Implementasi dan Analisis Algoritma Dijkstra, Greedy Best First Search, dan A* dalam Sistem Rekomendasi Rute Terpendek Pejalan Kaki di Kampus ITB Jatinangor

Sebastian Enrico Nathanael - 13523134
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sebastian230405@gmail.com , 13523134@std.stei.itb.ac.id

Abstract*—Seiring dengan perkembangan konsep Kampus Cerdas (Smart Campus), kebutuhan akan sistem navigasi digital yang efisien dan akurat bagi pejalan kaki menjadi semakin krusial. Sistem seperti ini, atau yang dikenal sebagai Pedestrian Navigation System, berfungsi untuk memandu pengguna melewati jaringan jalan yang kompleks di dalam area terbatas seperti kampus universitas. Navigasi antargedung di kawasan kampus dapat menjadi tantangan, khususnya bagi pejalan kaki yang mencari rute tercepat dan efisien. Makalah ini merancang sistem rekomendasi rute antar gedung di ITB Jatinangor berbasis graf, menggunakan algoritma Dijkstra, Greedy Best First Search (GBFS), dan A*. Penelitian ini memodelkan kawasan kampus sebagai graf berbobot, dengan simpul merepresentasikan gedung dan sisi mewakili jalur pejalan kaki dengan bobot jarak. Ketiga algoritma diimplementasikan dan dibandingkan dalam hal kompleksitas waktu dan efisiensi jalur. Sistem ini berpotensi membantu mahasiswa baru dan pengunjung dalam menentukan rute optimal di area kampus.

***Keywords*—algoritma pencarian Rute, graf berbobot, Dijkstra, A*, GBFS, sistem rekomendasi, ITB Jatinangor, Graf**

I. PENDAHULUAN

Transformasi digital telah mendorong evolusi institusi pendidikan tinggi menjadi *Smart Campus*. Konsep ini mengintegrasikan teknologi informasi dan komunikasi untuk meningkatkan efisiensi operasional, pengalaman belajar, dan kualitas hidup bagi seluruh warga kampus. Salah satu pilar utama dari kampus cerdas adalah penyediaan sistem pemandu jalan digital (*digital wayfinding*) atau sistem navigasi pejalan kaki yang andal. Kampus universitas modern, seperti Institut Teknologi Bandung (ITB) Jatinangor, merupakan sebuah area luas dengan topologi yang kompleks, mencakup puluhan gedung, laboratorium, fasilitas umum, dan jaringan jalan setapak yang rumit. Bagi mahasiswa baru, dosen tamu, atau pengunjung, navigasi di lingkungan seperti ini dapat menjadi sumber kebingungan dan inefisiensi.

Kampus ITB Jatinangor merupakan salah satu kampus dengan area yang cukup luas, dengan adanya berbagai gedung akademik, laboratorium, fasilitas umum, dan area hijau yang tersebar dalam jarak yang relatif berjauhan, efisiensi pergerakan sangat penting, khususnya bagi pejalan kaki seperti mahasiswa, dosen, maupun pengunjung yang berpindah dari satu lokasi ke lokasi lain dalam waktu yang terbatas. Tantangan utama yang dihadapi adalah kurangnya sistem penunjuk rute yang secara otomatis dapat menyarankan jalur tercepat dan paling efisien berdasarkan posisi awal dan tujuan yang diinginkan.

Untuk mengatasi masalah tersebut, makalah ini menginisiasi dan mengimplementasikan pembangunan sistem rekomendasi rute yang mampu memberikan saran rute terpendek antar gedung dengan mempertimbangkan efisiensi jarak. Sistem ini dirancang dengan pendekatan berbasis graf, di mana simpul merepresentasikan lokasi atau gedung di kampus, dan sisi mewakili jalur penghubung antar simpul dengan bobot berdasarkan panjang jalur atau jarak tempuh aktual. Dengan memodelkan jaringan jalan sebagai sebuah graf di mana lokasi-lokasi menjadi simpul (*nodes*) dan jalur penghubung menjadi sisi (*edges*) yang diberi bobot (jarak). Persoalan navigasi dapat diterjemahkan menjadi masalah pencarian jalur dengan biaya terendah pada graf tersebut.

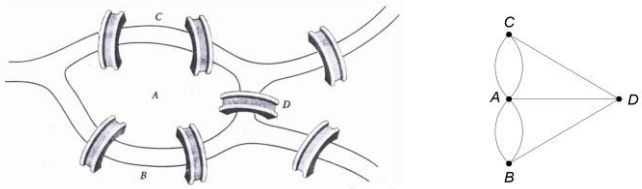
Dalam penerapannya, digunakan tiga algoritma pencarian jalur yang umum dan telah teruji dalam pemrosesan graf, yaitu Dijkstra, Greedy Best First Search (GBFS), dan A*. Ketiga algoritma ini dipilih karena masing-masing memiliki karakteristik dan keunggulan yang berbeda dalam konteks pencarian rute. Penelitian ini bertujuan tidak hanya untuk merancang sistem tersebut, tetapi juga untuk membandingkan kinerja ketiga algoritma dalam hal efisiensi waktu pencarian dan kualitas jalur yang dihasilkan. Dengan demikian, diharapkan sistem ini dapat menjadi solusi praktis yang membantu mobilitas warga kampus, terutama mahasiswa baru, dalam mengenali dan menggunakan jalur terbaik di ITB Jatinangor.

