PENCARIAN RUTE PERJALANAN TERPENDEK DENGAN ALGORITMA DIJKSTRA

Article · June 2022

4 authors, including:

Sherly Santiadi
Universitas Kristen Maranatha
4 PUBLICATIONS **1** CITATION

SEE PROFILE

LAPORAN TUGAS BESAR

IN244 STRATEGI ALGORITMIK

PENCARIAN RUTE PERJALANAN TERPENDEK DENGAN ALGORITMA DIJKSTRA

Diajukan untuk memenuhi persyaratan kelulusan mata kuliah IN244 Strategi Algoritmik Tahun Akademik 2022/2023

Dosen Pembimbing:

Dr. Ir. Mewati Ayub, MT

Disusun Oleh:

Sherly Santiadi	NIM: 2072025
Arya Tri Putra Majiah	NIM: 2072023
Nisa Deviani Agustin Ruis	NIM: 2072051
Stefanus	NIM: 2072013

UNIVERSITAS KRISTEN MARANATHA

BANDUNG

2022

LEMBAR PENGESAHAN

Laporan Tugas Besar IN244

Program Studi Teknik Informatika - Universitas Kristen Maranatha

Judul Penelitian Tugas Besar

PENCARIAN RUTE PERJALANAN TERPENDEK DENGAN ALGORITMA DIJKSTRA

Nama Mahasiswa : Sherly Santiadi

NIM : 2072025

Nama Mahasiswa : Arya Tri Putra Majiah

NIM : 2072023

Nama Mahasiswa : Nisa Deviani Agustin Ruis

NIM : 2072051

Nama Mahasiswa : Stefanus

NIM : 2072013

Telah diperiksa dan disetujui pada tanggal …….

Wakil Dekan Akademik

Fakultas Teknologi Informasi, **Dosen Pembimbing,**

Djoni Setiawan K., S.T., M.T. **Dr. Ir. Mewati Ayub, MT**

NIK. 710003 **NIK. 720140**

KATA PENGANTAR

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa karena dengan rahmat dan juga karunia-Nya kami dapat menyelesaikan tugas besar berjudul Pencarian Rute Perjalanan Terpendek dengan Algoritma Dijkstra dengan baik dan juga tepat waktu.

Adapun tujuan dari penulisan laporan tugas besar ini adalah untuk memenuhi tugas akhir pada Mata Kuliah Strategi Algoritmik. Selain daripada itu, makalah ini juga bertujuan untuk menambah wawasan tentang pemanfaatan algoritma dijkstra dalam pencarian rute terpendek.

Kami juga berterima kasih kepada Ibu Dr. Ir. Mewati Ayub, MT selaku Dosen Pembimbing pada Mata Kuliah Strategi Algoritmik Universitas Kristen Maranatha yang telah memberikan dukungan moral dan moril kepada kami tanpa henti. Kami juga berterima kasih kepada semua pihak yang telah memberi dukungan maupun pengetahuannya sehingga kami dapat menyelesaikan laporan tugas besar ini.

Kami menyadari laporan yang kami tulis ini masih jauh dari kata sempurna. Oleh sebab itu, kami berharap mendapatkan kritik dan saran yang membangun demi kesempurnaan laporan tugas besar ini.

Bandung, 22 Mei 2022

Peneliti

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

PENCARIAN RUTE PERJALANAN TERPENDEK DENGAN ALGORITMA DIJKSTRA

Sherly Santiadi[1], Arya Tri Putra Majiah[2], Nisa Deviani Agustin Ruis[3], Stefanus[4]
Informatics/Computer Science
Faculty of Information Technology
Maranatha Christian University
Bandung, Indonesia
2072025@maranatha.ac.id[1], 2072023@maranatha.ac.id[2], 2072051@maranatha.ac.id[3], 2072013@maranatha.ac.id[4]

***Abstrak*—Pada penelitian ini dilakukan studi untuk mencari rute terpendek. Metode untuk mencari rute terpendek ini terbilang cukup beragam, salah satu metode yang dapat diterapkan adalah dengan menggunakan algoritma Dijkstra. Pada penelitian ini, algoritma Dijkstra akan digunakan dalam menghitung jarak terdekat atau biasa disebut dengan bobot terkecil dari titik awal menuju titik tujuan. Data yang digunakan berasal dari Open Street Map. Selain itu kode program yang dibangun menggunakan Bahasa Pemrograman Python dan Java Script, kemudian Framework Leaflet dan juga Web Server Uvicorn serta penggunaan peta digital yaitu Open Street Map.**

Adapun batasan yang digunakan dalam melakukan uji kasus yaitu kasus dengan jalanan dua arah (*undirected graph*). Uji kasus yang digunakan untuk memeriksa kebenaran pencarian rute perjalanan terpendek dari algoritma Dijkstra akan divalidasi menggunakan hasil rute dari Google Maps dan Waze.

Hasil dari penelitian ini menunjukan bahwa program sudah sukses dalam uji kasus. Pada beberapa kali uji kasus, program sudah bisa mencari rute perjalanan terpendek sesuai dengan Google Maps maupun Waze.

Kata Kunci—Algoritma Dijkstra, Google Maps, Waze, Pencarian Rute Terpendek

I. PENDAHULUAN

Peta merupakan sebuah sketsa yang berisikan informasi mengenai tata letak dari berbagai lokasi di suatu daerah tertentu. Keberadaan peta sudah ada sejak ribuan tahun lalu. Pada awalnya, peta merupakan sebuah sketsa atau gambar yang dibuat menggunakan tanah, serta pemanfaatan kerang yang digunakan untuk menandai pulau atau juga *node* pada peta tersebut. Seiring berkembangnya waktu, peta pun mengalami digitalisasi yaitu sebuah peta yang dapat diakses secara daring.

Berbagai situs penyedia layanan pemetaan web sudah biasa digunakan oleh masyarakat Indonesia dalam mencari rute, diantaranya adalah Google Maps dan Waze. Google

Maps merupakan sebuah layanan yang dikembangkan oleh Google. Layanan ini dapat memberikan citra dari kondisi lalu lintas secara *real time*, sehingga layanan pemetaan ini dapat memberikan kondisi terkini. Begitu pula dengan situs penyedia layanan Waze yang dapat memberikan citra dari kondisi lalu lintas secara *real time*. Walaupun ada sedikit perbedaan dari sisi *User Interface* yang disajikan oleh Google Maps dengan Waze. Waze sendiri pada awalnya merupakan sebuah FreeMap Israel.

Baik Google Maps maupun Waze sudah mencakup peta sebagian besar wilayah di dunia, tidak terkecuali Indonesia. Hanya saja ada beberapa wilayah terpencil yang masih belum bisa ditemukan melalui Google Maps dan Waze.

