LAPORAN TUGAS STRUKTUR DATA
GRAPH DIJKSTRA ALGORITMA - GOOGLE MAPS
KELOMPOK 10

Disusun oleh:

1. Gerry Moeis M.D.P (23091397164)
2. Ahmad Aryobimo (23091397151)
3. Dea Ayu Novita Putri (23091397173)

Progam Studi D4 Manajemen Informatika
Fakultas Vokasi
Universitas Negeri Surabaya
2024

Link GitHub :

https://github.com/gerrymoeis/tugas-struktur-data/tree/tugas-3

1. kelompok_10_katakan_dora.py

Berisikan class **KatakanPeta** sebagai blueprint untuk studi kasus Google Maps.

Juga penerapan konsep algoritma Dijkstra untuk menentukan rute tercepat yang bisa ditempuh dari suatu kota.

```
class KatakanPeta():
    def __init__(self):
        self.daftarKota = {}
        self.jumlahKota = 0
        self.peta = {}

    def tampilkanKota(self):
        for kota in self.daftarKota:
            print(kota)

        print(f"Jumlah Kota: {self.jumlahKota}")

    def tampilkanPeta(self):
        for i, kota in enumerate(self.peta):
            print(f"{i+1}. {kota}")
            for kotaTetangga, rute in self.peta[kota].items():
                print(f"\t--> {kotaTetangga}:")
                for jalan in rute.values():
                    print(f"\t\t--> {jalan}")

    def tambahkanKota(self, kota):
        if kota not in self.daftarKota:
            self.daftarKota[kota] = {}
            self.jumlahKota += 1

    def tambahkanJalan(self, kota1, cities):
        for kota in cities:
            if kota1 and kota in self.daftarKota:
                self.daftarKota[kota1][kota] = cities[kota]
                self.daftarKota[kota][kota1] = cities[kota]
```

```
    def hapusKota(self, kotaDihapus):
        if kotaDihapus in self.daftarKota:
            for kota in self.daftarKota:
                if kotaDihapus in self.daftarKota[kota]:
                    del self.daftarKota[kota][kotaDihapus]
            del self.daftarKota[kotaDihapus]
            self.jumlahKota -= 1

    def hapusJalan(self, kota1, kota2):
        if kota1 and kota2 in self.daftarKota:
            del self.daftarKota[kota1][kota2]
            del self.daftarKota[kota2][kota1]

    def buatkanRute(self, start):
        peta = {}

        [distances, routes] = self.dijkstra(start)
        for kota, jarak in distances.items():
            peta[kota] = {}
            peta[kota]["rute"] = self.route(distances, routes, start, kota)
            peta[kota]["jarak"] = jarak

        self.peta[start] = peta

    def cariRuteTercepat(self, dari, ke):
        dari = dari.title()
        ke = ke.title()
        if dari in self.daftarKota and ke in self.daftarKota:
            print(f"\tDari {dari} ke {ke}, berjarak
{self.peta[dari][ke]["jarak"]} km")
            print(f"\tDengan rute: ")
            for jalan, jarak in reversed(self.peta[dari][ke]["rute"].items()):
                print("\t\t --->", jalan, jarak, "km")
        else:
            print(f"{dari} atau {ke} tidak berada di daftar kota")

    def route(self, distances, routes, start, target):
        path = {}
        city = target

        while city != start:
            if routes[city] == start:
                path[city] = distances[city]
            else:
                path[city] = round(distances[city] - distances[routes[city]],
1)
            city = routes[city]
```

```
        return path

    def dijkstra(self, start):
        unvisited_cities = [*self.daftarKota.keys()]
        distances = {}
        routes = {}

        for city in unvisited_cities:
            distances[city] = float("inf")
        distances[start] = 0

        while unvisited_cities:
            closest_city = None
            for city in unvisited_cities:
                if closest_city == None:
                    closest_city = city
                elif distances[city] < distances[closest_city]:
                    closest_city = city

            for neighbour, distance in self.daftarKota[closest_city].items():
                total_distance = round(distances[closest_city] + distance, 1)
                if total_distance < distances[neighbour]:
                    distances[neighbour] = total_distance
                    routes[neighbour] = closest_city

            unvisited_cities.remove(closest_city)

        del distances[start]
        return distances, routes

cities = ["Amsterdam", "Almere", "Amersfoort", "Utrecht", "Vianen", "Gouda",
"Rotterdam", "Delft", "Den Haag", "Leiden", "Haarlem"]

petaBelanda = KatakanPeta()
for city in cities:
    petaBelanda.tambahkanKota(city)

petaBelanda.tambahkanJalan("Amsterdam", {"Haarlem": 31.7, "Leiden": 49.6,
"Almere": 32.3})
petaBelanda.tambahkanJalan("Den Haag", {"Leiden": 33, "Delft": 13})
petaBelanda.tambahkanJalan("Gouda", {"Delft": 35.8, "Rotterdam": 23.7,
"Utrecht": 40.9})
petaBelanda.tambahkanJalan("Utrecht", {"Gouda": 40.9, "Vianen": 18.2,
"Amersfoort": 24.4})
petaBelanda.tambahkanJalan("Almere", {"Amsterdam": 32.3, "Amersfoort": 42})

print("=== KOTA BELANDA ===")
petaBelanda.tampilkanKota()
```

```
for city in cities:
    petaBelanda.buatkanRute(city)

print("=== PETA BELANDA ===")
petaBelanda.tampilkanPeta()

print("=== CARI RUTE TERCEPAT ===")
while True:
    dari = input("Cari Rute Tercepat dari (Keluar ketik 'exit'): ")
    if dari == "exit": break
    ke = input(f"Dari {dari.title()} hendak ke (Keluar ketik 'exit'): ")
    if ke == "exit": break

    petaBelanda.cariRuteTercepat(dari, ke)
    print()
```

