Penerapan Algoritma Brute Force dalam Pencarian Rute Optimal untuk Survei Kos di Bandung dengan Google Maps API

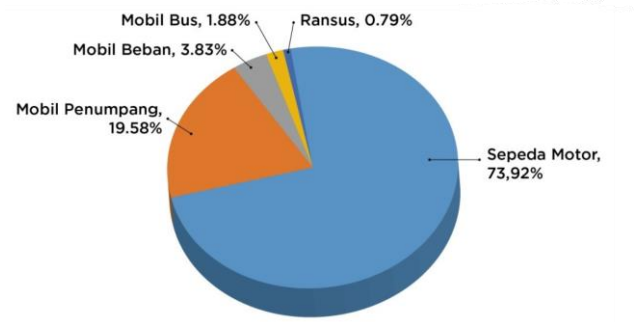
Gerald Abraham Sianturi - 13520138
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13520138@std.stei.itb.ac.id

***Abstrak*—Kunjungan ke beberapa tempat tujuan secara berkelanjutan adalah suatu aktivitas yang umum terjadi dan salah satu motif yang mungkin adalah mencari kos yang tepat, khususnya untuk mahasiswa baru. Google melalui Google APIs & Services memberikan kesempatan untuk pengembang mengakses layanan yang ada pada program atau aplikasinya. Makalah ini akan membahas bagaimana penggunaan algoritma *brute force* dalam mencari rute optimal pencarian kos berdasarkan waktu dan durasi optimal dengan bantuan Google APIs *distance matrix*.**

Kata kunci— brute force, Google-API, kos, kunjungan, survei

I. PENDAHULUAN

Aktivitas perjalanan dari satu tempat ke tempat lainnya dengan perbedaan titik merupakan salah satu aktivitas umum yang dilakukan manusia, baik perjalanan ke hanya satu tempat, maupun perjalanan ke beberapa tempat sekaligus pada rentang waktu yang pendek pada kunjungan tiap tempatnya. Perjalanan dapat dilakukan dengan menggunakan kendaraan, seperti mobil, motor, atau juga berjalan kaki yang dari segi kepemilikan juga dibagi lagi menjadi kendaraan pribadi atau kendaraan umum. Motif perjalanan juga beragam, di antaranya untuk liburan, berkunjung ke suatu tempat untuk interaksi sosial dengan teman, bekerja.



Gambar 1. Grafik komposisi jumlah kendaraan bermotor di Jakarta pada tahun 2018

(Sumber: http://jurusanku.com/utstraffic/ yang bersumber dari Badan Pusat Statistik (BPS) Indonesia)

Perjalanan juga turut memakan *cost*, seperti uang yang dibutuhkan untuk perjalanan yang terdiri dari biaya bahan bakar kendaraan yang dipakai, untuk menyewa kendaraan atau biaya yang dibutuhkan untuk memenuhi kebutuhan seperti makanan selama perjalanan, waktu yang dibutuhkan untuk melakukan perjalanan menggunakan moda transportasi tertentu, serta energi yang dibutuhkan untuk perjalanan tersebut.

Salah satu motif perjalanan yang dapat dilakukan adalah melakukan survei untuk membandingkan kelebihan dan kekurangan dari suatu tempat. Misalnya, survei yang dilakukan oleh seksi acara suatu kepanitiaan dalam memilih atau menentukan tempat yang paling efisien berdasarkan faktor harga, atau dapat juga survei yang dilakukan untuk memilih rumah atau tempat tinggal yang paling tepat.

Apapun itu motifnya, setiap orang tentu menginginkan perjalanan yang optimal, baik dari segi durasi yang diharapkan minimum (tidak memakan waktu banyak di jalan), dari segi biaya yang sebisa mungkin minimum, maupun dari segi jarak yang sebisa mungkin memiliki perpindahan yang sedekat mungkin guna menghemat penggunaan bahan bakar minyak misalnya.

