Teknik Optimasi Pencarian Lintasan Terbaik Tokyo Metro dengan Beberapa Simpul Wajib Kunjung Menggunakan Gabungan Algoritma Dijkstra dan Brute Force Permutation

Jesson Gosal Yo - 13519079
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519079@std.stei.itb.ac.id

***Abstract*—Tokyo Metro adalah sistem Major Rapid Transit atau biasa disingkat MRT di Tokyo,Jepang. Ada 9 jalur utama dan pada tahun 2020 tercatat sekitar 6,84 juta penumpang menggunakan Tokyo Metro setiap harinya. Jaringan Tokyo Metro melewati daerah-daerah perkantoran, perbelanjaan, hingga turisme dan berpusat di tengah Tokyo. Desain sistem Tokyo Metro yang diakui merupakan salah satu sistem MRT terkompleks di dunia dengan 179 stasiun dan 2702 kereta tidak menghentikan Tokyo Metro beroperasi dengan efisien dan tepat waktu. Namun kerumitan sistem bawah tanah ini dapat membingungkan bagi penggunanya terutama untuk turis. Tokyo Subway Navigation adalah sebuah aplikasi yang disediakan Tokyo Metro Co., Ltd yang bertujuan untuk memudahkan pengguna mendapat informasi rute antara dua stasiun. Aplikasi akan menampilkan rute, waktu yang diperlukan, beserta biaya transitnya. Namun ketika pengguna ingin bepergian ke beberapa stasiun sekaligus, hal ini masih menjadi masalah karena rute yang diambil pengguna bisa tidak efektif sehingga selain membuang waktu juga dapat memerlukan biaya yang lebih tinggi. Sistem Tokyo Metro dapat diabstraksi menjadi bentuk graf berbobot untuk dikalkulasi dan mendapat lintasan yang paling efektif dalam segi waktu atau segi biaya. Semua rute Tokyo Metro bekerja secara bolak-balik sehingga graf dapat direprentasikan dengan sisi tanpa arah. Di dalam makalah ini akan diberikan salah satu pendekatan penulis dalam penyelesaian masalah ini.**

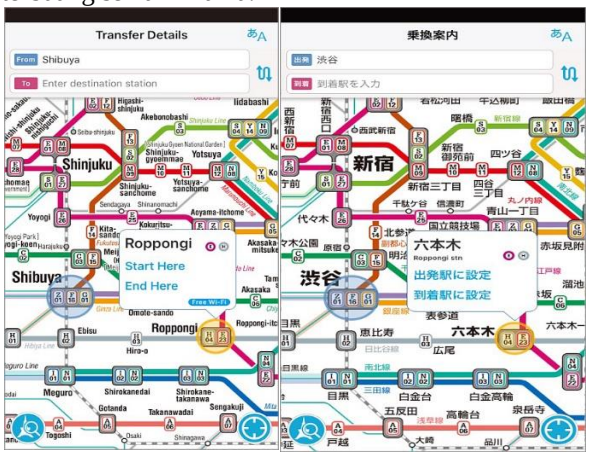
***Keywords*—Metro, rute, graf, dijkstra, lintasan.**

I. PENDAHULUAN

Dengan perkembangan zaman dan jumlah penduduk, dunia menjadi semakin sibuk dan padat. Salah satu tantangan dalam dunia modern ini adalah dalam hal transportasi. Waktu seorang pekerja dapat menjadi tidak efisien karena banyak terpotong di perjalanan. Metode transportasi yang cepat, efisien, dan aman dapat secara tidak langsung berpengaruh ke pertumbuhan ekonomi sebuah negara.MRT atau Major Rapid Transit menjadi salah satu solusi untuk memberikan pilihan transportasi yang cepat, efisien, dan aman.

Tokyo Metro yang merupakan sistem MRT di Tokyo,Jepang memiliki sejarah yang panjang.Jalur subway pertama di Jepang sudah dibuka sejak tahun 1927 yaitu jalur Tokyo Metro antara Ueno dan Asakusa yang berjarak sekitar 2,2 km. Pada masa ini ketika negara lain masih menggunakan kereta uap, Jepang sudah memulai pengembangan kereta listrik untuk digunakan sebagai kereta bawah tanah di masa depan. Kini, tercatat pada tahun makalah ini dibuat, Tokyo Metro sudah memiliki 179 stasiun dan 2702 kereta dengan jalur sepanjang 198,1 km yang terbagi dalam 9 jalur utama sudah menjangkau hampir seluruh kota Tokyo.

