Penerapan Merge Sort untuk Mencari Transportasi Massal yang Paling Efisien di Jakarta

Muhammad Rafi Zhafran
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
liInstitut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
m_rafi_z@hotmail.com

***Abstract*—Makalah ini menggambarkan salah satu masalah yang umum terjadi pada pengguna transportasi massal (KRL, Transjakarta, dan MRT) di Jakarta. Dengan banyaknya pilihan yang ada, pengguna terkadang bingung untuk memilih transportasi massal mana yang memiliki biaya total perjalanan yang minimum dari satu tempat ke tempat lainnya. Biaya total perjalanan yang dimaksud adalah harga tiket transportasi massal ditambah dengan perikiraan biaya ojek online dari stasiun terdekat ke tempat tujuan. *Merge Sort* akan digunakan sebagai alat untuk mencari transportasi massal mana yang paling efisien bagi pengguna dengan bantuan Google Maps, Go-Jek, dan Grab.**

Keywords—Merge Sort, Distance Matrix API, Transportasi Massal, KRL, MRT, Busway, Go-Jek, Grab

I. PENDAHULUAN

Jakarta merupakan ibukota Negara dan kota terbesar di Indonesia. Sebagai pusat bisnis, hiburan, politik dan kebudayaan, kantor pusat BUMN, perusahaan swasta, dan perusahaan asing hampir semuanya ada dan berpusat di Jakarta. Ditambah mall-mall besar yang selalu ramai tidak hanya pada akhir pekan, namun juga hari kerja. Jakarta sebagai jantung dari Indonesia ini menyebabkan banyaknya warga dari luar Jakarta ikut mengadu nasib di Jakarta yang mengakibatkan Jakarta semakin macet.

Menurut data dari Dinas Perhubungan, terdapat 46 kawasan dengan 100 titik simpang rawan macet di Jakarta. Definisi rawan macet disini adalah arus tidak stabil, kecepatan rendah, dan antrean panjang. Hal ini mendorong pemerintah untuk menyediakan transportasi massal yang bertujuan untuk mengurangi kemacetan di Jakarta. Dalam karya tulis ini penulis hanya akan mencantumkan tiga transportasi massal yaitu Transjakarta, KRL, dan MRT.

Dikarenakan adanya Transjakarta, KRL, dan MRT, warga Jakarta terkadang kebingungan untuk memilih transportasi massal mana yang paling efisien untuk menuju ke tempat tujuannya. Hal ini umum terjadi pada warga yang jarang menggunakan transportasi massal, ketika pengguna ingin menuju tempat yang jarang ia tuju, dan/atau apabila muncul sebuah transportasi massal baru, dalam hal ini contohnya MRT.

Warga Jakarta sangat dipermudah dengan adanya ojek online. Warga yang dulu kesulitan mencari transportasi ketika sudah sampai di stasiun tujuan, sekarang cukup memesan lewat smartphone masing-masing. Dengan adanya ojek online maka penulis akan menambahkan aspek harga ojek online dari stasiun ke tempat tujuan ke dalam perhitungan agar didapat transportasi massal mana yang paling efisien dalam hal pengeluaran uang.

II. TEORI DASAR

2.1 Merge Sort

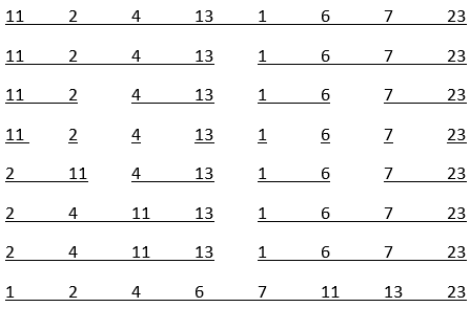
Merge Sort merupakan algoritma pengurutan yang menjadi bagian dari *Divide and Conquer*. Secara garis besar, *Merge Sort* membagi sebuah larik menjadi dua, lalu pembagian ini diulangi terus menerus sampai atomik atau masalah tidak dapat dipecah lagi.

