Faculty of Computing

Artificial Intelligence
Spring 2025
Lab # 7

Instructor

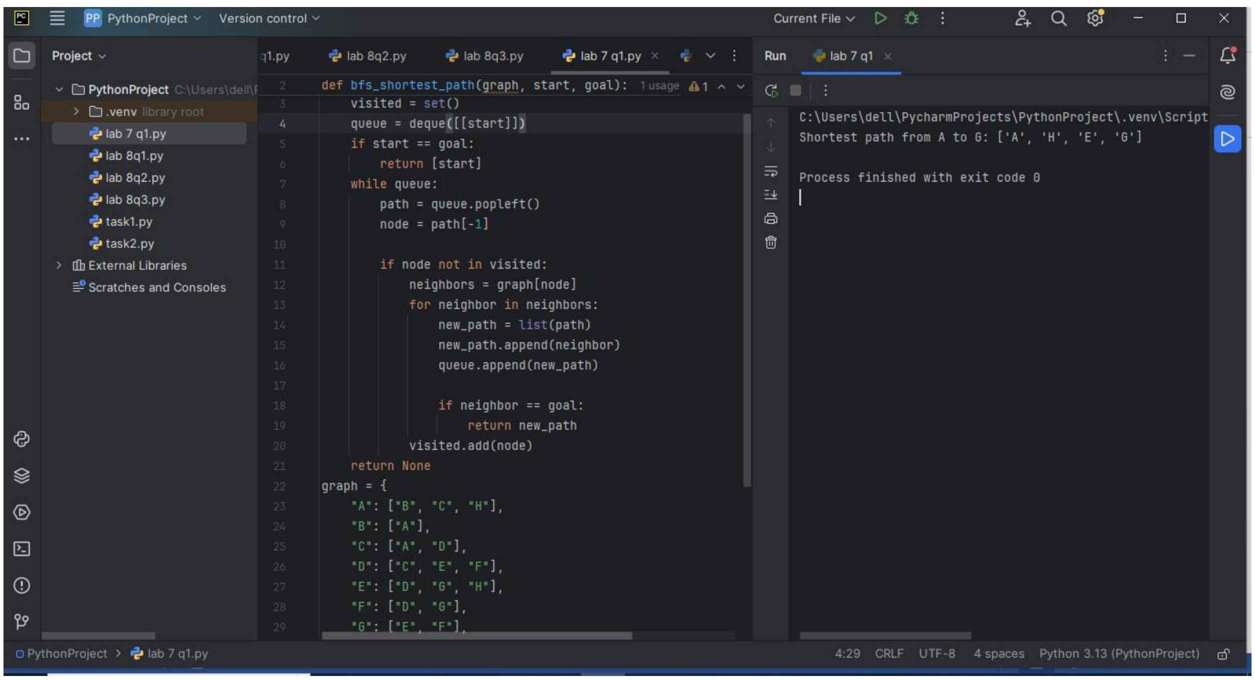
Ayesha Akram

Submitted by:
Faareha Raza(47431)

Question 01:

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

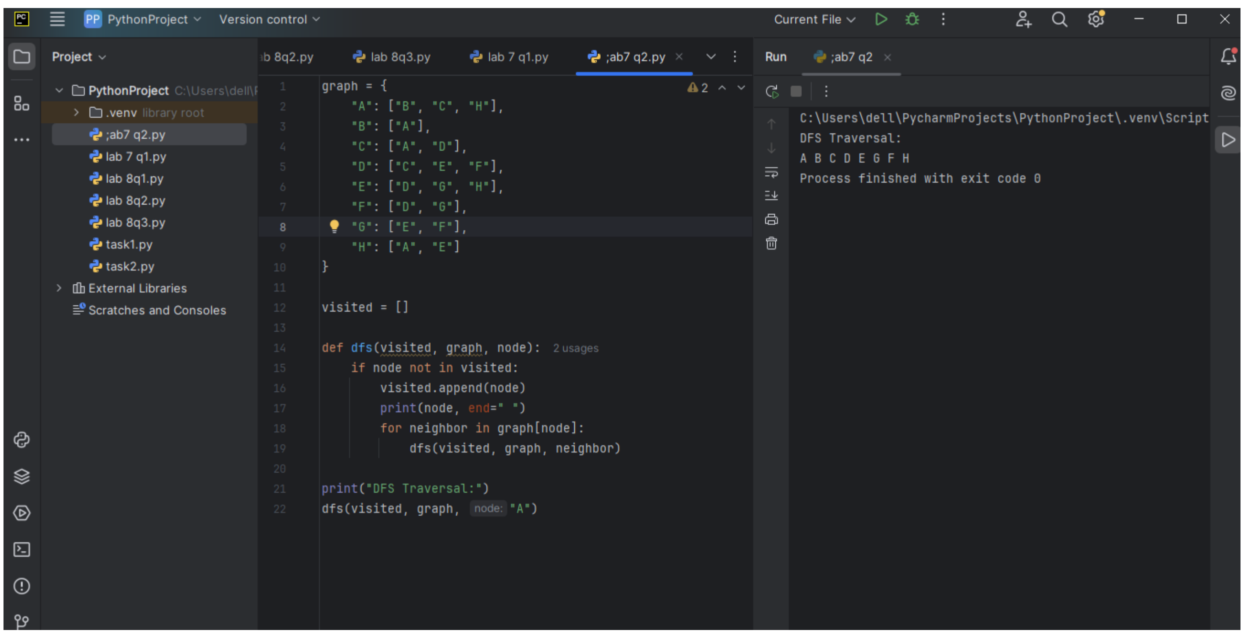
```
graph = {
  "A": ["B","C","H"],
  "B": ["A"],
