UPES

UNIVERSITY OF THE FUTURE

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
DEHRADUN

Advanced Algorithms

Lab Experiment 3-4

MTECH-COMPUTER SCIENCE ENGINEERING
CYBER SECURITY AND FORENSICS

Name: Jigesh Sheoran
SAP ID: 590025428

Aim:

To implement Breadth First Search (BFS) for finding the shortest path in unweighted graphs and Dijkstra’s algorithm for finding the shortest path in weighted graphs, and to compare their performance.

Theory:

1. BFS for Shortest Path (Unweighted Graph)

Breadth First Search explores the graph level by level starting from a source node.
In an unweighted graph, BFS guarantees the shortest path because all edges have equal cost (assumed as 1).
The first time a node is visited is through the shortest path.

Time Complexity:

$$O(V + E)$$

2. Dijkstra’s Algorithm (Weighted Graph)

Dijkstra’s algorithm finds the shortest path from a source node to all other nodes in a weighted graph with non-negative edge weights.
It uses a greedy approach and a priority queue to always expand the node with the minimum tentative distance.

Time Complexity (using heap):

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

Algorithm

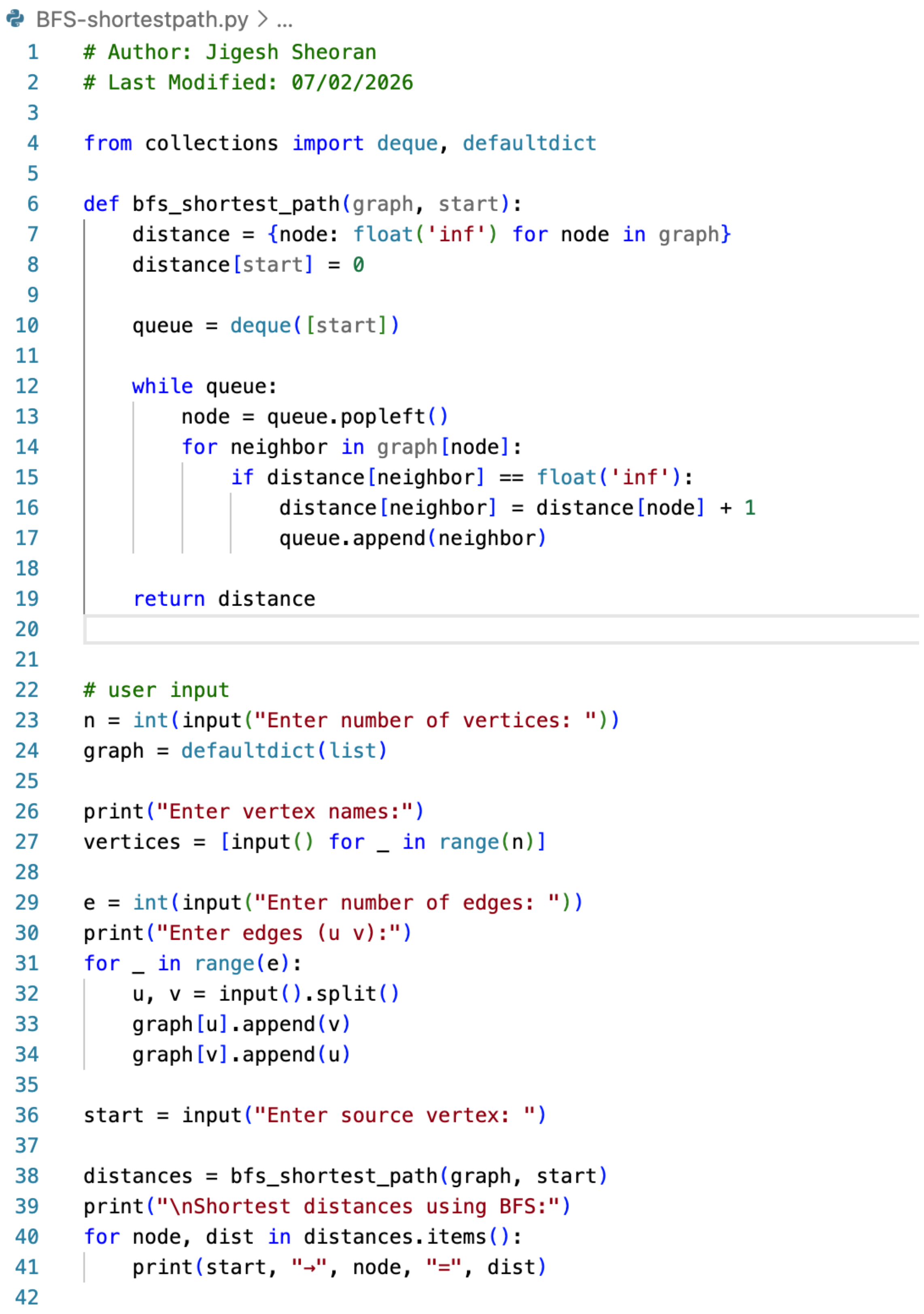
BFS (Unweighted Graph)

1. Initialize distance of all nodes as infinity.
2. Set source distance = 0.
3. Use a queue to explore neighbors.
   Update distance of unvisited neighbors.
4. Continue until all reachable nodes are visited.

