Pencarian rute Transjakarta dengan BFS dan DFS

William Gerald Briandelo-13222061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: briandelowilliam@gmail.com, 1322206@std.stei.itb.ac.id

***Abstract*—BFS dan DFS merupakan salah satu metode yang cukup umum digunakan dalam pencarian rute. Dalam makalah ini, dilakukan pemanfaatan metode BFS dan DFS, sehingga diperoleh rute yang dapat diambil supaya sampai ke halte tujuan. Hal ini bertujuan supaya user tidak salah dalam mengambil bus pada Transjakarta dan menjadi tersesat dalam perjalanan. Metode\ yang digunakan dengan menjadikan tiap halte sebagai simpul dan mencari tiap anak dari simpul tersebut yang terhubung oleh tiap bus. Selanjutnya, metode pencarian dilakukan sesuai dengan metode yang dipilih, baik BFS ataupun DFS**

Keywords—BFS, DFS, Transjakarta, Pencarian Rute, Graf, Graf berarah

I. Introduction (*Heading 1*)

Jakarta merupakan kota yang padat, sehingga banyak masyarakat yang menggunakan Transportasi umum, seperti Transjakarta, sebagai alternatif sehingga kemacetan berkurang. Namun, banyak dari pengguna Transjakarta yang tidak mengetahui rute dari masing-masing bus Transjakarta sehingga tidak jarang ada masyarakat yang salah menaiki bus dan tersesar. Hal ini, dikarenakan banyak rute dari masing-masing bus pada Transjakarta yang menghubungkan berbagai halte berbeda, sehingga rute-rute dari bus tersebut menjadi rumit untuk dihafal.

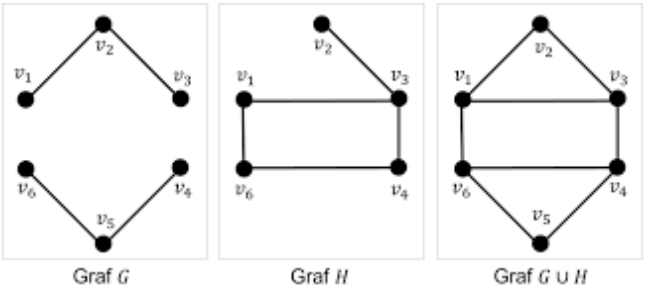
Untuk mengatasi permasalahan tersebut, dapat dirancang sebuah sistem pencarian rute, sehingga masyarakat tidak perlu menghafal semua rute Transjakarta jika ingin bepergian. Terdapat berbagai algoritma yang bisa diterapkan untuk menyelesaikan permasalahan ini, seperti metode bruteforce, backtracking, Breadth First Search, serta Depth First Search. Pada makalah ini, akan dilakukan studi untuk penerapan metode BFS serta DFS dalam penyelesaian masalah pencarian rute Transjakarta.

II. Landasan Teori

A. *Teori Graf*

Teori graf adalah cabang matematika yang mempelajari sifat-sifat graf, yaitu struktur abstrak yang digunakan untuk memodelkan relasi antar objek. Sebuah graf secara formal didefinisikan sebagai pasangan himpunan (V, E), di mana V adalah himpunan tak-kosong dari simpul-simpul yang merepresentasikan objek-objek, dan E adalah himpunan sisi yang merepresentasikan hubungan antara sepasang simpul. Keberadaan sebuah sisi antara dua simpul menandakan adanya relasi atau koneksi langsung. Struktur ini memungkinkan penyederhanaan masalah-masalah kompleks dalam berbagai bidang ke dalam bentuk visual yang dapat dianalisis secara sistematis.