Makalah ini ditujukan untuk menyediakan analisis dan implementasi pemilihan rute perjalanan yangoptimal (permasalahan *Travelling Salesman Problem* atau TSP) untuk melakukan survei ke beberapa tempat kos dari titik awal tertentu agar perjalanan bisa memiliki waktu yang terendah serta perpindahan yang sependek mungkin menggunakan *real world data* yang diperoleh menggunakan Google Maps API. Dari berbagai opsi algoritma untuk menentukan rute optimal, pada makalah ini digunakan algoritma *brute force*. Secara khusus, pada makalah ini akan diimplementasikan penyelesaian kasus pada survei kos di Bandung dari data kos-kosan yang diperoleh pada aplikasi Mamikos. Walaupun demikian, implementasi yang dibuat sebenarnya dapat diaplikasikan untuk kasus lain yang memiliki titik awal dan satu atau lebih titik destinasi.

II. LANDASAN TEORI

A. *Google Maps*

Google Maps adalah layanan berbasis *web* dan aplikasi yang disediakan oleh Google dengan memberikan informasi terkait wilayah-wilayah geografis di berbagai belahan dunia kepada penggunanya. Google Maps menyediakan beberapa

fitur teknis (terdapat perbedaan fitur yang bergantung *platform* atau versi yang digunakan, *platform* yang dimaksud dapat berupa *mobile device* atau berbasis *web*), di antaranya, mengukur jarak antara dua titik lokasi sesuai dengan moda transportasi dan jarak garis lurus antara dua titik tersebut (dapat juga mengukur akumulasi lebih dari dua lokasi secara kontinu untuk beberapa tempat sekaligus), melihat tampilan tiga dimensi *street view* beberapa titik lokasi tertentu, menghitung area dari *frame* yang dibentuk oleh pengguna pada GUI yang disediakan, serta *sharing* lokasi *real-time* dari pengguna tertentu.

Gambar 2. Fitur pada Google Maps pada sistem operasi Android untuk menentukan informasi perjalanan yang lebih dari dua titik

Google Maps API adalah serangkaian *tools* atau metode yang disediakan oleh Google untuk pihak tertentu agar mereka dapat berkomunikasi dengan layanan yang Google miliki. Secara umum, antarmuka aplikasi pemrograman ini dipakai oleh pengembang untuk membangun aplikasi, baik aplikasi berbasis web ataupun aplikasi yang berjalan pada sistem operasi tertentu.

B. Travelling Salesman Problem

Travelling Salesman Problem (TSP) adalah permasalahan klasik pada bidang matematika dan matematika terapan terkait kasus optimasi. Sejauh ini, catatan pertama yang mencatat formulasi permasalahan ini adalah "*Das botenproblem"* pada *Ergebnisse eines Mathematischen Kolloquiums* yang dicatat oleh Karl Menger dan dipublikasikan pada tahun 1932. Pada catatan tersebut, istilah yang dipakai adalah "Messenger Problem". Formulasi permasalahannya kurang lebih sebagai berikut:

- Seorang penjual keliling yang ingin mengunjungi $n \mid n \in N$ kota, dan tiap kota hanya dikunjungi tepat satu kali.
- Setiap perpindahan satu kota ke kota lainnya, terdapat *cost* c_{ij} yang merupakan *cost* yang dibutuhkan untuk perjalanan dari kota i ke kota j.
- Orang tersebut kembali ke kota awal.
- Permasalahan: Bagaimana orang tersebut dapat melakukan perjalanan ke tiap kota dan kembali ke kota awal dengan jarak seminimum mungkin?

Permasalahan TSP sendiri dibagi lagi menjadi dua, yakni:

1. *Travelling Salesman Optimization Problem*

   Permasalahan ini terkait dengan menentukan urutan tur agar menghasilkan bobot atau *cost* yang minimum.

2. *Travelling Salesman Decision Problem*

   Fokus kasus permasalahan ini adalah ada atau tidaknya tur dengan *cost* atau bobot tertentu dengan rentang nilai tertentu.

