Studi Implementasi BFS dan DFS dalam Penentuan Jalur Ambulans pada Simulasi Peta Kota Bandung ke Rumah Sakit Terdekat

Jonathan Kenan Budianto - 13523139
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: kenbud2001@gmail.com , 13523139@std.stei.itb.ac.id

Abstrak—Penelitian ini mengevaluasi kinerja dua algoritma graf dasar Breadth-First Search (BFS) dan Depth-First Search (DFS) dalam menentukan rute ambulans menuju rumah sakit terdekat pada jaringan jalan Kota Bandung. Data jalan diambil dari OpenStreetMap melalui pustaka osmnx kemudian diekspor ke format JSON dan diproses menjadi graf tak berarah dengan NetworkX. Daftar 70+ rumah sakit di Bandung dipetakan ke simpul-simpul graf menggunakan struktur spasial KD-tree dan ambang jarak 100 m untuk mengidentifikasi simpul "dekat RS." Selanjutnya BFS dan DFS diimplementasikan untuk menelusuri semua jalur dari simpul ambulans acak ke simpul rumah sakit, merekam hop count, panjang jalur, dan waktu eksekusi. Hasil eksperimen untuk N posisi ambulans acak menunjukkan bahwa BFS selalu menemukan jalur dengan hop count minimal dan jumlah simpul jauh lebih sedikit dibandingkan DFS, serta mengeksekusi pencarian dengan waktu rata-rata lebih singkat. Analisis statistik dan visualisasi jalur (simpul biru: semua simpul dekat RS; hijau: ambulans; merah: tujuan; garis hijau: rute) menegaskan keunggulan BFS untuk aplikasi routing ambulans di graf tidak berbobot .

Keywords—Breadth-First Search (BFS), Depth-First Search (DFS), routing ambulans, jaringan jalan Kota Bandung, OpenStreetMap, KD-tree, hop count.

I. Pendahuluan

Insiden darurat medis bersifat tidak terduga dan sering kali menuntut penanganan yang cepat serta tepat. Ambulans sebagai sarana utama dalam pelayanan gawat darurat memiliki peranan vital untuk menyelamatkan nyawa pasien. Kecepatan respons ambulans sangat dipengaruhi oleh efektivitas pemilihan rute menuju rumah sakit terdekat. Di kawasan perkotaan seperti Bandung, kompleksitas jaringan jalan yang meliputi ruas jalan utama dan jalan perkampungan menimbulkan tantangan tersendiri dalam menentukan jalur tercepat. Oleh karena itu, pemodelan jaringan jalan sebagai graf dan penerapan algoritma pencarian lintasan (pathfinding) menjadi langkah awal yang krusial untuk merancang sistem penentuan rute ambulans secara optimal.

Algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) merupakan dua metode dasar graf yang banyak digunakan dalam pencarian jalur. BFS bekerja dengan cara menjelajahi simpul graf secara berlapis (level by level), sehingga cocok untuk menemukan lintasan terpendek pada graf tidak berbobot. Sementara itu, DFS melakukan penelusuran sedalam mungkin sebelum kembali dan mengeksplorasi cabang lain, yang meski tidak selalu menghasilkan lintasan terpendek, menawarkan pendekatan sistematis dalam menjelajahi keseluruhan struktur graf. Pada konteks rute ambulans, implementasi BFS memungkinkan identifikasi rumah sakit terdekat berdasarkan jumlah edge (ruas jalan) yang dilalui, sedangkan DFS dapat digunakan untuk memeriksa alternatif rute dan memverifikasi keberadaan jalur menuju titik tujuan di dalam jaringan yang luas.

Makalah ini bertujuan untuk: Memodelkan jaringan jalan Kota Bandung sebagai graf tak berbobot, di mana simpul (vertex) merepresentasikan persimpangan atau titik tertentu (misalnya rumah sakit, lokasi kejadian), dan edge (sisi) merepresentasikan koneksi jalan antar simpul. Mengimplementasikan algoritma BFS untuk menentukan rute terpendek dari lokasi ambulans ke rumah sakit terdekat, sekaligus membandingkan hasilnya dengan rute yang ditemukan oleh algoritma DFS. Menganalisis kompleksitas waktu serta kelebihan dan kekurangan masing-masing algoritma dalam konteks penentuan rute ambulans di jaringan jalan kota.

II. Teori Dasar

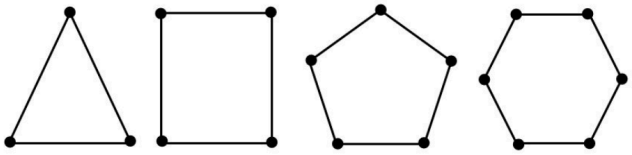
A. Graf

Graf merupakan struktur data yang tersusun dari kumpulan simpul (nodes atau vertices) dan sisi (edges) yang menghubungkan antar simpul. Struktur ini memfasilitasi pemodelan objek–objek diskrit beserta relasi di antara mereka. Secara formal, graf dilambangkan sebagai G = (V,E), di mana V adalah himpunan tidak kosong yang memuat semua simpul, dan E adalah himpunan sisi yang menunjukkan pasangan simpul mana saja yang saling terhubung.

Berdasarkan keberadaan loop (sisi yang kembali ke simpul asal) atau tepi paralel (beberapa sisi menghubungkan dua simpul yang sama), graf dibagi menjadi dua kategori utama:

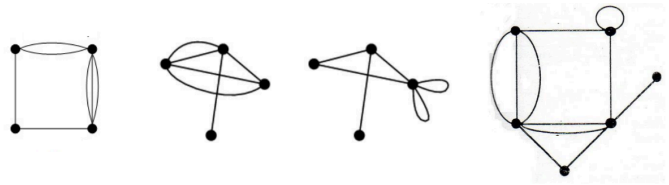
Graf sederhana, tidak mengandung loop maupun tepi paralel. Artinya, setiap pasangan simpul dihubungkan paling

satu sisi, dan tidak ada sisi yang bersarang kembali ke simpul itu sendiri.