  "C": ["A","D"],
  "D": ["C","E","F"],
  "E": ["D","G","H"],
  "F": ["D","G"],
  "G": ["E","F"],
  "H": ["A","E"]
}
```



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"]
}
```



Question 03:

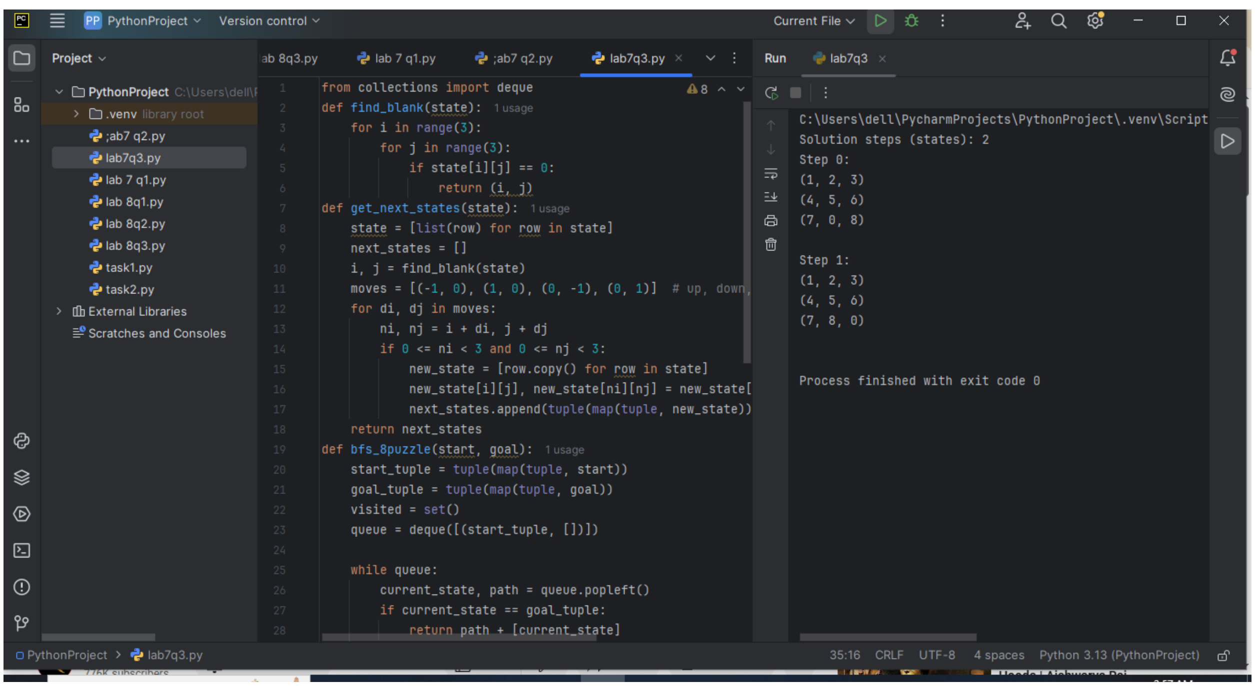
8-puzzle problem:

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.

