

LAPORAN PRAKTIKUM UTS KECERDASAN BUATAN
B1-KELOMPOK 1



**Sistem Rekomendasi Jalur Pengiriman Barang
dengan Metode *Breadth-First Search (BFS)* dan
Metode *Depth-First Search (DFS)***

Disusun Oleh :

Cindy Alfira (434231001)
Shendy Tria Amelyana (434231003)
Muhammad Zaky Irly (434231025)

UNIVERSITAS AIRLANGGA

2025

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
DAFTAR LAMPIRAN.....	5
BAB I.....	6
BREADTH-FIRST SEARCH.....	6
BAB II.....	7
DEPTH-FIRST SEARCH.....	7
BAB III.....	8
RUANG KEADAAN.....	8
a. Definisi Ruang Keadaan dalam Proses Pengiriman Barang.....	8
b. Representasi Ruang Keadaan sebagai Graph.....	8
c. Properti Ruang Keadaan.....	9
d. Keadaan Awal dan Tujuan Graph.....	9
BAB IV.....	12
GRAPH DAN POHON PELACAKAN.....	12
a. Pohon Pelacakan.....	12
b. Graph.....	13
c. Kesimpulan.....	14
BAB V.....	16
VISUALISASI.....	16
a. Gambaran Visualisasi.....	16
b. Cara Kerja Visualisasi.....	16
c. Pohon Pelacakan BFS dan DFS.....	19
d. Penjelasan Visualisasi.....	20
BAB VI.....	22
CODING.....	22

DAFTAR GAMBAR

Gambar 1 Graph BFS.....	13
Gambar 2 Graph DFS.....	14
Gambar 3 Gambaran Visualisasi.....	16
Gambar 4 Cara Kerja Visualisasi.....	16
Gambar 5 Mode Visualisasi Perbandingan BFS & DFS.....	17
Gambar 6 Pelacakan BFS.....	19
Gambar 7 Pelacakan DFS.....	20

DAFTAR TABEL

Tabel 1 Aturan Perpindahan dari Satu Kota ke Kota Lain.....	11
Tabel 2 Perbandingan BFS dan DFS.....	15

DAFTAR LAMPIRAN

(Semua lampiran telah disertakan langsung dalam isi laporan)

BAB I

BREADTH-FIRST SEARCH

Breadth-First Search (BFS) adalah salah satu algoritma pencarian dalam struktur data graf yang digunakan untuk menelusuri node atau simpul dalam urutan yang berlapis-lapis. BFS bekerja dengan mengeksplorasi semua tetangga dari simpul yang sedang dikunjungi sebelum melanjutkan ke simpul pada level berikutnya. Algoritma ini menggunakan struktur data queue (*FIFO - First In First Out*) untuk mengelola antrian node yang akan dikunjungi.

BFS sangat efektif digunakan dalam berbagai kasus pencarian jalur terpendek pada graf tak berbobot, sistem navigasi, serta pemetaan jaringan transportasi. Dalam konteks penelitian ini, BFS diterapkan untuk membangun Sistem Rekomendasi Jalur Pengiriman Barang, guna mencari rute optimal dari gudang ke kota tujuan dengan jumlah persinggahan paling sedikit.

Karakteristik utama BFS:

1. Menelusuri node per level: Semua node dalam satu tingkat eksplorasi dikunjungi sebelum berpindah ke tingkat berikutnya.
2. Menjamin jalur terpendek dalam graf tak berbobot: Jika terdapat beberapa jalur dari titik awal ke tujuan, BFS akan menemukan jalur dengan jumlah simpul paling sedikit.
3. Memiliki kompleksitas waktu $O(V + E)$: Di mana V adalah jumlah simpul (vertex) dan E adalah jumlah sisi (edge).
4. Menggunakan memori yang lebih besar dibanding DFS: Karena BFS menyimpan semua node pada satu tingkat sebelum melanjutkan ke tingkat berikutnya.

Dalam konteks Sistem Rekomendasi Jalur Pengiriman Barang, BFS membantu dalam mencari jalur pengiriman barang dengan jumlah transit paling sedikit, sehingga meningkatkan efisiensi waktu pengiriman.

BAB II

DEPTH-FIRST SEARCH

Depth-First Search (DFS) adalah algoritma pencarian dalam struktur data graf yang bekerja dengan menelusuri node secara mendalam sebelum kembali ke node sebelumnya untuk mengeksplorasi jalur lain. Algoritma ini menggunakan struktur data stack (baik secara eksplisit maupun melalui rekursi) untuk menelusuri jalur secara menyeluruh sebelum mundur. DFS digunakan dalam berbagai kasus seperti pencarian solusi dalam permainan (*game tree*), analisis hubungan dalam jaringan sosial, dan pemecahan masalah rekursif lainnya.

Karakteristik utama DFS:

1. Menelusuri jalur secara mendalam: DFS akan menjelajahi satu jalur hingga ke ujung sebelum kembali dan menelusuri jalur lain.
2. Tidak menjamin jalur terpendek: DFS tidak selalu menemukan solusi optimal dalam graf yang memiliki banyak kemungkinan jalur.
3. Kompleksitas waktu $O(V + E)$: Sama dengan BFS, di mana V adalah jumlah simpul dan E adalah jumlah sisi.
4. Menggunakan lebih sedikit memori dibanding BFS: Karena DFS hanya menyimpan jalur yang sedang ditelusuri dalam stack.

BAB III

RUANG KEADAAN

Dalam sistem ini, ruang keadaan merepresentasikan seluruh kemungkinan kondisi yang dapat terjadi dalam proses pengiriman barang dari satu kota ke kota tujuan. Identifikasi ruang keadaan sangat penting untuk memahami bagaimana sistem dapat menyusun jalur terbaik menggunakan metode *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS).

a. Definisi Ruang Keadaan dalam Proses Pengiriman Barang

Ruang keadaan dalam konteks sistem rekomendasi jalur pengiriman barang adalah semua kemungkinan lokasi dan rute yang dapat dilalui oleh barang dari titik awal ke tujuan. Setiap keadaan (*state*) merepresentasikan posisi barang dalam suatu kota tertentu. Transisi antar keadaan terjadi ketika barang berpindah dari satu kota ke kota lain melalui jalur yang tersedia. Dalam BFS, sistem menjelajahi ruang keadaan secara melebar untuk menemukan jalur dengan jumlah persinggahan paling sedikit, sedangkan dalam DFS, sistem menelusuri satu jalur secara mendalam sebelum kembali mencoba jalur lain jika diperlukan.

1. Ruang keadaan dalam konteks sistem rekomendasi jalur pengiriman barang adalah semua kemungkinan lokasi dan rute yang dapat dilalui oleh barang dari titik awal ke tujuan.
2. Setiap keadaan (*state*) merepresentasikan posisi barang dalam suatu kota tertentu.
3. Transisi antar keadaan terjadi ketika barang berpindah dari satu kota ke kota lain melalui jalur yang tersedia.

b. Representasi Ruang Keadaan sebagai Graph

Dalam sistem ini, ruang keadaan direpresentasikan dalam bentuk graph tidak berbobot (karena kita hanya menghitung jumlah persinggahan, bukan jarak).

1. Simpul (*Node*): Kota-kota yang dapat menjadi lokasi persinggahan barang.
2. Sisi (*Edge*): Jalur Pengiriman barang yang menghubungkan kota-kota tersebut.

BFS menelusuri semua simpul pada setiap level sebelum melanjutkan ke level berikutnya, sementara DFS memulai dari simpul awal dan menelusuri enced satu sisi secara mendalam hingga mencapai simpul akhir atau mundur untuk mencoba jalur lain.

c. Properti Ruang Keadaan

Ruang keadaan dalam sistem ini memiliki beberapa karakteristik yang mempengaruhi bagaimana BFS dan DFS digunakan untuk menemukan jalur pengiriman:

1. *Ruang Keadaan Bersifat Diskrit*: Ruang keadaan terdiri dari kota-kota yang direpresentasikan sebagai entitas terpisah. Barang tidak bisa berhenti di tengah perjalanan antara dua kota, melainkan harus berada di salah satu kota tertentu, seperti kota A, B, atau C. BFS memastikan eksplorasi merata, sedangkan DFS menelusuri satu cabang secara berurutan.
2. *Ruang Keadaan Bersifat Terbatas (Finite State Space)*: Jumlah keadaan terbatas pada jumlah kota dan jalur yang telah ditentukan. BFS menjamin solusi optimal, sedangkan DFS akan selesai menjelajahi semua kemungkinan meskipun tidak selalu efisien.
3. *Ruang Keadaan Bersifat Terhubung (Connected State Space)*: Setiap kota dapat dijangkau dari kota lain melalui jalur pengiriman. BFS menemukan jalur terpendek, sedangkan DFS pasti mencapai tujuan tetapi mungkin melalui jalur lebih panjang.
4. *Ruang Keadaan Tidak Berbobot*: Jalur pengiriman tidak memiliki bobot, sehingga setiap perpindahan dianggap sama. BFS sangat cocok karena menjamin jumlah persinggahan paling sedikit, sementara DFS hanya menemukan jalur berdasarkan urutan eksplorasi tanpa jaminan optimalitas.

d. Keadaan Awal dan Tujuan Graph

Berikut adalah representasi Keadaan Awal dan Tujuan dalam bentuk Graph untuk sistem rekomendasi jalur pengiriman barang menggunakan BFS dan DFS.