- ## Penjelasan Tiap Command

1. kelompok_10_katakan_dora.py

```
class KatakanPeta():
    def __init__(self):
        self.daftarKota = {}
        self.jumlahKota = 0
        self.peta = {}
```

Kode diatas bertujuan untuk membuat blueprint sebuah objek berupa peta dengan menerapkan konsep struktur data graph.

Di dalam class **KatakanPeta** kami menginisiasi atribut daftar kota yang berupa dictionary, jumlah kota yang berupa integer, dan atribut peta yang berupa dictionary untuk menampung seluruh rute dalam peta.

```
    def tampilkanKota(self):
        for kota in self.daftarKota:
            print(kota)

        print(f"Jumlah Kota: {self.jumlahKota}")
```

Setelah menginisiasi atribut, kemudian kami membuat metode **tampilkanKota** yang melakukan looping untuk menampilkan setiap kota di dalam daftar kota dan jumlah kotanya.

```
    def tampilkanPeta(self):
        for i, kota in enumerate(self.peta):
            print(f"{i+1}. {kota}")
            for kotaTetangga, rute in self.peta[kota].items():
                print(f"\t--> {kotaTetangga}:")
                for jalan in rute.values():
                    print(f"\t\t--> {jalan}")
```

Untuk menampilkan keseluruhan rute serta jarak antar kota pada peta, kami membuat metode tersendiri yaitu **tampilkanPeta** dengan konsep yang sama yaitu melakukan looping dari atribut peta.

```
    def tambahkanKota(self, kota):
        if kota not in self.daftarKota:
            self.daftarKota[kota] = {}
            self.jumlahKota += 1
```

Metode diatas bertujuan untuk menambahkan kota bilamana kota yang di-input belum ada di dalam daftar kota.

```
    def tambahkanJalan(self, kota1, cities):
        for kota in cities:
            if kota1 and kota in self.daftarKota:
                self.daftarKota[kota1][kota] = cities[kota]
                self.daftarKota[kota][kota1] = cities[kota]
```

Untuk menambahkan jalan, metode diatas mengambil input berupa kota asal dan kota-kota yang ingin dihubungkan. Kami menggunakan konsep dictionary untuk melakukan input jarak bagi masing-masing jalan antar kota.

```
    def hapusKota(self, kotaDihapus):
        if kotaDihapus in self.daftarKota:
            for kota in self.daftarKota:
                if kotaDihapus in self.daftarKota[kota]:
                    del self.daftarKota[kota][kotaDihapus]
            del self.daftarKota[kotaDihapus]
            self.jumlahKota -= 1
```

Metode **hapusKota** menerima input kota yang ingin dihapus lalu mengecek bila kota yang ingin dihapus memang berada di dalam daftar kota. Lakukan penghapusan kota tersebut pada hubungannya di kota-kota yang lain.

```
    def hapusJalan(self, kota1, kota2):
        if kota1 and kota2 in self.daftarKota:
            del self.daftarKota[kota1][kota2]
            del self.daftarKota[kota2][kota1]
```

Metode ini mengecek apabila kedua kota yang di-input berada di dalam daftar kota, bila memang ada maka jalan atau hubungan antar kota tersebut dihapus.

```
    def buatkanRute(self, start):
        peta = {}

        [distances, routes] = self.dijkstra(start)
        for kota, jarak in distances.items():
            peta[kota] = {}
            peta[kota]["rute"] = self.route(distances, routes, start, kota)
            peta[kota]["jarak"] = jarak

        self.peta[start] = peta
```

Metode diatas bertujuan untuk membuat rute jalan untuk seluruh kota dalam daftar kota dan memasukkannya ke dalam atribut peta.

