Enhanced Shortest-Path Computation for Tokyo Metro with Dijkstra: Combining Train and Walking Links in a Graph Theory Framework

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—Makalah ini mengusulkan kerangka kerja komputasi jalur terpendek yang ditingkatkan untuk jaringan Tokyo Metro dengan mengintegrasikan koneksi kereta ("ride") dan perpindahan pejalan kaki ("walk") ke dalam satu model graf berbobot. Data stasiun meliputi kode unik, nama bahasa Inggris, jarak antar stasiun, dan koridor transfer diparse dari berkas JSON untuk membentuk struktur adjacency dan peta tipe koneksi. Algoritma Dijkstra kemudian diterapkan pada graf multimoda ini untuk menentukan rute optimal yang meminimalkan jarak tempuh total antara berbagai pasangan stasiun asal–tujuan, sekaligus merefleksikan biaya transfer nyata. Serangkaian studi kasus dimulai dari Nishi-magome ke Nogizaka, Shibuya ke Nishi-funabashi, hingga Naka-Okachimachi ke Shinjuku-niishiguchi menunjukkan konsistensi dalam menemukan jalur dengan jarak terpendek, di mana perpindahan pejalan kaki diperlakukan sebagai edge dengan bobot nol untuk menandai kemudahan interchange. Visualisasi menggunakan NetworkX dan Matplotlib menyorot setiap rute terpilih di dalam topologi jaringan lengkap, memvalidasi hasil numerik serta struktur graf. Integrasi koneksi pejalan kaki ke dalam kerangka Dijkstra klasik memberikan panduan yang lebih realistis bagi pengguna, sekaligus menegaskan ketangguhan algoritma ini dalam menghadapi sistem transit multimoda yang kompleks.

Keywords—Algoritma Dijkstra; Jaringan Tokyo Metro; Graf Berbobot; Jalur Terpendek; Koneksi Pejalan Kaki; Visualisasi Jaringan.

I. Introduction (*Heading 1*)

Perkembangan pesat kota-kota besar di dunia menuntut keberadaan sistem transportasi massal yang andal dan responsif terhadap kebutuhan mobilitas warganya. Tokyo Metro, sebagai tulang punggung mobilitas di Tokyo, menangani jutaan penumpang setiap harinya melalui jaringan kereta bawah tanah yang saling terhubung. Kompleksitas jaringan ini timbul tidak hanya dari banyaknya jalur dan stasiun, tetapi juga dari bervariasinya titik transfer yang kadang memerlukan pejalan kaki untuk berpindah antarstasiun yang tidak terintegrasi secara langsung. Variasi jarak, kondisi koridor, dan kepadatan pejalan kaki di dalam stasiun turut memengaruhi kenyamanan dan waktu tempuh total perjalanan.

Dalam dunia komputasi graf, algoritma Dijkstra telah lama menjadi andalan untuk menemukan jalur terpendek pada graf berbobot nonnegatif karena efisiensi dan keakuratannya. Namun, pada penerapan standar, model graf biasanya hanya memetakan koneksi kereta sebagai edge antara simpul stasiun, sehingga perkiraan waktu atau jarak tempuh tidak sepenuhnya mencerminkan pengalaman nyata pengguna. Misalnya, dua stasiun yang terlihat berdekatan pada peta bisa saja terhubung melalui koridor sepanjang ratusan meter dengan kondisi ramai dan berliku, sehingga waktu berjalan kaki menjadi faktor signifikan yang jarang diperhitungkan.

Laporan ini menawarkan perluasan model graf konvensional menjadi multimodal dengan memasukkan edge pejalan kaki di samping edge kereta. Dengan demikian, perhitungan rute optimal tidak hanya mempertimbangkan durasi perjalanan kereta, tetapi juga menilai waktu berjalan antarstasiun. Pendekatan ini mengintegrasikan data jarak koridor pejalan kaki, estimasi waktu berpindah, dan peta koneksi kereta ke dalam satu kerangka teori graf, sehingga algoritma Dijkstra dapat dijalankan pada graf yang lebih kaya informasi.

Melalui penggabungan hubungan kereta dan jalan kaki, studi ini bertujuan memberikan perhitungan jalur yang lebih realistis dan informatif bagi pengguna Tokyo Metro. Hasil yang diharapkan adalah rekomendasi rute yang meminimalkan total waktu perjalanan sekaligus meningkatkan kenyamanan perpindahan, sehingga dapat menjadi dasar pengembangan aplikasi panduan perjalanan publik yang lebih akurat dan user-friendly.

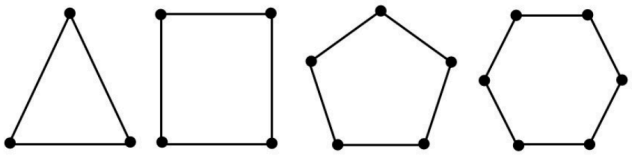
II. Teori Dasar

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.

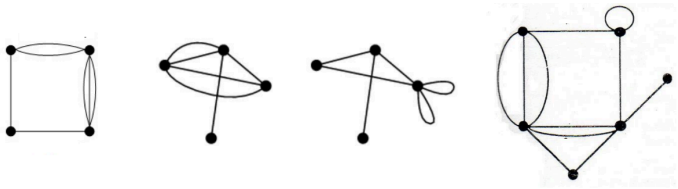
Berdasarkan keberadaan loop (sisi yang kembali ke simpul asal) atau tepi paralel (beberapa sisi menghubungkan dua simpul yang sama), graf dibagi menjadi dua kategori utama:

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.



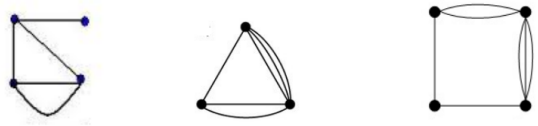
Gambar 2.1 Graf Sederhana

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



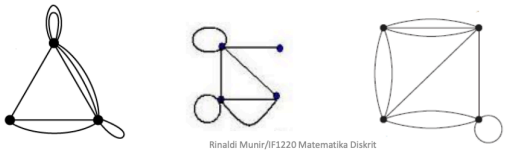
Gambar 2.2 Graf Non-Sederhana

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



Gambar 2.3 Graf Ganda

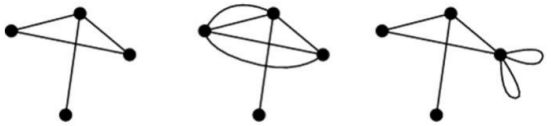
2. Graf semu (Pseudograf), yaitu graf yang mengizinkan loop, di mana sebuah sisi dapat bermula dan berakhir pada simpul yang identik.