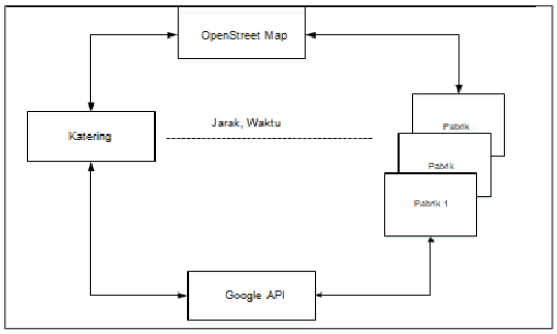
Selain itu pula untuk data pada penelitian kali ini digunakan sebuah *open source maps* yang berasal dari OpenStreetMap. Pada situs Open Street Map sudah dilengkapi berbagai fitur selayaknya pada Google Maps maupun Waze, walaupun untuk titik-titik node/lokasi memang tidak selengkap apabila menggunakan Google API dan juga tampilan dari Open Street Map ini terbilang jauh lebih sederhana daripada Google API. Adapun kelebihan sekaligus yang menjadi fondasi penggunaan Open Street Map adalah karakteristiknya yang *open source* dan gratis membuat peneliti menggunakan Open Street Map.

Problematika dalam pembuatan program sederhana adalah bagaimana cara mengimplementasikan sebuah algoritma dalam kasus ini adalah algoritma Dijkstra sehingga dapat menentukan rute terpendek dalam peta yang tersedia yaitu dengan menggunakan Open Street Map.

II. RISET TERKAIT

A. Algoritma Dijkstra untuk Penentuan Jarak Tempuh Terpendek Pengantaran Katering

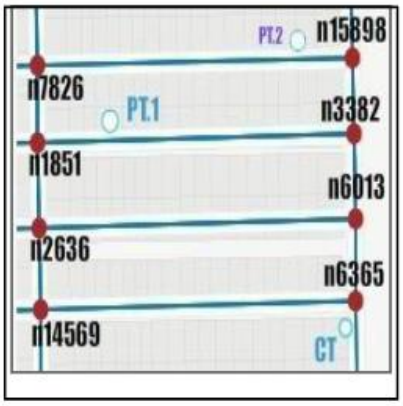
Pada penelitian yang dilakukan oleh Putri, Theta Dinnarwaty (2020) dalam artikel berjudul "Algoritma Dijkstra untuk Penentuan Jarak Tempuh Terpendek Pengantaran Katering" memanfaatkan algoritma dijkstra dalam pencarian jalur terpendek pada graf serta dengan pemanfaatan persamaan Haversine dalam pengukuran jarak dari node awal menuju 5 node tujuan yang merupakan lokasi dari pabrik. Tujuan pemanfaatan algoritma Dijkstra pada program yang dikerjakan oleh Putri adalah untuk merancang sebuah sistem paling optimal yaitu dalam satu kali pengantaran perusahaan katering dapat mengunjungi banyak pabrik dan tentunya dengan *cost* paling minimum.



GAMBAR 1 QUICK DESIGN PROSES KERJA SISTEM YANG DIBANGUN

Putri menggunakan *quick design* dalam membuat alur dari proses kerja sistem yang akan dibangun. Pada gambar ini yang merupakan batasan adalah jarak serta waktu,

dan juga *tools* yang digunakan adalah OpenStreetMap dan Google API.



GAMBAR 2 PETA

Gambar 2 merupakan tampilan pada peta yang terdapat node-node seperti N6365, N14569, N2636, N1851, N6013, N3382, N15898 yang merupakan persimpangan, sedangkan untuk PT.1, PT.2 merupakan lokasi tujuan dan CT merupakan lokasi katering tersebut.

TABEL 1 BOBOT KEMACETAN

Kondisi Jalan	Bobot
Normal	1
Ramai Lancar	0.8
Macet	0.65
Macet Parah	0.4

Kemudian pada Tabel 1 merupakan kondisi jalan yang nantinya akan diberi bobot kemacetan. Apabila kondisi jalan lancar tidak ada kemacetan terjadi maka akan diberikan bobot 1 atau 100% selanjutnya apabila jalan terbilang cukup ramai namun masih dalam batasan toleran maka akan diberikan bobot 0.8 atau 80% yang artinya kecepatan rata-rata dari kendaraannya 80% dari kecepatan rata-rata kendaraan pada saat kondisi lancar, dan seterusnya.

TABEL 2 HASIL PENGUJIAN

Pengujian		Koordinat Katering		Koordinat Pabrik			Jarak Tempuh (km)	Waktu Tempuh (menit)	Hasil Urutan ke-
		Latitude	Longitude	Nama Pabrik	Latitude	Longitude			
1	Tistas Catering (start)	-6.3019297	107.16500840000003	PT Supernova Flexible Packaging Cikarang Plant	-6.3098703,107	107.16020159999994	1.58	3.25	2
		-6.3019297	107.16020159999994	PT. Adaywinsa Electrical and Power	-6.3072119	107.16299679999997	1.64	3.17	1
		-6.3019297	107.16020159999994	PT. Bina Niaga MultiUsaha	-6.3097986	107.15524069999992	1.91	3.75	4
		-6.3019297	107.16020159999994	PT. Dinamika Makmur Sentosa	-6.31251399999	107.15442610000002	1.74	3.60	3
		-6.3019297	107.16020159999994	PT.Asahi Indonesia	-6.3161217	107.17980080000007	3.17	6.70	5

Tabel 2 merupakan hasil dari pengujian program. Terdapat 5 buah lokasi pabrik yang dijadikan node tujuan dengan titik awal yang sama yaitu dari Tistas Catering dengan latitude -6.3019297 dan longitude 107.16500850000003 pada peta yang ada di OpenStreetMap dan juga nama pabrik tujuan beserta dengan masing-masing latitude dan longitude. Kemudian pada tabel ini terdapat jarak tempuh dalam satuan km yang merupakan hasil perhitungan dari koordinat posisi dari Tistas Catering ke masing-masing pabrik kemudian waktu tempuh merupakan hasil dari perhitungan persamaan Haversine yang tadi sudah didefinisikan dengan masing-masing bobot. Kemudian waktu tempuh tersebut yang akan dijadikan bahan olah pada matriks dijkstra dalam menentukan hasil *ranking* sehingga menemukan solusi paling optimum dalam penentuan *cost* terendah. Sehingga didapatkan solusi yaitu dari Tistas Catering → PT. Adaywinsa Electrical and Power → PT. Supernova Flexible Packaging Cikarang Plant → PT. Dinamika → PT. Bina Niaga Multiusaha → Pt. Asahi Indonesia.

