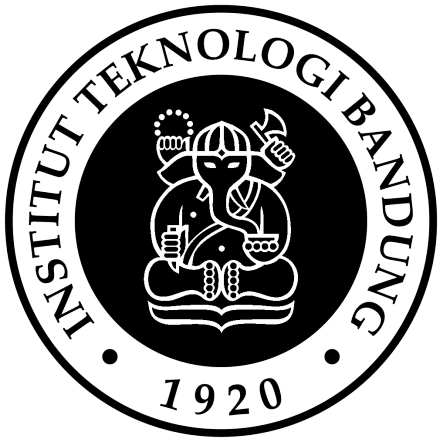
LAPORAN
KU1102 PENGENALAN KOMPUTASI
SEMESTER I 2023-2024

Perancangan Sistem Rute Terdekat dan Rute Tercepat Dengan Algoritma Dijkstra yang Dioptimasi *Priority Queue*

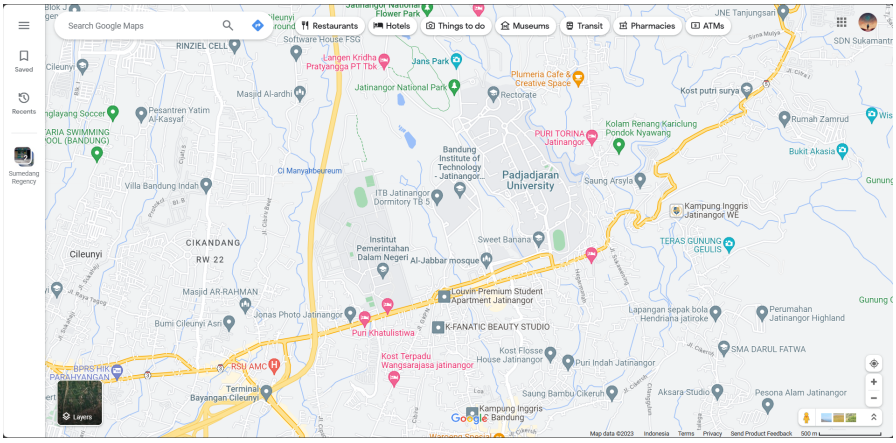


Dibimbing oleh:
Dr. Fadhil Hidayat, S.Kom., M.T.
Disusun oleh:
Yayat Nurhidayat (16523080)
Nafhan Hadiyan Shafwatudin (16523213)
Zaka Hanif Nabalah (19623010)
Atharizza Muhammad Athaya (19623143)
Ferro Arka Berlian (19623213)

PROGRAM TAHAP PERSIAPAN BERSAMA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

Daftar Isi

1 Pendahuluan

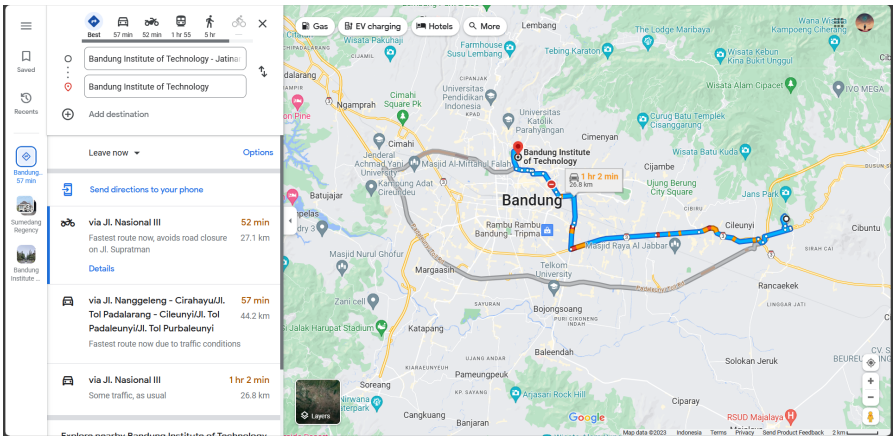


Gambar 1.1: Tangkapan layar dari peta elektronik G-Maps

Peta elektronik adalah bentuk terdigitalisasi dari peta konvensional. Terdapat banyak sekali aplikasi dari *e-maps* seperti Gmaps, Waze, Apple Maps, dan sebagainya. Dewasa ini, karena kemudahan dan fiturnya, *e-maps* atau peta elektronik sudah menjadi bagian dari sebagian besar masyarakat.

