

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II 2022-2023

**IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK
MENENTUKAN LINTASAN TERPENDEK**

Disusun oleh :

Athif Nirwasito 13521053

Addin Munawwar Yusuf 13521085

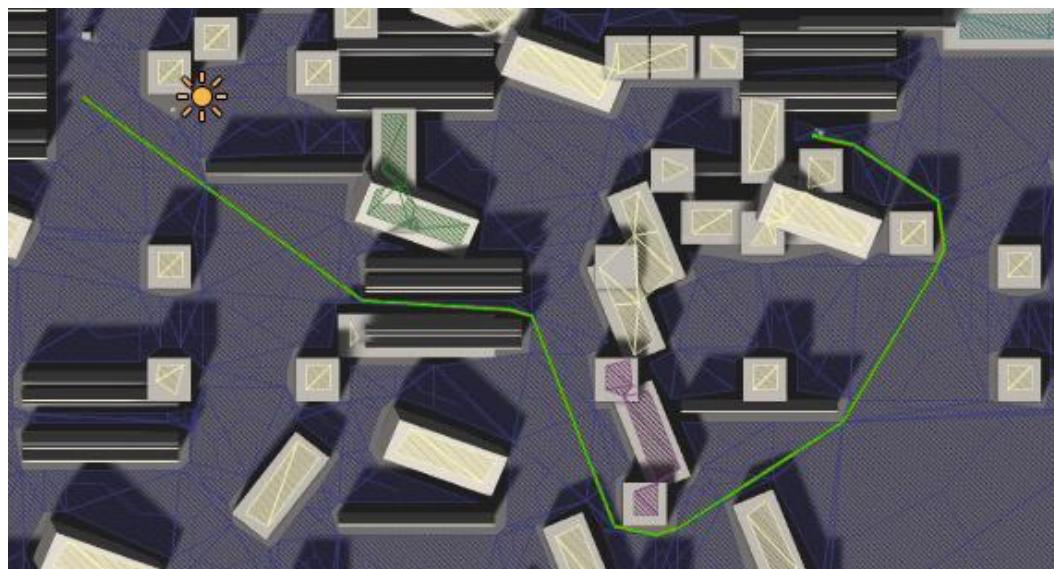


**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023**

1. Deskripsi Persoalan

Permasalahan pencarian rute, atau disebut juga sebagai persoalan lintasan terdekat adalah persoalan yang meminta lintasan terdekat antara dua node pada sebuah graf atau peta. Tujuan dari persoalan ini adalah mencari lintasan yang memiliki *cost/harga* yang paling minimum, dimana harga tersebut sendiri dapat digambarkan dengan berbagai macam parameter. Sebagai contoh, harga bisa digambarkan dengan jarak, waktu, atau misalnya biaya tertentu.

Secara formal, persoalan ini dideskripsikan sebagai berikut. Diberikan sebuah graf $G = (V, E)$ dengan sekumpulan *node* V dan *edge* E . Diberikan dua buah node s (*starting node*) dan t (*target node*), dimana $(s, t \in V)$. Carilah lintasan dari *node* s ke *node* t yang memiliki *cost* minimum. Persoalan ini memiliki aplikasi yang luas, seperti pada sistem transportasi, jaringan komputer, bahkan hingga dalam media interaktif seperti vidio gim.



Gambar 1. Aplikasi Pencarian Rute pada Game
[Sumber : <https://arongranberg.com/astar/>]

Solusi kasar/ *brute force* dari persoalan ini memerlukan pencarian seluruh lintasan yang mungkin antara node asal ke node tujuan. Pada sebuah graf lengkap dengan jumlah node n , jumlah lintasan yang mungkin adalah 2^n , sehingga kompleksitasnya adalah $O(2^n)$.

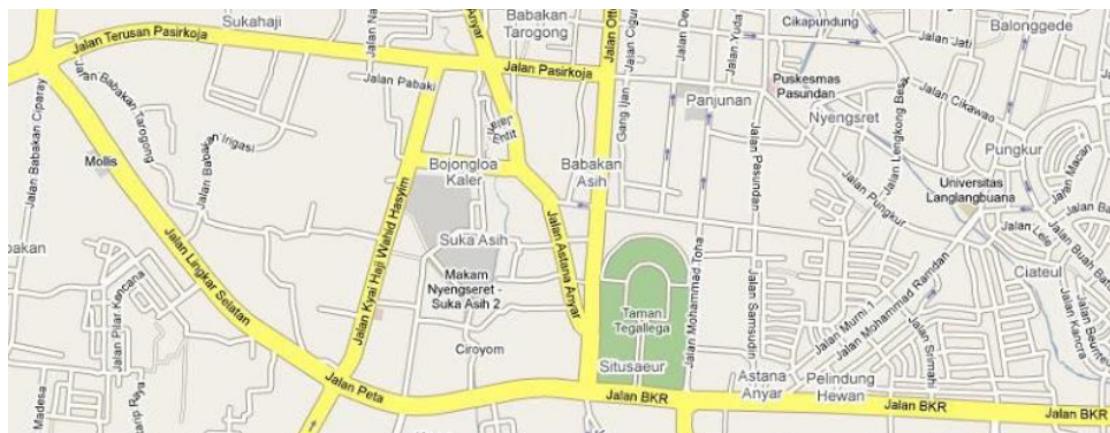
Kompleksitas bertumbuh secara eksponensial, sehingga untuk graph yang lebih besar, solusi ini kurang layak untuk digunakan.

Algoritma yang sering digunakan dalam menyelesaikan persoalan lintasan terdekat adalah algoritma *searching*, seperti Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), A* dan lain-lain. Pada tugas kali ini, penulis akan menggunakan algoritma UCS dan A* untuk mencari lintasan terdekat.

2. Spesifikasi Tugas

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan.

Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antar dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau caralainnya yang disediakan oleh Google Map.



Gambar 2. Peta Bandung di Daerah Tegallega

[Sumber : Spesifikasi Tugas Kecil 3 Strategi Algoritma IF2211 2022/2023]

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago).

Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

Spesifikasi program:

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

Bonus nilai diberikan jika dapat menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta (berupa jalan yang diberi warna). Simpul graf diperoleh dari peta (menggunakan API Google Map) dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Euclidean.

3. Teori Singkat

3.1. Representasi Graf

Graf adalah suatu struktur data yang mempresentasikan objek-objek diskrit dan relasinya dengan objek lain. Objek-objek tersebut direpresentasikan oleh *node* dan relasinya dengan *edge*. Graf dapat direpresentasikan dengan beberapa cara. Salah satu cara yang paling umum adalah dengan menggunakan matriks ketetanggan (*adjacency matrix*).

Graf dengan jumlah node n dapat direpresentasikan dengan matriks ketetanggan berukuran $n \times n$. Baris merepresentasikan node asal dan kolom merepresentasikan node tujuan. Sebagai contoh, suatu graf dengan 4 node memiliki matriks ketetanggan sebagai berikut

0	15	0	18
13	0	2	9
4	0	0	3
19	9	0	0

Hubungan dari node 1 ke node 4 dapat kita temukan di matriks pada baris 1 dan kolom 4. Dalam hal ini, nilai yang ditemukan adalah 18. Artinya, node 1 memiliki hubungan ke node 4 dengan bobot sebesar 18. Dalam masalah pencarian rute, node dapat digambarkan sebagai persimpangan jalan dan bobot sebagai jarak antara node.

Untuk menyelesaikan persoalan lintasan terpendek selanjutnya, penulis akan menggunakan matriks ketetanggan sebagai representasi graf.

3.2. Algoritma Pencarian Uniform Cost Search

Uniform Cost Search (selanjutnya disebut sebagai UCS) adalah pencarian graf secara traversal untuk mencari lintasan dari node asal ke node tujuan dengan biaya termurah. Syarat dari UCS adalah biaya dari tiap *edge* tidak boleh negatif dan boleh berbeda. Algoritma UCS termasuk ke dalam *Uninformed Search*, karena kita tidak memiliki informasi tambahan untuk membantu pencarian.

UCS bekerja dengan menyimpan sebuah *priority queue*, dengan prioritasnya adalah akumulasi *cost*/harga lintasan dari node asal ke node tersebut (sering direpresentasikan sebagai $g(n) \rightarrow cost$ dari node asal ke node n). Karena yang kita cari adalah minimasi, maka lintasan yang dipilih adalah lintasan dengan *cost* minimum.

Algoritma dimulai dengan menginisialisasi priority queue dengan node asal. Untuk setiap node yang kita kunjungi, kita bangkitkan anak-anak dari node tersebut mirip dengan BFS. Masukkan ke priority queue dengan nilai prioritas adalah *cost* dari node asal ke node tersebut. Kemudian, ulangi dengan mengakses elemen selanjutnya pada queue. Algoritma selesai jika node tujuan telah dicapai, atau queue habis, menandakan tidak ada lintasan yang ditemukan.

UCS menjamin lintasan yang ditemukan adalah lintasan yang optimum, selama bobot *edge* tidak negatif.

3.3. Algoritma Pencarian A*

Algoritma A* termasuk ke dalam algoritma pencarian dengan informasi (*informed search*). Pada A*, terdapat informasi tambahan yang dapat membantu pencarian menuju ke arah yang lebih mungkin untuk menjadi lintasan hasil. Informasi ini dibungkus ke dalam sebuah fungsi heuristik yang dapat memberikan aproksimasi jarak dari suatu node ke node tujuan.

Algoritma A* dapat disebut sebagai *extension* dari UCS, karena A* hanya menambahkan satu fitur tambahan dari UCS, yaitu pertimbangan secara heuristik. Selain itu, cara kerja algoritma A* dan UCS kurang lebih sama. Keduanya

sama-sama menggunakan priority queue, membangkitkan node-node tetangga seperti BFS dan mengakses elemen queue hingga node tujuan ditemukan atau queue habis.

Perbedaan UCS dan A* hanya terletak pada penentuan nilai prioritas node nya saja. Jika pada UCS, prioritas ditentukan dari jumlah *cost* lintasan dari node awal ke node n ($g(n)$), A* hanya menambahkan $g(n)$, dengan suatu nilai yang didapat dari fungsi heuristik ($h(n)$). Sehingga, nilai prioritas elemen queue pada algoritma A* adalah $g(n) + h(n)$.

Secara umum, A* lebih efektif dan efisien dibandingkan UCS, selama fungsi heuristik yang digunakan *admissible* dan konsisten (akan dibahas di bagian selanjutnya).

3.4. Fungsi Heuristik A*

Fungsi heuristik pada algoritma A* digunakan untuk memberikan estimasi jarak/bobot antar node tertentu dengan node tujuan. Fungsi heuristik ini dapat memandu pencarian agar lebih efisien dan cepat untuk menuju node tujuan.

Syarat dari fungsi heuristik dari algoritma A* adalah *admissible*. Fungsi heuristik $h(n)$ *admissible*, jika untuk setiap node n, $h(n) \leq h^*(n)$ dimana $h^*(n)$ adalah *cost* sesungguhnya dari node n ke node tujuan. Maka dari itu, fungsi heuristik harus optimistis, tidak pernah boleh meng-*overestimate cost* yang sebenarnya.

Fungsi heuristik yang umum dalam persoalan pencarian rute adalah jarak garis lurus dari node n ke node tujuan. Beberapa contohnya antara lain jarak *euclidean*, jarak *manhattan*, dan lain-lain. Selain itu, heuristik biasanya lebih spesifik, tergantung dengan domain persoalan yang dibahas.

Salah satu heuristik umum lainnya adalah *pattern database*, yaitu pemecahan ruang status ke dalam subproblem yang lebih kecil, selesaikan subproblem masing-masing, dan solusi tersebut digunakan untuk mengestimasi nilai heuristik dari persoalan awal.

4. Implementasi Teori

4.1. Flow Program

1. Pengguna memilih apakah ingin melakukan input melalui file, atau memilih langsung melalui peta
2. Jika memilih untuk input melalui file, maka pengguna akan diminta untuk memilih file yang akan digunakan dari file explorer. Sementara itu, untuk input melalui peta, maka pengguna menandai secara langsung titik-titik di peta untuk dicarikan rutenya.
3. Setelah input, program akan membuat visualisasi graf dari input tersebut.
4. Pengguna memilih node asal dan node tujuan.
5. Pengguna memilih algoritma yang akan digunakan.
6. Pengguna menekan tombol ‘Run Algorithm’, algoritma dijalankan. Setelah algoritma selesai, lintasan yang ditemukan akan ditampilkan secara berbeda di graf, dan *cost* lintasan tersebut akan ditunjukkan pada *message box*.

4.2. Algoritma

4.2.1. Uniform Cost Search

Pada data struktur node yang dibuat, node menyimpan informasi dari *cost* ke node tersebut dan rute yang dilewati.