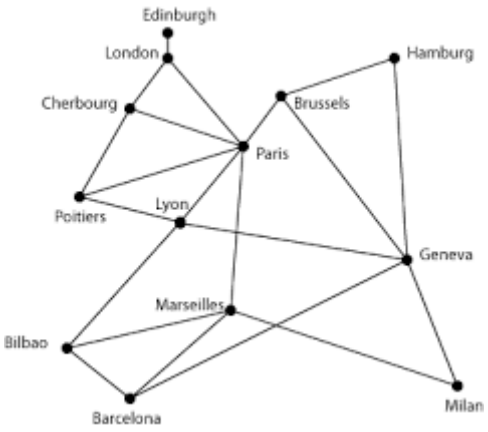
Graf dapat diklasifikasikan berdasarkan karakteristik sisi dan simpulnya. Berdasarkan arahnya, sisi dapat bersifat berarah, ketika relasi hanya berlaku satu arah, atau tidak berarah (undirected), di mana relasi bersifat timbal balik. Klasifikasi lainnya mencakup simple graph yang tidak mengizinkan adanya sisi ganda maupun loop di mana sebuah sisi menghubungkan simpul dengan dirinya sendiri. Konsep-konsep fundamental dalam teori graf seperti lintasan, yaitu urutan sisi yang menghubungkan sekuens simpul, dan keterhubungan menjadi dasar bagi banyak algoritma untuk menyelesaikan permasalahan seperti pencarian rute, analisis jaringan, dan optimasi.



Gambar 1.1 Visualisasi Graf (Sumber: https://mathcyber1997.com/materi-soal-operasi-graf-subgraf/)

B. *Pencarian Rute*

Pencarian rute merupakan permasalahan yang dapat direpresentasikan dan diselesaikan menggunakan teori graf. Lokasi atau titik-titik pemberhentian digambarkan sebagai simpul, sementara jalur yang menghubungkan lokasi-lokasi tersebut direpresentasikan sebagai sisi. Setiap sisi dapat memiliki bobot yang melambangkan berbagai metrik seperti jarak, waktu tempuh, atau biaya. Tujuan utama dari algoritma pencarian rute adalah untuk menemukan sebuah lintasan yang menghubungkan kedua simpul pada graf.

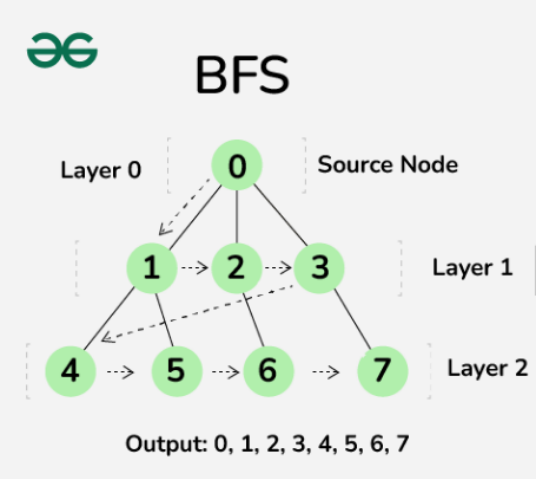


Gambar 1.2 Visualisasi Permasalahan Route Finding dengan Graf (Sumber: https://users.sussex.ac.uk/~christ/crs/kr-ist/lec01b.html)

C. *Breadth First Search*

Breadth-First Search (BFS) adalah algoritma penelusuran graf yang secara sistematis menjelajahi simpul-simpul graf lapis demi lapis. Proses dimulai dari sebuah simpul awal, kemudian algoritma akan mengunjungi semua tetangga terdekat dari simpul tersebut. Setelah semua simpul pada level pertama telah dikunjungi, barulah penelusuran berlanjut ke level berikutnya, dan begitu seterusnya. Untuk mengelola urutan simpul yang akan dikunjungi, BFS memanfaatkan struktur data queue yang menerapkan prinsip First-In, First-Out (FIFO), yaitu simpul yang pertama kali ditemukan akan menjadi yang pertama untuk dieksplorasi lebih lanjut.

Karena mekanisme penelusurannya yang melebar, BFS memiliki beberapa properti penting. Algoritma ini bersifat complete, artinya BFS akan selalu menemukan solusi atau lintasan jika memang ada. Selanjutnya, untuk graf tidak berbobot, BFS bersifat optimal karena lintasan yang pertama kali ditemukannya dari simpul awal ke simpul tujuan dijamin merupakan lintasan terpendek dalam hal jumlah sisi yang dilalui.