B. Penerapan Algoritma Dijkstra Untuk Menemukan Lintasan Terpendek pada Pengiriman Barang PT Kharisma Suma Jaya Sakti

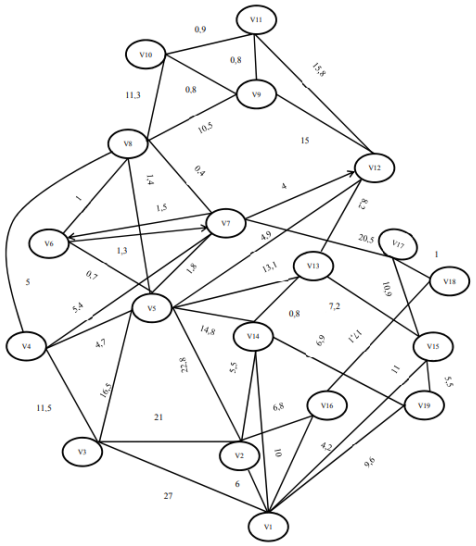
Penelitian dengan judul "Penerapan Algoritma Dijkstra Untuk Menemukan Lintasan Terpendek pada Pengiriman Barang PT Kharisma Suma Jaya Sakti" ini dilakukan oleh Retyana Fitari Ekasari (2017). Pengambilan data dalam penelitian ini dilakukan dengan pengukuran jarak tempuh menggunakan Odometer, kemudian penyusunan rute dari data yang diperoleh di PT Kharisma Suma Jaya Sakti dan mencari lintasan terpendek menggunakan Algoritma Dijkstra.

TABEL 3 JARAK ANTAR TITIK

Titik terhubung	Jarak (km)	Titik terhubung	Jarak (km)
v1 – v2	6	v6 – v8	1
v1 – v3	27	v6 – v4	4
v1 – v14	10	v7 – v8	0,4
v1 – v19	9, 6	v7 – v12	4
v1 – v15	11	v8 – v9	10,5
v2 – v3	21	v8 – v10	11,3
v3 – v4	11,5	v9 – v10	0,8
v3 – v5	16,5	v9 – v11	0,8
v4 – v5	4,7	v9 – v12	15
v4 – v2	5,5	v10 – v11	0,9
v4 – v7	5,4	v11 – v12	15,8
v4 – v8	5	v12 – v13	8,2
v5 – v6	0,7	v13 – v14	0,8
v5 – v7	1, 8	v13 – v15	7,2
v5 – v8	1, 4	v15 – v18	10,9
v5 – v12	4,9	v16 – v2	6,8
v5 – v13	13,1	v17 – v18	1
v5 – v15	14,8	v18 – v16	17,1
v6 – v7	1,3	v19 – v14	6, 9
v7 – v6	1,5	v19 – v15	5,5

Setelah mendapatkan data dari PT Kharisma Suma Jaya Sakti, peneliti melakukan observasi dengan menggunakan Odometer agar dapat mengetahui jarak antar titik yang menghubungkan bengkel AHASS dengan bengkel yang lainnya yang berada di Kabupaten Kediri.

Masing - masing titik yang berada dalam tabel tersebut harus dihubungkan dengan sebuah sisi. Sisi ini melambangkan panjang sebuah jarak pada graph, seperti pada gambar di bawah ini.



GAMBAR 3 GRAPH BENGKEL KABUPATEN KEDIRI

Peneliti selanjutnya melakukan perhitungan untuk rute terpendek pada pengiriman barang setelah mendapatkan data pengiriman barang. Algoritma Dijkstra secara otomatis akan menemukan lintasan terpendek dari setiap rute yang ada.

III. STUDI LITERATUR

Berikut merupakan hasil studi literatur yang dilakukan:

A. Struktur Data Graf

I. Pendahuluan

Suatu graf (kumpulan relasi antar objek) dapat dinyatakan struktur data abstrak dalam bidang ilmu Matematika.

Adapun graf dan struktur memiliki dua fungsi berbeda. Graf berfungsi untuk menampilkan visualisasi data daripada suatu graf, sedangkan struktur data menjelaskan penyimpanan informasi sehingga dapat diakses dengan baik.

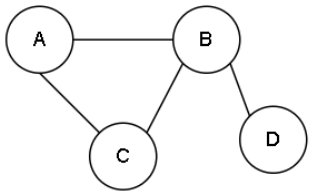
II. Implementasi

Dalam prakteknya, ilmu struktur data graf dapat diimplementasikan dalam bentuk daftar ketetanggaan (*adjacency list*) serta matriks ketetanggaan (*adjacency matrix*) yang baris-barisnya melambangkan simpul awal, sedangkan kolom-kolomnya melambangkan simpul tujuan.

1. Daftar Ketetanggaan (*Adjacency List)*

Daftar ini dapat dinotasikan dengan menelaah tiap titik dan keterhubungannya dengan tiap tetangganya.

Mari lihat contoh berikut :



GAMBAR 4 ADJACENCY LIST

Dapat dinotasikan secara berikut : {a,b}, {a,c}, {b,c}, {b,d}. Kemudian dapat disajikan dalam bentuk tabel di bawah ini.

TABEL 4 DAFTAR KETETANGGAAN

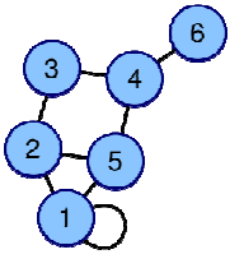
Titik	Ketetanggaan	Daftar tetangga
a	Tetangga dengan	b,c
b	Tetangga dengan	c,d
c	Tetangga dengan	a,b
d	Tetangga dengan	b

Kekurangan : nilai antar sisi tidak direpresentasikan

2. Matriks Ketetanggaan (*Adjacency Matrix)*

Matriks (daftar 2D) pada setiap kolom dan barisnya secara seranai merepresentasikan simpul-simpul yang ada. Setiap baris entri menyatakan hubungan antar simpul pula.

Mari lihat contoh berikut :



GAMBAR 5 GRAF TAK BERARAH BERLABEL

Keterangan :

Tak berarah -> matriks simetris
Berarah -> matriks tidak simetris

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

GAMBAR 6 ADJACENCY MATRIX

Baris 1 (titik 1) -> 2,5
Baris 2 (titik 2) -> 1,3,5
Dan seterusnya.

Adapun kelebihan dan kekurangan matriks ketetanggan :

i. Kelebihan
Elemen matriks diakses langsung dari indeksnya dan hubungan ketetanggaan kedua simpul dapat ditentukan langsung.

ii. Kekurangan
Jika banyak elemen bernilai nol (0), maka matriks menjadi boros dan tidak efisien karna penyimpanan elemen-elemen tersebut diproses namun nyatanya tidak perlu.

B. Algoritma Dijkstra

I. Abstraksi

Algoritma Dijkstra (yang dikemukakan oleh ilmuwan komputer bernama E.W Dijkstra) adalah langkah-langkah bertahap untuk mencari jarak terpendek suatu graf dari titik (*vertex*) awal ke titik tujuan manapun.