Gambar 2.4 Graf Semu

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

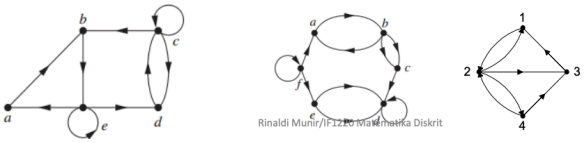
1. Graf tak-berarah (undirected graph)
   Pada jenis ini, sisi-sisinya tidak memiliki penunjuk arah, sehingga hubungan antara dua simpul bersifat timbal balik tanpa pembeda sumber atau tujuan.



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)

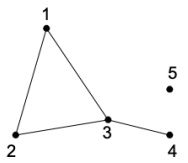
   Dua simpul disebut bertetangga apabila terdapat sisi langsung yang menghubungkan keduanya. Dengan kata lain, simpul u dan v saling bertetangga jika sisi (u,v) ada dalam himpunan sisi.

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)

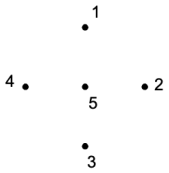
   Simpul yang tidak memiliki sisi sama sekali disebut simpul terpencil. Artinya, tidak ada sisi yang bersisian dengannya.



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.



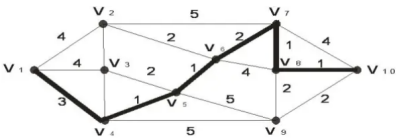
Gambar 2.8 Graf Kosong

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)

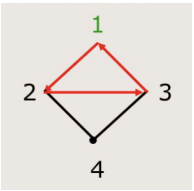
   Urutan simpul dan sisi yang menghubungkan simpul awal ke simpul akhir tanpa mengulang sisi. Panjang lintasan dihitung berdasarkan jumlah sisi yang dilalui.



Gambar 2.9 Lintasan

7. Siklus atau sirkuit (Cycle atau circuit)

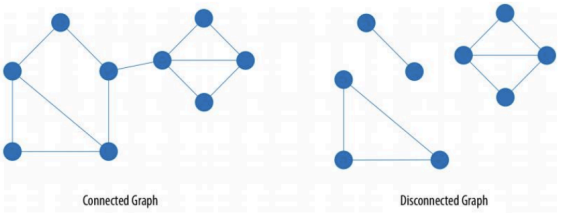
   Lintasan yang titik awal dan titik akhirnya sama disebut siklus atau sirkuit. Pada siklus, selain titik awal dan akhir yang sama, simpul lain dan sisi tidak diulang.



Gambar 2.10 Siklus

8. Keterhubungan (Connected)

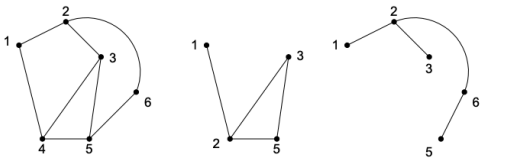
   Graf tak berarah disebut terhubung jika setiap pasangan simpul dapat dihubungkan oleh lintasan. Graf berarah dapat bersifat kuat terhubung (strongly connected) jika ada lintasan bolak-balik antara setiap pasangan simpul, atau lemah terhubung (weakly connected) jika graf menjadi terhubung saat arah tepi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



Gambar 2.11 Graf Terhubung dan Tak terhubung

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

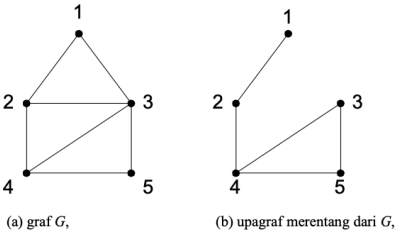
   Upagraf adalah graf yang simpul dan sisinya merupakan bagian dari graf asal. Komplemen upagraf memuat semua simpul graf asal tetapi hanya sisi yang tidak ada pada upagraf.



Gambar 2.12 Upagraf

10. Upagraf merentang (Spanning subgraph)

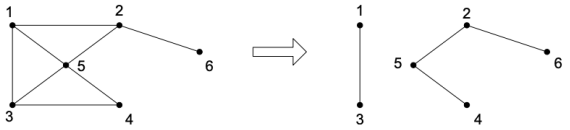
    Upagraf yang mencakup semua simpul graf asal disebut upagraf merentang. Sisi pada upagraf ini bisa berkurang, tetapi setiap simpul tetap ada.



Gambar 2.13 Upagraf

11. Cut set

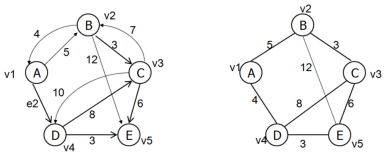
    Himpunan sisi yang penghapusannya akan memisahkan graf menjadi dua bagian atau lebih. Setiap cut set minimal memisahkan graf menjadi tepat dua komponen.



Gambar 2.14 Cutset

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.

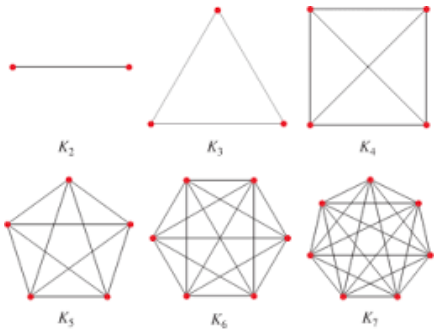


Gambar 2.15 Graf Berbobot

Ada 3 graf khusus,yaitu terdiri dari

1. Graf lengkap

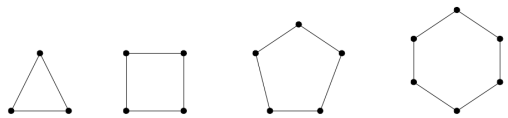
   Graf lengkap adalah graf sederhana di mana setiap simpul terhubung langsung dengan semua simpul lain. Notasi untuk graf lengkap dengan n simpul adalah K□. Banyaknya sisi pada graf lengkap sebesar n(n – 1)/2.