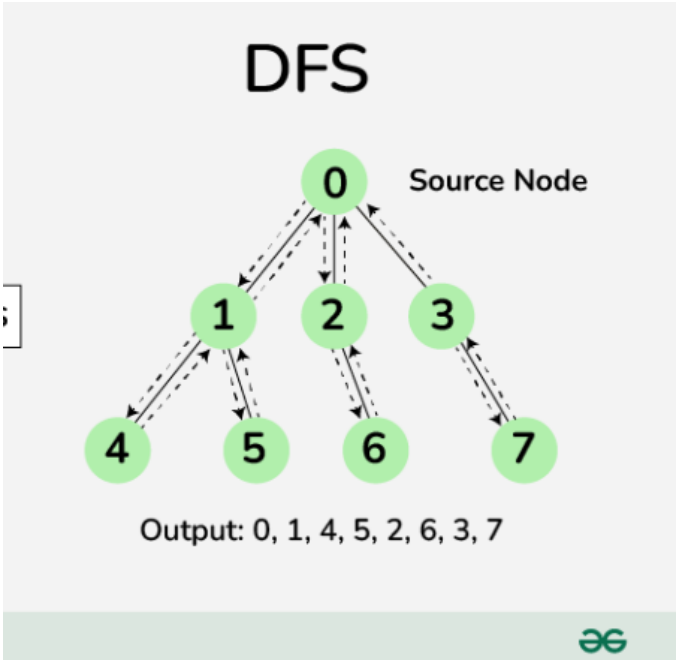


Gambar 1.3 Visualisasi BFS (Sumber: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/)

D. *Depth First Search*

Depth-First Search (DFS) merupakan algoritma penelusuran graf fundamental yang menerapkan strategi penjelajahan secara mendalam. Berbeda dengan BFS, DFS akan memulai dari simpul awal dan menelusuri satu cabang lintasan sejauh mungkin hingga mencapai simpul ujung atau kondisi buntu di mana tidak ada lagi simpul tetangga yang belum dikunjungi. Setelah itu, algoritma akan melakukan runut-balik ke simpul sebelumnya untuk menjelajahi cabang lain yang belum tersentuh. Implementasi DFS secara inheren menggunakan struktur data stack dengan prinsip Last-In, First-Out (LIFO), yang seringkali diterapkan dengan pemanggilan fungsi rekursif.

Seperti BFS, algoritma DFS juga bersifat complete, yang berarti akan menemukan sebuah lintasan antara dua simpul jika lintasan tersebut eksis. Namun, DFS tidak menjamin bahwa lintasan yang pertama kali ditemukan adalah lintasan yang terpendek. Jalur yang dihasilkan sangat bergantung pada urutan cabang yang dieksplorasi. Meskipun demikian, DFS memiliki keunggulan dalam penggunaan memori yang lebih efisien dibandingkan BFS pada graf yang sangat lebar. Aplikasinya meliputi pemecahan masalah labirin, pendeteksian siklus dalam graf, pengurutan topologis, dan sebagai basis dari algoritma lain yang lebih kompleks seperti pencarian runut-balik.



Gambar 1.4 Visualisasi DFS (Sumber: https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/)

III. Implementasi

Pada implementasi, digunakan bahasa pemrograman python.

