LAPORAN TUGAS STRUKTUR DATA
GRAPH – 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-2

1. kelompok_10_katakan_peta.py

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

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

    def tampilkanPeta(self):
        for kota in self.daftarKota:
            print(f"{kota} -> {self.daftarKota[kota]}")
        print(f"Jumlah Kota: {self.jumlahKota}")

    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 cariRuteTercepat(self, kota):
        kota = kota.title()
        if kota in self.daftarKota:
            terdekat = min(self.daftarKota[kota].values())
            rute = [jalan for jalan in self.daftarKota[kota] if
self.daftarKota[kota][jalan] == terdekat]
            print(", ".join([f"{jalan} {terdekat}km" for jalan in rute]))
        else:
            print(f"{kota} tidak berada di daftar kota")

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

rute_tercepat = input("Cari Rute Tercepat dari (keluar ketik 'exit'): ")
while rute_tercepat != "exit":
    petaBelanda.cariRuteTercepat(rute_tercepat)
    rute_tercepat = input("Cari Rute Tercepat dari (keluar ketik 'exit'): ")
```

- **Penjelasan Tiap Command**
  1. **kelompok_10_katakan_peta.py**

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

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 dan jumlah kota yang berupa integer.

```
    def tampilkanPeta(self):
        for kota in self.daftarKota:
            print(f"{kota} -> {self.daftarKota[kota]}")
        print(f"Jumlah Kota: {self.jumlahKota}")
```

Setelah menginisiasi atribut, kemudian kami membuat metode **tampilkanPeta** yang melakukan looping untuk setiap kota di dalam daftar kota dan ditampilkan jalan antar kota serta jumlah kota.

```
    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 cariRuteTercepat(self, kota):
        kota = kota.title()
        if kota in self.daftarKota:
            terdekat = min(self.daftarKota[kota].values())
            rute = [jalan for jalan in self.daftarKota[kota] if
self.daftarKota[kota][jalan] == terdekat]
            print(", ".join([f"{jalan} {terdekat}km" for jalan in rute]))
        else:
            print(f"{kota} tidak berada di daftar kota")
```

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

Jika kota tersebut ada, maka kita cari jarak terdekat dari kota asal tersebut. Lalu tampilkan kota terdekat beserta jaraknya.

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

rute_tercepat = input("Cari Rute Tercepat dari (keluar ketik 'exit'): ")
while rute_tercepat != "exit":
    petaBelanda.cariRuteTercepat(rute_tercepat)
    rute_tercepat = input("Cari Rute Tercepat dari (keluar ketik 'exit'): ")
```

Kode diatas bertujuan untuk menampilkan **petaBelanda** dan mengambil input rute tercepat yang ingin diketahui oleh user.

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

Hasil Output :

```
=== PETA BELANDA ===
Amsterdam -> {'Haarlem': 31.7, 'Leiden': 49.6, 'Almere': 32.3}
Almere -> {'Amsterdam': 32.3, 'Amersfoort': 42}
Amersfoort -> {'Utrecht': 24.4, 'Almere': 42}
Utrecht -> {'Gouda': 40.9, 'Vianen': 18.2, 'Amersfoort': 24.4}
Vianen -> {'Utrecht': 18.2}
Gouda -> {'Delft': 35.8, 'Rotterdam': 23.7, 'Utrecht': 40.9}
Rotterdam -> {'Gouda': 23.7}
Delft -> {'Den Haag': 13, 'Gouda': 35.8}
Den Haag -> {'Leiden': 33, 'Delft': 13}
Leiden -> {'Amsterdam': 49.6, 'Den Haag': 33}
Haarlem -> {'Amsterdam': 31.7}
Jumlah Kota: 11
Cari Rute Tercepat dari (keluar ketik 'exit'): Amsterdam
Haarlem 31.7km
Cari Rute Tercepat dari (keluar ketik 'exit'): utrecht
Vianen 18.2km
Cari Rute Tercepat dari (keluar ketik 'exit'): leiden
Den Haag 33km
Cari Rute Tercepat dari (keluar ketik 'exit'): Surabaya
Surabaya tidak berada di daftar kota
Cari Rute Tercepat dari (keluar ketik 'exit'): Sidoarjo
Sidoarjo tidak berada di daftar kota
Cari Rute Tercepat dari (keluar ketik 'exit'): Harlem
Harlem tidak berada di daftar kota
Cari Rute Tercepat dari (keluar ketik 'exit'): haarlem
Amsterdam 31.7km
Cari Rute Tercepat dari (keluar ketik 'exit'): exit
```

Model Struktur Data Graph Peta Belanda di Google Maps :