Langkah-langkahnya:

1. Algoritma menerima input berupa matriks ketetanggan yang berisi jarak sesungguhnya antar node. Algoritma juga menerima node asal dan node tujuan.
2. Priority queue dibuat dan diisi dengan node asal.
3. Selama belum sampai di node tujuan dan queue belum kosong, lakukan:
 - a. Ambil node di queue, jika node adalah node tujuan, maka lanjut ke step 4.

- b. Untuk setiap node $n \in$ tetangga dari node saat ini, tambahkan elemen queue baru dengan prioritas:

$$f(n) = g(n) = \underline{cost}(\text{node awal, saat ini}) + \underline{cost}(\text{node saat ini, node } n)$$

Pada elemen node yang ditambahkan, disimpan juga lintasan untuk mencapai node tersebut.

- c. Lakukan hingga node tujuan ditemukan atau queue habis.
4. Node tujuan ditemukan, kembalikan lintasan solusi dan *cost* yang didapat. Alternatifnya, jika queue habis, maka tidak ada lintasan yang ditemukan, kembalikan None.

4.2.2. Algoritma A*

Pada data struktur node yang dibuat, node menyimpan informasi dari *cost* ke node tersebut dan rute yang dilewati.

Untuk menjalankan algoritma A*, diperlukan informasi tambahan yang dapat merumuskan heuristik dari persoalan yang ada. Dalam hal ini, matriks ketetanggaan jarak antar node saja tidak akan cukup untuk menurunkan suatu fungsi heuristik (termasuk misalnya dengan minimum-spanning tree, tidak akan bisa).

Maka dari itu, diperlukan informasi tambahan dari pengguna, yaitu koordinat dari masing-masing node. Dari informasi tersebut, jarak ke node goal dapat diaproksimasi dengan jarak *euclidean*.

Langkah-langkahnya:

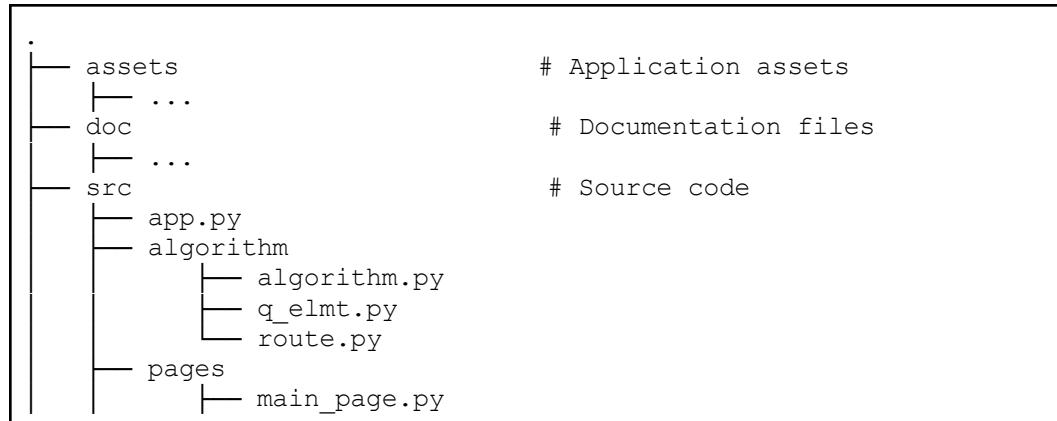
1. Algoritma menerima input berupa matriks ketetanggan yang berisi jarak sesungguhnya antar node. Algoritma juga menerima node asal dan node tujuan.

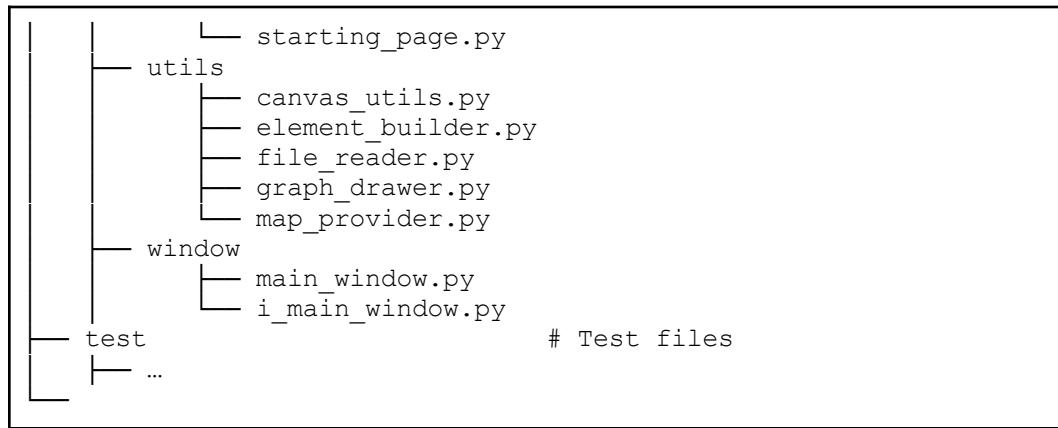
Sebagai tambahan, A* juga mendapatkan matriks ketetanggan $h(n)$ yang berisi nilai aproksimasi jarak node n ke node tujuan, dengan jarak *euclidean*.

2. Priority queue dibuat dan diisi dengan node asal.
3. Selama belum sampai di node tujuan dan queue belum kosong, lakukan:
 - a. Ambil node di queue, jika node adalah node tujuan, maka lanjut ke step 4.
 - b. Untuk setiap node $n \in$ tetangga dari node saat ini, tambahkan elemen queue baru dengan prioritas:
$$f(n) = g(n) + h(n) = \underline{\text{cost}}(\text{node awal}, \text{saat ini}) + \underline{\text{cost}}(\text{node saat ini}, \text{node } n) + \underline{\text{cost approx}}(\text{node } n, \text{node tujuan})$$
- Pada elemen node yang ditambahkan, disimpan juga lintasan untuk mencapai node tersebut.
- c. Lakukan hingga node tujuan ditemukan atau queue habis.
4. Node tujuan ditemukan, kembalikan lintasan solusi dan *cost* yang didapat. Alternatifnya, jika queue habis, maka tidak ada lintasan yang ditemukan, kembalikan None.

4.3. Struktur Program

Beberapa implementasi tidak akan ditunjukkan secara lengkap untuk mengurangi jumlah halaman, terutama yang berkaitan dengan GUI. Program dibuat dalam bahasa *python*. Berikut adalah struktur folder dari source code yang dibuat.





5. Source Code

5.1. Algoritma

algorithm.py

Modul untuk algoritma Uniform Cost Search dan A*

```
from queue import PriorityQueue
from algorithm.q_elmt import Elmt
from algorithm.route import Route

def solve(matrix, start, finish, astar, astarMatrix = None):
    # Algoritma USC dan astar
    # set astar = false untuk USC
    # set astar true untuk astar
    queue = PriorityQueue()
    queue.put(Elmt(Route(), start, 0, 0))
    found = False
    while not queue.empty() and not found:
        #get elmt
        temp = queue.get()
        if(temp.GetNode() in temp.GetRoute().GetBuffer()):
            continue

        newRoute = Route(temp.GetRoute())
        newRoute.addNode(temp.GetNode())
        if(temp.GetNode() == finish):
```

```
        found = True
        route = newRoute
    else:
        for i in getNeighbour(matrix, temp.GetNode()):
            new = Elmt(newRoute, i, matrix[temp.GetNode()][i] +
temp.GetCost(), matrix[temp.GetNode()][i] + temp.GetCost() +
(astarMatrix[i][finish] if astar else 0))
            queue.put(new)
    if(found):
        return route.GetBuffer(), temp.GetCost()
    else:
        return None, None

def createEuclidDistanceMatrix(arrayOfCoor):
    matrix = []
    for i in range(len(arrayOfCoor)):
        row = []
        for j in range(len(arrayOfCoor)):
            row.append(((arrayOfCoor[i][1]-arrayOfCoor[j][1])**2 +
(arrayOfCoor[i][0]-arrayOfCoor[j][0])**2)**0.5)
        matrix.append(row)
    return matrix

def getNeighbour(matrix, node):
    array = []
    for i in range(len(matrix[node])):
        if matrix[node][i] != 0:
            array += [i]
    return array
```

q_elmt.py

Modul kelas elemen dari priority queue

```
from algorithm.route import Route
class Elmt:
    def __init__(self, route, node, cost, priority = None):
        self.route = route
        self.node = node
        self.cost = cost
        self.priority = priority
    def __lt__(self, other):
        return self.priority<other.priority
    def __eq__(self, other):
        return self.priority == other.priority
    def GetNode(self):
        return self.node
    def GetRoute(self):
        return self.route
    def GetCost(self):
        return self.cost
```

route.py

Modul kelas rute pada algoritma pencarian

```
class Route:
    def __init__(self, other = None):
        if other == None:
            self.buffer = []
        else:
            self.buffer = other.GetBuffer().copy()
    def __str__(self):
        return str(self.buffer)
    def addNode(self, node):
        self.buffer += [node]

    def exists(self, node):
        for i in self.buffer:
            if(i == node):
                return True
        return False
    def GetBuffer(self):
        return self.buffer
```

5.2. Pages

starting_page.py

Frame yang terbuka ketika pertama kali membuka program. Pengguna memilih mode input (melalui file atau peta)

```
from tkinter import Frame, Canvas
from utils.canvas_utils import *
from window.i_main_window import IMainWindow

class StartingPage(Frame):
    def __init__(self, window : IMainWindow):
        super().__init__(window)
        self.window = window # save window reference
        self.assets = []      # save assets reference
        self.build()

    def build(self):
        # UI, LAYOUT, ETC.
        # ...

        # button 1 : file input
        add_img(page_canvas, "starting_page/file-input-btn.png")
        btn1 = page_canvas.create_image(120, 400, image=self.assets[-1],
            anchor="nw")
        make_button(page_canvas, btn1, lambda e:
            self.window.open_page(" MainPage_FileInput"))

        # button 2 : map pick
        add_img(page_canvas, "starting_page/map-pick-btn.png")
        btn2 = page_canvas.create_image(375, 400, image=self.assets[-1],
            anchor="nw")
        make_button(page_canvas, btn2, lambda e:
            self.window.open_page(" MainPage_MapPick"))

        # UI, LAYOUT, ETC.
        # ...
```

