

AI HW 1

Jackson Baker
jab132@uark.edu
011029933

September 2025

1 Question I

1.1 Solution in Python

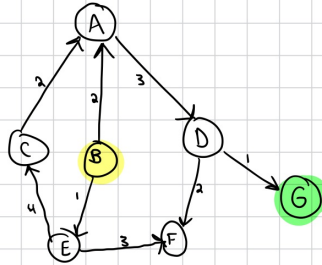
```
1 def dfs(matrix):
2     start = (0,0)
3     if matrix[0][0] == 0:
4         return -1
5
6     goal = (len(matrix)-1, len(matrix[0])-1)
7
8     stack = []
9     stack.append((start, [start]))
10    visited = set()
11
12    while stack:
13        (current, path) = stack.pop()
14        if current == goal:
15            return path
16        for neighbor in getNeighbors(current, matrix):
17            if neighbor not in visited:
18                visited.add(neighbor)
19                stack.append((neighbor, path + [neighbor]))
20    return -1
21
22 def bfs(matrix):
23     start = (0,0)
24     if matrix[0][0] == 0:
25         return -1
26
27     goal = (len(matrix)-1, len(matrix[0])-1)
28
29     queue = []
30     queue.append((start, [start]))
31     visited = set()
32
33     while queue:
34         (current, path) = queue.pop(0)
35         if current == goal:
36             return path
37         for neighbor in getNeighbors(current, matrix):
38             if neighbor not in visited:
39                 visited.add(neighbor)
40                 queue.append((neighbor, path + [neighbor]))
41    return -1
```

1.2 Sample output

```
1 Enter number of rows: 5
2 Enter number of columns: 5
3 Creating matrix and selecting one with a valid path...
4 Matrix:
5 [1, 1, 0, 0, 1]
6 [1, 1, 1, 0, 1]
7 [1, 1, 1, 0, 0]
8 [0, 1, 1, 0, 0]
9 [1, 0, 1, 1, 1]
10
11 DFS:
12 Path:
13 (0, 0) -> (1, 0) -> (2, 0) -> (2, 1) -> (3, 1)
14 -> (3, 2) -> (4, 2) -> (4, 3) -> (4, 4)
15 * 1 0 0 1
16 * 1 1 0 1
17 * * 1 0 0
18 0 * * 0 0
19 1 0 * * *
20
21 BFS:
22 Path:
23 (0, 0) -> (0, 1) -> (1, 1) -> (1, 2) -> (2, 2)
24 -> (3, 2) -> (4, 2) -> (4, 3) -> (4, 4)
25 * * 0 0 1
26 1 * * 0 1
27 1 1 * 0 0
28 0 1 * 0 0
29 1 0 * * *
```

2 Question II

Q2



DFS

Expanded	Fringe	Visited
	B	B
B	B-E, B-A	E, A
A	B-A-D	D
D	B-A-D-G, B-A-D-F	B , F
F		
G		

Expansion Order: B, A, D, F, G

Path Returned: B, A, D, G

Cost: 6

BFS

Expanded	Fringe	Visited
	B	B
B	B-A , B-E	A , E
A	B-A-D	D , F
E	B-E-C , B-E-F	C , F
D	B-A-D-F, B-A-D-G	F , G
C	B-E-C-A	A
F		
G		

Expansion Order: B, A, E, D, C, F, G

Path Returned: B, A, D, G

Cost: 6

UCS

Exp.	Fringe	Visited
	B	B
B	B-E , B-A	A , E
E	B-E-C , B-E-F	C , F
A	B-A-D	D: 5
F		
C	B-E-C-A	A: 6

