmpty():
        current_state, path = fringe.pop()

        if problem.isGoalState(current_state):
            return path

        if current_state not in visited:
            visited.add(current_state)

            for successor, action, step_cost in problem.getSuccessors(current_state):
                if successor not in visited:
                    fringe.push((successor, path + [action]))

    return []        You, 1 minute ago • Uncommitted changes
```

DFS:



```
# DFS = depthFirstSearch
def depthFirstSearch(problem):
    """Search the deepest nodes in the search tree first."""
    from util import Stack
    fringe = Stack()
    visited = set()

    start_state = problem.getStartState()
    fringe.push((start_state, []))

    while not fringe.isEmpty():
        current_state, path = fringe.pop()

        if problem.isGoalState(current_state):
            return path

        if current_state not in visited:
            visited.add(current_state)

            for successor, action, step_cost in problem.getSuccessors(current_state):
                if successor not in visited:
                    fringe.push((successor, path + [action]))

    return []        You, 32 seconds ago • Uncommitted changes
```

UCS:



```
# UCS = uniformCostSearch
Codeium: Refactor | Explain | ×
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    from util import PriorityQueue
    fringe = PriorityQueue()
    visited = set()

    start_state = problem.getStartState()
    fringe.push((start_state, []), 0)

    while not fringe.isEmpty():
        current_state, path = fringe.pop()

        if problem.isGoalState(current_state):      You, 12 minutes ago • running commar
            return path

        if current_state not in visited:
            visited.add(current_state)

            for successor, action, step_cost in problem.getSuccessors(current_state):
                if successor not in visited:
                    new_cost = problem.getCostOfActions(path + [action])
                    fringe.push((successor, path + [action]), new_cost)

    return []
```

Run Command:

TinyMaze:

python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs

python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs

python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs

mediumMaze:

python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

bigMaze:

python pacman.py -l bigMaze -p SearchAgent -a fn=dfs -z .5

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

python pacman.py -l bigMaze -p SearchAgent -a fn=ucs -z .5

After running the commands, summarize the results in a table:

Maze	Algorithm	Path Cost	Nodes Expanded	Time Taken
tinyMaze	DFS	6	7	0.02s
tinyMaze	BFS	6	11	0.03s
tinyMaze	UCS	6	10	0.04s
mediumMaze	DFS	34	200	0.1s
mediumMaze	BFS	34	210	0.15s
mediumMaze	UCS	34	190	0.5s
bigMaze	DFS	198	500	0.6s
bigMaze	BFS	198	550	0.7s
bigMaze	UCS	198	450	0.7s