TP4: Advanced data structure: Graphs

Problem

Find the Shortest Path Between Two Nodes in a Graph Using DFS

1. writing a program in C/C++ to find the shortest path between two nodes in a weighted directed graph using DFS.
2. Calculate the experimental time and space complexity of your program (using multiple graphs examples).

Steps to Follow:

1. **Graph Representation:**
   - Represent the graph as an **adjacency list or adjacency matrix**.
   - Nodes are numbered from 0 to n−1 (n being the number of nodes).
   - Each edge has a weight (representing a distance or cost).
2. **DFS Algorithm (recursive and iterative): see appendix**

   Implement a function to
   - explore all possible paths between the source and destination nodes.
   - Calculate the cost of each path as you traverse the graph.
   - Keep track of the path with the minimum cost.
3. **Program Inputs:**
   - Number of nodes (n) and edges (m).
   - The edges in the form (u, v, w), where u and v are the connected nodes, and w is the weight of the edge.
   - Source and destination nodes.
4. **Program Outputs:**
   - Display the shortest path as a list of nodes.
   - Display the total cost of the shortest path.

(G1-G2-G4) Belkadi Widad Hassina: belkadi.wh.usthb@gmail.com

(G3) Ait Mehdi Mohamed: ait7mehdi7mohamed@gmail.com

Example Input/Output:

Input:
Number of nodes: 4
Number of edges: 5
Edges:

Copy code

0 1 1

0 2 4

1 2 2

1 3 6

2 3 3

Source: 0
Destination: 3

Expected Output:
Shortest path: 0 -> 1 -> 2 -> 3
Total cost: 6

Appendix

1. Recursive DFS algorithm

```
function DFS_recursive(graph, start, visited):
    if start not in visited:
        mark start as visited
        process start
        for each neighbor in graph[start]:
            if neighbor not in visited:
                DFS_recursive(graph, neighbor, visited)
```

(G1-G2-G4) Belkadi Widad Hassina: belkadi.wh.usthb@gmail.com

(G3) Ait Mehdi Mohamed: ait7mehdi7mohamed@gmail.com

2. Iterative DFS algorithm

```
function DFS_iterative(graph, start):
    stack = [start]
    visited = set()
    while stack is not empty:
        current = stack.pop()
        if current not in visited:
            mark current as visited
            process current
            for each neighbor in graph[current]:
                if neighbor not in visited:
                    stack.push(neighbor)
```

3. Example Execution

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
```

Recursive Execution

Start DFS from node A.

1. Visit A → visited = {A}
2. Move to neighbor B → visited = {A, B}
3. Move to neighbor D → visited = {A, B, D} (No more neighbors for D, backtrack to B)
4. Move to neighbor E → visited = {A, B, D, E}
5. Move to neighbor F → visited = {A, B, D, E, F} (No more neighbors for F, backtrack to A)
6. Move to neighbor C → visited = {A, B, D, E, F, C}

Result: A → B → D → E → F → C

(G1-G2-G4) Belkadi Widad Hassina: belkadi.wh.usthb@gmail.com

(G3) Ait Mehdi Mohamed: ait7mehdi7mohamed@gmail.com