D: B-A-D-F, B-A-D-G F: 7, G: 6

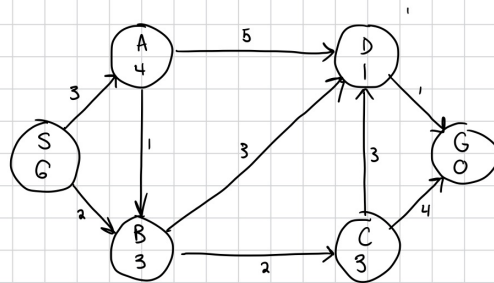
Expansion Order: B, E, A, F, C, D, G

Path Returned: B, A, D, G

Cost: 6

3 Question III

Q3



DFS:

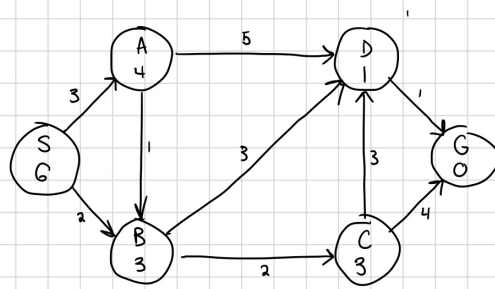
Exp	Fringe	Visited
	S	S
S	S-A , S-B	B , A
A	S-A-B , S-A-D	D , B
B	S-A-B-C , S-A-B-D	D , C
C	S-A-B-C-G , S-A-B-C-D	G , D
D	S-A-B-C-D-G	G
G		

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:

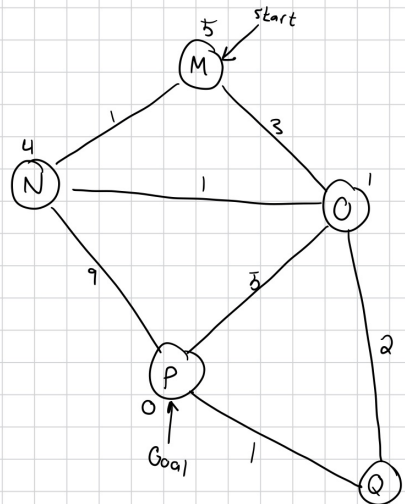
Exp	Fringe	Visited	
	S	S	Path: S-B-D-G
S	S-A, S-B	B: 2, A: 3	Expansion: S-B-A-C-D-G
B	S-B-C, S-B-D	C: 5, D: 5	Cost: 6
A	S-A-D, S-A-B	D: 8	
C	S-B-C-D, S-B-C-G	D: 7, G: 8	
D	S-B-D-G	G: 6	
G			

A*

Exp	Fringe	Visited	
	S	S	Path: S-B-D-G
S	S-A, S-B	A: 7, B: 5	Expansion: S-B-D-G
B	S-B-C, S-B-D	C: 7, D: 5	Cost: 6
D	S-B-D-G	G: 6	
G			

4 Question IV

Q4



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

```
1 def depthFirstSearch(problem):
2     """ YOUR CODE HERE """
3
4     stack = util.Stack()
5     visited = set()
6
7     # (curPos, wallsHit)
8     visited.add((problem.getStartState(),0))
9
10    # (curPos, wallsHit, path)
11    stack.push((problem.getStartState(),0,[]))
12
13    if problem.isGoalState(problem.getStartState()):
14        return []
15
16    while True:
17        if stack.isEmpty():
18            return []
19
20        curPos, wallsHit, path = stack.pop()
21        if wallsHit > 2:
22            continue
23
24        if problem.isGoalState(curPos) and wallsHit > 0 and
25        wallsHit <=2:
26            return path
27
28        for neighbor, dir, cost in problem.getSuccessors(curPos):
29            if problem.isWall(neighbor):
30                nextPos = (neighbor, wallsHit + 1)
31            else:
32                nextPos = (neighbor, wallsHit)
33
34            if nextPos not in visited and nextPos:
35                stack.push((nextPos[0],nextPos[1], path+[dir]))
36                visited.add(nextPos)
```


