NASHRAH UL EMAAN

220983

BSAI-4-A

Depth First Search (DFS)

```
def dfs(graph, start):
    visited = set()
    stack = [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited
```

Time complexity of DFS:

O(V + E)

V is the number of vertices and **E** is the number of edges.

Breadth First Search (BFS) implementation

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start

    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited
```

Time complexity of BFS:

O(V + E)

where V is the number of vertices and E is the number of edges.

Dijkstra's Algorithm implementation

```
import heapq

def dijkstra(graph, start):
    distances = {vertex: float('infinity') for vertex in graph}  # Dictionary to store shortest distances
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_distance, current_vertex = heapq.heappop(pq)  # Pop vertex with smallest tentative distance

        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(pq, (distance, neighbor))
    return distances
```

Time complexity of Dijkstra's Algorithm:

O((V + E) * log(V))

where V is the number of vertices and E is the number of edges.