Contoh:

Ada beberapa kota yang saling terhubung melalui jalur pengiriman. Tidak ada bobot pada jalur pengiriman, sehingga satu perpindahan dari satu kota ke kota lain dianggap sama nilainya.

Permasalahan:

- *BFS*: Menemukan jalur dengan jumlah persinggahan paling sedikit dengan menjelajahi level demi level, misalnya Gudang → A → F → Tujuan.
- *DFS*: Menelusuri satu jalur secara mendalam, misalnya Gudang → A → C → F → Tujuan, dan mundur untuk mencoba jalur lain jika perlu, tanpa jaminan jalur terpendek.

a. Identifikasi Ruang Keadaan

Masalah ini dapat dilambangkan dengan (x, y) , di mana:

- $x = \text{Gudang}$
- $y = \text{Kota tujuan}$

BFS menelusuri ruang keadaan secara sistematis untuk jalur terpendek, sedangkan DFS memilih satu jalur dari Gudang dan mengikuti aturan pergerakan hingga mencapai Tujuan atau mundur untuk mencoba alternatif.

b. Keadaan Awal dan Tujuan

- Keadaan awal: Barang berada di Gudang
- Keadaan tujuan: Barang sampai di kota Tujuan

Dengan BFS, barang mencapai tujuan melalui jalur terpendek seperti Gudang → A → F → Tujuan.

Dengan DFS, barang mungkin mengikuti jalur seperti Gudang → A → D → F → Tujuan, tergantung cabang yang dipilih pertama kali.

c. Aturan Pergerakan

Tabel di bawah menunjukkan aturan perpindahan dari satu kota ke kota lain.

Aturan Ke-	Kondisi Awal	Kondisi Tujuan	Aturan
1	(Gudang, -)	(A, Gudang)	Barang berpindah dari Gudang ke A
2	(Gudang, -)	(B, Gudang)	Barang berpindah dari Gudang ke B
3	(A, Gudang)	(C, A)	Barang berpindah dari A ke C
4	(A, Gudang)	(D, A)	Barang berpindah dari A ke D
5	(B, Gudang)	(E, B)	Barang berpindah dari B ke E
6	(C, A)	(F, C)	Barang berpindah dari C ke F
7	(D, A)	(F, D)	Barang berpindah dari D ke F

8	(E, B)	(F, E)	Barang berpindah dari E ke F
9	(F, C/D/E)	(Tujuan, F)	Barang berpindah dari F ke Tujuan

Tabel I Aturan Perpindahan dari Satu Kota ke Kota Lain.

BAB IV

GRAPH DAN POHON PELACAKAN

a. Pohon Pelacakan

1. BFS

Level 0: Gudang

/ \

Level 1: A B

/ \ /

Level 2: C D E

\ | /

Level 3: F

|

Level 4: (Tujuan)

Jalur: Gudang → A → F → Tujuan

Jumlah persinggahan: 3

2. DFS

Level 0: Gudang

/ \

Level 1: A B

/ \ /

Level 2: C D E

\ | /

Level 3: F

|

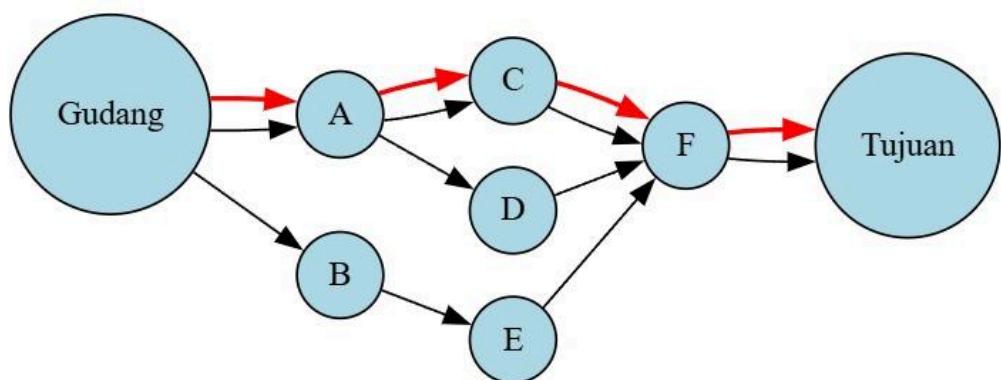
Level 4: (Tujuan)

Jalur yang Mungkin:

1. Gudang → A → C → F → Tujuan (4 persinggahan)
2. Gudang → A → D → F → Tujuan (4 persinggahan)
3. Gudang → B → E → F → Tujuan (4 persinggahan)

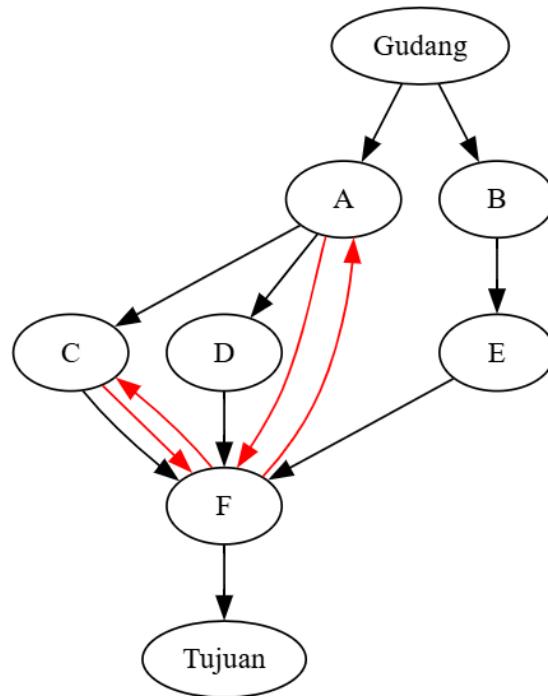
b. Graph

1. BFS



Gambar 1 Graph BFS

2. DFS



Gambar 2 Graph DFS

c. Kesimpulan

Dalam sistem rekomendasi jalur pengiriman barang, ruang keadaan direpresentasikan sebagai graf tak berbobot. Setiap kota berfungsi sebagai simpul (*node*) dan setiap jalur pengiriman sebagai sisi (*edge*). Tujuan dari sistem ini adalah untuk menemukan rute pengiriman dengan jumlah persinggahan paling sedikit agar proses pengiriman lebih efisien.

Salah satu metode pencarian yang direkomendasikan dalam kasus ini adalah Breadth-First Search (BFS). BFS bekerja dengan menelusuri simpul berdasarkan level atau kedalaman yang sama sebelum melanjutkan ke level berikutnya. Pendekatan ini memungkinkan sistem untuk menemukan jalur terpendek dari gudang ke tujuan secara sistematis dan efisien, terutama pada graf tak berbobot.

Proses Pencarian Jalur:

1. Jalur yang ditemukan dengan BFS akan memiliki jumlah langkah minimal dari simpul awal ke simpul tujuan.
2. Aturan pergerakan barang ditentukan berdasarkan struktur pohon yang dibentuk oleh BFS, mengikuti setiap simpul dari gudang hingga kota tujuan.
3. Dengan struktur ini, barang dapat dikirim melalui jalur yang paling singkat dalam hal jumlah simpul yang dilalui.