main_page.py

Modul kelas frame tkinter yang menjadi gerbang interaksi pengguna dengan program.

```
from tkinter import Frame, Canvas, StringVar, IntVar, Label, messagebox
from tkinter.ttk import Combobox
from utils.canvas_utils import *
from utils.file_reader import *
from utils.element_builder import DropdownBuilder, FilePickerBuilder
from utils.graph_drawer import GraphCanvas
from utils.map_provider import MapView
from window.i_main_window import IMainWindow
from algorithm.algorithm import solve, createEuclidDistanceMatrix
```

```
""" Main Page for File Input Mode"""
class MainPage_FileInput(Frame):
    def __init__(self, window : IMainWindow):
        super().__init__(window)
        self.window = window
        self.assets = []
        self.graph_canvas : GraphCanvas = None
        self.f_n = []
        self.h_n = []

        self.nodeIndexOf = {}
        self.node_names = []

        # dropdown options
        self.vars = {
            "start_node": StringVar(value="Load File First"),
            "dest_node": StringVar(value="Load File First"),
            "algorithm": StringVar(value="Not Selected"),
            "file_path": StringVar(value="None"),
            "num_of_nodes": IntVar(value=0),
            "message": StringVar(value="Please load a file first before
running the algorithm"),
        }

        self.build()

    def build(self):
        # layout
        self.rowconfigure(0, weight=2, uniform="a")
        self.rowconfigure(1, weight=8, uniform="a")
        self.rowconfigure(2, weight=2, uniform="a")
        self.columnconfigure(0, weight=3, uniform="a")
        self.columnconfigure(1, weight=7, uniform="a")

        # header
        header = MainPage_Header(self)
        header.grid(row=0, column=0, columnspan=2, sticky="nsew")
```

```
# body
body = MainPage_FileInput_Body(self)
body.grid(row=1, column=0, rowspan=2, sticky="nsew")

# footer
footer = MainPage_Footer(self)
footer.grid(row=2, column=1, sticky="nsew")

def run_algorithm(self):
    if self.vars["file_path"].get() == "None":
        self.vars["message"].set("Please load a file first before
running the algorithm")
        messagebox.showerror("Error", "Please load a file first
before running the algorithm")
        return
    elif self.vars["start_node"].get() == "Select Node" or
        self.vars["dest_node"].get() == "Select Node":
        self.vars["message"].set("Please select a start and
destination node")
        messagebox.showerror("Error", "Please select a start and
destination node")
        return
    elif self.vars["algorithm"].get() == "Not Selected":
        self.vars["message"].set("Please select the search
algorithm")
        messagebox.showerror("Error", "Please select the search
algorithm")
        return
    else:
        self.vars["message"].set("Running Algorithm...")
        if self.vars["algorithm"].get() == "Uniform-Cost Search":
            solution, cost = solve(self.f_n,
self.nodeIndexOf[self.vars["start_node"].get()],
self.nodeIndexOf[self.vars["dest_node"].get()],
False)
            if solution == None:
```

```
        self.vars["message"].set("No solution found")
        messagebox.showerror("Error", "No solution found")
    else:
        self.vars["message"].set("Solution Found with UCS!
Cost: " + str(cost) + " units. " +
                    "Route: " + " -> ".join([self.node_names[i] for
i in solution]))
        self.graph_canvas.draw_solution_route(solution)
elif self.vars["algorithm"].get() == "A* Search":
    solution, cost = solve(self.f_n,
self.nodeIndexOf[self.vars["start_node"].get()],
self.nodeIndexOf[self.vars["dest_node"].get()],
True, self.h_n)
    if solution == None:
        self.vars["message"].set("No solution found")
        messagebox.showerror("Error", "No solution found")
    else:
        self.vars["message"].set("Solution Found with A*!
Cost: " + str(cost) + " units. " +
                    "Route: " + " -> ".join([self.node_names[i] for
i in solution]))
        self.graph_canvas.draw_solution_route(solution)

"""
Header Component for Main Page"""
class MainPage_Header(Canvas):
    def __init__(self, parent):
        super().__init__(parent, bg="#07111F")
        self.build()

    def build(self):
        self.create_text(40, 35, text="Route Finder",
font=("Montserrat", 32, "bold"), fill="#E2BD45", anchor="nw")

        add_img(self, "main_page/header-logo.png")
        self.create_image(1000, 30, image=self.master.assets[-1],
```

```
        anchor="nw")

"""Body Component for Main Page (File Input Mode)"""
class MainPage_FileInput_Body(Canvas):
    def __init__(self, parent):
        super().__init__(parent)
        change_background(self, "main_page/body-bg.png")
        self.parent = parent

        self.start_dropdown : Combobox = None
        self.dest_dropdown : Combobox = None

        self.node_coors = []

        self.build()

    def build(self):
        builder_dropdown = DropdownBuilder(self)
        builder_file_picker = FilePickerBuilder(self)

        # Create file picker
        builder_file_picker.create(40, 40)

        # Create dropdowns
        self.start_dropdown = builder_dropdown.create("Starting Node",
            [], 40, 140, self.parent.vars["start_node"])
        self.start_dropdown.state(["disabled"])

        self.dest_dropdown = builder_dropdown.create("Destination Node",
            [], 40, 240, self.parent.vars["dest_node"])
        self.dest_dropdown.state(["disabled"])

        builder_dropdown.create("Algorithm", ["Uniform-Cost Search", "A*"
            Search"], 40, 340, self.parent.vars["algorithm"])

        # Start button
        self.create_text(40, 455, text="Start Finding",
```

```
        font=("Montserrat", 16, "bold"), fill="white", anchor="nw")
    img_start_btn = add_img(self, "main_page/start-button.png", 1)
    start_btn = self.create_image(200, 450, image=img_start_btn,
        anchor="nw")
    make_button(self, start_btn, lambda e:
        self.parent.run_algorithm())

    # Clear button
    self.create_text(40, 505, text="Clear", font=("Montserrat", 16,
        "bold"), fill="white", anchor="nw")
    img_clear_btn = add_img(self, "main_page/clear-button.png", 1)
    clear_btn = self.create_image(200, 500, image=img_clear_btn,
        anchor="nw")
    make_button(self, clear_btn, lambda e:
        self.parent.window.refresh_page())

# File picker callback
def on_file_picked(self, file_path : str):
    self.parent.vars["file_path"].set(file_path)

    nodes, node_names, node_coors = read_file_to_nodes(file_path)
    if (nodes == None):
        messagebox.showerror("Error", "File format is invalid")
        return
    self.parent.node_names = node_names
    self.create_graph(nodes, node_names)

    self.parent.f_n = nodes
    self.parent.h_n = createEuclidDistanceMatrix(node_coors)

    self.parent.vars["num_of_nodes"].set(len(nodes))
    self.parent.vars["message"].set("File " +
        file_path.split("/")[-1] + " loaded successfully")

    for i in range(len(node_names)):
        self.parent.nodeIndexOf[node_names[i]] = i
```

```
self.start_dropdown.config(values=[node_names[i] for i in
    range(len(node_names))])
self.start_dropdown.state(["!disabled"])
self.parent.vars["start_node"].set("Select Node")

self.dest_dropdown.config(values=[node_names[i] for i in
    range(len(node_names))])
self.dest_dropdown.state(["!disabled"])
self.parent.vars["dest_node"].set("Select Node")

def create_graph(self, nodes, node_names) :
    self.parent.graph_canvas = GraphCanvas(self.parent, nodes,
        node_names = node_names if node_names else None)
    self.parent.graph_canvas.grid(row=1, column=1, sticky="nsew")

"""Footer Component for Main Page"""
class MainPage_Footer(Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="#07111F")
        self.parent = parent
        self.build()

    def build(self):
        # layout
        self.rowconfigure(0, weight=1, uniform="a")
        self.rowconfigure(1, weight=1, uniform="a")
        self.rowconfigure(2, weight=1, uniform="a")
        self.rowconfigure(3, weight=1, uniform="a")

        self.columnconfigure(0, weight=5, uniform="a")
        self.columnconfigure(1, weight=10, uniform="a")
        self.columnconfigure(2, weight=2, uniform="a")
        self.columnconfigure(3, weight=80, uniform="a")

        # create footer
        total_nodes_label = Label(self, text="Total Nodes : ",
            font=("Montserrat", 12, "normal"), bg="#07111F", fg="white",
```

```
        anchor="w")
total_nodes_label.grid(row=1, column=1, sticky="nsew",
    columnspan=2)

total_nodes_value = Label(self,
    textvariable=self.parent.vars["num_of_nodes"],
    font=("Montserrat", 12, "normal"), bg="#07111F",
    fg="#E2BD45", anchor="w")
total_nodes_value.grid(row=1, column=3, sticky="nsew")

message_label = Label(self, text="Message : ",
    font=("Montserrat", 12, "normal"), bg="#07111F", fg="white",
    anchor="w")
message_label.grid(row=2, column=1, sticky="nsew")

message_value = Label(self,
    textvariable=self.parent.vars["message"],
    font=("Montserrat", 12, "normal"), bg="#07111F", fg="white",
    anchor="w")
message_value.grid(row=2, column=2, sticky="nsew", columnspan=2)

""" Main Page for File Input Mode"""
class MainPage_MapPick(Frame):
    def __init__(self, window : IMainWindow):
        super().__init__(window)
        self.window = window
        self.assets = []

        self.nodeIndexOf = {}

        # dropdown options
        self.vars = {
            "start_node": StringVar(value="Select 5 Nodes or More"),
            "dest_node": StringVar(value="Select 5 Nodes or More"),
            "algorithm": StringVar(value="Not Selected"),
            "num_of_nodes": IntVar(value=0),
            "message": StringVar(value="Select at least 5 nodes before"

```

```
        running the algorithm. Right click to add nodes"),
    }

    self.start_dropdown : Combobox = None
    self.dest_dropdown : Combobox = None

    self.map_view = None

    self.f_n = None
    self.h_n = None

    self.node_names = None

    self.build()

def build(self):
    # layout
    self.rowconfigure(0, weight=2, uniform="a")
    self.rowconfigure(1, weight=8, uniform="a")
    self.rowconfigure(2, weight=2, uniform="a")
    self.columnconfigure(0, weight=3, uniform="a")
    self.columnconfigure(1, weight=7, uniform="a")

    # header
    header = MainPage_Header(self)
    header.grid(row=0, column=0, columnspan=2, sticky="nsew")

    # body
    body = MainPage_MapPick_Body(self)
    body.grid(row=1, column=0, rowspan=2, sticky="nsew")
    self.start_dropdown = body.start_dropdown
    self.dest_dropdown = body.dest_dropdown

    # map view
    self.map_view = MapView(self)
    self.map_view.grid(row=1, column=1, sticky="nsew")
```

```
# footer
footer = MainPage_Footer(self)
footer.grid(row=2, column=1, sticky="nsew")

def run_algorithm(self):
    self_f_n = self.map_view.f_n
    self_h_n = self.map_view.h_n

    if self.vars["start_node"].get() == "Select Node" or
       self.vars["dest_node"].get() == "Select Node":
        self.vars["message"].set("Please select a start and
destination node")
        messagebox.showerror("Error", "Please select a start and
destination node")
        return
    elif self.vars["algorithm"].get() == "Not Selected":
        self.vars["message"].set("Please select the search
algorithm")
        messagebox.showerror("Error", "Please select the search
algorithm")
        return
    else:
        self.vars["message"].set("Running Algorithm...")
        if self.vars["algorithm"].get() == "Uniform-Cost Search":
            solution, cost = solve(self_f_n,
self.nodeIndexOf[self.vars["start_node"].get()],
self.nodeIndexOf[self.vars["dest_node"].get()],
False, None)
            if solution == None:
                self.vars["message"].set("No solution found")
                messagebox.showerror("Error", "No solution found")
            else:
                self.vars["message"].set("Solution Found with UCS!
Cost: " + str(cost) + " units. " +
"Route: " + " -> ".join([self.node_names[i] for
i in solution]))
                self.map_view.draw_solution_route(solution)
```

```
        elif self.vars["algorithm"].get() == "A* Search":
            solution, cost = solve(self_f_n,
self.nodeIndexOf[self.vars["start_node"].get()],
self.nodeIndexOf[self.vars["dest_node"].get()],
True, self.h_n)
            if solution == None:
                self.vars["message"].set("No solution found")
                messagebox.showerror("Error", "No solution found")
            else:
                self.vars["message"].set("Solution Found with A*!
Cost: " + str(cost) + " units. "
"Route: " + " -> ".join([self.node_names[i] for
i in solution]))
                self.map_view.draw_solution_route(solution)
def on_marker_added(self):
    self.vars["num_of_nodes"].set(len(self.map_view.markers))
    places = [place_name for place_name in self.nodeIndexOf.keys()]
    dropdown_values = places
    self.node_names = places

    if len(self.map_view.markers) >= 5:
        self.start_dropdown.config(values=dropdown_values)
        self.start_dropdown.state(["!disabled"])
        self.vars["start_node"].set("Select Node")

        self.dest_dropdown.config(values=dropdown_values)
        self.dest_dropdown.state(["!disabled"])
        self.vars["dest_node"].set("Select Node")

        self.vars["message"].set("Keep adding nodes or run the
algorithm")

"""Body Component for Main Page (Map Pick Mode)"""
class MainPage_MapPick_Body(Canvas):
    def __init__(self, parent):
        super().__init__(parent)
        change_background(self, "main_page/body-bg.png")
```

```
self.parent = parent

self.start_dropdown : Combobox = None
self.dest_dropdown : Combobox = None

self.build()

def build(self):
    builder_dropdown = DropdownBuilder(self)

    # Create dropdowns
    self.start_dropdown = builder_dropdown.create("Starting Node",
        [], 40, 40, self.parent.vars["start_node"])
    self.start_dropdown.state(["disabled"])

    self.dest_dropdown = builder_dropdown.create("Destination Node",
        [], 40, 140, self.parent.vars["dest_node"])
    self.dest_dropdown.state(["disabled"])

    builder_dropdown.create("Algorithm", ["Uniform-Cost Search", "A* Search"], 40, 240, self.parent.vars["algorithm"])

    # Start button
    self.create_text(40, 355, text="Start Finding",
        font=("Montserrat", 16, "bold"), fill="white", anchor="nw")
    img_start_btn = add_img(self, "main_page/start-button.png", 1)
    start_btn = self.create_image(200, 350, image=img_start_btn,
        anchor="nw")
    make_button(self, start_btn, lambda e:
        self.parent.run_algorithm())

    # Clear button
    self.create_text(40, 405, text="Clear", font=("Montserrat", 16,
        "bold"), fill="white", anchor="nw")
    img_clear_btn = add_img(self, "main_page/clear-button.png", 1)
    clear_btn = self.create_image(200, 400, image=img_clear_btn,
        anchor="nw")
```

```
make_button(self, clear_btn, lambda e:  
    self.parent.window.refresh_page())
```

