Aplikasi Algoritma Dijkstra dalam Optimalisasi Rute Penerbangan Domestik di Indonesia

Christian Justin Hendrawan - 13522135
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522135@std.stei.itb.ac.id

***Abstract*—Algoritma yang ditemukan Edsger Dijkstra atau biasa yang disebut dengan algoritma Dijkstra telah menjadi salah satu algoritma yang paling umum dan efektif untuk menyelesaikan masalah pencarian rute terpendek dalam graf. Konsep inti dari algoritma ini adalah menemukan jalur terpendek dari satu titik ke semua titik lain dalam sebuah jaringan dengan mempertimbangkan bobot yang terkait dengan setiap lintasan. Pada makalah ini, akan dibahas mengenai aplikasi algoritma Dijkstra dalam optimalisasi rute penerbangan domestik di Indonesia.**

Masalah mengenai mengatur penerbangan dari satu daerah ke daerah lain sudah menjadi masalah yang sangat umum di dunia penerbangan. Bandara harus dapat mengetahui Cara untuk mengangkut penumpang dan barang dagangan antara daerah asal ke daerah tujuan dengan waktu yang lebih singkat dan biaya yang lebih rendah. Penentuan rute dengan biaya yang terendah dan waktu tersingkat dapat dilakukan dengan berbagai algortima, Salah satu algoritma yang dapat digunakan adalah Algoritma Dijkstra.

***Keywords*— algoritma Dijkstra, graf, penerbangan domestik, graf berbobot, rute penerbangan**

I. PENDAHULUAN

Indonesia adalah negara kepulauan terbesar di dunia, dengan 17.024 pulau yang tersebar dari sabang sampai Merauke yang membentang di antara dua samudra besar, yaitu Samudra Pasifik dan Samudra Hindia. Keberagaman geografis ini menciptakan tantangan unik dalam menjalankan sistem transportasi. Pola geografis Indonesia yang terdiri dari pulau-pulau besar seperti Jawa, Sumatera, Kalimantan, Sulawesi, Papua, serta ratusan pulau kecil menuntut adanya penerbangan domestik untuk menyambungkan masing-masing pulau.

Maka dari itu, penerbangan domestik di Indonesia bukan sekadar urusan transportasi semata, melainkan juga merupakan tulang punggung konektivitas antar-wilayah yang membentang dari ujung barat hingga timur negeri ini. Setiap tahun, lalu lintas udara di Indonesia terus meningkat secara signifikan. Dampaknya, dibutuhkan sumber daya yang besar untuk mengoperasikan setiap penerbangan. Kurangnya manajemen lalu lintas udara yang efektif dapat mengakibatkan pemborosan sumber daya pada penerbangan dan transit yang tidak diperlukan. Oleh karena itu, optimalisasi jalur penerbangan menjadi krusial dalam menangani masalah ini. Tidak hanya untuk meningkatkan efisiensi waktu perjalanan, tetapi juga untuk mengurangi biaya operasional secara signifikan.

Dalam konteks ini, Algoritma Dijkstra muncul sebagai alat yang potensial untuk membantu mengatasi tantangan tersebut dengan menemukan jalur penerbangan terpendek dan paling efisien antara titik-titik bandara di seluruh kepulauan Indonesia. Penerapan algoritma ini dalam optimasi rute penerbangan domestik di Indonesia bisa menjadi langkah penting dalam membangun sistem transportasi udara yang lebih handal dan efisien bagi negara ini.

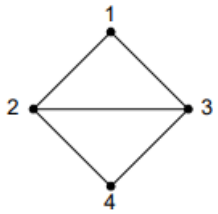
Makalah ini bertujuan untuk menjelajahi tidak hanya konsep dasar dari Algoritma Dijkstra, tetapi juga bagaimana aplikasinya secara konkret dalam konteks yang berkaitan dengan penerbangan domestik di Indonesia. Makalah ini akan membahas strategi untuk meningkatkan efisiensi manajemen lalu lintas udara dengan fokus pada penghubungan optimal antara berbagai daerah di Indonesia. Tujuannya adalah memindahkan penumpang sebanyak mungkin dengan jumlah penerbangan yang minimal, yang pada gilirannya akan membantu mengurangi biaya perjalanan udara secara signifikan.

II. LANDASAN TEORI

A. Graf

1) *Definisi Graf*

Graf merupakan sekumpulan objek terstruktur yang terdiri dari himpunan di mana beberapa pasangan objek mempunyai hubungan / keterkaitan tertentu.[2]



Gambar 1. Contoh graf [1]

Secara matematis, graf dinyatakan sebagai pasangan himpunan simpul dan sisi. Sebuah graf *G* dapat diungkapkan dengan notasi *G = (V, E)*, yang dalam hal ini :

V = himpunan tidak kosong dari simpul-simpul (*vertices*) = $\{v_1, v_2, \ldots v_n\}$

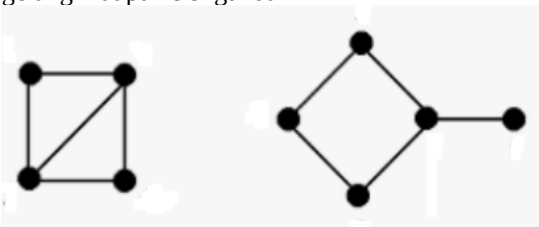
E = himpunan sisi yang menghubungkan sepasang simpul = $\{e_1, e_2, \ldots e_n\}$

Dari graf G pada gambar 1 adalah graf dengan V = {1,2, 3, 4} dan E ={ (1, 2), (1, 3), (2, 3), (2, 4), (3, 4) }

2) *Jenis Graf*

Graf dapat dikategorisasikan menjadi 2 jenis graf berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf yaitu :

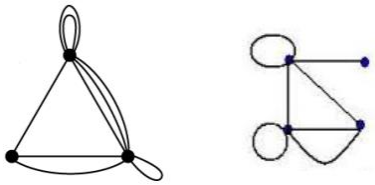
a) Graf sederhana (*simple graph*)
Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda



Gambar 2. Graf sederhana [1]

b) Graf tak sederhana (*unsimple graph*)
Graf yang mengandung sisi ganda atau gelang. Graf tak sederhana dibedakan lagi menjadi 2 jenis yakni jenis yang pertama adalah graf ganda (*multigraph*) dan graf semu (*pseudograph*). Definisi dari Graf ganda adalah graf yang mengandung sisi ganda tetapi tidak mengandung gelang. Definisi dari Graf semu adalah graf yang mengandung gelang.

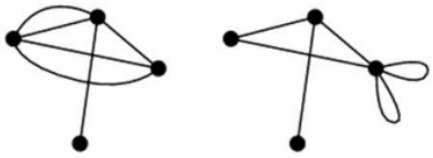
Gambar 3. Graf ganda [1]



Gambar 4. Graf semu [1]

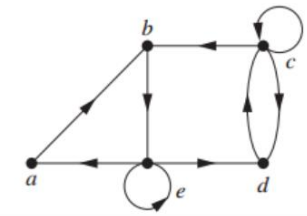
Berdasarkan orientasi arah pada sisi, secara umum graf dibedakan atas 2 jenis:

a) Graf tak-berarah (*undirected graph*)
Graf yang sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan.



