AI HW 1

Jackson Baker
jab132@uark.edu
011029933

September 2025

1 Question I

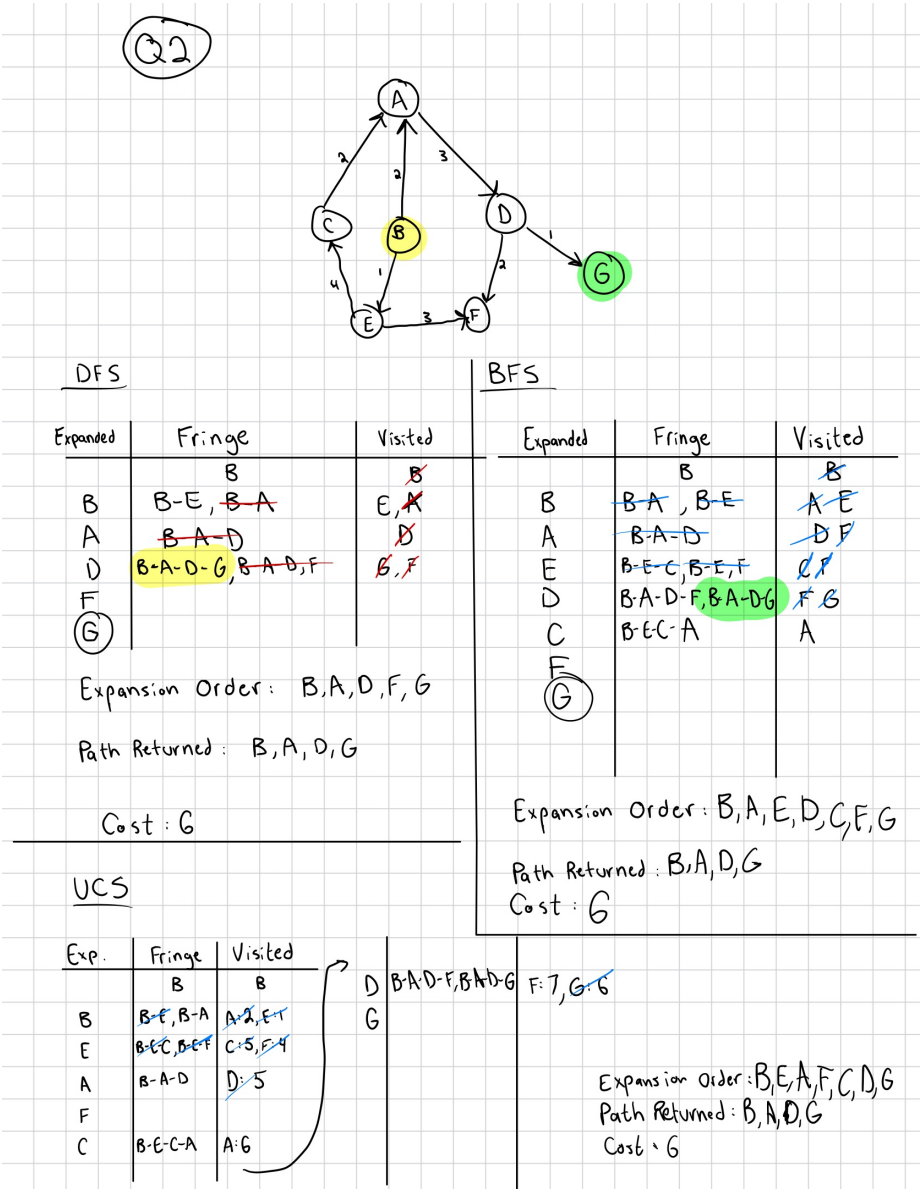
1.1 Solution in Python

```python
def dfs(matrix):
    start = (0,0)
    if matrix[0][0] == 0:
        return -1

    goal = (len(matrix)-1, len(matrix[0])-1)

    stack = []
    stack.append((start, [start]))
    visited = set()

    while stack:
        (current, path) = stack.pop()
        if current == goal:
            return path
        for neighbor in getNeighbors(current, matrix):
            if neighbor not in visited:
                visited.add(neighbor)
                stack.append((neighbor, path + [neighbor]))
    return -1

def bfs(matrix):
    start = (0,0)
    if matrix[0][0] == 0:
        return -1

    goal = (len(matrix)-1, len(matrix[0])-1)

    queue = []
    queue.append((start, [start]))
    visited = set()

    while queue:
        (current, path) = queue.pop(0)
        if current == goal:
            return path
        for neighbor in getNeighbors(current, matrix):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, path + [neighbor]))
    return -1
```