e-maps memiliki keunggulan dibandingkan dengan peta konvensional, seperti kemampuan untuk menampilkan informasi yang lengkap, akurat, dan terkini.

1.1 Latar Belakang Masalah



Gambar 1.2: Tangkapan layar rute tercepat dari ITB Kampus Jatinangor ke ITB Kampus Ganesha pada G-Maps

Fitur rute tercepat dan terpendek pada *e-maps* seringkali dipakai oleh masyarakat. Fitur ini sangat berguna karena bisa menyajikan rute yang bisa pengguna pilih dan manfaatkan untuk bepergian dengan cepat atau efisien. Pengguna dapat mengetahui rute terpendek dan aproksimasi waktu tempuh juga dapat mengetahui rute tercepat dengan aproksimasi waktu tempuhnya apabila terdapat kemacetan pada rute tertentu. Oleh karena itu rute tercepat dan terpendek ini penting untuk dieksplorasi.

Salah satu algoritma yang dapat digunakan dalam perancangan fitur rute tercepat dan terpendek adalah algoritma Dijkstra. Algoritma ini dapat digunakan karena merupakan salah satu algoritma shortest path yang efisien. Selain itu, algoritma Dijkstra bisa dioptimasi lagi dengan menggunakan priority queue.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, laporan ini membahas beberapa hal, yaitu

- Apa itu rute terpendek dan rute tercepat pada *e-maps*?
- Bagaimana cara kerja *e-maps* secara umum?
- Bagaimana implementasi algoritma Dijkstra dalam pencarian rute terdekat atau tercepat pada *e-maps*?
- Bagaimana struktur data *priority queue* dapat mengoptimasi algoritma dijkstra pada pencarian rute?
- Apa kelemahan dan kelebihan sistem pencarian rute terdekat atau tercepat dengan algoritma Dijkstra yang dioptimasi *priority queue*

1.3 Tujuan dan Manfaat

Berdasarkan latar belakang dan rumusan masalah di atas, tujuan perancangan ini adalah sebagai berikut.

- Mengetahui definisi rute terpendek dan rute tercepat pada *e-maps*
- Mengetahui cara kerja *e-maps*
- Dapat mengimplementasikan algoritma Dijkstra dalam pencarian rute terdekat atau tercepat
- Mengetahui pengaruh struktur data *priority queue* dalam pengoptimasian algoritma dijkstra

1.4 Ruang Lingkup

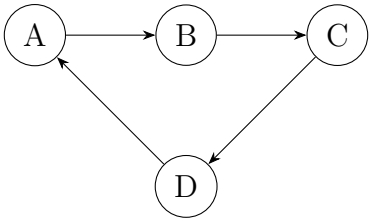
Pada *e-maps* mainstream, data didapat dari kombinasi citra satelit, survey, dan pengguna. Karena keterbatasan teknologi dan data, maka sistem yang kami buat:

- Hanya menggunakan data jalan yang diinputkan secara manual
- Hanya menggunakan data kemacetan yang digenerasi secara random
- Sistem berjalan secara *offline* dan tidak real time

2 Tinjauan Pustaka

Berikut adalah beberapa teori yang kami gunakan dalam program ini. Teori-teori ini kami dapatkan dari beberapa referensi. untuk pustaka yang kami gunakan, kami tuliskan di bagian Referensi

2.1 Teori Graf



Gambar 2.1: Contoh graf berarah sederhana

Teori graf merupakan salah satu bahasan dalam matematika dan ilmu komputer yang mempelajari graf. Graf adalah himpunan pasangan dari simpul (*node*) dan sisi (*edge*). Simpul adalah suatu objek dan sisi adalah sesuatu yang menghubungkan suatu objek dengan objek lainnya. Simpul dan sisi di sini bisa saja mewakilkan simpangan dan jalan, orang dan hubungan sosial, unsur dan ikatan. Terdapat banyak aplikasi teori graf seperti peta jalan, jaringan sosial, jaringan komputer, dan lain-lain.