5.3. Utils

canvas_utils.py

Modul yang berisi fungsi-fungsi utilitas dari *tkinter canvas*

```
from tkinter import Canvas  
from PIL import ImageTk, Image  
  
# notice : image_path must be relative to asset folder  
def change_background(canvas : Canvas, image_path : str):  
  
    canvas.master.assets.append(ImageTk.PhotoImage(Image.open("../  
        asset/" + image_path)))  
    canvas.create_image(0, 0, image=canvas.master.assets[-1],  
        anchor="nw")  
  
def add_img (canvas : Canvas, image_path : str, size_factor = 1):  
    img = Image.open("../asset/" + image_path)  
    img = img.resize((int(img.width * size_factor), int(img.height *  
        size_factor)), Image.ANTIALIAS)  
    canvas.master.assets.append(ImageTk.PhotoImage(img))  
    return canvas.master.assets[-1]  
  
def make_button(canvas : Canvas, tag : str, callback, change_cursor =  
    True):  
    canvas.tag_bind(tag, "<Button-1>", callback)  
    if change_cursor:  
        canvas.tag_bind(tag, "<Enter>", lambda e:  
            canvas.config(cursor="hand2"))  
        canvas.tag_bind(tag, "<Leave>", lambda e:  
            canvas.config(cursor=""))
```

element_builder.py

Modul kelas untuk builder elemen-elemen pada UI, seperti dropdown dan file picker

```
from tkinter import Canvas, filedialog  
from tkinter.ttk import Combobox  
from utils.canvas_utils import *
```

```
import os

class DropdownBuilder():
    def __init__(self, parent):
        self.parent = parent

    def create(self, title, options, x, y, setvar = None) -> None:
        self.parent.create_text(x, y, text=title, font=("Montserrat",
            18, "bold"), fill="white", anchor="nw")

        option_menu = Combobox(self.parent, textvariable=setvar,
            values=options, state="readonly", font=("Montserrat",
            12, "normal"), width=20)
        option_menu.place(x=x, y=y+40)

        return option_menu

class FilePickerBuilder():
    def __init__(self, parent : Canvas):
        self.parent = parent
        self.file_chosen_label = None

    def create(self, x, y) -> None:
        self.parent.create_text(x, y, text="Select Your File",
            font=("Montserrat", 18, "bold"), fill="white", anchor="nw")

        img_btn = add_img(self.parent,
            "main_page/file-choose-button.png", 1)
        btn = self.parent.create_image(x, y+40, image=img_btn,
            anchor="nw")
        make_button(self.parent, btn, lambda e: self.open_file_dialog())

        self.file_chosen_label = self.parent.create_text(x+175, y+50,
            text="No file chosen", font=("Montserrat", 12, "normal"),
            fill="white", anchor="nw")

    def open_file_dialog(self):
        starting_path = os.getcwd() + "/test/"
        try:
            file_path = filedialog.askopenfilename(
                initialdir = starting_path, title = "Select file",
                filetypes = (("Text files", "*.txt"), ("all files", "*.*")))
        except:
            return

        if file_path == "":
            return
        self.parent.itemconfig(self.file_chosen_label,
            text=self.file_path_to_file_name(file_path))
        self.parent.on_file_picked(file_path)
```

```
def file_path_to_file_name(self, file_path):  
    return file_path.split("/")[-1]
```

file_reader.py

Modul untuk membaca matriks ketetanggan, koordinat, dan penamaan node dari pengguna

```
def read_file_to_nodes(file_path : str):  
    nodes = []  
    node_names = []  
    nodes_coordinates = []  
    file = open(file_path, "r")  
    line = file.readline()  
    row = [float(x) for x in line.split()]  
    nodes.append(row)  
  
    matrix_size = len(row)  
  
    for _ in range(matrix_size - 1):  
        line = file.readline()  
        row = [float(x) for x in line.split()]  
        nodes.append(row)  
  
    try:  
        line = file.readline()  
    except:  
        return nodes, None, None  
  
    try:  
        row = line.split()  
        row_len = len(row)  
        node_name = ""  
        for i in range(0, row_len - 2):  
            node_name += row[i]  
            node_name += " " if i != row_len - 3 else ""  
  
        node_names.append(node_name)
```

```
        nodes_coordinates.append([float(row[-2]), float(row[-1])]))  
    except:  
        return None, None, None  
  
    for _ in range(matrix_size - 1):  
        try:  
            line = file.readline()  
            row = line.split()  
            row_len = len(row)  
            node_name = ""  
            for i in range(0, row_len - 2):  
                node_name += row[i]  
                node_name += " " if i != row_len - 3 else ""  
  
            node_names.append(node_name)  
            nodes_coordinates.append([float(row[-2]), float(row[-1])]))  
        except:  
            return None, None, None  
  
    return nodes, node_names, nodes_coordinates
```

graph_drawer.py

Modul kelas turunan canvas yang menggambar graf ke GUI

```
from tkinter import Canvas  
import networkx as nx  
import matplotlib.pyplot as plt  
from PIL import ImageTk, Image  
  
NODE_COLOR = "#07111F"  
SOLUTION_EDGE_COLOR = "red"  
  
class GraphCanvas(Canvas):  
    def __init__(self, parent, data, node_names = None):  
        super().__init__(parent)  
        self.data = data  
        self.graph = None  
        self.graph_image = None  
        self.output_file_path = ""  
        self.canvas_img = None
```

```
self.build()

def build(self):
    self.draw_graph()
    self.graph_image =
        ImageTk.PhotoImage(Image.open(self.output_file_path))
    self.canvas_img = self.create_image(100, 0,
        image=self.graph_image, anchor="nw")

def draw_graph(self):
    data = self.data
    labelpos = 0.4
    if isSymmetric(data, len(data)):
        self.graph = nx.Graph()
    else:
        self.graph = nx.DiGraph()

    self.graph.add_nodes_from([i for i in range(len(data))])
    self.graph.add_weighted_edges_from(
        [
            (i, j, data[i][j])
            for i in range(len(data))
            for j in range(len(data))
            if data[i][j] != 0
        ]
    )

    pos = nx.spring_layout(self.graph)
    nx.draw(self.graph, pos, node_color=NODE_COLOR, node_size=500,
        with_labels=True, alpha=0.85,
        font_size=10, font_color="white",
        connectionstyle="arc3,rad=0.08", arrows=True, width=2)

    edge_labels = nx.get_edge_attributes(self.graph, "weight")
    nx.draw_networkx_edge_labels(self.graph, pos, font_size=8,
        label_pos=labelpos, font_color="black",
        edge_labels=edge_labels, alpha=0.9)

    self.output_file_path = "../test/result/graph.png"
    plt.savefig(self.output_file_path)
    plt.clf()

# def draw_solution_route(buffer):

# To check if a matrix is symmetric (the graph is undirected) or not
# (which means the graph is directed)
def isSymmetric(mat, N):
    for i in range(N):
        for j in range(N):
            if (mat[i][j] != mat[j][i]):
                return False
```

```
    return True
```

map_provider.py

Modul kelas turunan frame yang berisi map dan segala interaksinya dengan pengguna. Terintegrasi dengan google map api.

```
from tkinterMapView import TkinterMapView,  
    convert_coordinates_to_address, canvas_position_marker  
import googlemaps  
from geopy.distance import geodesic  
  
# Jangan ada yang make yahh :((  
API_KEY = "AIzaSyD653qbfvijlwA6-6a6eQxMX3Q6fKpUasU"  
SAME_NODE_DISTANCE = 0.00015  
class MapView(TkinterMapView):  
    def __init__(self, parent):  
        super().__init__(parent)  
        self.parent = parent  
        self.gmaps : google maps.Client = googlemaps.Client(key=API_KEY)  
        self.markers = []  
        self.steps = []  
        self.node_names = []  
  
        self.f_n = [] # adjacency matrix for real distance  
        # data example : [  
        # [0, 100, 200],  
        # [100, 0, 300],  
        # [200, 300, 0]  
        # ]  
  
        self.h_n = [] # adjacency matrix for heuristic distance  
        # (straight line distance)  
        # data example : [  
        # [0, 100, 200],  
        # [100, 0, 300],  
        # [200, 300, 0]  
        # ]  
  
        self.init_options()  
  
    def init_options(self):  
        self.set_tile_server("https://mt0.google.com/vt/lyrs=m&hl=en&x={x}  
            &y={y}&z={z}&s=Ga", max_zoom=22)  
        self.set_address("Institut Teknologi Bandung")  
        self.add_right_click_menu_command("Add Node",  
            self.on_map_add_node, pass_coords=True)  
  
    def on_map_add_node(self, coords):
```

```
place_name = convert_coordinates_to_address(coords[0],
                                             coords[1]).address.split(",")[-1]
for marker in self.markers:
    if (place_name == marker.text):
        place_name += f" ({len(self.markers)})"
self.parent.nodeIndexOf[place_name] = len(self.markers)
self.node_names.append(place_name)

new_marker = self.set_marker(coords[0], coords[1], text =
                             place_name, text_color="#07111F")
self.markers.append(new_marker)

# ASSUMPTION : Road is undirectional, path from A to B is the
# same as path from B to A
# shorter path will be chosen if A to B and B to A have
# different distance

# Add distance to all nodes to f_n (real distance)
self.f_n.append([])
j = len(self.markers) - 1
for i in range(len(self.markers) - 1):
    distance = self.create_path(j, i)
    self.f_n[i].append(distance)
    self.f_n[j].append(distance)
self.f_n[j].append(0)

# Add distance to all nodes to h_n (heuristic, straight line
# distance)
self.h_n.append([])
j = len(self.markers) - 1
for i in range(len(self.markers) - 1):
    distance = geodesic(new_marker.position,
                         self.markers[i].position).meters # euclidean distance
    self.h_n[i].append(distance)
    self.h_n[j].append(distance)
self.h_n[j].append(0)

self.parent.on_marker_added()

def create_path(self, start, end):
    # Draw path in the map bidirectionally
    # Returns distance start → end, end → start
    directionsA =
        self.gmaps.directions(self.markers[start].position,
                              self.markers[end].position, mode="driving")
    directionsB = self.gmaps.directions(self.markers[end].position,
                                         self.markers[start].position, mode="driving")

    stepsA = directionsA[0]["legs"][0]["steps"]
    if (start == 1):
```

```
        self.steps.append([None])

    if (len(self.steps) <= start):
        self.steps.append([])
    self.steps[start].append(stepsA)

    stepsB = directionsB[0]["legs"][0]["steps"]
    self.steps[end].append(stepsB)

    self.steps[start].append(None)

    starting_pos = (stepsA[0]["start_location"]["lat"],
                    stepsA[0]["start_location"]["lng"])
    position_list = [starting_pos]
    for step in stepsA:
        position_list.append((step["end_location"]["lat"],
                              step["end_location"]["lng"]))
    self.set_path(position_list, color = "#E1341E")

    starting_pos = (stepsB[0]["start_location"]["lat"],
                    stepsB[0]["start_location"]["lng"])
    position_list = [starting_pos]
    for step in stepsB:
        position_list.append((step["end_location"]["lat"],
                              step["end_location"]["lng"]))
    self.set_path(position_list, color = "#E1341E")

    distA = directionsA[0]["legs"][0]["distance"]["value"]
    distB = directionsB[0]["legs"][0]["distance"]["value"]

    return distA, distB

def draw_solution_route(self, list_of_node):
    for i in range(len(list_of_node) - 1):
        steps = self.steps[list_of_node[i]][list_of_node[i + 1]]
        starting_pos = (steps[0]["start_location"]["lat"],
                        steps[0]["start_location"]["lng"])
        position_list = [starting_pos]
        for step in steps:
            position_list.append((step["end_location"]["lat"],
                                  step["end_location"]["lng"]))
        self.set_path(position_list, color = "#E2BD45")
```