A. Implementasi Graf

Simpul dari graf adalah setiap halte Transjakarta yang unik, direpresentasikan sebagai sebuah string. Sementara itu, sisi yang menghubungkan antar simpul bersifat implisit dan tidak disimpan sebagai satu objek terpisah. Sebuah sisi ada jika dua halte tercatat berada dalam urutan yang bersebelahan pada salah satu rute bus. Setiap koneksi antar halte tidak hanya sekadar sisi, tetapi memiliki label, yaitu nama bus itu sendiri. Dengan demikian, program memahami jaringan ini bukan hanya sebagai "Halte A terhubung ke Halte B", melainkan "Halte A terhubung ke Halte B melalui perjalanan dengan Koridor 1". Informasi label ini penting untuk membedakan rute dan merencanakan transit

B. Implementasi BFS

Pada implementasi BFS, pencarian diawali dengan penyiapan dua struktur data utama, yaitu sebuah queue dan sebuah catatan untuk halte yang sudah dikunjungi. Halte keberangkatan adalah item pertama yang dimasukkan ke dalam queue, dan pada saat yang sama, ia langsung ditandai dalam catatan kunjungan. Penggunaan queue bertujuan untuk memastikan proses pencarian berjalan level demi level sesuai prinsip First-In, First-Out. Sementara itu, catatan kunjungan sangat krusial untuk mencegah program memeriksa halte yang sama berulang kali, yang bisa menyebabkan terjadinya putaran tak terbatas pada rute yang melingkar.

Proses pencarian inti berjalan dalam sebuah putaran selama queue tidak kosong. Program secara konsisten mengambil halte dari posisi paling depan queue untuk diperiksa. Jika halte tersebut bukan tujuan akhir, program akan mencari tahu semua bus yang melintasinya. Untuk setiap bus yang ditemukan, algoritma tidak mengikuti seluruh rutenya, melainkan hanya melakukan satu langkah spesifik, yaitu mencari posisi halte saat ini dalam daftar rute bus tersebut, lalu menentukan satu halte berikutnya saja, dengan memperhitungkan kemungkinan rute yang kembali ke awal. Apabila halte berikutnya ini belum pernah tercatat dalam catatan kunjungan, ia akan ditandai dan kemudian ditambahkan ke bagian paling belakang queue bersama dengan seluruh riwayat perjalanan yang membawanya ke sana.

C. Implementasi DFS

Pada Implementasi DFS, digunakan struktur data stack dengan prinsip LIFO. Seperti BFS, DFS juga memanfaatkan catatan halte yang sudah dikunjungi untuk mencegah terjebak dalam siklus.

Proses pencarian DFS dimulai dengan mengambil halte teratas dari stack. Dari halte ini, program akan memilih satu jalur bus, lalu menelusuri seluruh kemungkinan perhentian berikutnya pada jalur bus tersebut. Semua kemungkinan langkah selanjutnya ini kemudian ditumpuk di bagian atas stack. Karena sifat stack ini, algoritma akan langsung mengambil langkah terakhir yang baru saja ditambahkan dan melanjutkan penelusuran lebih dalam ke sepanjang cabang rute tersebut. Proses ini terus berlanjut hingga tujuan ditemukan atau hingga mencapai jalan buntu, yaitu halte yang sudah pernah dikunjungi, di mana algoritma akan backtrack dengan mengambil item berikutnya dari stack.