Gambar 5. Graf tak-berarah [1]

b) Graf berarah (*directed graph atau digraph*)
Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah.



Gambar 6. Graf Berarah [1]

3) Terminologi Graf

a) Ketetanggaan
Dua buah simpul disebut bertetangga jika mereka saling terhubung secara langsung.

b) Bersisian
Untuk sembarang sisi e = (*vj* , *vk*) dikatakan e bersisian dengan simpul *vj* , atau e bersisian dengan simpul *vk*

c) Simpul Terpencil
Simpul terpencil merujuk pada simpul yang tidak memiliki simpul lain yang terhubung langsung dengannya.

d) Graf Kosong
Graf yang himpunan sisinya merupakan himpunan kosong (N_n).

e) Derajat
Derajat suatu simpul dalam graf adalah jumlah sisi yang terhubung langsung dengan simpul tersebut. Jumlah sisi ini dapat dihitung dengan menghitung jumlah tetangga dari simpul tersebut.

f) Lintasan
Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf *G* ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0 , e_1 , v_1 , e_2 , v_2 ,... , $v_n - 1$, e_n , v_n sedemikian sehingga $e1 = (v_0 , v_1)$, $e2 = (v_1 , v_2)$, ... , $en = (v_n\text{-}1 , v_n)$ adalah sisi-sisi dari graf *G*.

g) Siklus atau Sirkuit
Lintasan yang membentuk pola yang dimulai dan berakhir pada simpul yang sama disebut siklus atau sirkuit dalam sebuah graf.

h) Keterhubungan
Dua simpul, v_1 dan v_2, disebut terhubung jika ada jalur lintasan dari v_1 ke v_2. Sebuah graf *G* dikatakan terhubung (*connected graph*) jika untuk setiap pasangan simpul v_i dan v_j dalam himpunan *V*, terdapat jalur lintasan yang menghubungkan *vi* ke v_j. Jika tidak ada lintasan yang menghubungkan setiap pasangan simpul dalam graf, maka *G* disebut sebagai graf tak-terhubung (*disconnected graph*).

i) Upagraf atau Komplemen Upagraf
Upagraf atau subgraf G_1 dari graf $G = (V, E)$ adalah $G_1 = (V_1, E_1)$ dimana $V_1 \subseteq V$ dan $E_1 \subseteq E$.

Komplemen dari upagraf G_1 terhadap graf G adalah graf $G_2 = (V_2, E_2)$ dimana $E_2 = E - E_1$ dan V_2 adalah himpunan simpul dimana sisi-sisi dalam E_2 bertetangga dengan simpul-simpul dalam himpunan V_2.

j) Upagraf Merentang
Upagraf $G_1 = (V_1, E_1)$ dari graf $G = (V, E)$ disebut upagraf rentang jika himpunan simpul $V_1 = V$ (yaitu G_1 mengandung semua simpul yang ada di G).

k) Cut-Set
Cut-set dari sebuah graf terhubung G adalah himpunan sisi-sisi yang, jika dihapus dari G akan menyebabkan G tidak terhubung. Artinya, cut-set selalu memisahkan graf menjadi dua buah komponen terhubung yang terpisah.

l) Graf Berbobot
Graf berbobot adalah jenis graf di mana setiap sisi (*edge*) memiliki nilai atau bobot yang menggambarkan harga atau karakteristik tertentu pada setiap sisi tersebut.

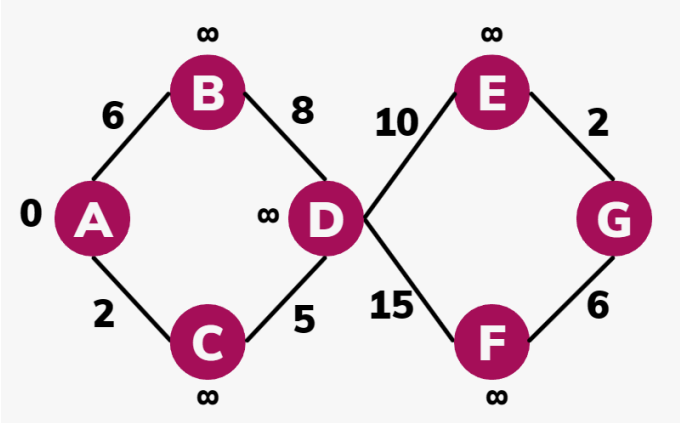
B. Algoritma Dijkstra

Algoritma Dijkstra adalah algoritma pencarian yang digunakan untuk menyelesaikan permasalahan dalam menentukan lintasan terpendek (*shortest path problem*) antara dua simpul / *node* dalam sebuah graf. Algoritma ini umumnya digunakan dalam graf berbobot positif, di mana setiap *edge* memiliki nilai numerik yang menunjukkan biaya atau jarak antara dua *node* yang terhubung.

Landasan dari algoritma Dijkstra terletak pada prinsip pencarian jalur terpendek secara bertahap. Algoritma ini menggunakan pendekatan *greedy*, artinya pada setiap langkah, algoritma memilih simpul yang memiliki jalur terpendek dari simpul awal yang sudah diketahui. Kemudian, algoritma secara iteratif memperbarui estimasi jalur terpendek ke setiap simpul yang terhubung dengan simpul saat ini, jika jalur baru yang ditemukan lebih pendek dari estimasi sebelumnya.

Algoritma Dijkstra memiliki langkah-langkah utama sebagai berikut :

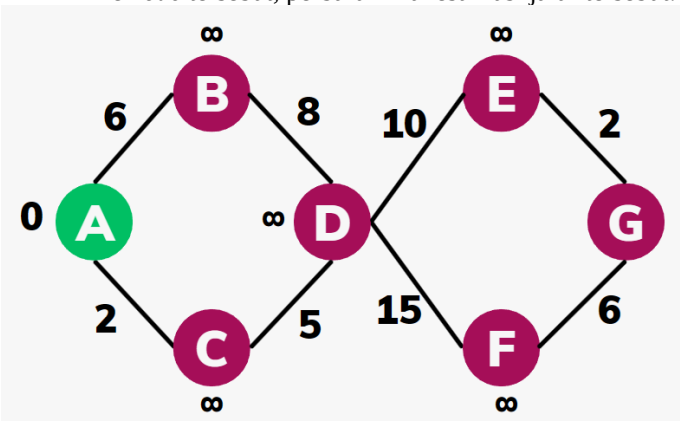
1. **Inisialisasi** : Tetapkan nilai estimasi jalur terpendek dari *node* awal ke semua *node* lainnya sebagai tak hingga (infiniti), kecuali *node* awal itu sendiri yang bernilai 0. Dalam kasus ini, kita akan menginisialisasi *node* B,C,D,E,F, dan G dengan infiniti dan menetapkan estimasi di *node* A dengan 0.