Terdapat banyak bahasan dalam teori graf, akan tetapi dalam laporan ini kita hanya akan membahas bahasan yang esensial dalam algoritma *shortest path*.

Jenis-Jenis Graf

Berdasarkan orientasi arah pada sisi, secara umum graf dapat dibedakan menjadi dua jenis:

1. **Graf tak berarah (*undirected graph*)**

   sisi (*edge*) pada graf tak berarah tidak memiliki arah sehingga graf bisa ditelusuri ke kedua arah

2. **Graf berarah (*directed graph*)**

   sisi (*edge*) pada graf berarah memiliki arah sehingga graf hanya bisa ditelusuri ke arah yang sudah ditentukan

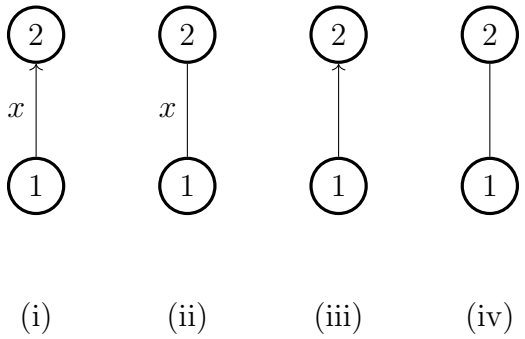
Berdasarkan eksistensi bobot pada sisi, secara umum graf dapat dibedakan menjadi dua jenis:

1. **Graf tak berbobot (*unweighted graph*)**

   Graf tak berbobot adalah graf yang tidak memiliki nilai pada setiap sisinya. Adapun sisi pada graf tak berbobot hanya menunjukkan hubungan antar simpul.

2. **Graf berbobot (*weighted graph*)**

   graf berbobot adalah graf yang memiliki nilai pada setiap sisi. Nilai pada setiap sisi tersebut bisa saja merepresentasikan jarak, waktu, biaya, atau hal lain yang dimodelkan dengan graf.



Gambar 2.2: Jenis graf berdasarkan *edge* dan bobotnya.

Berdasarkan Gambar 2.2, (i) adalah visualisasi dari graf berarah dengan bobot x, (ii) graf tak berarah dengan bobot x, (iii) graf berarah tak berbobot, (iv) graf tak berarah dan tak berbobot.

Representasi Graf

Hubungan antara simpul (*node*) dan sisi (*edge*) pada graf bisa direpresentasikan dalam bahasa pemrograman dengan beberapa jenis, seperti

1. ***Edge list***

   Edge list adalah sekumpulan pasangan dari simpul yang dihubungkan oleh sisi. Edge list efisien secara memori namun tidak efisien untuk mencari tetangga dari sebuah simpul

2. ***Adjacency matrix***

   Adjacency matrix adalah matrix dua dimensi yang berisikan hubungan dari simpul-simpul. Apabila graf berbobot, nilai dari matrix-nya adalah bobot dari simpul yang dihubungkan oleh sisi, sedangkan graf tak berbobot, nilai dari matrix hanya menunjukkan keberadaan sisi yang menghubungkan simpul.

3. ***adjacency list***

   Adjacency list merupakan daftar tetangga dari setiap simpul dalam sebuah graf. Adjacency list bisa diimplementasikan dengan menggunakan *dynamic array* yaitu array yang ukurannya dapat berubah secara dinamis.