Tokyo Metro diakui oleh banyak pihak sebagai salah satu sistem MRT yang paling rumit di dunia. Hal ini dapat membingungkan pengguna Tokyo Metro terutama bagi turis asing. Untuk menangani hal ini, pihak Tokyo Metro Co., Ltd sudah menyediakan sebuah aplikasi bernama Tokyo Subway Navigation untuk mempermudah pengguna mencari rute perjalanan antara dua stasiun. Namun masalah kadang muncul ketika pengguna ingin bepergian ke beberapa titik sekaligus. Masalah muncul karena pengguna tidak tahu jalur mana yang paling efisien dalam segi waktu dan biaya. Jika pengguna salah memilih jalur maka bisa saja rute yang ditempuh menjadi berputar-putar mengakibatkan biaya transit tinggi dan waktu yang terbuang semakin lama.



Gambar 1 : Tokyo Metro App sumber[5]

Graf dalam matematika adalah sebuah struktur yang merepresentasikan set dari objek-objek yang objeknya dalam suatu artian dapat memiliki relasi atau hubungan.Graf adalah

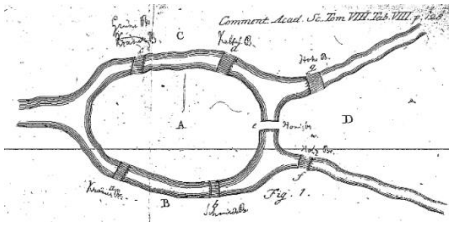
salah satu cara mengabstraksi sistem MRT dengan stasiun sebagai objeknya dan jalur antar stasiun sebagai relasinya. Dalam graf hal ini dapat diwujudkan dengan memisalkan stasiun sebagai simpul atau vertex dan jalur sebagai sisi. Tujuan dari abstraksi ini adalah agar bisa mengkomputasi permasalahan ini dengan algoritma-algoritma yang ada. Salah satu algoritma yang bisa dipakai untuk membantu menyelesaikan permasalahan ini adalah algoritma dijkstra.

Algoritma dijkstra ditemukan oleh Edsger Dijkstra dan dipublikasikan dalam makalahnya yang berjudul *A note on two problems in connexion with graphs.* Algoritma ini adalah jenis algoritma rakus (*greedy algorithm)* yang dapat digunakan untuk menyelesaikan permasalahan rute terpendek / *shortest path distance.* Algoritma ini dapat digunakan untuk mencari rute tersingkat antara dua buah stasiun. Dalam makalah ini akan dibahas modifikasi permasalahan sederhana mencari rute terpendek dari dua titik. Kini, rute terpendek harus dicari ketika ada lebih dari satu titik yang harus dikunjungi. Permasalahan ini mirip sekali dengan permasalahan *Travelling Salesman Problem* kecuali pada TSP setiap simpul hanya boleh dikunjungi maksimal 1 kali sedangkan untuk kasus ini semua simpul boleh dikunjungi sebanyak apapun.

II. LANDASAN TEORI

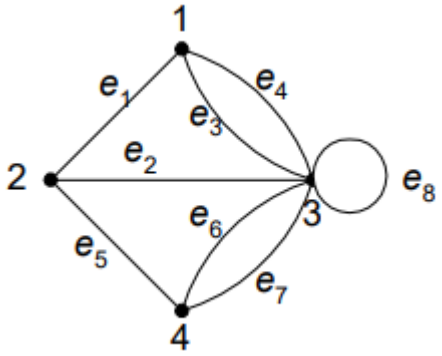
A. Graf

1. Sejarah Graf
   Penemuan graf bermula dari penyelesaian Euler terhadap permasalahan jembatan Konigsberg pada tahun 1735 yang berkembang menjadi pemahaman kita yang sekarang disebut graph eulerian.