II. Pengenalan

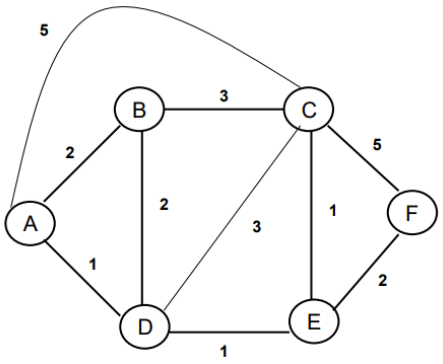
Secara ringkas, Algoritma Dijkstra dapat diteliti dengan metode analisis menggunakan tabel yang melampirkan kolom-kolom dengan bertajuk (*header*) berikut :

1. Titik keseluruhan (*node*) yang akan ditempuh
2. Analisis pencarian jarak tempuh minimum antara yang sudah ditemukan dengan yang sedang ditelaah
3. Nilai tempuh keseluruhan
4. Titik asal (*visited*)

III. Contoh Kasus

Adapun contoh kasus berikut sebagai penjelasan algoritmanya :

Diketahui : Ada suatu topologi jaringan memiliki 6 titik (*vertex*) dengan nilai jarak ketetanggaannya satu sama lain seperti demikian :



GAMBAR 7 CONTOH GRAF

Dicari : Jarak/biaya terpendek yang dapat ditempuh dari titik A menuju titik F

Dengan metode analisis tabel, dapat dilakukan langkah-langkah bertahap seperti berikut :

Keterangan :

S : titik sumber (sesuai kasus : A)

N : himpunan titik yang sudah teranalisis

b(i,j) : jarak/biaya titik ketetanggaan

∞ : titik yang hendak dituju tidak saling bertetanggaan

J(t) : jarak titik terpendek dari titik sumber ke titik yang hendak dituju yang teranalisis. (Menunjukkan biaya tempuh titik sumber ke titik manapun saat algoritma tuntas dijalankan)

P(t) : titik pendahulu sebelum titik - titik yang hendak dituju dan tersedia pada graf

Berikut tahapannya :

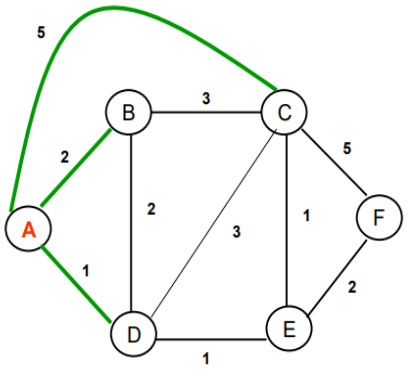
1. Inisialisasi

   N = {A} *baru titik awal*

   Menghitung J(t) untuk titik-titik non-himpunan N pada graf (B,C,D,E,F)

TABEL 5 ANALISIS GRAF

t	J(t)=b(s,t)= b(A,t)	P(t)
B	2	A
C	5	A
D	1	A
E	∞	buntu
F	∞	buntu



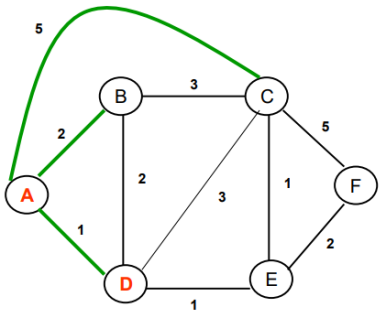
2. Analisis titik terdekat selanjutnya (1)

   Pilih J(t) minimum (jarak sebelumnya ditambah yang

terbaru), jika ada yang berimbang, pilihlah secara acak dari yang sama tersebut.

Pada kasus ini, J(D) = 1 adalah jarak minimum, jadi kita perlu memasukkan titik terpilih terbaru (D) ke himpunan N.

Maka N = {A,D}



3. Memperbaharui biaya/jarak terpendek lintasan (1)

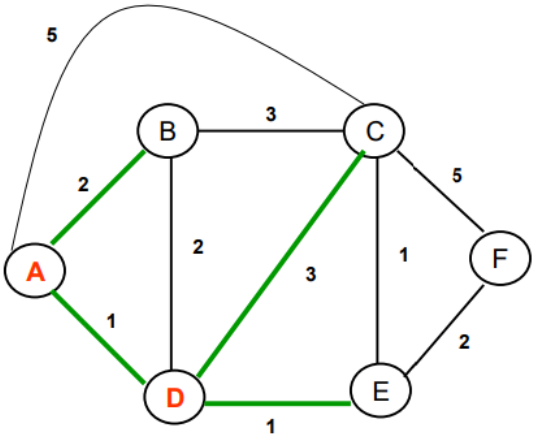
Caranya : J(t) = min[J(t), J(u) + b(u,t)] untuk semua t yang belum masuk himpunan N -> B,C,E,F

Pilih J(t) yang terkecil

TABEL 6 PERHITUNGAN MINIMUM COST

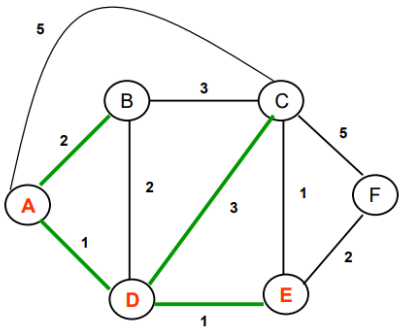
t	J(t) = min[J(t), J(D) +b(D,t)]	J(t)	P(t)
B	J(B) = min[J(B), 1+b(D,B)]= min[2, 1+2]	2	A
C	J(C) = min[J(C), 1+b(D,C)]= min[5, 1+3]	4 (b)	D
E	J(E) = min[J(E), 1+b(D,E)]= min[∞, 1+2]	2 (b)	D
F	J(F) = min[J(F), 1+b(D,F)]= min[∞, 1+∞]	∞	buntu

Keterangan : b -> baru; u -> updated member of N (anggota terbaru N)



4. Analisis titik terdekat selanjutnya (2)

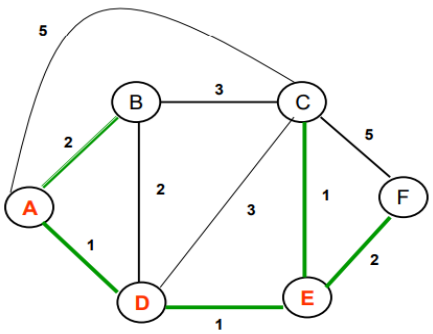
Mengikuti langkah nomor 2, pilih J(E) dan J(B) sama-sama berjarak 2 (ditinjau dari jarak sebelumnya dari titik A yg minimum ke B dengan peninjauan jarak dari titik D ditambah titik sebelumnya dari A). Pilih acak misalkan titik E saja, maka : N = {A,D,E}



