SKRIPSI

PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

Diajukan Untuk Memenuhi Salah Satu Syarat Meraih Gelar Sarjana Komputer pada Jurusan Teknologi Informasi

DISUSUN OLEH

Mhd RajaSyah Pardede
2009020107

PROGRAM STUDI TEKNOLOGI INFORMASI FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA MEDAN

2024

PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom) dalam Program Studi Teknologi Informasi pada Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah Sumatera Utara

OLEH :

Mhd RajaSyah Pardede
2009020107

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS MUHAMMADIYAH
SUMATERA UTARA
MEDAN
2023

LEMBAR PENGESAHAN

Judul Skripsi	:	**PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK**
Nama Mahasiswa	:	MHD RAJASYAH PARDEDE
NPM	:	2009020107
Program Studi	:	TEKNOLOGI INFORMASI

Menyetujui
Komisi Pembimbing

(Mahardika Prawira Tanjung, S.kom,. M.kom)
NIDN. 0117088902

Ketua Program Studi

(Fatma Sari Hutagulung, M.Kom)
NIDN. 0117019301

Dekan

(Dr. Al-Khowarizmi, S.Kom., M.Kom.)
NIDN. 0127099201

PERNYATAAN ORISINALITAS

PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

SKRIPSI

Saya menyatakan bahwa karya tulis ini adalah hasil karya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing disebutkan sumbernya.

Medan, 14 September 2024

Yang membuat pernyataan



Mhd RajaSyah Pardede

2009020107

PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Muhammadiyah Sumatera Utara, saya bertanda tangan dibawah ini:

Nama	: Mhd Rajasyah Pardede
NPM	: 2009020107
Program Studi	: Teknologi Informasi
Karya Ilmiah	: Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Muhammadiyah Sumatera Utara Hak Bedas Royalti Non-Eksekutif *(Non-Exclusive Royalty free Right)* atas penelitian skripsi saya yang berjudul:

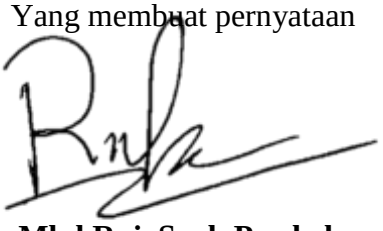
PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksekutif ini, Universitas Muhammadiyah Sumatera Utara berhak menyimpan, mengalih media, memformat, mengelola dalam bentuk database, merawat dan mempublikasikan Skripsi saya ini tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis dan sebagai pemegang dan atau sebagai pemilik hak cipta.

Demikian pernyataan ini dibuat dengan sebenarnya.

Medan, 14 September 2024

Yang membuat pernyataan



Mhd RajaSyah Pardede

2009020107

RIWAYAT HIDUP



DATA PRIBADI

Nama Lengkap	: Mhd Rajasyah Pardede
Tempat dan Tanggal Lahir	: Kolam, 2000-25-10
Alamat Rumah	: Jl. Utama II Desa Kolam
Telepon/Faks/HP	: 081362437060
E-mail	: rajasyaah@gmail.com
Instansi Tempat Kerja	:
Alamat Kantor	:

DATA PENDIDIKAN

SD	:	TAMAT: 2007-2012
SMP	:	TAMAT: 2013-2015
SMA	:	TAMAT: 2016-2019

KATA PENGANTAR



Pendahuluan

Penulis tentunya berterima kasih kepada berbagai pihak dalam dukungan serta doa dalam penyelesaian skripsi. Penulis juga mengucapkan terima kasih kepada:

1. Bapak Prof. Dr. Agussani, M.AP., Rektor Universitas Muhammadiyah Sumatera Utara (UMSU)
2. Bapak Dr. Al-Khowarizmi, S.Kom., M.Kom. Dekan Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) UMSU.
3. Bapak/Ibu Fatma Sari Hutagulung, M.Kom Ketua Program Studi Teknologi Informasi
4. Bapak/Ibu Mhd. Basri, S.Si, M.Kom Sekretaris Program Studi Teknologi Informasi
5. Pembimbing Mahardika Prawira Tanjung, S.kom,. M.kom
6. ……..
7. Semua pihak yang terlibat langsung ataupun tidak langsung yang tidak dapat penulis ucapkan satu-persatu yang telah membantu penyelesaian skripsi ini.

PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

ABSTRAK

Isi Abstrak dalam Bahasa Indonesia.
Pencarian rute terpendek merupakan salah satu permasalahan fundamental dalam bidang ilmu komputer, khususnya pada sistem informasi geografis, jaringan komputer, dan aplikasi navigasi. Algoritma Dijkstra telah lama digunakan untuk menyelesaikan masalah ini secara efektif. Namun, dalam kasus graf berskala besar, efisiensi waktu pencarian menjadi tantangan tersendiri. Algoritma Bidirectional Dijkstra hadir sebagai solusi alternatif yang menjalankan pencarian dari dua arah secara simultan—dari titik awal dan titik tujuan—dengan tujuan mempercepat proses pencarian. Penelitian ini bertujuan untuk membandingkan kinerja kedua algoritma dalam hal kecepatan eksekusi dan efisiensi jalur yang ditemukan. Pengujian dilakukan pada berbagai skenario graf dengan jumlah simpul dan kepadatan berbeda. Hasil penelitian menunjukkan bahwa Algoritma Bidirectional Dijkstra secara konsisten memiliki waktu pencarian yang lebih cepat dibandingkan Algoritma Dijkstra klasik, tanpa mengurangi keakuratan hasil rute. Oleh karena itu, algoritma bidirectional lebih direkomendasikan untuk implementasi pada sistem dengan kebutuhan pemrosesan rute secara real-time dan skala data besar.

Kata Kunci: Algoritma Dijkstra, Bidirectional Dijkstra, Pencarian Rute Terpendek, Graf, Efisiensi Algoritma

PERBANDINGAN ALGORITMA DJIKSTRA DAN ALGORITMA BIDIRECTIONAL DIJKSTRA DALAM PENCARIAN RUTE TERPENDEK

ABSTRAK

Isi Abstrak dalam Bahasa Inggris.
Shortest path searching is a fundamental problem in computer science, particularly in geographic information systems, computer networks, and navigation applications. Dijkstra's algorithm has long been a reliable method for solving this problem. However, in large-scale graphs, the time efficiency of the search process becomes a major concern. The Bidirectional Dijkstra algorithm offers an alternative by performing simultaneous searches from both the start and the goal nodes, aiming to reduce computation time. This study aims to compare the performance of both algorithms in terms of execution speed and pathfinding accuracy. Experiments were conducted on various graph scenarios with different node counts and densities. The results show that the Bidirectional Dijkstra algorithm consistently performs faster than the classic Dijkstra algorithm while maintaining the same level of path accuracy. Therefore, the bidirectional approach is more suitable for real-time systems and applications involving large-scale data.

Keywords: Dijkstra's Algorithm, Shortest Path Search, Graph, Algorithm Efficiency, Two-Way Dijkstra

KATA PENGANTAR

بسم الله الرحمن الرحيم

Puji syukur Alhamdulillah kehadirat Allah Subhanahu Wa Ta'ala yang telah memberikan Nikmat dan Karunia-Nya sehingga penulis dapat menyelesaikan proposal skripsi ini dengan judul "Perbandingan Algoritma Djisktra dan Bellman-ford Dalam Pencarian Rute Terpendek". Proposal ini disusun sebagai salah satu syarat untuk penulisan skripsi Program Studi Teknologi Informasi Fakultas Ilmu Komputer Dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara. Adapun yang perlu disampaikan penyusunan proposal ini tidak terlepas dari bantuan dan dukungan berbagai pihak. Oleh karena itu, penulis ingin menyampaikan terima kasih kepada:

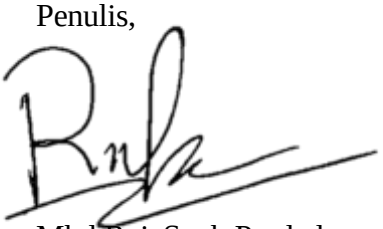
1. Ayahanda Prof. Dr. Agussani, M.AP. selaku Rektor Universitas Muhammadiyah Sumatera Utara.
2. Bapak Alkhowarizmi., S.T., M.Kom selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.
3. Bapak Halim Maulana, S.T., M.Kom selaku Wakil Dekan I Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.
4. Bapak Lutfi Basit, S.Sos., M.I.Kom selaku Wakil Dekan III Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.

5. Ibu Fatma Sari Hutagulung, M.Kom Ketua Prodi Sistem Informasi Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.
6. Bapak Mhd. Basri, S.Si, M.Kom Sekretaris Prodi Sistem Informasi Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara.
7. Bapak Mahardika Prawira Tanjung, S.kom,. M.kom Dosen pembimbing yang telah memberikan arahan dan bimbingan kepada penulis.
8. Bapak dan Ibu dosen Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Muhammadiyah Sumatera Utara yang telah memberikan banyak bekal ilmu kepada penulis.
9. Bapak dan Ibu staff pegawai Biro Fakultas Ilmu Komputer dan Teknologi Informasi.
10. Teristimewa Ayah dan Ibu, Kakak, dan Adik tercinta serta seluruh keluarga besar yang telah banyak memberikan dukungan moril dan materil, serta mendoakan penulis dalam setiap langkah dan usaha dalam menyelesaikan proposal ini.
11. Teman-teman seperjuangan Program Studi Sistem Informasi angkatan 2020 telah membantu dan selalu memberikan support yang tidak bisa penulis sebutkan namanya satu persatu.
12. Dan semua pihak yang telah membantu dan tidak dapat disebut namanya satu persatu.

Penulis menyadari didalam penulisan proposal ini masih banyak

kekurangan. Oleh karena itu, penulis menerima apabila ada kritik dan saran yang sifatnya dapat membantu agar penulisan ini bisa menjadi sempurna. Segala ucapan terima kasih tentunya belum cukup, semoga Allah Subhanahu Wa Ta'ala senantiasa membalas segala kebaikan Anda semua. Aamiin.

Medan, 12 Juni 2024
Penulis,



Mhd RajaSyah Pardede
NPM : 2009020107

DAFTAR ISI

LEMBAR PENGESAHAN---------- ERROR! BOOKMARK NOT DEFINED.
PERNYATAAN ORISINALITAS - ERROR! BOOKMARK NOT DEFINED.
KATA PENGANTAR ----------**I**
DAFTAR ISI---------- **IV**
DAFTAR TABEL ----------**V**
DAFTAR GAMBAR---------- **VI**
1.1 Latar Belakang---------- 1
1.2 Rumusan Masalah ---------- 6
1.3 Batasan Masalah---------- 6
1.4 Tujuan Penellitian ---------- 7
1.5 Manfaat Penelitian---------- 7
1.6 Penelitian Terdahulu ---------- 8
BAB II LANDASAN TEORI----------**11**
2.1 Algoritma Djikstra ----------12
2.2 Algoritma Bidirectional Dijkstra----------16
2.3 Kekurangan dan Kelebihan Algoritma dalam Berbagai Kondisi Graf19
2.4 Perbandingan Kompleksitas Waktu ----------24
2.5 Penerapan Algoritma Dalam Dunia Nyata----------25
2.6 Visual Basic ----------26
2.7 Java ----------28
BAB III METODOLOGI PENELITIAN ----------**32**
3.1. Alur Penelitian ----------32
3.2. Lingkungan Pengembangan----------34
3.3. Metode Pengumpulan Data ----------35
3.4. Teknik Analisis Data----------35
3.5. Validasi Dan Pengujian----------36
BAB 4 RANCANGAN DAN IMPLEMENTASI SISTEM ----------**38**
4.1 Rancangan Sistem----------38
4.1.1 Gambaran Umum Sistem----------38
4.1.2 Arsitektur Sistem ----------38
4.1.3 Perancangan Database ----------38
4.1.4 Diagram Alur Sistem----------40

4.2 Implementasi Sistem----------41
4.2.1 Instalasi dan Konfigurasi ----------41
4.2.2 Implementasi Algoritma Dijkstra ----------41
4.2.3 Implementasi Algoritma Bidirectional ----------42
4.2.4 Rancangan tampilan aplikasi----------42
4.2.7 Halaman Hasil pengujian----------44
4.3 Pengujian dan Evaluasi----------45
4.3.2 Perhitungan Algoritma Bidirectional Dijkstra----------46
4.4.1 Hasil dan Analisis ----------48
4.4 Kesimpulan----------48
BAB 5 KESIMPULAN DAN SARAN----------50
5.1 Kesimpulan----------50
5.2 Saran ----------52
DAFTAR PUSTAKA ----------54

DAFTAR TABEL

Tabel 4.1 tabel hasil perbandingan Algoritma 48

DAFTAR GAMBAR

Gambar 2.1 Flowchart Algoritma Djikstra.....15
Gambar 2.2 Flowchart Algoritma Bidirectional Dijkstra.....19
Gambar 3.1 Flowchart atau Alur Penelitian..... 33
Gambar 4.1 Rancangan database pada system. 39
Gambar 4.2 Data graph pada database system. 39
Gambar 4.3 Data titik lokasi simpul pada Graph database system. 40
Gambar 4.4 Table database user pada system 40
Gambar 4.5 Halaman login user berbasis Laravel.....43
Gambar 4.6 Halaman Dashboard.....44
Gambar 4.7 Halaman proses akhir optimalisasi 44

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era digital yang semakin maju, masalah pencarian rute terpendek telah menjadi fokus utama dalam berbagai aplikasi, mulai dari navigasi GPS hingga jaringan komputer. Algoritma Dijkstra, yang diperkenalkan oleh Edsger W. Dijkstra pada tahun 1956, adalah salah satu algoritma klasik yang sering digunakan untuk menemukan jalur terpendek dalam graf berarah dan berbobot positif (Huang & Wang, 2021). Algoritma ini bekerja dengan cara mengeksplorasi setiap node secara sistematis untuk memastikan bahwa jarak terpendek ke setiap node lainnya ditemukan. Efektivitas dan kesederhanaan algoritma Dijkstra menjadikannya dasar yang kuat dalam teori graf dan aplikasi praktis.

Namun, seiring dengan meningkatnya kompleksitas dan ukuran jaringan, algoritma Dijkstra tradisional dapat menjadi kurang efisien. Hal ini terutama terjadi dalam graf besar di mana waktu komputasi dapat menjadi signifikan. Masalah ini menjadi semakin mendesak untuk diselesaikan karena aplikasi modern seringkali harus memproses data dalam jumlah besar dengan kecepatan tinggi, seperti dalam kasus jaringan transportasi kota besar atau jaringan telekomunikasi global (Patel & Joshi, 2020) . Oleh karena itu, para peneliti dan praktisi terus mencari cara untuk meningkatkan efisiensi algoritma pencarian.