Gambar 2. Jembatan Konigsberg sumber[6]

   Graf sendiri dapat digunakan untuk memodelkan berbagai permasalahan yang kompleks, seperti jaringan pertemanan di media sosial, struktur molekul, jaringan ekonomi, internet, peta dan masih banyak lagi.
2. Definisi Graf
   Graf yang dimaksud adalah graf dalam bidang matematika dan ilmu komputer. Graf adalah sebuah set yang menampung sekumpulan objek yang dalam suatu artian memiliki relasi satu sama lain. Secara formal, graf adalah set dari simpul-simpul yang memiliki hubungan biner antar simpul yang dapat dituliskan sebagai pasangan (V,E) dinotasikan dengan G(V,E). Dengan V adalah set simpul(*vertices/V)* dan E adalah set sisi (*edges/*E) yang menggambarkan hubungan biner tersebut. Representasi gambar sebuah graf dapat dilakukan dengan menggambarkan simpul sebagai titik dan sisi sebagai garis antar simpul.
3. Terminologi Graf
   Ada beberapa terminologi graf yang perlu diketahui.



Gambar 3. sumber[6]

   Pertama adalah loop, loop adalah anggota dari set E yang menghubungkan sebuah elemen yang sama dari V. Contoh dari loop adalah sisi e8 yang berawal dari simpul 3 dan mengarah ke simpul 3. Kemudian ada yang disebut sisi ganda. Jika ada lebih dari satu sisi yang ujung-ujung simpulnya sama, maka sisi ini disebut sisi ganda. Contoh dari sisi ganda adalah sisi yang menghubungkan 1 dan 3.

   Ketetanggaan adalah ketika dua buah simpul memiliki hubungan langsung. Contoh dari hal ini adalah simpul 2 dan 4 memiliki hubungan langsung yaitu dihubungkan dengan sisi e5. Ada juga yang disebut derajat. Derajat adalah banyaknya sisi pada simpul. Derajat simpul 2 adalah 3 karena sisi e1,e2,e5.

   Lalu ada lintasan dan sirkuit. Lintasan adalah sebuah sekuen dari sisi-sisi pada graf yang menghubungkan simpul pada graf. Sedangkan sirkuit adalah jenis lintasan yang memiliki simpul akhir dan simpul awal yang sama. Sekuen e5,e6,e3 adalah salah satu contoh dari lintasan 2 menuju 1. Sedangkan lintasan e1,e3,e2 disebut sirkuit karena lintasan tersebut berasal dari 2 dan berakhir di 2.

   a. Subgraf
      Sebuah graf G' disebut subgraf dari G jika dan hanya jika $E' \subseteq E$ dan $V' \subseteq V$
4. Kategorisasi Graf
   Berdasarkan keberadaan sisi ganda atau sisi gelang, graf dapat dibedakan menjadi dua yaitu graf sederhana (*simple graph*) dan graf tak sederhana(*unsimple graph)*. Graf sederhana adalah graf yang tidak memiliki loop atau sisi ganda sedangkan graf tak sederhana adalah graf yang memiliki loop dan atau sisi ganda.

   Berdasarkan orientasi arah sisi, graf dapat dibagi menjadi dua yaitu graf tak berarah (*undirected graph)* dan graf berarah(*directed graph)*. Graf tak berarah adalah graf yang sisi-sisinya tidak memiliki orientasi arah. Graf berarah adalah graf yang sisi-sisinya memiliki orientasi arah..

   Graf juga dapat dikategorikan berdasarkan ada atau tidaknya label bobot pada sisi graf. Graf

disebut berbobot (*weighted graph)* ketika sisi-sisinya dilabeli sebuah bobot numerik. Graf disebut tidak berbobot (*unweighted graph*) jika sisi-sisinya tidak dilabeli dengan bobot.

Ada juga beberapa graf khusus namun dalam makalah ini hanya akan memanfaatkan graf lengkap (*complete graph*) yaitu sebuah graf yang setiap simpulnya memiliki hubungan langsung dengan simpul lain manapun pada graf.