Dijkstra (Weighted Graph)

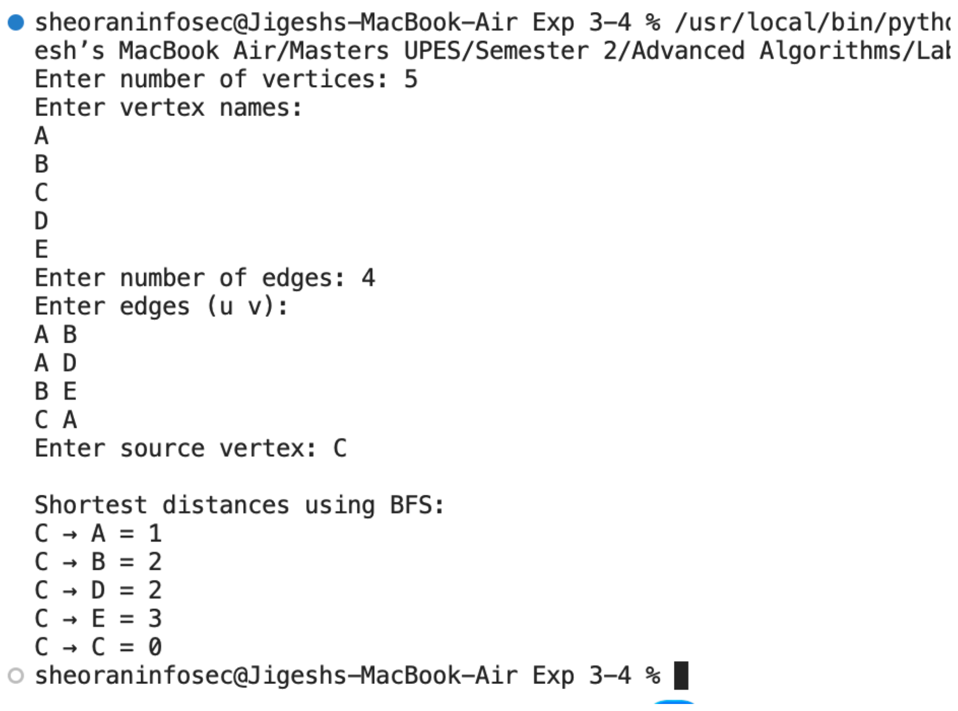
1. Initialize distances as infinity.
2. Set source distance = 0.
3. Use a min-heap (priority queue).
4. Relax edges and update distances.
   Continue until all nodes are processed.

Code: BFS (Unweighted Graph)


BFS-shortestpath.py > ...