Merge Sort merupakan algoritma pengurutan yang paling efisien dengan kompleksitas $T(n) = O(n\ ^2\log n)$. Berikut adalah penjelasan algoritmanya.

Algoritma:

1. Untuk n = 1, maka *table* sudah terurut
2. Untuk n > 1, maka
   a. Bagi larik A menjadi dua bagian, bagian kiri dan bagian kanan, masing-masing berukuran n/2 elemen
   b. Ulangi step a sampai elemen tidak dapat dibagi lagi (n = 1)
   c. Gabungkan masing-masing elemen sembari diurutkan
   d. Ulangi step c sampai larik kembali seperti semula dengan terurut

Gambaran:



2.2 *Distance Matrix API*

Untuk mengambil data dari Google Maps, penulis menggunakan *Distance Matrix API* yang disediakan oleh *Google Developer*. *Distance Matrix API* menyediakan jarak perjalanan dan waktu yang ditempuh dari titik asal ke tujuan, lalu menghasilkan informasi berdasarkan rute yang direkomendasikan yang dikalkulasi oleh *Google Maps API*. Informasi yang dihasilkan ini terdiri dari baris-baris yang berisi durasi dan jarak dari tiap perjalanan.

Bentuk request *Distance Matrix API*:

```
https://maps.googleapis.com/maps/api/distan
cematrix/outputFormat?parameters
```

Nilai outputFormat berbentuk *json* (*JavaScript Object Notation*).

2.2.1 Parameter Wajib

Ada beberapa parameter masukan yang diperlukan oleh *Distance Matrix API*:

1. Titik asal: Titik asal yang dicantumkan dapat lebih dari satu dan dipisahkan oleh '|'. Berikut adalah jenis masukan titik asal yang dapat diterima oleh *Distance Matrix API*:
   a. Alamat: jika masukan berbentuk alamat maka alamat yang dimasukan akan diubah menjadi kordinat oleh *Geocoding API*, lalu akan diproses oleh *Distance Matrix API*. Berikut contoh masukan berbentuk alamat:

      ```
      origins=Bobcaygeon+ON|24+Sussex+D
      rive+Ottawa+ON
      ```

   b. Kordinat: jika masukan berbentuk kordinat maka akan langsung diproses oleh *Distance Matrix API*. Berikut adalah contoh masukan berbentuk kordinat:

      ```
      origins=41.43206,-81.38992|-
      33.86748,151.20699
      ```

   c. *ID* dari sebuah tempat: jika masukan berbentuk *ID* maka harus diawali dengan "place_id". *ID* dapat diambil dari *Geocoding API* dan *Places API*. Berikut adalah contoh masukan berbentuk *ID*:

      ```
      origins=place_id:ChIJ3S-
      JXmauEmsRUcIaWtf4MzE
      ```

2. Destinasi: Titik destinasi yang dicantumkan dapat lebih dari satu dan dipisahkan oleh '|' Berikut adalah jenis masukan titik asal yang dapat diterima oleh *Distance Matrix API*:
   a. Alamat: jika masukan berbentuk alamat maka alamat yang dimasukan akan diubah menjadi kordinat oleh *Geocoding API*, lalu akan diproses oleh *Distance Matrix API*. Berikut contoh masukan berbentuk alamat:

      ```
      destinations=Bobcaygeon+ON|24+Sus
      sex+Drive+Ottawa+ON
      ```

   b. Kordinat: jika masukan berbentuk kordinat maka akan langsung diproses oleh *Distance Matrix API*. Berikut adalah contoh masukan berbentuk kordinat:

      ```
      destinations=41.43206,-81.38992|-
      33.86748,151.20699
      ```

   c. *ID* dari sebuah tempat: jika masukan berbentuk *ID* maka harus diawali dengan "place_id". *ID* dapat diambil dari *Geocoding API* dan *Places API*. Berikut adalah contoh masukan berbentuk *ID*:

      ```
      destinations=place_id:ChIJ3S-
      JXmauEmsRUcIaWtf4MzE
      ```

3. *Key*: *API key* dari program ini.