B. Algoritma Dijkstra

Algoritma Dijkstra adalah algoritma yang dipakai untuk menemukan lintasan terbaik dalam suatu graf berbobot dengan syarat bobot non negatif. Masalah yang diselesaikan dengan algoritma ini seringkali direferensikan sebagai permasalahan jalan terpendek/*shortest path problem.* Algoritma ini ditemukan oleh Edsger W. Dijkstra pada tahun 1956 dan dipublikasi 3 tahun kemudian pada jurnal yang berjudul *A note on two problems in connexion with graphs* . Algoritma ini digolongkan sebagai algoritma greedy/rakus. Algoritma rakus itu sendiri merupakan sebuah algoritma yang sering dipakai dalam masalah optimisasi. Algoritma disebut rakus karena pada setiap keputusan, dia akan selalu memilih pilihan dengan "*reward*" terbesar. Contohnya pada algoritma dijkstra, simpul yang dikunjungi adalah simpul terdekat yang dipilih dari queue yang berisi simpul-simpul yang belum pernah dikunjungi. Secara umum algoritma bekerja seperti pseudocode berikut jika menggunakan implementasi priority queue dan graf.

```
function dijkstra(graf, node_sumber)
    for each vertex in graf
        distance[i] <- INF
        parent[i] <- Nil
    distance[node_sumber] = 0
    while PQ is not empty
        u <- getMinimumDistanceNodeFromPQ
        dequeue u from PQ
        for each n of u // n adalah tetangga
            dist <- distance[u] + getDistance(u, n)
            if(dist < distance[n]) then
                distance[n] <- dist
                parent[n] <- u
    return (parent[],distance[])
```

Tabel 1. Pseudocode dari algoritma dijkstra

PQ merupakan priority queue yang awalnya diinisialiasi dengan semua simpul dari graf. Distance melambangkan jarak antara sebuah simpul dengan simpul sumber. Parent melambangkan simpul yang harus dilewati tepat sebelum sampai pada simpul tersebut. Implementasi kode di atas akan sekaligus mencatat *all pair shortest distance*. Langkah umum kode di atas adalah sebagai berikut:

1. Menginisiasi semua jarak menjadi tak hingga, parent menjadi nil, dan queue diisi dengan semua simpul pada graf
2. Mengunjungi simpul sumber dan mencatat jarak dari simpul sumber ke diri sendiri sebagai 0 dan membuat sumber menjadi current node.
3. Memperbaharui jarak simpul-simpul yang terhubung langsung relatif terhadap sumber dan menggantinya dengan jarak baru jika jarak yang baru lebih pendek dari jarak yang lama sekaligus memperbaharui parent.
4. Mengambil salah satu simpul dari queue yang memiliki hubungan langsung dan memiliki bobot terkecil dengan current node dan simpul ini akan menggantikan current node.
5. Langkah 3 dan 4 diulang terus menerus hingga queue kosong.
6. Pada akhirnya akan didapat data lengkap mengenai jarak minimum antar node-node yang berhubungan langsung serta parentnya.
7. Jarak terpendek antara 2 node manapun dapat dicari dengan melakukan traversal terhadap parent sebuah node tujuan hingga parent menunjuk ke sumber.

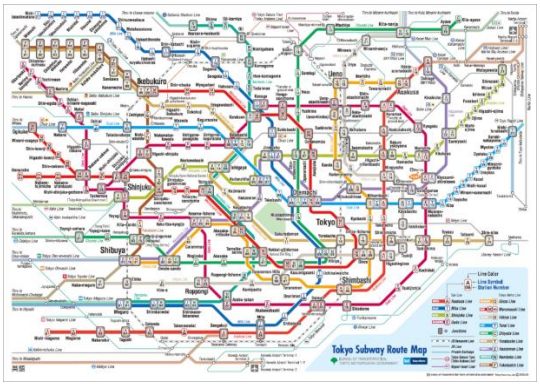
Dengan implementasi ini, algoritma dijkstra memiliki kompleksitas O(V + E log V) dengan V melambangkan jumlah simpul dan E melambangkan jumlah sisi. Algoritma dijkstra diharapkan mampu menangani jaringan Tokyo Metro yang memiliki 179 stasiun dalam waktu yang masih masuk akal.

C. Permutasi Brute Force

Algoritma brute force merupakan algoritma yang sederhana dan seringkali tidak efektif karena algoritma akan mencoba semua kemungkinan yang ada dan memilih kondisi yang sesuai dengan kriteria yang diberikan. Melakukan permutasi brute force terhadap peta Tokyo Metro secara langsung akan sangat tidak efisien dan tidak dapat diselesaikan dalam kurun waktu yang masuk akal karena kompleksitas algoritmanya tumbuh sebanding dengan O(n!).