5.4. Window

i_main_window.py

Abstraksi dari kelas MainWindow, untuk menghindari import loop

```
class IMainWindow(Tk, ABC):
    def __init__(self, *args, **kwargs):
        # init window
        super().__init__(*args, **kwargs)
        ctypes.windll.shcore.SetProcessDpiAwareness(1)

        # window config
        super().title("USC and A* Route Finder")
        super().geometry("1200x720")
        super().resizable(False, False)

    @abstractmethod
    def open_page(self, page_name):
        pass
```

main_window.py

Kelas MainWindow dari aplikasi yang mengatur *page* yang terbuka

```
from window.i_main_window import IMainWindow
from pages.starting_page import StartingPage
from pages.main_page import MainPage_FileInput, MainPage_MapPick

class MainWindow(IMainWindow):
    def __init__(self):
        super().__init__()

        # window layout
        self.rowconfigure(0, weight=1)
        self.columnconfigure(0, weight=1)

        # init pages
        self.pages = {
            "StartingPage": StartingPage(self),
            "MainPage_FileInput": MainPage_FileInput(self),
            "MainPage_MapPick": MainPage_MapPick(self)
        }

        self.opened_page = self.pages["StartingPage"]
        self.opened_page.grid(row=0, column=0, sticky="nsew")

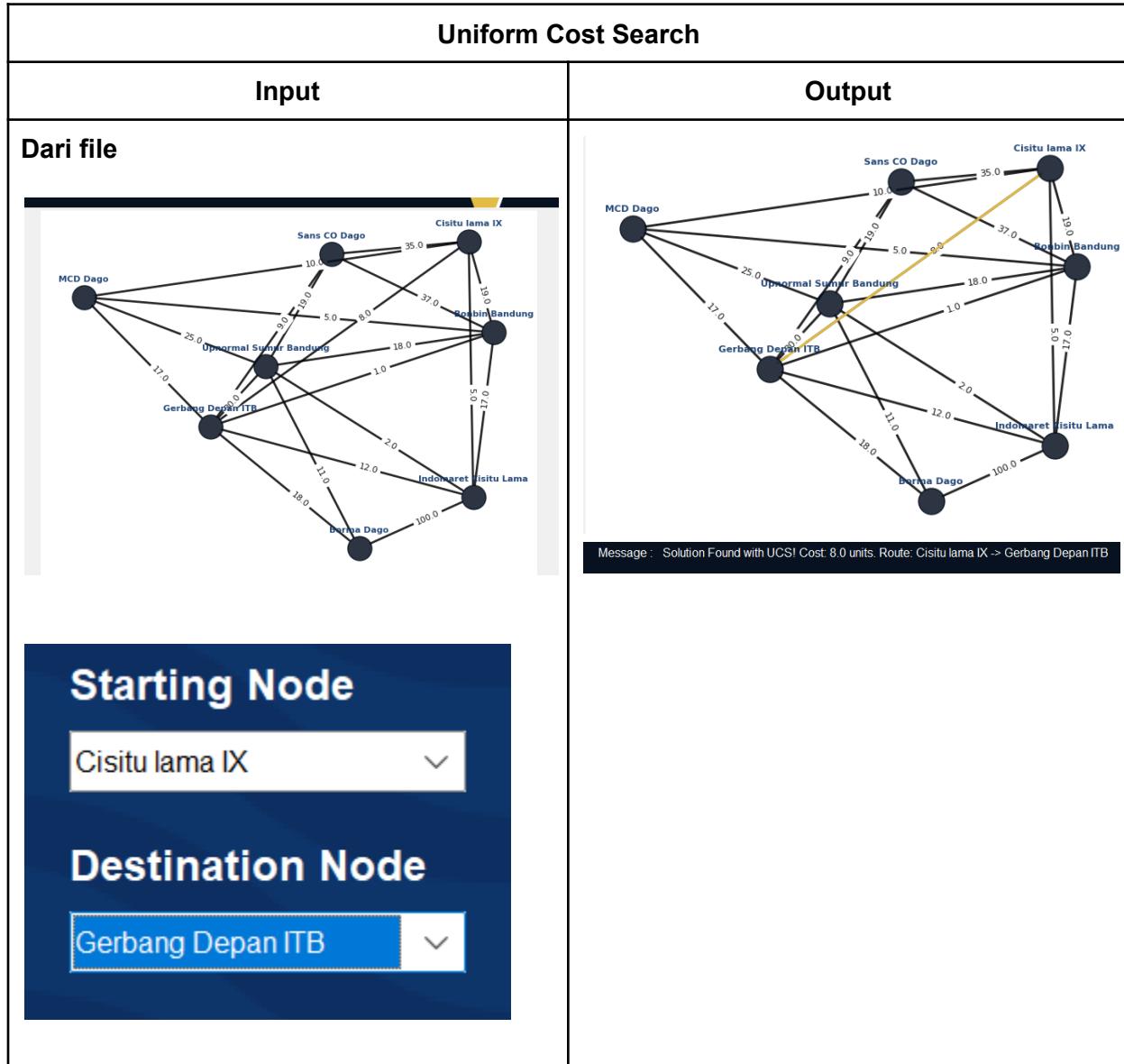
    def open_page(self, page_name):
        # change opened page
        self.opened_page.grid_forget()
        self.opened_page = self.pages[page_name]
        self.opened_page.grid(row=0, column=0, sticky="nsew")

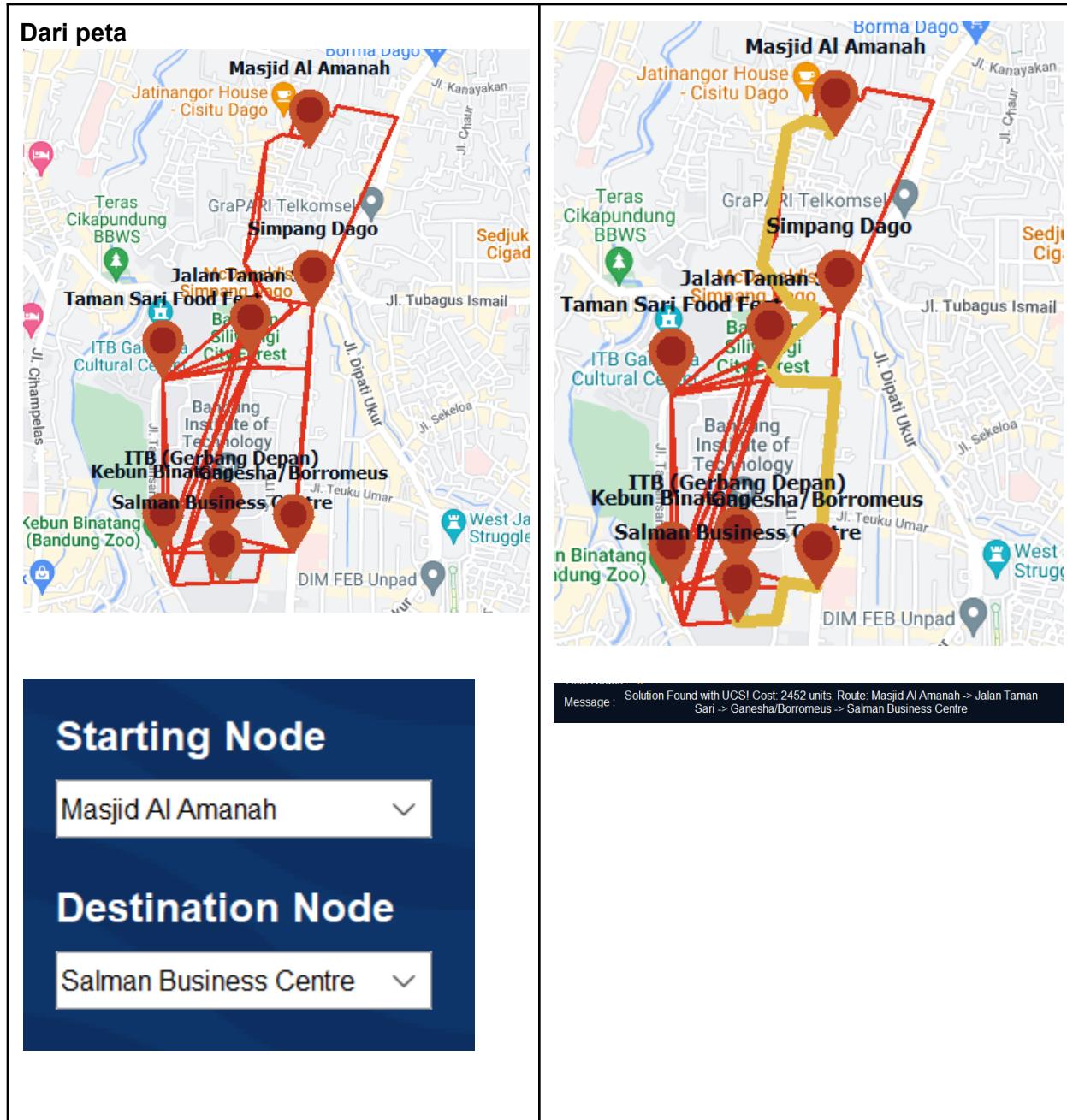
    def refresh_page(self):
        new_page = self.opened_page.__class__(self)
```

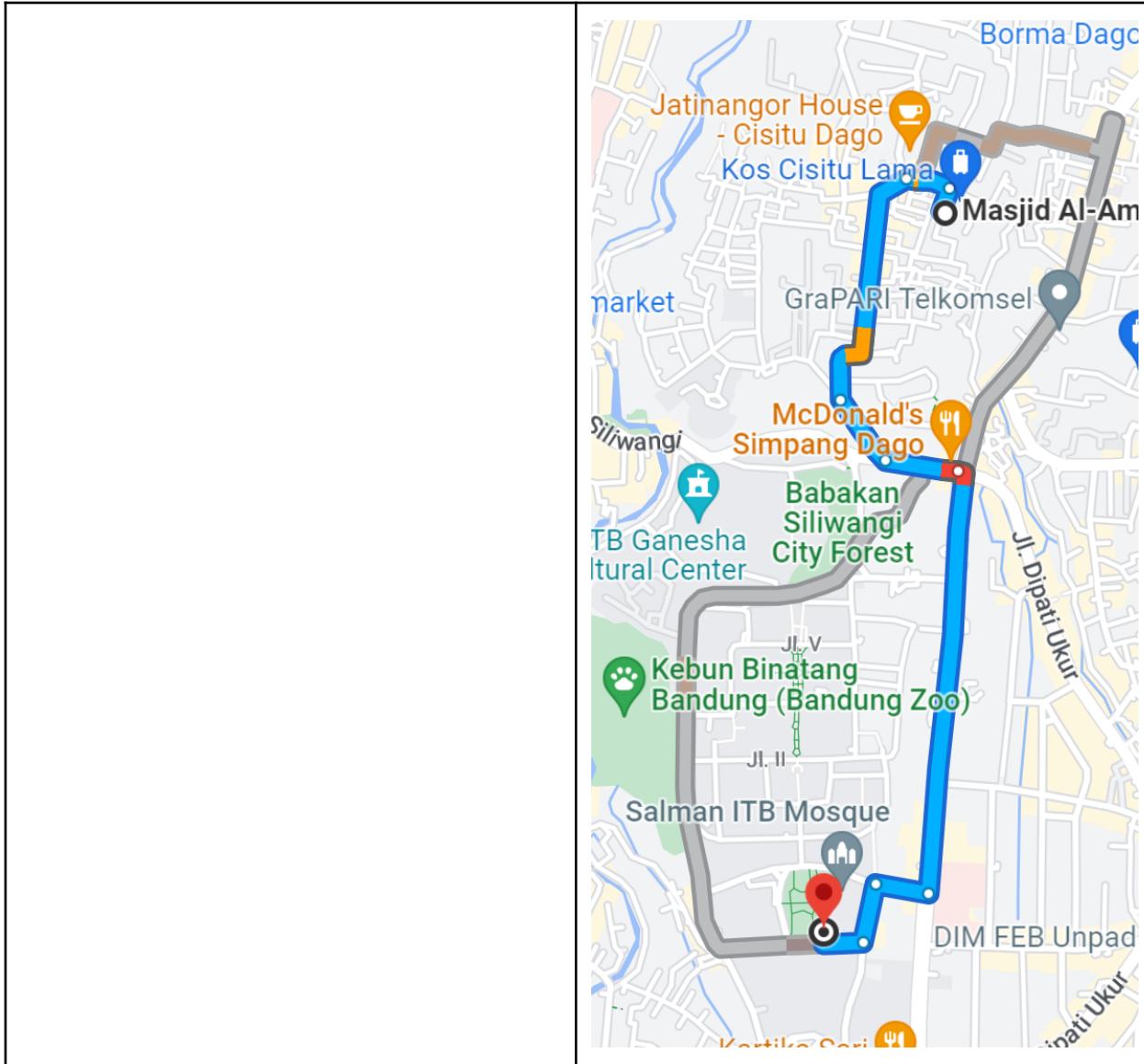
```
self.pages[self.opened_page.__class__.__name__] = new_page
self.open_page(self.opened_page.__class__.__name__)
```

