Tugas Implementasi Shortest Path & MST

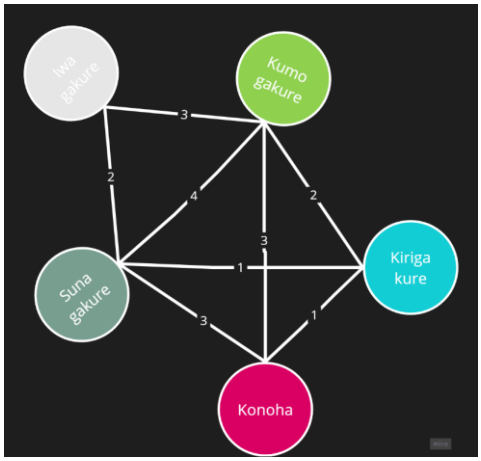
5054231021 - Abdan Hafidz

A. Implementasi Shortest Path

🦊 Naruto yang sedang berada di Konoha ingin menemui Deidara yang ada di Iwagakure. Untuk mencapai Iwagakure, Naruto bisa melewati rute yang berbentuk graph seperti gambar di bawah. Naruto harus sampai cepat di Iwagakure sebelum Deidara pergi ke desa lain. Tapi karena malas, Naruto meminta bantuanmu untuk menemukan jalan tercepatnya menggunakan algoritma djikstra!

Requirements:

1. Jawab manual menggunakan table yang sudah dijelaskan waktu sesi tutorial
2. Implementasi Code dan penjelasan
3. Visualisasi shortest path
4. Format PDF

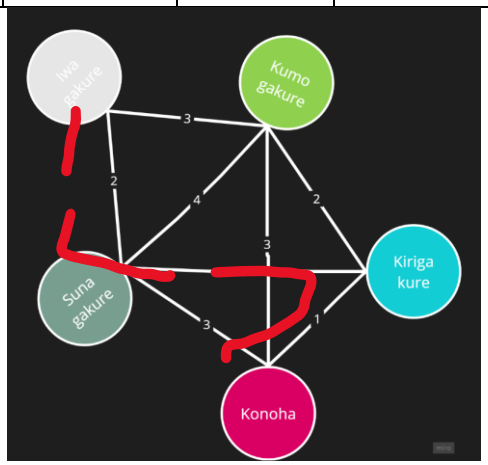


Djikstra

	Konoha	*Sunagakure*	*Iwagakure*	*Kumogakure*	*Kirigakure*
Konoha	INF	3	INF	3	1
Sunagakure	3	INF	2	4	1
Iwagakure	INF	2	INF	3	INF
Kumogakure	3	4	3	INF	2
Kirigakure	1	1	INF	2	INF

	Konoha	*Sunagakure*	*Iwagakure*	*Kumogakure*	*Kirigakure*	*Choiced*
Konoha	INF	3	INF	3	1	**Kirigakure**
Sunagakure	3	INF	2	4	1	**Iwagakure**
Iwagakure	INF	2	INF	3	INF	
Kumogakure	3	4	3	INF	2	
Kirigakure	1	1	INF	2	INF	**Sunagakure**

	Konoha	*Sunagakure*	*Iwagakure*	*Kumogakure*	*Kirigakure*	*Choiced*
Konoha	INF	3	4	3	1	**Kirigakure**
Sunagakure	3	INF	2	4	1	**Iwagakure**
Iwagakure	INF	2	INF	3	INF	
Kumogakure	3	4	3	INF	2	
Kirigakure	1	1	3	2	INF	**Sunagakure**



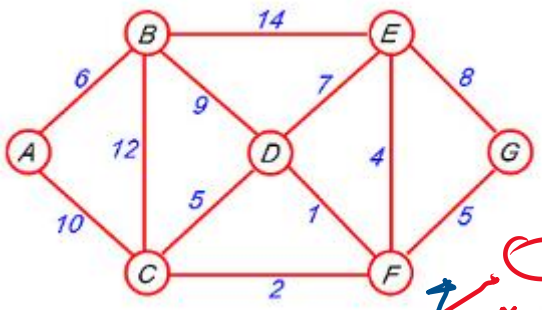
JAWABAN : 4

Source Code :