Untuk mencari rute dan jalan, kami menggunakan algoritma Dijkstra.

```
    def cariRuteTercepat(self, dari, ke):
        dari = dari.title()
        ke = ke.title()
        if dari in self.daftarKota and ke in self.daftarKota:
            print(f"\tDari {dari} ke {ke}, berjarak
{self.peta[dari][ke]["jarak"]} km")
            print(f"\tDengan rute: ")
            for jalan, jarak in reversed(self.peta[dari][ke]["rute"].items()):
                print("\t\t --->", jalan, jarak, "km")
        else:
            print(f"{dari} atau {ke} tidak berada di daftar kota")
```

Metode ini mengambil input kota asal dan kota tujuan lalu mengeceknya terlebih dahulu, jika kota tersebut tidak berada di dalam daftar kota, maka tampilkan pesan bahwa kota tersebut tidak berada di daftar kota.

Jika kota tersebut ada, maka kita cari jarak terdekat dari kota asal tersebut. Lalu tampilkan keseluruhan rute mulai dari kota awal hingga kota tujuan.

```
    def route(self, distances, routes, start, target):
        path = {}
        city = target

        while city != start:
            if routes[city] == start:
                path[city] = distances[city]
            else:
                path[city] = round(distances[city] - distances[routes[city]],
1)
            city = routes[city]

        return path
```

Metode **route** dibuat untuk menghasilkan keseluruhan rute yang ditempuh mulai dari kota awal hingga kota tujuan.

Metode ini membutuhkan parameter berupa distances dan routes yang diambil dari algoritma Dijkstra. Juga parameter kota awal dan kota tujuan.

Dalam menghitung jarak rute, looping yang dilakukan bersifat reverse.

```
    def dijkstra(self, start):
        unvisited_cities = [*self.daftarKota.keys()]
        distances = {}
        routes = {}

        for city in unvisited_cities:
            distances[city] = float("inf")
        distances[start] = 0

        while unvisited_cities:
            closest_city = None
            for city in unvisited_cities:
                if closest_city == None:
                    closest_city = city
                elif distances[city] < distances[closest_city]:
                    closest_city = city

            for neighbour, distance in self.daftarKota[closest_city].items():
                total_distance = round(distances[closest_city] + distance, 1)
                if total_distance < distances[neighbour]:
                    distances[neighbour] = total_distance
                    routes[neighbour] = closest_city

            unvisited_cities.remove(closest_city)

        del distances[start]
        return distances, routes
```

Metode diatas merupakan penerapan konsep algoritma Dijkstra. Dimulai dengan membuat variable untuk memproses penghitungan jarak antar rutenya.

Untuk pemrosesan utamanya, kita melakukan while loop pada unvisited cities, lalu menentukan kota terdekat dari kota awal. Kemudian menghitung total jarak dari kota terdekat tersebut ke kota tetangga yang terhubung dengan kota terdekat.

Setelah pemrosesan selesai dilakukan, hapus kota terdekat dari unvisited cities untuk setiap kali looping.

Terakhir, kembalikan data berupa keseluruhan jarak dan rute dari kota awal yang di-input.

```
cities = ["Amsterdam", "Almere", "Amersfoort", "Utrecht", "Vianen", "Gouda",
"Rotterdam", "Delft", "Den Haag", "Leiden", "Haarlem"]

petaBelanda = KatakanPeta()
for city in cities:
    petaBelanda.tambahkanKota(city)
```

Untuk penerapannya, kami membuat list kota yang akan di-input ke dalam object class nya.

