PRACTICUM REPORT

ALGORITHM AND DATA STRUCTURES

MODUL 11 : Shortest Path Problem and Traveling Salesman Problem

Disusun Oleh :

Onic Agustino

L200234275

X

INFORMATICS ENGINEERING

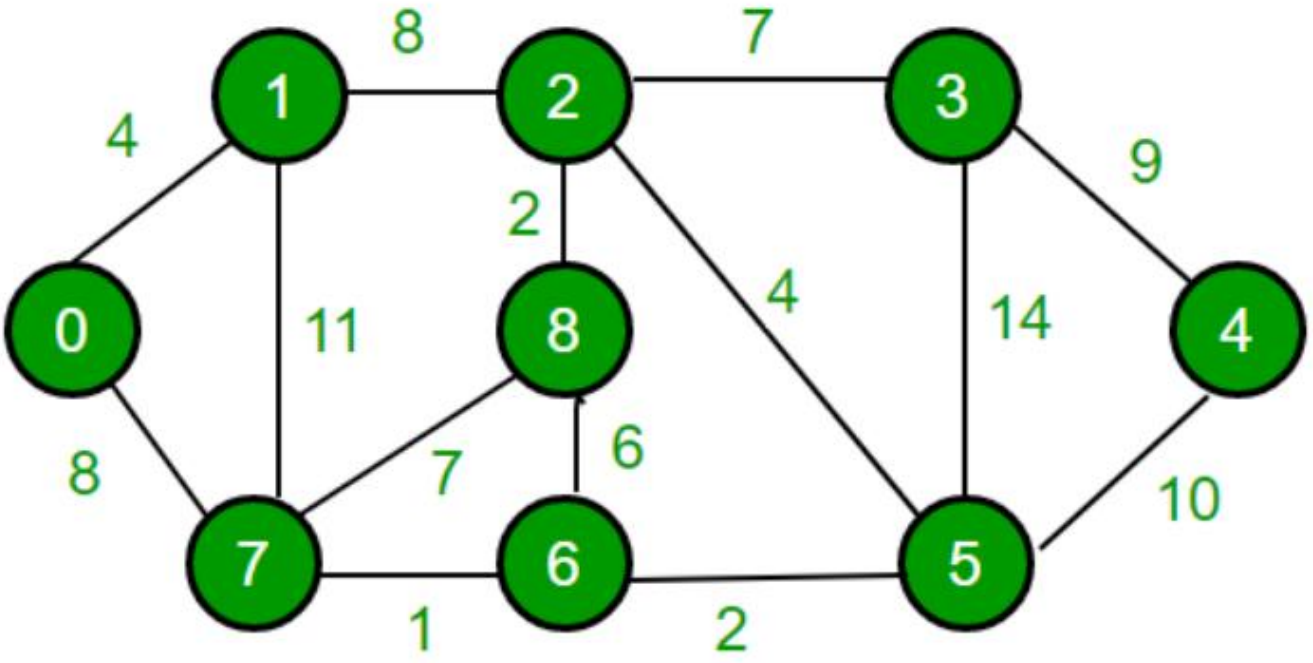
FACULTY OF COMMUNICATION AND INFORMATICS

UNIVERSITAS MUHAMMADIYAH SURAKARTA

YEARS 2024/2025

1.11 Questions

1.



Solve the shortest path problem from the graph above using Dijkstra algorithm from node 0 to node 4 with Python.

- Program Code

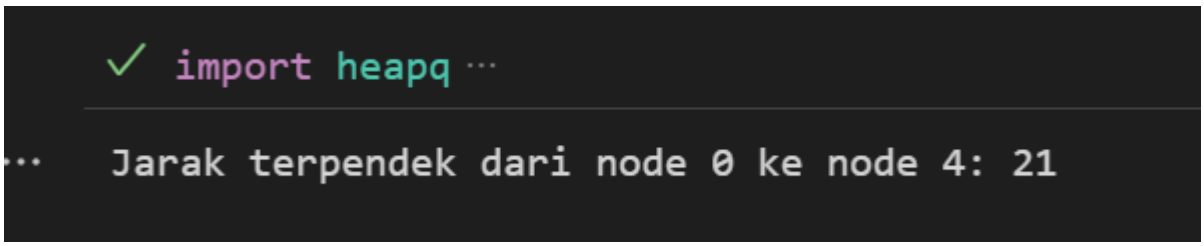
```
import heapq


def dijkstra(graph, start):
    n = len(graph)
    distances = [float('inf')] * n
    distances[start] = 0
    priority_queue = [(0, start)]
    prev = [None] * n  # Untuk menyimpan jalur terpendek


    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)
        if current_distance > distances[current_node]:
```

```
            continue
        for neighbor in range(n):
            if graph[current_node][neighbor] != 0:
                distance = current_distance + graph[current_node][neighbor]
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    prev[neighbor] = current_node
                    heapq.heappush(priority_queue, (distance, neighbor))
    return distances, prev

graph = [
    [0, 4, 0, 0, 0, 0, 0, 8, 0],  # 0
    [4, 0, 8, 0, 0, 0, 0, 11, 0], # 1
    [0, 8, 0, 7, 0, 4, 0, 0, 2],  # 2
    [0, 0, 7, 0, 9, 14, 0, 0, 0], # 3
    [0, 0, 0, 9, 0, 10, 0, 0, 0], # 4
    [0, 0, 4, 14, 10, 0, 2, 0, 0],# 5
    [0, 0, 0, 0, 0, 2, 0, 1, 6],  # 6
    [8, 11, 0, 0, 0, 0, 1, 0, 7], # 7
    [0, 0, 2, 0, 0, 0, 6, 7, 0]   # 8
]

start_node = 0
end_node = 4

distances, prev = dijkstra(graph, start_node)

print(f"Jarak terpendek dari node {start_node} ke node {end_node}: {distances[end_node]}")
```