Gambar 7. Langkah inisialisasi algoritma Dijkstra
(Sumber : Penulis)

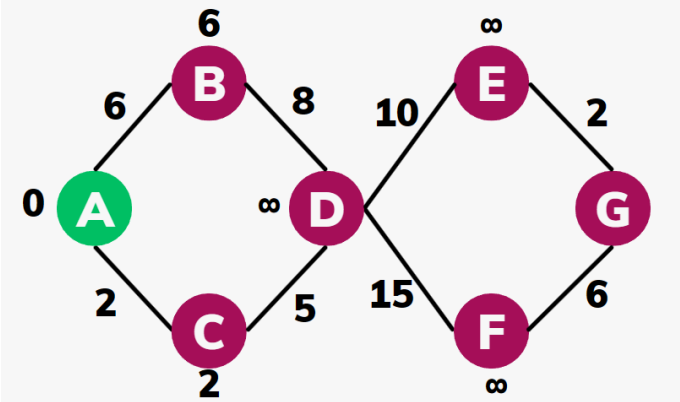
Keterangan tambahan : *node* awal pada graf ini adalah *node* A

2. **Iterasi** : Pilih *node* saat ini dan tandai sebagai sudah dikunjungi. Untuk setiap *node* tetangga yang belum dikunjungi, hitung panjang jalur baru dari *node* awal melalui *node* saat ini ke *node* tetangga tersebut. Jika jalur baru lebih pendek dari estimasi jalur sebelumnya ke *node* tersebut, perbarui nilai estimasi jalur tersebut.



Gambar 8. Langkah iterasi algoritma Dijkstra
(Sumber : Penulis)

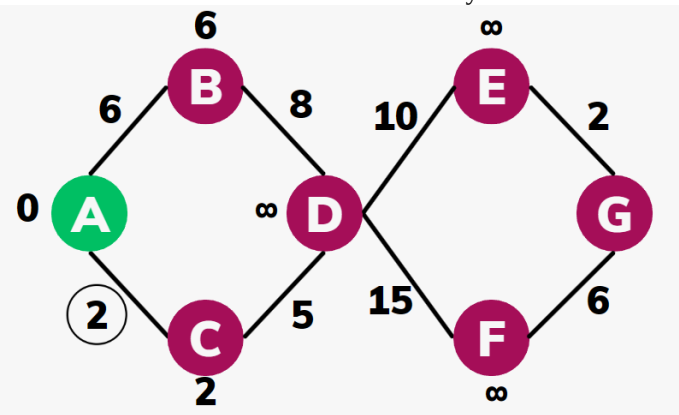
Keterangan tambahan : Saat ini *node* yang sedang diperiksa adalah *node* 0, maka tandai sudah dikunjungi.



Gambar 9. Langkah iterasi algoritma Dijkstra
(Sumber : Penulis)

Keterangan tambahan : Periksa tetangga yang belum dikunjungi dari *node* A yaitu B dan C.
Hitung jalur baru dari A ke B dan A ke C :
A ke B = 0 + 6 = 6 (lebih pendek dari ∞), update B menjadi 6
A ke C = 0 + 2 = 2 (lebih pendek dari ∞), update C menjadi 2

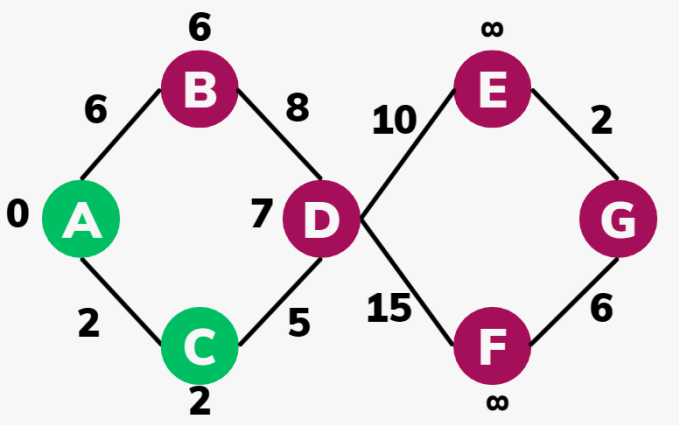
3. **Pemilihan node berikutnya** : Pilih *node* yang belum dikunjungi dengan estimasi jalur terpendek sebagai *node* saat ini untuk iterasi berikutnya.



Gambar 10. Langkah pemilihan node algoritma Dijkstra
(Sumber : Penulis)

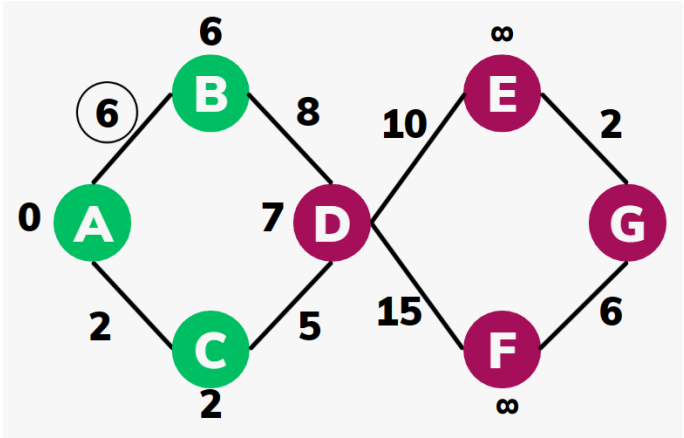
Keterangan tambahan : Pilih *node* yang belum dikunjungi dengan estimasi jalur terpendek sebagai *node* saat ini. Dalam hal ini, *node* C (dengan estimasi jalur 2) adalah *node* saat ini.

4. Ulangi Langkah 2 dan 3 sampai semua *node* telah dikunjungi atau estimasi jalur terpendek ke semua *node* telah ditentukan.