2.2 *Shortest Path*

Shortest path adalah salah satu topik bahasan dalam teori graf. Seperti namanya shortest path berarti mencari path/jalan/rute terdekat dari suatu node ke node lain yang terdapat dalam graph

Algoritma shortest path

Berikut disajikan tiga dari contoh dari sekian banyaknya algoritma shortest path.

- Algoritma Floyd-Warshall: dapat menemukan jalur terpendek antara semua pasang simpul dalam graf berbobot, baik positif maupun negatif, selama tidak ada siklus negatif. Algoritma ini menggunakan paradigma pemrograman dynamic programming untuk menghitung jarak dengan basis kasus dan aturan rekurens. Kompleksitas waktu: $O(V^3)$.

- Bellman Ford algorithm: dapat menghitung rute tercepat pada graf dengan bobot negatif, selama tidak ada siklus negatif. Algoritma ini melakukan relaksasi pada setiap sisi secara berulang. Kompleksitas waktu: O(VE) .

- Dijkstra Algorithm: menggunakan paradigma pemrograman *greedy* untuk mencari jalur terpendek dari satu simpul ke semua simpul lainnya dalam graf berbobot positif. Algoritma ini memilih simpul dengan jarak terkecil dan melakukan relaksasi pada tetangganya. Kompleksitas waktu: O(E log V) dengan emphpriority queue.

- A* Algorithm: menggunakan fungsi heuristik untuk mempercepat pencarian jalur terpendek dari satu simpul ke satu simpul tujuan dalam graf positif. Algoritma ini mengestimasi jarak dengan fungsi heuristik yang harus admissible dan consistent. Kompleksitas waktu: tergantung pada heuristik. Namun, besar kemungkinan lebih efisien dari algoritma Dijkstra.

2.3 *Priority Queue*

Struktur data *priority queue* adalah struktur data abstrak yang menyimpan elemen berdasarkan prioritasnya. Setiap elemen memeiliki nilai prioritas yang menentukan urutas pengaksesan atau penghapusan elemen.Jika suatu elemen di dalam struktur data *priority queue* memiliki nilai prioritas paling tinggi, maka elemen tersebutlah yang bisa diakses atau dihapus terlebih dahulu.

Priority queue dapat diimplementasikan dengan berbagai cara seperti dengan array, heap, linked list, atau tree. Masing-masing cara memiliki kelebihan dan kekurangan tersendiri.

2.4 Algoritma Dijkstra

Algoritma Dijkstra merupakan salah satu algoritma shortest path dengan paradigma pemrograman *greedy*. Ini merupakan algoritma yang efisien untuk menentukan rute terpendek dari salah satu node ke semua node yang lain. Algoritma Dijkstra dapat dioptimasi oleh priority queue. Berikut langkah-langkah algoritma Dijkstra yang dioptimasi priority queue:

1. Deklarasikan sebuah array sebagai jarak simpul ke semua simpul yang lain dengan nilai tak hingga kecuali pada simpul sumber yang bernilai nol. Deklarasikan

2. Deklarasikan *priority queue* q kosong yang nantinya akan diisi simpul dalam graf dengan prioritas jarak ke simpul sumber. Inisialisasi jarak semua simpul menjadi tak hingga kecuali simpul sumber yang diinisialisasi menjadi nol.

3. simpan simpul sumber dan bobot 0 ke dalam *priority queue* q

4. Selama q tidak kosong, lakukan hal berikut:

    4.1 ambil simpul u dengan nilai terkecil dari q dan hapus dari q.

    4.2 Untuk setiap simpul v yang bertetangga dengan u, lakukan proses relax yakni

        4.2.1 Hitung jarak baru dari sumber ke v melalui u, yaitu jarak[u] + bobot(u, v).

4.2.2 Jika jarak baru lebih kecil dari jarak[v], maka perbarui jarak[v] dengan jarak baru dan simpul pendahulu[v] dengan u. Masukkan atau perbarui v ke dalam q dengan prioritas jarak[v].

