Nama : Febi Rhmadia putri

Nim : 20220021

Matkul : Prak.DAA

Divide and Conquer

Tugas 1.

Fungsi untuk mencari rute terpendek dari titik awal ke titik tujuan

Contoh input data menggunakan uji coba Jupyter

```
def find_shortest_path(graph, start, end):
    # Basis: jika start dan end sama, maka jaraknya 0
    if start == end:
        return 0
    # Jika tidak, cari jarak terpendek dari titik awal ke titik tujuan
    else:
        # Inisialisasi jarak terpendek ke infinity
        shortest_path = float('inf')
        # Loop melalui setiap node yang terhubung dengan start
        for neighbor, path_length in graph[start].items():
            # Jika jarak tersebut lebih pendek dari jarak terpendek yang sudah ditemukan
            if path_length < shortest_path:
                # Cari rute terpendek dari node tersebut ke titik tujuan
                recursive_path = find_shortest_path(graph, neighbor, end)
                # Jika rute tersebut ditemukan dan jarak rute tersebut lebih pendek dari jarak terpendek yang
                if recursive_path is not None and recursive_path + path_length < shortest_path:
                    # Update jarak terpendek
                    shortest_path = recursive_path + path_length
        # Jika jarak terpendek tidak berubah dari nilai awal, artinya tidak ada jalur yang ditemukan
        if shortest_path == float('inf'):
            return None
        # Kembalikan jarak terpendek yang ditemukan
        return shortest_path
```

```
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'D': 2},
    'C': {'D': 3},
    'D': {}
}
start = 'A'
end = 'D'
shortest_path = find_shortest_path(graph, start, end)
print(f"Jarak terpendek dari {start} ke {end} adalah {shortest_path}")
```

Dengan Hasil Output

```
Jarak terpendek dari A ke D adalah 3
```

Pada kode di atas, kita menggunakan pendekatan Divide and Conquer untuk mencari rute terpendek dari titik awal ke titik tujuan dalam sebuah grafik (graph). Kita memulai dari titik awal dan melakukan iterasi melalui setiap node yang terhubung dengan titik awal.

Untuk setiap node yang terhubung, kita menghitung jarak dari titik awal ke node tersebut

dan kemudian mencari rute terpendek dari node tersebut ke titik tujuan dengan menggunakan pendekatan rekursif. Jika jarak rute terpendek yang ditemukan lebih pendek dari jarak terpendek yang sudah ditemukan sebelumnya, kita mengupdate jarak terpendek tersebut.

Latihan :

contoh kode python untuk mengetahui jarak terpendek dari node A ke Node E, jika diketahui : jarak node A ke B =30 jarak node A ke C = 50 jarak node B ke C =10 jarak node C ke D = 40 jarak node dari B ke D = 10 jarak dari node C ke D = 20 jarak dari node C ke E = 30 jarak dari node D ke E = 50

Uji coba kode python ini mengunakan Jupyter dengan input nilai yang ada di atas.

```python
import heapq

def dijkstra(graph, start, end):
    # Inisialisasi jarak ke setiap node dengan nilai infinity
    distances = {node: float('inf') for node in graph}
    # Jarak ke node awal diatur ke 0
    distances[start] = 0
    # Setiap node yang sudah dievaluasi disimpan di set visited
    visited = set()
    # Queue untuk menyimpan node yang akan dievaluasi berikutnya
    heap = [(0, start)]

    while heap:
        # Ambil node dengan jarak terdekat dari heap
        (distance, current) = heapq.heappop(heap)
        # Jika node tersebut belum dievaluasi sebelumnya
        if current not in visited:
            # Tandai node tersebut sebagai dievaluasi
            visited.add(current)
            # Loop melalui semua node yang terhubung dengan node tersebut
            for neighbor, weight in graph[current].items():
                # Hitung jarak baru ke node tersebut
                new_distance = distance + weight
                # Jika jarak baru lebih pendek dari jarak sebelumnya ke node tersebut
                if new_distance < distances[neighbor]:
                    # Update jarak ke node tersebut
                    distances[neighbor] = new_distance
                    # Tambahkan node tersebut ke heap
                    heapq.heappush(heap, (new_distance, neighbor))
```

```python
    # Kembalikan jarak terpendek ke node tujuan
    return distances[end]

# Definisikan grafik
graph = {
    'A': {'B': 30, 'C': 50},
    'B': {'C': 10, 'D': 10},
    'C': {'D': 20, 'E': 30},
    'D': {'E': 50},
    'E': {}
}

# Hitung jarak terpendek dari A ke E
shortest_distance = dijkstra(graph, 'A', 'E')

# Cetak jarak terpendek
print(f"Jarak terpendek dari A ke E: {shortest_distance}")
```

Dengan Hasil Output Nilai.

Dengan Jarak Terpendek dari A ke E adalah 70

```
Jarak terpendek dari A ke E: 70
```

Ini termasuk kode Python untuk menjalankan algoritma Dijkstra, sebuah algoritma untuk mencari jarak terpendek pada grafik berbobot positif tunggal. Namun, baris pertama pada kode di atas yaitu import heapq adalah kode untuk mengimpor modul heapq, yang digunakan di dalam fungsi Dijkstra untuk melakukan operasi heap pada data. Heapq adalah modul dalam Python yang menyediakan implementasi struktur data heap untuk data yang dapat diubah secara dinamis. Dalam konteks Dijkstra, struktur heap digunakan untuk memprioritaskan simpul-simpul yang belum dikunjungi yang memiliki jarak terpendek ke simpul awal (start node) saat ini.

a. Pertama-tama, kita melakukan import library heapq, yang digunakan untuk membuat priority queue.
b. Kemudian, kita mendefinisikan fungsi dijkstra yang akan mengambil tiga parameter: graph (graf yang akan dicari jarak terpendeknya), start (node awal), dan end (node tujuan).
c. Pertama-tama, kita inisialisasi jarak ke setiap node dengan nilai infinity, kecuali untuk node awal yang jaraknya adalah 0.
d. Kemudian, kita menyimpan node-node yang belum dikunjungi dalam priority queue yang diurutkan berdasarkan jaraknya dari node awal.
e. Selama ada node yang belum dikunjungi, kita ambil node yang jaraknya terpendek dan loop melalui setiap node yang terhubung dengannya.
f. Untuk setiap tetangga yang belum dikunjungi, kita hitung jarak baru dari node awal melalui node saat ini ke tetangganya. Jika jarak tersebut lebih pendek dari jarak yang
g. sudah ada ke tetangganya, maka kita update jarak terpendek ke tetangganya dan tambahkan tetangga ke daftar node yang belum dikunjungi.
h. Setelah semua node dikunjungi, kita kembalikan j