2.2.2 Parameter Tambahan

Ada beberapa parameter tambahan yang perlu dimasukan ke *Distance Matrix API* untuk program ini, yaitu mode perjalanan yang *default*-nya adalah mobil. Namun, dapat diubah menjadi mobil, motor, berjalan kaki, sepeda, dan transit.

2.2.3 Batasan

Dikarenakan perhitungan hanya akan membandingkan harga ojek *online* maka harus dibuat sebuah batasan. Batasan yang akan digunakan adalah:

```
avoid=highways
```

Batasan diatas berarti membatasi untuk menghindari jalan tol.

2.2.4 Unit

Unit yang dipakai secara *default* adalah kilometer dan meter.

2.2.5 Respon *Distance Matrix API*

Keluaran dari *Distance Matrix API* akan memiliki format yang ditandai oleh '*output'* flag yang ada di dalam *URL request*.

Contoh *request*:

```
https://maps.googleapis.com/maps/api/distan
cematrix/json?origins=Vancouver+BC|Seattle&
destinations=San+Francisco|Victoria+BC&mode
=bicycling&language=fr-FR&key=YOUR_API_KEY
```

Contoh keluaran dalam bentuk *json*:


```
{
  "status": "OK",
  "origin_addresses":  [  "Vancouver,  BC,
Canada", "Seattle, État de Washington, États-
Unis" ],
  "destination_addresses": [ "San Francisco,
Californie,  États-Unis",  "Victoria,  BC,
Canada" ],
  "rows": [ {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 340110,
        "text": "3 jours 22 heures"
      },
      "distance": {
        "value": 1734542,
        "text": "1 735 km"
      }
    }, {
      "status": "OK",
      "duration": {
        "value": 24487,
        "text": "6 heures 48 minutes"
      },
      "distance": {
        "value": 129324,
        "text": "129 km"
      }
    } ]
  }, {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 288834,
        "text": "3 jours 8 heures"
      },
      "distance": {
        "value": 1489604,
        "text": "1 490 km"
      }
    }, {
      "status": "OK",
      "duration": {
        "value": 14388,
        "text": "4 heures 0 minutes"
      },
      "distance": {
        "value": 135822,
        "text": "136 km"
      }
    } ]
  } ]
}
```


Respon *Distance Matrix*:

1. Status: mengandung respon program saat diberi request. Ada beberapa jenis status:
   a. OK menandakan hasilnya valid
   b. INVALID_REQUEST menandakan *request*-nya *invalid*
   c. MAX_ELEMENTS_EXCEEDED menandakan produk titik asal dan destinasi melebihi batas *query* yang ada
   d. OVER_DAILY_LIMIT menandakan:
      - *API key invalid* atau tidak dimasukan
      - Akun belum membayar
      - Melebihi batas penggunaan
      - Metode pembayaran sudah tidak valid
   e. OVER_QUERY_LIMIT menandakan *Distance Matrix API* menerima terlalu banyak *request* dalam waktu tertentu
   f. REQUEST_DENIED menandakan program menolak penggunaan *Distance Matrix API*
   g. UNKNOWN_ERROR menandakan *server error*
   h. NOT_FOUND menandakan titik asal dan destinasi tidak dapat dideteksi dengan *Geocoding API*
   i. ZERO_RESULTS menandakan tidak ada rute dari titik asal ke destinasi
   j. MAX_ROUTE_LENGTH_EXCEEDED menandakan rute nya terlalu panjang dan tidak dapat diproses
2. Origin_addresses: mengandung kumpulan dari titik awal
3. Destination_addresses: mengandung kumpulan dari titik tujuan
4. *Rows*: mengandung elemen yang berisi status, *duration*, dan *distance*.