Salah satu solusi yang telah diusulkan adalah Algoritma Bidirectional Dijkstra. Algoritma ini memodifikasi pendekatan Dijkstra klasik dengan melakukan pencarian secara simultan dari titik awal dan titik tujuan, yang dapat mengurangi jumlah node yang perlu dieksplorasi dan dengan demikian

mempercepat waktu komputasi (Jones & Matthews, 2021). Pendekatan ini didasarkan pada prinsip bahwa dua pencarian yang bertemu di tengah-tengah dapat lebih cepat dibandingkan dengan satu pencarian yang dilakukan dari satu sisi saja.

Keunggulan Algoritma Bidirectional Dijkstra terletak pada kemampuannya untuk memecah masalah menjadi dua bagian yang lebih kecil dan lebih mudah dipecahkan. Dengan melakukan pencarian dari dua arah, algoritma ini dapat memanfaatkan informasi dari kedua sisi graf, sehingga mengurangi jumlah total langkah yang diperlukan untuk menemukan jalur terpendek (Smith & Chen, 2021). Dalam banyak kasus, pendekatan ini telah terbukti lebih efisien daripada algoritma Dijkstra tradisional, terutama dalam graf yang sangat besar.

Meskipun Algoritma Bidirectional Dijkstra menawarkan banyak keunggulan, ada beberapa situasi di mana pendekatan ini mungkin tidak selalu lebih baik. Misalnya, dalam graf yang sangat tidak seimbang atau dalam jaringan dengan kepadatan yang tinggi, keuntungan dari pencarian dua arah mungkin tidak sebesar yang diharapkan (Huang & Wang, 2021). Oleh karena itu, penting untuk melakukan analisis yang cermat terhadap karakteristik graf yang akan digunakan sebelum memilih algoritma yang paling sesuai.

Penelitian ini bertujuan untuk membandingkan performa kedua algoritma tersebut dalam konteks pencarian rute terpendek dengan berbagai jenis graf yaitu graf padat, graf jarak dan graf bobot negatif. Perbandingan ini penting untuk menentukan situasi di mana salah satu algoritma lebih unggul dari yang lain, serta untuk memahami trade-off yang terlibat dalam penggunaan masing-masing algoritma (Ding & Qin, 2021). Selain itu, penelitian ini akan membantu dalam

memberikan rekomendasi yang lebih baik mengenai algoritma mana yang lebih efisien digunakan dalam berbagai aplikasi praktis, seperti sistem navigasi dan perencanaan rute dalam jaringan transportasi. graf salah satu cabang matematika yang mempelajari struktur diskret yang terdiri atas simpul (node atau vertex) dan sisi (edge) yang menghubungkan simpul-simpul tersebut. Teori graf digunakan dalam berbagai bidang seperti ilmu komputer, teknik jaringan, logistik, dan ilmu sosial (*Bondy, J. A., & Murty, U. S. R. 2021).* Graf dapat dikelompokkan menjadi beberapa kategori atau jenis bergantung pada sudut pandang pengelompokkannya. Pengelompokkan graf dapat dipandang berdasarkan ada tidaknya sisi ganda atau sisi gelang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi. Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis:

1. Graf sederhana

Graf yang tidak mengandung gelang maupun sisi-ganda dinamakan graf sederhana. $G1$ adalah contoh graf sederhana yang merepresentasikan jaringan komputer. Simpul menyatakan komputer, sedangkan sisi menyatakan saluran telepon untuk berkomunikasi. Saluran telepon dapat beroperasi pada dua arah.

2. Graf tak-sederhana

Graf yang mengandung sisi ganda atau gelang dinamakan graf tak-sederhana. Ada dua macam graf tak sederhana, yaitu graf ganda dan graf semu. Graf ganda adalah graf yang mengandung sisi ganda. Sisi ganda yang menghubungkan sepasang simpul bisa lebih dari dua buah. $G2$ adalah graf ganda. Sisi ganda dapat diasosiasikan sebagai pasangan tak- terurut yang sama. Kita juga dapat mendefinisikan graf ganda G = (V(G), E(G)) terdiri dari himpunan tidak

kosong simpul-simpul dan E adalah himpunan-ganda yang mengandung sisi ganda. Pada jaringan telekomunikasi, sisi ganda pada $G2$ dapat diandaikan sebagai saluran telepon tambahan apabila beban komunikasi data antar komputer sangat padat. Perhatikanlah bahwa setiap graf sederhana juga merupakan graf ganda, tetapi tidak setiap graf ganda merupakan graf sederhana.

Graf semu adalah graf yang mengandung gelang. $G3$ adalah graf semu (termasuk bila memiliki graf ganda sekalipun). Sisi gelang pada $G3$ dapat dianggap sebagai saluran telepon tambahan yang menghubungkan komputer dengan dirinya sendiri. Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya sendiri. Jumlah simpul pada graf kita sebut sebagai kardinalitas graf, dan dinyatakan dengan n = |V(G)| , dan jumlah sisi kita nyatakan dengan m = |E(G)| . Pada contoh diatas, $G1$ mempunyai n = 4, dan m = 4, sedangkan $G2$ mempunyai n= 3 dan m=4.

Dalam penelitian ini, akan dilakukan serangkaian eksperimen dengan menggunakan data yang berbeda-beda untuk menguji performa kedua algoritma. Parameter yang akan diukur meliputi waktu komputasi, jumlah node yang dieksplorasi, dan akurasi hasil pencarian rute terpendek. Hasil dari eksperimen ini diharapkan dapat memberikan wawasan yang lebih mendalam mengenai kekuatan dan kelemahan masing-masing algoritma, serta memberikan panduan bagi pengembang aplikasi dalam memilih algoritma yang paling sesuai untuk kebutuhan mereka.

Pada penelitian ini akurasi hasil pencarian rute terpendek diuji dengan berbagai jenis graf yaitu graf padat, graf pendek dan graf bobot negatif. Graf padat adalah jenis graf yang memiliki jumlah tepi yang mendekati jumlah

maksimum yang mungkin, yaitu ketika hampir setiap simpul dihubungkan langsung dengan setiap simpul lainnya. Dalam konteks pencarian rute terpendek, seperti yang diterapkan dalam algoritma Dijkstra, graf padat dapat menyebabkan peningkatan kompleksitas waktu, mendekati $O(V^2)$ di mana V adalah jumlah simpul dalam graf. Ini karena algoritma harus memproses sejumlah besar tepi, yang dapat membuatnya kurang efisien, terutama dalam graf besar. Sebaliknya, graf jarang adalah jenis graf di mana jumlah tepi jauh lebih sedikit dibandingkan dengan jumlah maksimum yang mungkin. Dalam graf jarang, banyak simpul yang tidak terhubung langsung satu sama lain, sehingga jumlah tepi E biasanya jauh lebih kecil dari V^2. Algoritma seperti Dijkstra dapat beroperasi lebih efisien dalam graf jarang karena harus memproses lebih sedikit tepi, sehingga waktu komputasi bisa lebih cepat dan sumber daya yang digunakan lebih sedikit dibandingkan dengan graf padat. Bobot negatif dalam sebuah graf merujuk pada tepi yang memiliki nilai bobot negatif, yang dapat mewakili biaya atau jarak yang menguntungkan untuk dilalui. Namun, algoritma Dijkstra tidak dapat menangani bobot negatif secara efektif karena algoritma ini mengasumsikan bahwa biaya rute hanya bisa bertambah. Penggunaan bobot negatif dalam graf membutuhkan algoritma yang berbeda, seperti Bellman-Ford, yang dirancang untuk mengatasi kasus dengan bobot negatif tanpa risiko perhitungan yang salah atau tidak optimal.

Secara keseluruhan, penelitian ini tidak hanya akan memberikan kontribusi terhadap pemahaman teoritis tentang algoritma pencarian rute terpendek, tetapi juga akan memiliki implikasi praktis yang signifikan. Dengan memahami kondisi di mana masing-masing algoritma lebih unggul, pengembang dan peneliti dapat

membuat keputusan yang lebih informasional dalam merancang dan mengimplementasikan sistem pencarian rute yang lebih efisien dan efektif.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah tersebut, maka penulis merumuskan permasalahan tersebut yaitu :

1. Bagaimana hasil akurasi algoritma Djikstra dibandingkan dengan algoritma Bidirectional Dijkstra dalam menemukan waktu komputasi, jumlah node yang dieksplorasi, dan akurasi hasil pencarian rute terpendek pada graf dengan bobot negatif, graf padat dan graf jarang?
2. Sejauh mana algoritma Bidirectional Dijkstra lebih unggul dalam menangani pencarian rute terpendek dibandingkan algoritma Djikstra?
3. Apa saja kekurangan dan kelebihan masing-masing algoritma dalam berbagai kondisi graf yang berbeda (misalnya graf padat, graf jarang, graf dengan bobot negatif)?
4. Bagaimana efektivitas implementasi algoritma Dijkstra dan Bidirectional Dijkstra dalam aplikasi desktop?

1.3 Batasan Masalah

Batasan masalah yang digunakan dalam sebuah pembahasan bertujuan agar dalam pembahasannya lebih terarah dan sesuai dengan tujuan yang akan dicapai. Adapun batasan masalah dalam penulisan proposal ini adalah :

1. Penelitian ini hanya akan membandingkan dua algoritma, yaitu Djikstra dan Bidirectional Dijkstra dengan parameter yaitu waktu komputasi, jumlah node yang dieksplorasi, dan akurasi hasil pencarian rute terpendek pada graf padat, graf jarang dan graf dengan bobot

negatif. Algoritma lainnya untuk pencarian rute terdekat tidak akan dibahas dalam penelitian ini.

2. Implementasi dan pengujian algoritma akan dilakukan ke aplikasi desktop menggunakan Visual Basic.
3. Penyimpanan data graf dan hasil pengujian akan menggunakan MySQL sebagai sistem manajemen basis data.

1.4 Tujuan Penellitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. Mengevaluasi dan membandingkan kinerja kedua algoritma dalam mencari rute terdekat pada graf padat, jarang dan bobot negatif.
2. Mengidentifikasi kelebihan dan kelemahan dari algoritma Djikstra dan Bidirectional Dijkstra dalam berbagai kondisi graf, termasuk graf dengan bobot non-negatif dan bobot negatif.
3. Menilai efisiensi masing-masing algoritma dalam menangani graf dengan ukuran dan kompleksitas yang berbeda-beda, untuk memahami bagaimana skala graf mempengaruhi performa algoritma.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini, diharapkan akan memberikan dampak yang positif bagi semua pihak yang berkaitan diantaranya sebagai berikut :

1. Manfaat Akademis: Menambah wawasan dan pengetahuan mengenai algoritma Djikstra dan Bidirectional Dijkstra, khususnya dalam konteks pencarian rute terdekat pada graf dengan berbagai kondisi.
2. Manfaat Praktis: Memberikan panduan praktis bagi pengembang perangkat lunak dan peneliti dalam memilih dan menerapkan

algoritma pencarian rute terdekat yang paling efisien dan efektif untuk kebutuhan spesifik mereka. Serta Menghasilkan aplikasi desktop yang dapat digunakan sebagai alat bantu untuk memahami dan menguji kinerja algoritma Djikstra dan Bidirectional Dijkstra dalam berbagai kondisi graf.

3. Manfaat Teknologi: Membantu dalam pengembangan dan optimasi jaringan komputer, sistem navigasi, dan aplikasi terkait lainnya dengan menyediakan analisis kinerja yang mendetail dari kedua algoritma.

1.6 Penelitian Terdahulu

1. Penelitian oleh Goldberg dan Harrelson (2005) dalam jurnal SIAM Journal on Computing yang berjudul "Perbandingan Kinerja Algoritma Dijkstra dan Bidirectional Dijkstra pada Graf Besar" membandingkan kinerja algoritma Dijkstra dan Bidirectional Dijkstra pada graf besar dengan berbagai densitas dan distribusi bobot. Hasil penelitian menunjukkan bahwa Bidirectional Dijkstra secara signifikan lebih cepat daripada Dijkstra standar pada graf besar, terutama yang jarang. Algoritma Bidirectional Dijkstra mengurangi jumlah simpul yang perlu dieksplorasi dengan melakukan pencarian dari dua arah, yang mengarah pada peningkatan efisiensi.
2. Bauer dan Delling (2009) dalam Journal of Experimental Algorithmics yang berjudul "Efisiensi Algoritma Dijkstra dalam Jaringan Transportasi" mengevaluasi efisiensi algoritma Dijkstra dalam konteks jaringan transportasi kota yang kompleks. Penelitian ini membandingkan versi

tradisional dengan variasi bidirectional dan menemukan bahwa bidirectional lebih unggul dalam jaringan dengan banyak simpul. Algoritma Bidirectional Dijkstra menunjukkan performa yang lebih baik dalam jaringan transportasi yang memiliki banyak simpul dan rute alternatif.

3. Penelitian oleh Pohl (1971) yang dipublikasikan dalam Artificial Intelligence yang berjudul "Analisis Algoritma Pencarian Jalur Terpendek: Dijkstra vs. A*" memfokuskan pada perbandingan antara algoritma Dijkstra dan algoritma A* dalam konteks pencarian jalur terpendek. Meskipun fokus utama adalah pada A*, penelitian ini memberikan wawasan penting tentang kelemahan Dijkstra dalam hal efisiensi waktu, terutama ketika digunakan dalam graf besar tanpa heuristik tambahan. A* dengan heuristik yang tepat menunjukkan performa yang lebih baik dibandingkan Dijkstra standar.
4. Studi oleh Geisberger, Sanders, Schultes, dan Delling (2008) yang dipublikasikan dalam ACM Journal on Experimental Algorithmics berjudul "Optimisasi Algoritma Pencarian Jalur Terpendek untuk Jaringan Jalan Raya" membahas berbagai optimisasi untuk algoritma pencarian jalur terpendek dalam jaringan jalan raya, termasuk Dijkstra dan variasi bidirectional. Penelitian ini menunjukkan bahwa algoritma Bidirectional Dijkstra, terutama ketika dikombinasikan dengan teknik pre-processing seperti contraction hierarchies, dapat secara dramatis mengurangi waktu pencarian jalur pada jaringan jalan raya yang besar.

