Nama : Muhammad Endriansyah Rahul Sudarsono

NIM : 20220029

Laporan praktikum algoritma jaringan Latihan 1 dan Latihan 2

1. kode Python untuk menyelesaikan masalah aliran maksimum dengan algoritma Ford-Fulkerson

```
def bfs(graph, start, goal, parent):
    # implementasi algoritma Breadth-First Search
    visited = [False] * len(graph)
    queue = []
    queue.append(start)
    visited[start] = True

    while queue:
        node = queue.pop(0)
        for index, val in enumerate(graph[node]):
            if not visited[index] and val > 0:
                queue.append(index)
                visited[index] = True
                parent[index] = node
                if index == goal:
                    return True

    return False

def ford_fulkerson(graph, source, sink):
    # inisialisasi variabel
    parent = [-1] * len(graph)
    max_flow = 0

    # jalankan algoritma Ford-Fulkerson
    while bfs(graph, source, sink, parent):
        path_flow = float("inf")
        s = sink
        while s != source:
            path_flow = min(path_flow, graph[parent[s]][s])
            s = parent[s]
```

```
        max_flow += path_flow
        v = sink
        while v != source:
            u = parent[v]
            graph[u][v] -= path_flow
            graph[v][u] += path_flow
            v = parent[v]

    return max_flow

# contoh penggunaan algoritma Ford-Fulkerson
graph = [[0, 16, 13, 0, 0, 0],
         [0, 0, 10, 12, 0, 0],
         [0, 4, 0, 0, 14, 0],
         [0, 0, 9, 0, 0, 20],
         [0, 0, 0, 7, 0, 4],
         [0, 0, 0, 0, 0, 0]]
source = 0
sink = 5
max_flow = ford_fulkerson(graph, source, sink)
print("Aliran maksimum adalah: %d" % max_flow)
```

Script di atas adalah implementasi algoritma Ford-Fulkerson untuk mencari aliran maksimum dalam sebuah grafik berbobot. Berikut adalah penjelasan singkat tentang script tersebut:

Fungsi bfs(graph, start, goal, parent) adalah implementasi algoritma Breadth-First Search (BFS) untuk mencari jalur dari titik awal (start) ke titik tujuan (goal) dalam grafik. Fungsi ini mengembalikan True jika jalur ditemukan, dan False jika tidak ditemukan. Selain itu, fungsi ini juga mengubah array parent untuk merekam jalur yang telah ditemukan.

Fungsi ford_fulkerson(graph, source, sink) adalah implementasi algoritma Ford-Fulkerson menggunakan algoritma BFS yang telah dijelaskan sebelumnya. Fungsi ini menghitung aliran maksimum antara titik sumber (source) dan titik tujuan (sink) dalam grafik yang diberikan. Fungsi ini mengembalikan nilai aliran maksimum.

Pada contoh penggunaan algoritma Ford-Fulkerson, sebuah grafik yang terdiri dari matriks adjacency digunakan untuk mengilustrasikan sumber, tujuan, dan kapasitas setiap sisi antara dua titik. Kemudian, fungsi ford_fulkerson dipanggil dengan grafik tersebut, sumber dengan indeks 0, dan tujuan dengan indeks 5. Hasil aliran maksimum kemudian dicetak.

Dengan menggunakan algoritma Ford-Fulkerson, script tersebut mencari aliran maksimum dalam grafik yang diberikan dari sumber ke tujuan.

2. Contoh kode python untuk algoritma Dijkstra

```
import heapq

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

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

        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(queue, (distance, neighbor))

    return distances

# contoh penggunaan algoritma Dijkstra
graph = {
    'A': {'B': 5, 'C': 2},
    'B': {'D': 1, 'E': 6},
    'C': {'B': 1, 'D': 4},
    'D': {'E': 1},
    'E': {}
}
start_node = 'A'
distances = dijkstra(graph, start_node)

print("Jarak terpendek dari node {} ke setiap node
lainnya:".format(start_node))
for node, distance in distances.items():
    print("Node {}: {}".format(node, distance))
```

Script di atas adalah implementasi algoritma Dijkstra untuk mencari jarak terpendek dari satu node ke semua node lain dalam sebuah grafik berbobot. Berikut adalah penjelasan singkat tentang script tersebut:

Fungsi dijkstra(graph, start) adalah implementasi algoritma Dijkstra. Fungsi ini menerima grafik sebagai input dalam bentuk kamus yang mewakili adjacency list. Setiap node dalam grafik memiliki kamus yang berisi tetangga-tetangganya beserta bobot sisi yang menghubungkan mereka. Fungsi ini mengembalikan kamus yang berisi jarak terpendek dari start ke setiap node lain dalam grafik.