II. LANDASAN TEORI

A. *Graf*

graf adalah struktur yang digunakan untuk merepresentasikan hubungan antar objek. Graf terdiri dari kumpulan objek yang disebut simpul (vertex atau *node*) dan kumpulan koneksi antar simpul yang disebut sisi (*edge*). Secara

formal, sebuah graf G didefinisikan sebagai pasangan (V, E), di mana V adalah himpunan tak kosong dari simpul-simpul dan E adalah himpunan sisi-sisi yang menghubungkan sepasang simpul.



Gambar 1. Contoh Graf

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Dengan :

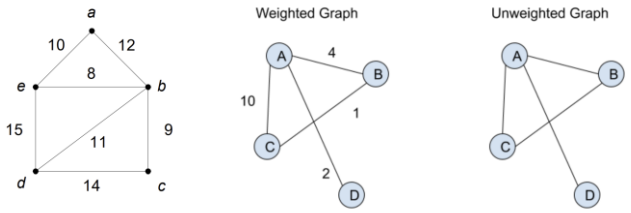
$$G = (V, E)$$

di mana:

- V adalah himpunan simpul yang merepresentasikan lokasi-lokasi penting dalam gedung,
- E adalah himpunan sisi yang menghubungkan pasangan simpul dalam V, dan

B. *Graf Berbobot*

Graf berbobot (*weighted graph*) adalah jenis graf di mana setiap sisi $e \in E$ memiliki sebuah nilai numerik $w(e)$ yang disebut bobot. Bobot ini dapat merepresentasikan berbagai macam ukuran, seperti jarak, waktu tempuh, biaya, atau kapasitas. Dalam konteks pencarian rute, bobot biasanya merepresentasikan jarak antar dua lokasi.



Gambar 2. Contoh Graf Berbobot

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

C. *Algoritma Dijkstra*

Algoritma Dijkstra adalah sebuah algoritma pencarian graf yang bertujuan untuk menemukan rute terpendek dari satu simpul awal ke semua simpul lainnya dalam sebuah graf berbobot dengan bobot non-negatif. Algoritma ini bekerja dengan cara membangun sebuah himpunan simpul yang rute terpendeknya dari awal sudah diketahui. Pada setiap langkah, algoritma ini memilih simpul di luar himpunan yang memiliki jarak terpendek dari simpul awal, lalu memperbarui jarak ke tetangga-tetangganya. Karena Dijkstra secara sistematis mengeksplorasi graf berdasarkan jarak kumulatif dari titik awal, ia menjamin penemuan rute terpendek (optimal).

D. *Greedy Best First Search (GBFS)*

Greedy Best-First Search adalah algoritma pencarian yang mencoba menemukan solusi dengan cepat dengan cara selalu memilih langkah selanjutnya yang terlihat paling menjanjikan. "Menjanjikan" di sini diukur oleh sebuah fungsi heuristik, $h(n)$, yang mengestimasi biaya dari simpul saat ini n ke simpul tujuan. GBFS akan selalu mengeksplorasi simpul yang memiliki nilai heuristik terendah. Kelebihan utama GBFS adalah kecepatannya karena ia langsung menuju ke arah tujuan tanpa mempertimbangkan biaya perjalanan yang sudah ditempuh. Namun, kelemahan utamanya adalah algoritma ini tidak lengkap dan tidak optimal. Ia bisa terjebak dalam jalur buntu atau memilih rute yang secara kasat mata dekat, namun total jaraknya lebih panjang.

E. *Algoritma A**

Algoritma A* adalah algoritma pencarian yang menggabungkan keunggulan dari Algoritma Dijkstra dan Greedy Best-First Search. A* tidak hanya mempertimbangkan estimasi biaya ke tujuan (heuristik), tetapi juga biaya nyata yang telah dikeluarkan dari titik awal. A* mengevaluasi setiap simpul n menggunakan fungsi evaluasi:

$$f(n) = g(n) + h(n)$$

di mana:

- $g(n)$ adalah biaya (jarak) sebenarnya dari simpul awal ke simpul n. Ini adalah komponen yang digunakan oleh Dijkstra.
- $h(n)$ adalah estimasi biaya (heuristik) dari simpul n ke simpul tujuan. Ini adalah komponen yang digunakan oleh GBFS.

Dengan menyeimbangkan kedua nilai ini, A* dapat memandu pencariannya ke arah tujuan (seperti GBFS) sambil tetap memperhitungkan jalur yang sudah ditempuh (seperti Dijkstra). Jika fungsi heuristik $h(n)$ yang digunakan bersifat **admissible** (yaitu tidak pernah melebih-lebihkan biaya sebenarnya untuk mencapai tujuan), maka Algoritma A* dijamin akan menemukan rute terpendek (optimal) dan bersifat komplit

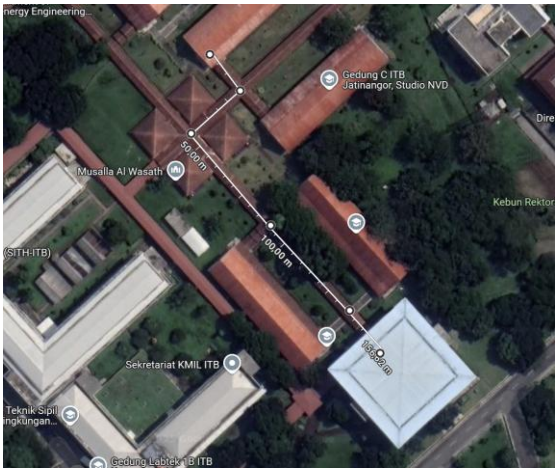
III. METODOLOGI PENELITIAN DAN IMPLEMENTASI

A. *Pengumpulan Data*

Tahap awal dalam penelitian ini adalah pengumpulan data jarak yang akurat untuk membangun model graf yang representatif. Untuk mencapai hal ini, pengumpulan data dilakukan dengan memanfaatkan platform Google Maps, khususnya menggunakan fitur "Ukur Jarak" (Measure Distance).

