23CCE201 Data Structures

Name : Tarun Sri Vathsan K

Roll No : CB.EN.U4CCE23058

AIM :

- To implement various traversal schemes on weighted directed graphs.

LOGIC :

In Breadth First Search, the traversal starts at the root vertex or a chosen starting point and explores all of its direct neighbors before moving on to their neighbors. This traversal method uses a queue to keep track of vertices to visit next, ensuring all vertices are visited level by level. Each vertex is marked as visited to prevent revisiting, and newly discovered vertices are enqueued until all reachable vertices have been processed.

In Depth First Search, the traversal starts at the root or a specified starting vertex and goes as deep as possible along each branch before backtracking. This approach can be implemented using recursion or a stack to keep track of the vertices in the current path. Vertices are marked as visited when first encountered to avoid revisiting, and unvisited adjacent vertices are explored recursively or pushed onto the stack. DFS is suitable for scenarios where exploring deeper connections or backtracking is necessary, such as finding paths or detecting cycles.

ALGORITHM :

- **Breadth First Search (BFS) :**

1. Get the total number of vertices,edges,neighbours and start vertex from the user.
2. Initialize a queue and enqueue the starting vertex.
3. Create a set to track visited vertices and add the starting vertex to it.
4. Initialize a list to store the traversal path and add the starting vertex to it.
5. While the queue is not empty, dequeue a vertex and set it as the current vertex.
6. For each neighbor of the current vertex, if the neighbor has not been visited, mark it as visited by adding it to the visited set, enqueue the neighbor, and append it to the path list.
7. Repeat the process until the queue is empty.
8. Return or display the path list to show the order of traversal.

- **Depth First Search (DFS) :**

1. Get the total number of vertices,edges,neighbours and start vertex from the user.
2. If visited is not provided, initialize visited as an empty set.
3. If path is not provided, initialize path as an empty list.
4. Add the starting vertex to the visited set to mark it as visited.
5. Append start to the path list to record the traversal path.
6. For each neighbor in the adjacency list of the current vertex start,Check if neighbor is not in visited.
7. If neighbor is not visited,Recursively call DFS with the neighbor as the current visited set, and the current path list.
8. The recursion automatically backtracks when all neighbors of a vertex have been visited.
9. Return the path list, which now contains the complete order of vertices visited during the DFS traversal.

CODE :

- **Breadth First Search (BFS) :**

```
from collections import deque
# BFS Traversal with path
def BFS(graph, start):
    queue = deque([start])
    visited = set()
    visited.add(start)

    path = [start]  # To store the traversal path

    while queue:
        vertex = queue.popleft()
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
                path.append(neighbor)

    return path
# Function to create input graph
def input_graph():
    graph = {}
    num_vertexs = int(input("Enter the number of vertices in the graph: "))

    for _ in range(num_vertexs):
        vertex = input("Enter the vertex: ")
        graph[vertex] = []
```

```
        num_edges = int(input(f"Enter the number of edges from vertex
{vertex}: "))
        for _ in range(num_edges):
            neighbor = input("Enter the neighbor: ")
            graph[vertex].append(neighbor)

            # For undirected graphs, add the reverse edge
            if not directed:
                if neighbor not in graph:
                    graph[neighbor] = []
                graph[neighbor].append(vertex)

    return graph, directed
# Main Execution
while True:
    print("1. BFS 2. exit")
    ch=int(input("enter your choice:"))
    if ch==1:
        graph, directed = input_graph()
        start_vertex = input("Enter the start vertex: ")

        print("\nBFS Traversal")
        bfs_path = BFS(graph, start_vertex)
        print("Path taken in BFS: ", " -> ".join(bfs_path))
    elif ch==2:
        print("terminating")
        break
```

```
        else:
            print("invalid option")
```