III. ANALISIS

3.1 *Input*

Program ini menerima masukan berupa dua buah titik, yaitu titik awal dan akhir. Kedua titik ini lalu akan dijadikan input untuk request ke *Distance Matrix API*, lalu program akan mengambil durasi dan jarak yang dihasilkan oleh *Distance Matrix API*. Contoh program menerima *input*:

```
asal = input("Masukan titik asal: ") #masukan
berupa nama tempat
destinasi = input("Masukan titik destinasi:
") #masukan berupa nama tempat
```

3.2 Database Stasiun KRL, MRT, dan Transjakarta

Ada tiga buah *database* yang dibuat yaitu *database* stasiun KRL, MRT, dan Transjakarta yang ada di Jakarta. Berikut adalah *database*-nya:

a. *Database* stasiun KRL di Jakarta

KRL merupakan salah satu transportasi massal yang umum digunakan oleh warga Jakarta. Contoh *database* stasiun KRL di Jakarta:

```
Rawa Buaya
Bojong Indah
Taman Kota
Pesing
Grogol
Palmerah
Kebayoran
Duri
Tanah Abang
Karet
Sudirman
Kampung Bandan
Tanjung Priok
Ancol
```

b. *Database* stasiun MRT di Jakarta
MRT merupakan transportasi massal yang paling baru diluncurkan oleh pemerintah dan warga Jakarta tampak memiliki antusias yang tinggi. Karena itulah MRT dijadikan salah satu transportasi massal yang dipertimbangkan. Berikut contoh *database* MRT di Jakarta:

```
Kampung Bandan
Kota
Glodok
Mangga Besar
Sawah Besar
Harmoni
Monas
Sarinah
Bundaran HI
Dukuh Atas
Setiabudi
Benhil
Istora
Senayan
```

c. *Database* halte Transjakarta di Jakarta
Transjakarta merupakan transportasi yang paling umum digunakan warga Jakarta. Selain itu diantara ketiga transportasi massal yang penulis perhitungkan ini, Transjakarta merupakan transportasi massal dengan stasiun/halte dengan jumlah terbanyak di Jakarta. Berikut contoh *database* Transjakarta:

```
Lebak Bulus
Pondok Pinang
Pondok Indah 1
Pondok Indah 2
Tanah Kusir Kodim
Kebayoran Lama Bungur
Pasar Kebayoran Lama
Simprug
Permata Hijau
Permata Hijau RS. Medika
Pos Pengumben
Kelapa Dua Sasak
Kebon Jeruk
Duri Kepa
```

3.3 Request *Distance Matrix API*

Setelah program menerima input dari pengguna berupa **asal** dan **destinasi**, kedua titik tersebut dijadikan acuan untuk mengirimkan *request*. Untuk tiap kali pengecekan akan dihitung jarak dari **asal** ke semua stasiun lalu dihitung pula jarak dari semua stasiun ke **destinasi.** Berikut adalah contoh programnya:

```
url1 =
"https://maps.googleapis.com/maps/api/dista
ncematrix/json?origins=Pondok+Indah&destina
tions=Kebayoran&mode=walking&language=fr-
FR&key=YOUR_API_KEY"
url2 =
"https://maps.googleapis.com/maps/api/dista
ncematrix/json?origins=Palmerah&destination
s=Senayan+City&mode=walking&language=fr-
FR&key=YOUR_API_KEY"

req1 = requests.get(url1)
req2 = requests.get(url2)
req1 = req1.json()
req2 = req2.json()
```

3.4 Pengambilan Data dari *Response Distance Matrix API*

Setelah program mengirimkan *request* ke *Distance Matrix API* maka *Distance Matrix API* akan mengirimkan respon dalam format *Json*. Berikut adalah contoh respon dalam format *Json*:

```
{
  "status": "OK",
  "origin_addresses": [ "Pondok Indah" ],
  "destination_addresses": [ "Kebayoran" ],
  "rows": [ {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 16,
        "text": "16 minutes"
      },
      "distance": {
        "value": 4312,
        "text": "4 km"
      }
    }
  } ]
}
```

```
{
  "status": "OK",
  "origin_addresses": [ "Palmerah" ],
  "destination_addresses": [ "Senayang City"
],
  "rows": [ {
    "elements": [ {
      "status": "OK",
      "duration": {
        "value": 14,
        "text": "14 minutes"
      },
      "distance": {
        "value": 3743,
        "text": "3 km"
      }
    }
  } ]
}
```

Dari respon tersebut, program akan mengambil elemen distance lalu akan dimasukan ke dalam sebuah kumpulan objek yang masing-masing objeknya mengandung ([jenis], [asal], [stasiun berangkat], [jarak asal-berangkat], [stasiun turun], [destinasi], [jarak turun-destinasi], [biayaGojekAsalStNaik], [biayaGojekStTurunDestinasi]).