Proses ini melibatkan identifikasi 38 lokasi penting (simpul) di Kampus ITB Jatinangor, seperti gedung perkuliahan, fasilitas umum, dan persimpangan jalan. Kemudian, untuk 73 jalur penghubung (sisi) yang telah ditentukan mungkin ini belum mancakup semua jalur yang ada di ITB Jatinangor, tetapi semoga kedepannya aka nada optimasi untuk menelusuri lebih banyak lagi jalur.J arak aktual untuk pejalan kaki diukur dengan menelusuri titik

demi titik di sepanjang jalan setapak yang tersedia. Gambar di bawah menunjukkan contoh konkret dari proses pengukuran ini, di mana jarak dari Gedung A ke GKU 1 diukur secara segmental untuk mendapatkan total jarak yang presisi. Metode ini memastikan bahwa data yang digunakan bukanlah jarak garis lurus (Euclidean), melainkan jarak tempuh nyata yang akan dilalui oleh pejalan kaki. Data jarak dalam satuan meter yang diperoleh dari setiap pengukuran ini kemudian dijadikan sebagai nilai bobot (*weight)* untuk setiap sisi dalam model graf, memastikan bahwa simulasi algoritma berjalan di atas data yang mendekati kondisi dunia nyata.



Gambar 3. Pengukuran jarak aktual menggunakan Google Maps
Sumber : Screenshot dari Google Maps

B. Permodelan Graf

Fondasi dari sistem pencarian rute ini adalah representasi akurat dari peta Kampus ITB Jatinangor ke dalam sebuah struktur data graf. Untuk tujuan ini, sebuah kelas Graph khusus dibuat dalam file graph.py.

1. Inisialisasi Graf dari Sumber Data Eksternal. Graf tidak dibuat secara statis di dalam kode, melainkan dimuat dari sebuah file eksternal berformat JSON (map.json). Ini memungkinkan fleksibilitas di mana data peta (simpul dan jalur) dapat diubah atau diperbarui tanpa harus memodifikasi kode program inti.
2. Struktur Data Simpul (Nodes). Saat inisialisasi, kelas Graph membaca data nodes dari file JSON dan menyimpannya dalam sebuah *dictionary* Python. *Key* dari *dictionary* ini adalah ID unik dari setiap gedung (misal: "G1"), dan *value* adalah objek yang berisi informasi detail tentang simpul tersebut, termasuk nama gedung serta koordinat x dan y yang krusial untuk perhitungan heuristik.
3. Struktur Data Sisi (*Edges*) dan Representasi Adjacency List. Koneksi antar simpul direpresentasikan menggunakan Daftar Ketetanggaan (Adjacency List). Kelas Graph menggunakan collections.defaultdict(*list*) untuk menyimpan sisi-sisi. Untuk setiap data *edge* dalam file JSON, koneksi dua arah (*undirected*) secara otomatis dibuat, di mana jalur dari A ke B juga berarti ada jalur dari B ke A dengan bobot yang sama. Ini adalah model yang tepat untuk merepresentasikan jalur pejalan kaki yang umumnya dapat dilalui dari kedua arah.
4. Metode Antarmuka. Kelas Graph menyediakan metode-metode penting sebagai antarmuka bagi algoritma pencarian, yaitu neighbors(node) untuk mendapatkan semua tetangga yang terhubung langsung dengan sebuah simpul, dan get_coords(node) untuk mengambil koordinat geografis sebuah simpul.

C. Aplikasi Algoritma Dijkstra

Implementasi Algoritma Dijkstra terdapat dalam file dijkstra.py dan dirancang untuk menemukan rute dengan biaya terendah secara sistematis.

1. Struktur Data Inti. Algoritma ini memanfaatkan sebuah antrian prioritas (priority queue) yang diimplementasikan menggunakan pustaka heapq Python. Antrian ini menyimpan *tuple* (cost, node), memastikan bahwa simpul dengan jarak kumulatif (cost) terendah dari titik awal selalu diproses terlebih dahulu.
2. Proses Pencarian:
    a. Sebuah *dictionary* distance digunakan untuk mencatat jarak terpendek yang telah ditemukan dari simpul awal ke setiap simpul lainnya.
    b. Sebuah *dictionary* parent digunakan untuk melacak simpul sebelumnya dalam rute terpendek, yang nantinya digunakan untuk merekonstruksi jalur akhir.
    c. Algoritma beriterasi selama antrian prioritas tidak kosong. Pada setiap langkah, ia mengambil simpul dengan biaya terendah, lalu melakukan proses relaksasi: untuk setiap tetangga, algoritma akan menghitung apakah jalur melalui simpul saat ini lebih pendek daripada jalur yang tercatat sebelumnya. Jika ya, distance dan parent untuk tetangga tersebut akan diperbarui, dan tetangga itu dimasukkan ke dalam antrian prioritas.
3. Fungsi dijkstra mengembalikan tiga nilai. Rute terpendek dalam bentuk *list* simpul, total jarak (bobot) dari rute tersebut, dan jumlah total simpul yang telah dikunjungi (len(visited)), yang menjadi metrik penting untuk menganalisis efisiensi komputasi.