5. Setelah q kosong, kembalikan array jarak dan array simpul pendahulu.

3 Metodologi

Untuk membuat program ini, kami menggunakan beberapa metode yang telah kami pelajari pada mata kuliah Pengenalan Komputasi. Tujuan kami menggunakan metode-metode ini supaya pembuatan program ini menjadi sistematis. Adapun metode-metode yang kami gunakan adalah sebagain berikut

3.1 Dekomposisi Masalah

Agar perancangan sistem rute tercepat bisa ditangani lebih mudah, kami membagi persoalan-persoalannya menjadi persoalan yang lebih kecil sebagai berikut.

- Input data
  - input jumlah simpangan/kota.
  - input jumlah jalan yang menghubungkan simpangan.
  - input graph (dua simpul/simpangan/kota yang terhubung oleh jalan dan panjang jalan).
  - input posisi awal pengguna.
  - input destinasi perjalanan pengguna.
- Pencarian rute terdekat dengan algoritma Dijkstra.
- Jika kasus kemacetan dimungkinkan, kemacetan dipilih secara random di rute tertentu.
- Pencarian rute tercepat apabila terjadi kasus kemacetan.
- Penyajian data hasil algoritma Dijkstra
  - Menampilkan rute terpendek dari posisi awal ke destinasi pengguna.
  - Menampilkan rute terpendek dari posisi awal ke semua simpangan atau kota yang lain.
  - Menampilkan rute tercepat dari posisi awal ke destinasi pengguna jika terjadi kemacetan.

- Menampilkan rute terpendek dari posisi awal ke semua simpangan atau kota yang lain.
- Menampilkan jarak dan waktu tempuh di semua rute tersebut.

3.2 Simulasi dan Desain Sistem

Dalam subbab ini, kami akan menjelaskan tentang desain dan juga cara kerja program yang kami buat seperti spesifikasi dan juga alur kerja nya

Spesifikasi sistem

Sistem yang kami buat adalah sistem penentu rute terpendek dan tercepat dengan memanfaatkan algoritma Dijkstra yang dioptimasi struktur data priority queue.

Sistem ini menerima input berupa data seperti jumlah simpangan atau kota, jumlah jalan yang menghubungkan simpangan, graf, posisi awal pengguna, dan destinasi perjalanan pengguna. Graf direpresentasikan dengan list tetangga (*adjacency list*).

Sistem mengirimkan keluaran berupa data rute dekat dengan jarak dan waktu tempuhnya beserta rute tercepat dengan jarak dan waktu tempuhnya jika terjadi kemacetan.

Alur kerja sistem

Dalam perancangan sistem, dibutuhkan simulasi atau *blue print* dari sistem yang akan dibuat. Oleh karena itu, berikut disajikan alur kerja sistem yang akan dibuat.

1. Sistem mendeklarasikan dan menerima masukan jumlah simpangan, jumlah jalan, nama kota/simpangan, posisi awal pengguna, destinasi perjalanan pengguna, kelajuan rata-rata pengguna, dan graf peta. Graf peta direpresentasikan dengan *adjacency list* sehingga bisa ditelusuri oleh algoritma Dijkstra
2. Program memberi opsi pada pengguna untuk melihat rute tercepat apabila kasus kemacetan terjadi
3. Program menelusuri rute terdekat dengan menggunakan algoritma dijkstra yang menyimpan data rute dan jarak.
4. Jika kemacetan terjadi:
    4.1 Program menelusuri rute tercepat dengan menggunakan algoritma Dijkstra yang menyimpan data rute dan waktu tempuh berdasarkan kecepatan pengguna ketika di keadaan normal dan kelajuan pengguna ketika melewati jalanan yang macet.