Aplikasi TSP dapat diimplementasikan pada beberapa kasus nyata, misalnya, sebuah perusahaan yang memiliki kurir pengantaran dan ingin setiap kurir menempuh jarak atau waktu seminimum mungkin untuk mengurangi biaya operasional, ataupun sebuah bus sekolah yang melakukan penjemputan terhadap beberapa anak yang tersebar di beberapa lokasi.

Dalam penyelesaian permasalahan ini, terdapat beberapa algoritma yang dapat digunakan, di antaranya:

1. Algoritma *brute force*

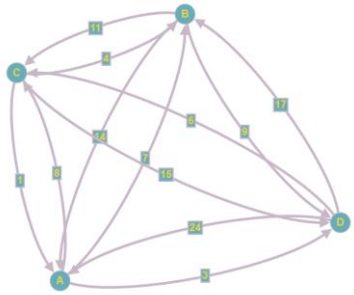
   Penyelesaian TSP dengan *brute force* dilakukan dengan mengenumerasi hasil permutasi semua kemungkinan pilihan jalur yang mungkin. Tiap opsi urutan jalur yang mungkin kemudian dihitung. Setelah tiap opsi urutan jalur dihitung besar bobotnya, kemudian opsi dengan urutan yang paling optimal—paling kecil—dipilih menjadi solusi.

2. Algoritma *branch and bound*

   Penyelesaian dengan algoritma ini didasarkan pada *space state tree* dan perhitungan *cost*, yang pertama, yakni *cost* yang dibutuhkan dari titik awal ke titik akhir sementara yang memiliki akumulasi penjumlahan bobot minimum sementara (pada implementasi graf adalah jumlah bobot-bobot sisi dari *root* ke simpul tertentu. Lalu, yang kedua, yakni *cost* dari titik akhir sementara. *Cost* dari setiap simpul dapat dihitung salah satunya menggunakan metode matriks ongkos tereduksi. Langkah-langkahnya adalah sebagai berikut:

   i) Lakukan pemetaan bobot tiap sisi dari satu titik atau kota ke kota lain dalam bentuk *adjacency matrix* M.

	A	B	C	D
A	∞	7	8	3
B	14	∞	11	9
C	1	4	∞	15
D	24	17	6	∞



Gambar 3. Ilustrasi kota (simpul) dan jalur (sisi berbobot berarah)

ii) Translasikan *adjacency matrix M* tersebut menjadi *reduced cost matrix* dengan mengurangi tiap baris dan kolom masing-masing dengan nilai baris dan kolom terkecil sehingga pada tiap kolom dan baris setidaknya memiliki minimal satu nilai 0

Hasil proses mereduksi baris:

	A	***B***	***C***	***D***	
A	∞	4	5	0	$[-3]$
B	5	∞	2	0	$[-9]$
C	0	3	∞	14	$[-1]$
D	18	11	0	∞	$[-6]$

Hasil proses mereduksi kolom:

	A	***B***	***C***	***D***
A	∞	1	5	0
B	5	∞	2	0
C	0	0	∞	14
D	18	8	0	∞
	$[0]$	$[-3]$	$[0]$	$[0]$

iii) Kalkulasikan jumlah semua pengurangan pada proses reduksi baris dan kolom, yakni:

$$(3 + 9 + 1 + 6) + (0 + 3 + 0 + 0) = 22$$

iv) Mulai bangun *space state tree* dengan *cost* akar adalah hasil kalkulasi pada poin (iii)

v) Turunkan anak (*child*) dari simpul dengan *cost* terkecil sementara, dan untuk masing-masing anak, hitung juga *cost*-nya, yakni nilai *cost parent* ditambah nilai *cost* dari parent ke anak tersebut ditambah nilai akumulasi dari semua pengurangan pada proses reduksi matriks S. Matriks S adalah matriks hasil reduksi pada level sebelumnya yang dimodifikasi sebagai berikut:

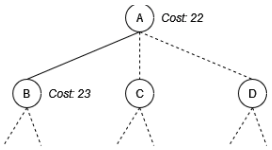
a. Tiap baris simpul orang tua, dan tiap kolom simpul anak diubah menjadi ∞.

b. Sel dengan baris simpul anak ke orang tua juga diubah menjadi ∞.