D. Aplikasi Algoritma Greedy Best First Search

GBFS diimplementasikan dalam file gbfs.py sebagai algoritma pencarian "serakah" yang terinformasi.

1. Prioritas Berbasis Heuristik. Perbedaan fundamental GBFS dengan Dijkstra terletak pada cara kerjanya. Antrian prioritas (heapq) pada GBFS tidak diurutkan berdasarkan biaya kumulatif, melainkan *hanya* oleh nilai heuristik. Heuristik yang digunakan adalah Jarak

Euclidean (jarak garis lurus) yang dihitung oleh fungsi euclidean_heuristic dari modul util.py.

2. Logika Pencarian. Algoritma ini selalu memilih untuk mengeksplorasi simpul yang secara geografis paling dekat dengan tujuan, mengabaikan bobot atau jarak tempuh aktual untuk mencapai simpul tersebut.
3. Kalkulasi Bobot Retroaktif. Sebuah detail implementasi yang penting adalah bahwa total biaya rute tidak dihitung selama proses pencarian. Sebaliknya, setelah jalur ditemukan dengan melacak parent, algoritma akan menelusuri kembali jalur tersebut untuk menjumlahkan bobot dari setiap sisi yang dilalui dan mendapatkan total jarak akhir. Ini menegaskan bahwa bobot sisi tidak mempengaruhi keputusan algoritma saat pencarian berlangsung.

*E. Aplikasi Algoritma A**

Algoritma A*, yang diimplementasikan dalam astar.py, dirancang untuk menjadi gabungan cerdas antara pendekatan Dijkstra dan GBFS.

1. Fungsi Evaluasi $f(n) = g(n) + h(n)$. A* menggunakan antrian prioritas di mana setiap elemen diurutkan berdasarkan nilai f_score.
    a. g_score $(g(n))$ adalah biaya atau jarak kumulatif dari simpul awal, serupa dengan yang digunakan Dijkstra. Nilai ini disimpan dalam *dictionary* g_score.
    b. Heuristik $(h(n))$ dihitung menggunakan fungsi euclidean_heuristic yang sama seperti pada GBFS.
    c. f_score adalah jumlah dari g_score dan nilai heuristik, yang menjadi nilai prioritas dalam heapq.
2. Proses Pencarian yang Terarah dan Optimal. Dengan mengevaluasi simpul berdasarkan f_score, A* mampu mengarahkan pencariannya ke tujuan (seperti GBFS) sambil tetap memperhitungkan jalur paling efisien yang telah ditempuh (seperti Dijkstra). Ini memungkinkannya untuk menghindari cabang-cabang pencarian yang tidak menjanjikan, sehingga secara signifikan lebih efisien daripada Dijkstra namun tetap menjamin hasil yang optimal.
3. Struktur dan Output. Struktur dasar, termasuk penggunaan parent untuk rekonstruksi jalur dan nilai kembalian (rute, jarak, simpul dikunjungi), sangat mirip dengan implementasi Dijkstra, menyoroti kesamaan fundamental mereka sebagai algoritma pencarian graf.

F. Visualisasi Rute dengan GUI

Untuk memudahkan pemahaman dan analisis hasil, sebuah modul visualisasi grafis sederhana dibuat dalam visualizer.py.

1. Pustaka yang Digunakan: Visualisasi ini dibangun di atas dua pustaka Python yaitu, networkx untuk pemodelan dan manipulasi struktur graf, serta matplotlib.pyplot untuk proses penggambaran (*plotting*).
2. Penggambaran Graf. Fungsi draw_graph pertama-tama membuat objek nx.Graph dari data yang ada. Posisi setiap simpul pada kanvas ditentukan oleh koordinat x dan y yang ada pada data, menciptakan representasi spasial yang akurat dari peta kampus.
3. Penyorotan Rute (Path Highlighting). Fitur kunci dari visualizer ini adalah kemampuannya untuk menyorot rute yang ditemukan. Jika sebuah argumen path diberikan kepada fungsi draw_graph, sisi-sisi yang merupakan bagian dari rute tersebut akan digambar ulang dengan warna yang kontras (merah) dan ketebalan yang lebih besar. Hal ini memberikan umpan balik visual yang instan dan jelas mengenai jalur yang dihasilkan oleh setiap algoritma.

G. Pengujian Program dan Analisis Hasil

Titik masuk utama dan orkestrasi keseluruhan program berada di dalam file main.py. Skrip ini bertanggung jawab untuk menjalankan alur kerja pengujian secara lengkap.

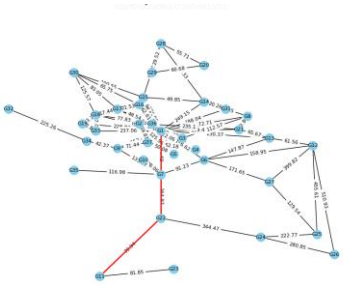
1. Alur Kerja:
    - Memuat data graf dari file map.json.
    - Meminta input dari pengguna untuk menentukan simpul awal (start) dan simpul tujuan (goal).
    - Menjalankan setiap algoritma (Dijkstra, A*, dan GBFS) untuk pasangan simpul yang sama.
2. Pengukuran Kinerja. Untuk setiap pemanggilan algoritma, skrip menggunakan pustaka time untuk mencatat waktu sebelum dan sesudah eksekusi. Selisih waktu ini kemudian dihitung untuk mendapatkan waktu eksekusi dalam milidetik, memberikan data kuantitatif untuk perbandingan kecepatan.
3. Output Komparatif. Hasil dari setiap algoritma termasuk jalur yang dilalui, jarak total, jumlah simpul yang dikunjungi, dan waktu eksekusi dicetak secara terstruktur ke terminal.
4. Pemanggilan Visualisasi. Setelah semua algoritma selesai dieksekusi, main.py memanggil fungsi draw_graph sebanyak tiga kali, masing-masing untuk setiap rute yang dihasilkan, sehingga menampilkan tiga jendela visualisasi yang berbeda untuk perbandingan visual.