5. Memperbaharui biaya/jarak terpendek lintasan (2)

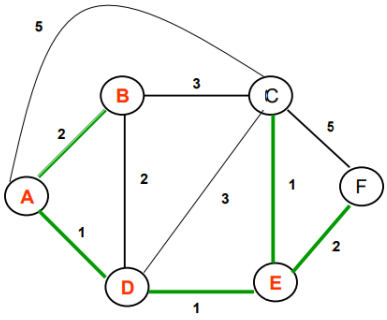
   Mengikuti instruksi nomor 3, maka :

t	J(t) = min[J(t), J(E) +b(E,t)]	J(t)	P(t)
B	J(B) = min[J(B), 2+b(E,B)]= min[2, 2 +∞]	2	A
C	J(C) = min[J(C), 2+b(E,C)]= min[4, 2 +1]	3 (b)	E
F	J(F) = min[J(F), 2 + b(E,F)] = min[∞, 2+2]	4 (b)	E



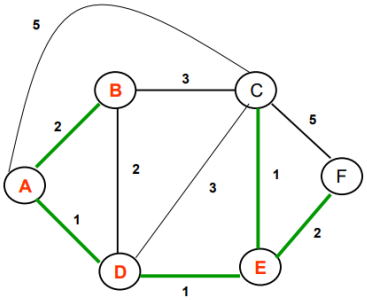
6. Analisis titik terdekat selanjutnya (3)

   J(B) dari titik ter analysis (D) yang merupakan titik sebelumnya berjarak minimum yaitu 2. Maka N = {A,D,E,B}



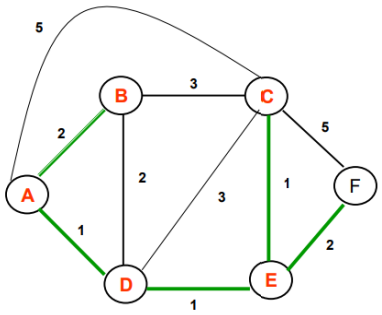
7. Memperbaharui biaya/jarak terpendek lintasan (3)

t	J(t) = min[J(t), J(B) +b(B,t)]	J(t)	P(t)
C	J(C) = min[J(C), 2 + b(B,C)] = min[3, 2+3]	3	E
F	J(F) = min[J(F), 2 + b(B,F)] = min[4, 2+∞]	4	E



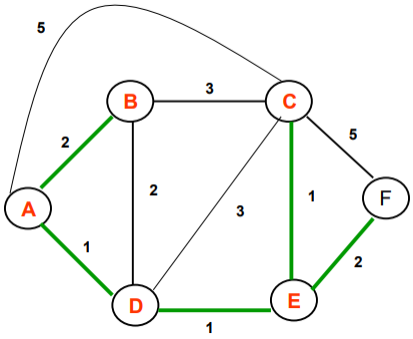
8. Analisis titik terdekat selanjutnya (4)

   J(C) menunjukkan biaya terkecil, maka N = {A,D,E,B,C}



9. Memperbaharui biaya/jarak terpendek lintasan (4)

t	J(t) = min[J(t), J(C) +b(C,t)]	J(t)	P(t)
F	J(F) = min[J(F), 3 + b(C,F)] = min[4, 3+5]	4	E



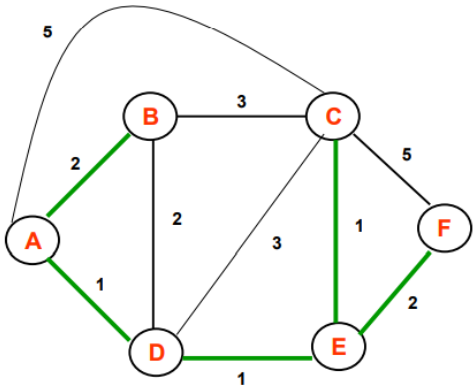
10. Analisis titik terdekat selanjutnya (5 -> FINAL)

    FINAL : sisa titik terakhir (F) masukkan ke himpunan N : {A,D,E,B,C,F}. Anggota himpunan N sudah meliputi semua titik dari graf. Maka dari itu, algoritma mencapai tahap terminasi.

11. Rekapitulasi lintasan dengan jarak terpendek dari titik sumber (A) ke titik-titik lainnya pada graf

TABEL 7 REKAPITULASI LINTASAN TERPENDEK

t	J(t)	Lintasan
F	**4**	**A-D-E-F**
D	**1**	**A-D**
C	**3**	**A-D-E-C**
E	**2**	**A-D-E**
B	**2**	**A-B**



(Lintasan Minimum)

C. Open Street Map

I. Abstraksi

OpenStreetMap (OSM) adalah sebuah proyek berbasis web untuk membuat peta seluruh dunia yang gratis dan

terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survey menggunakan GPS, mendigitasi citra satelit, dan mengumpulkan serta membebaskan data geografis yang tersedia di publik.

II. Pengenalan

Open Data Commons Open Database License 1.0 memungkinkan kontributor OSM untuk memiliki, memodifikasi dan berbagi data peta. Ada banyak jenis peta digital yang tersedia di Internet, tetapi kebanyakan dari mereka memiliki keterbatasan hukum dan teknis. Hal ini memberi masyarakat, pemerintah, peneliti, dan banyak pihak kepentingan lainnya tidak dapat mengakses gratis ke data yang tersedia di peta. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara bebas dan terbuka untuk digunakan dan didistribusikan kembali.

Di banyak tempat, terutama di daerah terpencil dan terbelakang secara ekonomi, tidak ada insentif komersial bagi perusahaan pemetaan untuk mengembangkan data di daerah tersebut. OSM dapat menjadi jawaban di banyak tempat ini untuk pembangunan ekonomi, perencanaan kota, mitigasi risiko bencana, atau berbagai tujuan lainnya.

IV. IMPLEMENTASI DAN PENGUJIAN

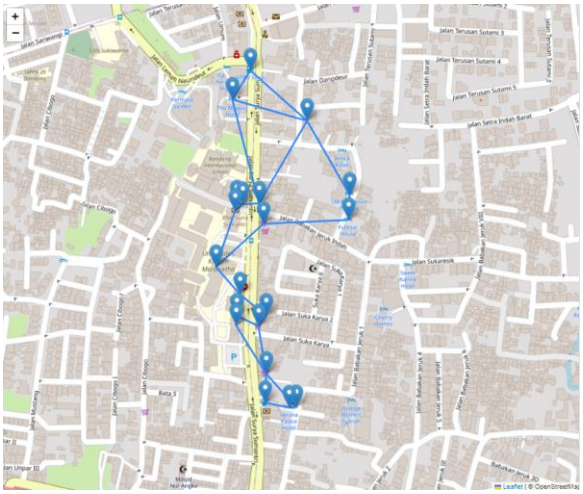
A. Deskripsi Persoalan

Dalam penelitian ini, topik permasalahan yang diambil adalah pencarian solusi lintasan terpendek dengan menggunakan algoritma Dijkstra. Untuk geolokasi dalam implementasi pencarian jalur yang digunakan dalam penelitian ini adalah geolokasi dari lingkungan di sekitar Universitas Kristen Maranatha. Adapun node yang digunakan ada 20 buah node (tidak termasuk Universitas Kristen Maranatha) yaitu: Terazza, Permata Bank, Wibisana, Indomaret, Inti Laut, Circle K, Indomaret, BNP, BCA, Take Ichi Japanese Café, Tujuh Sebelas, The Majesty Hotel, Martabak Bolu Golden Bell, SPBU, SM Residence Pasteur, FullMar House, SPBU, Verona Palace, Sugar Rush, dan BRI.