Algoritma brute force memiliki keunggulan mudah diimplementasikan karena sangat intuitif. Algoritma ini akan diimplementasikan terhadap graf yang sudah direduksi menggunakan algoritma dijkstra.

III. PENERAPAN ALGORITMA DIJKSTRA DAN PERMUTASI BRUTE FORCE



Gambar 3. sumber[5]

Gambar di atas merupakan peta jaringan MRT di Tokyo Jepang yang terdiri dari Tokyo Metro Line dan Toei Line. Sebuah graf dapat dibentuk dari peta tersebut dengan

memisalkan stasiun sebagai simpul. Jika optimasi ingin dilakukan terhadap waktu (tidak memperhitungkan waktu keberangkatan kereta dan waktu jalan di dalam stasiun menggunakan data rata-rata dari sumber[5]) maka setiap rute dapat dimisalkan sebagai sisi dengan nilai bobot waktu tempuh. Masalah timbul ketika optimasi ingin dilakukan terhadap biaya karena sistem pemberian harga Tokyo Metro yang non-linier. Sebagai contoh misalkan ada jalur A → B → C dengan biaya antar stasiun 170 yen. Intuisi pertama setelah memberi bobot pada A → B dan B → C sebesar 170 yen adalah menyimpulkan A → C memiliki bobot total 340 yen. Hal ini tidak salah ketika kita berhenti di setiap 1 stasiun. Nyatanya A → C dapat memiliki biaya minimal 170 yen. Hal ini terjadi untuk beberapa stasiun yang terletak dalam satu sub-line yang sama (misal M18-Otemachi dengan M16-Ginza Marunouchi Line). Kasus ini dapat ditangani dengan menambahkan sisi langsung dari A → C, namun harus diingat bahwa kompleksitas algoritma dijkstra tumbuh bergantung kepada banyak E sehingga hal ini perlu diminimalisasi.

Awalnya penulis mengira bahwa untuk mengatasi masalah optimasi biaya, sebuah sisi perlu ditambahkan antar sebuah stasiun dengan stasiun-stasiun lainnya sehingga terbentuk graf lengkap dari stasiun-stasiun MRT Tokyo dan hal ini jelas akan sangat membebani algoritma yang digunakan. Untungnya setelah pencarian data lebih lanjut, kasus perbedaan harga di atas hanya dapat terjadi jika transit dilakukan pada line yang sama dan ukuran setiap line hanya berkisar maksimal hingga sekitar 20 stasiun.

Penjelasan permasalahan adalah sebagai berikut. Jika diberikan sejumlah simpul yang harus dikunjungi oleh seorang turis yang sedang berdiam di salah satu simpul maka akan dicari sebuah lintasan sehingga setiap simpul berhasil dikunjungi setidaknya satu kali . Algoritma akan mengoptimasi jalur turis sehingga bisa meminimisasi biaya transit MRT sang turis tanpa mempedulikan urutan stasiun.Misalkan seorang turis dalam suatu hari ingin berkunjung ke stasiun shibuya, ginza, dan shiodome maka apakah ada jalur sehingga turis bisa meminimisasi harga atau waktu perjalanan dengan syarat rute yang dilewati mengenai ketiga stasiun tersebut setidaknya satu kali.