1. Pengujian 1: Gerbang Utama ke Gedung A

```
Masukkan ID gedung awal (misal: G1): G11
Masukkan ID gedung tujuan (misal: G3): G1

=== Dijkstra ===
Jalur: G11 -> G22 -> G7 -> G6 -> G1
Jarak total: 492.76
Node dikunjungi: 12
Waktu: 0.14 ms

=== A* ===
Jalur: G11 -> G22 -> G7 -> G6 -> G1
Jarak total: 492.76
Node dikunjungi: 12
Waktu: 0.09 ms

=== Greedy Best First Search (GBFS) ===
Jalur: G11 -> G22 -> G7 -> G1
Jarak total: 507.40000000000003
Node dikunjungi: 3
Waktu: 0.04 ms
```

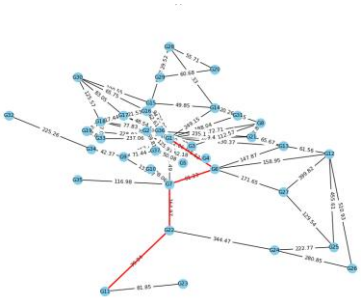
Gambar 4. Input, Hasil Jarak ,dan Waktu Eksekusi Pencarian 1. G11 (Gerbang Utama) – G1 (Gedung A)

Sumber : Dokumen pribadi penulis



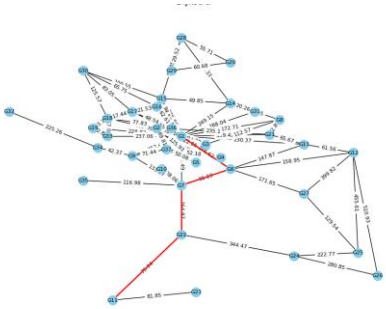
Gambar 5. Hasil GUI Algoritma GBFS

Sumber : Dokumen pribadi penulis



Gambar 6. Hasil GUI Algoritma A*

Sumber : Dokumen pribadi penulis



Gambar 7. Hasil GUI Algoritma Dijkstra

Sumber : Dokumen pribadi penulis

Untuk memvalidasi dan membandingkan performa dari ketiga algoritma, sebuah skenario pengujian spesifik dijalankan. Program diinstruksikan untuk mencari rute terpendek dari simpul G11 (Gerbang Depan) menuju simpul G1 (Gedung A).

1. Analisis Algoritma Optimal: Dijkstra dan A*

Berdasarkan hasil eksekusi, kedua algoritma yang dirancang untuk menjamin optimalitas, yaitu Dijkstra dan A*, berhasil menemukan rute yang identik dan optimal.

- Kedua algoritma menghasilkan rute G11 -> G22 -> G7 -> G6 -> G1. Rute ini secara visual terkonfirmasi pada gambar visualisasi masing-masing algoritma, di mana jalur yang ditandai dengan warna merah menunjukkan alur yang sama persis.
- Konsisten dengan rute yang sama, total jarak yang dihitung oleh kedua algoritma adalah 492.76 meter. Hal ini membuktikan keandalan dan akurasi kedua algoritma dalam menemukan jalur dengan total bobot terendah berdasarkan data graf yang ada.
- Terdapat perbedaan kinerja minor di mana A* menyelesaikan pencarian sedikit lebih cepat (0.09 ms) dibandingkan dengan Dijkstra (0.14 ms). Simpul yang Dikunjungi, Menariknya, dalam skenario pengujian ini, kedua algoritma mengunjungi jumlah simpul yang sama, yaitu 12 node. Hal ini dapat terjadi pada rute yang relatif tidak bercabang atau lurus, di mana panduan heuristik A* tidak memberikan keuntungan signifikan dalam memangkas cabang pencarian karena jalur optimal sudah menjadi pilihan yang paling jelas sejak awal. Meskipun demikian, efisiensi waktu A* yang sedikit lebih unggul menunjukkan overhead komputasi yang lebih rendah dalam setiap langkahnya.

2. Analisis Algoritma Heuristik: Greedy Best-First Search (GBFS)

Hasil pengujian GBFS secara jelas mendemonstrasikan karakteristik dan kelemahan fundamental dari pendekatan yang "serakah".