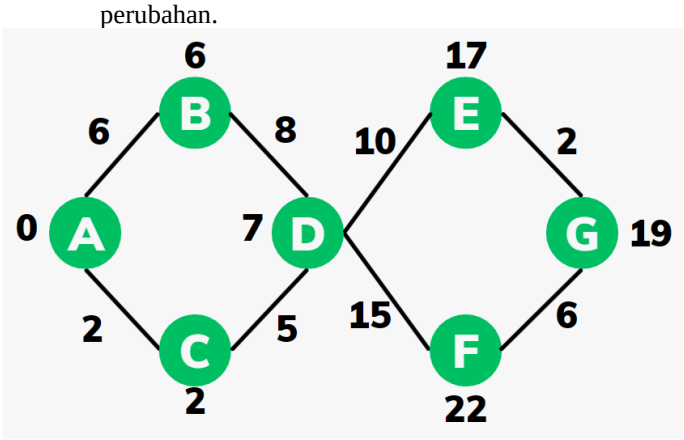
Gambar 11. Langkah iterasi ulang algoritma Dijkstra
(Sumber : Penulis)

Keterangan tambahan : Mengulangi langkah iterasi, dan menandai C sebagai *node* yang sudah dikunjungi. Periksa tetangga C yang belum dikunjungi dari C, yaitu D. Hitung jalur ke D yaitu C ke D : 2 + 5 = 7 (lebih pendek dari ∞), update D menjadi 7



Gambar 12. Langkah pemilihan node dan iterasi algoritma Dijkstra
(Sumber : Penulis)

Keterangan tambahan : Pilih *node* yang belum dikunjungi dengan estimasi jalur terpendek sebagai *node* saat ini. Di sini, *node* B (dengan estimasi jalur 6) adalah *node* saat ini. Periksa tetangga-tetangga yang belum dikunjungi dari B, yaitu D. Hitung jalur baru dari B ke D: 6 + 8 = 14 (lebih panjang dari 7), tidak ada perubahan.



Gambar 13. Semua simpul sudah terdefinisi jaraknya
(Sumber : Penulis)

Teruskan iterasi sampai semua *node* telah dikunjungi atau estimasi jalur terpendek ke semua *node* telah ditentukan.

C. Graf Rute Penerbangan

Graf dalam konteks jalur penerbangan sering digunakan untuk merepresentasikan hubungan antara bandara atau kota sebagai *node*. Setiap *node* memiliki identitas unik, misalnya, nama bandara atau kode kota. Dan penerbangan langsung antara kedua *node* sebagai *edge*. Dalam hal ini, bobot *edge* berupa waktu tempuh dan biaya bensin pesawat yang dikeluarkan. Dalam graf penerbangan, graf yang digunakan sering kali merupakan graf berarah. Artinya, penerbangan dari kota A ke kota B mungkin tidak memiliki penerbangan balik langsung dari kota B ke kota A.

Gambar 14. Graf rute penerbangan [6]

III. METODE

A. Rute Penerbangan Domestik

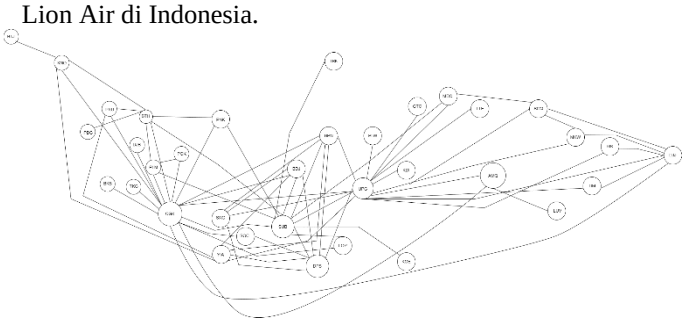
Untuk mendapatkan pemahaman lebih mendalam terhadap graf rute penerbangan, maka berikut akan ditinjau graf rute penerbangan domestik Lion Air pada tahun 2023.

Gambar 15. Rute Penerbangan Lion Air tahun 2023 [7]

Dalam menganalisis rute penerbangan domestik di Indonesia, terdapat batasan-batasan yang dibuat oleh kami. Pada kenyataannya, graf berarah pada peta rute penerbangan domestik diwakili dengan arah spesifik penerbangan antara kota. Artinya, penerbangan dari kota A ke kota B mungkin tidak memiliki penerbangan balik langsung dari kota B ke kota A. Namun, untuk tujuan penelitian ini, kami memodifikasi graf tersebut menjadi graf tak-berarah untuk mengeksplorasi keterhubungan lokasi secara lebih holistik.

Dengan mengonversi graf berarah menjadi graf tak-berarah, akan terbentuk gambaran potensi koneksi dua arah antara lokasi tanpa mempertimbangkan arah penerbangan spesifik, memungkinkan evaluasi yang lebih komprehensif terhadap rute alternatif dan hubungan antar lokasi.

Dengan batasan yang sudah dibuat, maka berikut adalah graf berarah yang menggambarkan rute penerbangan domestik Lion Air di Indonesia.



Gambar 16. Graf rute penerbangan domestik Lion Air tak berarah (Sumber : Penulis https://drive.google.com/file/d/130YXcB7FYEEEPN4Rii4N_HfQrDUDRX6l/view?usp=sharing)

B. Analisa Bobot Pada Edge Graf Penerbangan

Untuk mengimplementasikan algoritma Dijkstra, maka graf penerbangan harus merupakan graf berbobot. Dalam menganalisis bobot pada *edge*, kami mempertimbangkan beberapa faktor yang memengaruhi efisiensi dan performa penerbangan, termasuk jarak geografis, waktu tempuh penerbangan, biaya tiket, dan faktor lainnya yang relevan. Namun, dari sekian faktor yang kami evaluasi, kami menetapkan waktu tempuh sebagai faktor paling krusial dalam menilai efisiensi penerbangan. Melalui estimasi waktu tempuh, kami dapat mengkonversinya menjadi estimasi penggunaan bahan bakar dalam penerbangan. Meskipun menyadari banyak faktor yang dapat memengaruhi penggunaan bensin dalam penerbangan seperti kondisi cuaca, beban pesawat, dan rute penerbangan, dalam penelitian ini kami membatasi variabel yang digunakan untuk memastikan konsistensi dan fokus dalam analisis kami. Dalam studi ini kami memilih menggunakan model pesawat komersial yang umum digunakan, yakni Boeing 747, sebagai standar referensi. Berdasarkan data, Boeing 747 diperkirakan menggunakan sekitar 3.78541 liter bensin pesawat per detik, setara dengan 13627.48 liter bensin per jam[8]. Dengan mengacu pada data ini, kami melakukan perhitungan estimasi penggunaan bensin untuk penerbangan tertentu. Sebagai contoh, perjalanan dari bandara Jakarta Soekarno Hatta (CGK) menuju bandara Surabaya (SUB) dengan waktu tempuh 1 jam 30 menit menghasilkan estimasi penggunaan bensin sebesar 1,5 * 13627.48 = 20441.22 liter.

Dalam dunia nyata, transisi atau transit di bandara tertentu dalam perjalanan penerbangan merupakan faktor yang signifikan yang mempengaruhi waktu tempuh. Oleh karena itu, implementasi algoritma Dijkstra dalam manajemen rute penerbangan diharapkan dapat memaksimalkan pengelolaan transit dan mengurangi waktu tempuh secara keseluruhan. Hal ini diharapkan dapat berkontribusi pada pengurangan penggunaan bensin pesawat serta mengoptimalkan efisiensi dalam industri transportasi udara.