Code 1.1 *code about of Dijkstra*

*This code implements **Dijkstra's algorithm** using **heapq** to find the shortest path from a starting node to all other nodes in a weighted graph represented as an adjacency matrix. By using a **priority queue**, the algorithm ensures that each node is processed in order of the shortest known distance, speeding up the search for the optimal solution.*

- **Practicum Result Screenshot**



Gambar 1.2 *this picture display output form code above.*

2. Modify the Dijkstra algorithm that has been given by adding code to print the shortest path passes using python.

- Program code

```
def get_path(prev, target):
    path = []
    while target is not None:
        path.append(target)
        target = prev[target]
    return path[::-1]

path = get_path(prev, end_node)
print(f"Jalur terpendek: {path}")
```

Gambar 2.1 the code

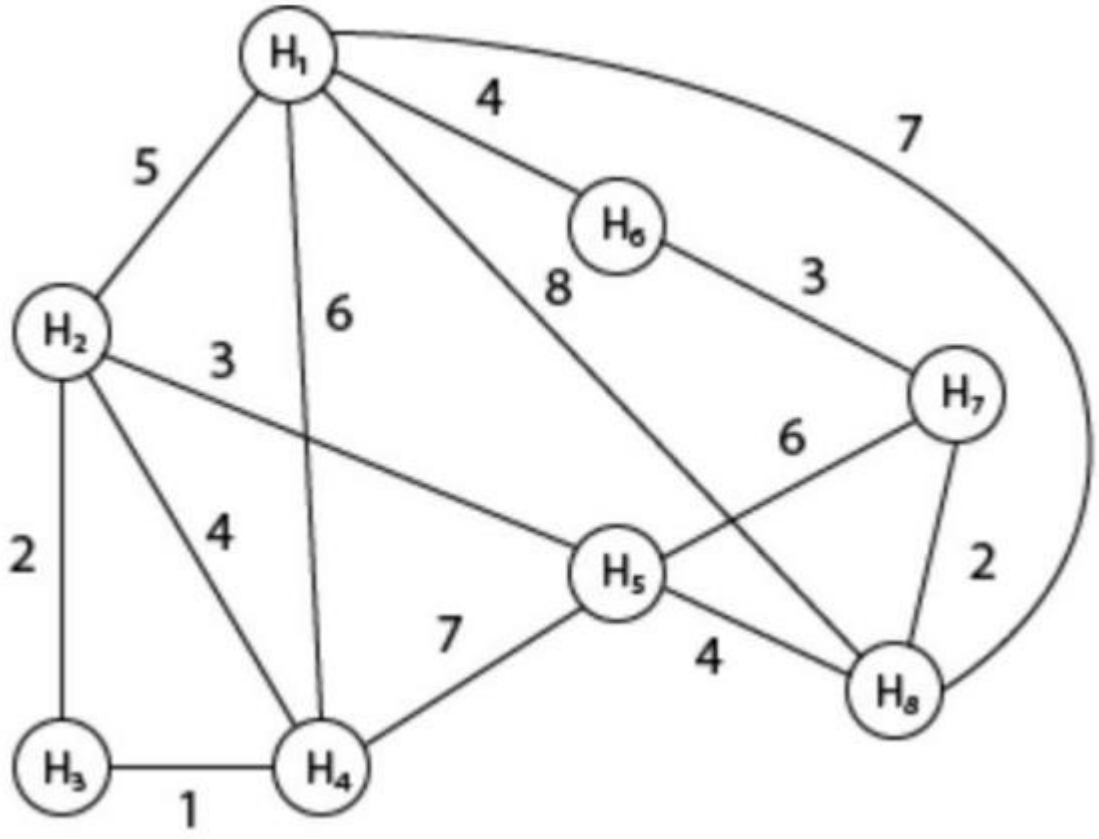
*This code implements **Dijkstra's algorithm** to find the shortest path in a weighted graph, represented as an adjacency matrix. It uses a **priority queue** (heapq) to efficiently determine the shortest known distance to each node.*

- Practicum Results Report

```
✓ import heapq ···

···  Jarak terpendek dari node 0 ke node 4: 21
     Jalur terpendek: [0, 7, 6, 5, 4]
```

Gambar 2.1 the results code

3.



Solve the traveling salesman problem for the graph above using a greedy algorithm with Python.