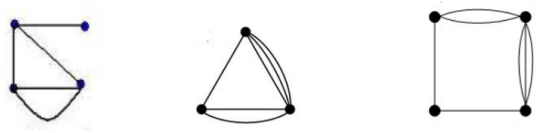
Gambar 2.1 Graf Sederhana

Graf non-sederhana memiliki minimal satu loop atau tepi paralel, dan dapat diklasifikasikan lebih lanjut menjadi:



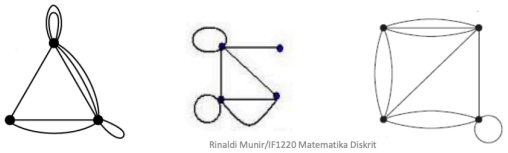
Gambar 2.2 Graf Non-Sederhana

1. Graf ganda (Multigraf) , yaitu graf yang memperbolehkan lebih dari satu sisi di antara dua simpul yang sama.



Gambar 2.3 Graf Ganda

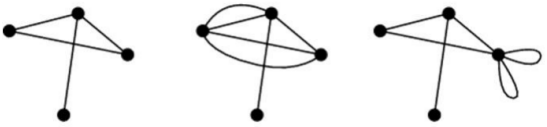
2. Graf semu (Pseudograf), yaitu graf yang mengizinkan loop, di mana sebuah sisi dapat bermula dan berakhir pada simpul yang identik.



Gambar 2.4 Graf Semu

Berdasarkan ada tidaknya arah pada setiap sisi, graf dapat dibagi menjadi dua kategori:

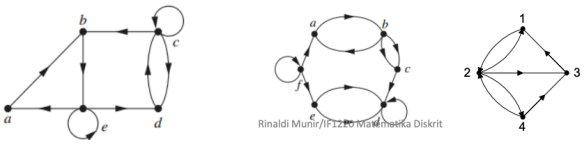
1. Graf tak-berarah (undirected graph)
   Pada jenis ini, sisi-sisinya tidak memiliki penunjuk arah, sehingga hubungan antara dua simpul bersifat timbal balik tanpa pembeda sumber atau tujuan.



Gambar 2.5 Graf Tak-Berarah

2. Graf berarah (directed graph atau digraph)
   Di sini, setiap sisi dilengkapi dengan panah yang menetapkan arah hubungan, menunjukkan simpul asal dan simpul tujuan secara eksplisit

.



Gambar 2.6 Graf Berarah

Terdapat 12 terminologi di dalam teori graf, yaitu :

1. Ketetanggaan (Adjacent)

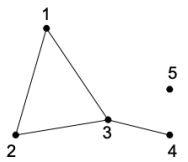
   Dua simpul disebut bertetangga apabila terdapat sisi langsung yang menghubungkan keduanya. Dengan kata lain, simpul u dan v saling bertetangga jika sisi (u,v) ada dalam himpunan sisi.

2. Bersisian (Incidency)

   Sebuah sisi dikatakan bersisian dengan simpul apabila simpul tersebut merupakan salah satu ujung sisi. Misalnya, sisi (u,v) bersisian dengan simpul u dan simpul v.

3. Simpul terpencil (Isolated vertex)

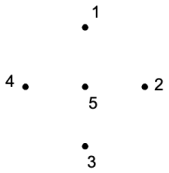
   Simpul yang tidak memiliki sisi sama sekali disebut simpul terpencil. Artinya, tidak ada sisi yang bersisian dengannya.



Gambar 2.7 Simpul Terpencil

4. Graf kosong (Null graph atau empty graph)

   Graf yang himpunan sisinya kosong tetapi memiliki satu atau lebih simpul. Semua simpul pada graf kosong bersifat terpencil karena tidak ada sisi.



Gambar 2.8 Graf Kosong

5. Derajat (Degree)

   Derajat sebuah simpul adalah jumlah sisi yang bersisian dengannya. Pada graf tak berarah, derajat simpul u sama dengan banyaknya sisi yang menghubungkan u. Pada graf berarah terdapat derajat masuk dan derajat keluar.

6. Lintasan (Path)

   Urutan simpul dan sisi yang menghubungkan simpul awal ke simpul akhir tanpa mengulang sisi. Panjang lintasan dihitung berdasarkan jumlah sisi yang dilalui.

7. Siklus atau sirkuit (Cycle atau circuit)

   Lintasan yang titik awal dan titik akhirnya sama disebut siklus atau sirkuit. Pada siklus, selain titik awal dan akhir yang sama, simpul lain dan sisi tidak diulang.

8. Keterhubungan (Connected)

   Graf tak berarah disebut terhubung jika setiap pasangan simpul dapat dihubungkan oleh lintasan. Graf berarah dapat bersifat kuat terhubung (strongly connected) jika ada lintasan bolak-balik antara setiap pasangan simpul, atau lemah terhubung (weakly connected) jika graf menjadi terhubung saat arah tepi diabaikan.

9. Upagraf dan komplemen upagraf (Subgraph dan complement subgraph)

   Upagraf adalah graf yang simpul dan sisinya merupakan bagian dari graf asal. Komplemen upagraf memuat semua simpul graf asal tetapi hanya sisi yang tidak ada pada upagraf.

10. Upagraf merentang (Spanning subgraph)

    Upagraf yang mencakup semua simpul graf asal disebut upagraf merentang. Sisi pada upagraf ini bisa berkurang, tetapi setiap simpul tetap ada.

11. Cut set

    Himpunan sisi yang penghapusannya akan memisahkan graf menjadi dua bagian atau lebih. Setiap cut set minimal memisahkan graf menjadi tepat dua komponen.

12. Graf berbobot (Weighted graph)

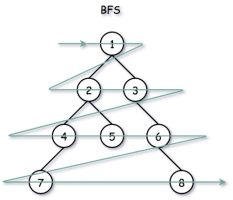
    Graf di mana setiap sisi memiliki nilai numerik yang disebut bobot. Bobot ini bisa merepresentasikan jarak, biaya, atau waktu dan menjadi dasar perhitungan jalur terpendek seperti pada algoritma Dijkstra.

B. Breadth-First Search (BFS)

BFS adalah algoritma pencarian pada graf yang menjelajahi semua simpul sedalam d level sebelum beralih ke level berikutnya. Dengan kata lain, BFS mengunjungi simpul bertetangga (neighbor) dari simpul awal secara berlapis (level by level), sehingga setiap simpul pada jarak d (dalam satuan jumlah sisi) dari sumber (source) akan diproses sebelum simpul pada jarak d + 1.

Steps of Breadth-First Search

1. Mulai dari satu simpul sumber, kemudian menjelajahi tingkat (level) demi tingkat.
2. Pertama, semua tetangga langsung (jarak satu sisi) dari simpul sumber dikunjungi.
3. Setelah itu, tetangga dari tiap simpul pada tingkat satu (jarak dua sisi) baru diproses, dan seterusnya.
4. Penggunaan struktur data: antrian untuk menyimpan simpul-simpul yang menunggu diproses.
5. Setiap simpul yang diambil dari antrian akan memeriksa semua tetangganya; tetangga yang belum dikunjungi akan langsung ditandai dan dimasukkan ke antrian.
6. Karena urutan kunjungan berdasarkan level, BFS menjamin menemukan lintasan terpendek (dalam jumlah sisi) pada graf tak berbobot.
7. Kompleksitas waktu: O(|V| + |E|) (|V| = jumlah simpul, |E| = jumlah sisi).