Perbandingan BFS dan DFS:

Aspek	BFS (Breadth-First Search)	DFS (Depth-First Search)
Strategi Penelusuran	Menelusuri simpul berdasarkan level	Menelusuri simpul secara mendalam terlebih dahulu
Jaminan Jalur Terpendek	Ya, untuk graf tak berbobot	Tidak selalu
Efisiensi Jalur	Lebih efisien untuk menemukan rute dengan langkah minimal	Dapat menjelajah jalur panjang terlebih dahulu
Cocok untuk Navigasi?	Cocok untuk sistem pengiriman barang	Kurang cocok jika tujuan adalah rute terpendek
Struktur Data	Menggunakan Queue	Menggunakan Stack atau rekursi

Tabel 2 Perbandingan BFS dan DFS

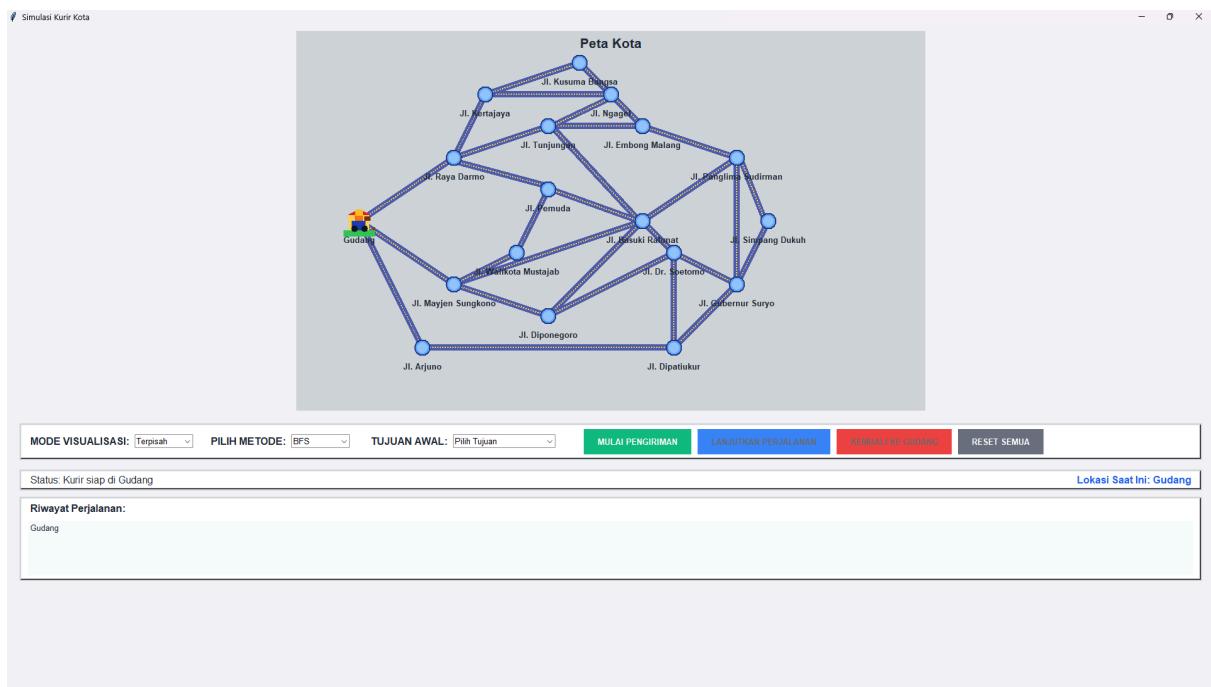
Berdasarkan struktur graf dan kebutuhan sistem dalam menemukan jalur dengan jumlah persinggahan paling sedikit, metode BFS direkomendasikan untuk digunakan dalam skenario ini. Dengan kemampuannya menelusuri jalur secara sistematis dan efisien, BFS membantu sistem dalam menyusun rute pengiriman yang optimal sesuai dengan struktur jaringan pengiriman.

BAB V

VISUALISASI

a. Gambaran Visualisasi

Pada bagian ini, aplikasi yang dibuat menampilkan simulasi pengiriman kurir dari "Gudang" ke berbagai lokasi tujuan di kota Surabaya menggunakan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS). Setiap lokasi direpresentasikan sebagai simpul (node), sedangkan jalur antar lokasi direpresentasikan sebagai sisi (edge) dengan koordinat yang telah ditentukan dalam node_pos.



Gambar 3 Gambaran Visualisasi

b. Cara Kerja Visualisasi

This screenshot shows the search interface of the delivery simulation application. It features a top bar with dropdown menus for "Pilih Metode" (BFS, Tercepat, DFS, Eksplorasi), "Tujuan Awal" (Pilih Tujuan), and "Tujuan Berikutnya" (Pilih Tujuan). Below the bar are buttons for "Mulai Pengiriman", "Lanjutkan Perjalanan", "Kembali ke Gudang", and "Reset Semua". A status message "Status: Kurir siap di Gudang" is displayed above a "Riwayat Perjalanan" section which shows "Gudang".

Gambar 4 Cara Kerja Visualisasi

Pilihan Kota Awal & Kota Tujuan

- Penjelasan di Gambar

Pengguna dapat memilih kota asal dan kota tujuan dari daftar dropdown. Dalam contoh gambar, Surabaya dipilih sebagai kota awal, dan Banyuwangi sebagai kota tujuan.

- Penjelasan di Kode:

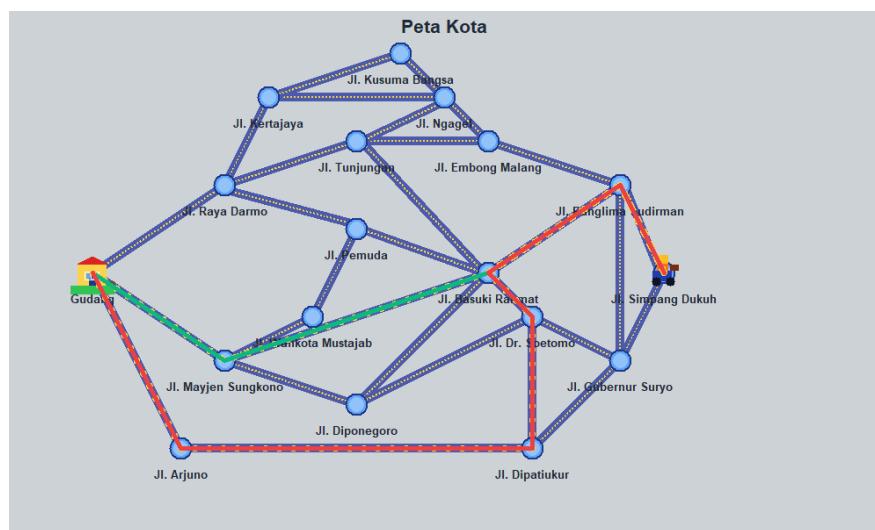
1. Dalam kode, dropdown untuk memilih kota awal dan tujuan diimplementasikan menggunakan ttk.Combobox di class CourierApp.
2. Dalam simulasi ini, kota awal selalu "Gudang" (default self.current_location = "Gudang"). Namun, pengguna memilih tujuan awal melalui dropdown self.destination_var di self.dropdown (dari daftar node_pos.keys()).
3. Pengguna memilih tujuan awal melalui dropdown pertama (self.dropdown) dan tujuan berikutnya melalui dropdown kedua (self.next_dropdown).

Contoh: Dalam kode, daftar kota seperti "Jl. Raya Darmo", "Jl. Tunjungan", "Banyuwangi", dll., diambil dari dictionary node_pos. Dropdown memungkinkan pengguna memilih tujuan seperti "Banyuwangi" (sesuai gambar).

Mode Visualisasi

- Penjelasan di Gambar:

1. Terpisah (BFS & DFS): Menampilkan hasil pencarian jalur dari kedua algoritma (BFS dan DFS) secara terpisah.
2. Gabungan (BFS & DFS): Menampilkan perbandingan jalur dari kedua algoritma dalam satu visualisasi, sehingga perbedaan jalur BFS dan DFS dapat dilihat.



Gambar 5 Mode Visualisasi Perbandingan BFS & DFS

- Penjelasan di Kode:

1. Pilihan Metode: Dalam kode, pengguna dapat memilih metode pencarian jalur (BFS atau DFS) melalui dropdown self.method_var di self.method_dropdown (opsi: "BFS" atau "DFS").
- Fungsi BFS dan DFS:
 1. bfs(start, goal): Menggunakan algoritma Breadth-First Search untuk mencari jalur terpendek dari start ke goal.
 2. dfs(start, goal): Menggunakan algoritma Depth-First Search untuk mencari jalur dari start ke goal.
- Mode Terpisah:
 1. Dalam kode, hanya satu metode (BFS atau DFS) yang digunakan pada satu waktu berdasarkan pilihan pengguna (self.get_search_method()). Jalur yang dihasilkan ditampilkan melalui animasi kurir di canvas (self.animate_courier(route)).

Contoh: Jika pengguna memilih BFS, maka jalur seperti "Gudang → Jl. Raya Darmo → Jl. Tunjungan → Banyuwangi" (seperti di gambar) akan ditampilkan.

- Mode Gabungan:
 1. Kode secara eksplisit menampilkan mode gabungan (BFS dan DFS bersamaan). Bisa dengan memanggil kedua fungsi (bfs dan dfs) dan menampilkan jalur masing-masing dengan warna berbeda di canvas.

