RIPHAH INTERNATIONAL UNIVERSITY, ISLAMABAD

Lab#7

Bachelors of Computer Science – 6th Semester

Course: Artificial Intelligence

Submitted to: Ms. Ayesha

Submitted by: Tabinda Hassan

SAP-46374

Date of Submission: 25-03-2025

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"]
}
```

AI_Lab#7_Task#6.py > ...

```python
from collections import deque

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"]
}

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")  # Print the node
            visited.add(node)
            queue.extend(graph[node])  # Add neighbors to the queue

# Call the function
bfs(graph, "A")
```

```
[Running] python -u
A B C H D E F G
[Done] exited with
```

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"]
}
```

AI_Lab#7_Task#2.py > ...

```python
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"]
}

def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()

    print(start, end=" ")  # Print the node
    visited.add(start)

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# Call the function
dfs(graph, "A")

```

```
[Running] python -u "d:\
A B C D E G F H
[Done] exited with code=
```

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.

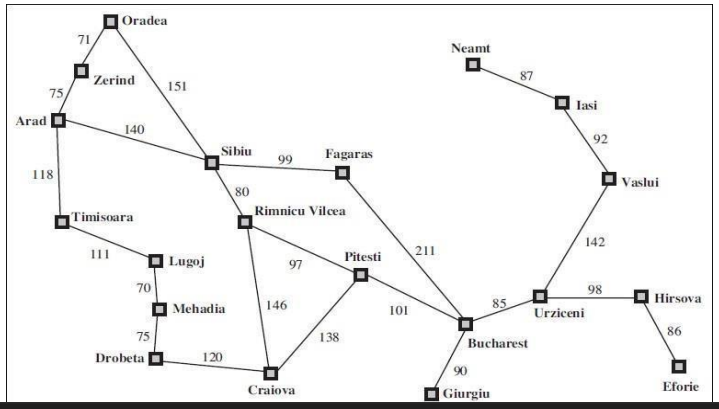
- Must give a dry run of your example

AI_Lab#7_Task#3.py > ...

```python
from collections import deque

def bfs_8_puzzle(start, goal):
    queue = deque([(start, [])])
    visited = set()

    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path

        if tuple(state) in visited:
            continue
        visited.add(tuple(state))

        empty = state.index(0)
        moves = []
        if empty % 3 > 0:  # Left move
            moves.append(empty - 1)
        if empty % 3 < 2:  # Right move
            moves.append(empty + 1)
        if empty // 3 > 0:  # Up move
            moves.append(empty - 3)
        if empty // 3 < 2:  # Down move
            moves.append(empty + 3)

        for move in moves:
            new_state = state[:]
            new_state[empty], new_state[move] = new_state[move], new_state[empty]
            queue.append((new_state, path + [new_state]))

    return None

start = [1, 2, 3, 4, 5, 6, 7, 0, 8]  # Example start state
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]  # Goal state

solution = bfs_8_puzzle(start, goal)
print(solution)
```

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.



AI_Lab#7_Task#4.py > ...

```python
graph = {
    "Arad": ["Zerind", "Sibiu", "Timisoara"],
    "Zerind": ["Arad", "Oradea"],
    "Oradea": ["Zerind", "Sibiu"],
    "Sibiu": ["Arad", "Oradea", "Fagaras", "Rimnicu Vilcea"],
    "Fagaras": ["Sibiu", "Bucharest"],
    "Timisoara": ["Arad", "Lugoj"],
    "Lugoj": ["Timisoara", "Mehadia"],
    "Mehadia": ["Lugoj", "Drobeta"],
    "Drobeta": ["Mehadia", "Craiova"],
    "Craiova": ["Drobeta", "Rimnicu Vilcea", "Pitesti"],
    "Rimnicu Vilcea": ["Sibiu", "Craiova", "Pitesti"],
    "Pitesti": ["Rimnicu Vilcea", "Craiova", "Bucharest"],
    "Bucharest": ["Fagaras", "Pitesti", "Giurgiu"]
}

def dfs_path(graph, start, goal, path=None):
    if path is None:
        path = []
    path.append(start)

    if start == goal:
        return path

    for neighbor in graph[start]:
        if neighbor not in path:
            new_path = dfs_path(graph, neighbor, goal, path[:])
            if new_path:
                return new_path
    return None

print(dfs_path(graph, "Arad", "Bucharest"))
```

```
[Running] python -u "d:\BSCS_6th_Semester\Artificial_Intelligence\L
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest']
```

Question 05:

Create a graph with weighted edges.

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

AI_Lab#7_Task#5.py > ...

```python
import heapq

graph = {
    "A": {"B": 1, "C": 4},
    "B": {"A": 1, "D": 2, "E": 5},
    "C": {"A": 4, "F": 3},
    "D": {"B": 2, "G": 7},
    "E": {"B": 5, "H": 8},
    "F": {"C": 3, "I": 6},
    "G": {"D": 7},
    "H": {"E": 8},
    "I": {"F": 6}
}

def a_star(graph, start, goal):
    pq = [(0, start, [])]
    visited = set()

    while pq:
        cost, node, path = heapq.heappop(pq)
        if node in visited:
            continue
        path.append(node)
        visited.add(node)

        if node == goal:
            return path

        for neighbor, weight in graph[node].items():
            heapq.heappush(pq, (cost + weight, neighbor, path[:]))

    return None

print(a_star(graph, "A", "G"))
```

```
[Running] python -u "d:\BSCS_6
['A', 'B', 'D', 'G']
```

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**.

AI_Lab#7_Task#6.py > ...

```python
import math

def is_winner(board, player):
    win_states = [
        [board[0], board[1], board[2]],
        [board[3], board[4], board[5]],
        [board[6], board[7], board[8]],
        [board[0], board[3], board[6]],
        [board[1], board[4], board[7]],
        [board[2], board[5], board[8]],
        [board[0], board[4], board[8]],
        [board[2], board[4], board[6]],
    ]
    return [player, player, player] in win_states

def minimax(board, depth, is_max):
    if is_winner(board, "X"):
        return 1
    if is_winner(board, "O"):
        return -1
    if " " not in board:
        return 0

    if is_max:
        best = -math.inf
        for i in range(9):
            if board[i] == " ":
                board[i] = "X"
                best = max(best, minimax(board, depth + 1, False))
                board[i] = " "
        return best
    else:
```

```
        best = math.inf
        for i in range(9):
            if board[i] == " ":
                board[i] = "O"
                best = min(best, minimax(board, depth + 1, True))
                board[i] = " "
        return best

# Example empty board
board = [" "] * 9
print(minimax(board, 0, True))

```

```
[Running] python -u '
0
```