6. Pengujian Program

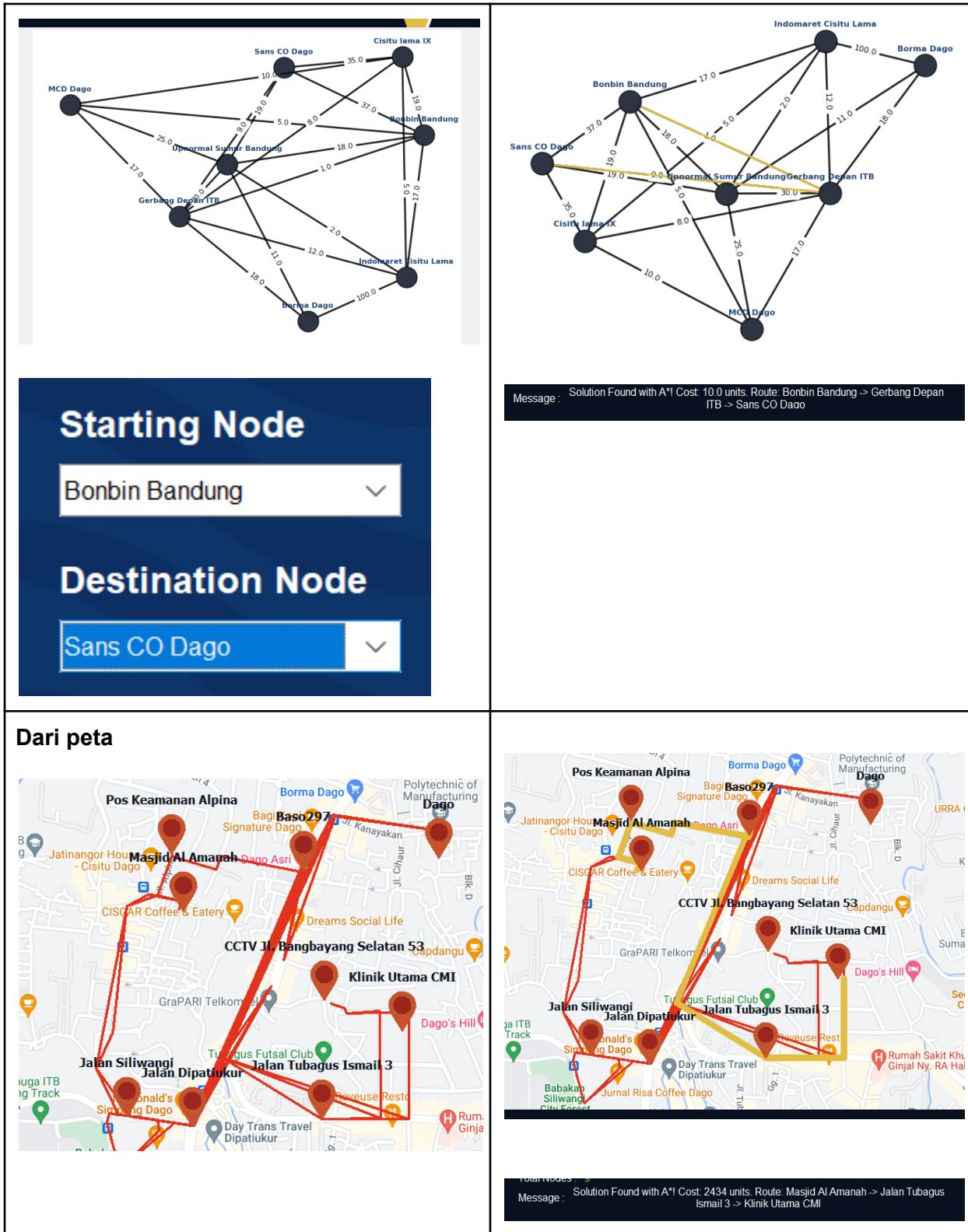
Peta jalan sekitar kampus ITB/Dago/Bandung Utara





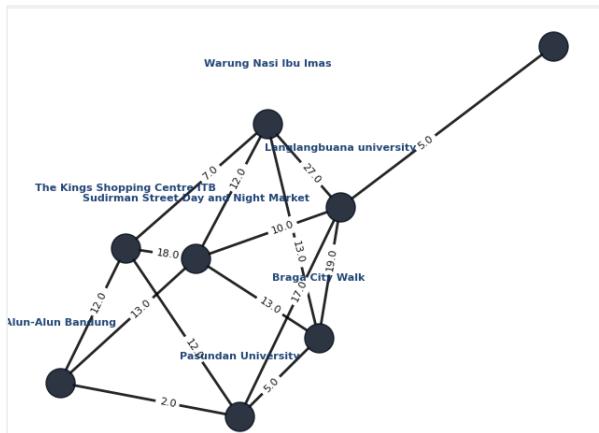
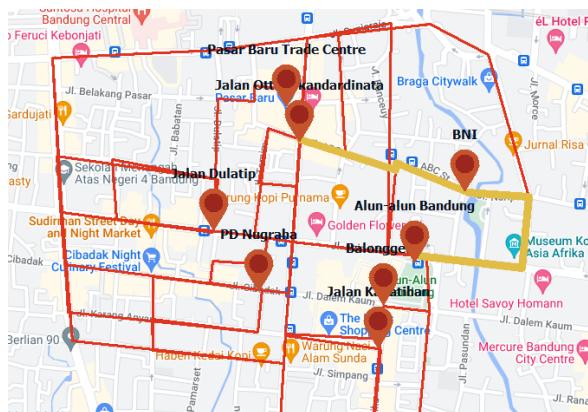


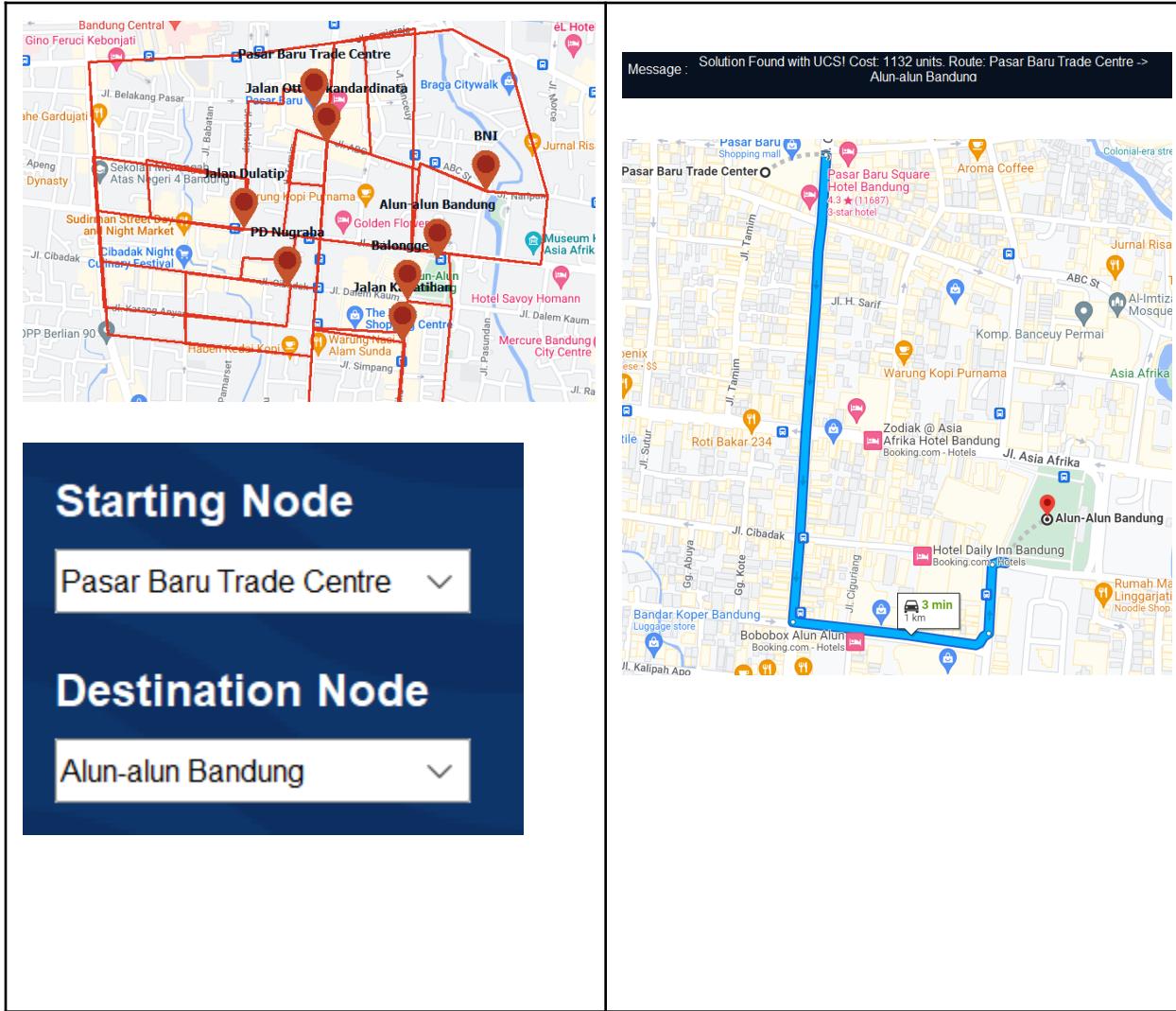
A* Search	
Input	Output
Dari file	



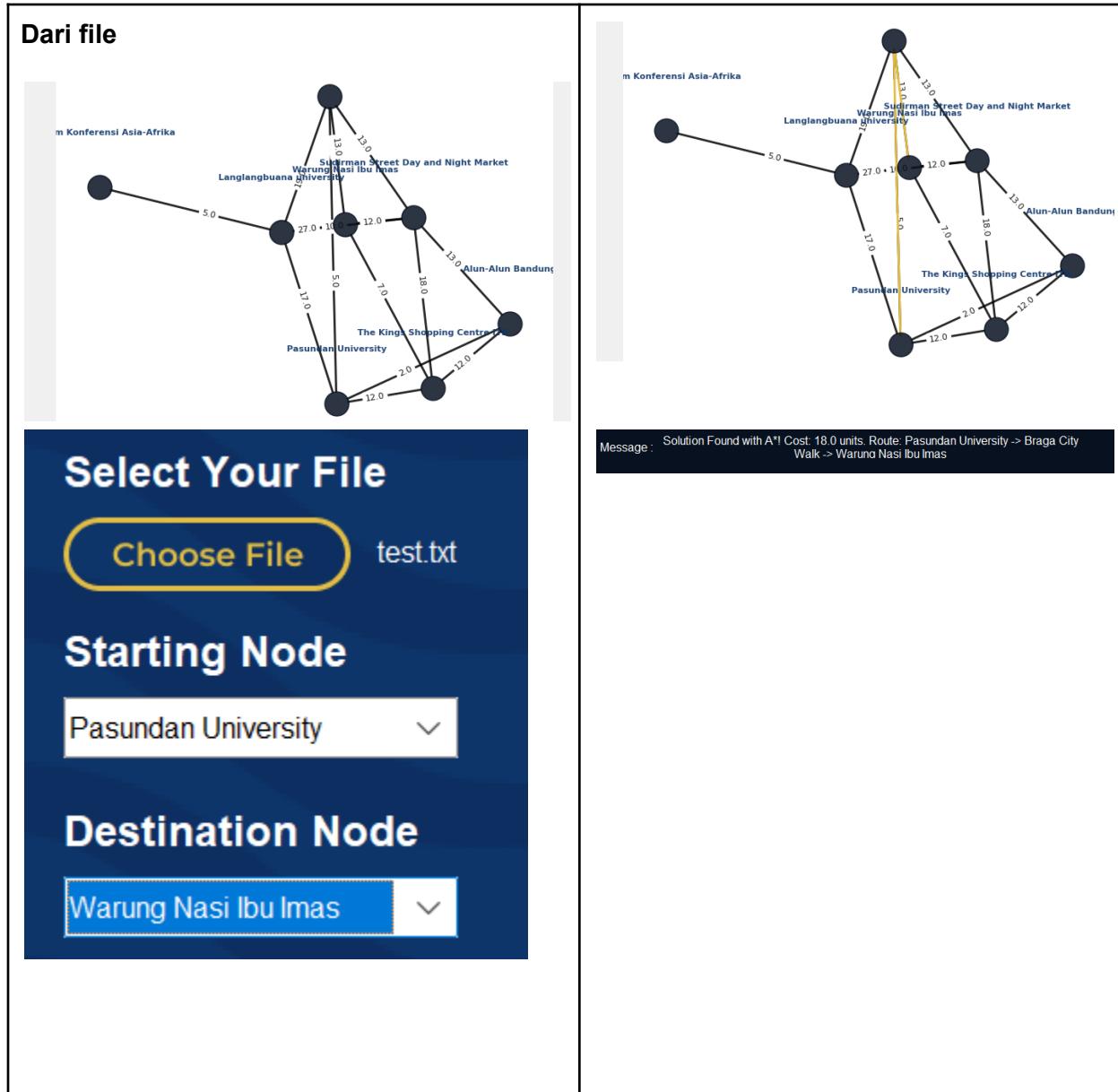
The figure shows a mobile application interface for a navigation service. On the left, a dark blue sidebar displays the "Starting Node" as "Masjid Al Amanah" and the "Destination Node" as "Klinik Utama CMI". On the right, a map view shows a blue line representing the route. The route starts at "Masjid Al-Amanah" and ends at "Klinik Utama CMI". Key waypoints along the route include "GraPARI Telkomsel", "Tubagus Futsal Club", "Reveuse Resto", "Day Trans Travel", "McDonald's Simpang Dago", "Rum Ginja", "Dago's Hill", "Capdangu", "Dreams Social Life", "Dago-Astri", "Kos Cisitu Lama", "Masjid Al-Amanah", "or House situ Dago", and "Bagi Kopi Signature Dago". A callout box on the map indicates a travel time of "8 min" and a distance of "2.3 km".