- **Depth First Search (DFS) :**

```
from collections import deque
# DFS Traversal with path
def DFS(graph, start, visited=None, path=None):
    if visited is None:
        visited = set()
    if path is None:
        path = []

    visited.add(start)
    path.append(start)

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

    return path

# Function to create input graph
def input_graph():
    graph = {}
    num_vertexs = int(input("Enter the number of vertexs in the graph: "))
    directed = input("Is the graph directed? (yes/no): ").strip().lower() == 'yes'
```

```
    for _ in range(num_vertexs):
        vertex = input("Enter the vertex: ")
        graph[vertex] = []
        num_edges = int(input(f"Enter the number of edges from vertex {vertex}:
"))
        for _ in range(num_edges):
            neighbor = input("Enter the neighbor: ")
            graph[vertex].append(neighbor)

            # For undirected graphs, add the reverse edge
            if not directed:
                if neighbor not in graph:
                    graph[neighbor] = []
                graph[neighbor].append(vertex)

    return graph, directed
# Main Execution
while True:
    print("1. DFS 2. exit")
    ch=int(input("enter your choice:"))
    if ch==1:
        graph, directed = input_graph()
        start_vertex = input("Enter the start vertex: ")

        print("\nDFS Traversal")
        dfs_path = DFS(graph, start_vertex)
```

```
        print("Path taken in BFS: ", " -> ".join(dfs_path))
    elif ch==2:
        print("terminating")
        break
    else:
        print("invalid option")
```

RESULTS :

- **Breadth First Search (BFS) :**

```
1. BFS 2. exit
enter your choice:1
Enter the number of vertices in the graph: 6
Is the graph directed? (yes/no): yes
Enter the vertex: 1
Enter the number of edges from vertex 1: 2
Enter the neighbor: 4
Enter the neighbor: 5
Enter the vertex: 2
Enter the number of edges from vertex 2: 3
Enter the neighbor: 1
Enter the neighbor: 5
Enter the neighbor: 6
Enter the vertex: 3
Enter the number of edges from vertex 3: 2
Enter the neighbor: 2
Enter the neighbor: 4
Enter the vertex: 4
Enter the number of edges from vertex 4: 1
Enter the neighbor: 5
Enter the vertex: 5
Enter the number of edges from vertex 5: 2
Enter the neighbor: 4
Enter the neighbor: 2
Enter the vertex: 6
Enter the number of edges from vertex 6: 1
Enter the neighbor: 4
Enter the start vertex: 3

BFS Traversal
Path taken in BFS:  3 -> 2 -> 4 -> 1 -> 5 -> 6
1. BFS 2. exit
enter your choice:2
terminating
```

- **Depth First Search (DFS) :**

```
1. DFS 2. exit
enter your choice:1
Enter the number of vertices in the graph: 6
Is the graph directed? (yes/no): yes
Enter the vertex: 1
Enter the number of edges from vertex 1: 2
Enter the neighbor: 4
Enter the neighbor: 5
Enter the vertex: 2
Enter the number of edges from vertex 2: 3
Enter the neighbor: 1
Enter the neighbor: 5
Enter the neighbor: 6
Enter the vertex: 3
Enter the number of edges from vertex 3: 2
Enter the neighbor: 2
Enter the neighbor: 4
Enter the vertex: 4
Enter the number of edges from vertex 4: 1
Enter the neighbor: 5
Enter the vertex: 5
Enter the number of edges from vertex 5: 2
Enter the neighbor: 4
Enter the neighbor: 2
Enter the vertex: 6
Enter the number of edges from vertex 6: 1
Enter the neighbor: 4
Enter the start vertex: 3

DFS Traversal
Path taken in BFS:  3 -> 2 -> 1 -> 4 -> 5 -> 6
1. DFS 2. exit
enter your choice:2
terminating
```

INFERENCES :

The breadth first search and depth first search traversal schemes both has distinct approaches. The BFS traverses through all the neighboring vertices of a given vertex then moves along by picking any one of the neighboring vertices as the root vertex whereas in DFS, any one of the neighboring vertex is traversed through the root vertex. Both the traversal methods visit all the vertices in a weighted directed graph.