Misal, pada kasus *generate child* akar (A), pada saat *generate* simpul anak B, matriks S adalah:

	A	***B***	***C***	***D***
A	∞	∞	∞	∞
B	5∞	∞	2	0
C	0	∞	∞	14
D	18	∞	0	∞
	$[0]$	$[0]$	$[0]$	$[0]$

Cost dari level 1 *node B* adalah $22 + 1 + 0 = 23$



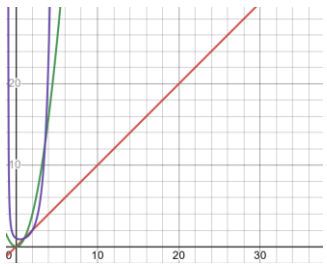
Gambar 4. *Space state tree* sementara

vi) Lakukan pengulangan pada poin (v) hingga mencapai simpul daun. Kemudian, solusi bisa disusun dari simpul daun hingga ke akar.

C. Algoritma Brute Force

Algoritma *brute force* adalah algoritma yang menggunakan pendekatan *straightforward*, atau dengan kata lain, algoritma yang memecahkan suatu masalah dengan sederhana dan secara langsung. Algoritma ini semata-mata bergantung pada kecepatan komputasi dan mencoba semua kemungkinan tanpa menggunakan teknik yang mutahir untuk peningkatan efisiensi.

Algoritma ini tepat digunakan untuk persoalan dengan ukuran masukan, data, atau informasi yang diproses yang kecil, misalnya pencocokan *string* yang berukuran kurang dari 10. Hal ini terjadi karena secara umum, pada kasus dengan ukuran kecil, tidak terdapat perbedaan yang signifikan antara algoritma dengan *big-o-notation* yang tidak efisien dengan yang cukup efisien, perhatikan grafik berikut:



Untuk penyederhanaan ilustrasi, misalnya, algoritma *bruteforce* $O(n!)$ dan algoritma lain yang lebih efisien $O(n^2)$ dan $O(n)$. Pada ukuran masukan 4, waktu yang dibutuhkan berturut-turut 4, 16, dan 24 nanosekon. Pada kasus ini, terlihat bahwa perbedaan tidak signifikan, namun pada kasus yang besar misal ukuran masukan 100, waktu yang dibutuhkan berturut-turut 10, 100, dan 3.628.800 nanosekon. Pada ukuran besar seperti ini, terlihat perbedaan waktu yang signifikan.

Secara detail, penyelesaian kasus *brute force* pada permasalahan TSP mengikuti langkah-langkah berikut:

1. Lakukan enumerasi dari setiap kemungkinan urutan

jalur. Misal kota awal adalah kota A dan kota-kota yang akan dikunjungi adalah kota B, C, dan D. Maka, terdapat 3! kemungkinan urutan, yakni:

1. A – B – C – D – A
2. A – B – D – C – A
3. A – C – B– D – A
4. A – C – D – B – A
5. A – D – B – C – A
6. A – D – C – B – A

2. Dari semua kemungkinan urutan, hitung masing-masing *cost* dengan menjumlahkan *cost* tiap sisi atau jalur yang menghubungkan antar kota, misal pada enumerasi pertama, dihitung *cost* A ke B + *cost* B ke C + *cost* C ke D + *cost* D ke A.
3. Dari semua enumerasi, pilih satu yang memberikan *total cost* teroptimal (terendah).

III. Implementasi dan Analisis

A. *Implementasi Program*

Untuk penyederhanaan kasus, implementasi program dilakukan menggunakan bahasa Python. Proses implementasi terdiri dari persiapan Google APIs *credential*, persiapan *library* atau *dependencies* Python, dan pembuatan modul yang berisi fungsi-fungsi yang akan dibutuhkan untuk proses pengambilan data menggunakan Google API *distance matrix* serta proses enumerasi, kalkulasi, dan seleksi kasus optimal.