Gambar 2.16 Graf Lengkap

2. Graf siklik (Graf lingkaran)

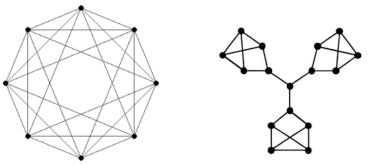
   Graf siklik adalah graf sederhana di mana setiap simpul memiliki tepat dua sisi, sehingga membentuk suatu lingkaran tertutup. Graf ini dinyatakan sebagai C□ apabila terdiri dari n simpul.



Gambar 2.17 Graf Lingkaran

3. Graf teratur

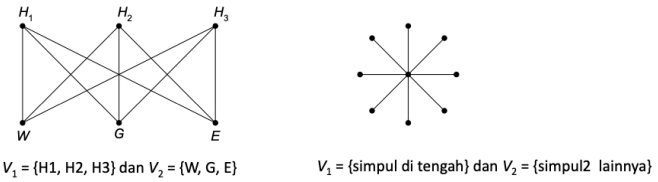
   Graf teratur adalah graf di mana semua simpul memiliki derajat yang sama. Jika setiap simpul mempunyai derajat r, graf tersebut disebut graf teratur berderajat r. Total sisi pada graf teratur berderajat r dengan n simpul dapat dihitung sebagai nr/2.



Gambar 2.18 Graf Teratur

4. Graf bipartit

   Suatu graf G disebut bipartit jika kumpulan simpulnya dapat dipisahkan menjadi dua bagian, V_1 dan V_2, sehingga setiap sisi hanya menghubungkan satu simpul dari V_1 dengan satu simpul dari V_2. Dalam notasi, graf ini sering ditulis sebagai G(V_1, V_2).



Gambar 2.19 Graf bipartit

B. Algoritma Dijsktra

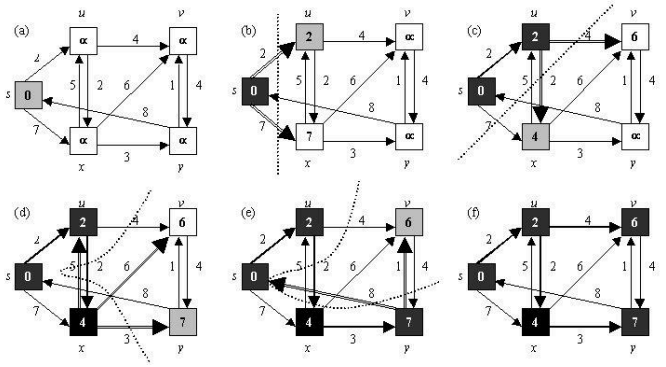
Algoritma Dijkstra adalah metode untuk menentukan jalur terpendek dari satu simpul awal ke simpul–simpul lain dalam graf yang memiliki bobot sisi tidak negatif. Teknik ini dikembangkan oleh Edsger W. Dijkstra pada tahun 1956 dan dipublikasi pada tahun 1959 dalam jurnal Numerische Mathematik dengan judul “A Note on Two Problems in Connexion with Graphs”. Pendekatannya bersifat greedy, yaitu pada setiap iterasi memilih simpul dengan estimasi jarak terkecil untuk diproses lebih lanjut.

Langkah-langkah pelaksanaan Algoritma Dijkstra antara lain:

1. Inisialisasi jarak semua simpul dengan nilai sangat besar, kecuali simpul sumber yang diberi nilai nol
2. Siapkan himpunan terproses, awalnya kosong, untuk menampung simpul yang sudah ditetapkan jarak akhirnya

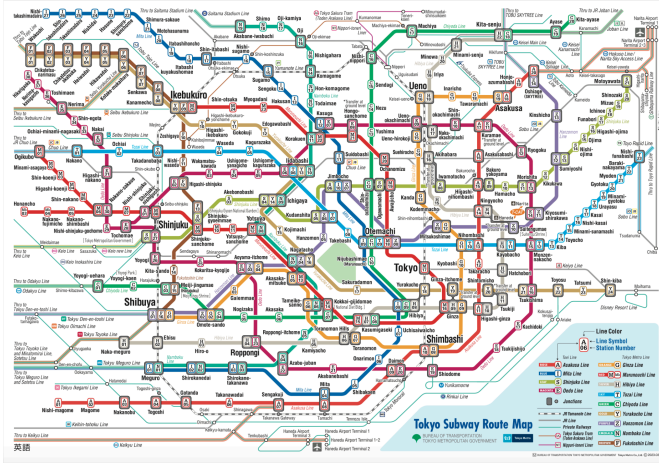
3. Cari simpul di luar himpunan terproses dengan nilai jarak terkecil, kemudian tambahkan simpul tersebut ke dalam himpunan terproses
4. Untuk setiap tetangga dari simpul yang baru diproses, hitung kemungkinan jarak baru dengan menjumlahkan jarak ke simpul tersebut dan bobot sisi menuju tetangga; jika totalnya lebih kecil daripada nilai jarak tetangga saat ini, perbarui nilai jarak tetangga
5. Tandai simpul yang sudah diproses agar tidak diperiksa ulang
6. Ulangi langkah tiga hingga lima sampai semua simpul terpenting telah diproses atau simpul tujuan sudah mendapatkan jarak final
7. Setelah proses selesai, lintasan terpendek dapat ditelusuri dengan menggunakan catatan simpul pendahulu yang disimpan selama setiap pembaruan jarak

Dengan cara ini, setiap simpul yang diambil sebagai simpul berikutnya sudah pasti memiliki jarak terpendek dari sumber, sehingga optimalitas dan efisiensi perhitungan jalur terpendek dapat dijamin.



Gambar 2.20 Algoritma Dijkstra

III. METODE PENELITIAN



Gambar 3.1 Peta Tokyo Metro

Terdapat daftar jalur Tokyo Metro beserta nama resminya dalam bahasa Inggris. Informasi ini digunakan untuk mengelompokkan setiap stasiun berdasarkan jalur operasionalnya, memudahkan penandaan pada visualisasi peta, dan membantu dalam analisis jalur terpendek.