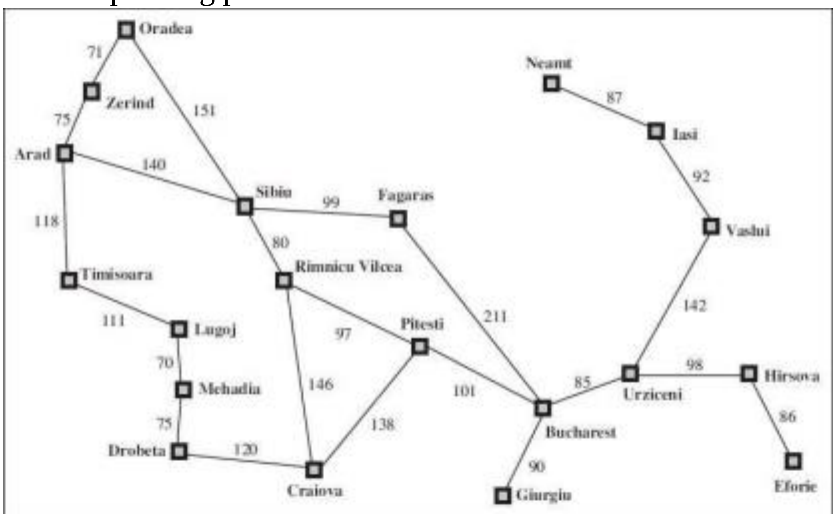
- Take an example matrix of 3x3 and a goal matrix of 3x3.

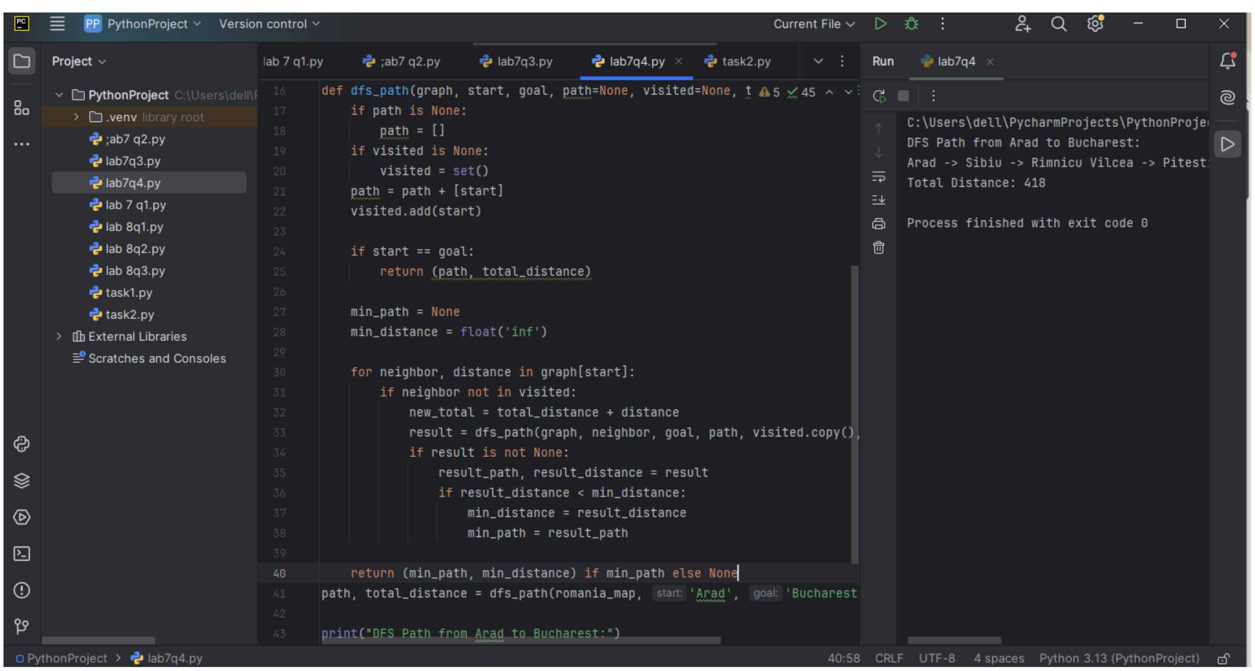
- Must give a dry run of your example



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.