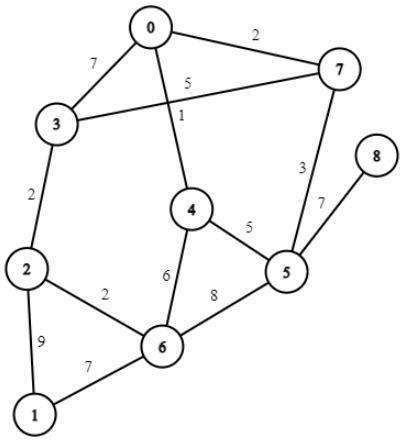
Pendekatan penulis adalah dengan mencari semua pasangan jalur terdekat antara simpul-simpul wajib kunjung dan simpul sumber. Cara mencari jalur terdekat ini adalah menggunakan algoritma dijkstra. Setelah mendapatkan semua pasangan jalur terdekat antara simpul, akan dibuat graf lengkap dari semua simpul wajib kunjung. Jika ada m buah simpul yang ingin dikunjungi maka akan terbentuk graf lengkap dengan m+1 simpul yang terdiri atas m simpul wajib kunjung dan 1 simpul sumber. Bobot dari sisi yang menghubungkan 2 simpul merepresentasikan jalur terdekat mereka. Graf lengkap yang terbentuk ini cukup unik karena selain lengkap, sisi yang menghubungkan sepasang simpul pastilah jalur tercepat yang menghubungkan mereka. Setelah graf lengkap terbentuk akan dilakukan permutasi untuk mendapatkan urutan simpul yang harus dikunjungi. Semua kemungkinan permutasi akan dibandingkan dan altenatif terpendek akan diambil.Jika diperhatikan, permutasi ini memiliki kompleksitas yang sangat buruk sehingga kompleksitas total algoritma sangat bergantung kepada nilai m (simpul wajib kunjung). Hal menyebabkan akan adanya *constrain* terhadap besar nilai m agar permasalahan dapat diselesaikan dalam waktu yang wajar.

IV. IMPLEMENTASI ALGORITMA

Implementasi algoritma dilakukan penulis dalam program Java. Cara kerja algoritma sebagai umum adalah mencatat semua *shortest distance pair* yang diperlukan dan menyimpan nilainya dalam matriks dan list sekaligus menjadi representasi graf lengkap baru yang diperlukan. Terakhir dilakukan permutasi dari elemen-elemen pada list dan dibandingkan bobot total masing-masing.

Sebagai contoh kasus, akan diuji dua buah contoh kasus. Kasus pertama adalah graf yang diciptakan secara acak dan kasus kedua menggunakan subgraf dari representasi graf Tokyo Metro karena jika menggunakan keseluruhan graf akan memerlukan waktu yang sangat lama dalam pencarian data. Namun secara teori, seharusnya algoritma masih mampu menangani keseluruhan peta Tokyo Metro dengan syarat jumlah simpul wajib kunjung kecil.

Sebagai kasus pertama, diberikan graf seperti pada gambar berikut.



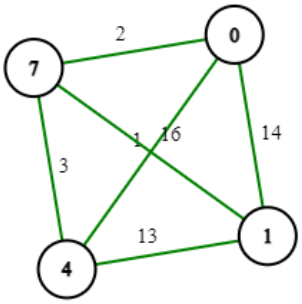
Gambar 4.

Untuk kasus ini, dimisalkan simpul sumber adalah 0 dan list simpul yang harus dikunjungi adalah 1,4, dan 7. Pertama akan dilakukan pencarian *shortest path* menggunakan algoritma dijkstra. Berikut adalah *screenshot* hasil perhitungan *shortest path* menggunakan program yang dibuat penulis dalam bahasa Java, *source code* dapat dilihat pada lampiran.

```
The shortest path from node :
0 to 0 is 0
0 to 1 is 14
0 to 4 is 1
0 to 7 is 2
The shortest path from node :
1 to 0 is 14
1 to 1 is 0
1 to 4 is 13
1 to 7 is 16
The shortest path from node :
4 to 0 is 1
4 to 1 is 13
4 to 4 is 0
4 to 7 is 3
The shortest path from node :
7 to 0 is 2
7 to 1 is 16
7 to 4 is 3
7 to 7 is 0
Permutations of sequence are:
=====================================
[1, 4, 7] -> 30
[1, 7, 4] -> 33
[4, 1, 7] -> 30
[4, 7, 1] -> 20
[7, 1, 4] -> 31
[7, 4, 1] -> 18
```

Gambar 5. Hasil perhitungan biaya lintasan dan permutasi

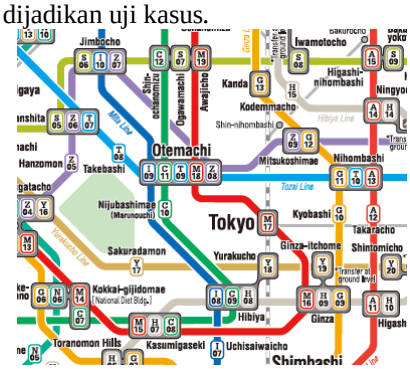
Dari data ini bisa dibangun sebuah graf lengkap dengan 4 buah simpul dan bobot yang melambangkan lintasan terpendek.