Berikut merupakan pseudocode dari implementasi yang dilakukan.

```
Procedure Main:
  Kamus Lokal:
    rute : map[string]->list[string]
    pencari : PencariRuteBus
    asal, tujuan, pilih : string
    hasil : string

  rute ← data rute bus
  pencari ← new PencariRuteBus(rute)

  asal ← input("Halte Asal : ")
  tujuan ← input("Halte Tujuan : ")
  asal ← pencari.match(asal)
  tujuan ← pencari.match(tujuan)
  if not asal or not tujuan then
    print("Halte tidak ditemukan.")
    return
  end if

  pilih ← input("Metode (BFS/DFS): ").toLower()
  if pilih = "bfs" then
    hasil ← pencari.search_bfs(asal, tujuan)
  else
    hasil ← pencari.search_dfs(asal, tujuan)
  end if

  print(hasil)
End Procedure

Class PencariRuteBus:
  Kamus Lokal (atribut):
    _rute_bus_asli : map[string]->list[string]
    _graf : map[string]->set[string]
    _semua_halte : list[string]

  Constructor(rute : map[string]->list[string])
    buildGraph(rute)

  Function buildGraph(rute)
    Kamus Lokal:
      graf : map[string]->set[string]
      halte : list[string]
    graf, halte ← empty map, empty list
    for bus, stops in rute:
      for s in stops:
        graf[s] add bus
        if s not in halte: add s
    return graf, halte

Function search_bfs(start, goal) → string
```

```
Kamus Lokal:
  queue   : queue of (string, list of (bus, from, to))
  visited : set of string
  node    : string
  path    : list of (bus, from, to)
  bus     : string
  route   : list of string
  idx     : integer
  neighbor : string
if start ∉ _graf OR goal ∉ _graf then
  return "Halte tidak valid."
queue   ← [(start, [])]
visited ← { start }
while queue is not empty do
  (node, path) ← dequeue(queue)
  for each bus in _graf[node] do
    route ← _rute_bus_asli[bus]
    idx   ← indexOf(node in route)
    neighbor ← route[(idx + 1) mod length(route)]
    if neighbor = goal then
      return format(path + [(bus, node, neighbor)])
    if neighbor ∉ visited then
      visited ← visited ∪ { neighbor }
      enqueue(queue, (neighbor, path + [(bus, node, neighbor)]))
    end if
  end for
end while
return "Tidak ditemukan."

Function search_dfs(start, goal) → string
  Kamus Lokal:
    stack : stack of (string, list)
    visited : set[string]
    node : string
    path : list of (bus, from, to)
    bus, neighbor : string
  if start or goal not in _graf then return "Halte tidak valid."
  stack ← [(start, [])]
  visited ← {}
  while stack:
    node, path ← pop(stack)
    if node = goal: return format(path)
    if node in visited: continue
    add node to visited
    for bus in _graf[node]:
      for neighbor in nextStops(bus, node):
        push(stack, (neighbor, path + [(bus, node, neighbor)]))
  return "Tidak ditemukan."

Function match(input : string) → string or null
  Kamus Lokal:
    input_lower : string
    nama_halte : string
  input_lower ← lowercase(input)
  for nama_halte in _semua_halte:
    if lowercase(nama_halte) = input_lower: return nama_halte
  return null

Function nextStops(bus, node) → list[string]
  // mengembalikan halte selanjutnya di rute circular
```

IV. PENGUJIAN

Dilakukan pengujian terhadap implementasi BFS dan DFS yang dilakukan. Digunakan daftar rute bus seperti yang tercantum pada sumber berikut (Rute Transjakarta).

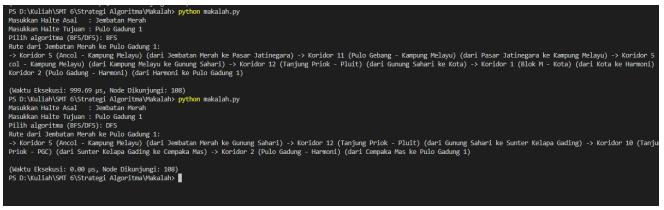
Pengujian program bertujuan untuk dua hal utama memverifikasi kebenaran rute yang dihasilkan dan menganalisis perbedaan kinerja antara algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS). Untuk mencapai tujuan ini, serangkaian kasus uji dirancang untuk mencakup berbagai skenario perjalanan. Skenario tersebut meliputi perjalanan sederhana yang hanya memerlukan satu bus, perjalanan kompleks yang membutuhkan beberapa kali transit, serta kasus-kasus khusus seperti tidak adanya rute yang memungkinkan, atau saat halte asal dan tujuan adalah sama. Setiap kasus uji dijalankan menggunakan kedua algoritma untuk membandingkan tidak hanya output rutenya, tetapi juga metrik kinerjanya, yaitu waktu eksekusi dan jumlah node (halte) yang dikunjungi.

