

Faculty of Computing



Artificial Intelligence Spring 2025 Lab # 7

Instructor

Ayesha Akram

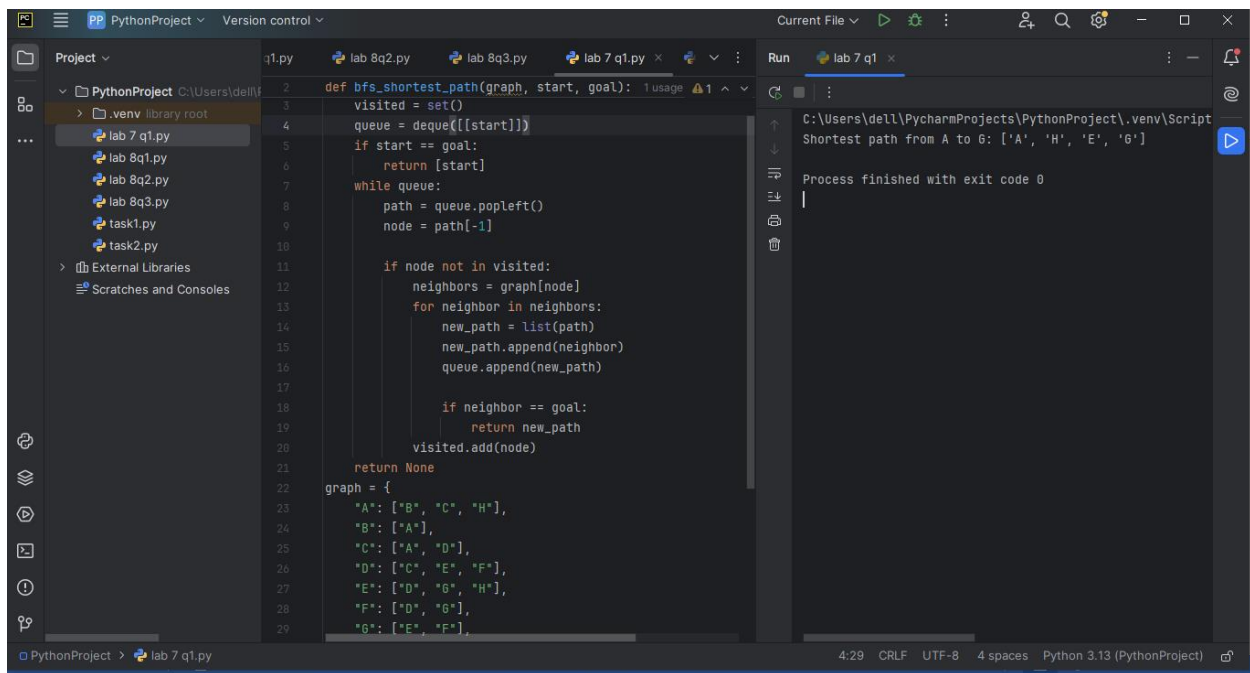
Submitted by:

Faareha Raza(47431)

Question 01:

Write a program to traverse a graph using the shortest BFS algorithm.

```
graph = {
    "A": ["B", "C", "H"],
    "B": ["A"],
    "C": ["A", "D"],
    "D": ["C", "E", "F"],
    "E": ["D", "G", "H"],
    "F": ["D", "G"],
    "G": ["E", "F"],
    "H": ["A", "E"]
}
```



The screenshot shows a PyCharm IDE with a Python project named 'PythonProject'. The file explorer on the left shows a directory structure with files like 'lab 7 q1.py', 'lab 8 q1.py', 'lab 8 q2.py', 'lab 8 q3.py', 'task1.py', and 'task2.py'. The main editor displays the code for 'lab 7 q1.py', which implements a BFS algorithm to find the shortest path from node 'A' to node 'G' in the given graph. The code defines a function 'bfs_shortest_path' that takes a graph, a start node, and a goal node as arguments. It uses a queue to explore nodes level by level, returning the shortest path as a list of nodes. The graph is defined as a dictionary with nodes as keys and lists of neighbors as values. The output of the program is shown in the Run console, indicating that the shortest path from A to G is ['A', 'H', 'E', 'G'] and the process finished with exit code 0.

```
def bfs_shortest_path(graph, start, goal):
    visited = set()
    queue = deque([[start]])
    if start == goal:
        return [start]
    while queue:
        path = queue.popleft()
        node = path[-1]
        if node not in visited:
            neighbors = graph[node]
            for neighbor in neighbors:
                new_path = list(path)
                new_path.append(neighbor)
                queue.append(new_path)
            if neighbor == goal:
                return new_path
            visited.add(node)
    return None

graph = {
    "A": ["B", "C", "H"],
    "B": ["A"],
    "C": ["A", "D"],
    "D": ["C", "E", "F"],
    "E": ["D", "G", "H"],
    "F": ["D", "G"],
    "G": ["E", "F"],
    "H": ["A", "E"]
}
```