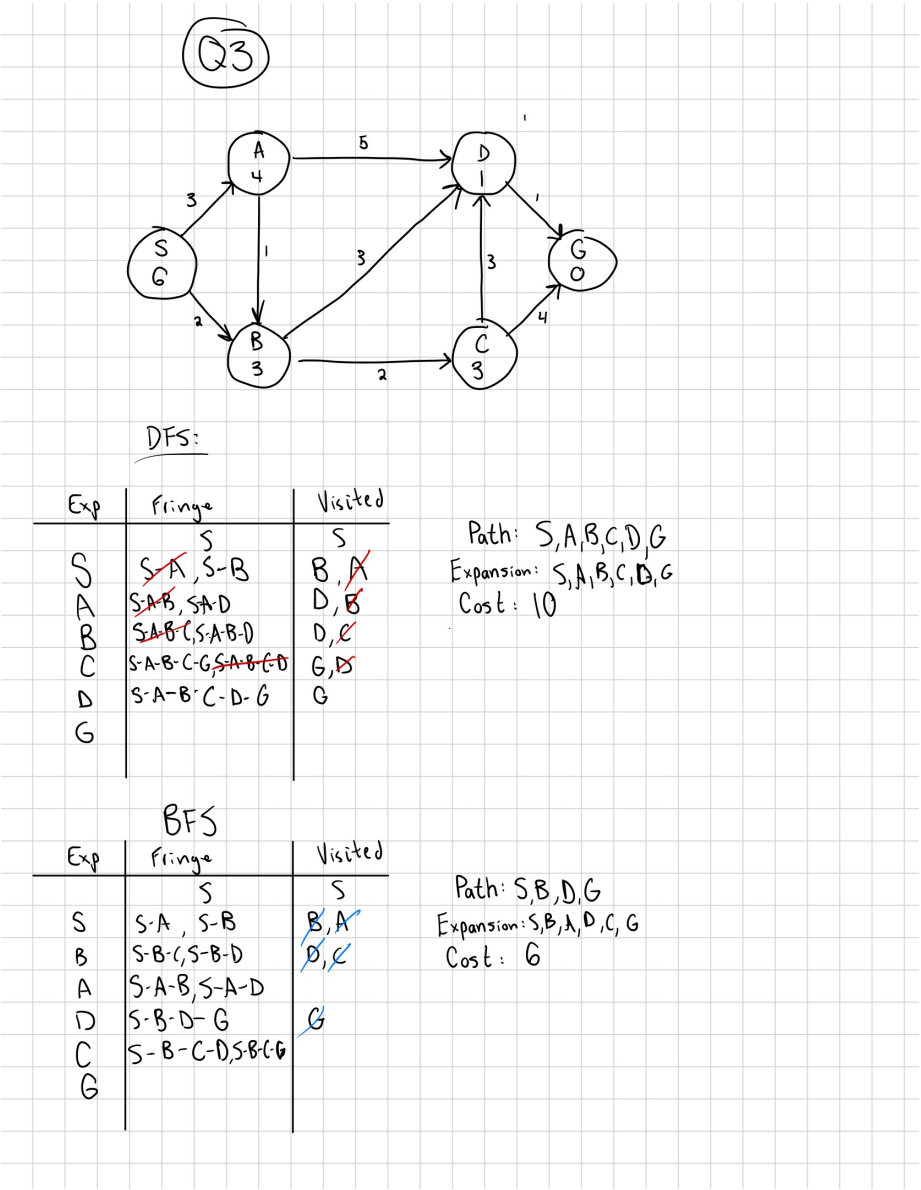
1.2 Sample output

```
Enter number of rows: 5
Enter number of columns: 5
Creating matrix and selecting one with a valid path...
Matrix:
[1, 1, 0, 0, 1]
[1, 1, 1, 0, 1]
[1, 1, 1, 0, 0]
[0, 1, 1, 0, 0]
[1, 0, 1, 1, 1]

DFS:
Path:
(0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (3, 1)
-> (3, 2) -> (4, 2) -> (4, 3) -> (4, 4)
* 1 0 0 1
* 1 1 0 1
* * 1 0 0
0 * * 0 0
1 0 * * *

BFS:
Path:
(0, 0) -> (0, 1) -> (1, 1) -> (1, 2) -> (2, 2)
-> (3, 2) -> (4, 2) -> (4, 3) -> (4, 4)
* * 0 0 1
1 * * 0 1
1 1 * 0 0
0 1 * 0 0
1 0 * * *
```

