AIR UNIVERSITY

DESIGN ANALYSIS AND ALGORITHM

NAME : Maham Sheikh

ROLL NO : 220961

ASSIGNMENT NO :03

BS AI (4- A)

Depth First Search (DFS)

```
def dfs(graph, start, visited=None):
if visited is None:
visited = set()
visited.add(start)
print(start, end=' ')
for neighbor in graph[start]:
if neighbor not in visited:
dfs(graph, neighbor, visited)
return visited


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

Time Complexity:

$$O(V+E)$$

V is the number of vertices.

E is the number of edges.

Breadth First Search (BFS)

```
from collections import deque

# Breadth First Search (BFS)

def bfs(graph, start):

visited = set()

queue = deque([start])

visited.add(start)

while queue:

vertex = queue.popleft()

print(vertex, end=' ')

for neighbor in graph[vertex]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

# Example usage

print("\nBFS traversal starting from 'A':")

bfs(graph, 'A')
```

Time Complexity:

$$O(V+E)$$

V is the number of vertices.

E is the number of edges.

DIJKSTRA’S ALGORITHM

```
import heapq
def dijkstra(graph, start):
distances = {vertex: float('infinity') for vertex in graph}
distances[start] = 0
priority_queue = [(0, start)]
while priority_queue:
current_distance, current_vertex = heapq.heappop(priority_queue)
if current_distance > distances[current_vertex]:
continue
for neighbor, weight in graph[current_vertex].items():
distance = current_distance + weight
if distance < distances[neighbor]:
distances[neighbor] = distance
heapq.heappush(priority_queue, (distance, neighbor))
return distances
```

TIME COMPLEXIY:

$$O((V+E)\log V)$$

V is the number of vertices.

E is the number of edges.