B. Deskripsi Solusi

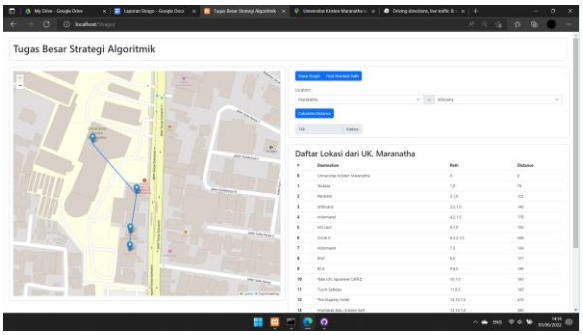
Oleh karena itu, pada penelitian ini kami menggunakan algoritma Dijkstra dalam mencari lintasan terpendek. Algoritma Dijkstra ini akan mencari lintasan terpendek dari suatu *vertex* dengan cara membangun sebuah pohon yang memiliki nilai paling minimum. Adapun dalam pembangunan graf maka dibutuhkan sebuah matriks untuk mengetahui node mana saja yang saling bertetangga.

C. Deskripsi Hasil Pengujian (Tampilan *Input/Output*)



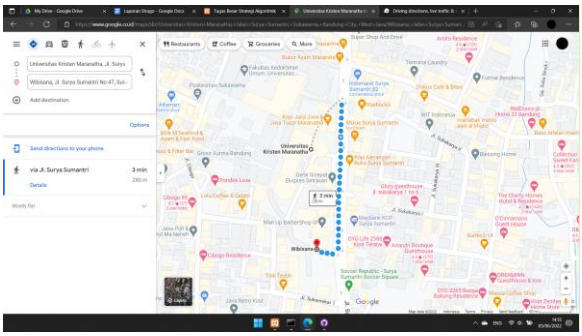
GAMBAR 8 TAMPILAN KESELURUHAN GRAF

Sebelum pembuatan algoritma Dijkstra, hal pertama yang kami lakukan adalah menentukan *vertex* apa saja yang akan digunakan serta menghitung jarak dari masing-masing *vertex*. Sehingga peneliti dapat membuat matriks yang nantinya akan diimpor ke dalam bentuk .csv.



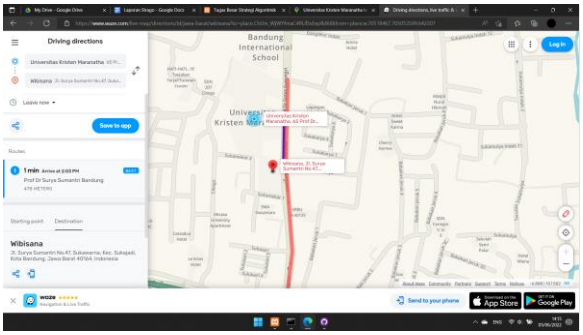
GAMBAR 9 TAMPILAN PROGRAM

Pada gambar 9 merupakan hasil eksekusi dari uji coba pada penelitian ini. Dilakukan pengujian secara berurutan dari *initial vertex* dari indeks ke-0 sampai dengan ke-20. Pada gambar 9 terlihat *initial vertex* dengan indeks ke-0 yaitu Universitas Kristen Maranatha menuju *vertex* dengan indeks ke-3 yaitu Wibisana. Untuk penempatan *vertex* digunakan *longitude* dan *latitude* dari OpenStreet Map yang kemudian dilakukan penarikan garis secara langsung antara dua *vertex* yang bersangkutan (tidak mengikuti jalur pada peta). Kemudian menggunakan algoritma Dijkstra dalam menentukan jalur terpendek yang kemudian dicatat vertex mana saja yang akan dilalui dalam penentuan jarak terpendek. Hasilnya adalah dengan menggunakan algoritma Dijkstra dari simpul asal yaitu Universitas Kristen Maranatha menuju simpul tujuan yaitu Wibisana diperlukan jarak minimum yaitu 146 meter dengan jalur Universitas Kristen Maranatha → Terazza → Permata Bank → Wibisana.



GAMBAR 10 TAMPILAN GOOGLE MAPS

Kemudian pada gambar X dilakukan uji validitas, tentunya sama seperti pada hasil eksekusi sebelumnya. Uji validitas ini dilakukan secara berurutan dari *initial vertex* dari indeks ke-0 sampai dengan ke-20. Kemudian pada gambar 10 terlihat jalur yang ditempuh sama seperti pada hasil eksekusi pada program yang dilakukan oleh peneliti. Hanya saja jika peta diperbesar ada beberapa *vertex* yang tampaknya sudah tidak ada lagi di dalam peta digital milik Google.



GAMBAR 11 TAMPILAN WAZE

Kemudian pada gambar 10 dilakukan uji validitas yang kedua menggunakan Waze, tentunya sama seperti pada hasil eksekusi sebelumnya. Uji validitas ini dilakukan secara berurutan dari *initial vertex* dari indeks ke-0 sampai dengan ke-20. Kemudian pada gambar 10 terlihat jalur yang ditempuh sama seperti pada hasil eksekusi pada program yang dilakukan oleh peneliti. Hanya saja *downside* ketika menggunakan Waze yaitu *vertex-vertex* yang ada pada peta digital tampak kurang lengkap serta tidak adanya simbol untuk fasilitas-fasilitas layanan yang berbeda misalkan untuk *vertex* yang memberikan fasilitas layanan seperti rumah makan lebih identik dengan simbol sendok dan garpu, membuat beberapa lintasan dalam peta digital tampak begitu rancu.

V. KESIMPULAN DAN SARAN

Berikut adalah kesimpulan dari Tugas Akhir:

Dari hasil analisis beberapa hal yang sudah dilakukan yaitu pencarian rute terdekat dari simpul A menuju simpul X di lingkungan Universitas Kristen Maranatha. Kemudian dilakukan perhitungan penjumlahan dari setiap *vertex* sebelumnya yang dilewati dalam pencarian rute sesuai dengan algoritma Dijkstra. Selanjutnya memperhitungkan total jarak tempuh dari simpul A menuju simpul X sekaligus memberikan lintasan apa saja yang akan dilewati dalam rute dari simpul A ke simpul X. Ini dapat dipastikan merupakan rute dengan jarak tempuh paling optimal atau paling dekat total jaraknya karena telah diberlakukan uji validitas dengan 2 peta digital lainnya. Dengan penentuan rute menggunakan algoritma Dijkstra, diharapkan dapat membantu manusia dalam persoalan pencarian rute terpendek dengan *cost* paling minimum. Selain itu, pada penelitian kali ini peneliti telah berhasil membuat daftar yang memperhitungkan jarak dan lintasan apa saja dari Universitas Kristen Maranatha ke simpul tujuan kemudian dilakukan *sorting* dari lokasi terdekat sampai dengan terjauh dari Universitas Kristen Maranatha.