- Program code

```
def greedy_tsp(graph, start):
    n = len(graph)
    visited = [False] * n
```

```
    path = [start]
    total_cost = 0
    current = start
    visited[current] = True

    for _ in range(n - 1):
        next_node = None
        min_dist = float('inf')
        for j in range(n):
            if not visited[j] and 0 < graph[current][j] < min_dist:
                min_dist = graph[current][j]
                next_node = j
        if next_node is not None:
            path.append(next_node)
            total_cost += min_dist
            visited[next_node] = True
            current = next_node

    # Kembali ke titik awal
    if graph[current][start] > 0:
        path.append(start)
        total_cost += graph[current][start]
    else:
        # Tidak bisa kembali ke awal (bukan siklus Hamiltonian)
        return path, float('inf')

    return path, total_cost

# Matriks adjacency sesuai gambar
graph = [
    # H1 H2 H3 H4 H5 H6 H7 H8
    [ 0, 5, 0, 6, 8, 4, 0, 7],  # H1
```

```
    [ 5, 0, 2, 4, 3, 0, 0, 0],  # H2
    [ 0, 2, 0, 1, 0, 0, 0, 0],  # H3
    [ 6, 4, 1, 0, 7, 0, 0, 0],  # H4
    [ 8, 3, 0, 7, 0, 0, 6, 4],  # H5
    [ 4, 0, 0, 0, 0, 0, 3, 0],  # H6
    [ 0, 0, 0, 0, 6, 3, 0, 2],  # H7
    [ 7, 0, 0, 0, 4, 0, 2, 0]   # H8
]

start_node = 0  # H1
path, cost = greedy_tsp(graph, start_node)

node_names = ['H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8']
path_names = [node_names[i] for i in path]

print(f"Greedy TSP dari {node_names[start_node]}:")
print(f"Jalur: {' -> '.join(path_names)}")
print(f"Total jarak: {cost}")
```

Gambar 3.1 the code.

This code implements a **Greedy Algorithm** for solving the **Traveling Salesperson Problem (TSP)** using an adjacency matrix representation of a weighted graph.

- **Practicum Results Report**

```
Greedy TSP dari H1:
Jalur: H1 -> H6 -> H7 -> H8 -> H5 -> H2 -> H3 -> H4 -> H1
Total jarak: 25
```

Gambar 3.2 the results.

4. Implement a modified greedy algorithm to complete the TSP starting from all starting points. Run the greedy algorithm from every possible starting node, then compare and choose the resulting path with the shortest total distance using Python

- Program code

```
#nomer 4
# Kita masih menggunakan fungsi greedy_tsp dan graph_h dari Latihan 3

# Inisialisasi variabel untuk menyimpan hasil terbaik
best_path = None
min_cost = float('inf')

n_nodes = len(graph)

print("\n--- Hasil Latihan 4 (Mencoba semua titik awal) ---")

# 1. Jalankan algoritma dari setiap node sebagai titik awal
for i in range(n_nodes):
    current_path, current_cost = greedy_tsp(graph, i)
    print(f"Mencoba mulai dari node {i}: Biaya = {current_cost}, Jalur = {current_path}")

    # 2. Bandingkan dan pilih jalur dengan biaya terpendek
    if current_cost < min_cost:
        min_cost = current_cost
        best_path = current_path

# 3. Cetak hasil terbaik setelah semua kemungkinan dicoba
print("\n------------------------------------------------")
print("Hasil Greedy TSP Terbaik setelah mencoba semua titik awal:")
print(f"Jalur Terpendek: {best_path}")
print(f"Total Biaya Minimum: {min_cost}")
```

Gambar 4.1 the code.

This code enhances the Greedy Traveling Salesperson Problem (TSP) algorithm by testing every possible starting node to find the best path with the lowest cost.

- Practicum Results Report

```
--- Hasil Latihan 4 (Mencoba semua titik awal) ---
Mencoba mulai dari node 0: Biaya = 25, Jalur = [0, 5, 6, 7, 4, 1, 2, 3, 0]
Mencoba mulai dari node 1: Biaya = 25, Jalur = [1, 2, 3, 0, 5, 6, 7, 4, 1]
Mencoba mulai dari node 2: Biaya = inf, Jalur = [2, 3, 1, 4, 7, 6, 5, 0]
Mencoba mulai dari node 3: Biaya = 25, Jalur = [3, 2, 1, 4, 7, 6, 5, 0, 3]
Mencoba mulai dari node 4: Biaya = 25, Jalur = [4, 1, 2, 3, 0, 5, 6, 7, 4]
Mencoba mulai dari node 5: Biaya = 25, Jalur = [5, 6, 7, 4, 1, 2, 3, 0, 5]
Mencoba mulai dari node 6: Biaya = 25, Jalur = [6, 7, 4, 1, 2, 3, 0, 5, 6]
Mencoba mulai dari node 7: Biaya = 28, Jalur = [7, 6, 5, 0, 1, 2, 3, 4, 7]

------------------------------------------------
Hasil Greedy TSP Terbaik setelah mencoba semua titik awal:
Jalur Terpendek: [0, 5, 6, 7, 4, 1, 2, 3, 0]
Total Biaya Minimum: 25
```

Gambar 4.2 the results.