Tombol "Cari Jalur"

1. Penjelasan di Gambar: Setelah tombol "Cari Jalur" ditekan, aplikasi akan menampilkan hasil pencarian jalur menggunakan kedua algoritma (BFS dan DFS).
2. Penjelasan di Kode:

Dalam kode, tombol "Cari Jalur" diwakili oleh beberapa tombol dengan fungsi berbeda:

- i. Tombol "Mulai Pengiriman" (self.start_button): Memulai pengiriman dari "Gudang" ke tujuan awal yang dipilih (self.start_delivery()). Ini memanggil algoritma BFS atau DFS atau Gabungan (berdasarkan pilihan metode) untuk mencari jalur, lalu menampilkan animasi kurir di canvas.
- ii. Tombol "Lanjutkan Perjalanan" (self.continue_button): Melanjutkan perjalanan dari lokasi saat ini ke tujuan berikutnya (self.continue_to_new_destination()). Ini juga menggunakan algoritma yang dipilih untuk mencari jalur baru.

- iii. Tombol "Kembali ke Gudang" (self.home_button): Mengembalikan kurir ke "Gudang" dengan mencari jalur dari lokasi saat ini ke "Gudang" (self.return_home()).
- iv. Tombol "Reset Semua" (self.reset_button): Mengatur ulang simulasi ke kondisi awal (self.reset_courier()).

Setelah tombol "Mulai Pengiriman" atau "Lanjutkan Perjalanan" ditekan, kode akan:

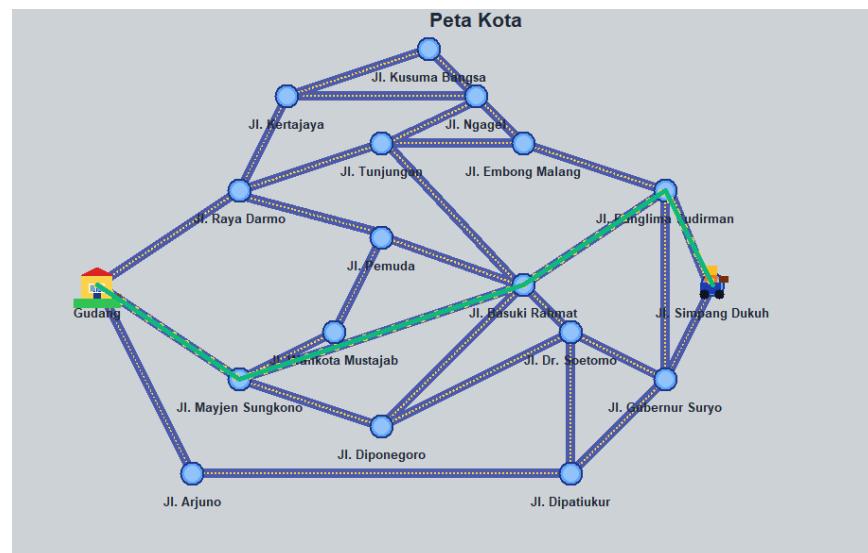
- v. Memanggil fungsi pencarian jalur (bfs atau dfs).
- vi. Menampilkan jalur di canvas dengan garis (self.path_line dan self.path_glow).
- vii. Menganimasikan pergerakan kurir dari satu node ke node berikutnya (self.animate_courier()).
- viii. Memperbarui status dan riwayat perjalanan di self.status_label dan self.journey_log.

c. Pohon Pelacakan BFS dan DFS

Untuk contoh, kita akan melacak rute dari "Gudang" ke "Jl. Simpang Dukuh".

Pohon Pelacakan BFS

BFS mencari jalur terpendek dengan menjelajahi semua tetangga pada satu level sebelum ke level berikutnya.



Gambar 6 Pelacakan BFS

Langkah-langkah BFS:

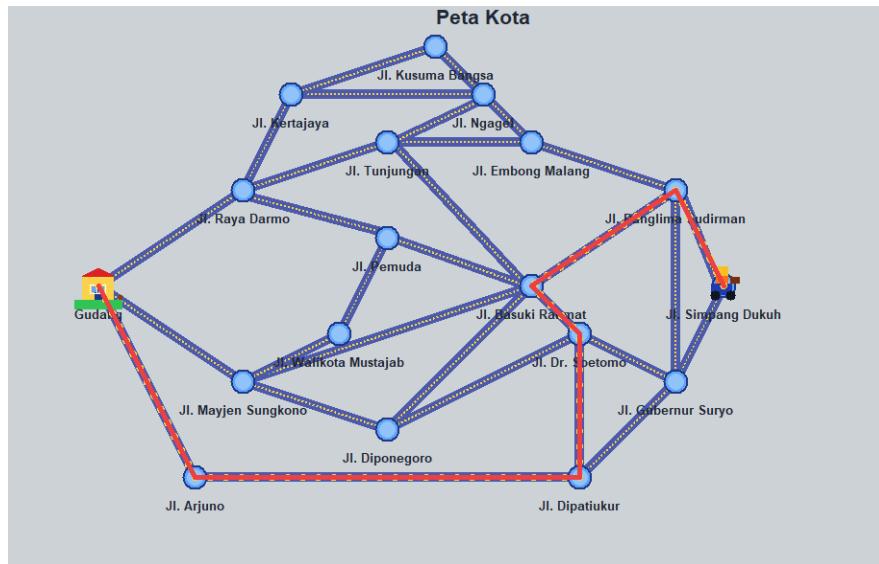
- "Gudang" → "Jl. Mayjen Sungkono"
- "Jl. Mayjen Sungkono" → "Jl. Basuki Rahmat"

- "Jl. Basuki Rahmat" → "Jl. Panglima Sudirman"
- "Jl. Panglima Sudirman" → "Jl. Simpang Dukuh" (tujuan ditemukan)

Jalur: ["Gudang", "Jl. Mayjen Sungkono", "Jl. Basuki Rahmat", "Jl. Panglima Sudirman", "Jl. Simpang Dukuh"]

Pohon Pelacakan DFS

DFS menjelajahi satu cabang hingga akhir sebelum mundur.



Gambar 7 Pelacakan DFS

Langkah-langkah DFS:

- "Gudang" → "Jl. Arjuno" (terakhir dalam daftar)
- "Jl. Arjuno" → "Jl. Dipatiukur"
- "Jl. Dipatiukur" → "Jl. Dr. Soetomo"
- "Jl. Dr. Soetomo" → "Jl. Basuki Rahmat"
- "Jl. Basuki Rahmat" → "Jl. Panglima Sudirman"
- "Jl. Panglima Sudirman" → "Jl. Simpang Dukuh" (tujuan ditemukan)

Jalur: ["Gudang", "Jl. Arjuno", "Jl. Dipatiukur", "Jl. Dr. Soetomo", "Jl. Basuki Rahmat", "Jl. Panglima Sudirman", "Jl. Simpang Dukuh"]

d. Penjelasan Visualisasi

Visualisasi dalam aplikasi ini mengintegrasikan algoritma pencarian (BFS dan DFS) dengan elemen grafis interaktif

1. Warna Hijau menunjukkan metode Breadth-First Search (BFS).
2. Warna Merah menunjukkan metode Depth-First Search (DFS).

3. Peta Statis dibuat oleh draw_map dengan jalur abu-abu dan node berlabel. Ikon gudang dan lingkaran biru membedakan "Gudang" dari lokasi lain.
4. Feedback Real-time status menunjukkan rute, perjalanan, dan kedatangan. Log perjalanan mencatat semua node yang dilalui, diperbarui oleh update_journey_log.