Gambar 2.9 *BFS*

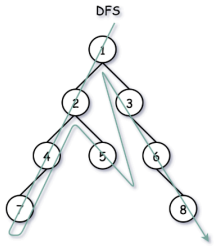
C. Depth-First Search (DFS)

DFS adalah algoritma pencarian pada graf yang menjelajahi sedalam mungkin setiap cabang sebelum mundur (backtracking). Dengan kata lain, algoritma akan mengikuti satu lintasan (path) dari simpul awal hingga tidak ada tetangga baru yang dapat dikunjungi, baru kemudian "kembali" (backtrack) ke simpul sebelumnya untuk mengeksplorasi cabang lain. DFS menekankan eksplorasi ke "kedalaman"

(depth) graf, berbeda dengan BFS yang fokus ke “lebar” (breadth).

Steps of *Depth-First Search*

1. Mulai dari satu simpul sumber, kemudian menjelajahi sedalam mungkin ke satu jalur sebelum mundur (backtrack).
2. Dari simpul awal, pilih satu tetangga yang belum dikunjungi, lalu lanjutkan ke tetangga berikutnya hingga mencapai simpul yang tidak memiliki tetangga baru.
3. Setelah mentok, “kembali” ke simpul sebelumnya untuk mencari tetangga lain yang belum dikunjungi, dan proses ini diulang hingga semua simpul yang dapat dicapai telah dieksplorasi.
4. Penggunaan struktur data: rekursi (stack pemanggilan fungsi) atau tumpukan (stack) eksplisit untuk melacak jalur yang sedang dikerjakan.
5. Karena fokus pada satu jalur hingga ujung, DFS tidak menjamin lintasan terpendek dalam graf tak berbobot.
6. Kompleksitas waktu: O(|V| + |E|), sama dengan BFS, selama representasi graf berupa adjacency list.



Gambar 2.10 *DFS*

III. Metode Penelitian

Gambar 3.1 *Peta Bandung*

List rumah sakit yang ada di Kota Bandung adalah RS DR. HASAN SADIKIN, RSUD KOTA BANDUNG, RS SARTIKA ASIH, RS SARININGSIH, RS DR. SALAMUN, RS PINDAD, RS MUHAMMADIYAH, RS AL-ISLAM, RS ST. BORROMEUS, RS BUNGSU, RS RAJAWALI, RS ST. YUSUF, RS IMMANUEL, RS KEBONJATI, RS ADVENT, RS HERMINA ARCAMANIK, RS SANTOSA HOSPITAL, RS EDELWEISS, RS KARTINI, RS MATA CICENDO, RS ISLAM DR. CHASANI – BSD, RS SARI ASIH CILEUNYI, RS SAHABAT MEDIKA, RS MITRA PLUMBON, RS BIBIR TIMUR, RS ANANDA CIGUGUR, RS ANANDYA – LEBAK SILIWANGI, RS MARTHALAND, RS PANGLIMA SEBRANG, RS FLAMBERT, RS ST. CAROLUS – HOLISTIK, RS SITI HAWA, RS GUNUNG EMAS, RS PERTAMINA – KEDONGKALANG, RS HARAPAN KITA, RS BHAKTI HUSADA, RS PRIMA MEDIKA, RS DHARMA HUSADA, RS BUNDA CIRAGA, RS CIKARANG BAHAGIA, RS MITRA KELUARGA CIBABAT, RS BUNDA BONGKAR, RS UMUM SULTAN AGUNG, RS AL AMIN CILAMPEN, RS HOLISTIK ISLAM, RS NUR HUSADA KHASANAH, RS GINJAL H A HABIBIE, RSIA HERMINA PASTEUR, RSIA MELINDA, RS GIGI DAN MULUT KOTA BANDUNG, RS GIGI DAN MULUT UNPAD, RSIA HUMANA PRIMA, RS GIGI DAN MULUT MARANATHA, RS BEDAH MELINDA 2, RSIA HARAPAN BUNDA, RS MATA BANDUNG EYE CENTER, RSIA AL-ISLAM, RSK PARAMARTA, RSU PASUNDAN, RSKJP MELINDA CVC, RSU MURNI TEGUH, RS MAYAPADA.

```python
import osmnx as ox
import networkx as nx
import json

def main():
    place_name = "Bandung, Indonesia"

    G = ox.graph_from_place(place_name, network_type="drive")

    data = nx.node_link_data(G)

    for link in data["links"]:
        if "geometry" in link:
            del link["geometry"]

    out_filepath = "graf_bandung.json"
    with open(out_filepath, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)

    print(f"✅ Selesai: graf tanpa geometri disimpan di {out_filepath}")

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

Gambar 3.2 *Implementasi Program Pengambilan Data Jalan Kota bandung*

Skrip export_osm_graph_to_json.py secara otomatis mengambil graf jalan tipe “drive” untuk wilayah “Bandung, Indonesia” langsung dari OpenStreetMap menggunakan pustaka osmnx. Setelah objek NetworkX terbentuk, fungsi nx.node_link_data mengonversi struktur simpul-edge menjadi format JSON yang mudah diolah, lalu atribut geometri pada setiap edge dihapus untuk memangkas ukuran berkas. Terakhir, data disimpan ke graf_bandung.json dengan

encoding UTF-8 dan indentasi rapi, menghasilkan file ringkas yang siap untuk analisis graf atau sebagai input visualisasi peta.

Gambar 3.3 *Implementasi Program Pengambilan Data Jalan Kota bandung*

Sedangkan plot_graf_bandung.py bertugas memvisualisasikan jaringan jalan Bandung yang telah diekspor. Skrip ini membuka kembali graf_bandung.json dengan json.load dan merekonstruksi graf terarah bertipe multigraph menggunakan nx.node_link_graph. Untuk keperluan tampilan yang lebih sederhana, graf tersebut kemudian diubah menjadi graf tak berarah. Posisi tiap node diambil dari atribut koordinat longitude (x) dan latitude (y), lalu digambar menggunakan matplotlib dan fungsi draw_networkx dari NetworkX: titik-titik node berwarna merah kecil menunjukkan lokasi simpul, sementara edge digambarkan sebagai garis tipis abu-abu. Akhirnya, grafik diberi judul “Visualisasi Graf Jalan Bandung” dan ditampilkan tanpa sumbu agar fokus pada struktur jaringan jalan .

Langkah-langkah implementasi algoritma BFS dan DFS

1. Memuat data jaringan jalan yang telah diekspor dalam format JSON OSMnx dan mengembalikannya sebagai objek graf tak berarah siap pakai.

Gambar 3.4 *Implementasi Program Memuat Data Jaringan Jalan*

Fungsi load_street_graph membuka file JSON (ditentukan oleh graf_json_path) dengan encoding UTF-8 dan mem-parse kontennya menjadi struktur Python. Kemudian, nx.node_link_graph merekonstruksi graf terarah multiedge dari format node-link OSMnx, setelah itu to_undirected(reciprocal=True) mengubahnya menjadi graf tak berarah hanya mempertahankan edge yang memiliki koneksi dua arah. Dengan demikian, fungsi ini menghasilkan objek graf G yang lebih sederhana dan langsung dapat dipakai untuk penelusuran jalur (misalnya BFS atau DFS) tanpa memikirkan arah jalan.

2. Memuat dan menyiapkan data rumah sakit dari berkas JSON agar mudah diproses dalam bentuk tabel.

Gambar 3.5 *Implementasi Program Memuat dan Menyimpan Data Rumah Sakit*

Fungsi load_hospitals membuka berkas JSON (hosp_json_path) dengan encoding UTF-8, mem-parse isinya menjadi list of dicts, lalu mengonversinya menjadi pandas.DataFrame. DataFrame ini difilter untuk hanya mencakup kolom "id", "nama_rs", "lat", dan "long", kemudian .copy() dibuat agar perubahan selanjutnya tidak memengaruhi data asli. Hasilnya adalah tabel rapi siap pakai untuk analisis atau mapping simpul terdekat.

3. Membangun struktur pencarian spasial

Gambar 3.6 Implementasi Program Membangun Struktur Pencarian Spasial

Fungsi build_kdtree_for_graph membuat KD-tree dari koordinat semua simpul graf G untuk mempercepat pencarian tetangga terdekat. Setelah mengekstrak daftar nodes dan membentuk array coords berisi pasangan (longitude, latitude) masing-masing simpul, array ini dipakai sebagai input cKDTree, menghasilkan struktur yang memungkinkan kueri spasial dalam $O(\log N)$ O(logN). Fungsi kemudian mengembalikan kd_tree, nodes, dan coords, sehingga setiap

indeks hasil kueri dapat langsung dipetakan ke ID simpul yang sesuai.

4. Membuat dua peta penting dari data rumah sakit ke simpul-simpul graf pertama