4.2 Memberikan keluaran berupa rute, jarak, dan waktu tempuh untuk rute terdekat destinasi pengguna.

4.3 Memberikan keluaran berupa rute, jarak dan waktu tempuh untuk rute terdekat ke kota atau simpangan lain.

4.4 Memberikan keluaran berupa rute, jarak, dan waktu tempuh untuk rute tercepat destinasi pengguna.

4.5 Memberikan keluaran berupa rute, jarak dan waktu tempuh untuk rute tercepat ke kota atau simpangan lain.

5. Jika kemacetan tidak terjadi:

5.1 Memberikan keluaran berupa rute, jarak, dan waktu tempuh untuk rute terdekat destinasi pengguna.

5.2 Memberikan keluaran berupa rute, jarak dan waktu tempuh untuk rute terdekat ke kota atau simpangan lain.

Gambar 3.1: Diagram alir sistem

3.3 Implementasi Sistem dengan Python

Dalam perancangan sistem yang dibuat, kami memilih bahasa python karena tuntutan tugas dan kemudahannya. Pembahasan di bawah membahas potongan kode yang kami anggap penting untuk diketahui dalam laporan ini.

Berikut library dan package yang kami gunakan dalam python:

```
from queue import PriorityQueue
from rich import print
from rich.console import Console
from rich.table import Table
from rich.align import Align
from rich.live import Live
import os
import random
```

Dalam proyek ini, kami menggunakan PriorityQueue sebagai struktur data priority queue, rich dan os untuk menata CLI, dan random untuk generasi kemacetan secara acak.

Proyek ini membutuhkan jumlah kota atau simpangan, jumlah jalan, posisi awal, destinasi, dan kelajuan rata-rata pengguna. Beri index untuk setiap kota atau simpangan dengan *dictionary*.

```
inf = 99999999
N = int(input("Masukkan jumlah kota atau simpangan: "))
E = int(input("Masukkan jumlah jalan: "))
print()
index = {}
print("Masukkan kota atau simpangan:")
for i in range (1, N+1):
    node = input()
    index[node] = i
    index[i] = node
print()
Start_input = input("Dimulai dari? ")
Start = index[Start_input]
End_input = input("Mau ke mana? ")
End = index[End_input]
V_avg = int(input("Masukkan kelajuan rata-rata (dalam km/jam): "))
```

Kami menggunakan list of list dalam python sebagai representasi graf adjacency list dan mencatat rute dengan list.

```
distance_adj_list = [[] for i in range(N + 1)]
time_adj_list = [[] for i in range(N + 1)]
dpred = [-1 for i in range(N + 1)]
tpred = [-1 for i in range(N + 1)]
q = PriorityQueue()
```

```
def add_distance_adj(a, b, w):
    distance_adj_list[index[a]].append([index[b], w])


def add_time_adj(a, b, s):
    t = float(float(s) / float(V_avg))
    time_adj_list[index[a]].append([index[b], t])

print(
    "Masukkan rute dan jarak (dalam km):"
)
for i in range(E):
    a, b, w = input().split()
    add_distance_adj(a, b, int(w))
    add_time_adj(a, b, int(w))
```

Jika dimungkinkan kasus kemacetan, kasus kemacetan kami generasi secara random sebagai asumsi karena keterbatasan teknologi yang kami sebutkan di ruang lingkup. Kasus kemacetan yang digenerasi akan mengubah graf dengan bobot waktu.

```
def add_traffic_jam(total_traffic_jam):
    added = [[False for j in range(N + 1)] for i in range(N + 1)]
    for i in range(total_traffic_jam):
        a = random.randint(1, N)
        if not distance_adj_list[a]:
            continue
        cont = random.choice(distance_adj_list[a])
        b = cont[0]
        b_index = -1
        dist = cont[1]
        if added[a][b]:
            continue
        counter = 0
        for j in distance_adj_list[a]:
            if b == j[0]:
                b_index = counter
            counter += 1
        added[a][b] = True
        v = random.randint(1, V_avg - 1)
        x = random.randint(1, dist)
        t = float((dist - x) / V_avg + x / v)
        time_adj_list[a][b_index][1] = t
```