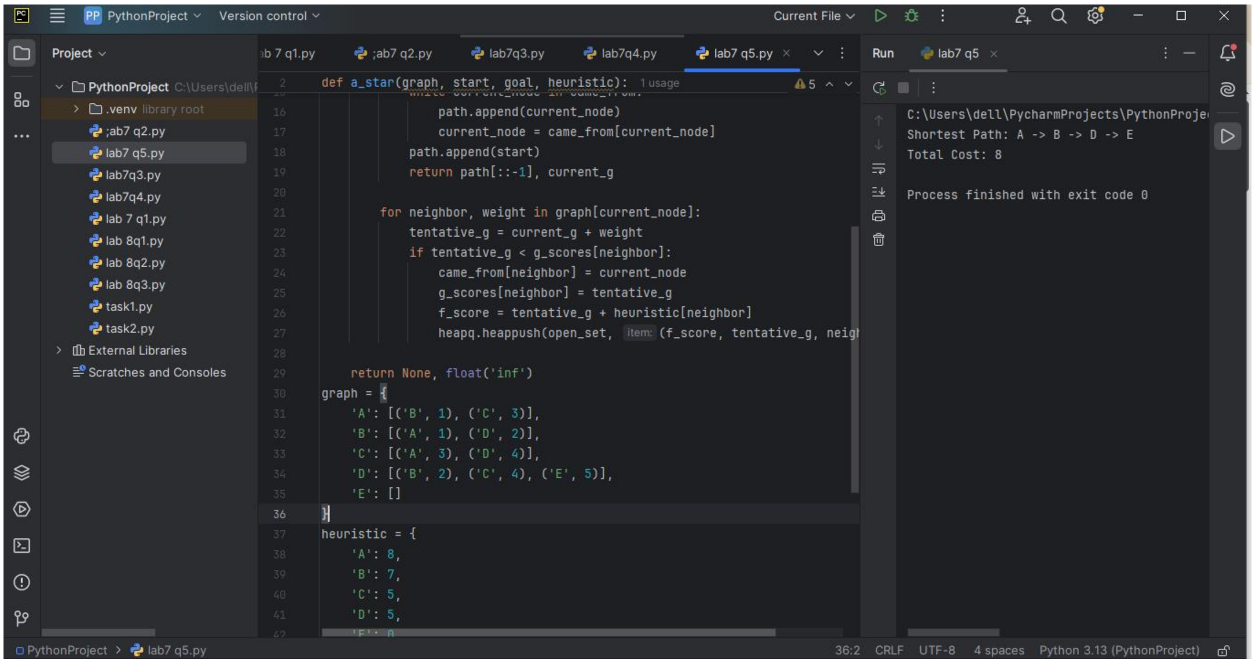




Question 05:

Create a graph with weighted edges.