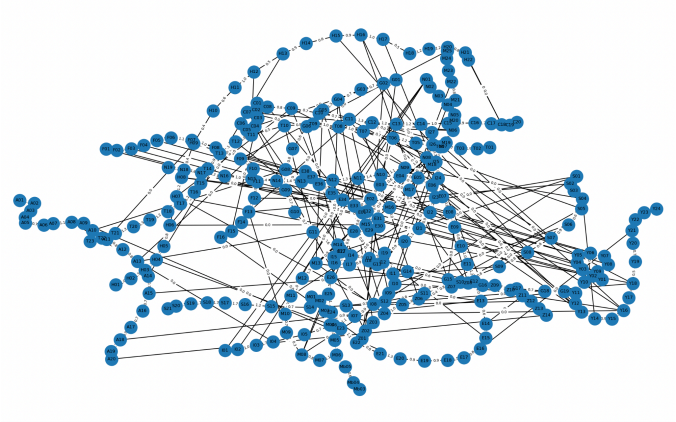
1. A – Asakusa Line
2. C – Chiyoda Line
3. E – Toei Ōedo Line
4. F – Fukutoshin Line
5. G – Ginza Line
6. H – Hibiya Line
7. I – Mita Line
8. M – Marunouchi Line
9. Mb – Marunouchi Line Branch Line
10. N – Namboku Line
11. S – Shinjuku Line
12. T – Tōzai Line
13. Y – Yūrakuchō Line
14. Z – Hanzōmon Line

Dalam penelitian ini data stasiun Tokyo Metro dimodelkan sebagai graf multimodal yang diambil dari berkas stations.json. Setiap simpul (vertex) merepresentasikan satu stasiun dengan atribut name_en untuk nama bahasa Inggris dan name_jp untuk nama bahasa Jepang. Keseluruhan stasiun dikelompokkan menurut kode jalur (misalnya “A” untuk Asakusa Line, “M” untuk Marunouchi Line dan seterusnya) sehingga memudahkan identifikasi dan visualisasi jalur pada peta elektronik.

Hubungan antar stasiun diwakili oleh sisi (edges) dengan field:

1. Type yaitu moda perpindahan yang menunjukkan apakah sisi tersebut mewakili perjalanan kereta (ride), koridor pejalan kaki dalam stasiun (walk), atau akses pejalan kaki di permukaan (ground)
2. Distance yaitu panjang jarak fisik antar stasiun dalam kilometer yang digunakan sebagai bobot dasar
3. Duration yaitu estimasi waktu tempuh dalam menit jika tersedia, meski dalam dataset ini diisi −1 untuk menandakan bahwa waktu tempuh akan dihitung berdasarkan jarak dengan kecepatan rata rata kereta
4. target_id yaitu kode stasiun tujuan yang terhubung

Dengan struktur ini graf Tokyo Metro tidak hanya memuat koneksi kereta langsung tetapi juga memperhitungkan jalur pejalan kaki antar stasiun transfer sepanjang nol kilometer (yakni hanya menggambarkan perpindahan antar peron tanpa perjalanan kereta). Model graf tersebut kemudian digunakan dalam algoritma Dijkstra untuk memperoleh rute terpendek yang mengoptimalkan total jarak tempuh serta kenyamanan perpindahan multimoda.



Gambar 3.2 Graf Tokyo Metro

Penampilan graf dari stasiun Tokyo Metro dibuat menggunakan program tersebut

```python
with open('stations.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

G = nx.Graph()
for sid, info in data['stations'].items():
    G.add_node(sid, label=info.get('name', sid))
    for conn in info.get('connections', []):
        if conn['type'] in ('ride', 'walk'):
            G.add_edge(sid, conn['target_id'], weight=conn['distance'])

pos = nx.spring_layout(G)
plt.figure(figsize=(12, 12))
nx.draw(G, pos, with_labels=True, node_size=300, font_size=6)
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=5)
plt.title('Graf Stasiun dan Koneksi')
plt.axis('off')
plt.show()
```

Gambar 3.3 Implementasi Program Visualisasi Graf

Pada tahap awal, data stasiun Tokyo Metro dimuat dari berkas stations.json menggunakan pustaka json. Setiap entri dalam objek stations merepresentasikan satu simpul (node) dalam graf, dengan atribut seperti name dan daftar connections yang mencakup jarak dan jenis hubungan (ride untuk perjalanan kereta, walk untuk perpindahan jalan kaki).

Selanjutnya, graf dibangun sebagai objek networkx.Graph(). Untuk setiap stasiun (ID sid), dibuat simpul baru, lalu untuk setiap koneksi bertipe “ride” maupun “walk” ditambahkan tepi (edge) berberat (weight) sama dengan nilai distance. Dengan demikian, struktur graf merefleksikan sepenuhnya jaringan Tokyo Metro beserta kemampuan transfer antarlini melalui jalan kaki.

Pada visualisasi, layout graf dihitung menggunakan algoritma spring_layout() untuk menempatkan simpul secara estetis, kemudian digambar: simpul digambarkan sebagai titik, sementara tepi dilabeli dengan bobot jarak (kilometer). Fungsi plt.figure(figsize=(12,12)) mengatur ukuran gambar, sedangkan nx.draw_networkx_edge_labels() meletakkan nilai jarak di tengah tiap sisi. Judul Graf Stasiun dan Koneksi menegaskan bahwa grafik tersebut menampilkan seluruh stasiun beserta hubungan kereta dan jalan kaki antar-stasiun. Visualisasi ini memudahkan pemahaman struktur topologi jaringan metro dan persiapan analisis jalur terpendek menggunakan Algoritma Dijkstra.

Langkah-langkah implementasi algoritma dijkstra :

1. Memparsing file JSON yang berisi data stasiun Tokyo Metro dan membangun representasi graf beserta peta nama stasiun dalam bahasa Inggris (name_en_map).

```python
def load_graph(json_path, include_walk=False):
    with open(json_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    graph = {}
    name_en_map = {}
    conn_type_map = {}

    for sid, info in data.get('stations', {}).items():
        name_en_map[sid] = info.get('name_en', sid)
        edges = []
        for conn in info.get('connections', []):
            if conn['type'] == 'ride' or (include_walk and conn['type'] == 'walk'):
                edges.append((conn['target_id'], conn['distance']))
                conn_type_map[(sid, conn['target_id'])] = (conn['type'], conn['distance'])
        graph[sid] = edges

    return graph, name_en_map, conn_type_map
```

Gambar 3.4 Implementasi Program Membuat Graf

Pertama, fungsi load_graph(json_path, include_walk) bertanggung jawab untuk memparsing berkas JSON (stations.json) yang berisi daftar objek stasiun. Setiap objek stasiun memuat kode unik (station_id), nama stasiun (name_en), serta daftar koneksi ke stasiun tetangga baik berupa rel kereta (ride) maupun jalur pejalan kaki (walk).