Pada tahap pertama, yakni persiapan Google APIs *credential*, diperlukan pembuatan akun Google, lalu pada Google Cloud Platform, buat proyek. Setelah membuat proyek, aktifkan APIs & Services pada https://console.cloud.google.com/apis/library/distance-matrix-backend.googleapis.com dengan menekan tombol *enable*. Selanjutnya, buat *credential* pada https://console.cloud.google.com/apis/credentials dan simpan API Key.

Pada tahap selanjutnya, yakni persiapan *dependencies*, lakukan instalasi dengan memasukan *command* berikut pada terminal:

```
pip3 install googlemaps
```

Sebelum pembuatan modul, diperlukan penghubung ke Google API dengan *source code* berikut:

```python
import googlemaps
import re
from itertools import permutations

API_KEY = '<API Key yang di-generate saat membuat credential>'

gmaps = googlemaps.Client(API_KEY)
```

Selanjutnya, pada tahap pembuatan modul, dibutuhkan fungsi-fungsi sebagai berikut:

1. Fungsionalitas untuk mem-*prompt* pengguna pada terminal untuk memasukan tempat atau titik awal dan tempat-tempat kos yang akan dikunjungi

```python
def promptOriginPoint():
    origin = input("Masukan tempat awal kamu: ")
    print(f"Alamat yang kamu masukan: {gmaps.geocode(origin)[0][
'formatted_address']}")
    return origin
```

```python
def promptKosanPoint():
    amountKos = int(input("Masukan banyaknya kos yang ingin kamu survey: "))
    amountPlaces = amountKos + 1
    semuaKos = []

    for i in range(amountKos):
        tempKos = input(f"Masukan kos ke-{i + 1}: ")
        semuaKos.append(tempKos)
        print(f"Alamat yang kamu masukan: {gmaps.geocode(tempKos)[0][
'formatted_address']}")

    return amountPlaces, semuaKos
```

```python
def promptChoice():
    print("Opsi optimasi: \n1. Berdasarkan Jarak\n2. Berdasarkan waktu")
    pilihan = int(input("Masukan pilihan (1 atau 2): "))
    if(pilihan == 1):
        return "distance"
    elif(pilihan == 2):
        return "duration"
```

2. Membuat *utility* untuk mendapat jarak dari suatu titik ke titik tertentu sesuai dengan data yang diperoleh dengan Google APIs & services

```python
def measureTool(origin, destination, choice):
    tempInfo = gmaps.distance_matrix(origin, destination)
    tempDistance = tempInfo['rows'][0]['elements'][0][choice]['text']
    result = float(re.sub("[^0-9.\-]","", tempDistance))
    return result
```

3. Membuat *utility* untuk men-generate *adjacency matrix* dari tiap lokasi

```python
def generateMatrix(origin, amountPlaces, semuaKos, choice):
    matrixAdj = [[0 for _ in range(amountPlaces)] for _ in range (amountPlaces)]
    for i in range(amountPlaces):
        for j in range(amountPlaces):
            if(i != j):
                if(i == 0):
                    matrixAdj[i][j] = measureTool(origin, semuaKos[j - 1], choice)
                else:
                    if(j == 0):
                        matrixAdj[i][j] = measureTool(semuaKos[i - 1], origin, choice)
                    else:
                        matrixAdj[i][j] = measureTool(semuaKos[i - 1], semuaKos[j - 1], choice)

    return matrixAdj
```

4. Membuat fungsionalitas untuk men-*generate* setiap enumerasi kemungkinan, mengkalkulasikan, dan menentukan hasil rute optimal menggunakan algoritma *brute force*

```python
def TSPBruteforce(matrix, r, size):
    global funcCall
    node = []
    dict = {}
    for i in range(size):
        if i != r:
            node.append(i)

    permutate = permutations(node)
    res = float('inf')
    for val in permutate:
        j = 0
        k = r
        for l in val:
            j += matrix[k][l]
            k = l
        j += matrix[k][r]
        res = min(j, res)
        dict[val] = res
    return
res, list(dict.keys())[list(dict.values()).index(res)]
```