Pada contoh penggunaan algoritma Dijkstra, sebuah grafik direpresentasikan dalam bentuk kamus. Setiap node dalam grafik memiliki tetangga-tetangga dan bobot sisi yang menghubungkannya. Kemudian, fungsi dijkstra dipanggil dengan grafik tersebut dan node awal (start_node). Hasil jarak terpendek dari start_node ke setiap node lain kemudian dicetak.

Dengan menggunakan algoritma Dijkstra, script tersebut mencari jarak terpendek dari satu node ke semua node lain dalam grafik yang diberikan.

3. Kode python untuk membangun grafik dan mencari rute terpendek menggunakan algoritma jaringan minimum

```
import heapq

# Definisikan grafik dengan jarak antara setiap kota
graph = {
    'Jakarta': {'Bandung': 140, 'Semarang': 400, 'Surabaya': 800},
    'Bandung': {'Jakarta': 140, 'Semarang': 350, 'Surabaya': 900, 'Medan':
1250},
    'Semarang': {'Jakarta': 400, 'Bandung': 350, 'Surabaya': 650},
    'Surabaya': {'Jakarta': 800, 'Bandung': 900, 'Semarang': 650, 'Medan':
1100},
    'Medan': {'Bandung': 1250, 'Surabaya': 1100}
}

# Definisikan fungsi untuk mencari rute terpendek menggunakan algoritma
jaringan minimum
def minimum_spanning_tree(graph, start):
    visited = set([start])
    edges = [(cost, start, end) for end, cost in graph[start].items()]
    heapq.heapify(edges)
    mst_cost = 0
    mst_edges = []
```

```
    while edges:
        cost, start, end = heapq.heappop(edges)

        if end not in visited:
            visited.add(end)
            mst_cost += cost
            mst_edges.append((start, end, cost))

            for end_next, cost_next in graph[end].items():
                if end_next not in visited:
                    heapq.heappush(edges, (cost_next, end, end_next))

    return mst_edges, mst_cost

# Cetak rute terpendek dan biaya minimum menggunakan algoritma jaringan
minimum
rute, biaya = minimum_spanning_tree(graph, 'Jakarta')
print("Rute Terpendek:")
for start, end, cost in rute:
    print(f"{start} -> {end} (Biaya: {cost})")
print(f"Biaya Minimum: {biaya}")
```

Script di atas adalah implementasi algoritma jaringan minimum (minimum spanning tree) dengan menggunakan algoritma Prim. Berikut adalah penjelasan singkat tentang script tersebut:

Grafik yang merepresentasikan jarak antara setiap kota disimpan dalam variabel graph. Grafik tersebut dinyatakan dalam bentuk kamus, di mana setiap kota memiliki tetangga-tetangga lainnya beserta bobot yang menghubungkannya.

Fungsi minimum_spanning_tree(graph, start) digunakan untuk mencari rute terpendek dan biaya minimum menggunakan algoritma jaringan minimum. Fungsi ini menerima grafik dan kota awal (start) sebagai input. Fungsi ini mengembalikan kamus yang berisi rute terpendek dan biaya minimum.

Dalam fungsi minimum_spanning_tree, algoritma jaringan minimum (algoritma Prim) diimplementasikan. Variabel visited digunakan untuk melacak node yang telah dikunjungi. Variabel edges adalah sebuah heap yang berisi sisi-sisi yang mungkin ditambahkan ke jaringan minimum. Heap ini diinisialisasi dengan sisi-sisi yang terhubung langsung dengan kota awal (start). Setiap elemen dalam heap adalah tupel yang berisi bobot sisi, kota awal, dan kota tujuan.

Selama heap edges tidak kosong, sisi dengan bobot terkecil diekstraksi. Jika kota tujuan belum dikunjungi, maka sisi tersebut ditambahkan ke jaringan minimum dengan memperbarui biaya total (mst_cost) dan menambahkan rute terpendek ke mst_edges. Kemudian, tetangga-tetangga kota tujuan yang belum dikunjungi ditambahkan ke heap edges untuk dipertimbangkan selanjutnya.

Setelah selesai, rute terpendek dan biaya minimum dicetak.

Dengan menggunakan algoritma jaringan minimum (Prim), script tersebut mencari rute terpendek dan biaya minimum untuk menghubungkan kota-kota dalam grafik yang diberikan.