Traversal Schemes in Weighted Directed Graphs

Aim:

To understand and implement various traversal schemes for a weighted directed graph.

Logic:

There are two different traversal schemes to traverse a weighted directed graph:

BFS (Breath First Search):

In Breadth-First Search, we explore all the immediate neighbors of a starting point before moving further out.

DFS (Depth First Search):

In Depth-First Search, we go as far as possible down one path before backing up to explore other paths. It's like following a tunnel until it ends, then backtracking to check other tunnels.

Algorithm:

BFS (Breath First Search):

1) Create an empty queue and add the starting node to it.
2) Mark the starting node as visited to avoid revisiting.
3) While the queue is not empty:
    3.1) Remove the first node from the queue.
    3.2) For each unvisited neighbor of this node:
        3.2.1) Mark the neighbor as visited.
        3.2.2) Add the neighbor to the queue.

4) End

DFS (Depth First Search):

1) Create a stack and add the starting node to it.
2) Mark the starting node as visited.
3) While the stack is not empty:
    3.1) Remove the last node from the stack.
    3.2) For each unvisited neighbor of this node:
        3.2.1) Mark the neighbor as visited.
        3.2.2) Push the neighbor onto the stack.
4) End

Code:

```
class Graph:

  def __init__(self, num_vertices):

    self.num_vertices = num_vertices

    # Initialize adjacency matrix with None for no direct edge

    self.graph = [[None] * num_vertices for _ in range(num_vertices)]


  # Method to add a directed, weighted edge to the graph

  def add_edge(self, u, v, weight):

    self.graph[u][v] = weight  # Only one direction for directed graph


    def bfs(self, start_vertex):

    visited = set()  # Set to keep track of visited nodes

    queue = [start_vertex]  # Queue to support BFS traversal

    bfs_result = []
```

```
while queue:
    vertex = queue.pop(0)
    if vertex not in visited:
        visited.add(vertex)
        bfs_result.append(vertex)

        # Add all unvisited neighbors to the queue
        for neighbor in range(self.num_vertices):
            # Check for a valid edge and if neighbor is unvisited
            if self.graph[vertex][neighbor] is not None and neighbor not in visited:
                queue.append(neighbor)

return bfs_result

def dfs(self, start_vertex):
visited = set()  # Set to keep track of visited nodes
stack = [start_vertex]  # Stack to support DFS traversal
dfs_result = []

while stack:
    vertex = stack.pop()
```

```
        if vertex not in visited:
            visited.add(vertex)
            dfs_result.append(vertex)

            # Add all unvisited neighbors to the stack
            for neighbor in range(self.num_vertices - 1, -1, -1):  # Reverse order to maintain stack behavior
                if self.graph[vertex][neighbor] is not None and neighbor not in visited:
                    stack.append(neighbor)

    return dfs_result

if __name__ == "__main__":
    num_vertices = 4  # 0, 1, 2, 3
    g = Graph(num_vertices)

    # Adding directed, weighted edges to the graph
    edges = [
        (0, 1, 3),   # Edge from 0 to 1 with weight 3
        (0, 2, 1),   # Edge from 0 to 2 with weight 1
        (1, 3, 2),   # Edge from 1 to 3 with weight 2
        (2, 3, 1)    # Edge from 2 to 3 with weight 1
    ]
```

```
for u, v, weight in edges:
    g.add_edge(u, v, weight)


print("Breadth First Search (starting from vertex 0):", g.bfs(0))
print("Depth First Search (starting from vertex 0):", g.dfs(0))
```

Result:

```
[Running] python -u "f:\Academics\sem 3\DS LAB\Experiments\bfs,dfs.py"
Breadth First Search: [0, 1, 2, 3]
Depth First Search: [0, 1, 3, 2]

[Done] exited with code=0 in 0.157 seconds
```

Inference:

BFS explores nodes level by level, resulting in the order [0, 1, 2, 3], while DFS goes deep along branches first, yielding [0, 1, 3, 2]. This illustrates their fundamental difference: BFS prioritizes breadth of exploration, whereas DFS focuses on depth.