5. Membuat fungsionalitas untuk menampilkan hasil kepada *user*

```
def TranslateResult(res, origin, semuaKos):
    print("Hasil optimasi rute:")
    path = []
    path.append(origin)
    for seq in res:
        path.append(semuaKos[seq - 1])
    path.append(origin)

    for i in range(len(path)):
        if i == len(path) - 1:
            print(path[i])
        else:
            print(path[i], end = " -> ")
```

6. Main program

```
if __name__ == '__main__':
    origin = promptOriginPoint()
    amountPlaces, dest = promptKosanPoint()
    userChoice = promptChoice()

    matrixAdj = generateMatrix(origin, amountPlaces, dest, userChoice)
    minimumVal, result = TSPBruteforce(matrixAdj, 0, amountPlaces)
    TranslateResult(result, origin, dest)
    print(f"Total: {minimumVal} {'km' if userChoice == 'distance' else 'minutes'}")
```

B. Uji Kasus

Pengujian dilakukan dengan menerapkan **kasus uji 1** sebagai berikut:

Data uji:

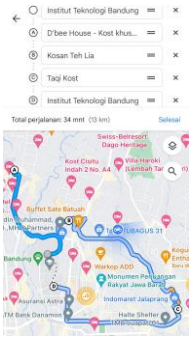
Lokasi awal	Institut Teknologi Bandung
Kos pertama	Kost Teh Lia Bandung Coblong
Kos kedua	Kost Apik Teduh Sukaluyu
Kos ketiga	Kost Apik UNPAR Ciumbeleuit
Opsi optimasi	1 (Jarak)

```
Masukan tempat awal kamu: Institut Teknologi Bandung
Alamat yang kamu masukan: Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong, Kota Bandung, Jawa Barat 40132, Indonesia
Masukan banyaknya kos yang ingin kamu survey: 3
Masukan kos ke-1: Kost Teh Lia Bandung Coblong
Alamat yang kamu masukan: Jl. L. L. R.E. Martadinata No.104, Cihapit, Kec. Bandung Wetan, Kota Bandung, Jawa Barat 40114
, Indonesia
Masukan kos ke-2: Kost Apik Teduh Sukaluyu Bandung Cibeunying
Alamat yang kamu masukan: Jl. Cikutra No.171, Neglasari, Kec. Cibeunying Kaler, Kota Bandung, Jawa Barat 40124, Indonesi
a
Masukan kos ke-3: Kost Apik UNPAR Bandung Ciumbeleuit
Alamat yang kamu masukan: Jl. Ciumbuleuit No.94, Hegarmanah, Kec. Cidadap, Kota Bandung, Jawa Barat 40141, Indonesia
Opsi optimasi:
1. Berdasarkan Jarak
2. Berdasarkan waktu
Masukan pilihan (1 atau 2): 1
```

Hasil keluaran*:*

```
Hasil optimasi rute:
Institut Teknologi Bandung -> Kost Apik UNPAR Bandung Ciumbeleuit -> Kost Teh Lia Bandung Coblong -> Kost Apik Teduh Suk
aluyu Bandung Cibeunying -> Institut Teknologi Bandung
Total: 13.100000000000001 km
```

Perbandingan hasil dengan *keyword* pencarian yang sama:



Gambar 5. Tampilan menggunakan fitur pada Google Maps menghasilkan ukuran jarak yang sama (13 km)

Jika dilakukan pengacakan pada aplikasi Google Maps pada sistem operasi Android, didapatkan jarak tempuh lainnya antara lain:

1. ITB – D'bee House – Taqi Kost – Kosan Teh Lia – ITB: 15 km
2. ITB – Taqi Kost – D'bee House – Kosan Teh Lia – ITB: 13 km
3. ITB – Taqi Kost – Kosan Teh Lia– D'bee House – ITB: 14 km
4. ITB – Kosan Teh Lia – Taqi Kost – D'bee House – ITB: 14 km
5. ITB – Kosan Teh Lia – D'bee House – Taqi Kost – ITB: 14 km

Berdasarkan hasil dari kasus tersebut, disimpulkan bahwa hasil yang diperoleh program dan hasil pada poin 2 memiliki ukuran jarak yang sama, yakni 13 km, sehingga hanya diambil salah satu, atau terdapat kemungkinan bahwa terdapat perbedaan pada angka di belakang koma (tidak ditampilkan pada tampilan antarmuka Google Maps). Namun, hasil yang diperoleh merupakan hasil optimal dibandingkan kasus lain, yakni 13 km.