Jika kemacetan terjadi, maka itu akan mengubah graf yang berbobotkan waktu tempuh. Oleh karena itu jika ada kemacetan, maka akan dilakukan dua kali algoritma dijkstra untuk graf berbobotkan jarak dan graf berbobotkan waktu tempuh

```
def distance_dijkstra(N, S):
```

```
    distance = [inf for x in range(N + 1)]
    visited = [False for x in range(N + 1)]
    distance[S] = 0
    q.put([0, S])
    while not q.empty():
        a = q.queue[0][1]
        q.get()
        if visited[a]:
            continue
        visited[a] = True
        for i in distance_adj_list[a]:
            b = i[0]
            w = i[1]
            if distance[a] + w < distance[b]:
                dpred[b] = a
                distance[b] = distance[a] + w
                q.put([distance[b], b])
    return distance


def time_dijkstra(N, S):
    time = [inf for x in range(N + 1)]
    visited = [False for x in range(N + 1)]
    time[S] = 0
    q.put([0, S])
    while not q.empty():
        a = q.queue[0][1]
        q.get()
        if visited[a]:
            continue
        visited[a] = True
        for i in time_adj_list[a]:
            b = i[0]
            w = i[1]
            if time[a] + w < time[b]:
                tpred[b] = a
                time[b] = time[a] + w
                q.put([time[b], b])
    return time
```

source code dilampirkan untuk mengetahui implementasi program dengan python lebih lanjut.

4 Kesimpulan dan *Lesson Learned*

Salah satu tujuan dari laporan tugas besar ini adalah untuk menunjukkan bagaimana algoritma Dijkstra yang dioptimasi dengan priority queue dapat digunakan untuk

merancang sistem rute terdekat dan rute tercepat. Dalam bab kesimpulan, kami akan menyajikan hasil-hasil yang kami peroleh dari pengujian dan analisis sistem kami, serta membahas kelebihan dan kekurangan dari pendekatan kami. Dalam bab lesson learned, kami akan merefleksikan proses pembelajaran kami selama mengerjakan tugas besar ini, serta memberikan saran-saran untuk pengembangan lebih lanjut dari sistem kami.

4.1 Kesimpulan

Rute terpendek adalah himpunan jalan yang menghubungkan dua simpangan dengan jumlah beban atau jarak jalan terkecil. Berbeda dengan rute terpendek, rute tercepat adalah himpunan jalan yang menghubungkan dua simpangan dengan jumlah beban atau jarak jalan terkecil. Apabila tidak ada kemacetan, rute tercepat akan sama dengan rute terpendek.

E-maps adalah salah satu aplikasi dari teori graf dengan simpangan jalan berlaku sebagai *node*/simpul dan jalan berlaku sebagai *edge*/sisi. Input data dari *e-maps* mungkin diambil dari citra satelit, survey, dan data dari pengguna. Data tersebut kemudian diproses sesuai dengan teori graf dan ditampilkan dalam bentuk peta interaktif yang bisa diakses melalui berbagai perangkat.

Dalam fitur rute terpendek atau rute tercepat pada *e-maps*, algoritma yang paling mungkin digunakan adalah algoritma Dijkstra, algoritma A*, dan algoritma Best-First search. Algoritma Dijkstra bisa diimplementasikan karena algoritma ini bisa mencari rute tercepat/terdekat dari satu node ke semua node yang lain dengan cepat dan efisien. Algoritma ini bekerja dengan memilih node dengan jarak terkecil dari node sumber dan mengupdate jarak node tetangganya.

Struktur data *priority queue* dapat digunakan untuk mengoptimasi algoritma Dijkstra sehingga algoritma akan menjadi lebih efisien lagi. *priority queue* bekerja untuk mempercepat pencarian simpul dengan jarak terdekat dari simpul sumber. *Priority queue* menyimpan simpul-simpul berdasarkan prioritasnya, yaitu jarak ke suatu simpul dari simpul sumber. Simpul dengan prioritas tertinggi, dalam hal ini jarak terkecil, akan dikeluarkan pertama dari *priority queue*.