Lalu kita buat variable **petaBelanda** sebagai objek dari class **KatakanPeta**, kemudian dalam menambahkan kota ke objek **petaBelanda**, kami menggunakan looping.

```
petaBelanda.tambahkanJalan("Amsterdam", {"Haarlem": 31.7, "Leiden": 49.6,
"Almere": 32.3})
petaBelanda.tambahkanJalan("Den Haag", {"Leiden": 33, "Delft": 13})
petaBelanda.tambahkanJalan("Gouda", {"Delft": 35.8, "Rotterdam": 23.7,
"Utrecht": 40.9})
petaBelanda.tambahkanJalan("Utrecht", {"Gouda": 40.9, "Vianen": 18.2,
"Amersfoort": 24.4})
petaBelanda.tambahkanJalan("Almere", {"Amsterdam": 32.3, "Amersfoort": 42})
```

Sekarang, setelah semua kota telah di-input, kita bisa menambahkan jalan antar kota beserta jaraknya dengan menggunakan metode **tambahkanJalan**.

```
print("=== KOTA BELANDA ===")
petaBelanda.tampilkanKota()

for city in cities:
    petaBelanda.buatkanRute(city)

print("=== PETA BELANDA ===")
petaBelanda.tampilkanPeta()
```

Kode diatas bertujuan untuk menampilkan daftar kota di Belanda, setelah itu membuat rute untuk masing-masing kota di dalam **petaBelanda**.

Kemudian ditampilkan sesuai format print pada metode **tampilkanPeta**.

Dalam mengambil input user, kami menggunakan while loop yang mana akan berhenti jika user meng-input exit.

```
print("=== CARI RUTE TERCEPAT ===")
while True:
    dari = input("Cari Rute Tercepat dari (Keluar ketik 'exit'): ")
    if dari == "exit": break
    ke = input(f"Dari {dari.title()} hendak ke (Keluar ketik 'exit'): ")
    if ke == "exit": break

    petaBelanda.cariRuteTercepat(dari, ke)
    print()
```

Kode diatas memberikan kesempatan kepada user bilamana ingin mencari rute tercepat dari suatu kota ke kota tujuan.

Program akan terus meminta input dari user selama user tidak mengetikkan kata **exit**.

Program akan menampilkan rute tercepat yang diproses melalui metode **cariRuteTercepat**.

Hasil Output :

```
PS C:\Users\X441U> & C:/Users/X441U/AppData/Local/Programs/Python/Python312/python.exe "d:/Struktur Data/kelompok_10_katakan_dora.py"
=== KOTA BELANDA ===
Amsterdam
Almere
Amersfoort
Utrecht
Vianen
Gouda
Rotterdam
Delft
Den Haag
Leiden
Haarlem
Jumlah Kota: 11

=== PETA BELANDA ===
1. Amsterdam
        --> Almere:
                --> {'Almere': 32.3}
                --> 32.3
        --> Amersfoort:
                --> {'Amersfoort': 42.0, 'Almere': 32.3}
                --> 74.3
        --> Utrecht:
                --> {'Utrecht': 24.4, 'Amersfoort': 42.0, 'Almere': 32.3}
                --> 98.7
        --> Vianen:
                --> {'Vianen': 18.2, 'Utrecht': 24.4, 'Amersfoort': 42.0, 'Almere': 32.3}
                --> 116.9
        --> Gouda:
                --> {'Gouda': 35.8, 'Delft': 13.0, 'Den Haag': 33.0, 'Leiden': 49.6}
                --> 131.4
        --> Rotterdam:
```

```
=== CARI RUTE TERCEPAT ===
Cari Rute Tercepat dari (Keluar ketik 'exit'): Amsterdam
Dari Amsterdam hendak ke (Keluar ketik 'exit'): rotterdam
        Dari Amsterdam ke Rotterdam, berjarak 155.1 km
        Dengan rute:
                ---> Leiden 49.6 km
                ---> Den Haag 33.0 km
                ---> Delft 13.0 km
                ---> Gouda 35.8 km
                ---> Rotterdam 23.7 km

Cari Rute Tercepat dari (Keluar ketik 'exit'): Haarlem
Dari Haarlem hendak ke (Keluar ketik 'exit'): Vianen
        Dari Haarlem ke Vianen, berjarak 148.6 km
        Dengan rute:
                ---> Amsterdam 31.7 km
                ---> Almere 32.3 km
                ---> Amersfoort 42.0 km
                ---> Utrecht 24.4 km
                ---> Vianen 18.2 km

Cari Rute Tercepat dari (Keluar ketik 'exit'): Sidoarjo
Dari Sidoarjo hendak ke (Keluar ketik 'exit'): Surabaya
Sidoarjo atau Surabaya tidak berada di daftar kota

Cari Rute Tercepat dari (Keluar ketik 'exit'): exit
```

Model Struktur Data Graph Peta Belanda di Google Maps :