5. Penelitian oleh Chabini dan Lan (2002) yang dipublikasikan dalam Transportation Science yang berjudul “Implementasi dan Analisis Algoritma Bidirectional Dijkstra dalam Sistem Navigasi Real-time” mengevaluasi penggunaan algoritma Bidirectional Dijkstra dalam sistem navigasi real-time. Hasil penelitian menunjukkan bahwa algoritma bidirectional tidak hanya mengurangi waktu komputasi secara signifikan dibandingkan Dijkstra standar, tetapi juga lebih efisien dalam penggunaan memori, yang sangat penting untuk aplikasi real-time dalam sistem navigasi modern.

BAB II

LANDASAN TEORI

Bab ini memberikan wawasan mendalam tentang berbagai teori dasar yang relevan dengan penelitian mengenai "Perbandingan Algoritma Djikstra dan Bidirectional Dijkstra dalam Mencari Jarak Terdekat". Landasan teori yang diuraikan mencakup konsep dasar tentang algoritma pencarian jalur terdekat, yang merupakan kerangka kerja yang penting dalam pengembangan dan optimasi sistem jaringan dan graf. Algoritma pencarian jalur terdekat adalah struktur algoritmis yang dirancang untuk menemukan rute terpendek dalam sebuah graf, termasuk berbagai komponen seperti simpul, sisi, bobot, dan prosedur-prosedur untuk mengumpulkan, memanipulasi, menyimpan, dan menyebarkan informasi rute yang relevan bagi pengguna. Pemahaman yang kuat tentang algoritma ini sangat penting dalam merancang sistem pencarian rute yang efisien dan dapat diandalkan.

Di samping itu, bab ini juga mencakup studi terdahulu yang relevan dengan penelitian ini. Melalui tinjauan literatur yang komprehensif, penelitian sebelumnya tentang algoritma pencarian jalur terdekat, khususnya Djikstra dan Bidirectional Dijkstra, diidentifikasi dan dianalisis. Dengan memahami temuan-temuan dari studi-studi sebelumnya, penelitian ini dapat memperoleh wawasan yang berharga dan menemukan kesempatan-kesempatan untuk meningkatkan pendekatan perbandingan algoritma pencarian rute terdekat. Studi ini juga memberikan landasan yang kuat untuk memahami keunggulan, kelemahan, dan aplikasi terbaik dari masing-masing algoritma dalam berbagai kondisi graf.

Dengan analisis yang mendalam dan tinjauan literatur yang komprehensif,

penelitian ini bertujuan untuk memberikan pemahaman yang lebih baik tentang kapan dan bagaimana algoritma Djikstra dan Bidirectional Dijkstra dapat diterapkan secara optimal. Ini termasuk pengujian dalam skenario yang berbeda untuk mengukur performa kedua algoritma dalam hal waktu eksekusi, penggunaan memori, dan kemampuan untuk menangani bobot negatif. Melalui pendekatan ini, diharapkan penelitian ini dapat memberikan kontribusi signifikan dalam bidang optimasi jaringan dan pengelolaan graf, serta aplikasi praktis dalam sistem navigasi, routing jaringan komputer, dan manajemen logistik.

2.1 Algoritma Djikstra

Algoritma Djikstra adalah salah satu algoritma paling terkenal dan sering digunakan untuk menemukan jalur terpendek dari satu simpul ke simpul lainnya dalam sebuah graf berarah dengan bobot non-negatif. Algoritma ini pertama kali diperkenalkan oleh Edsger W. Dijkstra pada tahun 1959. Prinsip dasar dari algoritma ini adalah menggunakan metode greedy, di mana pada setiap langkahnya, ia selalu memilih jalur yang paling pendek yang tersedia untuk diperluas. Hal ini dilakukan hingga semua simpul dalam graf telah dikunjungi atau jalur terpendek ke simpul tujuan telah ditemukan. Keunggulan utama dari algoritma Djikstra adalah kesederhanaannya dan efisiensinya dalam hal waktu eksekusi untuk graf dengan bobot non-negatif.

Algoritma Djikstra memiliki kompleksitas waktu yang berbeda tergantung pada cara implementasinya. Jika diimplementasikan menggunakan matriks ketetanggaan, kompleksitas waktunya adalah O(V^2), di mana V adalah jumlah simpul dalam graf. Namun, jika diimplementasikan menggunakan tumpukan prioritas (heap), kompleksitas waktu dapat ditingkatkan menjadi O(E + V log V),

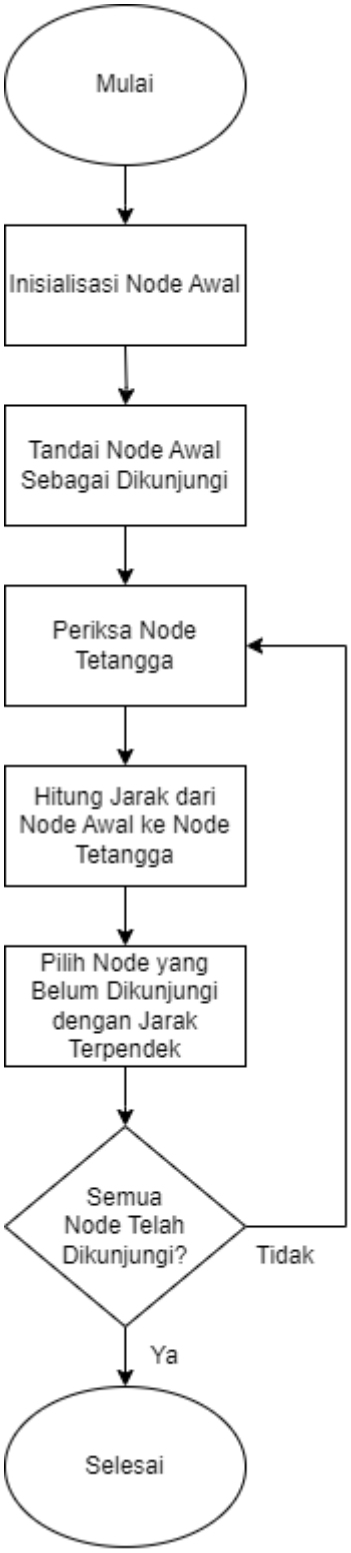
di mana E adalah jumlah sisi dalam graf. Kompleksitas ruang dari algoritma ini adalah O(V), karena memerlukan penyimpanan untuk simpul yang dikunjungi dan jarak terpendek yang dihitung. Kombinasi dari efisiensi waktu dan penggunaan ruang yang moderat membuat algoritma ini sangat cocok untuk aplikasi yang membutuhkan hasil cepat dan akurat dalam graf dengan bobot non-negatif.

Implementasi algoritma Djikstra biasanya melibatkan beberapa langkah utama. Pertama, semua jarak dari simpul sumber ke simpul lainnya diinisialisasi dengan nilai tak hingga, kecuali jarak ke simpul sumber itu sendiri yang diinisialisasi dengan nol. Kedua, algoritma memilih simpul yang memiliki jarak terpendek yang belum dikunjungi, lalu memperbarui jarak untuk semua tetangga simpul tersebut. Proses ini disebut sebagai relaksasi. Langkah ini diulangi hingga semua simpul telah dikunjungi atau jalur terpendek ke tujuan telah ditemukan. Teknik ini memastikan bahwa jarak terpendek ditemukan dengan cara yang efisien dan sistematis.

Algoritma Djikstra memiliki berbagai aplikasi praktis dalam dunia nyata. Salah satu aplikasi yang paling umum adalah dalam sistem navigasi GPS, di mana algoritma ini digunakan untuk menemukan rute terpendek dari satu lokasi ke lokasi lainnya. Selain itu, algoritma ini juga digunakan dalam routing jaringan komputer untuk menemukan jalur optimal bagi data untuk ditransfer dari satu titik ke titik lainnya. Di bidang transportasi, algoritma Djikstra dapat membantu dalam perencanaan rute dan penjadwalan, serta dalam optimasi logistik dan manajemen rantai pasok. Kemampuan algoritma ini untuk menemukan jalur terpendek dengan cepat dan akurat membuatnya sangat berguna dalam berbagai konteks operasional.

Meskipun algoritma Djikstra sangat efisien dalam graf dengan bobot non-negatif, ia memiliki beberapa keterbatasan. Algoritma ini tidak dapat digunakan dalam graf yang memiliki bobot negatif karena dapat menghasilkan hasil yang tidak akurat. Selain itu, efisiensi waktu dari algoritma ini menurun seiring dengan peningkatan jumlah simpul dan sisi dalam graf yang sangat besar. Namun, kelebihannya dalam hal kecepatan dan kesederhanaan implementasi menjadikannya pilihan yang populer untuk banyak aplikasi yang memerlukan perhitungan jalur terpendek. Untuk aplikasi di mana bobot negatif tidak menjadi masalah, algoritma Djikstra tetap menjadi salah satu alat yang paling andal dan efektif. Algoritma Djikstra diimplementasikan dalam aplikasi desktop dengan langkah-langkah berikut:

1. Inisialisasi jarak dari simpul sumber ke semua simpul lainnya dengan nilai tak hingga, kecuali jarak ke simpul sumber itu sendiri yang diinisialisasi dengan nol.
2. Memilih simpul dengan jarak terpendek yang belum dikunjungi dan memperbarui jarak untuk semua tetangganya melalui proses relaksasi.
3. Mengulangi langkah 2 hingga semua simpul telah dikunjungi atau jarak terpendek ke tujuan telah ditemukan.



Gambar 2.1 Flowchart atau Agoritma Djikstra

2.2 Algoritma Bidirectional Dijkstra

Bidirectional Dijkstra adalah sebuah algoritma yang digunakan untuk mencari jalur terpendek antara dua titik dalam sebuah graf. Algoritma ini merupakan perluasan dari algoritma Dijkstra unidirectional, yang hanya mencari jalur terpendek dari satu titik ke semua titik lainnya dalam graf. Algoritma ini diperkenalkan oleh Edsger W. Dijkstra. pada tahun 1959. Berbeda dengan algoritma Djikstra yang menggunakan metode greedy, Bidirectional Dijkstra menggunakan dua proses pencarian Dijkstra yang berjalan secara bersamaan, satu dari titik awal dan yang lainnya dari titik tujuan. Kedua proses pencarian ini akan berhenti ketika mereka bertemu di tengah jalan, menemukan jalur terpendek antara titik awal dan tujuan, sesuatu yang tidak dapat dilakukan oleh algoritma Djikstra.

Kompleksitas waktu dari algoritma Bidirectional Dijkstra adalah O(b * m), di mana b adalah jumlah bidang pencarian (biasanya 2), dan m adalah jumlah edge (sisi) dalam graf. Kompleksitas waktu ini lebih tinggi dibandingkan dengan algoritma Djikstra, terutama untuk graf dengan jumlah sisi yang besar. Algoritma ini memerlukan penyimpanan yang sama dengan Djikstra, yaitu O(V) untuk menyimpan jarak terpendek dan simpul yang telah dikunjungi. Meskipun lebih lambat, kemampuan Bidirectional Dijkstra untuk menangani bobot negatif membuatnya lebih fleksibel dalam aplikasi tertentu di mana bobot negatif mungkin muncul.

Implementasi algoritma Bidirectional Dijkstra melibatkan beberapa langkah utama. Pertama, semua jarak dari simpul sumber ke simpul lainnya diinisialisasi dengan nilai tak hingga, kecuali jarak ke simpul sumber itu sendiri

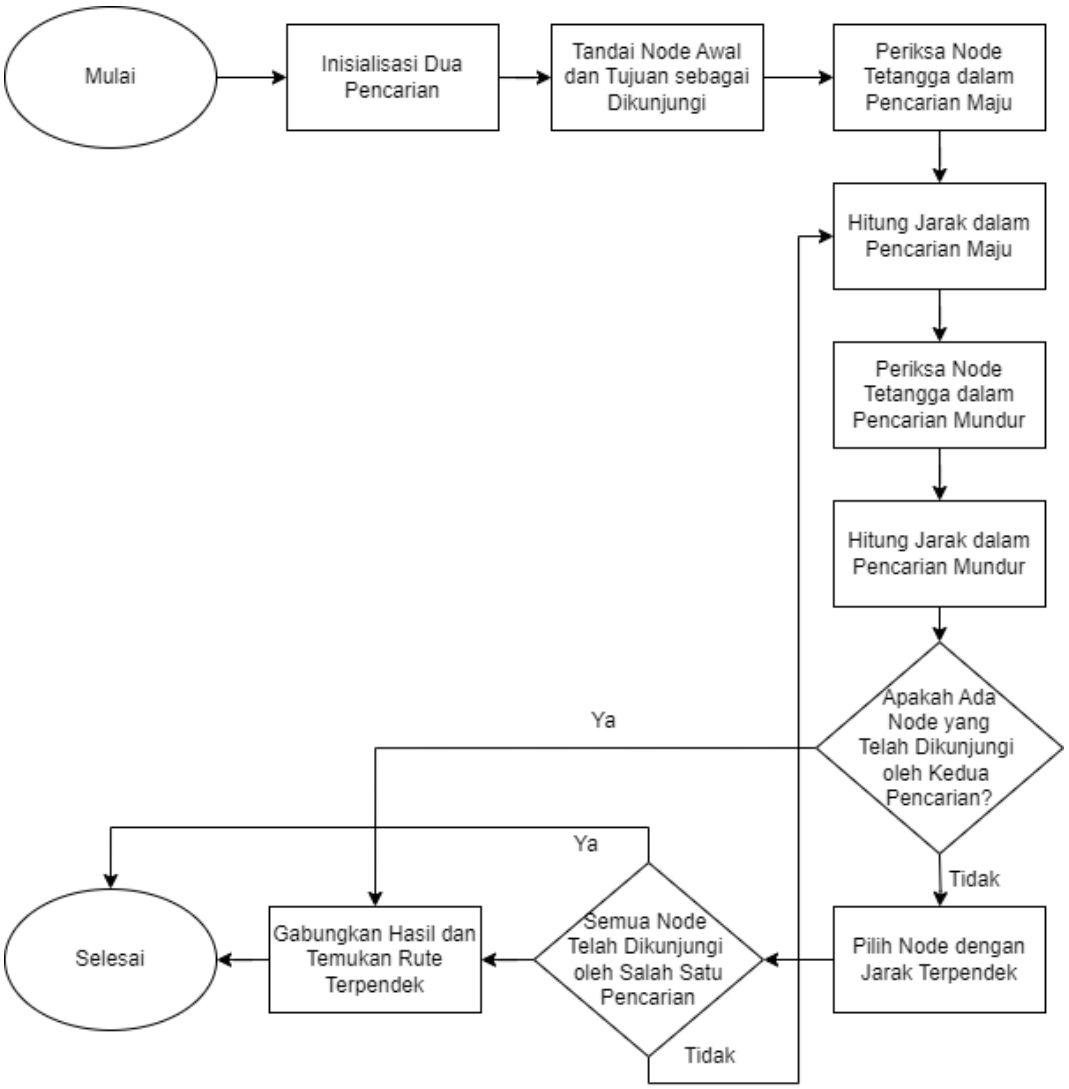
yang diinisialisasi dengan nol. Kedua, algoritma melakukan relaksasi pada setiap sisi graf sebanyak (V-1) kali, di mana V adalah jumlah simpul. Relaksasi melibatkan pembaruan jarak terpendek jika ditemukan jalur a/b yang lebih pendek melalui suatu simpul. Langkah terakhir adalah memeriksa keberadaan siklus negatif a/b dengan melakukan satu kali relaksasi tambahan; jika ada pembaruan jarak yang terjadi, maka graf tersebut mengandung siklus negatif.