Berikut adalah saran dari Tugas Akhir:

- Pada penelitian ini algoritma Dijkstra yang telah dilakukan tidak menjamin dapat menuju simpul tujuan dengan waktu paling minimum atau tercepat dikarenakan perlunya sebuah pendekatan yang memperhitungkan bobot dalam faktor-faktor lainnya misalkan dengan memperhitungkan bobot kemacetan, bobot jalan tol, bobot kendaraan yang digunakan dan lain sebagainya.

- Penggunaan Open Street Map masih memiliki beberapa *downside* diantaranya peta digital yang tidak *up-to-date* sehingga tidak dapat menyajikan simpul-simpul secara akurat.

VI. TAUTAN

Adapun tautan yang dapat membantu dalam proses penggunaan program ini:

- Untuk penjelasan kode program:

Pencarian Rute Perjalanan Terpendek dengan Algoritma Dijkstra

REFERENSI

Javaid, Adeel (2013). Understanding Dijkstra Algorithm. *SSRN Electronic Journal*, 1, 14-26.

Putri, Theta Dinnarwaty, Winarno Sugeng, & Eka Safitri (2020). Algoritma Dijkstra untuk Penentuan Jarak Tempuh Terpendek Pengantaran Katering Pabrik. doi: 10.26760

Bambang, A., Sulistyono, M., Si, Jatmiko, M., & Pd. (2017). PENERAPAN ALGORITMA DIJKSTRA UNTUK MENEMUKAN LINTASAN TERPENDEK PADA PENGIRIMAN BARANG PT KHARISMA SUMA JAYA SAKTI THE IMPLEMENTATION USING DIJKSTRA ALGORITHM FOR FINDING SHORTEST PATH OF DELIVERY ITEM IN PT KHARISMA SUMA JAYA SAKTI Dibimbing oleh. http://simki.unpkediri.ac.id/mahasiswa/file_artikel/2017/12.1.01.05.0101.pdf

Anggara, F. D. & Munir, R., Institut Teknologi Bandung. (2008). Studi dan Implementasi

Struktur Data Graf. Retrieved June 1, 2022 from https://informatika.stei.itb.ac.id/

Tentang OpenStreetMap (OSM) – Perkumpulan OpenStreetMap Indonesia. (2012). Openstreetmap.or.id. https://openstreetmap.or.id/about/tentang-openstreetmap/

LAMPIRAN KODE SUMBER

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Tugas Besar Strategi Algoritmik</title>
    <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.8.0/dist/leaflet.css" integrity="sha512-
hoalWLoI8r4UszCkZ5kL8vayOGVae1oxXe/2A4AO6J9+580uKHDO3JdHb7NzwwzK5xr/Fs0W40ki
NHxM9vyTtQ==" crossorigin="" />
    <script src="https://unpkg.com/leaflet@1.8.0/dist/leaflet.js"
integrity="sha512-
BB3hKbKWOc9Ez/TAwyWxNXeoV9c1v6FIeYiBieIWkpLjauysF18NzgR1MBNBXf8/KABdlkX68nAh
lwcDFLGPCQ==" crossorigin=""></script>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"
crossorigin="anonymous">
    <style>
        #map {
            height: 80vh;
            width: 100%;
        }
    </style>
  </head>

<body>
      <div class="m-4">
        <div class="row">
          <div class="col">
            <div class="card">
              <div class="card-body">
                <h1>Tugas Besar Strategi Algoritmik</h1>
              </div>
            </div>
          </div>
        </div>
        <div class="row mt-4">
```

```
          <div class="col">
            <div class="card">
              <div class="card-body">
                <div id="map"></div>
              </div>
            </div>
          </div>
          <div class="col">
            <div class="row">
              <div class="card">
                <div class="card-body">
                  <div class="btn-group mb-4">
                    <button onclick="addAllMarkers()" href="#" class="btn
btn-primary">Show Graph</button>
                    <button href="#" class="btn btn-primary">Find Shortest
Path</button>
                  </div>
                  <br>
                  <label class="form-label">Location :</label>
                  <div class="input-group mb-3">
                    <select class="form-select" name="start"
id="start"></select>
                    <span class="input-group-text">-></span>
                    <select class="form-select" name="end"
id="end"></select>
                  </div>
                  <button onclick="addMarker()" href="#" class="btn btn-
primary">Calculate Distance</button>
                  <div class="input-group mt-4 w-25">
                    <input type="number" class="form-control" id="distance"
disabled>
                    <span class="input-group-text" id="basic-
addon2">Meters</span>
                  </div>
                </div>
              </div>
            </div>
            <div class="row mt-4">
              <div class="card">
                <div class="card-body">
                  <h2>Daftar Lokasi dari UK. Maranatha</h2>
                  <table class="table table-hover">
                    <thead>
                      <tr>
                        <th scope="col">#</th>
                        <th scope="col">Destination</th>
                        <th scope="col">Path</th>
                        <th scope="col">Distance</th>
```

```
                    </tr>
                  </thead>
                  <tbody></tbody>
                </table>
              </div>
            </div>
          </div>
        </div>
      </div>


    <script>
        // Inisialisasi map, simpan di div dengan id 'map'
        var map = L.map("map").setView([-6.88727, 107.58082], 18);

        // Menambah TileLayer(data map) ke map dari OpenStreetMap
        L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
            maxZoom: 19,
            attribution: "© OpenStreetMap",
        }).addTo(map);

        // Membuat group untuk menyimpan semua marker yang ada di map
        var markers = L.layerGroup().addTo(map);

        // Menentukan Jarak Maranatha ke lokasi lain (Tabel & Select)
        $.get('http://127.0.0.1:8000/maranatha', function(data) {

          $.each(data.data, function(index, value) {

            $('#start').append(new Option(value.end, value.node));
            $('#end').append(new Option(value.end, value.node));

            $('table tbody').append(
              '<tr>' +
                '<th scope="row">'+ value.node +'</th>' +
                '<td>' + value.end + '</td>' +
                '<td>' + value.path + '</td>' +
                '<td>' + value.distance + '</td>' +
              '</tr>'
            );
          });

        });
    </script>
    <script src="scripts/marker.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
```

```
pprn3073KE6tl6bjs2QrFaJGz5/SUsLqktiwsUTF55Jfv3qYSDhgCecCxMW52nD2"
crossorigin="anonymous"></script>
</body>