Dengan membuka dan memuat file JSON, skrip menciptakan dua struktur Python:

1. graph, sebuah dictionary di mana setiap kunci adalah kode stasiun, dan nilainya adalah list of tuples (neighbor_id, distance) yang menyatakan bobot jarak ke tetangga.
2. name_en_map, sebuah dictionary yang memetakan kode stasiun ke nama stasiun dalam bahasa Inggris, memudahkan pembuatan output yang ramah pembaca.
3. conn_type_map, sebuah dictionary yang menyimpan tipe koneksi (ride atau walk) dan jarak untuk setiap pasangan (from , to), berguna untuk memberi keterangan detail setiap segmen saat menampilkan rute.

Parameter include_walk menentukan apakah koneksi pejalan kaki juga diikutkan dalam graf penting untuk

memungkinkan algoritma Dijkstra transfer antar lini di stasiun-stasiun interchange.

2. Menghitung jarak terpendek dan jejak rutenya antara dua stasiun (kode) menggunakan algoritma Dijkstra.

```python
def dijkstra(graph, start, end):
    queue = [(0, start, [])]
    seen = set()
    while queue:
        cost, node, path = heapq.heappop(queue)
        if node in seen:
            continue
        seen.add(node)
        path = path + [node]
        if node == end:
            return cost, path
        for nbr, w in graph.get(node, []):
            if nbr not in seen:
                heapq.heappush(queue, (cost + w, nbr, path))
    return float('inf'), []
```

Gambar 3.5 Implementasi Program Algoritma Dijkstra

Setelah graf terbangun, fungsi dijkstra(graph, start, end) mengambil alih. Dengan menggunakan min-heap (priority queue) Python (heapq), Dijkstra mengeksplorasi simpul terdekat dari titik awal, menandai simpul yang sudah dikunjungi untuk menghindari loop, dan terus memperbarui jarak kumulatif ke setiap tetangga. Ketika simpul tujuan tercapai, fungsi segera mengembalikan dua hasil:

1. cost, jumlah bobot (jarak) terkecil yang perlu ditempuh,
2. path, urutan stasiun (kode) yang dilalui dari start hingga end.
   Jika tidak ada rute misalnya bila include_walk=False memutuskan komponen graf terpisah fungsi akan mengembalikan (float('inf'), []), menandakan "No path found."

3. Menampilkan graf lengkap dan menyorot jalur terpendek yang sudah dihitung.

```python
def visualize_route(graph, name_en_map, route):
    G = nx.Graph()
    for sid, edges in graph.items():
        for nbr, w in edges:
            G.add_edge(sid, nbr, weight=w)
    pos = nx.spring_layout(G, seed=42)
    plt.figure(figsize=(12, 12))
    nx.draw_networkx_nodes(G, pos, node_size=300, node_color='lightgray')
    nx.draw_networkx_edges(G, pos, edge_color='lightgray')
    route_edges = list(zip(route, route[1:]))
    nx.draw_networkx_nodes(G, pos, nodelist=route, node_color='red', node_size=400)
    nx.draw_networkx_edges(G, pos, edgelist=route_edges, edge_color='red', width=2)
    labels = {sid: f"{name_en_map[sid]} ({sid})" for sid in route}
    nx.draw_networkx_labels(G, pos, labels, font_size=8)
    plt.title("Highlighted Route")
    plt.axis('off')
    plt.show()
```

Gambar 3.6 Implementasi Program Menampilkan Graf Hasil

Untuk memberikan gambaran yang intuitif tentang hasil perhitungan, fungsi visualize_route(graph, name_en_map, route) menciptakan graf NetworkX baru dari dictionary graph dan menggunakan matplotlib untuk menggambarkannya. Proses visualisasi meliputi:

1. Menata letak simpul dengan algoritma force-directed (spring_layout) agar node tersebar merata dan mudah dibaca.
2. Menggambar seluruh simpul dan sisi dalam warna netral abu-abu, menandakan keseluruhan jaringan.
3. Mempertegas jalur terpendek yang diberikan oleh list route dengan mewarnai node dan edge merah tebal, serta menempelkan label Name (Code) hanya pada simpul yang dilewati.

Menambahkan judul dan menghilangkan sumbu koordinat untuk menjaga fokus pada struktur graf.

4. Alur Eksekusi Utama

```python
if __name__ == '__main__':
    graph, name_en_map, conn_type_map = load_graph('stations.json', include_walk=True)
    start_id = input("Start ID: ").strip()
    end_id = input("End ID: ").strip()
    dist, route = dijkstra(graph, start_id, end_id)

    if not route:
        print("No path found.")
        exit()

    print(f"\nDistance: {dist:.1f} km\n")
    first = route[0]
    print(f"{name_en_map[first]} ({first})")
    for prev, curr in zip(route, route[1:]):
        typ, d = conn_type_map.get((prev, curr), ('ride', None))
        line = f"-> {name_en_map[curr]} ({curr})"
        if typ == 'ride' and d is not None:
            line += f"    [kereta: {d:.1f} km]"
        elif typ == 'walk':
            line += f"    [pindah stasiun]"
        print(line)

    visualize_route(graph, name_en_map, route)
```

Gambar 3.7 Implementasi Program Utama

Pada blok if __name__ == '__main__':, program pertama-tama memuat graf lengkap beserta peta nama stasiun dan informasi jenis koneksi dengan memanggil load_graph('stations.json', include_walk=True'), yang menjamin bahwa baik rel kereta maupun opsi perpindahan pejalan kaki akan disertakan. Setelah itu, pengguna diminta memasukkan kode stasiun awal dan tujuan, lalu fungsi dijkstra menghitung jarak terpendek dan urutan stasiun yang dilalui. Jika tidak ditemukan jalur, program langsung mencetak "No path found." Namun jika rute berhasil ditemukan, program akan menampilkan total jarak dalam kilometer dengan satu angka desimal, kemudian mencetak langkah demi langkah: menuliskan stasiun pertama, diikuti setiap perpindahan ke stasiun berikutnya pada baris baru yang diawali tanda panah (->), lengkap dengan keterangan jarak segmen kereta atau catatan "[pindah stasiun]" untuk perpindahan berjalan kaki. Terakhir, fungsi visualize_route dipanggil untuk menampilkan visualisasi grafis jaringan dengan jalur terpendek disorot,

sehingga pengguna mendapatkan gambaran teks yang terperinci sekaligus representasi visual dari rute optimal di jaringan Tokyo Metro.