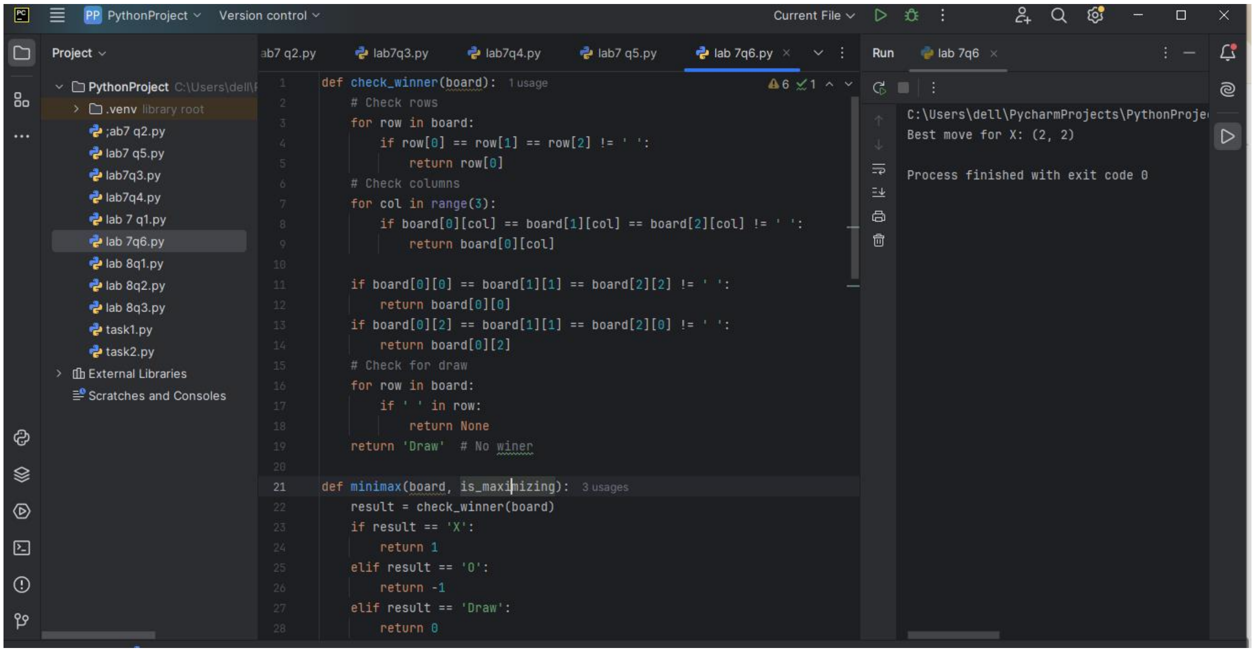
Implement A* to find the shortest path between two nodes.



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**.


PythonProject
Version control
Current File
Project
PythonProject C:\Users\dell\
.venv library root
;ab7 q2.py
lab7 q5.py
lab7q3.py
lab7q4.py
lab 7 q1.py
lab 7q6.py
lab 8q1.py
lab 8q2.py
lab 8q3.py
task1.py
task2.py
External Libraries
Scratches and Consoles
ab7 q2.py
lab7q3.py
lab7q4.py
lab7 q5.py
lab 7q6.py
def check_winner(board): 1 usage
# Check rows
for row in board:
if row[0] == row[1] == row[2] != ' ':
return row[0]
# Check columns
for col in range(3):
if board[0][col] == board[1][col] == board[2][col] != ' ':
return board[0][col]
if board[0][0] == board[1][1] == board[2][2] != ' ':
return board[0][0]
if board[0][2] == board[1][1] == board[2][0] != ' ':
return board[0][2]
# Check for draw
for row in board:
if ' ' in row:
return None
return 'Draw' # No winer
def minimax(board, is_maximizing): 3 usages
result = check_winner(board)
if result == 'X':
return 1
elif result == 'O':
return -1
elif result == 'Draw':
return 0
Run
lab 7q6
C:\Users\dell\PycharmProjects\PythonProje
Best move for X: (2, 2)
Process finished with exit code 0