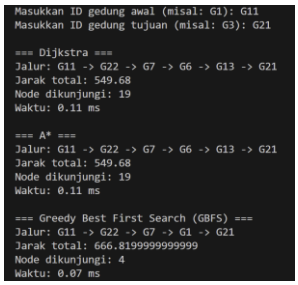
- GBFS menemukan rute yang berbeda, yaitu G11 -> G22 -> G7 -> G1. Terlihat bahwa algoritma ini mengambil "jalan pintas" dari simpul G7 langsung ke G1, tanpa melalui G6.
- Rute yang dihasilkan oleh GBFS memiliki total jarak 507.40 meter. Jarak ini lebih panjang sekitar 14.64 meter dibandingkan rute optimal yang ditemukan oleh Dijkstra dan A*. Ini adalah bukti konkret bahwa GBFS tidak menjamin optimalitas.
- Kegagalan GBFS menemukan rute optimal terjadi pada simpul G7. Pada titik ini, algoritma harus memilih antara bergerak ke G6 atau langsung ke G1. Karena GBFS hanya mempertimbangkan nilai heuristik (estimasi jarak lurus ke tujuan), simpul G1 kemungkinan besar tampak lebih "menjanjikan" dari G7. Keputusan ini mengabaikan fakta bahwa jalur melalui G6 (G7 ->

G6 -> G1) memiliki total bobot aktual yang lebih rendah daripada jalur langsung (G7 -> G1).

- Di sisi lain, kelemahan GBFS diimbangi oleh kecepatannya yang superior. Algoritma ini hanya membutuhkan 0.04 ms untuk menemukan solusi dan hanya mengunjungi 3 simpul. Ini menjadikannya algoritma tercepat di antara ketiganya, namun dengan mengorbankan akurasi.

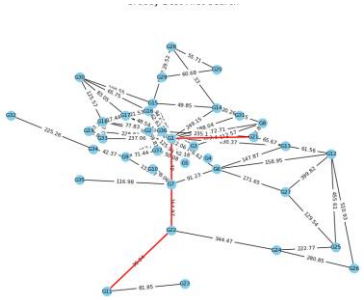
Dari analisis ini, dapat disimpulkan bahwa hasil pengujian secara empiris memvalidasi karakteristik teoretis dari setiap algoritma. Dijkstra dan A* terbukti andal dalam menemukan rute terpendek, dengan A* menunjukkan potensi efisiensi yang lebih tinggi. Sebaliknya, GBFS menunjukkan dirinya sebagai algoritma yang sangat cepat namun tidak dapat diandalkan untuk aplikasi yang membutuhkan jaminan optimalitas, karena sifat "serakah"-nya yang rentan menghasilkan solusi sub-optima

2. Pengujian 2: Gerbang Utama ke Gedung KOICA



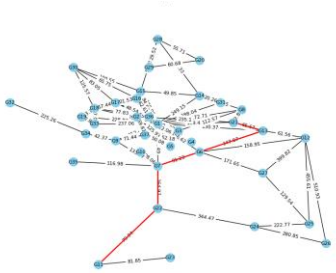
Gambar 8. Input, Hasil Jarak ,dan Waktu Eksekusi Pencarian 2. G11 (Gerbang Utama) – G21 (Gedung KOICA)

Sumber : Dokumen pribadi penulis



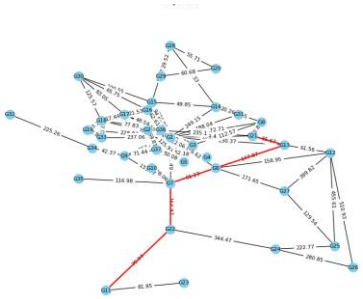
Gambar 9. **Hasil GUI Algoritma GBFS**

Sumber : Dokumen pribadi penulis



Gambar 10. **Hasil GUI Algoritma A***

Sumber : Dokumen pribadi penulis



Gambar 11. Hasil GUI Algoritma Dijkstra

Sumber : Dokumen pribadi penulis

Pengujian kedua dilakukan untuk mengevaluasi kemampuan algoritma dalam menemukan rute dari G11 (Gerbang Depan) ke G21 (Gedung KOICA). Skenario ini melibatkan rute yang sedikit lebih jauh dan kompleks dibandingkan skenario pertama.

1. Analisis Algoritma Optimal: Dijkstra dan A*

- Kedua algoritma menemukan rute yang sama persis, yaitu G11 -> G22 -> G7 -> G6 -> G13 -> G21. Visualisasi rute pada Gambar 10 dan Gambar 11 mengonfirmasi alur jalur ini. Total jarak yang ditempuh untuk rute optimal ini adalah 549.68 meter.
- Dalam skenario ini, kinerja kedua algoritma tercatat identik. Keduanya mengunjungi 19 node dan mencatatkan waktu eksekusi 0.11 ms. Hal ini menunjukkan bahwa untuk jalur ini, panduan heuristik A* tidak memberikan keuntungan signifikan dalam mengurangi jumlah simpul yang perlu dieksplorasi dibandingkan dengan pencarian berbasis biaya murni oleh Dijkstra.

2. Analisis Algoritma Heuristik: Greedy Best-First Search (GBFS)

- Algoritma ini menghasilkan rute yang berbeda, yaitu G11 -> G22 -> G7 -> G1 -> G21, yang dapat dilihat pada Gambar 9. Rute ini memiliki total jarak 666.82 meter, yang secara signifikan lebih panjang sekitar 117 meter dari rute optimal.
- Titik di mana GBFS menyimpang dari jalur optimal adalah pada simpul G7. Alih-alih melanjutkan ke G6 lalu ke G13 sesuai rute terpendek, GBFS memilih untuk berbelok melalui G1. Keputusan "serakah" ini kemungkinan besar diambil karena dari G7, simpul G1 memiliki nilai heuristik (jarak lurus ke G21) yang lebih rendah dibandingkan simpul G6, sehingga G1 tampak sebagai pilihan yang lebih cepat. Namun, keputusan ini mengabaikan bobot sisi yang lebih besar pada jalur tersebut, yang pada akhirnya menghasilkan rute keseluruhan yang lebih panjang.
- Kecepatan tetap menjadi keunggulan GBFS. Algoritma ini hanya mengunjungi 4 node dan selesai dalam waktu 0.07 ms, menjadikannya yang tercepat di antara ketiganya.