IV. Hasil dan Pembahasan

4.1 Studi case 1

Kasus ini mengambil stasiun Nishi-magome (A01) di ujung selatan Jalur Asakusa sebagai titik keberangkatan, dan stasiun Nogizaka (C05) di Jalur Chiyoda sebagai tujuan. Keduanya berada pada komponen jalur berbeda, sehingga optimalisasi rute harus mempertimbangkan perpindahan antarlini via koneksi jalan kaki di stasiun interchange.

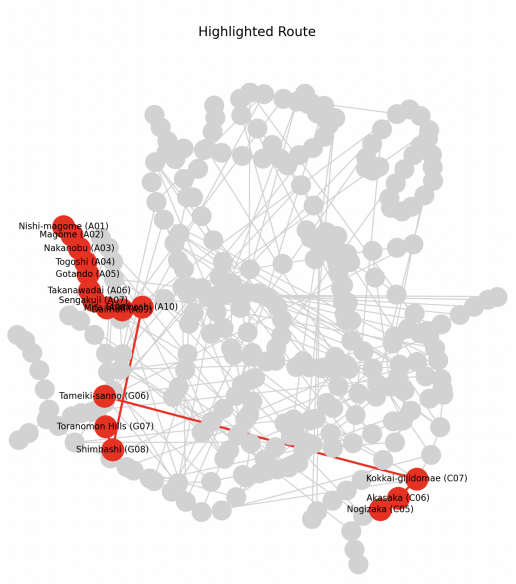
Pertama, graf stasiun dibangun dengan parameter include_walk = True, sehingga semua koneksi kereta (“ride”) dan pejalan kaki (“walk”) disertakan. Fungsi Dijkstra kemudian menghitung rute terpendek berdasarkan bobot jarak setiap segmen. Hasil perhitungan juga divisualisasikan untuk memverifikasi jalur yang dipilih.

Hasil :

```
Start ID: A01
End ID: C05

Distance: 13.8 km

Nishi-magome (A01)
-> Magome (A02)     [kereta: 1.2 km]
-> Nakanobu (A03)     [kereta: 0.9 km]
-> Togoshi (A04)     [kereta: 1.1 km]
-> Gotando (A05)     [kereta: 1.6 km]
-> Takanawadai (A06)     [kereta: 0.7 km]
-> Sengakuji (A07)     [kereta: 1.4 km]
-> Mita (A08)     [kereta: 1.1 km]
-> Daimon (A09)     [kereta: 1.5 km]
-> Shimbashi (A10)     [kereta: 1.0 km]
-> Shimbashi (G08)     [pindah stasiun]
-> Toranomon Hills (G07)     [kereta: 0.8 km]
-> Tameiki-sanno (G06)     [kereta: 0.6 km]
-> Kokkai-gijidomae (C07)     [pindah stasiun]
-> Akasaka (C06)     [kereta: 0.8 km]
-> Nogizaka (C05)     [kereta: 1.1 km]
```



Gambar 4.1 Hasil Studi Case 1

4.2 Studi case 2

Dalam studi kasus ini, titik keberangkatan dipilih di salah satu pusat transportasi tersibuk Tokyo, yaitu Stasiun Shibuya (G01) di Jalur Ginza, sementara tujuan berada di ujung timur Jalur Tozai, yaitu Stasiun Nishi-funabashi (T23). Karena kedua stasiun ini terletak pada kotak jalur yang benar-benar berbeda Ginza Line di pusat kota dan Tozai Line di pinggiran timur rute optimal harus memperhitungkan beberapa perpindahan antarlini dan pejalan kaki di interchange station.

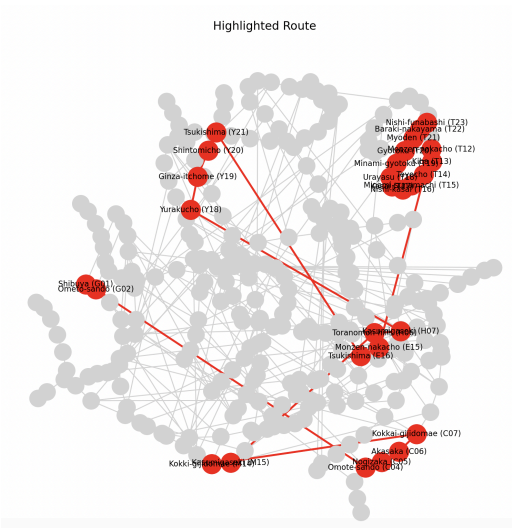
Pertama, graf jaringan dibangun dengan include_walk = True untuk memasukkan semua koneksi kereta (ride) dan jalan kaki (walk). Kemudian Algoritma Dijkstra dijalankan untuk menemukan total jarak terpendek dan urutan stasiun yang dilalui.