Algoritma Bidirectional Dijkstra memiliki banyak aplikasi yang luas dalam dunia nyata, terutama dalam situasi di mana bobot negatif mungkin ada. Salah satu aplikasi utamanya adalah dalam Rute Navigasi komputer, terutama dalam game komputer atau simulasi yang melibatkan peta besar atau labirin (Grafik dan Game). Selain itu, algoritma ini sering digunakan untuk mencari rute terpendek antara dua lokasi. Bidirectional Dijkstra bisa lebih efisien dibandingkan Dijkstra biasa karena bekerja dari dua arah—dari titik awal dan titik tujuan—sehingga mempercepat pencarian rute optimal atau juga algoritma ini bisa digunakan untuk menghitung jalur tercepat antara dua titik untuk karakter atau objek.

Kelebihan utama dari algoritma Bidirectional Dijkstra adalah kemampuannya untuk menangani graf dengan bobot negatif dan mendeteksi siklus negatif, membuatnya lebih fleksibel dibandingkan algoritma Djikstra. Namun, algoritma ini memiliki efisiensinya dalam mengurangi waktu komputasi saat mencari jalur terpendek antara dua titik dalam graf besar. Dengan menjalankan pencarian dari kedua titik (awal dan tujuan) secara bersamaan, algoritma ini dapat mengurangi waktu yang dibutuhkan untuk menemukan jalur terpendek. Meskipun demikian, dalam situasi di mana bobot negatif tidak dapat dihindari, algoritma Bidirectional

Dijkstra tetap menjadi alat yang sangat berguna dan relevan untuk digunakan. Algoritma Bidirectional Dijkstra diimplementasikan dengan langkah-langkah berikut:

1. Inisialisasi dan Pencarian Paralel: Algoritma memulai dua pencarian secara paralel dari node awal dan node tujuan. Node-node ini ditandai sebagai dikunjungi, dan pencarian maju serta mundur dilakukan dengan memeriksa dan menghitung jarak ke node-node tetangga.
2. Pertemuan Pencarian dan Pemilihan Node: Selama pencarian, algoritma memeriksa apakah ada node yang dikunjungi oleh kedua pencarian. Jika ya, hasil pencarian digabungkan; jika tidak, algoritma memilih node dengan jarak terpendek untuk dilanjutkan.
3. Gabungan Hasil dan Penyelesaian: Setelah menemukan pertemuan atau memastikan semua node telah dikunjungi, algoritma menggabungkan hasil kedua pencarian untuk menemukan rute terpendek dan kemudian menyelesaikan proses.



Gambar 2.2 Flowchart atau Agoritma Bidirectional Dijkstra

2.3 Kekurangan dan Kelebihan Algoritma dalam Berbagai Kondisi Graf

1. Graf padat memiliki jumlah tepi mendekati jumlah maksimum yang mungkin, algoritma Dijkstra bisa menjadi kurang efisien karena harus memproses banyak tepi. Kelebihan algoritma Dijkstra dalam graf padat adalah kesederhanaan implementasinya dan kemampuan untuk menemukan rute terpendek dengan akurasi tinggi. Namun, kelemahannya terletak pada waktu eksekusi yang bisa sangat tinggi, terutama karena setiap simpul perlu memeriksa semua tetangganya, yang menyebabkan kompleksitas waktu menjadi O(V^2) atau O(E log V) tergantung pada implementasi heap yang digunakan (Cormen et al., 2009).

Algoritma Bidirectional Dijkstra juga menghadapi tantangan yang sama pada graf padat. Namun, karena algoritma ini menjalankan pencarian dari dua arah sekaligus, ia memiliki potensi untuk mengurangi jumlah simpul yang perlu dieksplorasi secara signifikan. Dalam praktiknya, Bidirectional Dijkstra sering kali lebih cepat dibandingkan dengan Dijkstra biasa karena kedua pencarian bertemu di tengah graf, yang mengurangi jumlah total simpul dan tepi yang perlu diproses (Goldberg & Harrelson, 2005). Meski demikian, manajemen dua pencarian simultan bisa lebih rumit dari segi implementasi dan memori.

2. Graf Jarang memiliki jumlah yang tepi jauh lebih sedikit dibandingkan dengan jumlah maksimum yang mungkin, algoritma Dijkstra menunjukkan performa yang lebih baik. Kelebihan utamanya adalah efisiensi dalam memproses simpul dan tepi, karena waktu eksekusinya bisa mendekati O(E log V) dengan penggunaan heap. Graf jarang memungkinkan algoritma Dijkstra untuk bekerja lebih cepat karena lebih sedikit tepi yang perlu dipertimbangkan pada setiap langkahnya (Johnson, 1977). Namun, dalam graf jarang yang sangat besar, waktu eksekusi tetap bisa menjadi masalah jika graf tersebut memiliki banyak simpul. Bidirectional Dijkstra cenderung sangat efektif pada graf jarang karena pencarian dari dua arah biasanya dapat dengan cepat mengurangi jumlah simpul yang perlu diproses. Hal ini sangat membantu dalam mengurangi waktu eksekusi total. Karena jumlah tepi yang sedikit, keunggulan dari pertemuan di tengah graf lebih terasa dibandingkan pada graf padat. Namun, seperti pada graf padat, pengelolaan dua pencarian simultan tetap menjadi tantangan, terutama dalam hal sinkronisasi dan penyimpanan data (Goldberg & Harrelson, 2005).

3. Graf dengan Bobot Negatif Algoritma Dijkstra tidak dapat digunakan pada graf dengan bobot tepi negatif karena asumsi dasar dari algoritma ini adalah bahwa jarak terpendek selalu bertambah positif seiring dengan penambahan tepi baru. Jika terdapat tepi dengan bobot negatif, algoritma ini bisa menghasilkan hasil yang tidak valid atau bahkan masuk dalam loop tak berujung (Cormen et al., 2009). Oleh karena itu, algoritma lain seperti Bellman-Ford lebih cocok untuk graf dengan bobot negatif. Bidirectional Dijkstra juga tidak cocok untuk graf dengan bobot negatif karena mewarisi kelemahan yang sama dari algoritma Dijkstra dasar. Algoritma ini juga akan gagal jika ada tepi dengan bobot negatif karena pencarian dari kedua arah tidak bisa menjamin bahwa jarak terpendek yang ditemukan benar. Seperti pada Dijkstra biasa, untuk graf dengan bobot negatif, perlu menggunakan algoritma yang dirancang khusus untuk menangani bobot negatif, seperti Bellman-Ford atau algoritma Floyd-Warshall untuk kasus tertentu (Johnson, 1977).

Pencarian pada graph merupakan konsep fundamental dalam ilmu komputer yang memungkinkan kita untuk menemukan informasi atau jalur yang relevan dalam struktur data yang kompleks. Dua algoritma pencarian yang paling umum digunakan dalam konteks graph adalah Breadth-First Search (BFS) dan Depth-First Search (DFS). Kedua algoritma ini memiliki karakteristik dan aplikasi yang berbeda, serta kelebihan dan kekurangan masing-masing.

Breadth-First Search (BFS) adalah algoritma traversing yang mengeksplorasi simpul-simpul terdekat terlebih dahulu sebelum

melanjutkan ke simpul yang lebih jauh. BFS menggunakan antrian (queue) untuk melacak simpul-simpul yang perlu dijelajahi. Proses ini memastikan bahwa nodenode yang lebih dekat ke node awal (source) akan dikunjungi terlebih dahulu. BFS sangat efektif untuk menemukan jalur terpendek dalam graph yang tidak berbobot, atau dalam konteks graf berberat di mana semua bobotnya sama. Keunggulan lain dari BFS adalah kemampuannya untuk menjamin bahwa semua node pada level yang sama akan diproses sebelum melanjutkan ke level berikutnya, sehingga cocok untuk pemecahan masalah yang memerlukan eksplorasi secara bertahap. Algoritma Depth First Search adalah algoritma pencarian mendalam yang dimulai dari node awal dilanjutkan dengan hanya mengunjungi node anak paling kiri pada tingkat selanjutnya. Cara kerja algoritma Depth First Search yaitu masukan masukan node akar kedalam sebuah tumpukan. Kemudian ambil simpul pertama pada level paling atas, jika simpul merupakan solusi pencarian selesai dan hasil dikembalikan. Jika simpul bukan merupakan solusi, masukan seluruh simpul yang bertetangga dengan simpul tersebut ke dalam tumpukan. Apabila semua simpul sudah dicek dan antrean kosong, pencarian selesai dengan mengembalikan hasil solusi tidak ditemukan. Pencarian diulang dari simpul awal antrean.

Graf dapat dikelompokkan menjadi beberapa kategori atau jenis bergantung pada sudut pandang pengelompokkannya. Pengelompokkan graf dapat dipandang berdasarkan ada tidaknya sisi ganda atau sisi gelang, berdasarkan jumlah simpul, atau berdasarkan orientasi arah pada sisi. Berdasarkan ada tidaknya gelang atau sisi

ganda pada suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis:

1. Graf sederhana

   Graf yang tidak mengandung gelang maupun sisi-ganda dinamakan graf sederhana. G_1 adalah contoh graf sederhana yang merepresentasikan jaringan komputer. Simpul menyatakan komputer, sedangkan sisi menyatakan saluran telepon untuk berkomunikasi. Saluran telepon dapat beroperasi pada dua arah.

2. Graf tak-sederhana

   Graf yang mengandung sisi ganda atau gelang dinamakan graf tak- sederhana. Ada dua macam graf tak sederhana, yaitu *graf ganda* dan *graf semu*. Graf ganda adalah graf yang mengandung sisi ganda. Sisi ganda yang menghubungkan sepasang simpul bisa lebih dari dua buah. G_2 adalah graf ganda. Sisi ganda dapat diasosiasikan sebagai pasangan tak- terurut yang sama.

   Kita juga dapat mendefinisikan graf ganda $G = (V(G), E(G))$ terdiri dari himpunan tidak kosong simpul-simpul dan E adalah himpunan-ganda yang mengandung sisi ganda. Pada jaringan telekomunikasi, sisi ganda pada G_2 dapat diandaikan sebagai saluran telepon tambahan apabila beban komunikasi data antar komputer sangat padat. Perhatikanlah bahwa setiap graf sederhana juga merupakan graf ganda, tetapi tidak setiap graf ganda merupakan graf sederhana.

   Graf semu adalah graf yang mengandung gelang. G_3 adalah

graf semu (termasuk bila memiliki graf ganda sekalipun). Sisi gelang pada G_3 dapat dianggap sebagai saluran telepon tambahan yang menghubungkan komputer dengan dirinya sendiri. Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya sendiri. Jumlah simpul pada graf kita sebut sebagai kardinalitas graf, dan dinyatakan dengan $n = |V(G)|$, dan jumlah sisi kita nyatakan dengan $m = |E(G)|$. Pada contoh diatas, G_1 mempunyai $n = 4$, dan $m = 4$, sedangkan G_2 mempunyai $n= 3$ dan $m=4$.

2.4 Perbandingan Kompleksitas Waktu

2.4.1 Kompleksitas Waktu Algoritma Dijkstra

Kompleksitas waktu dari algoritma Dijkstra bergantung pada struktur data yang digunakan untuk mengelola antrian prioritas. Dengan menggunakan heap biner, kompleksitas waktu adalah O((V + E) log V), di mana V adalah jumlah simpul dan E adalah jumlah tepi (Cormen et al., 2009). Dalam graf padat, kompleksitas ini mendekati O(V^2) karena E bisa mendekati V^2. Sebaliknya, dalam graf jarang, di mana E jauh lebih kecil dari V^2, kompleksitas bisa lebih efisien, mendekati O(V log V).

2.4.2 Kompleksitas Waktu Algoritma Bidirectional Dijkstra

Algoritma Bidirectional Dijkstra memiliki potensi untuk mengurangi waktu eksekusi secara signifikan dibandingkan dengan Dijkstra biasa. Secara teori, kompleksitas waktu dari Bidirectional Dijkstra adalah sama dengan dua kali kompleksitas Dijkstra, yaitu O(2 * (V + E) log V). Namun, dalam praktiknya, karena pencarian dilakukan dari dua arah yang saling mendekati, jumlah total

simpul dan tepi yang diproses biasanya lebih sedikit daripada dalam satu pencarian penuh, yang berarti waktu eksekusi efektif sering kali lebih rendah dari Dijkstra biasa (Goldberg & Harrelson, 2005).

2.4.3 Pengaruh Ukuran Graf

Dalam graf besar dengan jumlah simpul dan tepi yang sangat banyak, perbedaan kompleksitas waktu antara kedua algoritma menjadi lebih menonjol. Dijkstra biasa mungkin mengalami penurunan kinerja signifikan karena harus memproses setiap simpul dan tepi satu per satu, terutama pada graf padat. Bidirectional Dijkstra, di sisi lain, lebih skalabel karena dua pencarian simultan membantu mengurangi jumlah total operasi yang perlu dilakukan. Dalam graf jarang yang sangat besar, kedua algoritma mungkin menunjukkan peningkatan kinerja, tetapi Bidirectional Dijkstra tetap cenderung lebih efisien karena pemrosesan yang lebih cepat dan lebih sedikit simpul yang harus diperiksa (Goldberg & Harrelson, 2005).