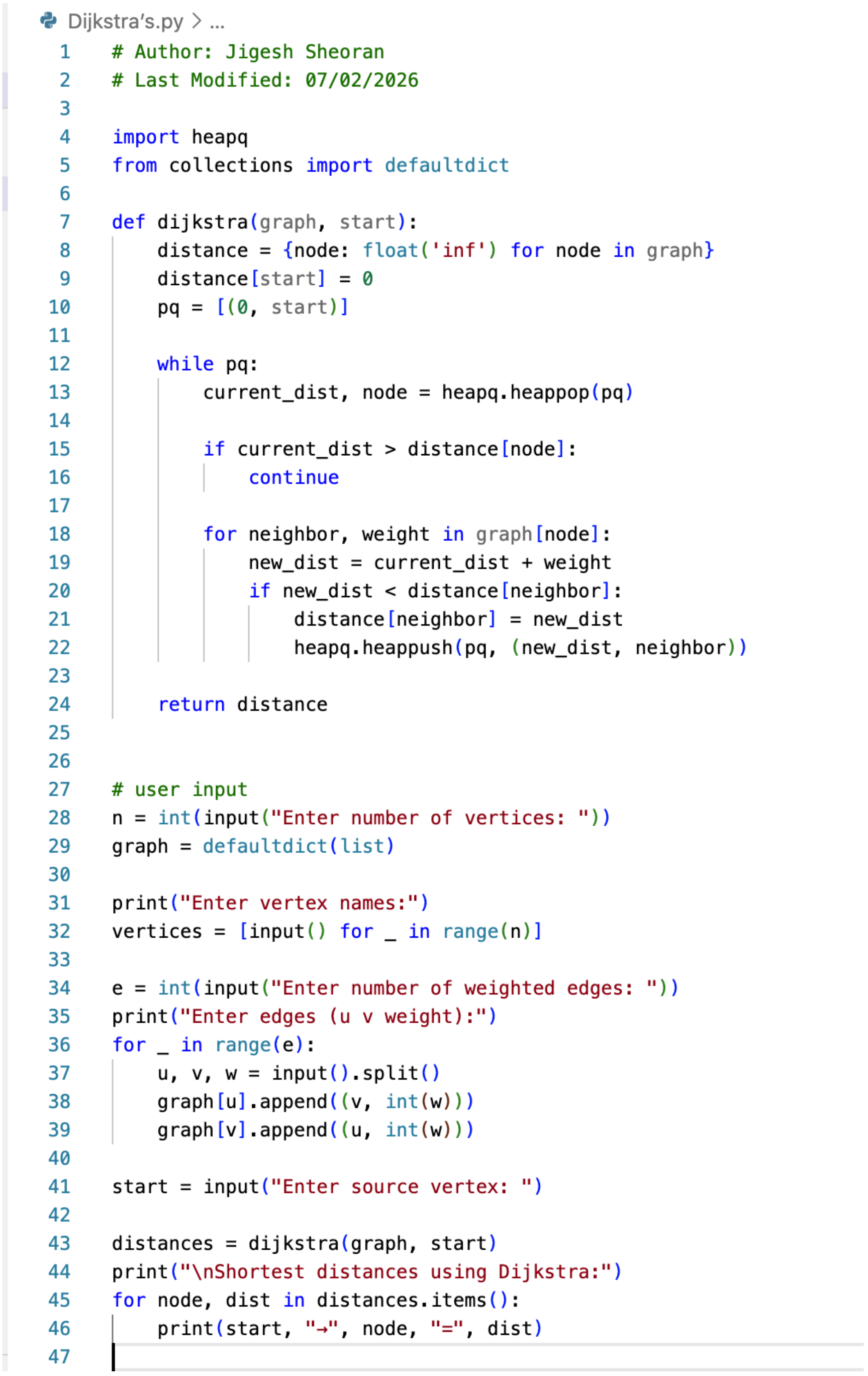
```python
# Author: Jigesh Sheoran
# Last Modified: 07/02/2026

from collections import deque, defaultdict

def bfs_shortest_path(graph, start):
    distance = {node: float('inf') for node in graph}
    distance[start] = 0

    queue = deque([start])

    while queue:
        node = queue.popleft()
        for neighbor in graph[node]:
            if distance[neighbor] == float('inf'):
                distance[neighbor] = distance[node] + 1
                queue.append(neighbor)

    return distance


# user input
n = int(input("Enter number of vertices: "))
graph = defaultdict(list)

print("Enter vertex names:")
vertices = [input() for _ in range(n)]

e = int(input("Enter number of edges: "))
print("Enter edges (u v):")
for _ in range(e):
    u, v = input().split()
    graph[u].append(v)
    graph[v].append(u)

start = input("Enter source vertex: ")

distances = bfs_shortest_path(graph, start)
print("\nShortest distances using BFS:")
for node, dist in distances.items():
    print(start, "→", node, "=", dist)
```

Output:



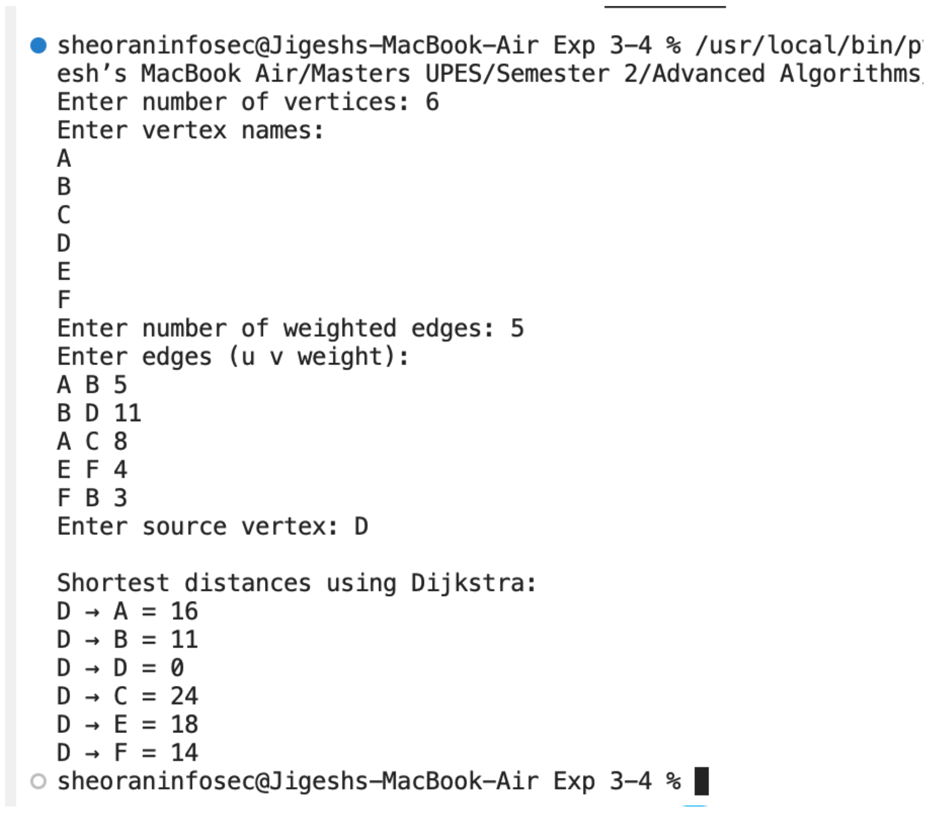
```
● sheoraninfosec@Jigeshs-MacBook-Air Exp 3-4 % /usr/local/bin/pytho
  esh's MacBook Air/Masters UPES/Semester 2/Advanced Algorithms/Lal
  Enter number of vertices: 5
  Enter vertex names:
  A
  B
  C
  D
  E
  Enter number of edges: 4
  Enter edges (u v):
  A B
  A D
  B E
  C A
  Enter source vertex: C

  Shortest distances using BFS:
  C → A = 1
  C → B = 2
  C → D = 2
  C → E = 3
  C → C = 0
○ sheoraninfosec@Jigeshs-MacBook-Air Exp 3-4 %
```

Code: Dijkstra (Weighted Graph)



```python
# Author: Jigesh Sheoran
# Last Modified: 07/02/2026

import heapq
from collections import defaultdict

def dijkstra(graph, start):
    distance = {node: float('inf') for node in graph}
    distance[start] = 0
    pq = [(0, start)]

    while pq:
        current_dist, node = heapq.heappop(pq)

        if current_dist > distance[node]:
            continue

        for neighbor, weight in graph[node]:
            new_dist = current_dist + weight
            if new_dist < distance[neighbor]:
                distance[neighbor] = new_dist
                heapq.heappush(pq, (new_dist, neighbor))

    return distance


# user input
n = int(input("Enter number of vertices: "))
graph = defaultdict(list)

print("Enter vertex names:")
vertices = [input() for _ in range(n)]

e = int(input("Enter number of weighted edges: "))
print("Enter edges (u v weight):")
for _ in range(e):
    u, v, w = input().split()
    graph[u].append((v, int(w)))
    graph[v].append((u, int(w)))

start = input("Enter source vertex: ")

distances = dijkstra(graph, start)
print("\nShortest distances using Dijkstra:")
for node, dist in distances.items():
    print(start, "→", node, "=", dist)

```

Output:


```
● sheoraninfosec@Jigeshs-MacBook-Air Exp 3-4 % /usr/local/bin/p
  esh's MacBook Air/Masters UPES/Semester 2/Advanced Algorithms
  Enter number of vertices: 6
  Enter vertex names:
  A
  B
  C
  D
  E
  F
  Enter number of weighted edges: 5
  Enter edges (u v weight):
  A B 5
  B D 11
  A C 8
  E F 4
  F B 3
  Enter source vertex: D

  Shortest distances using Dijkstra:
  D → A = 16
  D → B = 11
  D → D = 0
  D → C = 24
  D → E = 18
  D → F = 14
○ sheoraninfosec@Jigeshs-MacBook-Air Exp 3-4 %
```

Observation Table:

Algorithm	Graph Type	Time Complexity	Suitable When
BFS	Unweighted	O(V + E)	All edges have equal cost
Dijkstra	Weighted	O((V+E) log V)	Non-negative weights

Result:

1. BFS finds shortest path based on levels (edge count).
2. Dijkstra finds shortest path based on minimum accumulated weight.

BFS is efficient for unweighted graphs, while Dijkstra is necessary when edge weights are involved.

—----------------------------END OF DOCUMENT —-------------------------