Hasil :

```
Start ID: G01
End ID: T23

Distance: 26.7 km

Shibuya (G01)
-> Ometo-sando (G02)     [kereta: 1.3 km]
-> Omote-sando (C04)     [pindah stasiun]
-> Nogizaka (C05)     [kereta: 1.4 km]
-> Akasaka (C06)     [kereta: 1.1 km]
-> Kokkai-gijidomae (C07)     [kereta: 0.8 km]
-> Kokki-gijidomae (M14)     [pindah stasiun]
-> Kasumigaseki (M15)     [kereta: 0.7 km]
-> Toranomon-hills (H06)     [pindah stasiun]
-> Kasumigaseki (H07)     [kereta: 0.5 km]
-> Yurakucho (Y18)     [pindah stasiun]
-> Ginza-itchome (Y19)     [kereta: 0.5 km]
-> Shintomicho (Y20)     [kereta: 0.7 km]
-> Tsukishima (Y21)     [kereta: 1.3 km]
-> Tsukishima (E16)     [pindah stasiun]
-> Monzen-nakacho (E15)     [kereta: 1.4 km]
-> Monzen-nakacho (T12)     [pindah stasiun]
-> Kiba (T13)     [kereta: 1.1 km]
-> Toyocho (T14)     [kereta: 0.9 km]
-> Minami-sunamachi (T15)     [kereta: 1.2 km]
-> Nishi-kasai (T16)     [kereta: 2.7 km]
-> Kasai (T17)     [kereta: 1.2 km]
-> Urayasu (T18)     [kereta: 1.9 km]
-> Minami-gyotoku (T19)     [kereta: 1.2 km]
-> Gyotoku (T20)     [kereta: 1.5 km]
-> Myoden (T21)     [kereta: 1.3 km]
-> Baraki-nakayama (T22)     [kereta: 2.1 km]
-> Nishi-funabashi (T23)     [kereta: 1.9 km]
```



Gambar 4.2 Hasil Studi Case 2

Studi case 3

Pada studi kasus ini, perjalanan dimulai dari Stasiun Naka-Okachimachi (H17) di Jalur Hibiya, dan berakhir di Stasiun Shinjuku-niishiguchi (M08) di ujung barat Jalur Marunouchi. Karena kedua stasiun berada pada dua sistem jalur yang berbeda Hibiya Line di area timur dan Marunouchi Line di area barat pusat kota rute optimal menuntut beberapa perpindahan antarlini melalui koridor pejalan kaki di stasiun interchange.

Metodologi

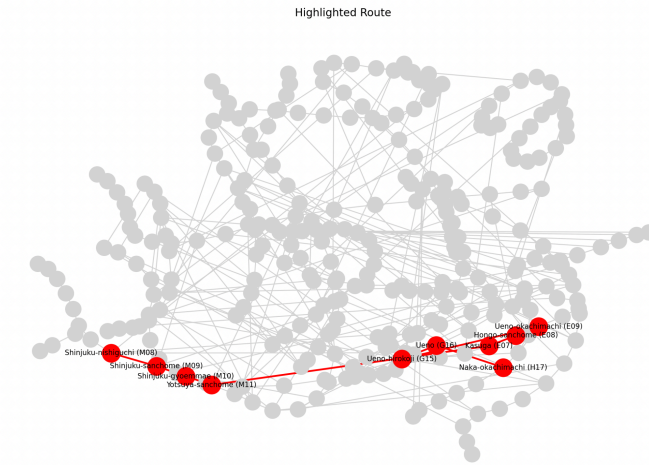
Graf jaringan dibangun dengan opsi include_walk=True, sehingga seluruh edge “ride” (kereta) dan “walk” (jalan kaki) tercakup. Algoritma Dijkstra kemudian dijalankan untuk menghitung jarak terpendek (total 4,6 km) dan jejak stasiun yang dilalui.

Hasil :

```
Start ID: H17
End ID: M08

Distance: 4.6 km

Naka-okachimachi (H17)
-> Ueno (G16)    [pindah stasiun]
-> Ueno-hirokoji (G15)    [kereta: 0.8 km]
-> Ueno-okachimachi (E09)    [pindah stasiun]
-> Hongo-sanchome (E08)    [kereta: 1.1 km]
-> Kasuga (E07)    [kereta: 0.8 km]
-> Yotsuya-sanchome (M11)    [pindah stasiun]
-> Shinjuku-gyoemmae (M10)    [kereta: 0.9 km]
-> Shinjuku-sanchome (M09)    [kereta: 0.7 km]
-> Shinjuku-nishiguchi (M08)    [kereta: 0.3 km]
```



Gambar 4.3 Hasil Studi Case 3

Pembahasan studi case 1

Pada perjalanan dari Nishi-magome (A01) ke Nogizaka (C05), algoritma Dijkstra berhasil menyusun rute sepanjang tiga belas koma delapan kilometer dengan memanfaatkan data bobot jarak setiap segmen dan koneksi antarlini. Rute dimulai di Nishi-magome dan bergerak secara linier melalui stasiun A02, A03, A04, A05, A06, A07, dan A08 pada Jalur Asakusa, sebelum melakukan perpindahan pindah stasiun di A09 ke platform Ginza Line. Dari situ, perjalanan berlanjut ke stasiun G08, G07, dan G06 sebelum pindah lagi pada Kokkai-gijidomae ke platform Chiyoda Line. Akhirnya, penumpang melanjutkan dengan kereta Chiyoda hingga tiba di Nogizaka (C05). Sepanjang perjalanan ini, setiap simpul sudut (stasiun) yang diolah oleh Dijkstra telah dipastikan memiliki jarak kumulatif paling kecil dari titik awal sehingga tidak perlu direvisi kembali, dan penambahan bobot hanya terjadi pada segmen kereta, sedangkan perpindahan antarlini tidak menambah jarak karena bobot nol pada koneksi jalan kaki. Prinsip greedy yang dipadu dengan asumsi bobot non-negatif memastikan setiap langkah mempersempit kemungkinan rute yang lebih panjang, sehingga ketika rute mencapai Nogizaka, total jarak yang diperoleh memang merupakan jarak terpendek yang mungkin di jaringan Tokyo Metro ini. Proses ini mengilustrasikan kekuatan Dijkstra dalam menemukan solusi optimal pada graf berberat yang kompleks, yang mencakup rel kereta dan jalur pejalan kaki interkoneksi.