```python
def build_node_to_hospital_mapping(hosp_df, kd_tree, nodes, coords, threshold_meters=100):
    node_to_hosp = {}
    hosp_info = {}
    threshold_deg = threshold_meters / 111000.0

    for _, row in hosp_df.iterrows():
        hosp_id = row["id"]
        nama = row["nama_rs"]
        lat, lon = row["lat"], row["long"]
        hosp_info[hosp_id] = {"nama_rs": nama, "lat": lat, "long": lon}

        idxs = kd_tree.query_ball_point([lon, lat], r=threshold_deg)
        for idx in idxs:
            node_id = nodes[idx]
            node_to_hosp.setdefault(node_id, []).append(hosp_id)

    return node_to_hosp, hosp_info
```

Gambar 3.7 Implementasi Program Membuat dua peta penting dari data rumah sakit ke simpul-simpul graf pertama

Fungsi build_node_to_hospital_mapping bertugas menghasilkan dua struktur data utama: hosp_info, yang berisi metadata setiap rumah sakit (ID, nama, lintang dan bujur), serta node_to_hosp, yang memetakan ID simpul graf ke daftar rumah sakit terdekat. Pertama ia mengonversi threshold_meters (misalnya 100 m) menjadi satuan derajat lintang–bujur, lalu untuk setiap baris dalam hosp_df mengekstrak hosp_id, nama, dan koordinat, yang disimpan di hosp_info. Dengan memanggil kd_tree.query_ball_point([lon, lat], r=threshold_deg), fungsi ini memperoleh semua indeks simpul dalam radius ambang tersebut; setiap indeks kemudian dihubungkan kembali ke node_id dan hosp_id ditambahkan ke daftar pada node_to_hosp[node_id]. Hasil akhirnya memudahkan penelusuran cepat rumah sakit mana saja yang dapat diakses dari suatu simpul graf tertentu.

5. Implementasi BFS

```python
def bfs_find_all_hospitals(G, node_to_hosp, start_node):
    visited = {start_node}
    queue = deque([(start_node, 0)])
    parent = {start_node: None}

    found = {}

    while queue:
        current, dist_hop = queue.popleft()

        if current in node_to_hosp:
            for hosp_id in node_to_hosp[current]:
                if hosp_id not in found:
                    found[hosp_id] = (current, dist_hop)

        for nbr in G.neighbors(current):
            if nbr not in visited:
                visited.add(nbr)
                parent[nbr] = current
                queue.append((nbr, dist_hop + 1))

    results = []
    for hosp_id, (node_id, hop_dist) in found.items():
        path = []
        x = node_id
        while x is not None:
            path.append(x)
            x = parent[x]
        path.reverse()
        results.append((hosp_id, node_id, hop_dist, path))

    return results
```

Gambar 3.8 Implementasi Program BFS

Fungsi bfs_find_all_hospitals melakukan penelusuran graf secara breadth-first dimulai dari start_node untuk menemukan semua simpul yang terhubung ke rumah sakit (berdasarkan node_to_hosp), mencatat jarak hop minimal dan simpul perantaranya. Setiap kali simpul yang memetakan satu atau lebih hosp_id ditemui, fungsi menyimpan jarak hop pertama kali ditemukan (karena BFS menjamin minimal hop) dan menggunakan peta parent untuk merekonstruksi jalur kembali ke start_node. Setelah antrean kosong, fungsi mengembalikan daftar tuple (hosp_id, node_id, hop_dist, path) yang memuat ID rumah sakit, simpul wakil, jumlah hop, dan jalur simpul terpendek ke masing-masing rumah sakit.

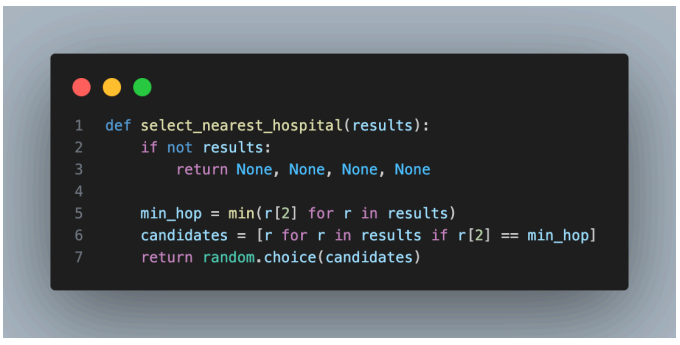
6. Implementasi DFS