BAB VI

CODING

```
● ● ●
1 import tkinter as tk
2 from tkinter import ttk
3 import time
4 from collections import deque
5 from PIL import Image, ImageTk
6 import tkinter.font as tkFont
7
8 # Data koordinat node dan graf
9 node_pos = {
10     "Gudang": (100, 300), "Jl. Raya Darmo": (250, 200), "Jl. Mayjen Sungkono": (250, 400),
11     "Jl. Tunjungan": (400, 150), "Jl. Pemuda": (400, 250), "Jl. Diponegoro": (400, 450),
12     "Jl. Basuki Rahmat": (550, 300), "Jl. Embong Malang": (550, 150), "Jl. Panglima Sudirman": (700, 200),
13     "Jl. Gubernur Suryo": (700, 400), "Jl. Kertajaya": (300, 100), "Jl. Ngagel": (500, 100),
14     "Jl. Dr. Soetomo": (600, 350), "Jl. Walikota Mustajab": (350, 350), "Jl. Kusuma Bangsa": (450, 50),
15     "Jl. Arjuno": (200, 500), "Jl. Dipatiukur": (600, 500), "Jl. Simpang Dukuh": (750, 300)
16 }
17
18 graph = {
19     "Gudang": ["Jl. Raya Darmo", "Jl. Mayjen Sungkono", "Jl. Arjuno"],
20     "Jl. Raya Darmo": ["Gudang", "Jl. Tunjungan", "Jl. Pemuda", "Jl. Kertajaya"],
21     "Jl. Mayjen Sungkono": ["Gudang", "Jl. Diponegoro", "Jl. Basuki Rahmat", "Jl. Walikota Mustajab"],
22     "Jl. Tunjungan": ["Jl. Raya Darmo", "Jl. Basuki Rahmat", "Jl. Embong Malang", "Jl. Ngagel"],
23     "Jl. Pemuda": ["Jl. Raya Darmo", "Jl. Basuki Rahmat", "Jl. Walikota Mustajab"],
24     "Jl. Diponegoro": ["Jl. Mayjen Sungkono", "Jl. Basuki Rahmat", "Jl. Dr. Soetomo"],
25     "Jl. Basuki Rahmat": ["Jl. Tunjungan", "Jl. Pemuda", "Jl. Diponegoro", "Jl. Panglima Sudirman", "Jl. Dr. Soetomo"],
26     "Jl. Embong Malang": ["Jl. Tunjungan", "Jl. Panglima Sudirman", "Jl. Ngagel"],
27     "Jl. Panglima Sudirman": ["Jl. Basuki Rahmat", "Jl. Embong Malang", "Jl. Gubernur Suryo", "Jl. Simpang Dukuh"],
28     "Jl. Gubernur Suryo": ["Jl. Panglima Sudirman", "Jl. Dr. Soetomo", "Jl. Dipatiukur", "Jl. Simpang Dukuh"],
29     "Jl. Kertajaya": ["Jl. Raya Darmo", "Jl. Ngagel", "Jl. Kusuma Bangsa"],
30     "Jl. Ngagel": ["Jl. Kertajaya", "Jl. Tunjungan", "Jl. Kusuma Bangsa", "Jl. Embong Malang"],
31     "Jl. Dr. Soetomo": ["Jl. Gubernur Suryo", "Jl. Dipatiukur", "Jl. Diponegoro", "Jl. Basuki Rahmat"],
32     "Jl. Walikota Mustajab": ["Jl. Pemuda", "Jl. Mayjen Sungkono"],
33     "Jl. Kusuma Bangsa": ["Jl. Kertajaya", "Jl. Ngagel"],
34     "Jl. Arjuno": ["Gudang", "Jl. Dipatiukur"],
35     "Jl. Dipatiukur": ["Jl. Arjuno", "Jl. Gubernur Suryo", "Jl. Dr. Soetomo"],
36     "Jl. Simpang Dukuh": ["Jl. Panglima Sudirman", "Jl. Gubernur Suryo"]
37 }
38
39 path_adjustments = {
40     ("Jl. Raya Darmo", "Jl. Pemuda"): [(0, 5), (0, -5), "#4B5EAA"],
41     ("Jl. Tunjungan", "Jl. Basuki Rahmat"): [(-5, 0), (-5, 0), "#4B5EAA"],
42     ("Jl. Pemuda", "Jl. Basuki Rahmat"): [(5, 0), (-5, 0), "#4B5EAA"],
43     ("Jl. Ngagel", "Jl. Tunjungan"): [(0, 5), (5, 0), "#4B5EAA"],
44     ("Jl. Kertajaya", "Jl. Ngagel"): [(5, 0), (-5, 0), "#4B5EAA"],
45     ("Jl. Panglima Sudirman", "Jl. Gubernur Suryo"): [(0, 5), (0, -5), "#4B5EAA"],
46     ("Jl. Panglima Sudirman", "Jl. Simpang Dukuh"): [(-5, 0), (-5, 0), "#4B5EAA"],
47     ("Jl. Walikota Mustajab", "Jl. Simpang Dukuh"): [(5, 0), (-5, 0), "#4B5EAA"],
48     ("Jl. Dr. Soetomo", "Jl. Walikota Mustajab"): [(0, 5), (0, -5), "#4B5EAA"],
49 }
50 }
```



```
1 def bfs(start, goal):
2     queue = deque([[start]])
3     visited = set()
4     while queue:
5         path = queue.popleft()
6         node = path[-1]
7         if node in visited:
8             continue
9         visited.add(node)
10        if node == goal:
11            return path
12        for neighbor in graph.get(node, []):
13            queue.append(path + [neighbor])
14    return None
15
16 def dfs(start, goal):
17     stack = [[start]]
18     visited = set()
19     while stack:
20         path = stack.pop()
21         node = path[-1]
22         if node == goal:
23             return path
24         if node not in visited:
25             visited.add(node)
26             for neighbor in graph.get(node, []):
27                 if neighbor not in visited:
28                     new_path = path + [neighbor]
29                     stack.append(new_path)
30     return None
31
```

```

1  class CourierApp:
2      def __init__(self, root):
3          self.root = root
4          self.root.title("Simulasi Kurir Kota")
5          self.root.geometry("1000x800")
6          self.root.configure(bg="#F3F4F6")
7
8      # Canvas
9      self.canvas = tk.Canvas(root, width=1000, height=600, bg="#E5E7EB", highlightthickness=0)
10     self.canvas.pack(pady=10)
11     self.canvas.create_rectangle(0, 0, 1000, 600, fill="#D1D5DB", outline="")
12     self.canvas.create_text(500, 20, text="Peta Kota", font=("Helvetica", 16, "bold"), fill="#1F2937")
13
14     self.current_location = "Gudang"
15     self.is_delivery_started = False # Untuk melacak apakah pengiriman sudah dimulai
16
17     # Control frame
18     self.control_frame = tk.Frame(root, bg="#FFFFFF", relief=tk.RAISED, borderwidth=2, highlightbackground="#D1D5DB", highlightthickness=2)
19     self.control_frame.pack(pady=10, padx=20, fill=tk.X)
20
21     # Mode selection (Terpisah atau Gabungan)
22     self.mode_frame = tk.Frame(self.control_frame, bg="#FFFFFF")
23     self.mode_frame.pack(side=tk.LEFT, padx=10, pady=5)
24     tk.Label(self.mode_frame, text="MODE VISUALISASI:", font=("Helvetica", 12, "bold"), bg="#FFFFFF", fg="#1F2937").pack(side=tk.LEFT)
25     self.mode_var = tk.StringVar(value="Terpisah")
26     self.mode_dropdown = ttk.Combobox(self.mode_frame, textvariable=self.mode_var,
27                                     values=["Terpisah", "Gabungan"], state="readonly", width=10, font=("Helvetica", 10))
28     self.mode_dropdown.pack(side=tk.LEFT, padx=5)
29
30     # Method selection (hanya aktif jika mode Terpisah)
31     self.method_frame = tk.Frame(self.control_frame, bg="#FFFFFF")
32     self.method_frame.pack(side=tk.LEFT, padx=10, pady=5)
33     tk.Label(self.method_frame, text="PILIH METODE:", font=("Helvetica", 12, "bold"), bg="#FFFFFF", fg="#1F2937").pack(side=tk.LEFT)
34     self.method_var = tk.StringVar(value="BFS")
35     self.method_dropdown = ttk.Combobox(self.method_frame, textvariable=self.method_var,
36                                         values=["BFS", "DFS"], state="readonly", width=10, font=("Helvetica", 10))
37     self.method_dropdown.pack(side=tk.LEFT, padx=5)
38
39     # Destination (awalnya "Tujuan Awal")
40     self.sel_frame1 = tk.Frame(self.control_frame, bg="#FFFFFF")
41     self.sel_frame1.pack(side=tk.LEFT, padx=15, pady=5)
42     self.destination_label = tk.Label(self.sel_frame1, text="TUJUAN AWAL:", font=("Helvetica", 12, "bold"), bg="#FFFFFF", fg="#1F2937")
43     self.destination_label.pack(side=tk.LEFT)
44     self.destination_var = tk.StringVar()
45     self.dropdown = ttk.Combobox(self.sel_frame1, textvariable=self.destination_var,
46                                values=list(node_pos.keys()), state="readonly", width=20, font=("Helvetica", 10))
47     self.dropdown.pack(side=tk.LEFT, padx=5)
48     self.dropdown.set("Pilih Tujuan")