Run console output:

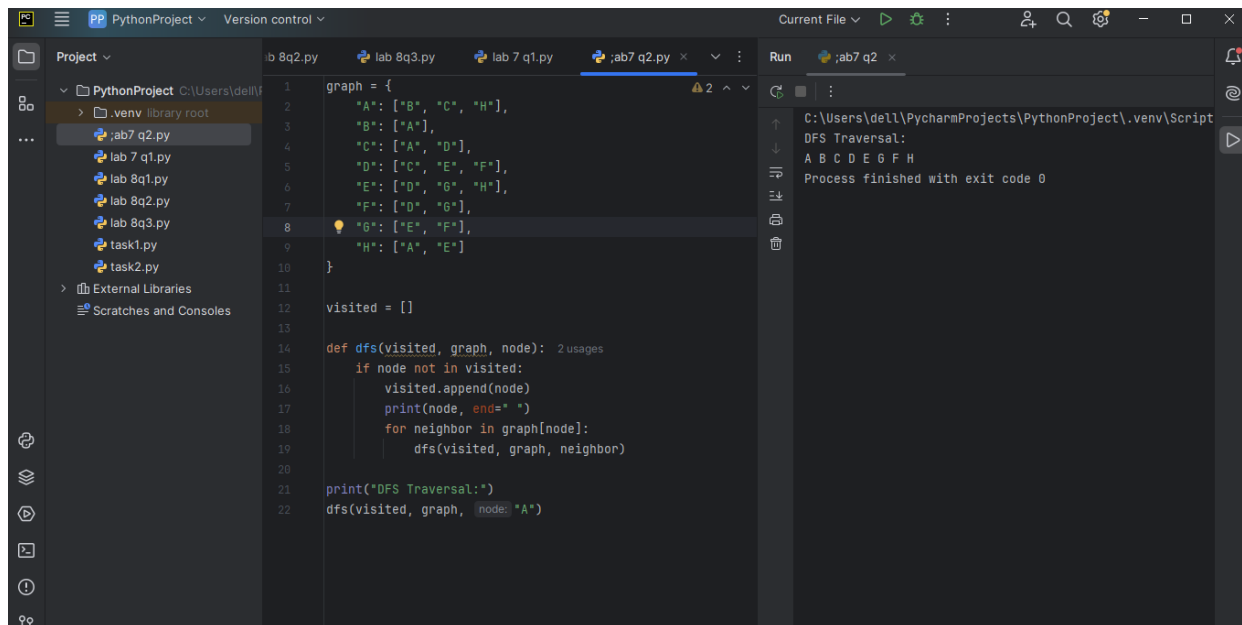
```
C:\Users\dell\PycharmProjects\PythonProject\.venv\Script
Shortest path from A to G: ['A', 'H', 'E', 'G']

Process finished with exit code 0
```

Question 02:

Write a program for Depth First Search on the graph below

```
graph = {
    "A": ["B", "C", "H"],
    "B": ["A"],
    "C": ["A", "D"],
    "D": ["C", "E", "F"],
    "E": ["D", "G", "H"],
    "F": ["D", "G"],
    "G": ["E", "F"],
    "H": ["A", "E"]
}
```