```python
def dfs_find_all_hospitals(G, node_to_hosp, start_node):
    visited = set()
    stack = [(start_node, [start_node])]

    found = {}

    while stack:
        current, path = stack.pop()
        if current in visited:
            continue
        visited.add(current)

        if current in node_to_hosp:
            for hosp_id in node_to_hosp[current]:
                if hosp_id not in found:
                    found[hosp_id] = (current, len(path)-1, list(path))

        for nbr in G.neighbors(current):
            if nbr not in visited:
                stack.append((nbr, path + [nbr]))

    results = []
    for hosp_id, (node_id, hop_dist, path_list) in found.items():
        results.append((hosp_id, node_id, hop_dist, path_list))

    return results
```

Gambar 3.9 Implementasi Program DFS

Fungsi dfs_find_all_hospitals menelusuri graf secara depth-first mulai dari start_node, menggunakan stack berisi pasangan (simpul, path). Setiap simpul yang diambil dari stack dicek apakah sudah dikunjungi; jika belum, ditandai dan jika ada di node_to_hosp dicatat sebagai temuan rumah sakit beserta simpul, hop count (len(path)-1), dan jalur (path). Semua tetangga yang belum dikunjungi kemudian ditambahkan ke stack dengan jalur yang diperbarui. Setelah penelusuran selesai, fungsi mengembalikan daftar tuple (hosp_id, node_id, hop_dist, path_list) untuk setiap rumah sakit yang ditemukan.

7. Pencarian rumah sakit terdekat

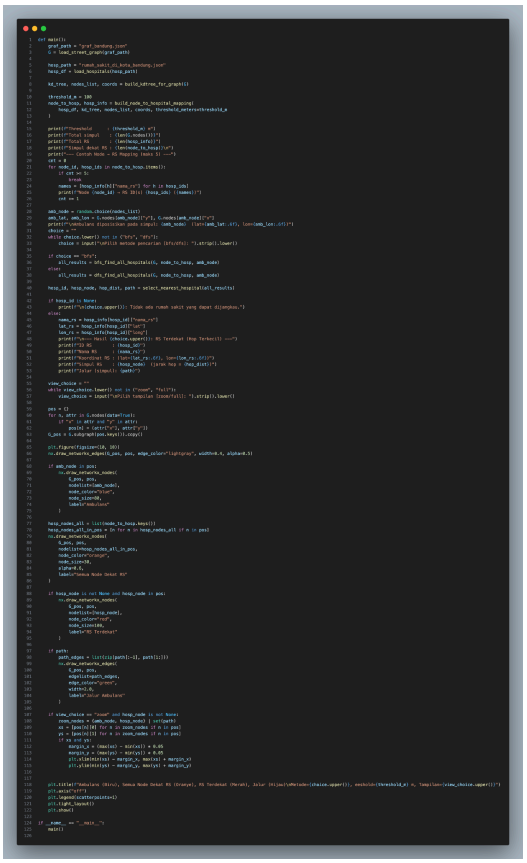


```python
def select_nearest_hospital(results):
    if not results:
        return None, None, None, None

    min_hop = min(r[2] for r in results)
    candidates = [r for r in results if r[2] == min_hop]
    return random.choice(candidates)
```

Gambar 3.10 Implementasi Program Pencarian Rumah Sakit Terdekat

Fungsi select_nearest_hospital memilih satu rumah sakit dengan jarak hop terkecil dari daftar results di mana setiap elemen r adalah tuple (hosp_id, node_id, hop_dist, path). Jika results kosong, ia mengembalikan empat None. Pertama, min_hop dihitung sebagai nilai minimum dari hop_dist di seluruh results. Kemudian, semua entri dengan hop_dist == min_hop dikumpulkan sebagai candidates, dan satu entri dipilih secara acak dengan random.choice(candidates). Dengan demikian, fungsi ini memastikan pemilihan rumah sakit terdekat (berdasarkan hop count), sekaligus menangani situasi ties secara acak.

8. Menjalankan keseluruhan simulasi routing ambulans



Gambar 3.11 Implementasi Program Main

Fungsi main menjalankan keseluruhan simulasi routing ambulans: pertama memuat graf jalan Bandung dan data rumah sakit, lalu membangun KD-tree dan memetakan simpul ke rumah sakit dalam radius 100 m. Program menampilkan ringkasan simpul dan RS, memilih secara acak simpul awal ambulans, meminta pilihan metode (bfs atau dfs), mengeksekusi pencarian jalur ke RS dan memilih yang terdekat, lalu mencetak detail RS terpilih (ID, nama, koordinat, hop count, jalur). Terakhir, ia memplot jaringan jalan dengan Matplotlib titik biru untuk ambulans, oranye untuk semua simpul dekat RS, merah untuk RS terdekat, serta jalur hijau dan menyesuaikan tampilan (zoom atau full) sesuai input pengguna.

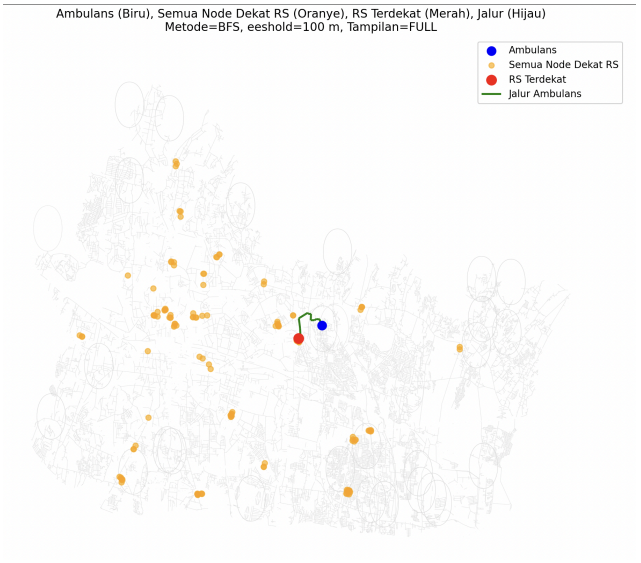
IV. Hasil dan Pembahasan

Sekarang, pada bagian Metode dan Hasil, kami akan membandingkan kinerja BFS dan DFS dalam menentukan rute ambulans ke rumah sakit terdekat. Metode BFS menjamin jalur dengan jumlah hop minimal karena menjelajah level demi level, sedangkan DFS bisa saja menemukan jalur lebih panjang terlebih dahulu karena menelusuri sedalam mungkin. Untuk setiap metode kita ukur hop count, panjang jalur, dan waktu eksekusi, lalu bandingkan rata-rata serta distribusinya. Dalam visualisasi hasil:

- Simpul biru: Semua simpul yang berasosiasi dengan satu atau lebih rumah sakit (node “dekat RS”).
- Simpul merah: Rumah sakit terpilih (jarak hop terkecil).
- Simpul hijau: Posisi ambulans saat memulai penelusuran.
- Garis hijau: Jalur yang ditempuh ambulans dari simpul hijau ke simpul merah.

Hasil :

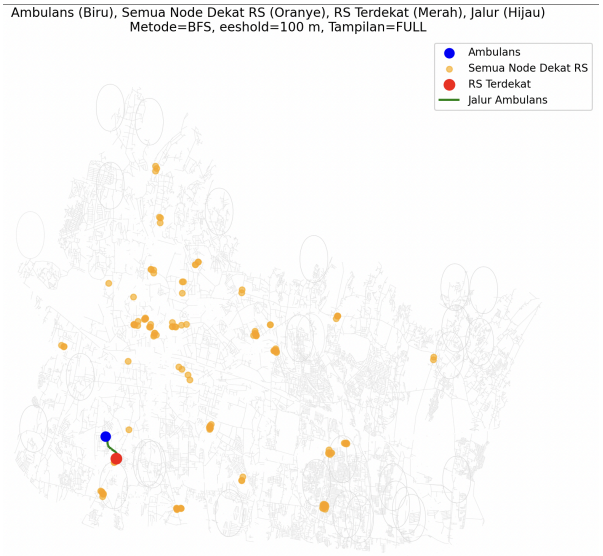
4.1 Studi case BFS 1