```

```

1 # Buttons
2     self.btn_frame = tk.Frame(self.control_frame, bg="#FFFFFF")
3     self.btn_frame.pack(side=tk.LEFT, padx=20, pady=5)
4     self.start_button = tk.Button(self.btn_frame, text="MULAI PENGIRIMAN", command=self.start_delivery,
5                                     bg="#10B981", fg="white", font=("Helvetica", 10, "bold"), relief=tk.FLAT, padx=20, pady=10,
6                                     activebackground="#059669", bd=0, highlightthickness=0)
7     self.start_button.pack(side=tk.LEFT, padx=5)
8
9     self.continue_button = tk.Button(self.btn_frame, text="LANJUTKAN PERJALANAN", command=self.continue_to_new_destination,
10                                    bg="#3B82F6", fg="white", font=("Helvetica", 10, "bold"), relief=tk.FLAT, padx=20, pady=10,
11                                    state=tk.DISABLED, activebackground="#2563EB", bd=0, highlightthickness=0)
12     self.continue_button.pack(side=tk.LEFT, padx=5)
13
14     self.home_button = tk.Button(self.btn_frame, text="KEMBALI KE GUDANG", command=self.return_home,
15                                   bg="#EF4444", fg="white", font=("Helvetica", 10, "bold"), relief=tk.FLAT, padx=20, pady=10,
16                                   state=tk.DISABLED, activebackground="#DC2626", bd=0, highlightthickness=0)
17     self.home_button.pack(side=tk.LEFT, padx=5)
18
19     self.reset_button = tk.Button(self.btn_frame, text="RESET SEMUA", command=self.reset_courier,
20                                   bg="#6B7280", fg="white", font=("Helvetica", 10, "bold"), relief=tk.FLAT, padx=20, pady=10,
21                                   activebackground="#4B5563", bd=0, highlightthickness=0)
22     self.reset_button.pack(side=tk.LEFT, padx=5)
23
24 # Status frame
25     self.status_frame = tk.Frame(root, bg="#FFFFFF", relief=tk.RAISED, borderwidth=2, highlightbackground="#D1D5DB", highlightthickness=2)
26     self.status_frame.pack(pady=5, padx=20, fill=tk.X)
27     self.status_label = tk.Label(self.status_frame, text="Status: Kurir siap di Gudang", font=("Helvetica", 12), bg="#FFFFFF", fg="#1F2937", anchor="w")
28     self.status_label.pack(side=tk.LEFT, padx=10)
29     self.location_label = tk.Label(self.status_frame, text="Lokasi Saat Ini: Gudang", font=("Helvetica", 12, "bold"), bg="#FFFFFF", fg="#2563EB", anchor="e")
30     self.location_label.pack(side=tk.RIGHT, padx=10)
31
32 # Journey log
33     self.log_frame = tk.Frame(root, bg="#FFFFFF", relief=tk.RAISED, borderwidth=2, highlightbackground="#D1D5DB", highlightthickness=2)
34     self.log_frame.pack(pady=5, padx=20, fill=tk.X)
35     tk.Label(self.log_frame, text="Riyata Perjalanan:", font=("Helvetica", 12, "bold"), bg="#FFFFFF", fg="#1F2937").pack(anchor="w", padx=10, pady=2)
36     self.journey_log = tk.Text(self.log_frame, height=5, width=80, font=("Helvetica", 10), bg="#F9FAFB", fg="#1F2937", relief=tk.FLAT, borderwidth=2)
37     self.journey_log.pack(padx=10, pady=5, fill=tk.X)
38     self.journey_log.insert(tk.END, "Gudang")
39     self.journey_log.config(state=tk.DISABLED)
40
41     self.tooltip = None
42     self.load_images()
43     self.draw_map()
44
45     x, y = node_pos["Gudang"]
46     self.kurir = self.canvas.create_image(x, y, image=self.courier_img, tags="kurir")
47     self.path_lines = []
48     self.path_gloves = []
49     self.journey_history = ["Gudang"]
50
51 def load_images(self):
52     try:
53         self.courier_img = self.create_svg_motorcycle_rider(40, 40)
54         self.house_img = self.create_svg_house_with_items(50, 50)
55     except Exception as e:
56         print(f"Error loading images: {e}")
57         self.courier_img = None
58         self.house_img = None
59
60 def create_svg_motorcycle_rider(self, width, height):
61     img = tk.PhotoImage(width=width, height=height)
62     for y in range(height//2, height-8):
63         for x in range(8, width-10):
64             if 0 <= x < width and 0 <= y < height:
65                 img.put("#1E40AF", (x, y))
66     for y in range(5, height//2):
67         for x in range(width//3, 2*width//3):
68             if 0 <= x < width and 0 <= y < height:
69                 img.put("#F97316", (x, y))
70     for y in range(2, 12):
71         for x in range(width//2-6, width//2+6):
72             if 0 <= x < width and 0 <= y < height:
73                 img.put("#FBFF24", (x, y))
74     for y in range(height//3, height//2):
75         for x in range(width-12, width-2):
76             if 0 <= x < width and 0 <= y < height:
77                 img.put("#7C2D12", (x, y))
78     wheel_centers = [(12, height-8), (width-12, height-8)]
79     for cx, cy in wheel_centers:
80         for dx in range(-5, 6):
81             for dy in range(-5, 6):
82                 if dx*dx + dy*dy <= 25 and 0 <= cx + dx < width and 0 <= cy + dy < height:
83                     img.put("#111827", (cx + dx, cy + dy))
84
85     return img

```

```

1  def create_svg_house_with_items(self, width, height):
2      img = tk.PhotoImage(width=width, height=height)
3      wall_color = "#FC0D34D"
4      roof_color = "#DC2626"
5      door_color = "#1E3A8A"
6      window_frame = "#FFFFFF"
7      window_pane = "#60A5FA"
8      package_color = "#92400E"
9      grass_color = "#34C759"
10
11     for y in range(4*height//5, height):
12         for x in range(width):
13             if 0 <= x < width and 0 <= y < height:
14                 img.put(grass_color, (x, y))
15     wall_top, wall_bottom = height//3, 4*height//5
16     wall_left, wall_right = width//6, 5*width//6
17     for y in range(wall_top, wall_bottom):
18         for x in range(wall_left, wall_right):
19             if 0 <= x < width and 0 <= y < height:
20                 img.put(wall_color, (x, y))
21     roof_peak = height//8
22     for y in range(roof_peak, wall_top):
23         roof_width = (y - roof_peak) * 2
24         for x in range(width//2 - roof_width, width//2 + roof_width):
25             if wall_left <= x < wall_right and 0 <= y < width and 0 <= y < height:
26                 img.put(roof_color, (x, y))
27     door_width, door_height = width//6, height//4
28     door_x, door_y = width//2 - door_width//2, wall_bottom - door_height
29     for y in range(door_y, wall_bottom):
30         for x in range(door_x, door_x + door_width):
31             if 0 <= x < width and 0 <= y < height:
32                 img.put(door_color, (x, y))
33     window_width, window_height = width//8, height//8
34     window_positions = [(wall_left + 10, wall_top + 10), (wall_right - window_width - 10, wall_top + 10)]
35     for wx, wy in window_positions:
36         for y in range(wy, wy + window_height):
37             for x in range(wx, wx + window_width):
38                 if 0 <= x < width and 0 <= y < height:
39                     img.put(window_pane, (x, y))
40         for y in range(wy-2, wy + window_height + 2):
41             for x in range(wx-2, wx + window_width + 2):
42                 if (y in (wy-2, wy + window_height + 1) or x in (wx-2, wx + window_width + 1)) and 0 <= x < width and 0 <= y < height:
43                     img.put(window_frame, (x, y))
44     package_width, package_height = width//8, height//10
45     for y in range(wall_bottom - package_height, wall_bottom):
46         for x in range(wall_right + 5, wall_right + 5 + package_width):
47             if 0 <= x < width and 0 <= y < height:
48                 img.put(package_color, (x, y))
49     return img
50
51 def draw_map(self):
52     drawn_paths = set()
53     for node, neighbors in graph.items():
54         x1, y1 = node_pos[node]
55         for neighbor in neighbors:
56             if (node, neighbor) not in drawn_paths and (neighbor, node) not in drawn_paths:
57                 x2, y2 = node_pos[neighbor]
58                 key = (node, neighbor)
59                 reverse_key = (neighbor, node)
60                 if key in path_adjustments:
61                     (off_x1, off_y1), (off_x2, off_y2), color = path_adjustments[key]
62                     self.canvas.create_line(x1 + off_x1, y1 + off_y1, x2 + off_x2, y2 + off_y2,
63                                            fill=color, width=10, smooth=True, capstyle=tk.ROUND)
64                     self.canvas.create_line(x1 + off_x1, y1 + off_y1, x2 + off_x2, y2 + off_y2,
65                                            fill="#FC0D34D", width=2, dash=(5, 5))
66                 elif reverse_key in path_adjustments:
67                     (off_x2, off_y2), (off_x1, off_y1), color = path_adjustments[reverse_key]
68                     self.canvas.create_line(x1 + off_x1, y1 + off_y1, x2 + off_x2, y2 + off_y2,
69                                            fill=color, width=10, smooth=True, capstyle=tk.ROUND)
70                     self.canvas.create_line(x1 + off_x1, y1 + off_y1, x2 + off_x2, y2 + off_y2,
71                                            fill="#FC0D34D", width=2, dash=(5, 5))
72                 else:
73                     self.canvas.create_line(x1, y1, x2, y2, fill="#4B5EAA", width=10, capstyle=tk.ROUND)
74                     self.canvas.create_line(x1, y1, x2, y2, fill="#FC0D34D", width=2, dash=(5, 5))
75                 drawn_paths.add((node, neighbor))
76                 drawn_paths.add((neighbor, node))
77     for node, pos in node_pos.items():
78         x, y = pos
79         if node == "Gudang":
80             if self.house_img:
81                 self.canvas.create_image(x, y, image=self.house_img)
82             else:
83                 self.canvas.create_oval(x-12, y-12, x+12, y+12, fill="#60A5FA", outline="#1E3A8A", width=2)
84                 self.canvas.create_oval(x-8, y-8, x+8, y+8, fill="#92400E", outline="")
85                 self.canvas.create_text(x, y+30, text=node, font=("Helvetica", 10, "bold"), fill="#1F2937")
86