IV. PEMBAHASAN

Langkah pertama dalam analisis rute penerbangan adalah memodelkan peta rute tersebut ke dalam struktur yang memungkinkan evaluasi dengan lebih terperinci. Dalam hal ini, menggunakan representasi graf berbobot menjadi kunci utama. Graf ini memungkinkan kami untuk menerapkan algoritma Dijkstra, alat penting dalam perhitungan rute terpendek. Di sini, setiap *node* dalam graf kami menandakan sebuah bandara, sementara setiap *edge* mencerminkan estimasi penggunaan bensin dalam liter. Pendekatan ini memfasilitasi analisis yang komprehensif terhadap pengeluaran bahan bakar pesawat dalam rangka mengevaluasi efisiensi dan kinerja rute penerbangan.

Pada bagian metode, sub bagian A, sudah terdapat graf berarah yang merepresentasikan rute penerbangan. Namun graf tersebut belum dapat dianalisis karena belum memuat bobot pada setiap *edge*-nya.

Pada kalkulasi bobot *edge* ini, kami memberikan asumsi bahwa waktu tempuh dari bandara A1 ke bandara A2 adalah waktu tempuh yang sama dengan bandara A2 ke bandara A1. Dengan memanfaatkan rumus perhitungan pengeluaran bensin pada bagian metode, sub bagian B, kami dapat menghitung pengeluaran bensin untuk setiap penerbangan. Berikut adalah

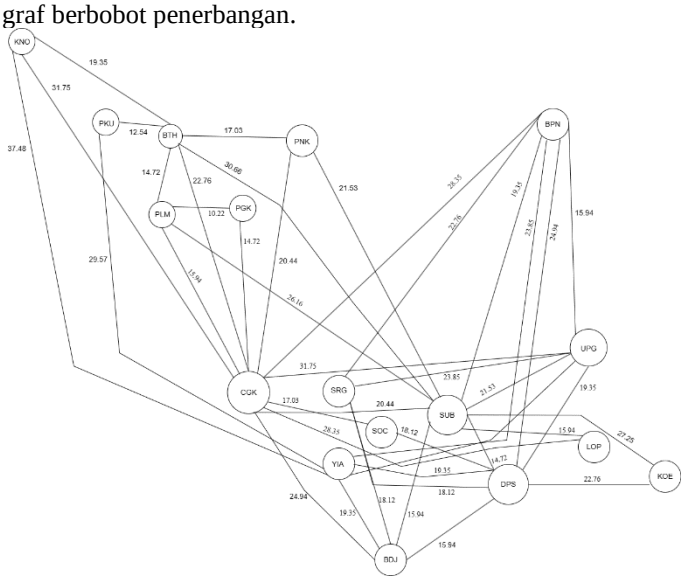
tabel pengeluaran bensin untuk setiap penerbangan yang nantinya akan dipakai sebagai bobot *edge*.

Tabel 1. Kalkulasi pengeluaran bensin untuk setiap penerbangan maskapai Lion Air di tahun 2023

Bandara		Waktu tempuh (jam)[9]	Pengeluaran bensin (kL)
A1	A2		
CGK	KNO	2.33	31.75
CGK	PKU	1.75	23.85
CGK	DJB	1.33	18.12
CGK	PLM	1.17	15.94
CGK	BKS	1.25	17.03
CGK	TKG	0.83	11.31
CGK	BTH	1.67	22.76
CGK	PGK	1.08	14.72
CGK	PNK	1.5	20.44
CGK	BPN	2.08	28.35
CGK	BDJ	1.83	24.94
CGK	UPG	2.33	31.75
CGK	SUB	1.5	20.44
CGK	LOP	2.08	28.35
CGK	SOC	1.25	17.03
CGK	DJJ	5	68.14
UPG	BPN	1.17	15.94
UPG	SRG	1.75	23.85
UPG	YIA	2	27.25
UPG	SUB	1.58	21.53
UPG	DPS	1.42	19.35
UPG	TIM	2.83	38.57
UPG	DJJ	3.5	47.70
UPG	AMQ	1.75	23.85
UPG	BIK	2.75	37.48
UPG	MKW	2.67	36.39
UPG	SOQ	2.08	28.35
UPG	TTE	1.92	26.16
UPG	MDC	1.92	26.16
UPG	GTO	1.42	19.35
UPG	PLW	1.17	15.94
UPG	KDI	0.92	12.54
DPS	YIA	1.42	19.35
DPS	SOC	1.33	18.12
DPS	SRG	1.33	18.12
DPS	SUB	1.08	14.72
DPS	BDJ	1.17	15.94
DPS	BPN	1.83	24.94
DPS	KOE	1.67	22.76
KNO	BTJ	1.17	15.94
KNO	BTH	1.42	19.35
KNO	YIA	2.75	37.48
PKU	BTH	0.92	12.54
PKU	YIA	2.17	29.57
PLM	BTH	1.08	14.72
PLM	PGK	0.75	10.22
PLM	SUB	1.92	26.16
BTH	PDG	1.25	17.03
BTH	SUB	2.25	30.66
BTH	PNK	1.25	17.03
PNK	SUB	1.58	21.53
BDJ	SRG	1.33	18.12
BDJ	YIA	1.42	19.35
BDJ	SUB	1.17	15.94
TRK	SUB	2	27.25
BPN	SRG	1.67	22.76
BPN	YIA	1.75	23.85
BPN	SUB	1.42	19.35
MDC	SUB	2.83	38.57
MDC	SOQ	1.42	19.35
AMQ	LUV	1.17	15.94
SOQ	DJJ	1.75	23.85
SOQ	MKW	0.92	12.54
DJJ	MKW	1.42	19.35
DJJ	BIK	1.33	18.12
DJJ	TIM	1.25	17.03
LOP	SUB	1.17	15.94
KOE	SUB	2	27.25

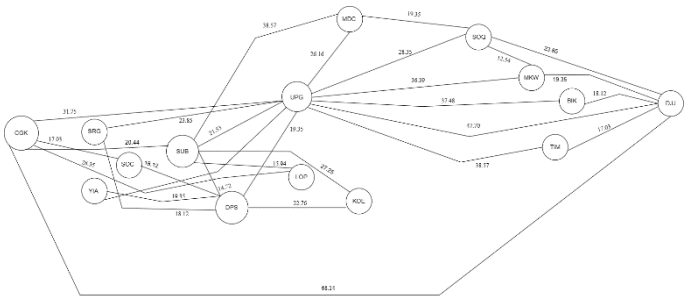
Dengan memanfaatkan data yang tersaji dalam tabel 1 mengenai bobot *edge*, sekarang kami dapat mengimplementasikan algoritma Dijkstra untuk menentukan jalur dengan pengeluaran bensin pesawat yang paling efisien. Untuk mempermudah perhitungan di tahap berikutnya, kami menyederhanakan graf yang digunakan sebagai dasar analisis. Upaya penyederhanaan ini melibatkan penghapusan beberapa titik bandara pada peta penerbangan yang tidak berperan sebagai bandara pertemuan antara dua jalur atau lebih. Langkah ini bertujuan untuk menyajikan analisis yang lebih terfokus pada rute-rute yang berpotensi memiliki pengeluaran bahan bakar yang lebih optimal dalam konteks penerbangan. Penyederhaan selanjutnya melibatkan membagi graf menjadi beberapa upagraf. Langkah ini bertujuan untuk memberikan penjelasan yang lebih jelas dan terperinci.