5.2 BFS Implementation

```
1 def breadthFirstSearch(problem):
2     """Search the shallowest nodes in the search tree first."""
3     """ YOUR CODE HERE """
4     queue = util.Queue()
5     visited = set()
6
7     # (curPos, wallsHit)
8     visited.add((problem.getStartState(),0))
9
10    # (curPos, wallsHit, path)
11    queue.push((problem.getStartState(),0,[]))
12
13    if problem.isGoalState(problem.getStartState()):
14        return []
15
16    while True:
17        if queue.isEmpty():
18            return []
19
20        curPos, wallsHit, path = queue.pop()
21        if wallsHit > 2:
22            continue
23
24        if problem.isGoalState(curPos) and wallsHit > 0 and
25        wallsHit <=2:
26            return path
27
28        for neighbor, dir, cost in problem.getSuccessors(curPos):
29            if problem.isWall(neighbor):
30                nextPos = (neighbor, wallsHit + 1)
31            else:
32                nextPos = (neighbor, wallsHit)
33
34            if nextPos not in visited and nextPos:
35                queue.push((nextPos[0],nextPos[1], path+[dir]))
36                visited.add(nextPos)
```

5.3 UCS Implementation

```
1 def uniformCostSearch(problem):
2     """Search the node of least total cost first."""
3     """ YOUR CODE HERE """
4     priorityQueue = util.PriorityQueue()
5     distance = dict()
6
7     # [(pos), cost]
8     distance[(problem.getStartState(), 0)] = 0
9
10    # (curPos, wallsHit, path), priority
11    priorityQueue.push((problem.getStartState(), 0, []), 0)
12
13    if problem.isGoalState(problem.getStartState()):
14        return []
15
16    while True:
17        if priorityQueue.isEmpty():
18            return []
19
20        curPos, wallsHit, path = priorityQueue.pop()
21        if wallsHit > 2:
22            continue
23
24        if problem.isGoalState(curPos) and wallsHit > 0 and
25        wallsHit <= 2:
26            return path
27
28        for neighbor, dir, cost in problem.getSuccessors(curPos):
29            if problem.isWall(neighbor):
30                nextPos = (neighbor, wallsHit + 1)
31            else:
32                nextPos = (neighbor, wallsHit)
33
34            costOfNextPos = (problem.getCostOfActions(path +
35            [dir]))
36            if costOfNextPos < distance.get(nextPos, float('inf')):
37                priorityQueue.push((nextPos[0], nextPos[1],
38                path+[dir]), costOfNextPos)
39                distance[nextPos] = costOfNextPos
```

5.4 A* Implementation

```
1 def aStarSearch(problem, heuristic=nullHeuristic):
2     """Search the node that has the lowest combined cost and
3     heuristic first."""
4     """ *** YOUR CODE HERE *** """
5     priorityQueue = util.PriorityQueue()
6     distance = dict()
7
8     # [(pos), cost]
9     distance[(problem.getStartState(), 0)] = 0
10    # (curPos, wallsHit, path), priority
11    priorityQueue.push((problem.getStartState(), 0, []), priority=0)
12
13    if problem.isGoalState(problem.getStartState()):
14        return []
15
16    while True:
17        if priorityQueue.isEmpty():
18            return []
19
20        curPos, wallsHit, path = priorityQueue.pop()
21        if wallsHit > 2:
22            continue
23
24        if problem.isGoalState(curPos) and wallsHit > 0 and
25        wallsHit <= 2:
26            return path
27
28        for neighbor, dir, cost in problem.getSuccessors(curPos):
29            if problem.isWall(neighbor):
30                nextPos = (neighbor, wallsHit + 1)
31            else:
32                nextPos = (neighbor, wallsHit)
33
34            costOfNextPos = (problem.getCostOfActions(path +
35            [dir]) + heuristic(neighbor, problem))
36            if costOfNextPos < distance.get(nextPos, float('inf')):
37                priorityQueue.push((nextPos[0], nextPos[1],
38                path+[dir]), costOfNextPos)
39                distance[nextPos] = costOfNextPos
```