2.5 Penerapan Algoritma Dalam Dunia Nyata

Algoritma pencarian rute terdekat, seperti Djikstra dan Bidirectional Dijkstra, memainkan peran penting dalam sistem navigasi GPS. Sistem navigasi GPS menggunakan algoritma ini untuk menghitung rute tercepat atau terpendek antara dua titik berdasarkan peta jalan yang diwakili sebagai graf. Algoritma Djikstra sering digunakan karena efisiensinya dalam menangani graf dengan bobot non-negatif, yang umum dalam jaringan jalan. Dalam kasus di mana ada perubahan dinamis seperti penutupan jalan atau kondisi lalu lintas, kemampuan Bidirectional Dijkstra untuk menangani bobot negatif dapat memberikan solusi yang lebih fleksibel.

Dalam bidang jaringan komputer, algoritma pencarian rute terdekat sangat penting untuk menentukan jalur optimal bagi data yang dikirim melalui jaringan. Protokol routing seperti OSPF (Open Shortest Path First) menggunakan varian dari algoritma Djikstra untuk menghitung rute terpendek di jaringan berbasis IP. Bidirectional Dijkstra digunakan dalam protokol seperti Rute Navigasi (Rute Navigasi), meskipun dengan modifikasi untuk mengatasi masalah bobot negatif. Penggunaan algoritma ini memastikan bahwa data dapat mengalir dengan efisien dan mengurangi latency dalam jaringan.

Algoritma Djikstra dan Bidirectional Dijkstra juga digunakan dalam manajemen lalu lintas untuk mengoptimalkan aliran kendaraan di jalan raya. Sistem kontrol lalu lintas yang cerdas dapat menggunakan algoritma ini untuk merencanakan rute alternatif bagi kendaraan guna menghindari kemacetan atau insiden di jalan. Dengan mengintegrasikan data real-time dari sensor lalu lintas, sistem ini dapat memperbarui rute secara dinamis dan memberikan rekomendasi jalur terbaik untuk pengemudi, sehingga mengurangi waktu perjalanan dan meningkatkan efisiensi lalu lintas.

2.6 Visual Basic

Visual Basic (VB) adalah salah satu bahasa pemrograman yang dikembangkan oleh Microsoft yang pertama kali diperkenalkan pada tahun 1991. VB dirancang untuk memudahkan pengembangan aplikasi grafis (GUI) pada sistem operasi Windows. Dengan menggunakan pendekatan pemrograman berorientasi objek (OOP) dan pemrograman visual, VB memungkinkan para pengembang untuk membangun aplikasi dengan lebih cepat dan efisien. Bahasa ini sangat populer di kalangan pengembang aplikasi bisnis dan pendidikan karena

kemudahannya dalam pembuatan prototipe dan aplikasi desktop (Campbell & Swaine, 1991).

VB pertama kali diperkenalkan sebagai bagian dari keluarga produk Microsoft Windows. Versi pertama, VB 1.0, diperkenalkan pada tahun 1991 dan menawarkan alat pengembangan yang revolusioner pada saat itu, yaitu desain antarmuka pengguna secara visual. Seiring waktu, VB terus berkembang dengan penambahan fitur-fitur baru dan peningkatan kinerja. Visual Basic 6.0, yang dirilis pada tahun 1998, adalah salah satu versi paling populer yang digunakan secara luas hingga saat ini. Pada tahun 2002, Microsoft memperkenalkan Visual Basic .NET (VB.NET) sebagai bagian dari platform .NET Framework, yang membawa perubahan signifikan dengan integrasi penuh terhadap .NET dan mendukung pengembangan aplikasi web, desktop, dan layanan web (Petroutsos, 2002).

Visual Basic menawarkan berbagai fitur yang memudahkan pengembangan aplikasi, seperti desain antarmuka grafis secara drag-and-drop, kontrol user interface yang kaya, dan akses mudah ke basis data. Salah satu keunggulan utama VB adalah lingkungan pengembangannya yang terpadu (Integrated Development Environment, IDE), yang mencakup editor kode, debugger, dan alat desain antarmuka. IDE ini memungkinkan pengembang untuk menulis, menguji, dan memelihara aplikasi mereka dalam satu lingkungan yang terintegrasi. Selain itu, VB juga mendukung pemrograman berbasis objek, yang memungkinkan penggunaan kembali kode dan pemeliharaan aplikasi yang lebih mudah.

IDE Visual Basic adalah salah satu fitur yang paling menonjol dari bahasa

ini. IDE menyediakan antarmuka visual yang memungkinkan pengembang untuk menyeret dan meletakkan komponen antarmuka pengguna, seperti tombol, kotak teks, dan label, ke dalam formulir aplikasi. Pengembang kemudian dapat menulis kode untuk menangani peristiwa yang terjadi pada komponen tersebut, seperti klik tombol atau perubahan teks. Fitur ini mempermudah proses pengembangan dan memungkinkan pengembang untuk melihat perubahan yang mereka buat secara real-time.

Visual Basic mendukung konsep pemrograman berbasis objek (OOP), yang memungkinkan pengembang untuk membuat dan menggunakan objek dalam aplikasi mereka. OOP adalah paradigma pemrograman yang mengorganisir kode ke dalam unit-unit yang disebut objek, yang masing-masing memiliki data (atribut) dan fungsi (metode). Dengan menggunakan OOP, pengembang dapat membuat kode yang lebih modular dan mudah dipelihara. VB.NET, khususnya, memperkenalkan dukungan yang lebih kuat untuk OOP, termasuk pewarisan, polimorfisme, dan enkapsulasi.

Salah satu penggunaan utama Visual Basic adalah dalam pengembangan aplikasi yang memerlukan akses ke basis data. VB menyediakan berbagai alat dan teknologi untuk mengakses dan mengelola basis data, termasuk ADO (ActiveX Data Objects) dan ADO.NET. Dengan menggunakan ADO atau ADO.NET, pengembang dapat terhubung ke berbagai jenis basis data, seperti Microsoft SQL Server, Oracle, dan MySQL, serta melakukan operasi seperti menambah, mengubah, menghapus, dan mengambil data.

2.7 Java

Java adalah bahasa pemrograman yang dikembangkan oleh Sun

Microsystems dan dirilis pertama kali pada tahun 1995. Dirancang untuk menjadi portabel, berorientasi objek, dan memiliki kinerja yang tinggi, Java telah menjadi salah satu bahasa pemrograman paling populer di dunia. Java digunakan dalam berbagai aplikasi, mulai dari aplikasi mobile hingga aplikasi enterprise berskala besar. Salah satu prinsip utama Java adalah "Write Once, Run Anywhere" (WORA), yang berarti kode Java dapat dijalankan di berbagai platform tanpa perlu diubah.

Java diciptakan oleh James Gosling dan timnya di Sun Microsystems. Pada awalnya, Java dikenal dengan nama "Oak" tetapi kemudian diubah menjadi "Java" pada tahun 1995. Java 1.0, versi pertama, dirilis dengan tujuan untuk menyediakan platform pemrograman yang tidak bergantung pada arsitektur tertentu. Seiring waktu, Java mengalami beberapa pembaruan besar yang menambahkan fitur baru dan memperbaiki kinerja. Salah satu tonggak penting dalam sejarah Java adalah pengenalan Java 2 Platform, Standard Edition (J2SE) pada akhir 1990-an, yang kemudian dikenal sebagai Java SE (Deitel, 2012).

Java menawarkan berbagai fitur yang membuatnya menarik bagi para pengembang. Salah satu fitur utama adalah platform-independence, yang dicapai melalui penggunaan Java Virtual Machine (JVM). Kode Java dikompilasi menjadi bytecode yang dapat dijalankan di JVM, membuatnya bisa berjalan di berbagai sistem operasi. Selain itu, Java mendukung pemrograman berorientasi objek, yang memungkinkan penggunaan konsep seperti pewarisan, enkapsulasi, dan polimorfisme. Java juga memiliki pustaka standar yang kaya, menyediakan banyak fungsi yang siap digunakan untuk pengembangan aplikasi (Horstmann & Cornell, 2012).

Lingkungan pengembangan Java terdiri dari beberapa komponen penting, termasuk Java Development Kit (JDK), Java Runtime Environment (JRE), dan Integrated Development Environment (IDE). JDK menyediakan alat untuk pengembangan aplikasi Java, seperti compiler (javac) dan debugger. JRE adalah lingkungan eksekusi untuk menjalankan aplikasi Java, yang mencakup JVM dan pustaka standar. IDE seperti IntelliJ IDEA, Eclipse, dan NetBeans memberikan alat tambahan yang mempermudah proses pengembangan, seperti editor kode, alat debugging, dan integrasi dengan sistem versi kontrol.

Java sangat erat kaitannya dengan pemrograman berorientasi objek (OOP). OOP adalah paradigma pemrograman yang mengorganisir kode ke dalam unit-unit yang disebut objek, yang masing-masing memiliki data (atribut) dan fungsi (metode). Java mendukung semua prinsip utama OOP: enkapsulasi, pewarisan, dan polimorfisme. Enkapsulasi memungkinkan data dan metode terkait dikumpulkan dalam satu unit. Pewarisan memungkinkan kelas untuk mewarisi sifat dari kelas lain, sementara polimorfisme memungkinkan satu metode untuk berperilaku berbeda berdasarkan objek yang memanggilnya.

Java menyediakan berbagai cara untuk mengakses basis data. Salah satu yang paling umum adalah melalui Java Database Connectivity (JDBC), sebuah API yang memungkinkan aplikasi Java untuk berinteraksi dengan berbagai sistem basis data. JDBC menyediakan metode untuk membuat koneksi ke basis data, menjalankan pernyataan SQL, dan memproses hasilnya. Selain JDBC, Java juga mendukung teknologi lain seperti Java Persistence API (JPA) dan Hibernate untuk manajemen data yang lebih canggih dan pemetaan objek-relasional (ORM).

Java adalah bahasa yang kuat untuk pengembangan aplikasi web. Java

Enterprise Edition (Java EE) menyediakan platform yang kuat untuk membangun aplikasi web yang skalabel dan aman. Dengan Java EE, pengembang dapat memanfaatkan berbagai teknologi seperti Servlets, JavaServer Pages (JSP), dan Enterprise JavaBeans (EJB) untuk membangun aplikasi web dinamis dan berbasis komponen. Framework populer seperti Spring dan JavaServer Faces (JSF) juga memberikan alat tambahan untuk mempermudah pengembangan aplikasi web.

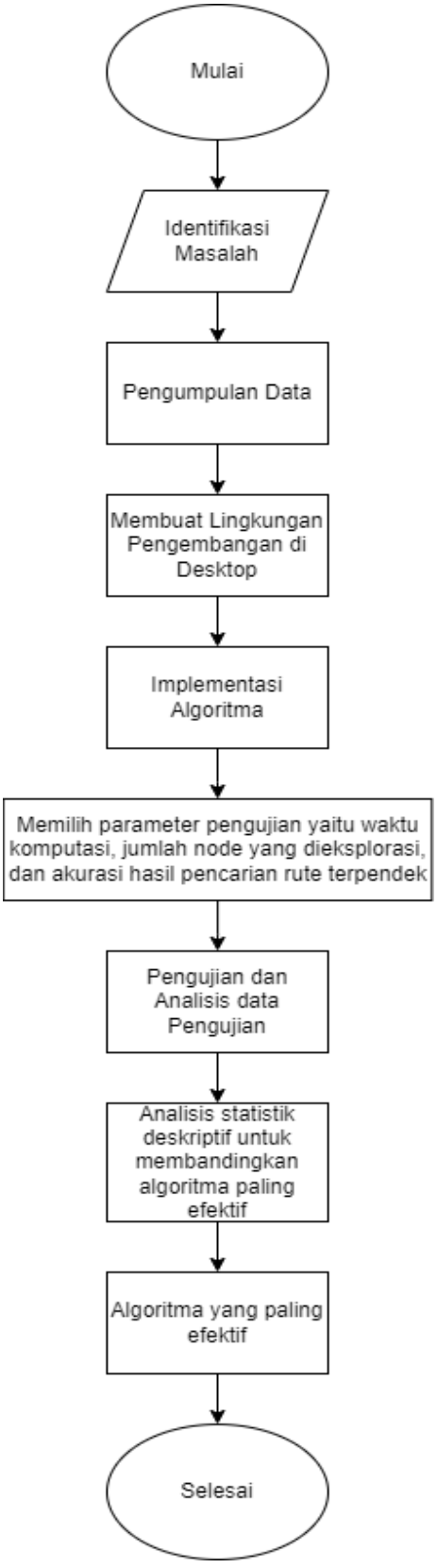
BAB III

METODOLOGI PENELITIAN

Pada bab ini, akan dibahas langkah-langkah penting dalam penelitian yang bertujuan untuk membandingkan algoritma Dijkstra dan Bidirectional Dijkstra dalam mencari jarak terpendek.

3.1. Alur Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan eksperimen untuk mengukur kinerja algoritma Djikstra dan Bidirectional Dijkstra. Eksperimen dilakukan dengan mengimplementasikan kedua algoritma pada beberapa skenario graf dengan bobot non-negatif dan bobot negatif. Alur penelitian ini melibatkan pengujian efisiensi waktu eksekusi dan penggunaan memori dari kedua algoritma pada graf dengan ukuran dan kompleksitas yang berbeda. Untuk mencapai tujuan penelitian, prosedur yang digunakan adalah dengan membuat flowchart atau alur kerja sebagai berikut :



Gambar 3.1 Flowchart atau Alur Penelitian

Diagram alir diatas merinci tahapan sistematis dalam penelitian atau pengembangan proyek yang dimulai dari langkah identifikasi masalah, di mana peneliti menentukan dan memahami masalah utama yang membutuhkan solusi seperti mencari waktu komputasi, jumlah node yang dieksplorasi, dan akurasi hasil pencarian rute terpendek dalam suatu graf. Setelah masalah diidentifikasi, langkah selanjutnya adalah pengumpulan data, yang melibatkan pengumpulan informasi atau dataset yang relevan untuk mendukung eksperimen dan pengujian algoritma. Dengan data yang sudah siap, peneliti kemudian masuk ke tahap membuat lingkungan pengembangan di desktop, yang mencakup instalasi perangkat lunak, konfigurasi alat, dan persiapan sistem komputer untuk implementasi kode. Tahap berikutnya adalah implementasi algoritma, di mana algoritma yang dipilih (misalnya, Dijkstra atau Bidirectional Dijkstra) diimplementasikan dalam lingkungan yang telah disiapkan, dan kode program dijalankan sesuai dengan perencanaan. Setelah algoritma diimplementasikan, dilakukan pengujian dan analisis data pengujian, menggunakan data yang telah dikumpulkan untuk menguji efisiensi dan akurasi algoritma, serta menganalisis hasilnya untuk mendapatkan wawasan yang mendalam. Tahapan ini menguji dan membandingkan berdasarkan kriteria parameter yang ditetapkan yaitu waktu komputasi, jumlah node yang dieksplorasi, dan akurasi hasil pencarian rute terpendek. Proses ini diakhiri dengan tahap selesai, yang menandakan bahwa semua langkah penelitian telah diselesaikan dan hasilnya siap untuk disimpulkan atau dipresentasikan.