Berikut adalah 2 upagraf yang merepresentasikan keseluruhan graf berbobot penerbangan.



Gambar 17. Upagraf berbobot yang merepresentasikan penerbangan yang menghubungkan Jawa, Kalimantan, dan Sumatera (Sumber : Penulis https://drive.google.com/file/d/1L5aphfyOqQFD8NqhFtPPsGghdsNwJC9V/view?usp=sharing)

Keterangan tambahan : Bandara BDJ sengaja diletakkan di bawah, bukan di atas SUB agar pembaca lebih mudah untuk melihat setiap bobot *edge*.



Gambar 18. Upagraf berbobot yang merepresentasikan penerbangan yang menghubungkan Jawa, Sulawesi, dan Papua (Sumber : Penulis https://drive.google.com/file/d/1Pas6MajBJBjsK451D9xErp1cu32CT8YF/view?usp=sharing)

Bagian penting dari penelitian ini adalah aplikasi praktis dari algoritma Dijkstra dalam konteks optimalisasi rute penerbangan. Maka dari itu, kami mengimplementasikan program yang memanfaatkan algoritma Dijkstra dalam bahasa Python. Program ini memungkinkan kami untuk mengidentifikasi dan mengevaluasi jalur penerbangan yang memiliki penggunaan bahan bakar yang lebih optimal secara efisien dengan menggunakan langkah-langkah algoritma Dijkstra yang sudah dibahas di makalah ini.

Terdapat 2 file yang kami buat untuk menciptakan program ini, dijksta.py dan main_app.py. dijkstra.py adalah file yang bekerja untuk melakukan kalkulasi dan pengimplementasian algoritma Dijkstra, serta menggambar graf yang nantinya akan ditampilkan program. Lalu ada main_app.py yang berfungsi sebagai main file dan pengeksekusian *GUI* program. Berikut adalah penjelasan lebih lanjut mengenai kode program.

Pertama-tama, kami mengimplementasikan algoritma dijkstra dalam 1 fungsi yang disebut dengan dijkstra. Parameter yang diminta berupa graph dan start *node*. Struktur graph pada program ini adalah representasi dari sebuah graf dalam bentuk kamus Python. Setiap kunci dalam kamus utama mewakili sebuah simpul dalam graf, dan nilai dari setiap kunci tersebut adalah kamus lain yang berisi simpul-simpul yang dapat dicapai dari simpul tersebut dan bobot dari setiap tepi yang menghubungkan simpul tersebut dengan tetangganya. Contohnya adalah, "CGK": {"KNO": 31.75, "PKU": 23.85, "DJB": 18.12, ...}.

Awal-awal, fungsi tersebut akan menginisialisasi jarak dari simpul awal ke semua simpul lainnya tak berhingga dan Jarak dari simpul awal ke dirinya sendiri diatur menjadi 0. Setelah itu, dibuatlah kamus paths dimana setiap simpul diinisialisasi dengan jalur kosong dan jalur dari simpul awal ke dirinya sendiri hanya berisi simpul awal. Langkah selanajutnya, dibuatlah daftar nodes_to_visit yang berisi semua simpul dalam graph. Lalu, fungsi menentukan simpul yang sedang dikunjungi / current_node dengan jarak terkecil dari simpul awal. Ketika simpul sudah dikunjungi, maka simpul tersebut dihapus dari daftar nodes_to_visit. Lalu untuk setiap tetangga dari current_node, fungsi menghitung jarak melalui current_node ke tetangga tersebut. Jika jarak ini lebih kecil dari jarak terpendek yang ditemukan sejauh ini, jarak terpendek dan jalur terpendek diperbarui. Terakhir, fungsi mengembalikan kamus distances dan paths yang berisi jarak terpendek dan jalur terpendek dari simpul awal ke semua simpul lainnya.

```
def dijkstra(graph, start):
    # Inisialisasi jarak dari simpul awal ke semua simpul
    lainnya sebagai tak terhingga
    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    # Inisialisasi semua jalur sebagai jalur kosong
    paths = {node: [] for node in graph}
    # Jalur dari simpul awal ke dirinya sendiri hanya berisi
    simpul awal
    paths[start] = [start]

    # Daftar simpul yang harus dikunjungi
    nodes_to_visit = list(graph.keys())

    while nodes_to_visit:
        # Pilih simpul dengan jarak terkecil dari simpul awal
        current_node = min(nodes_to_visit, key=lambda node:
    distances[node])
        # Hapus simpul tersebut dari daftar simpul yang harus
    dikunjungi
        nodes_to_visit.remove(current_node)

        for neighbor, weight in graph[current_node].items():
            # Hitung jarak melalui simpul saat ini ke tetangga
            distance = distances[current_node] + weight
            # Jika jarak ini lebih kecil dari jarak terpendek yang
    ditemukan sejauh ini
            if distance < distances[neighbor]:
                # Perbarui jarak terpendek
                distances[neighbor] = distance
                paths[neighbor] = paths[current_node] + [neighbor]

    return distances, paths
```

Setelah itu pada dijkstra.py, terdapat fungsi draw_graph, dengan memanfaatkan *library networkx* dan *matplotlib.pyplot*, maka kami dapat membuat visualisasi dari graf rute penerbangan. Selain itu, kami juga dapat membuat visualisasi arah terpendek yang harus diambil dari start *node* menuju *node* tujuan

```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(graph, path):
    # Membuat objek graf kosong
    G = nx.Graph()

    # Menambahkan setiap tepi dan bobotnya ke graf
    for node, edges in graph.items():
        for edge, weight in edges.items():
            G.add_edge(node, edge, weight=weight)

    pos = nx.spring_layout(G, seed=42)
    nx.draw(G, pos, with_labels=True)
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos,
    edge_labels=labels)

    # Membuat graf berarah untuk jalur terpendek
    H = nx.DiGraph()
```

```
    path_edges = [(path[i], path[i+1]) for i in range(len(path) -
    1)]
    H.add_edges_from(path_edges)
    nx.draw_networkx_edges(H, pos, edgelist=path_edges,
    edge_color='r', width=2, arrowstyle='-|>', arrowsize=20)

    plt.show()
```

Yang terakhir, main_app.py adalah aplikasi antarmuka pengguna (GUI) untuk menerima input simpul awal dan simpul akhir, menghitung jalur terpendek menggunakan algoritma Dijkstra, dan menampilkan hasilnya baik dalam bentuk teks maupun visualisasi graf. Kami memanfaatkan *library customtkinter* untuk membuat *GUI* program. *customtkinter* adalah modul kustom yang memperluas fungsi dari modul tkinter bawaan Python.