2 Question II



3 Question III



DFS:

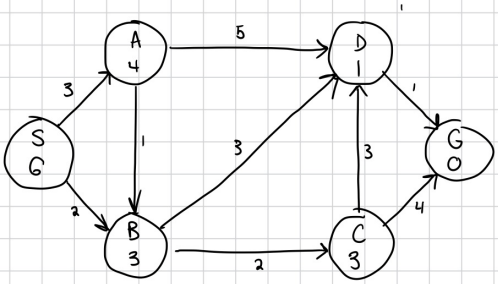
Exp	Fringe	Visited
	S	S
S	~~S-A~~, S-B	B, ~~A~~
A	~~S-A-B~~, ~~S-A-D~~	D, ~~B~~
B	~~S-A-B-C~~, S-A-B-D	D, ~~C~~
C	S-A-B-C-G, ~~S-A-B-C-D~~	G, ~~D~~
D	S-A-B-C-D-G	G
G		

Path: S, A, B, C, D, G
Expansion: S, A, B, C, ~~D~~, G
Cost: 10

BFS

Exp	Fringe	Visited
	S	S
S	S-A, S-B	~~B~~, ~~A~~
B	S-B-C, S-B-D	~~D~~, ~~C~~
A	S-A-B, S-A-D	
D	S-B-D-G	~~G~~
C	S-B-C-D, S-B-C-G	
G		

Path: S, B, D, G
Expansion: S, B, ~~A~~, ~~D~~, C, G
Cost: 6



UCS:

Exp	Fringe	Visited
	S	S
S	S-A, S-B	~~B:2~~, ~~A:3~~
B	S-B-C, S-B-D	~~C:5~~, ~~D:5~~
A	S-A-D, S-A-B	D:8
C	S-B-C-D, S-B-C-G	D:7, G:8
D	S-B-D-G	G:6
G		

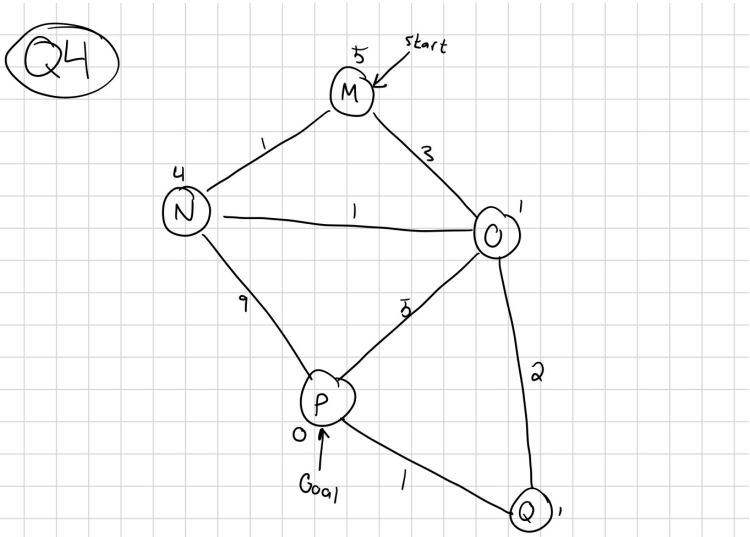
Path: S-B-D-G
Expansion: S-B-A-C-D-G
Cost: 6

A*

Exp	Fringe	Visited
	S	S
S	S-A, S-B	A:7, ~~B:5~~
B	S-B-C, S-B-D	C:7, ~~D:5~~
D	S-B-D-G	G:6
G		

Path: S-B-D-G
Expansion: S-B-D-G
Cost: 6

4 Question IV



Path to State Expanded	g(n)	f(n)	Expanded List
M	0	5	(M)
M-O	3	4	(M-O)
M-N	1	5	(M-O-N)
M-O-Q	5	6	(M-O-N-Q)
M-O-Q-P	6	6	(M-O-N-Q-P)