```
(base) jonathankenanbudianto@Jonathans-MacBook-Pro makalahstima % python bfs_d
fs_rumahsakit.py
Threshold      : 100 m
Total simpul    : 26021
Total RS        : 42
Simpul dekat RS : 136

--- Contoh Node → RS Mapping (maks 5) ---
Node 5639376089 → RS ID(s) [1] (['RS DR. HASAN SADIKIN'])
Node 2518040298 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6384595126 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6428110647 → RS ID(s) [3] (['RS SARTIKA ASIH'])
Node 6428110634 → RS ID(s) [3] (['RS SARTIKA ASIH'])

Ambulans diposisikan pada simpul: 12079576846  (lat=-6.909278, lon=107.653542)

Pilih metode pencarian [bfs/dfs]: bfs

--- Hasil BFS: RS Terdekat (Hop Terkecil) ---
ID RS        : 25
Nama RS      : RSIA GRAHA BUNDA
Koordinat RS : (lat=-6.913334, lon=107.645480)
Simpul RS    : 5392239472  (jarak hop = 24)
Jalur (simpul): [12079576846, 5369867359, 4176368888, 5369867362, 4176368879,
4176368885, 4176368874, 4176368877, 6445448344, 4176362168, 4176362256, 417636
2169, 4176369269, 4176369273, 4176369272, 4176369270, 5369867460, 7622389158,
3271916849, 5369662928, 5369867693, 9660735540, 5394844969, 5392239468, 539223
9472]

Pilih tampilan [zoom/full]: full
2025-06-23 12:45:36.567 python[3586:5449236] +[IMKClient subclass]: chose IMKC
lient_Modern
2025-06-23 12:45:36.567 python[3586:5449236] +[IMKInputSession subclass]: chos
e IMKInputSession_Modern
```

Gambar 4.1 Hasil Studi Case BFS 1

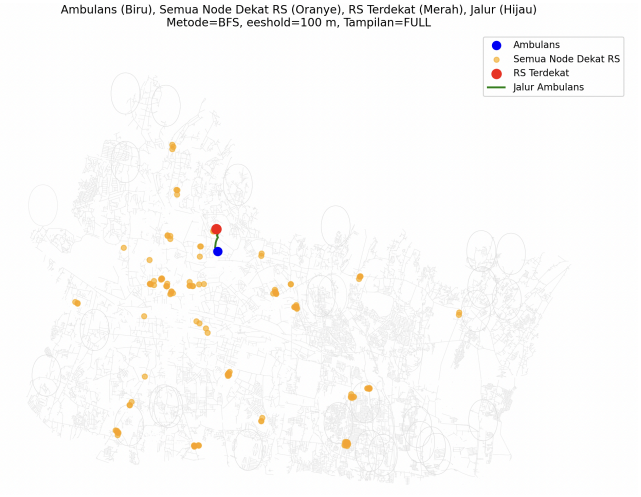
4.2 Studi case BFS 2




```
(base) jonathankenanbudianto@Jonathans-MacBook-Pro makalahstima % python bf
s_dfs_rumahsakit.py
Threshold      : 100 m
Total simpul    : 26021
Total RS        : 42
Simpul dekat RS : 136

--- Contoh Node → RS Mapping (maks 5) ---
Node 5639376089 → RS ID(s) [1] (['RS DR. HASAN SADIKIN'])
Node 2518040298 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6384595126 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6428110647 → RS ID(s) [3] (['RS SARTIKA ASIH'])
Node 6428110634 → RS ID(s) [3] (['RS SARTIKA ASIH'])

Ambulans diposisikan pada simpul: 3115483818  (lat=-6.936753, lon=107.58786
5)

Pilih metode pencarian [bfs/dfs]: bfs

--- Hasil BFS: RS Terdekat (Hop Terkecil) ---
ID RS        : 21
Nama RS      : RSUD BANDUNG KIWARI
Koordinat RS : (lat=-6.943208, lon=107.591410)
Simpul RS    : 3445649629  (jarak hop = 17)
Jalur (simpul): [3115483818, 5355717570, 3115483810, 5354151316, 5862651670
, 5353903554, 3115483977, 3445649579, 8775044333, 8775044337, 3445680801, 1
2361269743, 3115501991, 3115502013, 12741199907, 5348639839, 12741150891, 3
445649629]