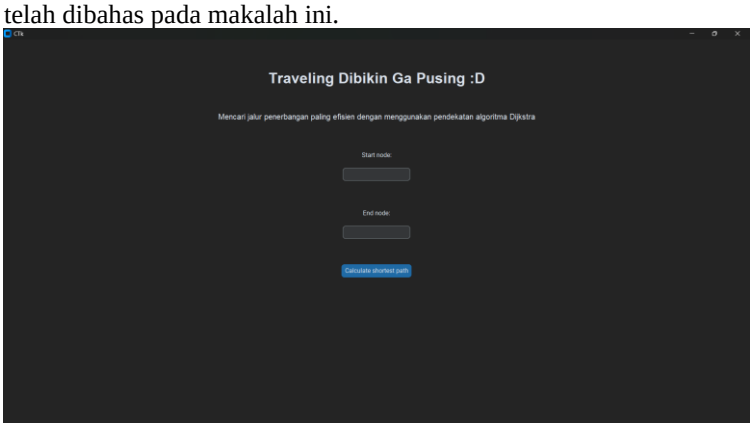
```
import customtkinter as ctk
from tkinter import messagebox
from dijkstra import dijkstra, draw_graph
def calculate_path():
    start_node = start_entry.get()
    end_node = end_entry.get()

    if start_node not in graph or end_node not in graph:
        messagebox.showerror("Error", "Invalid start or end
    node.")
        return

    distances, paths = dijkstra(graph, start_node)
    formatted_distance = "{:.2f}".format(distances[end_node])

    result_text.set(
        f"The shortest distance from {start_node} to
    {end_node} is {formatted_distance}\n"
        f"The shortest path from {start_node} to {end_node} is
    {paths[end_node]}"
    )

    draw_graph(graph, paths[end_node])

#Data graf dalam bentuk kamus python yang berukuran
    terlalu besar jika diperlihatkan di makalah ini, berikut
    ditunjukan sedikit data graf yang digunakan
graph = {
    "CGK": {
        "KNO": 31.75,
        "PKU": 23.85,
        "DJB": 18.12,
        "PLM": 15.94,
        "BKS": 17.03,
}

root = ctk.CTk()
root.geometry("500x400")

title = ctk.CTkLabel(
    root, text="Traveling Dibikin Ga Pusing :D",
    font=("Helvetica", 30, "bold")
)
title.place(relx=0.5, rely=0.1, anchor="center")
```

```
subtitle = ctk.CTkLabel(root, text="Mencari jalur
    penerbangan paling efisien dengan menggunakan
    pendekatan algoritma Dijkstra", font=("Helvetica", 16))
subtitle.place(relx=0.5, rely=0.2, anchor='center')

start_label = ctk.CTkLabel(root, text="Start node:")
start_label.place(relx=0.5, rely=0.3, anchor="center")

start_entry = ctk.CTkEntry(root)
start_entry.place(relx=0.5, rely=0.35, anchor="center")

end_label = ctk.CTkLabel(root, text="End node:")
end_label.place(relx=0.5, rely=0.45, anchor="center")

end_entry = ctk.CTkEntry(root)
end_entry.place(relx=0.5, rely=0.5, anchor="center")

calculate_button = ctk.CTkButton(
    root, text="Calculate shortest path",
    command=calculate_path
)
calculate_button.place(relx=0.5, rely=0.6, anchor="center")

result_text = ctk.StringVar()
result_label = ctk.CTkLabel(root, textvariable=result_text)
result_label.place(relx=0.5, rely=0.7, anchor="center")

root.mainloop()
```

V. TAMPILAN PROGRAM DAN TEST CASE

Sebelum menunjukkan tampilan dan testcase program, harus diketahui bahwa terdapat *error* yang belum dapat ditangani oleh kami, yakni peletakkan simpul pada visualisasi graf *matplotplib* yang tidak sesuai dengan graf penerbangan pada gambar 16,17, atau 18. Namun, di luar dari peletakannya yang tidak sesuai, semua simpul dan sisi sudah terhubung dengan sesuai. Dan arah dari jalur yang ditemukan juga sesuai.

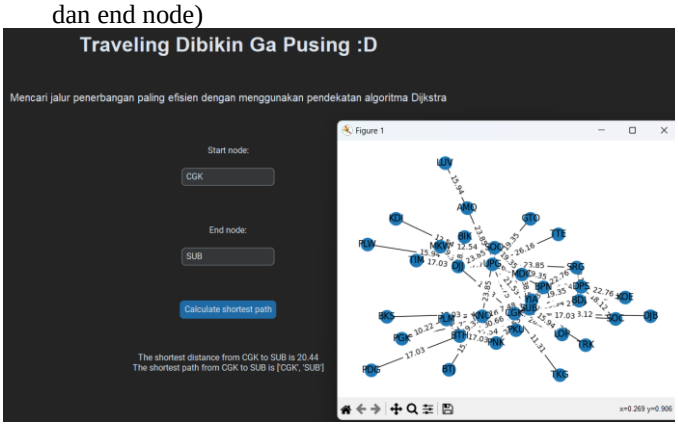
Berikut adalah contoh tampilan dari aplikasi GUI yang telah dibahas pada makalah ini.



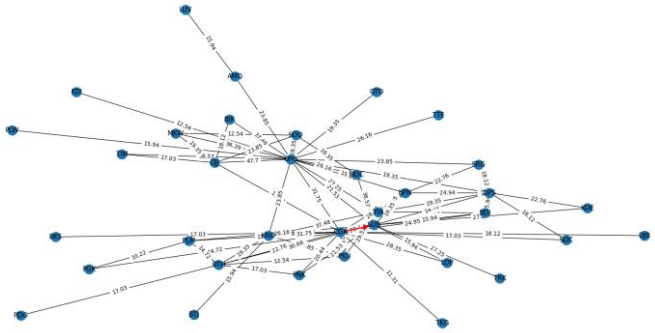
Gambar 19. Tampilan aplikasi GUI program
(Sumber : Penulis)

Seperti yang sudah dibahas sebelumnya, program akan meminta input simpul awal dan simpul akhir atau simpul tujuan. Sekarang, kami akan membahas testcase pada program.