The screenshot shows a PyCharm IDE with a project named 'PythonProject'. The file explorer on the left shows a directory structure with files like 'lab 7 q1.py', 'lab 8 q1.py', 'lab 8 q2.py', 'lab 8 q3.py', 'task1.py', and 'task2.py'. The main editor window displays a Python script for Depth First Search (DFS) traversal of a graph. The script defines a graph and a recursive function 'dfs' to traverse it. The output of the program is shown in the Run console.

```
graph = {
    "A": ["B", "C", "H"],
    "B": ["A"],
    "C": ["A", "D"],
    "D": ["C", "E", "F"],
    "E": ["D", "G", "H"],
    "F": ["D", "G"],
    "G": ["E", "F"],
    "H": ["A", "E"]
}

visited = []

def dfs(visited, graph, node):
    if node not in visited:
        visited.append(node)
        print(node, end=" ")
        for neighbor in graph[node]:
            dfs(visited, graph, neighbor)

print("DFS Traversal:")
dfs(visited, graph, "A")
```

Run console output:

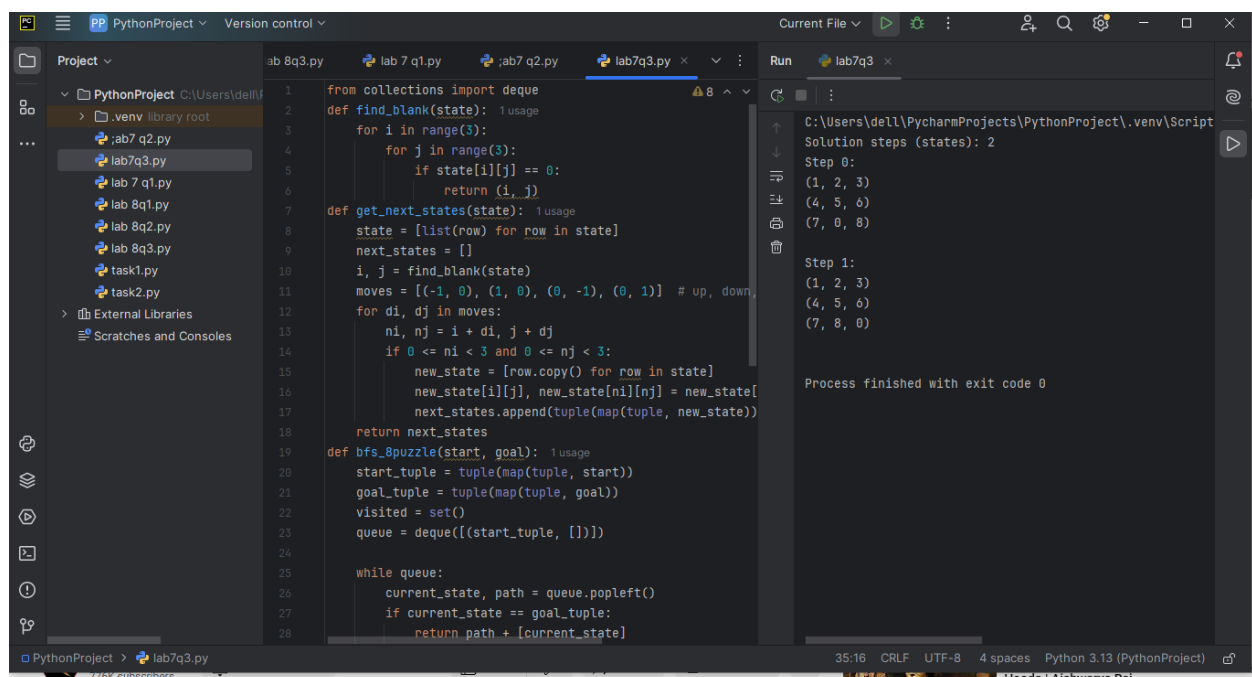
```
C:\Users\de\PycharmProjects\PythonProject\.venv\Script
DFS Traversal:
A B C D E G F H
Process finished with exit code 0
```

Question 03:

8-puzzle problem:

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match the final configuration using the empty space. We can slide four adjacent (left, right, above, and below) tiles into the empty space

- Solve this problem using the BFS algorithm in python.
- Take an example matrix of 3x3 and a goal matrix of 3x3.
- Must give a dry run of your example