3.5 *Merge Sort* Data

Dari kumpulan objek tersebut akan diurutkan berdasarkan jarak terendah, namun karena penulis menggunakan tiga database maka pengurutan akan dipecah menjadi 4 tahap, yaitu *Merge Sort* Data KRL, MRT, dan Transjakarta.

Berikut adalah contoh Algoritma *Merge Sort*:

```
def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]

        mergeSort(L)
        mergeSort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            jarakA = L[i].jarakAsalStNaik +
L[i].jarakStTurunDestinasi
            jarakB = R[j].jarakAsalStNaik +
R[j].jarakStTurunDestinasi
            if jarakA < jarakB:
                arr[k] = L[i]
                i =  i + 1
            else:
                arr[k] = R[j]
                j = j + 1
            k = k + 1

        while i < len(L):
            arr[k] = L[i]
            i =  i + 1
            k = k + 1

        while j < len(R):
            arr[k] = R[j]
            j = j + 1
            k = k + 1
```

Berikut adalah contoh penggunaan *Merge Sort* Data KRL:

```
mergeSort(arrKRL)
arrTerdekat[0] = arrKRL[0]
```

Berikut adalah contoh penggunaan *Merge Sort* Data MRT:

```
mergeSort(arrMRT)
arrTerdekat[1] = arrMRT[0]
```

Berikut adalah contoh penggunaan *Merge Sort* Data Transjakarta:

```
mergeSort(arrTransjakarta)
arrTerdekat[2] = arrTransjakarta[0]
```

Lalu, elemen dengan jarak terendah dari masing-masing kategori akan diurutkan lagi dan hasilnya akan ditampilkan ke pengguna. Berikut adalah contoh *Merge Sort* Data dari ketiganya:

```
mergeSort(arrTerdekat)
```

3.6 Penghitungan Estimasi Tarif Ojek Online

Setelah ketiga elemen diurutkan maka program akan mengkalkulasi estimasi harga ojek online yang diperlukan dari stasiun turun ke destinasi. Terhitung mulai 1 Mei 2019 Kemenhub akan memberlakukan kebijakan baru mengenai tarif ojek online. Untuk daerah Jabodetabek, biaya minimalnya adalah Rp. 10.000 dengan penambahan Rp. 2000 tiap km-nya.

Program ini baru akan menghitung estimasi tarif ojek online apabila berjarak lebih dari satu kilometer. Apabila jaraknya kurang dari satu kilometer maka program akan menampilkan anjuran untuk berjalan kaki. Berikut adalah contoh program perhitungan estimasinya:

```
for i in (0,3) :
    if (arrTerdekat[i].jarakAsalStNaik >
1000) :
        arrTerdekat[i].biayaGojekAsalStNaik
= 10000
        arrTerdekat[i].biayaGojekAsalStNaik
= arrTerdekat[i].biayaGojekAsalStNaik +
```

```
arrTerdekat[i].jarakAsalStNaik / 1000 *
2000
    if
(arrTerdekat[i].jarakStTurunDestinasi >
1000) :

arrTerdekat[i].biayaGojekStTurunDestinasi  =
10000

arrTerdekat[i].biayaGojekStTurunDestinasi  =
arrTerdekat[i].biayaGojekStTurunDestinasi  +
arrTerdekat[i].jarakAsalStNaik / 1000 * 2000
```