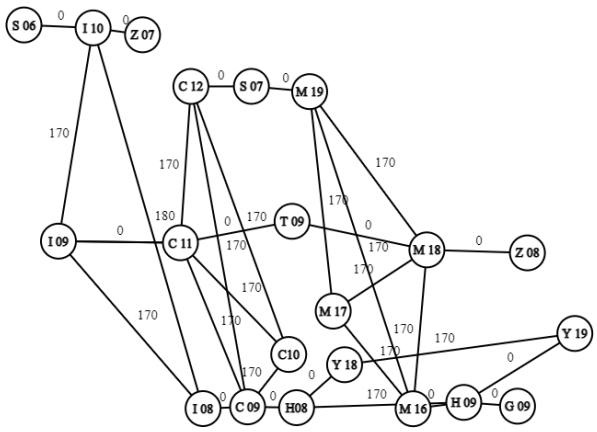
Gambar 6. Graf lengkap dijkstra

Terakhir akan dilakukan *brute force* untuk menentukan urutan simpul yang paling efektif. Hasil permutasi adalah sebagai berikut, pada gambar 5 dapat dilihat permutasi urutan simpul yang harus dilewati dan didapat bahwa lintasan terbaik dapat dicapai ketika kita mengunjungi simpul dengan urutan 7 -> 4 -> 1 dengan total biaya 18.

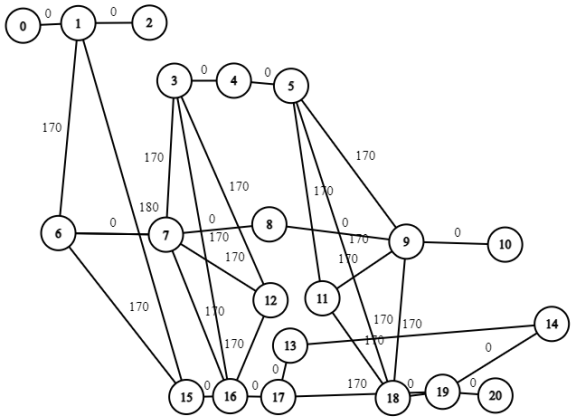
Studi kasus kedua menggunakan sebagian dari peta Tokyo Metro untuk dijadikan uji kasus.



Gambar 7. sumber[5]



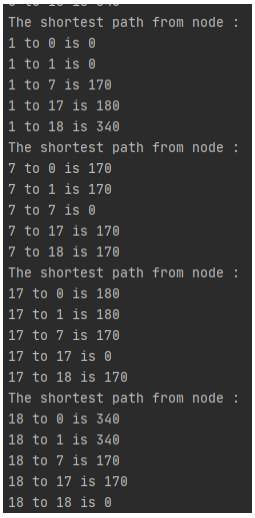
Gambar 8. Representasi graf stasiun



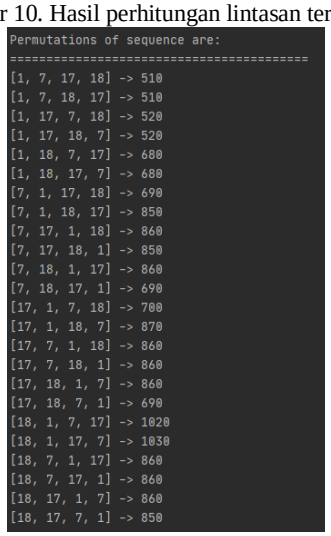
Gambar 9. Representasi graf stasiun

Gambar di atas adalah sebagian dari jaringan MRT Tokyo. Gambar 8 berisi representasi graf dari beberapa stasiun yang terdapat dalam gambar 7 dengan nama simpul yang sama dengan kode stasiun. Untuk kemudahan perhitungan, setiap simpul telah diubah namanya menjadi angka dari 0 sampai 20 inklusif seperti gambar 9. Pada beberapa sisi terlihat bobot dengan besar 0. Hal ini menandakan sang turis hanya perlu berjalan kaki di dalam stasiun yang sama untuk berpindah jalur utama. Selain itu telah ditambahkan beberapa sisi tambahan untuk mengatasi kasus pemberian harga perjalanan yang non linier seperti telah disebut pada bab sebelumnya.