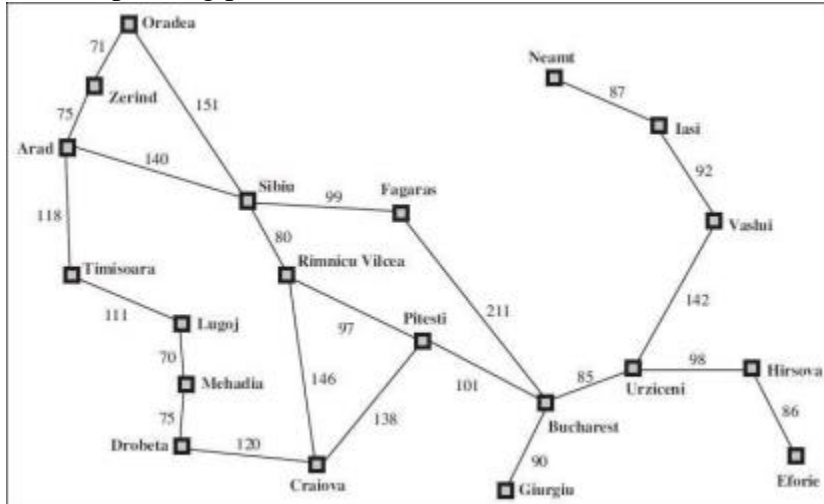
```
1 from collections import deque
2 def find_blank(state): 1 usage
3     for i in range(3):
4         for j in range(3):
5             if state[i][j] == 0:
6                 return (i, j)
7 def get_next_states(state): 1 usage
8     state = [list(row) for row in state]
9     next_states = []
10    i, j = find_blank(state)
11    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down,
12    for di, dj in moves:
13        ni, nj = i + di, j + dj
14        if 0 <= ni < 3 and 0 <= nj < 3:
15            new_state = [row.copy() for row in state]
16            new_state[i][j], new_state[ni][nj] = new_state[
17                ni][nj], new_state[i][j]
18            next_states.append(tuple(map(tuple, new_state)))
19    return next_states
20 def bfs_8puzzle(start, goal): 1 usage
21    start_tuple = tuple(map(tuple, start))
22    goal_tuple = tuple(map(tuple, goal))
23    visited = set()
24    queue = deque([(start_tuple, [])])
25    while queue:
26        current_state, path = queue.popleft()
27        if current_state == goal_tuple:
28            return path + [current_state]
```

Run console output:

```
C:\Users\dell\PycharmProjects\PythonProject\.venv\Script
Solution steps (states): 2
Step 0:
(1, 2, 3)
(4, 5, 6)
(7, 0, 8)
Step 1:
(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
Process finished with exit code 0
```

Question 04:

Imagine going from Arad to Bucharest in the following map. Your goal is to minimize the distance mentioned in the map during your travel. Implement a depth first search to find the corresponding path.



```
PythonProject Version control Current File Run lab7q4 x
Project C:\Users\dell\PythonProject
  .venv library root
  lab7 q2.py
  lab7q3.py
  lab7q4.py
  lab 7 q1.py
  lab 8q1.py
  lab 8q2.py
  lab 8q3.py
  task1.py
  task2.py
  External Libraries
  Scratches and Consoles

lab 7 q1.py lab7 q2.py lab7q3.py lab7q4.py task2.py Run lab7q4 x
def dfs_path(graph, start, goal, path=None, visited=None, total_distance=0):
    if path is None:
        path = []
    if visited is None:
        visited = set()
    path = path + [start]
    visited.add(start)

    if start == goal:
        return (path, total_distance)

    min_path = None
    min_distance = float('inf')

    for neighbor, distance in graph[start]:
        if neighbor not in visited:
            new_total = total_distance + distance
            result = dfs_path(graph, neighbor, goal, path, visited.copy(), new_total)
            if result is not None:
                result_path, result_distance = result
                if result_distance < min_distance:
                    min_distance = result_distance
                    min_path = result_path

    return (min_path, min_distance) if min_path else None

path, total_distance = dfs_path(romania_map, start='Arad', goal='Bucharest')
print("DFS Path from Arad to Bucharest:")
```

