**PRACTICUM REPORT**

**ALGORITHM AND DATA STRUCTURES**

**MODUL 11 : Shortest Path Problem and Traveling Salesman Problem**



**Disusun Oleh :**

**Onic Agustino**

**L200234275**

**X**

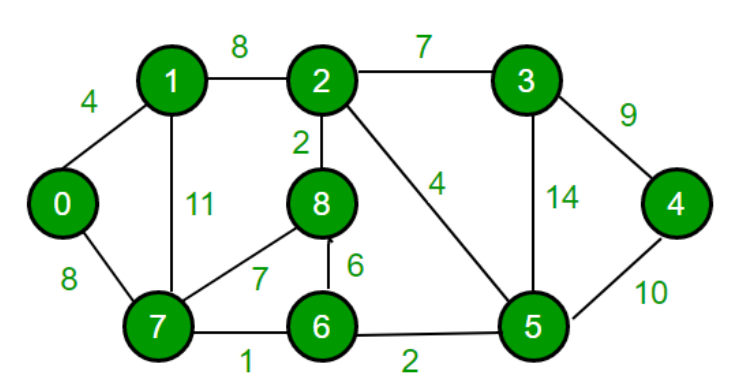
**INFORMATICS ENGINEERING**

**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.

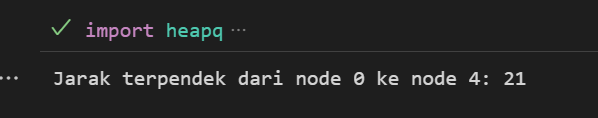
**- Program Code**

|  |
| --- |
| *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.

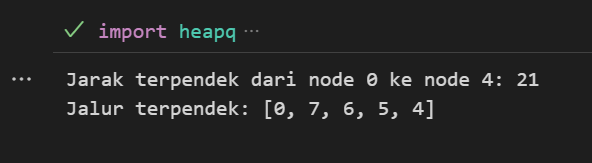
**- Program code**

|  |
| --- |
| *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}")* |

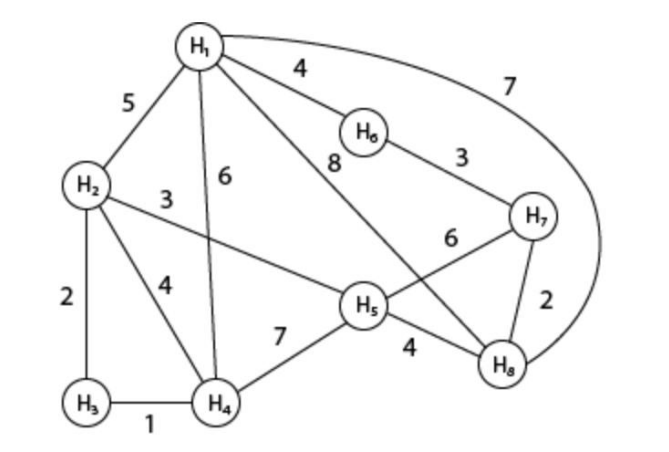
Gambar 2.1 the code

|  |
| --- |
| *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**

****

Gambar 2.1 the results code

3. 

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

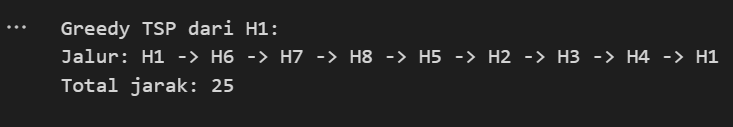
- **Program code**

|  |
| --- |
| *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**

****

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

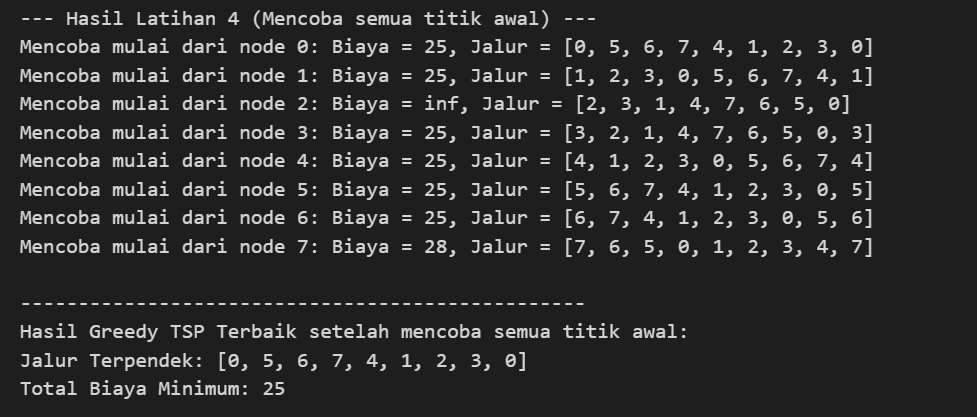
**- Program code**

|  |
| --- |
| *#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**

****

Gambar 4.2 the results.