1. Start node : CGK, End node: SUB (Kasus dimana terdapat jalur yang langsung menghubungkan start node dan end node)

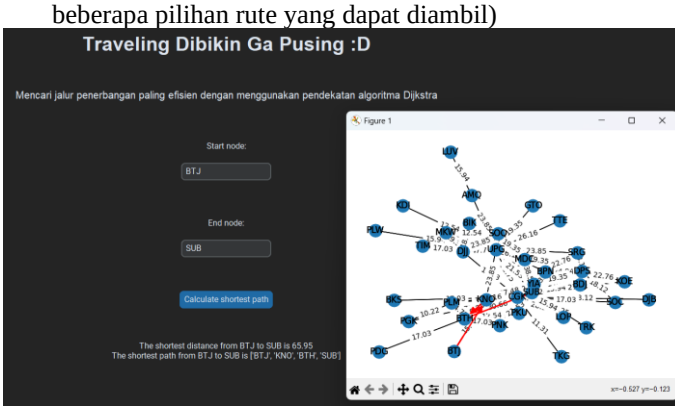


Gambar 20. Tampilan program test case 1
(Sumber : Penulis)

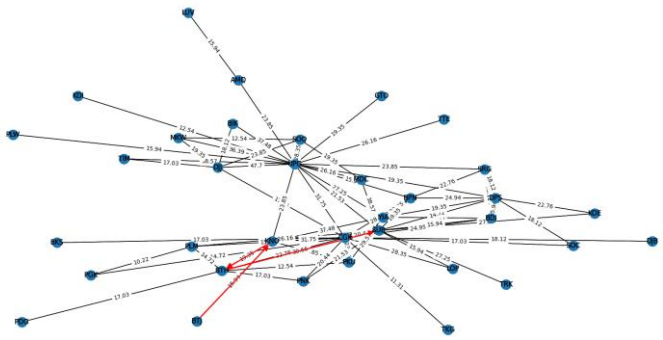


Gambar 21. Tampilan visuallisasi graf full screen 1
(Sumber : Penulis)

2. Start node : BTJ, End node: SUB (Kasus dimana terdapat beberapa pilihan rute yang dapat diambil)

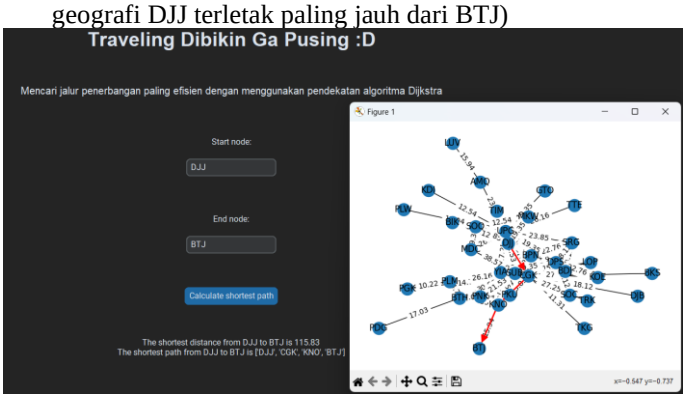


Gambar 22. Tampilan program test case 2
(Sumber : Penulis)

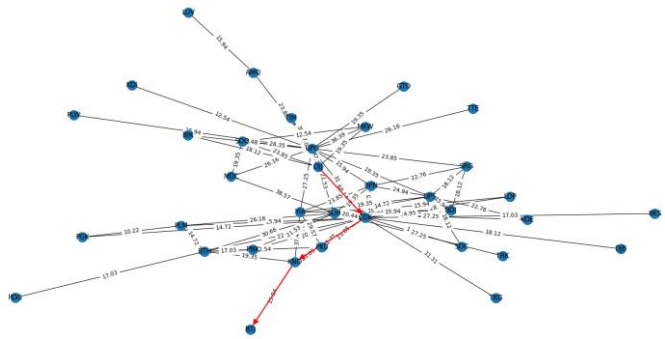


Gambar 23. Tampilan visuallisasi graf full screen 2
(Sumber : Penulis)

3. Start node : DJJ, End node : BTJ (Kasus dimana secara geografi DJJ terletak paling jauh dari BTJ)



Gambar 23. Tampilan program test case 3
(Sumber : Penulis)



Gambar 24. Tampilan visuallisasi graf full screen 3
(Sumber : Penulis)

VI. KESIMPULAN DAN SARAN

Makalah ini telah membahas implementasi yang mendalam tentang algoritma Dijkstra dalam konteks manajemen rute penerbangan. Dijkstra, dengan keunggulan dalam menentukan jalur terpendek dalam graf berbobot, terbukti menjadi alat yang kuat dalam merencanakan rute penerbangan yang efisien.

Dalam makalah ini juga, kami berhasil merancang dan mengimplementasikan algoritma Dijkstra dalam analisis rute penerbangan sebagai upaya untuk mengoptimalkan penggunaan bahan bakar pesawat. Analisis kami menunjukkan bahwa pendekatan ini memberikan kontribusi dalam pemahaman yang lebih mendalam terhadap rute penerbangan yang memiliki potensi penggunaan bahan bakar yang lebih hemat.

Namun, masih terdapat beberapa kekurangan dari program ini yang belum dapat ditangani seperti salah satunya adalah visualisasi graf yang tidak sesuai dengan bentuk graf yang

diingini. Saran jika program ini ingin dikembangkan adalah memperindah GUI program serta memperbaiki visualisasi graf.

VII. Ucapan Terima Kasih

Saya panjatkan puji dan syukur kepada Tuhan yang Maha Esa atas berkat dan rahmat-Nya, makalah matematika diskrit ini dapat diselesaikan dengan tepat waktu. Terima kasih sebesar-besarnya juga kepada Pak Dr. Ir. Rinaldi, M.T. atas jasa beliau yang sudah mengajarkan prinsip matematika diskrit dan pelajaran hidup lainnya yang tak dapat dihitung nilainya. Terima kasih juga kepada seluruh teman-teman saya yang sudah membantu dan menyemangati dalam pengerjaan makalah ini. Saya juga ingin berterima kasih kepada diri saya sendiri karena sudah mengeluarkan seluruh tenaga, waktu, dan hal lainnya dalam pengerjaan makalah ini.

VIII. Lampiran

Berikut adalah link github untuk program yang telah dibuat pada makalah ini : ChrisCS50X/Aplikasi-Dijkstra-13522135: Mengoptimalisasi rute penerbangan domestik di Indonesia menggunakan algoritma Dijkstra (github.com)

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/matdis23-24.htm
[2] Mushthofa (2021). Informatika untuk SMA Kelas X (PDF). Jakarta: Pusat Kurikulum dan Perbukuan. hlm. 246. ISBN 978-602-244-506-7. Diarsipkan (PDF) dari versi asli tanggal 2022-05-28. Diakses tanggal 2021-12-28W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.
[3] Rinaldi Munir, Diktat Kuliah Matematika Diskrit Edisi Keempat
[4] What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm - GeeksforGeeks
[5] Algoritma Dijkstra – Master of Computer Science (binus.ac.id)
[6] Flight-Scheduling - FLTDepartment.com
[7] rute pesawat: Rute Penerbangan Lion Air
[8] How Much Fuel Does an International Plane Use for a Trip? | HowStuffWorks
[9] Peta Rute dan Destinasi Lion Air - FlightConnections

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 2023

Christian Justin Hendrawan 13522135