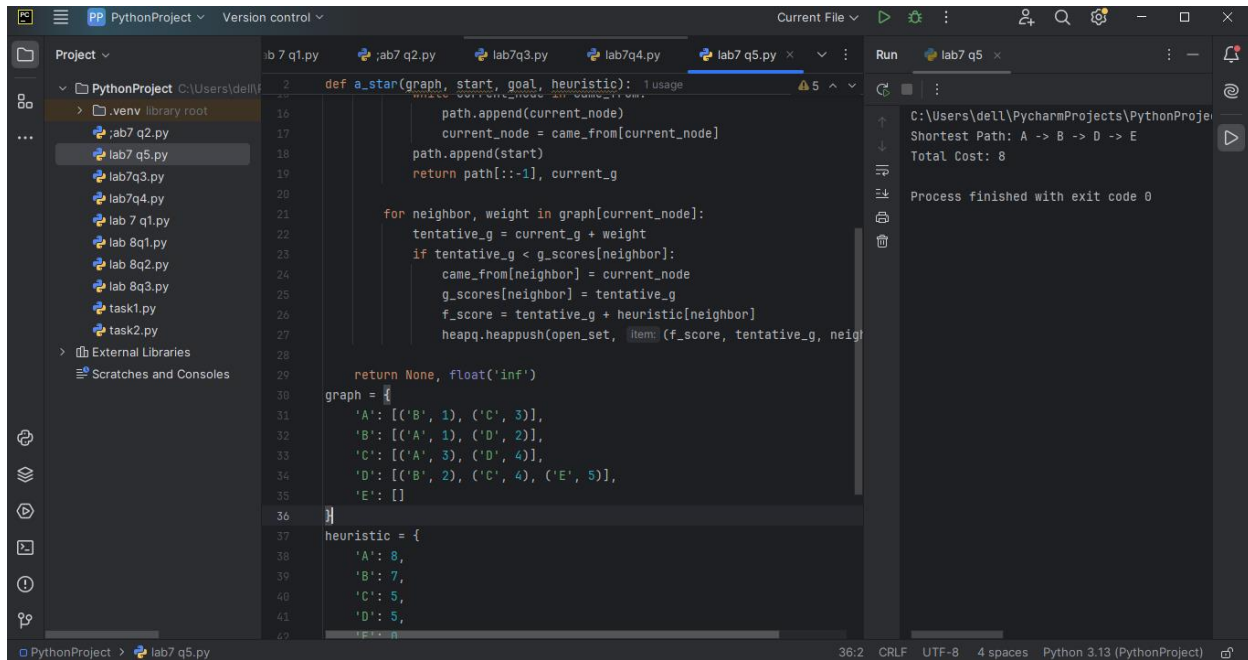
```
C:\Users\dell\PythonProjects\PythonProject> python lab7q4.py
DFS Path from Arad to Bucharest:
Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti
Total Distance: 418
Process finished with exit code 0
```

PythonProject > lab7q4.py 40:58 CRLF UTF-8 4 spaces Python 3.13 (PythonProject)

Question 05:

Create a graph with weighted edges.

Implement A* to find the shortest path between two nodes.



```
def a_star(graph, start, goal, heuristic):
    path.append(current_node)
    current_node = came_from[current_node]
    path.append(start)
    return path[::-1], current_g

    for neighbor, weight in graph[current_node]:
        tentative_g = current_g + weight
        if tentative_g < g_scores[neighbor]:
            came_from[neighbor] = current_node
            g_scores[neighbor] = tentative_g
            f_score = tentative_g + heuristic[neighbor]
            heapq.heappush(open_set, (f_score, tentative_g, neighbor))

    return None, float('inf')

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 2)],
    'C': [('A', 3), ('D', 4)],
    'D': [('B', 2), ('C', 4), ('E', 5)],
    'E': []
}

heuristic = {
    'A': 8,
    'B': 7,
    'C': 5,
    'D': 5,
    'E': 0
}
```

Shortest Path: A -> B -> D -> E
Total Cost: 8
Process finished with exit code 0

Question 06:

Implement a Basic Minimax for Tic-Tac-Toe

- Create a **3x3 Tic-Tac-Toe board**.
- Use **Minimax** to find the best move for a player.
- Assume 'X' is the maximizer and 'O' is the minimizer.
- Use a recursive function that assigns **+1 (win), -1 (loss), or 0 (draw)**.
- Implement a **function to check winning conditions**.