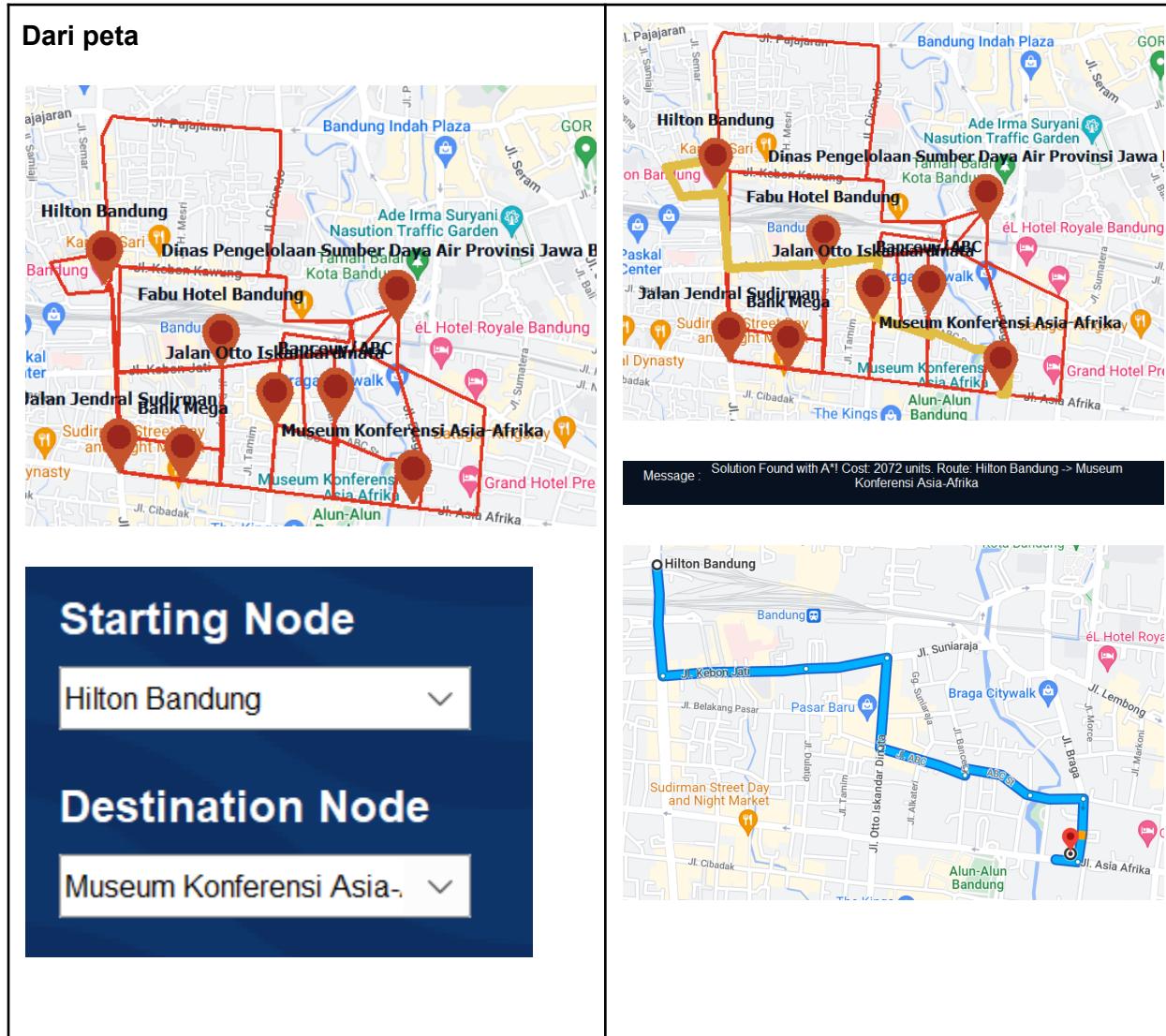
Peta jalan sekitar Alun-alun Bandung

Uniform Cost Search	
Input	Output
Dari file	 <p>The graph shows nodes representing locations and edges representing roads with their respective costs:</p> <ul style="list-style-type: none"> Alun-Alun Bandung Warung Nasi Ibu Imas The Kings Shopping Centre ITB Sudirman Street Day and Night Market Lembangbuana university Braga City Walk Pasundan University <p>Edges and costs:</p> <ul style="list-style-type: none"> Alun-Alun Bandung to Warung Nasi Ibu Imas: 2.0 Alun-Alun Bandung to The Kings Shopping Centre ITB: 12.0 Alun-Alun Bandung to Pasundan University: 13.0 Alun-Alun Bandung to Braga City Walk: 13.0 Warung Nasi Ibu Imas to Lembangbuana university: 19.0 Warung Nasi Ibu Imas to The Kings Shopping Centre ITB: 12.0 Warung Nasi Ibu Imas to Braga City Walk: 17.0 The Kings Shopping Centre ITB to Lembangbuana university: 10.0 The Kings Shopping Centre ITB to Braga City Walk: 18.0 The Kings Shopping Centre ITB to Pasundan University: 13.0 Lembangbuana university to Braga City Walk: 13.0 Lembangbuana university to Pasundan University: 12.0 Braga City Walk to Pasundan University: 10.0 Braga City Walk to Alun-Alun Bandung: 13.0 Pasundan University to Alun-Alun Bandung: 2.0 Pasundan University to Braga City Walk: 13.0 Pasundan University to Lembangbuana university: 19.0 <p>Total Nodes: 7 Message : Solution Found with UCS! Cost: 24.0 units. Route: Museum Konferensi Asia-Afrika -> Lembangbuana university -> Pasundan University -> Alun-Alun Bandung</p>
Starting Node <input type="text" value="Museum Konferensi Asia-Afrika"/>	
Destination Node <input type="text" value="Alun-Alun Bandung"/>	
Dari peta	 <p>A map of the Alun-alun Bandung area with red outlines around major streets. A yellow line highlights the path from the starting point (Museum Konferensi Asia-Afrika) to the destination (Alun-Alun Bandung). Key landmarks shown include Pasar Baru Trade Centre, Jalan Ottokar Mandarinata, Jalan Dukuh Pakis, Alun-alun Bandung, Braga Citywalk, and various hotels and markets.</p>



A* Search	
Input	Output

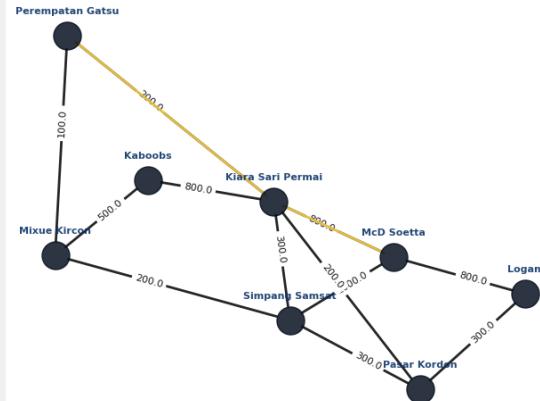




Peta jalan sekitar Buahbatu atau Bandung Selatan

Uniform Cost Search	
Input	Output
Dari file Input file txt	Tampilan Rute (warna emas)

	0	1000	300	300	0	200	0	0	0
✓	1000	0	0	800	800	0	0	0	0
	300	0	0	200	300	0	0	0	0
✓	300	800	200	0	0	0	200	800	
	0	800	300	0	0	0	0	0	0
✓	200	0	0	0	0	0	100	500	
	0	0	0	200	0	100	0	0	0
	0	0	0	800	0	500	0	0	0
Simpang Samsat 10 10									
McD Soetta 27 13									
Pasar Kordon 9 6									
Kiara Sari Permai 12 9									
Logam 15 4									
Mixue Kircon 9 13									
Perempatan Gatsu 8 18									
Kaboobs 9 14									



Rute dan cost nya

Solution Found with UCS! Cost: 1000.0 units. Route: Perempatan Gatsu -> Kiara Sari Permai -> McD Soetta

Lokasi awal dan akhir

Starting Node

Perempatan Gatsu

Destination Node

McD Soetta

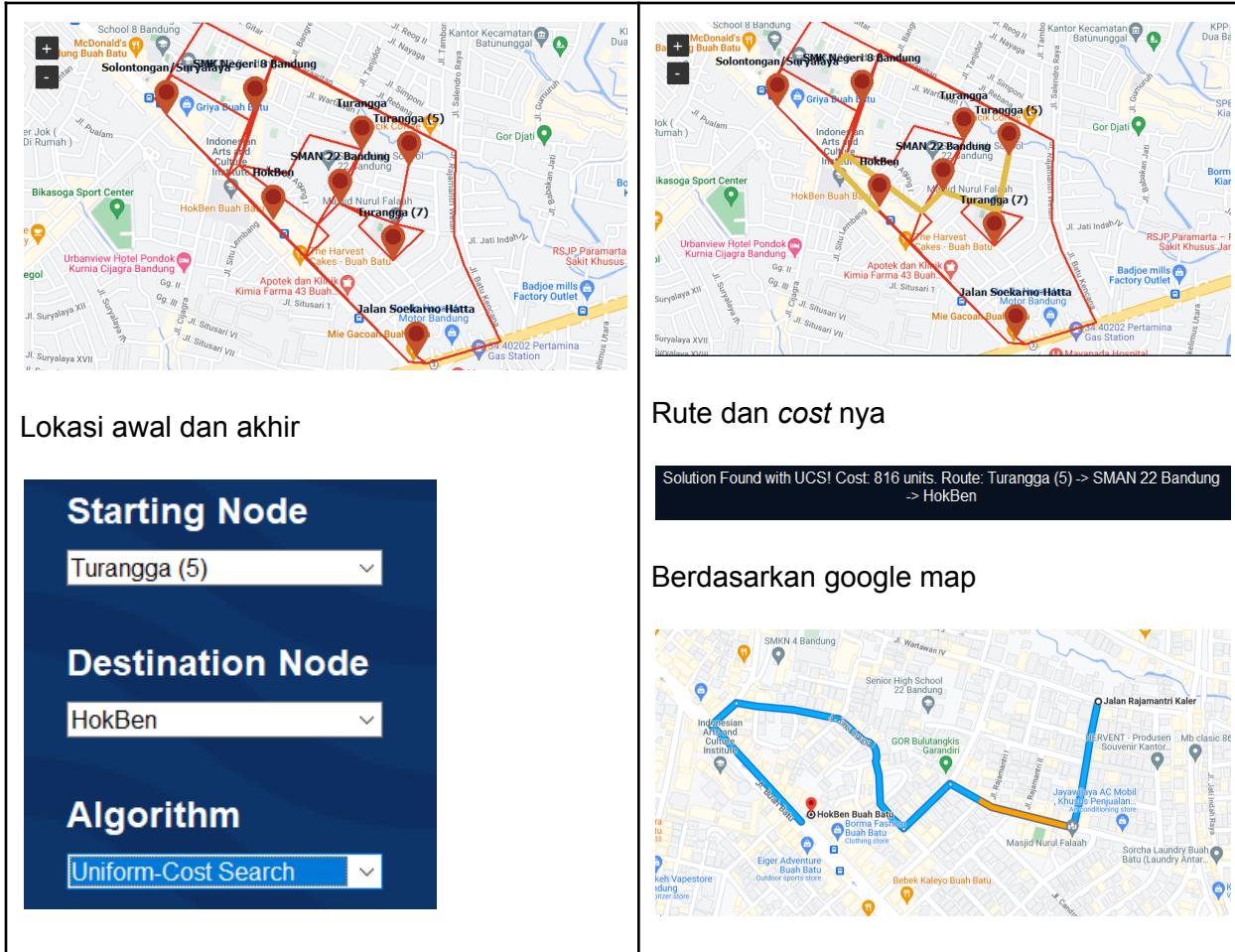
Algorithm

Uniform-Cost Search

Dari peta

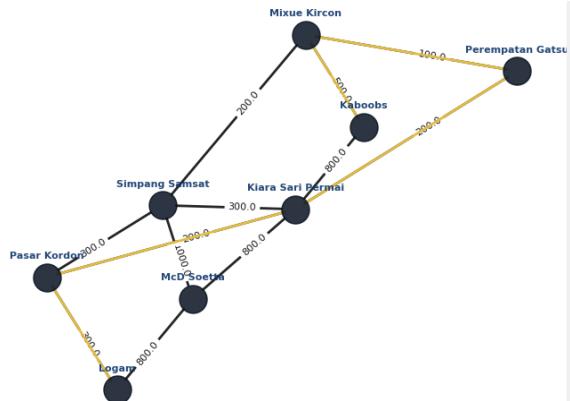
Pilihan Lokasi

Tampilan Rute (warna emas)



A* Search	
Input	Output
Dari file Input file txt	Tampilan Rute (warna emas)

	0	1000	300	300	0	200	0	0	0
✓	1000	0	0	800	800	0	0	0	0
	300	0	0	200	300	0	0	0	0
✓	300	800	200	0	0	0	200	800	0
	0	800	300	0	0	0	0	0	0
✓	200	0	0	0	0	0	100	500	0
	0	0	0	200	0	100	0	0	0
	0	0	0	800	0	500	0	0	0
Simpang Samsat 10 10									
McD Soetta 27 13									
Pasar Kordon 9 6									
Kiara Sari Permai 12 9									
Logam 15 4									
Mixue Kircon 9 13									
Perempatan Gatsu 8 18									
Kaboobs 9 14									



