23CCE201 DATA STRUCTURES

ASSIGNMENT

NAME: VETHAVIYASH.K **ROLL.NO:** CB.EN.U4CCE23063

Report on Traversal Schemes on Weighted Directed Graphs

1. Aim

The aim of this report is to explore various traversal schemes applicable to weighted directed graphs, such as Breadth-First Search (BFS) and Depth-First Search (DFS), and analyze them.

2. Logic Followed

Breadth-First Search (BFS) :

- **Initialization**: Set all vertices to "UNVISITED," initialize distances to infinity (0 for the start vertex), and create a queue for exploration.
- **Queue Processing**: Enqueue the starting vertex and mark it as "VISITING." While the queue is not empty, dequeue a vertex to process.
- **Neighbor Exploration**: For each unvisited neighbor of the processed vertex, mark it as "VISITING," update its distance and parent, and enqueue it.
- **Output**: The algorithm returns the order of traversal and distances from the starting vertex to all reachable vertices.

Depth-First Search (DFS) :

- **Initialization**: Set all vertices to "UNVISITED," initialize parent nodes, and set up a time counter for discovery and finish times.
- **Vertex Exploration**: For each unvisited vertex, call the EXPLORE function to mark it as "VISITING" and record its discovery time.
- **Recursive Neighbor Exploration**: For each unvisited neighbor, set its parent and recursively call EXPLORE until all reachable vertices are visited.
- **Output**: The algorithm returns discovery and finish times for each vertex, capturing the exploration order and structure of the graph.

3. Algorithm

Breadth-First Search (BFS) Algorithm

- **Input**: A graph with vertices and edges, and a starting vertex.
- **Output**: The order of traversal and distances from the starting vertex.
    - **Initialization**:
        - Set all vertices to "UNVISITED."
        - Set distances to infinity and parent nodes to "NONE."
        - Set the starting vertex to "VISITING" and its distance to 0.
        - Create an empty queue and add the starting vertex to it.
    - **Traversal Process**:
        - While the queue is not empty:
            - Remove a vertex from the queue and process it.
            - For each connected vertex:
                - If it is "UNVISITED":
                    - Change its status to "VISITING."
                    - Update its distance and set its parent to the current vertex.
                    - Add the connected vertex to the queue.
            - Set the processed vertex’s status to "VISITED."
    - **End**:
        - The algorithm completes when the queue is empty.
        - The output shows the order of traversal and the distances from the starting vertex to each vertex.

Depth-First Search (DFS) Algorithm

- **Input**: A graph with vertices and edges, and a starting vertex.
- **Output**: The order of traversal and discovery/finish times for each vertex.
    - **Initialization**:
        - Set all vertices to "UNVISITED."
        - Set all parent nodes to "NONE."
        - Initialize a time counter to 0.
    - **Traversal Process**:
        - For each vertex in the graph:

        - If it is "UNVISITED," call the EXPLORE function on it.
  - **EXPLORE Function**:
    - Set the vertex to "VISITING."
    - Increase the time counter by 1.
    - Record the start (discovery) time.
    - For each connected vertex:
      - If it is "UNVISITED":
        - Set its parent to the current vertex.
        - Call EXPLORE on it.
    - After exploring all connected vertices:
      - Set the vertex's status to "VISITED."
      - Record the end (finish) time and increase the counter by 1.
  - **End**:
    - The algorithm finishes when all vertices are either "VISITED" or unreachable.
    - The output shows the order of traversal and discovery/finish times for each vertex.

---

4. Codes

BFS

```
from collections import deque

def bfs(graph, start):
    # Initialization
    visited = {node: False for node in graph}
    distance = {node: float('inf') for node in graph}
    parent = {node: None for node in graph}

    visited[start] = True
    distance[start] = 0

    queue = deque([start])
    order_of_traversal = []

    # Traversal Process
    while queue:
        current = queue.popleft()
        order_of_traversal.append(current)

        for neighbor in graph[current]:
            if not visited[neighbor]:
                visited[neighbor] = True
                distance[neighbor] = distance[current] + 1
                parent[neighbor] = current
                queue.append(neighbor)

    return order_of_traversal, distance
```

DFS

```
def dfs(graph):
    visited = {node: False for node in graph}
    parent = {node: None for node in graph}
    discovery_time = {}
    finish_time = {}
    time_counter = [0]  # Using list to allow modification inside inner
function

    def explore(node):
        visited[node] = True
        time_counter[0] += 1
        discovery_time[node] = time_counter[0]

        for neighbor in graph[node]:
            if not visited[neighbor]:
                parent[neighbor] = node
                explore(neighbor)

        time_counter[0] += 1
        finish_time[node] = time_counter[0]

    # Traversal Process
    for vertex in graph:
        if not visited[vertex]:
            explore(vertex)

    return discovery_time, finish_time
```

---

5. Results

- **Breadth-First Search (BFS):**
  - **Output:**
    - The order of traversal, indicating the sequence in which vertices are visited starting from the given starting vertex.
    - An array with distances from the starting vertex to each reachable vertex, representing the shortest path in terms of edge count.
    - A parent array to reconstruct paths from the starting vertex to other vertices.
- **Depth-First Search (DFS):**
  - **Output:**
    - The order of traversal, showing the depth-first sequence of vertex visits.
    - Discovery and finish times for each vertex, indicating when each vertex is first encountered and when exploration from it is completed.

        - A parent array to reconstruct the DFS traversal tree or forest, which represents the paths explored.

---

6. Inference

- **Breadth-First Search (BFS)** is effective for solving shortest path problems in unweighted graphs, as it provides the minimum edge count from the starting vertex to each reachable vertex. The traversal order and distance calculations are especially useful in applications requiring level-by-level exploration, such as social networks, network broadcasting, and finding all nodes within a specified distance.
- **Depth-First Search (DFS)** is well-suited for problems requiring full exploration of paths or sequences, as it records both discovery and finish times for each vertex. This makes DFS valuable for applications involving pathfinding, cycle detection, and dependency resolution. The parent array also allows for reconstruction of traversal paths, which is useful in identifying connectivity and analyzing graph structure.