Pilih tampilan [zoom/full]: full
2025-06-23 13:06:44.106 python[3955:5471604] +[IMKClient subclass]: chose I
MKClient_Modern
2025-06-23 13:06:44.106 python[3955:5471604] +[IMKInputSession subclass]: c
hose IMKInputSession_Modern
```

Gambar 4.2 Hasil Studi Case BFS 2

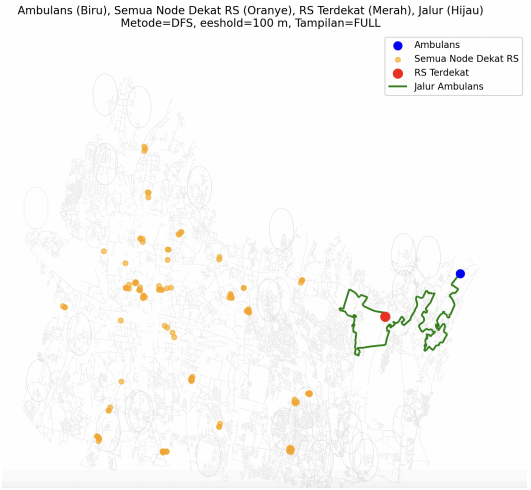
4.3 Studi case BFS 3

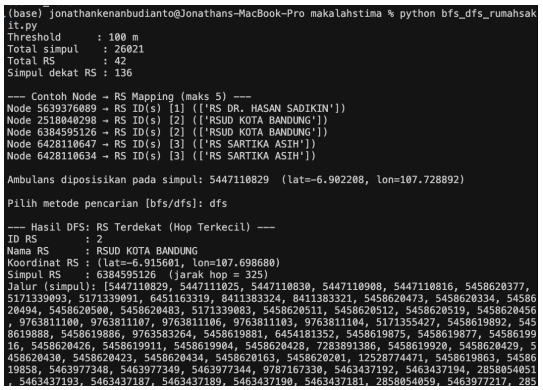



```
(base) jonathankenanbudianto@Jonathans-MacBook-Pro makalahstima % python bfs_dfs_
rumahsakit.py
Threshold      : 100 m
Total simpul    : 26021
Total RS        : 42
Simpul dekat RS : 136

--- Contoh Node → RS Mapping (maks 5) ---
Node 5639376089 → RS ID(s) [1] (['RS DR. HASAN SADIKIN'])
Node 2518040298 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6384595126 → RS ID(s) [2] (['RSUD KOTA BANDUNG'])
Node 6428110647 → RS ID(s) [3] (['RS SARTIKA ASIH'])
Node 6428110634 → RS ID(s) [3] (['RS SARTIKA ASIH'])

Ambulans diposisikan pada simpul: 32524478  (lat=-6.896411, lon=107.619831)

Pilih metode pencarian [bfs/dfs]: bfs

--- Hasil BFS: RS Terdekat (Hop Terkecil) ---
ID RS        : 31
Nama RS      : RS GIGI DAN MULUT UNPAD
Koordinat RS : (lat=-6.890172, lon=107.619090)
Simpul RS    : 5465992535  (jarak hop = 12)
Jalur (simpul): [32524478, 5866928158, 1857160830, 9587184408, 3443084271, 185716
0835, 11051197981, 3443083945, 3443083920, 5472412373, 9209019500, 5465992453, 54
65992535]