Untuk **kasus uji 2**, optimasi dilakukan berdasarkan waktu

Data uji:

Lokasi awal	Hotel Sheraton Bandung
Kos pertama	Gg. Bukit Jarian IV No.15/165 D
Kos kedua	Jl. Kebun Binatang No.24/56 RT 3 RW 7
Kos ketiga	Jl. Sadang luhur 3 no10 Sadang serang
Opsi optimasi	2 (Waktu atau durasi)

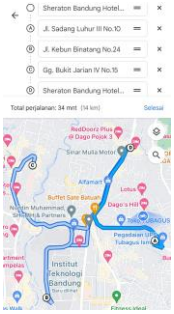
```
Masukan tempat awal kamu: Hotel Sheraton Bandung
Alamat yang kamu masukan: Jl. Ir. H. Juanda No.390, Dago, Kecamatan Coblong, Kota Bandung, Jawa Barat 40135, Indonesia
Masukan banyaknya kos yang ingin kamu survey: 3
Masukan kos ke-1: Gg. Bukit Jarian IV No.15/165 D
Alamat yang kamu masukan: Gg. Bukit Jarian IV No.15, Hegarmanah, Kec. Cidadap, Kota Bandung, Jawa Barat 40141, Indonesia

Masukan kos ke-2: Jl. Kebun Binatang No.24/56 RT 3 RW 7
Alamat yang kamu masukan: 3, Jl. Kebun Binatang No.24, Lb. Siliwangi, Kecamatan Coblong, Kota Bandung, Jawa Barat 40132,
 Indonesia
Masukan kos ke-3: Jl. Sadang luhur 3 no10 Sadang serang
Alamat yang kamu masukan: Jl. Sadang Luhur No.3, Sekeloa, Kecamatan Coblong, Kota Bandung, Jawa Barat 40134, Indonesia
Opsi optimasi:
1. Berdasarkan Jarak
2. Berdasarkan waktu
Masukan pilihan (1 atau 2): 2
```

Pada data uji ini, alih-alih menggunakan *keyword* nama kos secara eksplisit, dimasukkan data alamat yang diperoleh dari aplikasi Mamikos dari beberapa kos yang ada.

Hasil keluaran*:*

```
Hasil optimasi rute:
Hotel Sheraton Bandung -> Jl. Sadang luhur 3 no10 Sadang serang  -> Jl. Kebun Binatang No.24/56 RT 3 RW 7 -> Gg. Bukit J
arian IV No.15/165 D -> Hotel Sheraton Bandung
Total: 40.0 minutes
```



Gambar 6. Tampilan menggunakan fitur pada Google Maps menghasilkan ukuran waktu yang berbeda (terdapat selisih 6 menit)

Jika dilakukan pengacakan pada aplikasi Google Maps pada sistem operasi Android, didapatkan waktu tempuh lainnya antara lain:

1. Hotel Sheraton Bandung – Jl. Sadang Luhur 3 – Jl. Kebun Binatang No. 24 – Gg. Bukit Jarian IV – Hotel Sheraton: 34 menit
2. Hotel Sheraton Bandung – Jl. Sadang Luhur 3 – Gg. Bukit Jarian IV– Jl. Kebun Binatang No. 24 – Hotel Sheraton: 35 menit
3. Hotel Sheraton Bandung – Gg. Bukit Jarian IV – Jl. Kebun Binatang No. 24 – Jl. Sadang Luhur 3– Hotel Sheraton: 37 menit
4. Hotel Sheraton Bandung – Gg. Bukit Jarian IV – Jl. Sadang Luhur 3– Jl. Kebun Binatang No. 24 – Hotel Sheraton: 37 menit
5. Hotel Sheraton Bandung – Jl. Kebun Binatang No. 24 – Jl. Sadang Luhur 3 – Gg. Bukit Jarian IV – Hotel Sheraton: 38 menit
6. Hotel Sheraton Bandung – Jl. Kebun Binatang No. 24 – Gg. Bukit Jarian IV– Jl. Sadang Luhur 3 – Hotel Sheraton: 36 menit

Berdasarkan hasil dari kasus kedua tersebut, disimpulkan bahwa hasil yang diperoleh program dan hasil pada penggunaan fitur Google Maps secara langsung pada aplikasi memiliki ukuran waktu yang memiliki perbedaan $|40 - 34| = 6$ menit. Namun, jika dilihat pada keenam hasil yang diperoleh pada aplikasi, hasil optimal didapatkan pada urutan tempat pada poin 1, yakni sesuai dengan hasil yang diperoleh dengan program.

IV. Kesimpulan

Penerapan algoritma *brute force* pada aplikasi nyata kasus TSP, yakni optimasi jalur berdasarkan jarak atau waktu dalam memilih kos dapat memanfaatkan salah satu produk pada Google APIs & Services, yakni *distance matrix*. Berdasarkan kedua uji kasus tersebut, hasil yang didapatkan dengan menggunakan algoritma *brute force* pada fungsi `TSPBruteforce` yang ada pada program menghasilkan hasil yang optimal. Namun demikian, terdapat *error* pada uji kasus kedua dalam penghitungan total durasi perjalanan, yakni sebesar 6 menit. Program ini dapat dimanfaatkan bagi para pencari kos, khususnya mahasiswa baru dalam memilih urutan kos yang akan dikunjungi.

Saran untuk para peneliti dalam melakukan penelitian selanjutnya adalah optimasi pencarian jalur dengan teknik yang lebih optimal, salah satunya dengan memanfaatkan *dynamic programming*.

V. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah menganugerahkan kesempatan untuk penulis bisa melakukan penulisan makalah ini. Tidak lupa juga saya ucapkan pada para dosen –Ibu Masayu Leylia Khodra, Bapak Rinaldi Munir, dan Ibu Nur Ulfa Maulidevi– yang telah membimbing penulis dengan memberikan ilmu kepada saya. Penulis juga mengucap syukur untuk keluarga yang telah mendukung dengan memberikan bantuan materi ataupun moril untuk saya bisa mengikuti pembelajaran perkuliahan. Turut saya ucapkan terima kasih kepada teman-teman yang telah memberi dukungan moril dalam pengerjaan laporan ini.

Video Link at Youtube

https://www.youtube.com/watch?v=brlF6KuAK00

References

[1] https://towardsdatascience.com/how-to-solve-the-traveling-salesman-problem-a-comparative-analysis-39056a916c9f, diakses pada tanggal 22 Mei 2022 pukul 14.22 WIB.

[2] https://developers.google.com/maps/documentation/distance-matrix, diakses pada tanggal 22 Mei 2022 pukul 16.55 WIB.

[3] David L. Applegate, *The Travelling Salesman Problem: A Computational Study.* New Jersey: Princeton University Press, 2006, pp. 1-2.

[4] https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/, diakses pada tanggal 23 Mei 2022 pukul 16.25 WIB.

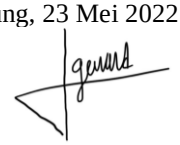
[5] R. Munir, *Algoritma Brute Force (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf, diakses pada tanggal 23 Mei 2022 pukul 16.47 WIB.

[6] R. Munir, *Algoritma Branch and Bound (Bagian 2)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf, diakses pada tanggal 23 Mei 2022 pukul 16.47 WIB.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 23 Mei 2022



Gerald Abraham Sianturi - 13520138