Sebagai contoh test case, perjalanan dari "Blok M" ke "Kota" digunakan untuk menguji skenario rute tunggal. Diharapkan kedua algoritma menghasilkan rute yang sama di sepanjang Koridor 1, namun dengan perbedaan pada waktu eksekusi dan jumlah node yang dikunjungi. Untuk skenario kompleks, digunakan rute dari "Jembatan Merah" ke "Pulo Gadung 1" yang membutuhkan beberapa kali transit. Pada kasus ini, BFS dapat memberikan rute paling efisien dengan jumlah halte minimal. Sebaliknya, DFS dapat menghasilkan rute alternatif yang lebih panjang.

Dari pengujian yang dilakukan terlihat bahwa karakteristik teoretis dari kedua algoritma ditunjukkan secara konsisten pada hasil. BFS selalu berhasil menemukan rute "terbaik" dalam hal jumlah halte yang harus dilalui, menjadikannya pilihan yang lebih unggul untuk penggunaan praktis. Namun, jumlah node yang dikunjungi pada BFS menunjukkan seberapa luas pencarian yang perlu dilakukan untuk menjamin solusi optimal. Selanjutnya, DFS menyajikan rute yang valid, tapi seringkali tidak efisien. Data waktu eksekusi dan jumlah node yang dikunjungi pada DFS secara empiris membuktikan sifat penelusurannya yang mendalam, yang meskipun lengkap, tidak dirancang untuk optimasi pencarian rute terpendek. Pada test case kedua, terlihat bahwa meskipun jumlah transit yang dilakukan lebih banyak, metode BFS menghasilkan jumlah halte yang lebih sedikit yaitu 33 halte, dibandingkan dengan DFS yaitu 45 halte.

Berikut merupakan contoh test case pada program.

Gambar 4.1 Test Case “Blok M” menuju “Kota”



Gambar 4.1 Test Case “Jembatan Merah” menuju “Pulo Gadung 1”

V. Analisis

BFS melakukan penelusuran dengan kompleksitas wakt O(n + m), dengan n adalah jumlah halte dan m adalah jumlah koneksi antar halte. Setiap halte dimasukkan ke queue dan diambil sekali saja, dan setiap koneksi diperiksa sekali. Dari sisi memori, queue dan himpunan “visited” akan menampung paling banyak n halte ditambah memori untuk menyimpan jalur terpanjang juga hingga O(n).

Karena BFS selalu melebar per lapis, begitu menemui halte tujuan artinya jalur yang ditemukan sudah memiliki jumlah ganti bus dan hop halte paling sedikit. Pada kasus TransJakarta, BFS tidak akan menjelajah halte-halte yang tidak perlu, sehingga rute optimal ditemukan lebih cepat meski queue bisa memakan memori untuk rute sangat panjang.

DFS memiliki kompleksitas waktu O(n + m), karena setiap halte dan setiap koneksi dieksplorasi paling banyak sekali. Penggunaan stack dan himpunan “visited” serta penyimpanan jalur terpanjang memerlukan kompleksitas memori O(n).

Namun karena DFS mendahulukan penelusuran sedalam mungkin di satu koridor sebelum mencoba jalur lain, ia sering mengunjungi hampir semua halte di koridor awal meski rute yang lebih pendek memerlukan transfer lebih cepat. Dampaknya, hasil rute bisa lebih panjang dan waktu praktis menjadi lebih lama akibat banyak backtracking.

VI. Kesimpulan dan Saran

A. Kesimpulan

Breadth-First Search (BFS) dan Depth-First Search (DFS) dapat diimplementasikan untuk menyelesaikan masalah pencarian rute pada layanan bus Transjakarta. Berdasarkan pengujian yang dilakukan, kedua algoritma mampu menemukan lintasan yang valid dari halte asal ke halte tujuan, namun dengan karakteristik dan kualitas hasil yang berbeda. Algoritma BFS secara konsisten terbukti unggul dalam menemukan rute "terbaik" atau optimal, yaitu rute dengan jumlah total halte yang dilalui paling sedikit, sehingga lebih cocok untuk penggunaan praktis. Hal ini divalidasi pada kasus uji kompleks dari "Jembatan Merah" ke "Pulo Gadung 1", di mana BFS menghasilkan rute dengan 33 halte.