Pilih tampilan [zoom/full]: full
2025-06-23 17:51:20.696 python[7902:5982356] +[IMKClient subclass]: chose IMKClie
nt_Modern
2025-06-23 17:51:20.696 python[7902:5982356] +[IMKInputSession subclass]: chose I
MKInputSession_Modern
```

Gambar 4.3 Hasil Studi Case BFS 3

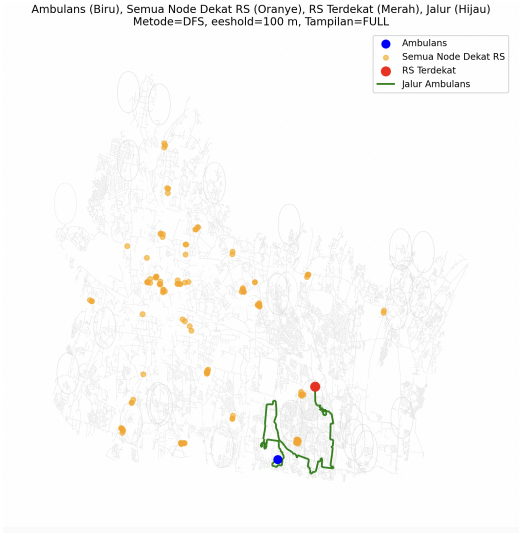
4.4 Studi case DFS 1

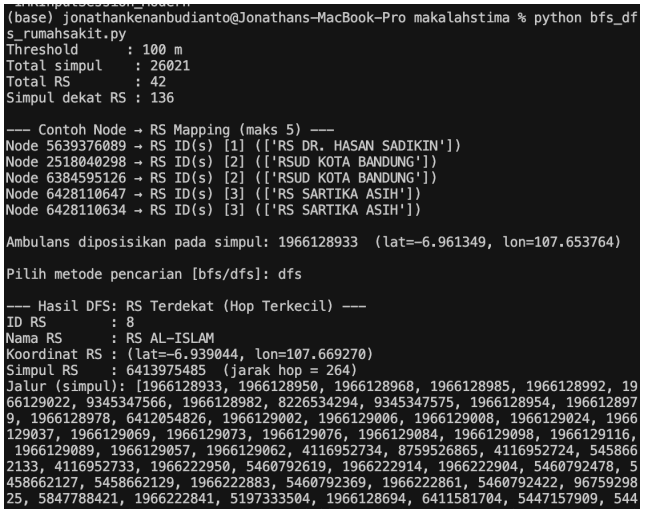




Gambar 4.4 Hasil Studi Case DFS 1

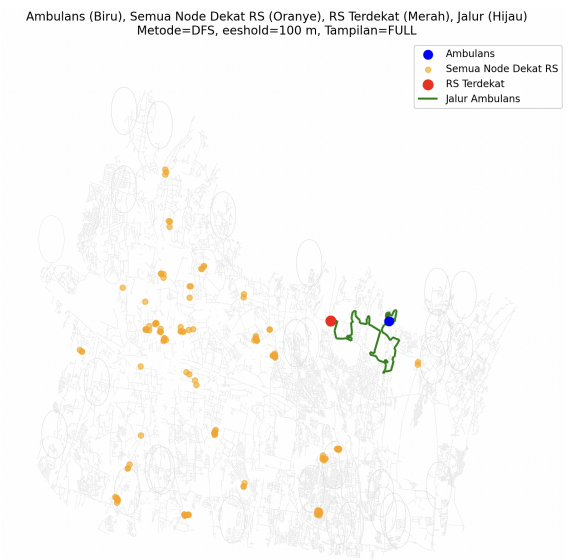
4.5 Studi case DFS 2

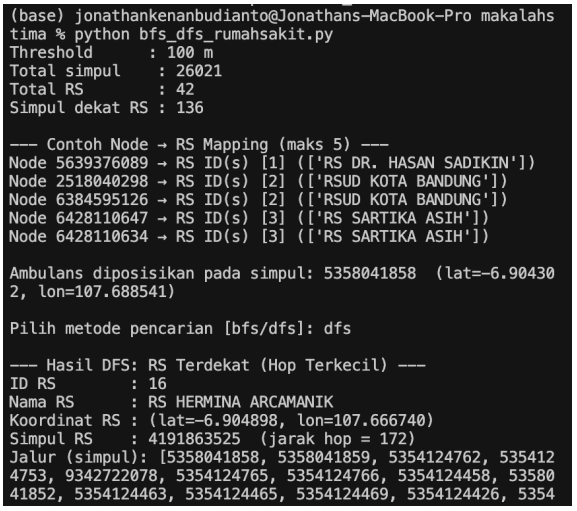




Gambar 4.5 Hasil Studi Case DFS 2

4.6 Studi case DFS 3





Gambar 4.6 Hasil Studi Case DFS 3

Pembahasan :
Berdasarkan hasil simulasi, BFS terbukti lebih unggul untuk penentuan rute ambulans ke rumah sakit terdekat karena menjelajah graf secara level demi level dari simpul awal (ambulans), sehingga selalu menemukan rumah sakit dengan jumlah hop minimal artinya jalur terpendek dalam satuan hop. Hal ini membuat BFS sangat efisien dalam memastikan ambulans langsung diarahkan ke rumah sakit terdekat tanpa perlu menelusuri semua cabang secara mendalam. Visualisasi menunjukkan bahwa jalur yang dilalui BFS melalui sangat sedikit simpul, yang menjadi bukti keefektifan algoritma ini dalam meminimalkan langkah.

Sebaliknya, DFS menelusuri graf dengan mendahulukan kedalaman, sehingga meski terkadang menemukan jalur pendek, algoritma ini tidak menjamin hop count terkecil karena tidak mengeksplorasi semua simpul di level yang sama terlebih dahulu. Akibatnya, jalur yang dihasilkan DFS seringkali lebih panjang dan kompleks, terlihat dari jumlah simpul yang jauh lebih banyak pada path bahkan pada

visualisasi kita harus memotong sebagian simpul yang dilalui agar tetap terbaca. Fenomena ini mengindikasikan ketidakefisienan DFS untuk kasus pencarian rumah sakit terdekat, karena ia bisa “terjebak” menjelajah rute panjang sebelum menemukan tujuan. Dengan demikian, untuk kebutuhan routing ambulans yang menuntut kecepatan dan kepastian hop minimal, BFS adalah pilihan yang lebih tepat.

V. Kesimpulan

Berdasarkan studi implementasi dan simulasi pada graf jaringan jalan Bandung, algoritma BFS terbukti lebih efektif dan efisien daripada DFS dalam menentukan rute ambulans ke rumah sakit terdekat. BFS menjelajah graf secara berlapis sehingga selalu menghasilkan jalur dengan hop count minimal, meminimalkan jumlah simpul yang dilalui serta waktu komputasi, sedangkan DFS cenderung “terjebak” menelusuri jalur panjang sebelum menemukan tujuan, menghasilkan rute kurang optimal dan proses yang lebih lambat. Untuk sistem penentuan rute ambulans di lingkungan perkotaan dengan graf tidak berbobot, kami merekomendasikan penggunaan BFS. Untuk kebutuhan masa depan yang mempertimbangkan jarak fisik atau bobot waktu tempuh, dapat dikembangkan algoritma berbobot seperti Dijkstra atau A* sebagai tindak lanjut.

VII. Appendix

Program yang digunakan dalam makalah ini dapat dilihat pada tautan berikut:

https://github.com/jonathankenan/AmbulanceRoutingBandung-BFS-DFS.git

VIII. Pengakuan

Pertama-tama, penulis mengucapkan syukur ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan petunjuk-Nya sehingga penulisan makalah ini dapat terselesaikan dengan baik. Ungkapan terima kasih juga penulis sampaikan kepada Bapak Monterico Adrian,terima kasih atas kepercayaan dan kesempatan yang diberikan untuk mengerjakan makalah ini, serta atas segala arahan dan masukan yang sangat berharga selama proses pembelajaran Matematika Diskrit. Semoga karya ini dapat memberikan manfaat dan inspirasi bagi pembaca serta menjadi sumbangsih kecil dalam pengembangan ilmu pengetahuan.

References

[1] Rinaldi Munir's Paper: R. Munir, "Title of the paper," in *Informatika STEI ITB*, 2006.(diakses 19-6-2025)

[2] H. R. P. Roosadi, “Penggunaan BFS dan DFS dalam Penentuan Rute Alur Cerita Serial Fate,” *Makalah IF2211 Strategi Algoritma*, Semester II 2020/2021, Institut Teknologi Bandung (diakses 20-6-2025)

[3] M. A. Adiputra, “Penerapan Algoritma BFS dan DFS untuk Penjadwalan Rencana Studi,” *Makalah IF2211 Strategi Algoritma*, Semester II 2017/2018, Institut Teknologi Bandung (diakses 20-6-2025)

[4] A. Muhardono, “Penerapan Algoritma Breadth First Search dan Depth First Search pada Game Angka,” *Jurnal Minfo Polgan*, vol. 12, no. 1, Maret 2023. DOI: 10.33395/jmp.v12i1.12340 (diakses 19-6-2025)

[5] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982]. (diakses 19-6-2025)

[6] Pemerintah Kota Bandung, “Rumah Sakit di Kota Bandung,” *Open Data Bandung*. [Online]. Available: https://opendata.bandung.go.id/dataset/rumah-sakit-di-kota-bandung-2 (diakses 19-6-2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 23 Juni 2025

Jonathan Kenan Budianto
13523139