Untuk kasus kali ini, misalkan turis mulai berangkat dari stasiun S06(simpul 0) dan ingin mengunjungi 4 buah stasiun yaitu C11 (simpul 7), I10 (simpul 1), H08 (simpul 17), dan M16 (simpul 18). Berikut adalah *screenshot* dari hasil perhitungan algoritma.



Gambar 10. Hasil perhitungan lintasan terpendek



Gambar 11. Hasil permutasi

Terlihat pada gambar beberapa kemungkinan urutan stasiun yang dikunjungi. Dalam kasus ini terdapat lebih dari satu lintasan yang optimal yaitu [1,7,17,18] dan [1,7,18,17]. Alasan penulis tidak menggunakan keseluruhan peta Tokyo Metro adalah karena data yang perlu dituliskan secara terlalu besar untuk dilakukan dengan tangan. Walau demikian algoritma yang tersedia pada *github* penulis dalam lampiran secara teori masih dapat melakukan perhitungan bahkan jika banyak simpul dan sisi lebih besar dari Tokyo Metro.

V. KESIMPULAN

Pencarian shortest path pada Tokyo Metro dengan beberapa buah simpul wajib kunjung dengan gabungan algoritma dijkstra dan permutasi *brute force* hanya dapat dilakukan untuk nilai m yang kecil. Nilai m disini melambangkan jumlah simpul wajib kunjung. Banyaknya jumlah simpul/stasiun dan sisi/rute tidak terlalu bermasalah karena *bottle neck* algoritma ada di permutasi *brute force* yang lamanya ditentukan oleh banyaknya nilai m. Alternatif permutasi *brute force* adalah menggunakan *bitmasking* yang memiliki kompleksitas yang lebih rendah namun walau demikian, permasalahan masih tidak dapat diselesaikan dalam waktu polinomial dengan pendekatan ini.

VI. LAMPIRAN

Peta jaringan subway Tokyo Metro dan Toei dapat diunduh dan dilihat pada laman (https://www.tokyometro.jp/en/subwaymap/). Implementasi kode penulis dalam Java dapat dilihat pada laman https://github.com/Aphostrophy/TokyoMetroPathFinder . Website yang digunakan untuk membantu menggambar graf adalah https://csacademy.com/app/graph_editor/.

VII. UCAPAN TERIMAKASIH

Segala puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan makalah ini dengan judul : "Teknik Optimasi Pencarian Lintasan Terbaik Tokyo Metro dengan Beberapa Simpul Wajib Kunjung Menggunakan Gabungan Algoritma Dijkstra dan Brute Force" yang jauh dari sempurna ini.Penulis juga turut berterima kasih kepada Ibu Fariska Zakhralativa Ruskanda, M.T sebagai dosen yang membimbing saya dalam kelas IF2120 Matematika Diskrit, Pak Dr. Ir. Rinaldi Munir, M.T, Ibu Harlili, M.Sc, dan Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc sebagai dosen pengampu dalam mata kuliah IF2120 Matematika Diskrit.

VIII. REFERENSI

[1] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.
[2] Munir, Rinaldi. Matematika Diskrit, Bandung: Informatika, 2010, edisi ketiga.
[3] West, Douglas B.. *Introduction to Graph Theory*. 2 : Prentice Hall, 2000B
[4] Tokyo Metro Co., Ltd., (Online), https://www.tokyometro.jp/en, diakses tanggal 6 Desember 2020.
[5] Euler, Leonhard, 'Solutio problematis ad geometriam situs pertinentis' (1741), Eneström 53, MAA Euler Archive.
[6] Munir, Rinaldi. *Graf.*, (online) https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf ,diakses tanggal 6 Desember 2020.
[7] Robin J. Wilson. 17 Dec 2013, History of Graph Theory from: Handbook of Graph Theory CRC Press Accessed on: 08 Dec 2020 https://www.routledgehandbooks.com/doi/10.1201/b16132-3

IX. 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, 3 Desember 2020

Ttd (scan atau foto ttd)

Jesson Gosal Yo
13519079