```



```
1 def draw_combined_paths(self, bfs_route=None, dfs_route=None):
2     for line in self.path_lines:
3         self.canvas.delete(line)
4     for glow in self.path_glow:
5         self.canvas.delete(glow)
6     self.path_lines = []
7     self.path_glow = []
8
9     if bfs_route:
10        bfs_coords = []
11        for node in bfs_route:
12            x, y = node_pos[node]
13            bfs_coords.extend([x, y])
14        self.path_glow.append(self.canvas.create_line(bfs_coords, fill="#FCD34D", width=8, dash=(4, 4), capstyle=tk.ROUND))
15        self.path_lines.append(self.canvas.create_line(bfs_coords, fill="#10B981", width=5, capstyle=tk.ROUND, joinstyle=tk.ROUND))
16
17     if dfs_route:
18        dfs_coords = []
19        for node in dfs_route:
20            x, y = node_pos[node]
21            dfs_coords.extend([x, y])
22        self.path_glow.append(self.canvas.create_line(dfs_coords, fill="#FCD34D", width=8, dash=(4, 4), capstyle=tk.ROUND))
23        self.path_lines.append(self.canvas.create_line(dfs_coords, fill="#EF4444", width=5, capstyle=tk.ROUND, joinstyle=tk.ROUND))
24
25 def animate_courier(self, route, method_name, callback=None):
26     route_coords = []
27     for node in route:
28         x, y = node_pos[node]
29         route_coords.extend([x, y])
30     route_text = f'{method_name}: " + ".join(route)
31     self.status_label.config(text=f'Rute {route_text}')
32     self.disable_buttons()
33     for i in range(len(route) - 1):
34         start_x, start_y = node_pos[route[i]]
35         end_x, end_y = node_pos[route[i + 1]]
36         steps = 40
37         self.status_label.config(text=f'Bergerak ({method_name}) dari {route[i]} ke {route[i+1]}'")
38         for j in range(steps):
39             new_x = start_x + (end_x - start_x) * j / steps
40             new_y = start_y + (end_y - start_y) * j / steps
41             self.canvas.coords(self.kurir, new_x, new_y)
42             glow = self.canvas.create_oval(new_x-10, new_y-10, new_x+10, new_y+10, fill="#F97316", outline="", tags="glow")
43             self.root.update()
44             time.sleep(0.02)
45             self.canvas.delete(glow)
46     self.current_location = route[-1]
47     self.location_label.config(text=f'Lokasi Saat Ini: {self.current_location}')
48     for node in route[1:]:
49         if node not in self.journey_history or self.journey_history[-1] != node:
50             self.journey_history.append(node)
51     self.status_label.config(text=f'Kurir sampai di {route[-1]} ({method_name})!')
52     self.enable_buttons()
53     # Ubah label dropdown menjadi "Tujuan Berikutnya" setelah animasi selesai
54     if self.current_location != "Gudang":
55         self.destination_label.config(text="TUJUAN BERIKUTNYA:")
56     if callback:
57         self.root.after(500, callback)
58
```

```
1  def update_journey_log(self, bfs_route=None, dfs_route=None):
2      self.journey_log.config(state=tk.NORMAL)
3      self.journey_log.delete(1.0, tk.END)
4      if bfs_route and dfs_route:
5          bfs_text = "BFS: " + " → ".join(bfs_route) + "\n"
6          dfs_text = "DFS: " + " → ".join(dfs_route) + "\n"
7          self.journey_log.insert(tk.END, bfs_text + dfs_text + "\nMengikuti jalur BFS")
8      else:
9          log_text = " → ".join(self.journey_history)
10         self.journey_log.insert(tk.END, log_text)
11     self.journey_log.config(state=tk.DISABLED)
12
13 def reset_courier(self):
14     x, y = node_pos["Gudang"]
15     self.canvas.coords(self.kurir, x, y)
16     for line in self.path_lines:
17         self.canvas.delete(line)
18     for glow in self.path_glows:
19         self.canvas.delete(glow)
20     self.path_lines = []
21     self.path_glows = []
22     self.current_location = "Gudang"
23     self.is_delivery_started = False
24     self.location_label.config(text=f"Lokasi Saat Ini: {self.current_location}")
25     self.journey_history = ["Gudang"]
26     self.update_journey_log()
27     self.dropdown.set("Pilih Tujuan")
28     self.destination_label.config(text="TUJUAN AWAL:") # Kembalikan label ke "Tujuan Awal"
29     self.enable_buttons()
30     self.status_label.config(text="Status: Kurir siap di Gudang")
31
32 def get_search_method(self):
33     method = self.method_var.get()
34     return bfs if method == "BFS" else dfs
35
36 def start_delivery(self):
37     destination = self.destination_var.get()
38     if destination == "Pilih Tujuan" or destination == self.current_location:
39         self.status_label.config(text="Pilih tujuan yang berbeda dari lokasi saat ini!")
40         return
41
42     mode = self.mode_var.get()
43     if mode == "Gabungan":
44         bfs_route = bfs("Gudang", destination)
45         dfs_route = dfs("Gudang", destination)
46         if not bfs_route or not dfs_route:
47             self.status_label.config(text=f"Tidak ada rute dari Gudang ke {destination}!")
48             return
49         self.draw_combined_paths(bfs_route, dfs_route)
50         self.update_journey_log(bfs_route, dfs_route)
51         self.animate_courier(bfs_route, "BFS")
52         self.is_delivery_started = True
53     else:
54         search_method = self.get_search_method()
55         route = search_method("Gudang", destination)
56         if route:
57             self.draw_combined_paths(route if self.method_var.get() == "BFS" else None,
58                                     route if self.method_var.get() == "DFS" else None)
59             self.animate_courier(route, self.method_var.get())
60             self.is_delivery_started = True
61         else:
62             self.status_label.config(text=f"Tidak ada rute dari Gudang ke {destination}!")
```

```
1  def continue_to_new_destination(self):
2      destination = self.destination_var.get() # Gunakan dropdown yang sama
3      if destination == "Pilih Tujuan" or destination == self.current_location:
4          self.status_label.config(text="Pilih tujuan yang berbeda dari lokasi saat ini!")
5          return
6
7      mode = self.mode_var.get()
8      if mode == "Gabungan":
9          bfs_route = bfs(self.current_location, destination)
10         dfs_route = dfs(self.current_location, destination)
11         if not bfs_route or not dfs_route:
12             self.status_label.config(text="Tidak ada rute dari {self.current_location} ke {destination}!")
13             return
14         self.draw_combined_paths(bfs_route, dfs_route)
15         self.update_journey_log(bfs_route, dfs_route)
16         self.animate_courier(bfs_route, "BFS")
17     else:
18         search_method = self.get_search_method()
19         route = search_method(self.current_location, destination)
20         if route:
21             self.draw_combined_paths(route if self.method_var.get() == "BFS" else None,
22                                     route if self.method_var.get() == "DFS" else None)
23             self.animate_courier(route, self.method_var.get())
24         else:
25             self.status_label.config(text="Tidak ada rute dari {self.current_location} ke {destination}!")
26
27 def return_home(self):
28     if self.current_location != "Gudang":
29         mode = self.mode_var.get()
30         if mode == "Gabungan":
31             bfs_route = bfs(self.current_location, "Gudang")
32             dfs_route = dfs(self.current_location, "Gudang")
33             if not bfs_route or not dfs_route:
34                 self.status_label.config(text="Tidak ada rute kembali ke Gudang!")
35                 return
36             self.draw_combined_paths(bfs_route, dfs_route)
37             self.update_journey_log(bfs_route, dfs_route)
38             self.animate_courier(bfs_route, "BFS", self.delivery_completed)
39         else:
40             search_method = self.get_search_method()
41             route = search_method(self.current_location, "Gudang")
42             if route:
43                 self.draw_combined_paths(route if self.method_var.get() == "BFS" else None,
44                                         route if self.method_var.get() == "DFS" else None)
45                 self.animate_courier(route, self.method_var.get(), self.delivery_completed)
46             else:
47                 self.status_label.config(text="Tidak ada rute kembali ke Gudang!")
48         else:
49             self.status_label.config(text="Kurir sudah di Gudang!")
50
51 def delivery_completed(self):
52     self.status_label.config(text="Pengiriman selesai! Kurir kembali ke Gudang.")
53     self.destination_label.config(text="TUJUAN AWAL:") # Kembalikan label ke "Tujuan Awal"
54     self.is_delivery_started = False
55     self.enable_buttons()
56
57 def disable_buttons(self):
58     self.start_button.config(state=tk.DISABLED)
59     self.continue_button.config(state=tk.DISABLED)
60     self.home_button.config(state=tk.DISABLED)
61     self.reset_button.config(state=tk.DISABLED)
62
63 def enable_buttons(self):
64     self.reset_button.config(state=tk.NORMAL)
65     if self.current_location == "Gudang":
66         self.start_button.config(state=tk.NORMAL)
67         self.continue_button.config(state=tk.DISABLED)
68         self.home_button.config(state=tk.DISABLED)
69     else:
70         self.start_button.config(state=tk.DISABLED)
71         self.continue_button.config(state=tk.NORMAL)
72         self.home_button.config(state=tk.NORMAL)
73
74 root = tk.Tk()
75 app = CourierApp(root)
76 root.mainloop()
```

Penjelasan:

Aplikasi ini adalah simulasi kurir kota menggunakan GUI Tkinter di Python. Aplikasi memvisualisasikan pengiriman kurir dari gudang ke berbagai lokasi di kota menggunakan dua algoritma pencarian jalur, yaitu BFS (Breadth-First Search) dan DFS (Depth-First Search).