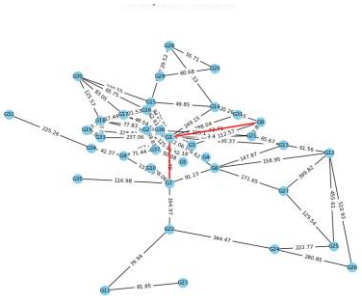
Pengujian skenario kedua ini semakin memperkuat kesimpulan dari analisis sebelumnya. Dijkstra dan A* secara konsisten memberikan jaminan optimalitas, yang merupakan syarat mutlak untuk sebuah sistem rekomendasi rute yang andal. Sementara itu, GBFS, meskipun sangat cepat, terbukti tidak dapat diandalkan karena kecenderungannya untuk menghasilkan rute yang lebih panjang secara signifikan.

3. Pengujian 3: Gerbang Utama ke Gedung KOICA

```
Masukkan ID gedung awal (misal: G1): G7
Masukkan ID gedung tujuan (misal: G3): G8

=== Dijkstra ===
Jalur: G7 -> G6 -> G13 -> G21 -> G8
Jarak total: 327.6
Node dikunjungi: 19
Waktu: 0.12 ms

=== A* ===
Jalur: G7 -> G6 -> G13 -> G21 -> G8
Jarak total: 327.6
Node dikunjungi: 18
Waktu: 0.12 ms

=== Greedy Best First Search (GBFS) ===
Jalur: G7 -> G1 -> G8
Jarak total: 435.20000000000005
Node dikunjungi: 2
Waktu: 0.05 ms
```

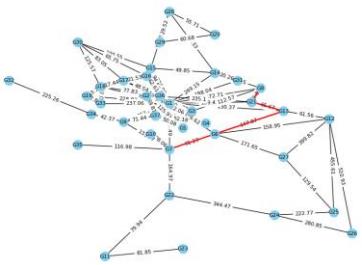
Gambar 12. Input, Hasil Jarak ,dan Waktu Eksekusi Pencarian 3. G7 (GKU 2) – G8 (GKU 3)

Sumber : Dokumen pribadi penulis



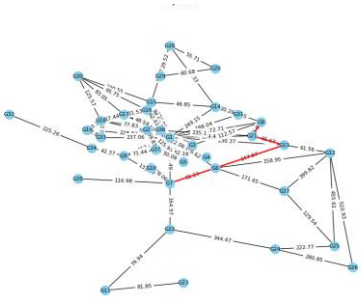
Gambar 13. Hasil GUI Algoritma GBFS

Sumber : Dokumen pribadi penulis



Gambar 14. Hasil GUI Algoritma A*

Sumber : Dokumen pribadi penulis



Gambar 15. Hasil GUI Algoritma Dijkstra

Sumber : Dokumen pribadi penulis

Pengujian ini dirancang untuk mengevaluasi performa algoritma pada rute jarak pendek antar gedung dalam satu kompleks yang sama, yaitu dari G7 (GKU 2) ke G8 (GKU 3). Skenario ini menarik karena meskipun secara geografis berdekatan, rute optimalnya mungkin tidak intuitif.

1. Analisis Algoritma Optimal: Dijkstra dan A*

Dijkstra dan A* sekali lagi membuktikan keandalannya dalam menemukan rute terpendek yang sebenarnya, meskipun rute tersebut tidak lurus.

- Kedua algoritma berhasil mengidentifikasi rute optimal yang sama: G7 -> G6 -> G13 -> G21 -> G8. Rute ini menunjukkan bahwa untuk berpindah antar GKU, jalur terpendek melibatkan jalan memutar melalui area Gedung Rektorat (G13) dan Gedung KOICA (G21). Jarak total untuk rute ini adalah 327.6 meter.
- Pada skenario ini, keunggulan efisiensi A* mulai terlihat lebih jelas. Simpul yang Dikunjungi A* hanya perlu 18 node untuk menemukan solusi, sedangkan Dijkstra harus mengunjungi 19 node. Ini adalah demonstrasi klasik dari kemampuan heuristik A* untuk memangkas satu atau lebih cabang pencarian yang tidak perlu, yang tetap harus dieksplorasi oleh Dijkstra.Waktu Eksekusi yang meskipun perbedaan jumlah simpul yang dikunjungi kecil, waktu eksekusi keduanya tercatat sama yaitu 0.12 ms.

2. Analisis Algoritma Heuristik: Greedy Best-First Search (GBFS)

- GBFS menghasilkan rute G7 -> G1 -> G8, yang secara visual dapat dilihat pada Gambar 13. Rute ini memiliki total jarak 435.2 meter, atau lebih dari 107 meter lebih panjang dari rute optimal.
- Kegagalan ini menyoroti kelemahan terbesar dari heuristik Jarak Euclidean. Dari simpul G7, simpul G1 secara visual (garis lurus) jauh lebih dekat ke tujuan akhir (G8) daripada simpul G6. Akibatnya, GBFS dengan "serakah" memilih jalur melalui G1, mengabaikan fakta bahwa jalur tersebut memiliki bobot yang jauh lebih besar dan tidak efisien dibandingkan rute memutar melalui G6 dan G13.

- Seperti biasa, GBFS adalah yang tercepat secara absolut. Ia hanya mengunjungi 2 node dan selesai dalam 0.05 ms. Namun, kecepatan ini dicapai dengan menghasilkan solusi yang kualitasnya sangat buruk.