Rute dan cost nya

Solution Found with A*! Cost: 1300.0 units. Route: Kaboobs -> Mixue Kircon -> Peremoatan Gatsu -> Kiara Sari Permai -> Pasar Kordon -> Logam

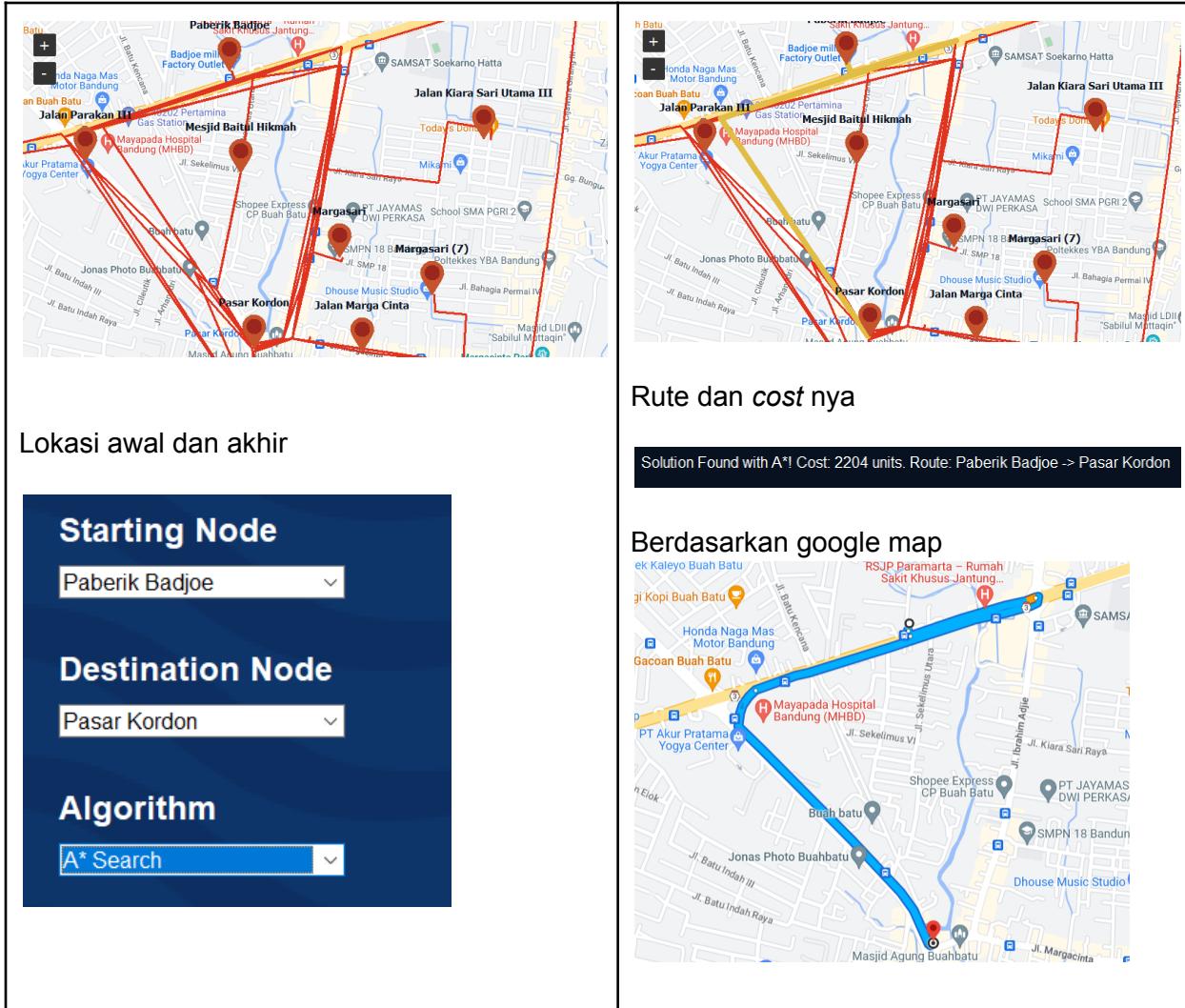
Lokasi awal dan akhir

Starting Node
Kaboobs
Destination Node
Logam
Algorithm
A* Search

Dari peta

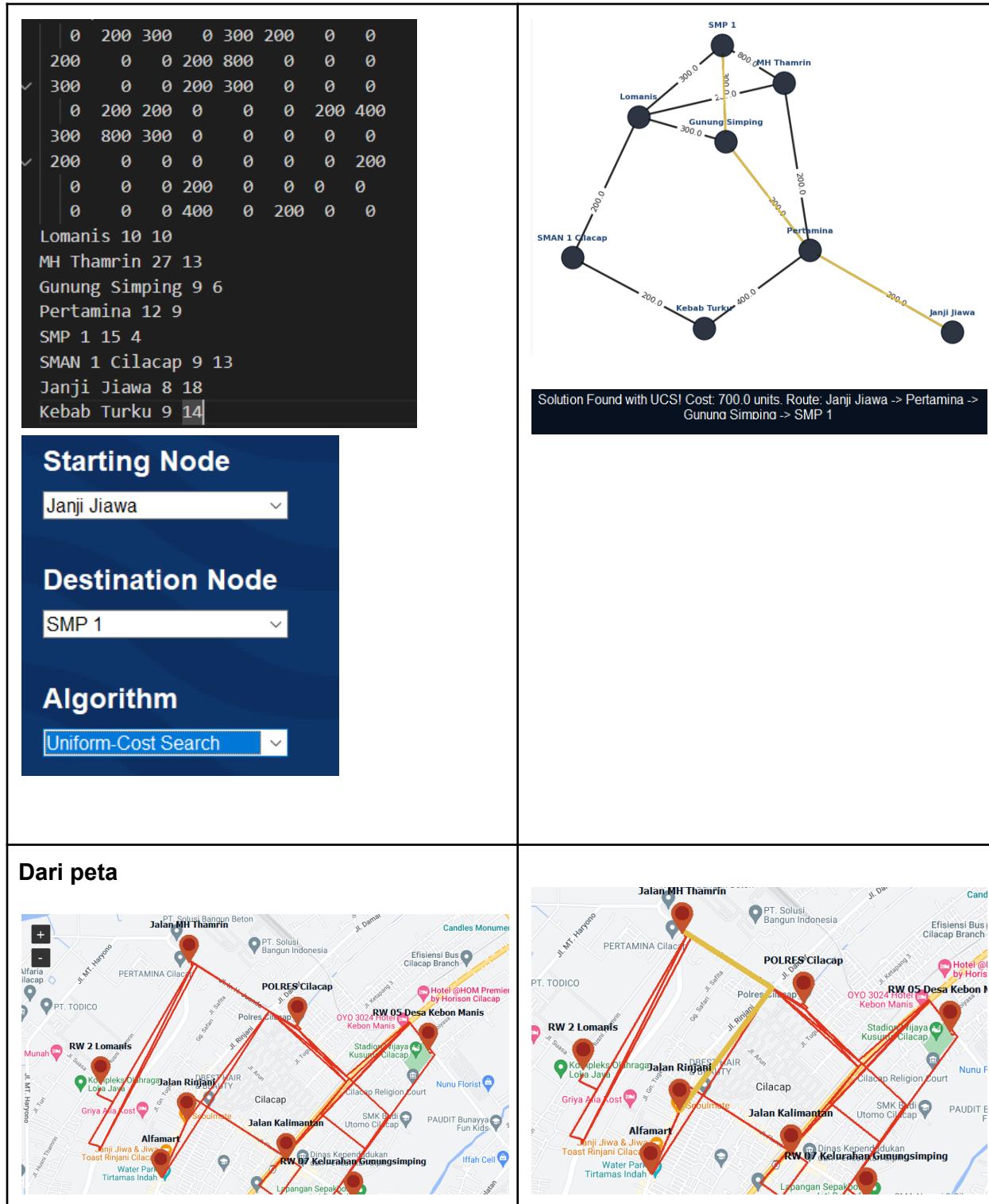
Pilihan Lokasi

Tampilan Rute (warna emas)



Peta jalan sebuah kawasan di kota asalmu (Cilacap, Jawa Tengah)

Uniform Cost Search	
Input	Output
Dari file	



Starting Node

Destination Node

Algorithm

Dari peta

Starting Node Jalan Rinjani	
Destination Node Jalan MH Thamrin	
Algorithm Uniform-Cost Search	

A* Search	
Input	Output
Dari file <pre>0 200 300 0 300 200 0 0 200 0 0 200 800 0 0 0 300 0 0 200 300 0 0 0 0 200 200 0 0 0 200 400 300 800 300 0 0 0 0 0 200 0 0 0 0 0 0 200 0 0 0 200 0 0 0 0 0 0 0 400 0 200 0 0 Lomanis 10 10 MH Thamrin 27 13 Gunung Simping 9 6 Pertamina 12 9 SMP 1 15 4 SMAN 1 Cilacap 9 13 Janji Jiawa 8 18 Kebab Turku 9 14</pre>	<p>The diagram illustrates an A* search graph with nodes represented by black circles and edges by lines with numerical weights. The nodes are labeled: Kebab Turku, Lomanis, Pertamina, Gunung Simping, MH Thamrin, SMP 1, Janji Jiawa, and Jalan Rinjani (the starting node). The edges and their weights are: Jalan Rinjani to Pertamina (200.0), Pertamina to Kebab Turku (200.0), Pertamina to Gunung Simping (200.0), Gunung Simping to Lomanis (200.0), Gunung Simping to MH Thamrin (200.0), Lomanis to MH Thamrin (200.0), and MH Thamrin to SMP 1 (800.0). A yellow line highlights the path from Pertamina to Kebab Turku.</p> <p>Solution Found with A* Cost: 400.0 units. Route: Pertamina -> Kebab Turku</p>

The figure displays a navigation application interface with two main sections: a configuration section at the top and two maps below it.

Configuration Section:

- Starting Node:** Pertamina
- Destination Node:** Kebab Turku
- Algorithm:** A* Search

Map 1 (Left): This map shows a local area with several locations marked by red dots and labeled in blue text. The locations include: DP UPR Cilacap, PT Sumber Alfaria Trijaya Tbk - Cilacap, SDN Lomanis 02, Kos Bu M., Puskesmas Pembantu, Pertamina, PT Bima Karya Kaloka, Impian Olahraga Zoka Jaya, Griya Alia Kost, Alfamart, Soulmate, Janji Jiwa & Jiwa Toast Rinjani Cilacap, Water Park, and DBEST HAIR & BEAUTY. A red line indicates a path from Pertamina to SDN Lomanis 02.

Map 2 (Right): This map shows a larger area with more detailed street names and additional locations. The same starting and destination nodes are indicated. The path from Pertamina to SDN Lomanis 02 is shown in red, while the return path from SDN Lomanis 02 to Pertamina is highlighted in yellow. Other visible locations include: 3ft Outdoor, PT Sumber Alfaria Trijaya Tbk - Cilacap, PT TODICO, Dinas PUPR Cilacap, Kos Bu M., Puskesmas Pembantu, Pertamina, SDN Lomanis 02, PT Bima Karya Kaloka, Impian Olahraga Zoka Jaya, Griya Alia Kost, Alfamart, Soulmate, Janji Jiwa & Jiwa Toast Rinjani Cilacap, Water Park, DBEST HAIR & BEAUTY, Polres Cilacap, and PT Sumber Alfaria Trijaya Tbk - Cilacap. A text box at the bottom right of Map 2 states: "Solution Found with A*! Cost: 3396 units. Route: Jalan MH Thamrin -> SDN Lomanis 02 -> Pertamina".

Bottom Configuration Section:

- Starting Node:** Jalan MH Thamrin
- Destination Node:** Pertamina
- Algorithm:** A* Search

7. Penutup

7.1. Kesimpulan

Algoritma UCS dan A* adalah algoritma yang dapat menjamin hasil lintasan yang minimum, selama batasan-batasannya dijaga. Secara umum, algoritma A* lebih efisien dibandingkan UCS, karena A* mempertimbangkan heuristik tambahan yang dapat memandu pencarian ke arah node tujuan dengan lebih cepat.

7.2. Komentar

Tugas kecil kali ini cukup berat karena harus membuat GUI, sehingga beban lebih berat dari biasanya, mendekati beban tugas besar. Akan tetapi, dari spesifikasi tugas lumayan seru.

Spesifikasi bonus juga asik dan menantang, karena jadi belajar hal baru, yaitu API. Apalagi, API yang digunakan adalah API google, walaupun harus mengisi data kartu debit terlebih dahulu.

8. Lampiran

Link Repository Github

<https://github.com/moonawar/RouteFinder.git>

Tabel Kelengkapan

1	Program dapat menerima input graf	✓
2	Program dapat menghitung lintasan terpendek dengan UCS	✓
3	Program dapat menghitung lintasan terpendek dengan A*	✓
4	Program dapat menampilkan lintasan terpendek serta jaraknya	✓
5	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