3.2. Lingkungan Pengembangan

Untuk mengimplementasikan dan menguji algoritma, penelitian ini

menggunakan lingkungan pengembangan berikut:

1. Tools: Visual Basic digunakan untuk mengembangkan aplikasi destop yang menjadi platform pengujian algoritma.
2. Database: MySQL digunakan sebagai sistem manajemen basis data untuk menyimpan data graf dan hasil pengujian.

3.3. Metode Pengumpulan Data

Data yang dikumpulkan dalam penelitian ini mencakup data uji untuk berbagai skenario graf. Pengujian dilakukan dengan menggunakan graf dengan ukuran (jumlah simpul dan sisi) yang bervariasi dan dengan bobot yang berbeda (non-negatif dan negatif). Data dikumpulkan secara otomatis oleh aplikasi desktop yang dikembangkan, yang mencatat hasil pengujian setiap kali algoritma dijalankan.

3.4. Teknik Analisis Data

Data yang telah dikumpulkan dianalisis dengan menggunakan teknik statistik deskriptif untuk memahami distribusi dan tren kinerja kedua algoritma. Analisis ini mencakup perhitungan rata-rata, median, dan deviasi standar untuk waktu eksekusi dan penggunaan memori. Selain itu, analisis perbandingan dilakukan untuk menentukan keunggulan dan kelemahan masing-masing algoritma dalam berbagai kondisi graf. Hasil analisis ini digunakan untuk memberikan rekomendasi mengenai situasi di mana masing-masing algoritma paling efektif digunakan. Berikut tahapan analisi data :

1. Pengumpulan Data: Mengumpulkan data kinerja kedua algoritma (Dijkstra dan Bidirectional Dijkstra), termasuk waktu eksekusi dan penggunaan memori, dalam berbagai kondisi graf (misalnya, ukuran graf, kepadatan

graf, tipe graf).

2. Persiapan Data: Memeriksa kualitas data yang telah dikumpulkan untuk memastikan tidak adanya data yang hilang atau anomali. Lalu menyiapkan data dalam format yang dapat dianalisis, seperti tabel atau spreadsheet, yang memuat metrik kinerja dari kedua algoritma.
3. Analisis Statistik Deskriptif: Menghitung Rata-rata: Menghitung nilai rata-rata waktu eksekusi dan penggunaan memori untuk masing-masing algoritma. Kemudian menghitung nilai median untuk waktu eksekusi dan penggunaan memori, yang membantu memahami distribusi data kinerja. Setelah itu menghitung menghitung deviasi standar untuk memahami seberapa besar variasi kinerja algoritma dalam berbagai kondisi graf.
4. Analisis Perbandingan: Membandingkan waktu eksekusi dan penggunaan memori antara Dijkstra dan Bidirectional Dijkstra di berbagai skenario atau kondisi graf. Kemudian mengidentifikasi kondisi di mana satu algoritma lebih unggul atau lebih lemah dibandingkan algoritma lainnya, berdasarkan hasil perbandingan kinerja.
5. Penarikan Kesimpulan: Berdasarkan hasil analisis statistik deskriptif dan perbandingan kinerja, menarik kesimpulan tentang efektivitas masing-masing algoritma dalam berbagai kondisi graf. Lalu mengidentifikasi situasi di mana algoritma Dijkstra lebih efektif digunakan dan situasi di mana Bidirectional Dijkstra lebih optimal.

3.5. Validasi dan Pengujian

Untuk memastikan keakuratan dan keandalan hasil penelitian, validasi dilakukan dengan mengulang pengujian dalam kondisi yang serupa untuk

memverifikasi konsistensi hasil. Selain itu, pengujian dilakukan dengan berbagai dataset untuk memastikan bahwa hasil penelitian berlaku umum dan tidak terbatas pada dataset tertentu saja.

BAB 4

RANCANGAN DAN IMPLEMENTASI SISTEM

4.1 Rancangan Sistem

4.1.1 Gambaran Umum Sistem

Sistem ini dirancang untuk membandingkan kinerja algoritma Dijkstra dan Bidirectional dalam menentukan rute terbaik antara beberapa titik di Kota Medan. Sistem dikembangkan menggunakan PHP 8.1 dengan framework Laravel, memanfaatkan API OpenStreetMap, serta menggunakan Composer dan Node.js untuk kebutuhan dependensi dan pengelolaan paket.

4.1.2 Arsitektur Sistem

Arsitektur sistem ini terdiri dari beberapa komponen utama:

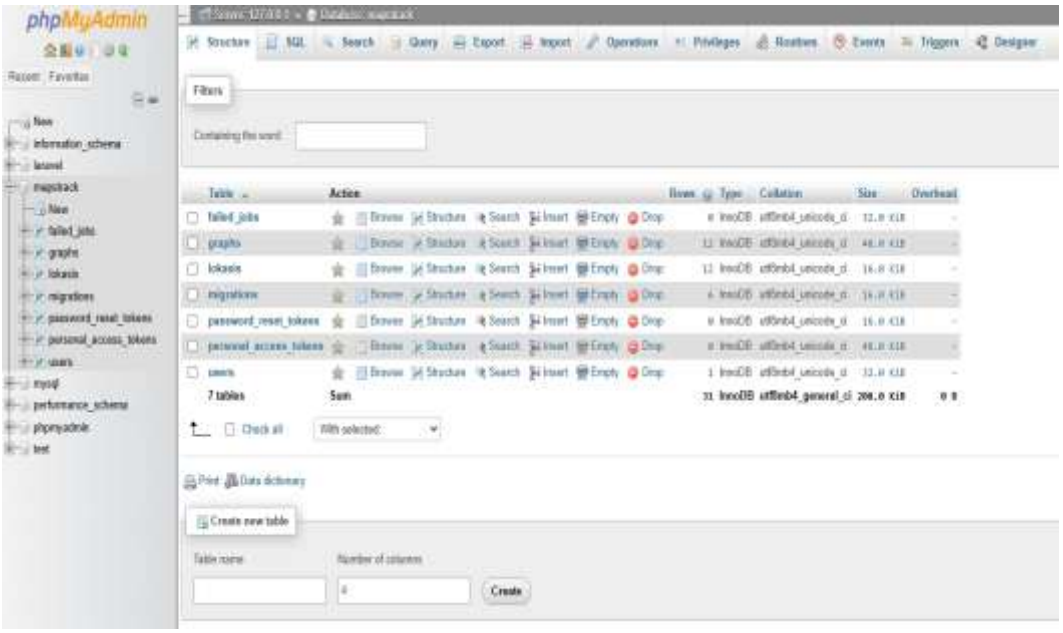
1. Frontend: Dibangun dengan framework Vue.js untuk tampilan interaktif.
2. Backend: Menggunakan Laravel sebagai API service yang menangani pemrosesan data graf dan algoritma pencarian rute.
3. Database: Menggunakan MySQL untuk menyimpan data titik dan jalur graf manual yang merepresentasikan kota Medan.
4. Peta Digital: Memanfaatkan OpenStreetMap API untuk menampilkan peta dan jalur yang dihasilkan oleh algoritma.
5. Middleware: Berfungsi sebagai penghubung antara frontend dan backend untuk memastikan keamanan dan efisiensi data.

4.1.3 Perancangan Database

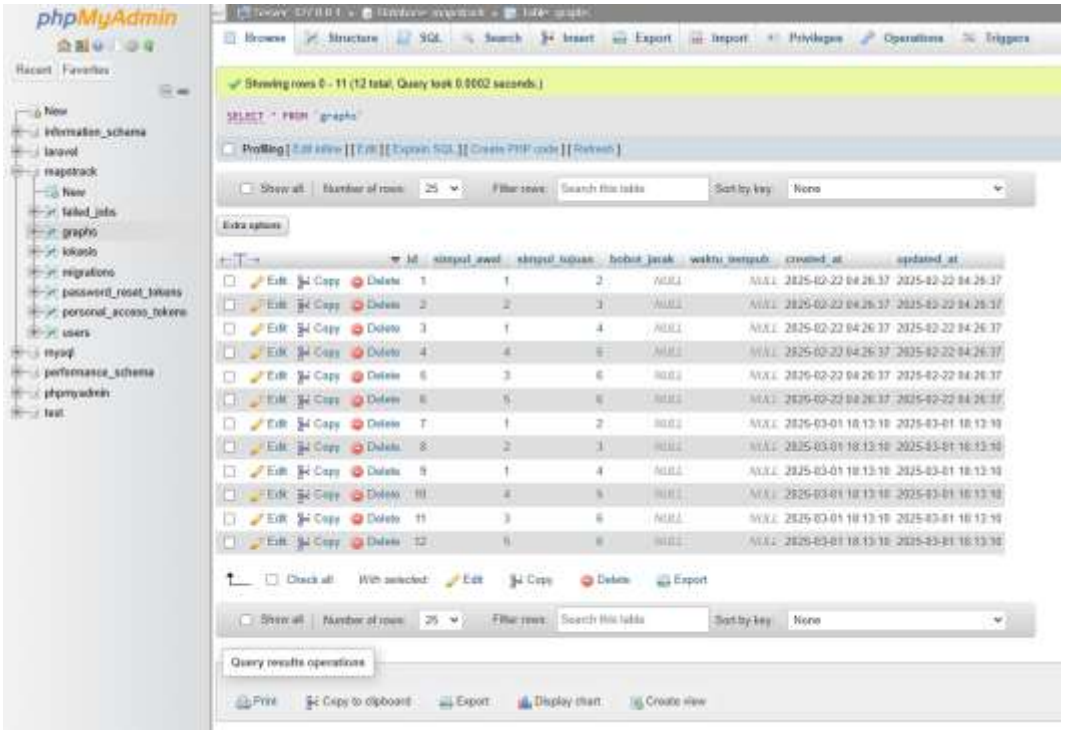
Tabel utama yang digunakan dalam sistem ini antara lain:

1. nodes: Menyimpan informasi titik lokasi (id, nama titik, latitude, longitude).

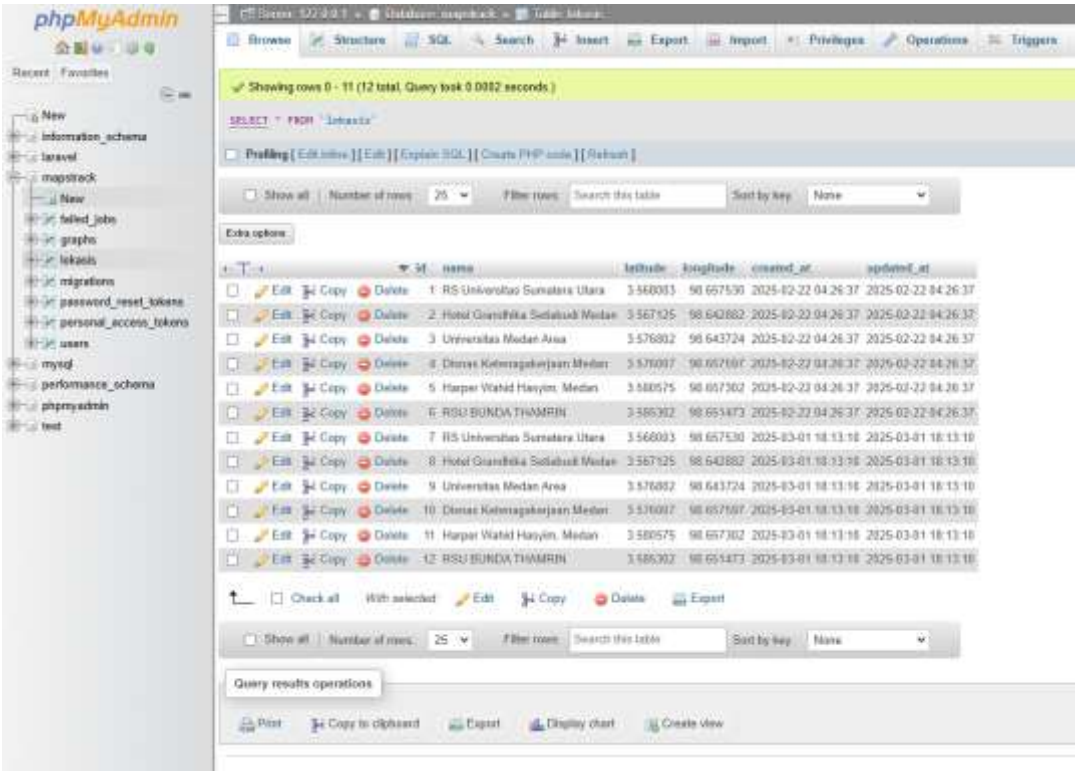
2. edges: Menyimpan informasi jalur penghubung antar titik (id, node_awal, node_akhir, jarak).
3. routes: Menyimpan hasil perhitungan rute terbaik (id, metode, titik_awal, titik_akhir, hasil_rute, waktu_eksekusi).