</html>
```

marker.js

```
// Node
// R - Restaurant
// M - Market
// B - Bank
// H - Hotel
// S - SPBU

// Koordinat di map
nodes = [
  [-6.88643, 107.58069],    // Universitas Kristen Maranatha
  [-6.886967, 107.581161],  // Terazza (R)
  [-6.887382, 107.581094],  // Permata Surya Sumantri (B)
  [-6.887572, 107.581089],  // Wibisana (R)
  [-6.887588, 107.581528],  // Indomaret (M)
  [-6.88738, 107.581681],   // Inti Laut (R)
  [-6.888509, 107.581671],  // Circle K (M)
  [-6.885594, 107.581622],  // Indomaret (M)
  [-6.885344, 107.581067],  // BNP (B)
  [-6.88521, 107.581217],   // BCA (B)
  [-6.885197, 107.581537],  // Take Ichi Japanese Cafe (R)
  [-6.885178, 107.581099],  // Tujuh Sebelas (M)
  [-6.883174, 107.581011],  // The Majesty Hotel (H)
  [-6.883589, 107.58248],   // Martabak Bolu Golden Bell (R)
  [-6.88261, 107.581364],   // SPBU (S)
  [-6.885022, 107.583322],  // SM Residence Pasteur (H)
  [-6.885516, 107.583295],  // FullMar House (H)
  [-6.889073, 107.581652],  // SPBU (S)
  [-6.889166, 107.582127],  // Verona Palace (H)
  [-6.889154, 107.582271],  // Sugar Rush (R)
  [-6.88924 , 107.58168]    // BRI (B)
];


function addAllMarkers() {
  deleteAllMarker();
  $.each(nodes, function(index, value) {
    L.marker(value).addTo(markers);
  });
  addAllPath();
}

function addAllPath() {

  // Meminta data get dari server.py
  $.get("http://127.0.0.1:8000/list", function (data) {
```

```
    $.each(data, function (i, value) {
      $.each(value, function (j, node) {
        if (node != 0) {
          L.polygon([nodes[i],nodes[j]]).addTo(markers);
        }
      });
    })
  });

}

function deleteAllMarker() {
  markers.clearLayers()
}

function addMarker() {
  deleteAllMarker();

  let start = $("#start").val();
  let end = $("#end").val();

  // Meminta data get dari server.py
  $.get(`http://127.0.0.1:8000/path?start=${start}&end=${end}`, function
(data) {
    $('#distance').val(data.distance);
    $.each(data.path, function (i, value) {
      L.marker(nodes[value]).addTo(markers);
    });
    addPath(data.path);
  });
}

function addPath(path) {
  $.each(path, function (i, value) {
    L.polygon([nodes[value],nodes[path[i+1]]]).addTo(markers);
  });
}

function calculateDistance(start, finish) {
  let finalPoint = L.latLng(nodes[start], nodes[start]);
  let startPoint = L.latLng(nodes[finish], nodes[finish]);
  dist = finalPoint.distanceTo(startPoint);
  return dist;
}
```

algoritma.py

```
INFINITY = float('inf')

def dijkstra(graf, awal):
    L = [None] * len(graf)
    for i in range(len(graf)):
        L[i] = [INFINITY, None]

    L[awal] = [0, None]
```

```
    S = []

    for k in range(len(graf)):
        mini = INFINITY
        for j in range(len(graf)):
            if L[j][0] < mini and j not in S:
                mini = L[j][0]
                minind = j
        # [minind, , mini]
        S.append(minind)

        for i in [l for l in range(len(graf)) if l not in S]:
            if graf[minind][i] != 0:
                if L[i][0] > L[minind][0] + graf[minind][i]:
                    L[i][0] = L[minind][0] + graf[minind][i]
                    L[i][1] = minind
                    # L[i] = L[minind] + graf[minind][i]

    return L

def lintasan(akhir, simpul, graf_hasil):
    array_simpul = [akhir]
    while simpul[1] != None:
        array_simpul.append(simpul[1])
        simpul = graf_hasil[simpul[1]]
    return array_simpul
```

csv_util.py

```
import csv

def read_csv_to_matrix(file):
    array = []
    with open(file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            # asumsi baris 0 adalah header
            if line_count != 0:
                array.append([int(i) for i in row])
            line_count += 1
        print(f'Processed {line_count} lines.')
        # return matrix
        return array

def read_csv_header(file):
    array = []
    with open(file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        for row in csv_reader:
            # asumsi baris 0 adalah header
            if line_count == 0:
                array = row
```

```
            line_count += 1
        print(f'Processed {line_count} lines.')
        return array

def write_csv():
    pass
```

main.py

```
from csv_util import read_csv_to_matrix, read_csv_header
from algoritma import dijkstra, lintasan


def fetchAllNodes():
    graf = read_csv_to_matrix('assets/Matriks.csv')
    return graf

def shortestPath(start, end):
    graf = read_csv_to_matrix('assets/Matriks.csv')

    simpul_dipilih = dijkstra(graf,start)

    return {
        'start': start,
        'end': end,
        'path': lintasan(end, simpul_dipilih[end], simpul_dipilih),
        'distance': simpul_dipilih[end][0]
    }

def shortestPathtoAllNode(start):
    graf = read_csv_to_matrix('assets/Matriks.csv')
    node_names = read_csv_header('assets/Matriks.csv')

    simpul_dipilih = dijkstra(graf,start)

    array = []

    for i in range(len(simpul_dipilih)):
        array.append({
            'node': i,
            'end': node_names[i],
            'path': lintasan(i, simpul_dipilih[i], simpul_dipilih),
            'distance': simpul_dipilih[i][0]
        })

    return {
        'data': sort(array),
        'start': start
    }

def sort(graf, type='asc'):
    new_graf = []
    for i in range(len(graf)):
        j = 0
        k = 0
        if  len(new_graf) > 0:
            while j < len(new_graf) and graf[i]['distance'] >
new_graf[j]['distance']:
                j += 1
                k = j
        new_graf.insert(k, {
            'node': graf[i]['node'],
            'end': graf[i]['end'],
```

```
            'path': graf[i]['path'],
            'distance': graf[i]['distance']
        })

    return new_graf

def main():
    pass


if __name__ == '__main__':
    main()
```

server.py

```
# Import untuk membuat server
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

# Import algoritma
from main import fetchAllNodes, shortestPath, shortestPathtoAllNode

# Inisialisasi server
app = FastAPI()

# Mengatur supaya server bisa diakses dari mana saja
origins = ["*"]
app.add_middleware(CORSMiddleware,allow_origins=origins,allow_credentials=Tr
ue,allow_methods=["*"],allow_headers=["*"])


# Daftar URL dari server

@app.get("/list")
def list():
    return fetchAllNodes()

@app.get("/path")
def shortest_path(start: int, end: int):
    return shortestPath(start, end)

@app.get("/maranatha")
def maranatha():
    return shortestPathtoAllNode(0)
```