Skenario pengujian ketiga ini memberikan bukti paling kuat mengenai superiotas A* dalam aplikasi praktis. Ia tidak hanya menjamin rute optimal pada kasus yang tidak intuitif, tetapi juga melakukannya dengan lebih efisien dibandingkan Dijkstra. Kasus ini juga secara definitif menunjukkan bahwa GBFS tidak cocok untuk aplikasi navigasi yang andal karena kecenderungannya untuk "tertipu" oleh heuristik, yang menghasilkan rute yang salah secara signifikan.

IV. KESIMPULAN

Berdasarkan implementasi, pengujian, dan analisis komparatif yang telah dilakukan terhadap tiga algoritma pencarian rute pada model graf Kampus ITB Jatinangor, dapat ditarik beberapa kesimpulan fundamental sebagai berikut:

1. Validitas Model dan Implementasi. Model graf yang direpresentasikan menggunakan kelas Graph dan dimuat dari sumber data JSON terbukti menjadi pendekatan yang fleksibel dan efektif. Implementasi dari ketiga algoritma Dijkstra, A*, dan GBFS berhasil dijalankan dan menunjukkan perilaku yang konsisten dengan prinsip teoretis masing-masing.
2. Kinerja dan Jaminan Optimalitas. Dijkstra dan A*, Kedua algoritma ini secara konsisten berhasil menemukan rute terpendek yang sebenarnya (optimal) di semua skenario pengujian. Keduanya memberikan jaminan keandalan yang mutlak, yang merupakan syarat esensial untuk sebuah sistem navigasi.
3. Perbandingan Efisiensi A* dan Dijkstra. Meskipun keduanya optimal, A* menunjukkan keunggulan dalam efisiensi. Pada skenario ketiga, A* mampu menemukan solusi dengan mengunjungi jumlah simpul yang lebih sedikit (18 node) dibandingkan Dijkstra (19 node). Ini membuktikan bahwa panduan heuristik mampu mengurangi beban komputasi tanpa mengorbankan kualitas solusi.
4. Greedy Best-First Search (GBFS). Algoritma ini secara konsisten menjadi yang tercepat dan menjelajahi jumlah simpul paling sedikit di semua skenario. Namun, kecepatan ini harus dibayar dengan harga yang sangat mahal: GBFS terbukti tidak dapat diandalkan dan secara konsisten menghasilkan rute sub-optimal yang jaraknya jauh lebih panjang daripada rute sebenarnya.
5. Rekomendasi Algoritma Terbaik. Dengan mempertimbangkan *trade-off* antara kecepatan, penggunaan sumber daya, dan keandalan, Algoritma A* secara konklusif direkomendasikan sebagai metode yang paling superior untuk aplikasi sistem rekomendasi rute pejalan kaki. A* menawarkan keseimbangan terbaik, yaitu kecepatan yang jauh lebih baik dari Dijkstra dengan tetap mempertahankan jaminan optimalitas yang tidak dimiliki oleh GBFS.

TAUTAN REPOSITORY GITHUB

Akses program melalui pranala berikut : https://github.com/iannn23/ITB-Route.git

UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa. Karena berkat, Rahmat dan karunia-Nya sehingga penulisa dapan menyelesaikan makalah yang berjudul "Implementasi dan Analisis Algoritma Dijkstra, Greedy Best First Search, dan A* dalam Sistem Rekomendasi Rute Terpendek Pejalan Kaki di Kampus ITB Jatinangor". Dalam penyusunan makalah ini, penulis tidak luput dari berbagai kesulitan dan hambatan, namun atas bantuan dan dorongan dari berbagai pihak akhirnya makalah ini dapat terselesaikan. Untuk itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya dan penghargaan yang setinggi-tingginya kepada semua pihak yang telah membantu serta mendukung penulis dalam menyusun dan menyelesaikan makalah ini, yaitu kepada :

1. Bapak Dr. Rinaldi Munir, dan Bapak Arrival Dwi Sentosa, M.T, sebagai dosen pengajar mata kuliah IF1220 Matematika Diskrit atas pengajaran materi-materi yang telah dibagikan di kelas Teknik Informatika Semester II Tahun 2024/2025.
2. Para penulis yang telah menciptakan karya-karya yang menjadi landasan bagi penulisan makalah ini. Referensi dari jurnal dan artikel-artikel telah memberikan wawasan dan kontribusi penting pada pemahaman topik.
3. Untuk semua pihak yang tidak dapat disebutkan satu persatu, yang secara langsung maupun tidak langsung telah membantu untuk menyelesaikan makalah ini.

Sekali lagi penulis sampaikan terima kasih atas segala bantuan dan dukungan yang ada. Makalah ini tidak mungkin terwujud tanpa dukungan dan kontribusi dari setiap individu yang disebutkan di atas. Semoga makalah ini dapat menjadi manfaat bagi orang lain yang membacanya.

REFERENCES

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.

[3] Maulidevi, N. U. "*Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search*". Program Studi Teknik Informatika, STEI-ITB 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf

[4] Maulidevi, N. U., & Munir, R. "*Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A**". Program Studi Teknik Informatika, STEI-ITB 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf

[5] Munir, R. "*Graf (Bagian 1)*". Program Studi Teknik Informatika, STEI-ITB 2024.

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

[6] Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

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.

Jatinangor, 20 Juni 2025

Sebastian Enrico Nathanael
13523134