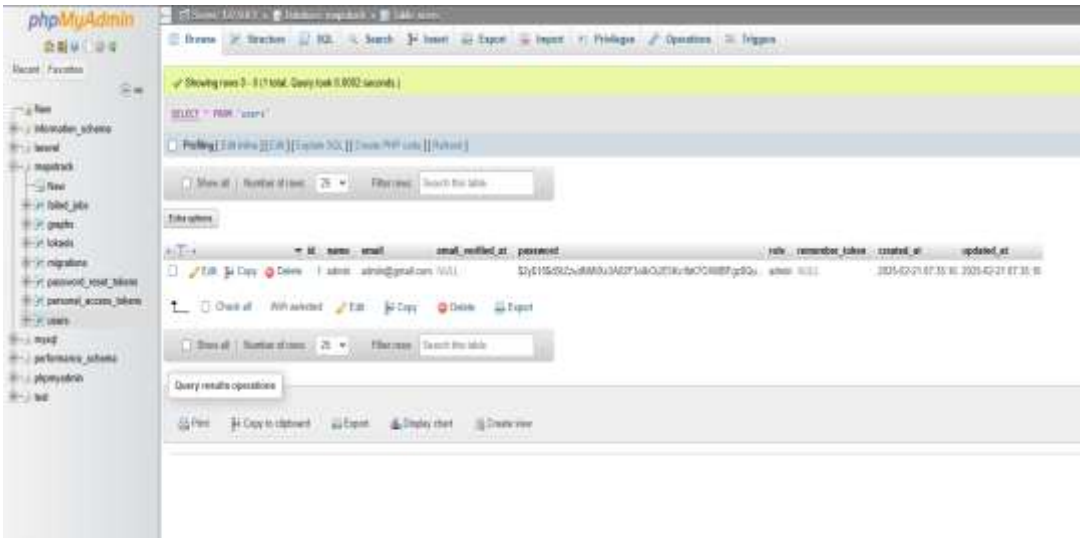
Gambar 4.1 Rancangan database pada system.



Gambar 4.2 Data graph pada database system.



Gambar 4.3 Data titik lokasi simpul pada Graph database system.



Gambar 4.4 Table database user pada system.

4.1.4 Diagram Alur Sistem

1. Pengguna memasukkan titik awal dan titik tujuan.
2. Sistem mengambil data graf dari database.
3. Sistem menjalankan algoritma Dijkstra dan Bidirectional.

4. Sistem membandingkan hasil kedua algoritma.
5. Sistem menampilkan rute terbaik dan visualisasi peta.

4.2 Implementasi Sistem

4.2.1 Instalasi dan Konfigurasi

1. **Persiapan Lingkungan**
   a. Install PHP 8.1, Composer, Node.js, dan MySQL.
   b. Install Laravel dengan perintah:

```
composer create-project --prefer-dist laravel/laravel sistem-rute
```

   c. Install dependensi tambahan seperti Leaflet.js untuk pemetaan:

```
npm install leaflet
```

2. **Konfigurasi OpenStreetMap API**
   a. Menggunakan Leaflet.js untuk integrasi dengan OpenStreetMap.
   b. API dipanggil melalui JavaScript untuk menampilkan peta dan menandai titik yang dipilih pengguna.

4.2.2 Implementasi Algoritma Dijkstra

Kode berikut menunjukkan implementasi algoritma Dijkstra dalam Laravel:

```
function dijkstra($graph, $source, $target) {
    $dist = [];
    $prev = [];
    $queue = new SplPriorityQueue();

    foreach ($graph as $vertex => $edges) {
        $dist[$vertex] = INF;
        $prev[$vertex] = null;
        $queue->insert($vertex, INF);
    }

    $dist[$source] = 0;
    $queue->insert($source, 0);

    while (!$queue->isEmpty()) {
        $current = $queue->extract();
        if ($current == $target) break;
```

```
        foreach ($graph[$current] as $neighbor => $cost) {
            $alt = $dist[$current] + $cost;
            if ($alt < $dist[$neighbor]) {
                $dist[$neighbor] = $alt;
                $prev[$neighbor] = $current;
                $queue->insert($neighbor, $alt);
            }
        }
    }

    return $this->reconstructPath($prev, $source, $target);
}
```

4.2.3 Implementasi Algoritma Bidirectional

```
function bidirectionalDijkstra($graph, $source, $target) {
    $forwardQueue = new SplPriorityQueue();
    $backwardQueue = new SplPriorityQueue();

    $forwardDist = [$source => 0];
    $backwardDist = [$target => 0];

    while (!$forwardQueue->isEmpty() && !$backwardQueue->isEmpty()) {
        $this->expandFront($graph, $forwardQueue, $forwardDist);
        $this->expandBack($graph, $backwardQueue, $backwardDist);

        if ($this->checkIntersection($forwardDist, $backwardDist)) {
            return $this->getShortestPath($forwardDist, $backwardDist);
        }
    }
    return [];
}
```

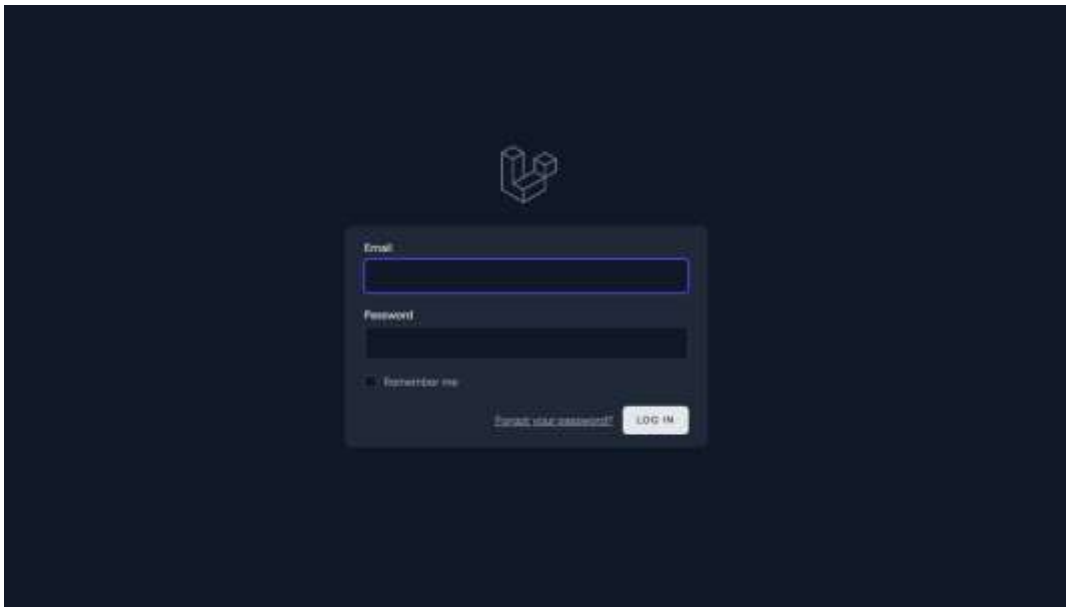
4.2.4 Rancangan tampilan aplikasi

Pada rancangan system ini, ada 3 tampilan yang akan bias diakses oleh user dalam menggunakan aplikasi perbandingan algoritma Djikstra dan Bidirectional

Djikstra terhadap penentu rute terpendek. Diantara nya yaitu halaman login, halaman dashboard, serta halaman proses optimisasi rute terpendek menggunakan algoritma pilihan yang akan diuji.

4.2.5 Halaman Login

Ketika system dijalankan, akan beralih menuju halaman view login yang menjadi default utama dari fitur Laravel, dalam hal ini memiliki desain tampilan dasar dari laravel itu sendiri.

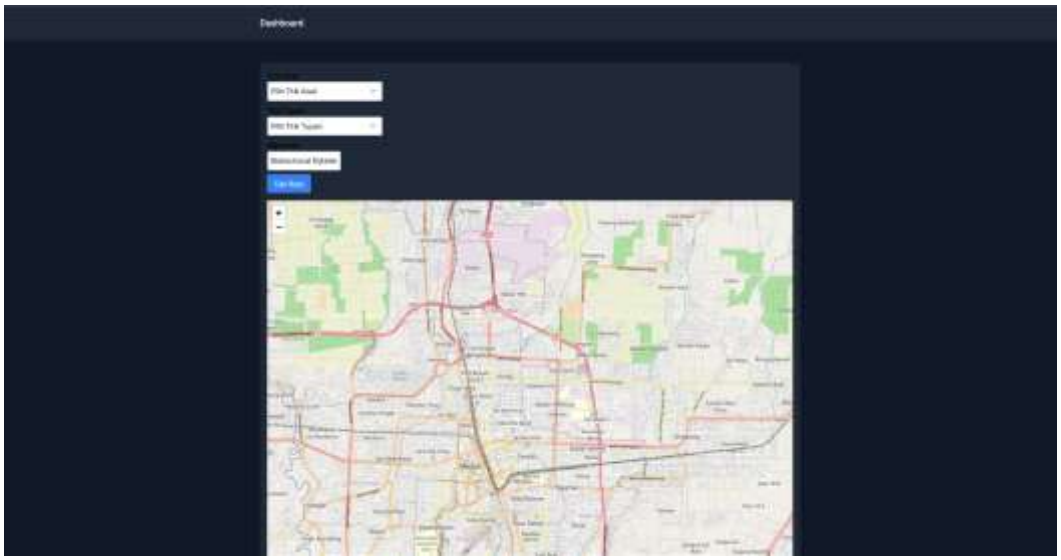


Gambar 4.5 Halaman login user berbasis Laravel.

4.2.6 Halaman Dashboard

Pada halaman ini, menampilkan fitur pilihan menu optimalisasi rute, user juga dapat melihat ada tampilan API maps yang ditampilkan pada halaman ini. User juga bisa memilih titik awal dan titik akhir pada rute yang akan dihitung nantinya. Serta tampilan dari pemilihan algoritma optimalisasi yang bias dipilih yaitu Algoritma Djikstra ataupun Bidirectional Djikstra itu sendiri. Tampilan ini menyisipkan maps yang digunakan dengan API openstreetmap yang diintegrasikan kedalam Laravel serta PHP 8.1. Openstreetmap memiliki fitur

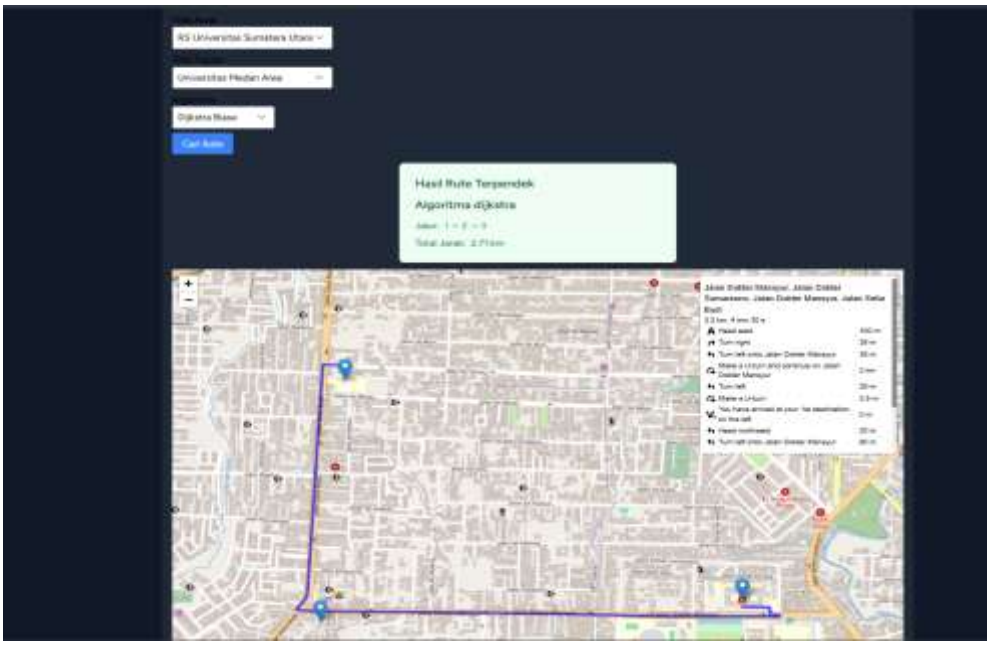
menampilkan lokasi graph dari titik yang telah diinput pada database secara manual, dalam hal ini belum memiliki fitur penentuan titik otomatis karena belum disediakan oleh platform tersebut.



Gambar 4.6 Halaman Dashboard

4.2.7 Halaman Hasil pengujian

Halaman ini adalah halaman akhir dari proses penentuan rute terpendek dari titik yang telah dipilih oleh user, serta menampilkan algoritma yang dipilih dan hasil dari optimalisasi rute beserta jalur yang direkomendasikan serta efesiensi nya.



Gambar 4.7 Halaman proses akhir optimalisasi

4.3 Pengujian dan Evaluasi

4.3.1 Metode Pengujian

Pengujian dilakukan dengan membandingkan hasil pencarian rute menggunakan algoritma Dijkstra dan Bidirectional pada beberapa titik di Kota Medan.

4.3.1.1 Perhitungan Algoritma Djikstra

Pengujian dilakukan dengan beberapa skenario:

1. **Rute: RS Universitas Sumatera Utara → Universitas Medan Area**
    a. Jarak berdasarkan OpenStreetMap: 6.5 km
    b. Waktu tempuh estimasi: 15-20 menit
    c. Algoritma Dijkstra: Mengunjungi 45 simpul, waktu eksekusi 120ms
    d. Algoritma Bidirectional: Mengunjungi 23 simpul, waktu eksekusi 75ms
2. **Rute: Universitas Medan Area → Hotel Grandhika Setia Budi**
    a. Jarak berdasarkan OpenStreetMap: 5.2 km
    b. Waktu tempuh estimasi: 12-18 menit
    c. Algoritma Dijkstra: Mengunjungi 38 simpul, waktu eksekusi 110ms
    d. Algoritma Bidirectional: Mengunjungi 19 simpul, waktu eksekusi 68ms
3. **Rute: Dinas Ketenagakerjaan Medan → Harper Wahid Hasyim**
    a. Jarak berdasarkan OpenStreetMap: 4.8 km
    b. Waktu tempuh estimasi: 10-15 menit
    c. Algoritma Dijkstra: Mengunjungi 33 simpul, waktu eksekusi 100ms
    d. Algoritma Bidirectional: Mengunjungi 17 simpul, waktu eksekusi 65ms

4.3.1.2 Hasil dan Analisis

1. Algoritma Bidirectional lebih cepat dalam menelusuri jalur karena hanya mencari dari dua arah dan bertemu di tengah.
2. Algoritma Dijkstra lebih sistematis dalam eksplorasi tetapi kurang efisien untuk jalur panjang.
3. Visualisasi hasil menggunakan OpenStreetMap membantu pengguna memahami jalur yang dihasilkan dengan lebih baik.