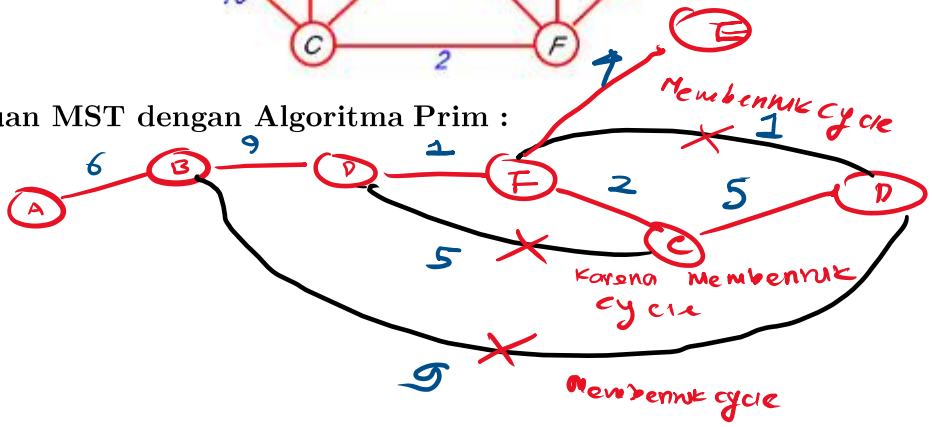
```
import heapq

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

    # Antrian prioritas untuk menyimpan node yang akan
dieksplorasi
    pq = [(0, start)]  # (distance, node)

    while pq:
        current_distance, current_node = heapq.heappop(pq)

       #JIKA NODE SAAT INI ADALAH NODE TUJUAN, SELESAIKAN
PENCARIAN
        if current_node == end:
            return current_distance

        # Visit all neighbors of the current node
        for neighbor, weight in graph[current_node]:
            distance = current_distance + weight

            #JIKA JARAK YANG BARU LEBIH PENDEK DARI JARAK
YANG SUDAH ADA
            if distance < distances[neighbor]:
                distances[neighbor] = distance

                #MASUKAN NODE TETANGGA KE ANTRIAN PRIORITAS
DENGAN JARAK YANG BARU
```

```
                heapq.heappush(pq, (distance, neighbor))

    #JIKA TIDAK ADA JALUR
    return float('inf')


graph = {
    'IWAGAKURE': [('KUMOGAKURE', 3), ('SUNAGAKURE', 2)],
    'KUMOGAKURE': [('IWAGAKURE', 3), ('SUNAGAKURE', 4),
('KONOHAGAKURE', 3), ('KIRIGAKURE', 2)],
    'SUNAGAKURE': [('IWAGAKURE', 2), ('KUMOGAKURE', 4),
('KIRIGAKURE', 1), ('KONOHAGAKURE', 3)],
    'KONOHAGAKURE': [('SUNAGAKURE', 3), ('KUMOGAKURE', 3),
('KIRIGAKURE', 1)],
    'KIRIGAKURE': [('KUMOGAKURE', 2), ('SUNAGAKURE', 1),
('KONOHAGAKURE', 1)]
}

start_node = 'KONOHAGAKURE'
end_node = 'IWAGAKURE'
shortest_distance = dijkstra(graph, start_node, end_node)
for village, dis in shortest_distances.items() :
  print('=' * 5)
  print('village:', village)
  print('distances', dis)

if shortest_distance == float('inf'):
    print("Tidak ada jalur yang tersedia dari", start_node,
"ke", end_node)
else:
    print("Jarak terpendek dari", start_node, "ke", end_node,
"adalah", shortest_distance)
```

B. Implementasi MST



Penentuan MST dengan Algoritma Prim :



MST = 6 + 9 + 1 + 2 + 5 + 4 = 27

```
import networkx as nx
import matplotlib.pyplot as plt
import heapq

graph = {
    'A': {'B': 6, 'C': 10},
    'B': {'A': 6, 'C': 12, 'D': 9, 'E': 14},
    'C': {'A': 10, 'B': 12, 'D': 5, 'F': 2},
    'D': {'B': 9, 'C': 5, 'E': 7, 'F': 1},
    'E': {'B': 14, 'D': 7, 'F': 4, 'G': 8},
    'F': {'C': 2, 'D': 1, 'E': 4, 'G': 5},
    'G': {'E': 8, 'F': 5}
}


def prim_mst(graph):
    start_node = list(graph.keys())[0]  #MENENTUKAN NODE AWAL
UNTUK MEMULAI
    visited = set([start_node]) #MENGINISIALISASI SET
    mst = [] #LIST UNTUK UNTUK MENYIMPAN MST
    total_cost = 0  #TOTAL BIAYA MST
    print(f"Graph length: {len(graph)}")
    print(f"Visited: {visited}")

    while len(visited) < len(graph):
        candidates = [] #MENYIMPAN CALON EDGES YANG AKAN
DIEKSPLORASI
        for node in visited:
            print(f"Visited: {visited}")
            print(f"Graph node items: {graph[node].items()}")
            for neighbor, cost in graph[node].items(): #LOOP NODE
TETANGGA YANG SEDANG DIPROSES SEBELUMNYA
                if neighbor not in visited:
                    heapq.heappush(candidates, (cost, node,
neighbor)) #JIKA NODE TETANGGA BELUM DIKUNJUNGI, EDGE SEKARANG KE
NODE TETANGGA AKAN MASUK KE HEAP
                    print(f"Pushed ({cost}, {node}, {neighbor})
into candidates heap")
                    print(f"Candidates: {candidates}")

        cost, from_node, to_node = heapq.heappop(candidates)
#MENGAMBIL EDGE DENGAN BIAYA TERKECIL
        mst.append((from_node, to_node, cost)) #MENAMBAHKAN EDGE
KE MST
        total_cost += cost #MENAMBAHKAN BIAYA KE TOTAL BIAYA MST
        visited.add(to_node)
        print(f"Updated Visited: {visited}")


        visualize_mst(graph, mst)
        print(f"Step: Add edge ({from_node}-{to_node}) with cost
{cost}, Total cost: {total_cost}")
        print("Current MST:", mst)
        print()
```

```
    return mst

def visualize_mst(graph, mst_edges):
    G = nx.Graph()
    G.add_weighted_edges_from(mst_edges)
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_color='skyblue',
node_size=1500, edge_color='black', linewidths=1, font_size=15,
font_weight='bold')
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels,
font_size=12)
    plt.title("Minimum Spanning Tree")
    plt.show()

mst = prim_mst(graph)
print("Minimum Spanning Tree (Prim):")
for edge in mst:
    print(f"{edge[0]} - {edge[1]} : Weight {edge[2]}")
```