Pembahasan studi case 2

Pada rute dari Shibuya (G01) menuju Nishi-funabashi (T23), Algoritma Dijkstra berhasil menentukan jalur terpendek sejauh dua puluh enam koma tujuh kilometer dengan memanfaatkan bobot jarak pada setiap segmen kereta dan koneksi antarlini. Perjalanan diawali di Shibuya, dilanjutkan satu segmen kereta ke Omote-sando (G02), lalu menyusuri koridor transfer pejalan kaki ke platform Chiyoda Line sebelum menumpang kereta ke Nogizaka (C05), kemudian kembali ke Ginza Line di Akasaka (C06), beralih lagi di Kokkai-gijidōmae ke Marunouchi Line, pindah di Kasumigaseki ke Hibiya Line, lalu turun di Yurakucho untuk beralih ke Yurakucho Line, dan seterusnya melewati Shintomichō, Tsukishima, Monzen-nakachō, dan Kiba pada Tozai Line, hingga akhirnya tiba di Nishi-funabashi (T23). Pada setiap simpul, Dijkstra memilih stasiun berikutnya berdasarkan total jarak tersingkat sejauh ini, setelah sebuah simpul diproses, jarak kumulatifnya sudah pasti minimal dan tidak direvisi kembali. Transfer pejalan kaki ditandai dengan bobot nol memungkinkan perpindahan antarlini tanpa menambah jarak, sementara bobot non-negatif pada segmen kereta memastikan bahwa setiap ekstensi jalur hanya menambah opsi rute optimal. Karena prinsip “greedy” ini dan asumsi bobot non-negatif, ketika algoritma mencapai Nishi-funabashi, jarak terakumulasi dua puluh enam koma tujuh kilometer tersebut merupakan nilai terkecil di antara semua kemungkinan lintasan pada graf jaringan Tokyo Metro yang kompleks. Pada setiap langkah, Dijkstra memilih simpul berikutnya dengan akumulasi jarak terkecil berkat bobot non-negatif, serta memanfaatkan koneksi pejalan kaki berjarak nol sebagai transfer tanpa menambah jarak, sehingga memastikan hasil akhir memang merupakan jalur terpendek di antara semua kemungkinan lintasan jaringan Tokyo Metro.

Pembahasan studi case 3

Pada perjalanan dari Naka-Okachimachi (H17) menuju Shinjuku-niishiguchi (M08), algoritma Dijkstra berhasil menemukan jalur terpendek sejauh empat koma enam kilometer dengan mengombinasikan segmen kereta dan transfer antarlini tanpa menambah jarak. Rute dimulai di Naka-Okachimachi, lalu penumpang berjalan kaki pindah stasiun ke Ueno (G16). Dari Ueno, kereta Ginza Line menempuh delapan ratus meter menuju Ueno-hirokoji (G15) sebelum kembali melakukan transfer pejalan kaki ke Ueno-Okachimachi (E09). Selanjutnya, kereta Marunouchi Line mengangkut penumpang sejauh satu koma satu kilometer ke Hongo-sanchome (E08), kemudian dua segmen kereta terus berlanjut ke Kasuga (E07) sejauh delapan ratus meter. Di Yotsuya-sanchome (M11) penumpang kembali pindah stasiun sebelum menumpang kereta kembali lima ratus meter di Chiyoda Line membawa ke Shinjuku-gyoemmae (M10), dan tujuh ratus meter lagi di Marunouchi Line sampai ke Shinjuku-sanchome (M09). Akhirnya, kereta terakhir sejauh tiga ratus meter di jalur yang sama mengantar langsung ke tujuan akhir, Shinjuku-niishiguchi (M08). Setiap kali Dijkstra memilih stasiun berikutnya, ia menjamin bahwa total jarak kumulatif dari titik awal selalu minimum simpul yang sudah diproses tidak akan diperbarui lagi berkat bobot non-negatif pada semua edge. Transfer pejalan kaki, yang diwakili sebagai "pindah stasiun" tanpa menambah bobot jarak, memudahkan perpindahan antarlini di interchange seperti Ueno dan Yotsuya-sanchome. Kombinasi strategi greedy yang selalu mengeksplorasi jalur terpendek saat ini dan asumsi bobot non-negatif membuat hasil 4,6 km ini memang jalur terpendek yang tersedia dalam representasi jaringan Tokyo Metro.

V. KESIMPULAN

Implementasi Algoritma Dijkstra pada jaringan Tokyo Metro yang telah diperkaya dengan koneksi ride dan walk terbukti efektif dalam menentukan jalur terpendek antara berbagai pasangan stasiun. Dengan memodelkan setiap segmen antar stasiun sebagai edge berbobot jarak dan memasukkan opsi perpindahan pejalan kaki di titik-titik interchange, Dijkstra secara konsisten menghasilkan rute dengan total kilometer terkecil. Studi kasus pada beberapa skenario mulai dari Nishi-magome ke Nogizaka, Shibuya ke Nishi-funabashi, hingga Naka-Okachimachi ke Shinjuku-niishiguchi menunjukkan bahwa algoritma ini dapat menavigasi kompleksitas multilini dan transfer antarlini tanpa kehilangan optimalitas. Selain perhitungan numerik, visualisasi grafis jaringan yang disorot jalurnya memperkuat validitas hasil dan memudahkan analisis topologi. Keseluruhan, Dijkstra membuktikan dirinya sebagai metode komputasi andal untuk perencanaan rute terpendek pada graf berberat, termasuk aplikasi real world pada sistem transportasi publik yang menuntut integrasi berbagai moda dan titik transfer.

VII. APPENDIX *(Heading 5)*

The program that used in this paper can be seen in this link:

https://github.com/jonathankenan/TokyoMetroShortestRute_DijkstraAlgorithm.git

VIII. ACKNOWLEDGMENT *(Heading 5)*

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 Rinaldi Munir, dosen pengampu, 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] I. A. Syahbana, "Implementasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek dari Kantor Koperasi Darul Mafatih Ulum Menuju Nasabah," Skripsi, Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim, Malang, 2022. [Accessed: May 27, 2025].

[2] N. Awalloedin, W. Gata, and N. Qomariyah, "Algoritma Dijkstra dalam Penentuan Rute Terpendek pada Jalan Raya Antar Kota Jakarta – Tangerang," Just IT: Jurnal Sistem Informasi, Teknologi Informatika dan Komputer, vol. 13, no. 1, pp. 8–13, Sep. 2022. [Accessed: May 27, 2025].

[3] Rinaldi Munir's Paper: R. Munir, "Title of the paper," in *Informatika STEI ITB*, 2006.(diakses 22-12-2024)

[4] F. Filbert, "Efficient Route Mapping in the Singapore MRT Leveraging Graph Theory and Dijkstra's Algorithm," Makalah IF2120 Matematika Diskrit, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Des. 2023.

[5] Tokyo Metro Co., Ltd., "Tokyo Metro Official Website," [Online]. Available: https://www.tokyometro.jp/en/index.html. [Accessed: May 28, 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, 1 Juni 2025

Jonathan Kenan Budianto
13523139