Struktur Data Rute dan Lokasi

a. Nodes dan Koordinat

```
node_pos = {
```

Ini adalah dictionary yang menyimpan koordinat (x, y) untuk setiap node atau lokasi pada peta.

b. Struktur Graf

```
graph = {
```

Dictionary graph mendefinisikan koneksi antar lokasi (jalan) di kota. Kunci adalah nama lokasi dan nilai adalah daftar lokasi yang terhubung langsung.

Rute dan Koneksi Jalan

Dari struktur graf di atas, berikut adalah koneksi lengkap pada peta:

1. Gudang terhubung ke:
 - Jl. Raya Darmo
 - Jl. Mayjen Sungkono
 - Jl. Arjuno
2. Jl. Raya Darmo terhubung ke:
 - Gudang
 - Jl. Tunjungan
 - Jl. Pemuda
 - Jl. Kertajaya
3. Jl. Mayjen Sungkono terhubung ke:
 - Gudang
 - Jl. Diponegoro
 - Jl. Basuki Rahmat
 - Jl. Walikota Mustajab
4. Jl. Tunjungan terhubung ke:
 - Jl. Raya Darmo
 - Jl. Basuki Rahmat
 - Jl. Embong Malang
 - Jl. Ngagel

5. Jl. Pemuda terhubung ke:
 - Jl. Raya Darmo
 - Jl. Basuki Rahmat
 - Jl. Walikota Mustajab
6. Jl. Diponegoro terhubung ke:
 - Jl. Mayjen Sungkono
 - Jl. Basuki Rahmat
 - Jl. Dr. Soetomo
7. Jl. Basuki Rahmat terhubung ke:
 - Jl. Tunjungan
 - Jl. Pemuda
 - Jl. Diponegoro
 - Jl. Panglima Sudirman
 - Jl. Dr. Soetomo
8. Jl. Embong Malang terhubung ke:
 - Jl. Tunjungan
 - Jl. Panglima Sudirman
 - Jl. Ngagel
9. Jl. Panglima Sudirman terhubung ke:
 - Jl. Basuki Rahmat
 - Jl. Embong Malang
 - Jl. Gubernur Suryo
 - Jl. Simpang Dukuh
10. Jl. Gubernur Suryo terhubung ke:
 - Jl. Panglima Sudirman
 - Jl. Dr. Soetomo
 - Jl. Dipatiukur
 - Jl. Simpang Dukuh
11. Jl. Kertajaya terhubung ke:
 - Jl. Raya Darmo
 - Jl. Ngagel
 - Jl. Kusuma Bangsa
12. Jl. Ngagel terhubung ke:
 - Jl. Kertajaya
 - Jl. Tunjungan
 - Jl. Kusuma Bangsa
 - Jl. Embong Malang
13. Jl. Dr. Soetomo terhubung ke:
 - Jl. Gubernur Suryo
 - Jl. Dipatiukur
 - Jl. Diponegoro
 - Jl. Basuki Rahmat
14. Jl. Walikota Mustajab terhubung ke:
 - Jl. Pemuda

- Jl. Mayjen Sungkono
15. Jl. Kusuma Bangsa terhubung ke:
- Jl. Kertajaya
 - Jl. Ngagel
16. Jl. Arjuno terhubung ke:
- Gudang
 - Jl. Dipatiukur
17. Jl. Dipatiukur terhubung ke:
- Jl. Arjuno
 - Jl. Gubernur Suryo
 - Jl. Dr. Soetomo
18. Jl. Simpang Dukuh terhubung ke:
- Jl. Panglima Sudirman
 - Jl. Gubernur Suryo

Penyesuaian Jalur Visual

```
path_adjustments = {
```

Dictionary ini digunakan untuk menyesuaikan posisi garis rute pada tampilan visual agar jalanan yang berdekatan bisa terlihat lebih jelas.

Algoritma Pencarian Rute

a. BFS (Breadth-First Search)

```
def bfs(start, goal):
```

BFS menggunakan queue untuk menelusuri graf level per level. Ini akan menemukan jalur terpendek dalam hal jumlah node.

b. DFS (Depth-First Search)

```
def dfs(start, goal):
```

DFS menggunakan stack untuk menelusuri graf lebih dalam sebelum kembali. DFS tidak menjamin jalur terpendek.

Fitur Utama Aplikasi:

1. Interface Pengguna

Aplikasi memiliki elemen-elemen GUI berikut:

- Canvas untuk menampilkan peta
- Dropdown untuk memilih mode (Terpisah/Gabungan)

- Dropdown untuk memilih metode pencarian (BFS/DFS)
- Dropdown untuk memilih tujuan
- Tombol untuk memulai, melanjutkan, kembali ke gudang, dan reset
- Panel status yang menampilkan lokasi kurir dan status pengiriman
- Area log untuk riwayat perjalanan

2. Mode Visualisasi

- Mode Terpisah: Hanya menampilkan satu jalur berdasarkan metode yang dipilih (BFS/DFS)
- Mode Gabungan: Menampilkan kedua jalur BFS dan DFS secara bersamaan

3. Animasi Pergerakan Kurir

```
def animate_courier(self, route, method_name, callback=None):
```

Metode ini menganimasikan pergerakan kurir melalui rute yang ditentukan dengan efek glowing.

Penjelasan Fungsi-Fungsi Kunci:

1. __init__

Inisialisasi aplikasi, membuat interface, dan menyiapkan variabel state.

2. load_images dan fungsi create_svg

Membuat gambar representasi kurir dan gudang menggunakan teknik gambar sederhana.

3. draw_map

Menggambar peta kota dengan jalan dan simpul.

4. draw_combined_paths

Menggambar jalur BFS dan DFS pada peta.

5. start_delivery

Memulai pengiriman dari gudang ke tujuan yang dipilih.

6. continue_to_new_destination

Melanjutkan perjalanan dari lokasi saat ini ke tujuan berikutnya.

7. return_home

Membawa kurir kembali ke gudang dari lokasi saat ini.

8. reset_courier

Mereset semua status dan mengembalikan kurir ke gudang.

Flow Aplikasi:

1. User memilih tujuan awal dari dropdown
2. User memilih mode visualisasi (Terpisah/Gabungan)
3. Jika mode Terpisah dipilih, user juga memilih metode pencarian (BFS/DFS)
4. User menekan "MULAI PENGIRIMAN" untuk memulai animasi
5. Setelah sampai di tujuan, user bisa:
 - Memilih tujuan berikutnya dan menekan "LANJUTKAN PERJALANAN"
 - Menekan "KEMBALI KE GUDANG" untuk kembali
 - Menekan "RESET SEMUA" untuk reset aplikasi

Perbedaan BFS dan DFS dalam Aplikasi:

- BFS cenderung menemukan jalur yang lebih pendek dalam jumlah node
- DFS cenderung mengeksplorasi jalur yang lebih panjang sebelum menemukan tujuan
- Dalam mode Gabungan, rute BFS ditampilkan dengan warna hijau (#10B981) sedangkan DFS dengan warna merah (#EF4444)

Komponen Visual dan Estetika:

- Jalan utama menggunakan warna biru (#4B5EAA) dengan garis kuning dashed (#FCD34D)
- Node lokasi menggunakan lingkaran biru dengan teks
- Gudang direpresentasikan dengan ikon rumah
- Kurir direpresentasikan dengan ikon pengendara motor

Kesimpulan

Aplikasi ini merupakan visualisasi interaktif yang menggabungkan konsep teori graf, algoritma pencarian jalur (BFS/DFS), dan pemrograman GUI untuk mensimulasikan pengiriman barang di kota. Aplikasi membantu memahami perbedaan antara algoritma BFS dan DFS dalam menemukan rute dari satu titik ke titik lainnya dalam sebuah jaringan jalan.