Tetapi, algoritma DFS menyajikan rute alternatif yang valid tetapi seringkali tidak efisien dan lebih panjang. Pada kasus uji yang sama, DFS menghasilkan rute dengan 45 halte. Perbedaan ini secara empiris membuktikan sifat penelusuran DFS yang mendalam dan tidak dirancang untuk optimasi pencarian rute terpendek. Meskipun kedua algoritma memiliki kompleksitas waktu teoretis yang sama, yaitu O(n+m), analisis kinerja menunjukkan bahwa BFS adalah pilihan yang lebih baik untuk menyelesaikan permasalahan ini secara efektif, sementara DFS berfungsi sebagai pembanding yang menunjukkan pendekatan strategi algoritma yang berbeda.

B. Saran

Melalui implementasi yang dilakukan dapat dilakukan beberapa pengembangan sebagai berikut.

1. Implementasi Graf Berbobot
   Pada implementasi yang dilakukan, setiap sisi dianggap memiliki bobot yang sama. Pengembangan selanjutnya dapat menambahkan bobot pada setiap sisi yang merepresentasikan estimasi waktu tempuh, jarak, atau bahkan biaya tiket.

2. Integrasi Data Real-time:
   Untuk meningkatkan akurasi dan kegunaan praktis, sistem dapat diintegrasikan dengan data real-time. Data ini bisa mencakup posisi bus aktual melalui GPS, kondisi lalu lintas, atau informasi mengenai penutupan halte sementara, sehingga dapat memberikan rekomendasi rute yang lebih dinamis dan relevan dengan kondisi di lapangan.

3. Pengembangan Antarmuka Pengguna (UI/UX)
   Sistem yang ada saat ini masih berbasis teks. Pengembangan antarmuka pengguna yang lebih visual dan interaktif, misalnya dengan menampilkan rute pada peta digital, dapat secara signifikan meningkatkan pengalaman pengguna dan kemudahan dalam memahami rute yang disarankan.

4. Memodelkan Jarak Jalan Kaki Antar Halte Transit
   Graf dapat diperluas dengan menambahkan node atau edge khusus yang merepresentasikan waktu atau jarak berjalan kaki antar halte transit yang berdekatan. Hal ini akan menghasilkan kalkulasi rute yang lebih realistis bagi pengguna yang perlu berpindah koridor.

VII. Lampiran

Github

https://github.com/Azekhiel/Pencarian-Rute-Transjakarta-dengan-BFS-dan-DFS

Video Link at Youtube

https://youtu.be/gdOTiYzVTdw

Video Link at Google Drive

https://drive.google.com/drive/folders/12w8Gk1_kNZ_SLRJrs6Byo92bbhkMeAHG?usp=drive_link

Acknowledgment

Saya ucapkan terima kasih yang tulus kepada Ibu Nur Ulfa Maulida, selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma K1 atas bimbingan, ilmu, dan kesempatan yang

telah diberikan dalam penyusunan makalah ini. Ucapan terima kasih juga saya sampaikan kepada seluruh tim dosen pengampu dan tim asisten mata kuliah Strategi Algoritma atas segala dukungan dan materi pembelajaran yang sangat bermanfaat selama satu semester.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[2] R. Sedgewick dan K. Wayne, *Algorithms*, 4th ed. Boston: Addison-Wesley, 2011.

[3] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 4th ed. Boston: Pearson, 2013.

[4] Rinaldi Munir, *Strategi Algoritma*, Sekolah Teknik Elektro dan Informatika (STEI) ITB. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24. [Diakses: 22 Jun 2025].

[5] V. Vivianisa, "13 Rute Koridor TransJakarta, Ketahui sebelum Berangkat ke Kantor," *Glints*, 15-Jan-2024. [Online]. Available: https://glints.com/id/lowongan/rute-transjakarta/. [Accessed: 22 Jun 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, 24 Juni 2025

William Gerald Briandelo (13222061)