3.7 *Output*

Output dari program ini berupa titik asal, stasiun naik, jarak dari titik asal ke stasiun naik, stasiun turun, destinasi, jarak dari stasiun turun ke stasiun destinasi, dan estimasi harga atau anjuran berjalan kaki. Berikut adalah contoh program pengeluaran outputnya:

```
for i in (0,3) :
    print ("Anda disarankan untuk menaiki
", arrTerdekat[i].jenis)
    print ("Berangkat dari stasiun ",
arrTerdekat[i].stNaik, " dan turun dari
stasiun ", arrTerdekat[i].stTurun)
    if (arrTerdekat[i].jarakAsalStNaik >
1000) :
        print ("Anda disarankan untuk
menggunakan ojek online dari titik asal ke
stasiun keberangkatan dengan estimasi harga
= ", arrTerdekat[i].biayaGojekAsalStNaik)
    else :
        print ("Anda disarankan untuk
berjalan kaki dari titik asal ke stasiun
keberangkatan")
    if
(arrTerdekat[i].jarakStTurunDestinasi >
1000) :
        print ("Anda disarankan untuk
menggunakan ojek online dari stasiun
kedatangan ke destinasi dengan estimasi
harga = ",
arrTerdekat[i].biayaGojekStTurunDestinasi)
    else :
        print ("Anda disarankan untuk
berjalan kaki dari stasiun kedatangan ke
destinasi")
```

IV. MASUKAN DAN KELUARAN

Masukan

Pertama-tama masukan titik asal dan destinasi, pada hal ini contohnya Pondok Indah dan Senayan City.

```
Masukan titik asal: Pondok Indah
Masukan titik destinasi: Senayan City
```

Keluaran

Setelah diproses oleh program maka program akan mengeluarkan saran-saran mengenai transportasi apa yang dapat pengguna gunakan.

```
Anda disarankan untuk menaiki Transjakarta
Berangkat dari stasiun pondok indah 1 dan turun di bundaran senayan
Anda disarankan untuk berjalan kaki dari stasiun kedatangan ke destinasi
Anda disarankan untuk berjalan kaki dari stasiun kedatangan ke destinasi
Anda disarankan untuk menaiki MRT
Berangkat dari stasiun fatmawati dan turun di stasiun senayan
Anda disarankan untuk menggunakan ojek online dari titik asal
ke stasiun keberangkatan dengan estimasi harga = 12000
Anda disarankan untuk berjalan kaki dari stasiun kedatangan ke destinasi
Anda disarankan untuk menaiki KRL
Berangkat dari stasiun kebayoran dan turun di stasiun palmerah
Anda disarankan untuk menggunakan ojek online dari titik asal
ke stasiun keberangkatan dengan estimasi harga = 12000
Anda disarankan untuk menggunakan ojek online dari stasiun kedatangan
ke destinasi dengan estimasi harga = 12000
```

V. KESIMPULAN

Banyaknya transportasi massal yang ada sangat membantu kemacetan di Jakarta. Selain itu, warga Jakarta juga memiliki lebih banyak opsi untuk berpergian. Dengan banyaknya opsi ini maka penulis berharap program yang penulis buat dapat membantu warga Jakarta agar tidak kebingungan lagi antara transportasi massal satu dengan lainnya.

Sejauh ini algoritma Merge Sort dapat bekerja dengan baik. Merge Sort merupakan algoritma pengurutan yang paling efisien, hal ini sangat membantu dalam mengoptimisasi program.

DAFTAR PUSTAKA

http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Divide-and-Conquer-(2018).pdf

https://developers.google.com/maps/documentation/distance-matrix/intro

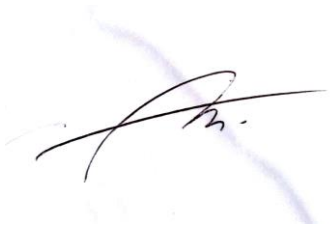
https://developers.google.com/maps/documentation/distance-matrix/start

http://makassar.tribunnews.com/2019/03/25/tarif-ojek-online-grab-gojek-terbaru-berlaku-mulai-1-mei-2019-termahal-rp-10-ribu-per-km?page=2

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 April 2019



Muhammad Rafi Zhafran/13517005