Pencarian rute tercepat dari satu titik ke semua titik lainnya menyebabkan implementasi algoritma Dijkstra pada pencarian rute tercepat *e-maps* memiliki kelebihan sekaligus kekurangan. Algoritma bisa mencari rute tercepat dari satu titik ke semua titik lain. Namun, jelas akan lebih efisien jika pencarian rute dilakukan hanya untuk rute tujuan, hal ini dapat dilakukan dengan algoritma lain yaitu algoritma A*. Sementara itu, pengoptimasian algoritma Dijkstra dengan priority queue memang membuat algoritma Dijkstra efisien dalam hal waktu, tetapi *priority queue* membutuhkan lebih banyak memori untuk menyimpan elemen-elemennya.

4.2 *Lesson Learned*

Dalam perancangan sistem rute terdekat dan tercepat ini, kami mempelajari hal-hal penting, seperti

1. Teori graf, yaitu cabang ilmu matematika yang mempelajari tentang kumpulan objek yang disebut simpul (node) dan hubungan antara objek yang disebut sisi (edge). Teori graf dapat digunakan untuk merepresentasikan berbagai macam masalah nyata, seperti jaringan komputer, jaringan sosial, peta jalan, dan lain-lain.

2. Shortest path, yaitu masalah yang mencari jalur terpendek antara dua simpul dalam sebuah graf berbobot. Shortest path dapat digunakan untuk menentukan rute tercepat, biaya terendah, jarak terdekat, dan sebagainya.

3. Struktur data *priority queue*, yaitu struktur data abstrak yang menyimpan elemen berdasarkan prioritasnya. Struktur data priority queue dapat digunakan untuk mengoptimalkan algoritma shortest path, karena dapat memilih simpul dengan jarak terkecil dari sumber dengan cepat.

4. Algoritma Dijkstra, yaitu salah satu algoritma shortest path dengan paradigma pemrograman greedy. Algoritma Dijkstra dapat menentukan rute terpendek dari salah satu simpul ke semua simpul yang lain dalam sebuah graf berbobot positif dengan kompleksitas waktu O(E log V), E adalah jumlah sisi dan V adalah jumlah simpul.

5. Optimasi algoritma, yaitu proses untuk meningkatkan kinerja atau efisiensi dari sebuah algoritma. Optimasi algoritma dapat dilakukan dengan berbagai cara, seperti memilih struktur data yang sesuai, mengurangi kompleksitas waktu atau ruang, menggunakan teknik-teknik seperti memoization, dynamic programming, atau divide and conquer, dan lain-lain.

5 Peran dan Tugas Tim

Dalam pengerjaan tugas besar ini, kami membagi peran dan tugas tim dengan rincian sebagai berikut

Table 1: Tabel Peran dan Tugas Tim

Nama	NIM	**Peran**	**Tugas**
Yayat Nurhidayat	16523080	pemrogram	membuat program dan membuat laporan
Zaka Hanif	19623010	UI/UX Designer	Mendesain GUI
Nafhan Hadiyan	16523213	Dokumentasi	Mengedit video
Atharizza Muhammad A.	19623143	Tester	Menguji program dan membuat power point
Ferro Arka Berlian	19623213	Dokumentasi	Menyunting laporan

6 Referensi

Berikut disajikan referensi yang kami gunakan untuk perancangan sistem rute tercepat dan rute terdekat *e-maps* dan penulisan laporan.

- Antti Laaksonen, *Competitive Programmer's Handbook*, CSES, 2018.
- William Ghozali & Alham Fikri Azi, *Pemrograman Kompetitif Dasar*, CV. Nulisbuku Jendela Dunia, 2014.
- Rinaldi Munir, *Matematika Diskrit*, INFORMATIKA, 2020.

7 Lampiran

source code: `https://github.com/ynht-edu/maps-shortest-path/tree/CLI_optimization`