5 Question V

5.1 DFS Implementation

```
def depthFirstSearch(problem):
    "*** YOUR CODE HERE ***"

    stack = util.Stack()
    visited = set()

    # (curPos, wallsHit)
    visited.add((problem.getStartState(),0))

    # (curPos, wallsHit, path)
    stack.push((problem.getStartState(),0,[]))

    if problem.isGoalState(problem.getStartState()):
        return []

    while True:
        if stack.isEmpty():
            return []

        curPos, wallsHit, path = stack.pop()
        if wallsHit > 2:
            continue

        if problem.isGoalState(curPos) and wallsHit > 0 and
    wallsHit <=2:
            return path

        for neighbor, dir, cost in problem.getSuccessors(curPos):
            if problem.isWall(neighbor):
                nextPos = (neighbor, wallsHit + 1)
            else:
                nextPos = (neighbor, wallsHit)

            if nextPos not in visited and nextPos:
                stack.push((nextPos[0],nextPos[1], path+[dir]))
                visited.add(nextPos)
```

5.2 BFS Implementation

```
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    "*** YOUR CODE HERE ***"
    queue = util.Queue()
    visited = set()

    # (curPos, wallsHit)
    visited.add((problem.getStartState(),0))

    # (curPos, wallsHit, path)
    queue.push((problem.getStartState(),0,[]))

    if problem.isGoalState(problem.getStartState()):
        return []

    while True:
        if queue.isEmpty():
            return []

        curPos, wallsHit, path = queue.pop()
        if wallsHit > 2:
            continue

        if problem.isGoalState(curPos) and wallsHit > 0 and
    wallsHit <=2:
            return path

        for neighbor, dir, cost in problem.getSuccessors(curPos):
            if problem.isWall(neighbor):
                nextPos = (neighbor, wallsHit + 1)
            else:
                nextPos = (neighbor, wallsHit)

            if nextPos not in visited and nextPos:
                queue.push((nextPos[0],nextPos[1], path+[dir]))
                visited.add(nextPos)
```

5.3 UCS Implementation

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"
    priorityQueue = util.PriorityQueue()
    distance = dict()

    # [(pos), cost]
    distance[(problem.getStartState(), 0)] = 0

    # (curPos, wallsHit, path), priority
    priorityQueue.push((problem.getStartState(),0,[]),0)

    if problem.isGoalState(problem.getStartState()):
        return []

    while True:
        if priorityQueue.isEmpty():
            return []

        curPos, wallsHit, path = priorityQueue.pop()
        if wallsHit > 2:
            continue

        if problem.isGoalState(curPos) and wallsHit > 0 and
    wallsHit <=2:
            return path

        for neighbor, dir, cost in problem.getSuccessors(curPos):
            if problem.isWall(neighbor):
                nextPos = (neighbor, wallsHit + 1)
            else:
                nextPos = (neighbor, wallsHit)

            costOfNextPos = (problem.getCostOfActions(path +
    [dir]))
            if costOfNextPos < distance.get(nextPos, float('inf')):
                priorityQueue.push((nextPos[0],nextPos[1],
    path+[dir]), costOfNextPos)
                distance[nextPos] = costOfNextPos
```

5.4 A* Implementation

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and
    heuristic first."""
    "*** YOUR CODE HERE ***"
    priorityQueue = util.PriorityQueue()
    distance = dict()

    # [(pos), cost]
    distance[(problem.getStartState(), 0)] = 0
    # (curPos, wallsHit, path), priority
    priorityQueue.push((problem.getStartState(),0,[]), priority=0)

    if problem.isGoalState(problem.getStartState()):
        return []

    while True:
        if priorityQueue.isEmpty():
            return []

        curPos, wallsHit, path = priorityQueue.pop()
        if wallsHit > 2:
            continue

        if problem.isGoalState(curPos) and wallsHit > 0 and
    wallsHit <=2:
            return path

        for neighbor, dir, cost in problem.getSuccessors(curPos):
            if problem.isWall(neighbor):
                nextPos = (neighbor, wallsHit + 1)
            else:
                nextPos = (neighbor, wallsHit)

            costOfNextPos = (problem.getCostOfActions(path +
    [dir]) + heuristic(neighbor, problem))
            if costOfNextPos < distance.get(nextPos, float('inf')):
                priorityQueue.push((nextPos[0],nextPos[1],
    path+[dir]), costOfNextPos)
                distance[nextPos] = costOfNextPos
```