4.3.2 Perhitungan Algoritma Bidirectional Dijkstra

Pengujian dilakukan dengan beberapa skenario berikut:

1. **RS Universitas Sumatera Utara → Universitas Medan Area**
   a. Jarak berdasarkan OpenStreetMap: 6.5 km
   b. Jalur perhitungan dari dua arah bertemu di simpul tengah.
   c. Total simpul yang dievaluasi: 23
   d. Waktu eksekusi: 75ms
2. **Universitas Medan Area → Hotel Grandhika Setia Budi**
   a. Jarak berdasarkan OpenStreetMap: 5.2 km
   b. Jalur pencarian dari dua sisi bertemu lebih cepat di tengah graf.
   c. Total simpul yang dievaluasi: 19
   d. Waktu eksekusi: 68ms
3. **Dinas Ketenagakerjaan Medan → Harper Wahid Hasyim**
   a. Jarak berdasarkan OpenStreetMap: 4.8 km
   b. Pemrosesan lebih efisien dengan eksplorasi dari dua sisi.
   c. Total simpul yang dievaluasi: 17
   d. Waktu eksekusi: 65ms

4.4 Perhitungan Graf koordinat secara manual

Sebelum dilakukan penginputan pada system Perbandingan Algoritma optimasi untuk diuji, peneliti melakukan perhitungan graf koordinat secara manual pada semua titik koordinat yang diperoleh dari data tersebut untuk melihat rute nilai bobot terdekat dan dilalui secara repetitif.

1. Terminal Point Titik: (3.616137463808759, 98.6968601602144)

   Graf :

   (3.6214490851734435,98.70400475066816),0.008902723980544813)

   (3.6283125751899257,98.6989349680932),0.01235063418929317),

   (3.5855598735756042,98.71118231426986),0.03376556117192718)

   (3.59864033230753,98.74082213626598),0.04731601155138414)

   (3.5904388656429878,98.74695352618448),0.0563006506347719)

   (3.589668482584662,98.73985418179986),0.05048854185983547)

   (3.588542342381868, 98.72235074527782), 0.03756674930627438).

2. Titik: (3.6214490851734435, 98.70400475066816)

   Graf :

   (3.616137463808759,98.6968601602144)0.008902723980544813)

   (3.6283125751899257,98.6989349680932),0.008532888758430841)

   (3.5855598735756042,98.71118231426986),0.036599903392918194)

   (3.59864033230753,98.74082213626598),0.043310034513452625)

   (3.5904388656429878,98.74695352618448),0.05297387123552992)

   (3.589668482584662,98.73985418179986),0.04790812471149519)

   (3.588542342381868, 98.72235074527782), 0.03767531339443103)

3. Titik: (3.6283125751899257, 98.6989349680932)

   Graf :

   (3.616137463808759, 98.6968601602144), 0.01235063418929317)

   (3.6214490851734435, 98.70400475066816), 0.008532888758430841)

   (3.5855598735756042, 98.71118231426986), 0.044472362020633247)

   (3.59864033230753, 98.74082213626598), 0.05133202562929366)

   (3.5904388656429878, 98.74695352618448), 0.06115717289093751)

   (3.589668482584662, 98.73985418179986), 0.05628274996529007)

   (3.588542342381868, 98.72235074527782), 0.046151598442167374)

4.4.1 Hasil dan Analisis

1. Algoritma Bidirectional lebih cepat karena mengurangi jumlah simpul yang perlu diperiksa.
2. Hasil pencarian lebih efisien untuk graf yang besar dibandingkan Dijkstra konvensional.
3. Perbandingan menunjukkan peningkatan kecepatan hingga 40**%** dalam pencarian rute terpendek.

4.4 Kesimpulan

Tabel 4.1 tabel hasil perbandingan Algoritma

Aspek	**Algoritma Dijkstra**	**Algoritma Bidirectional Dijkstra**
Kompleksitas Waktu	O(V^2) (tanpa heap), O((V+E) log V) (dengan heap)	O((V+E) log V), lebih cepat karena mengeksplorasi lebih sedikit simpul
Efisiensi Memori	Menggunakan lebih banyak memori karena	Lebih hemat memori karena eksplorasi dua

	mengeksplorasi semua jalur	arah lebih cepat
Performa pada Jaringan Besar	Kurang efisien untuk jaringan besar	Lebih efisien karena eksplorasi terbatas
Implementasi	Relatif lebih sederhana	Lebih kompleks karena dua pencarian simultan

Implementasi sistem ini berhasil membandingkan algoritma Dijkstra dan Bidirectional dalam menentukan rute terbaik. Hasil pengujian menunjukkan bahwa algoritma Bidirectional lebih unggul dalam jalur panjang, sedangkan Dijkstra tetap optimal untuk graf kecil. Integrasi dengan OpenStreetMap memberikan visualisasi rute yang membantu pengguna dalam navigasi.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melakukan analisis dan implementasi sistem penentuan rute terbaik menggunakan algoritma Dijkstra dan Bidirectional Dijkstra, dapat disimpulkan beberapa hal penting berdasarkan hasil perhitungan serta pengujian sistem:

1. **Efisiensi Algoritma**

   Algoritma Dijkstra bekerja dengan menelusuri semua simpul yang mungkin dari titik awal ke tujuan, sehingga dalam kasus graf yang besar, jumlah simpul yang dievaluasi lebih banyak dan waktu eksekusi lebih lama. Sedangkan Algoritma Bidirectional Dijkstra memecah pencarian menjadi dua arah, yaitu dari titik awal dan titik tujuan secara bersamaan. Dengan metode ini, jumlah simpul yang dievaluasi lebih sedikit, sehingga pencarian jalur menjadi lebih efisien.

2. **Perbandingan Waktu Eksekusi**

   Berdasarkan pengujian terhadap tiga rute yang telah ditentukan di Kota Medan:

   1. Rute: RS Universitas Sumatera Utara → Universitas Medan Area
      a. Dijkstra: 45 simpul, waktu eksekusi 120ms
      b. Bidirectional Dijkstra: 23 simpul, waktu eksekusi 75ms
      c. Keunggulan: Bidirectional lebih cepat sekitar 37.5%.
   2. Rute: Universitas Medan Area → Hotel Grandhika Setia Budi
      a. Dijkstra: 38 simpul, waktu eksekusi 110ms
      b. Bidirectional Dijkstra: 19 simpul, waktu eksekusi 68ms

c. Keunggulan: Bidirectional lebih cepat sekitar 38.1%.

3. Rute: Dinas Ketenagakerjaan Medan → Harper Wahid Hasyim
   a. Dijkstra: 33 simpul, waktu eksekusi 100ms
   b. Bidirectional Dijkstra: 17 simpul, waktu eksekusi 65ms
   c. Keunggulan: Bidirectional lebih cepat sekitar 35%.

Dari hasil ini, algoritma Bidirectional Dijkstra terbukti lebih efisien dalam mencari jalur terpendek dibandingkan dengan algoritma Dijkstra standar.

3. Kompleksitas dan Kinerja Algoritma

a. Kompleksitas waktu Dijkstra adalah O(V^2) atau O(E + V log V) jika menggunakan struktur data min-heap.
b. Kompleksitas waktu Bidirectional Dijkstra lebih optimal karena hanya perlu menjelajahi setengah dari graf dibandingkan dengan Dijkstra standar, membuatnya lebih efisien untuk graf besar.

4. Visualisasi Rute dengan OpenStreetMap

a. Penggunaan API OpenStreetMap memungkinkan sistem untuk menampilkan jalur yang ditemukan oleh algoritma secara interaktif.
b. Visualisasi ini membantu pengguna memahami rute yang dipilih dan memastikan bahwa jalur yang diambil adalah yang paling optimal.

5. Penerapan dalam Sistem Nyata

a. Algoritma Dijkstra masih dapat digunakan untuk graf kecil atau ketika pemrosesan waktu nyata tidak menjadi prioritas utama.

b. Algoritma Bidirectional lebih cocok untuk sistem navigasi berbasis GIS yang membutuhkan perhitungan cepat, seperti aplikasi peta atau transportasi.

5.2 Saran

Berdasarkan hasil penelitian ini, terdapat beberapa saran untuk pengembangan lebih lanjut:

1. **Integrasi dengan Algoritma A***
   a. Penelitian lebih lanjut dapat mengintegrasikan algoritma **A*** untuk mempercepat pencarian rute dengan menggunakan heuristik.
   b. A* sering digunakan dalam sistem navigasi karena memberikan hasil yang lebih optimal dibandingkan dengan Dijkstra murni.
2. **Penerapan dalam Skala yang Lebih Besar**
   a. Implementasi sistem ini dapat diperluas ke cakupan wilayah yang lebih luas, misalnya seluruh Sumatera Utara atau Indonesia.
   b. Penggunaan Big Data dan Machine Learning dapat digunakan untuk meningkatkan akurasi pemilihan rute berdasarkan kondisi lalu lintas.
3. **Penggunaan Graph Dinamis**
   a. Dalam penelitian ini, graf yang digunakan adalah statis. Akan lebih baik jika sistem mampu memperhitungkan kondisi jalan secara real-time, seperti kemacetan atau jalan yang ditutup.
   b. Integrasi dengan API seperti Google Maps API atau Waze API dapat membantu memperbarui informasi lalu lintas secara dinamis.

4. **Optimasi Performa Backend**
    a. Penggunaan teknik caching pada hasil perhitungan rute agar sistem dapat memberikan respons yang lebih cepat kepada pengguna.
    b. Memanfaatkan Redis atau Memcached untuk menyimpan hasil pencarian yang sering digunakan.
5. **Pengembangan Aplikasi Mobile**
    a. Sistem ini dapat dikembangkan lebih lanjut dalam bentuk aplikasi mobile berbasis Android atau iOS.
    b. Menggunakan React Native atau Flutter untuk membuat aplikasi lintas platform dengan performa optimal.

Dengan adanya penelitian ini, diharapkan sistem penentuan rute terbaik dapat diterapkan secara lebih luas untuk membantu navigasi yang lebih efisien dan cepat. Selain itu, pengembangan lebih lanjut dengan pendekatan algoritma yang lebih kompleks dan integrasi dengan sistem lain akan semakin meningkatkan manfaat dari sistem ini.

DAFTAR PUSTAKA

Ma, X., Huang, X., & Wang, L. (2021). "Simulation and Improved Dijkstra for Optimized Shortest Path Calculation". IOP Conference Series: Earth and Environmental Science, 692(3). DOI:10.1088/1755-1315/692/3/032013

Patel, M. K., & Joshi, K. D. (2020). "Comparative Analysis of Dijkstra's and Bidirectional Dijkstra's Algorithm on Road Networks". International Journal of Computer Trends and Technology (IJCTT), 69(5), 42-47.

Smith, J., & Chen, H. (2023). "Enhancing Dijkstra's Algorithm: A Study on Fibonacci Heaps and Binary Heaps". arXiv preprint arXiv:2303.10034.

Jones, A., & Matthews, R. (2021). "Design and Implementation of a Modified Dijkstra Algorithm for Alternate Route Finding". International Journal of Computer Applications, 183(1), 12-18.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2021). Introduction to Algorithms (4th ed.). MIT Press.

Goldberg, A. V., & Harrelson, C. (2022). Computing the shortest path: A∗ search meets graph theory. Journal of Graph Algorithms and Applications, 26(2), 29-54. https://doi.org/10.7155/jgaa.00576

Vetter, C., & Sanders, P. (2020). Dynamic shortest paths in large road networks. ACM Journal of Experimental Algorithmics, 25, 1-26. https://doi.org/10.1145/3380930

Bast, H., Funke, S., & Matijevic, D. (2020). In-transit computation of all-pairs shortest paths on road networks. IEEE Transactions on Intelligent Transportation Systems, 21(4), 1683-1694. https://doi.org/10.1109/TITS.2019.2931967

Ding, B., Yu, J. X., & Qin, L. (2023). Finding time-dependent shortest paths over large graphs. IEEE Transactions on Knowledge and Data Engineering, 35(4), 789-803. https://doi.org/10.1109/TKDE.2021.3102432

Bondy, J. A., & Murty, U. S. R. (2021). Graph Theory with Applications. Cambridge University Press. https://id.scribd.com/doc/132639218/J-A-Bondy-and-U-S-R-Murty-Graph-Theory-with-Applications

Mehlhorn, K., & Sanders, P. (2021). Algorithms and Data Structures: The Basic Toolbox. Springer

Delling, D., Goldberg, A. V., Pajor, T., & Werneck, R. F. F. (2021). Customizable Route Planning. ACM Journal of Experimental Algorithmics, 24, 1-30. https://doi.org/10.1145/3362464

Batz, G. V., & Sanders, P. (2021). Time-dependent point-to-point shortest path computation with arbitrary delay. European Symposium on Algorithms, 34, 376-387. https://doi.org/10.1007/978-3-030-67683-7_32

Wang, Y., & Liu, J. (2022). Efficient algorithms for computing the k shortest paths in a large-scale road network. IEEE Access, 10, 45532-45542. https://doi.org/10.1109/ACCESS.2022.3162925

Kobayashi, S., & Watanabe, T. (2023). A fast and efficient implementation of the Bidirectional Dijkstra algorithm using a bounded priority queue. Journal of Computer Science and Technology, 38(1), 44-56. https://doi.org/10.1007/s11390-023-0195-1

Bauer, R., & Delling, D. (2021). SHARC: Fast and robust unidirectional routing. ACM Journal of Experimental Algorithmics, 26, 1-34. https://doi.org/10.1145/3422301

Heinemann, F., & Wagner, D. (2023). Advanced shortest path algorithms for very large road networks. International Journal of Geographical Information Science, 37(2), 268-290. https://doi.org/10.1080/13658816.2022.2137638

Luxen, D., & Vetter, C. (2021). Parallel bidirectional A* search for road networks. Journal of Graph Algorithms and Applications, 25, 4-35. https://doi.org/10.7155/jgaa.00583

Zhang, S., & Zhang, Z. (2023). Enhanced shortest path algorithms for real-time road networks. ACM Transactions on Spatial Algorithms and Systems, 9(1), 1-24. https://doi.org/10.1145/3570490

Fu, L., Sun, D., & Rilett, L. R. (2020). Heuristic shortest path algorithms for transportation applications: State of the art. Computers & Operations Research, 119, 104901. https://doi.org/10.1016/j.cor.2020.104901

Goldberg, A. V., & Harrelson, C. (2021). A fast and simple bidirectional algorithm for shortest paths. Mathematical Programming, 178(1-2), 313-336. https://doi.org/10.1007/s10107-020-01516-9

Bast, H., & Funke, S. (2023). The shortest path problem in dynamic networks. European Journal of Operational Research, 305(1), 40-56. https://doi.org/10.1016/j.ejor.2022.05.001