

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

*Diajukan sebagai pemenuhan tugas kecil III.*



Oleh:

1. 13521111 - Tabitha Permalla
2. 13521133 - Cetta Reswara Parahita

Dosen Pengampu : Dr. Ir. Rinaldi, M.T

IF2211 - Strategi Algoritma

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

**2022**

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB 1 DESKRIPSI MASALAH</b>	<b>3</b>
<b>BAB 2 TEORI DASAR</b>	<b>4</b>
<b>2.1. Route/ Path Planning</b>	<b>4</b>
<b>2.1.1. Algoritma UCS</b>	<b>4</b>
<b>2.1.2. Algoritma A*</b>	<b>5</b>
<b>2.2. Peta dan Graf</b>	<b>7</b>
<b>BAB 3 ALGORITMA A* DAN UCS</b>	<b>8</b>
<b>BAB 4 IMPLEMENTASI</b>	<b>11</b>
<b>BAB 5 PENGUJIAN</b>	<b>28</b>
<b>5.1. Validasi Input</b>	<b>28</b>
<b>5.3.1. Validasi file txt (jumlah simpul harus &gt;= 8)</b>	<b>28</b>
<b>5.3.2. Validasi nama simpul mulai dan simpul akhir</b>	<b>29</b>
<b>5.2. Pengujian Test Case</b>	<b>30</b>
<b>5.3.1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara</b>	<b>30</b>
<b>5.3.2. Peta jalan sekitar Alun-alun Bandung</b>	<b>31</b>
<b>5.3.3. Peta jalan sekitar Buahbatu atau Bandung Selatan</b>	<b>32</b>
<b>5.3.4. Peta jalan sebuah kawasan di kota asalmu</b>	<b>33</b>
<b>5.3. Pengujian Lainnya</b>	<b>34</b>
<b>5.3.1. Tidak Ada Jalur</b>	<b>34</b>
<b>BAB 6 PENUTUP</b>	<b>35</b>
<b>LAMPIRAN DAN DAFTAR PUSTAKA</b>	<b>36</b>

## **BAB 1**

### **DESKRIPSI MASALAH**

Penggunaan peta digital telah menjadi kebutuhan umum masyarakat modern untuk mempermudah mobilisasi, baik dengan menggunakan kendaraan pribadi maupun kendaraan digital. Peta digital kerap dilengkapi dengan fitur pencarian rute terdekat untuk mencapai titik tujuan untuk mempermudah penggunanya. Dalam pencarian rute terdekat ini, terdapat berbagai macam algoritma yang dapat digunakan untuk optimalisasi pencarian rute; dua diantaranya yang paling banyak digunakan adalah UCS (Uniform cost search) dan A\* (A star).

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, kedua algoritma ini digunakan untuk menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung dengan ruas-ruas jalan di peta dibentuk menjadi sebuah 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 antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

Beberapa jalan yang perlu ditinjau antara lain:

1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara
2. Peta jalan sekitar Alun-alun Bandung
3. Peta jalan sekitar Buahbatu atau Bandung Selatan
4. Peta jalan sebuah kawasan di kota asalmu

Peninjauan ini dilakukan pada sebuah program yang memiliki spesifikasi sebagai berikut:

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggan berbobot) dengan jumlah simpul minimal 8 buah.
2. Program menampilkan peta/ graf.
3. Program menerima input simpul asal dan simpul tujuan.
4. Program menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Program menampilkan antarmuka pada sistem GUI.

## BAB 2

### TEORI DASAR

#### 2.1. Route/ Path Planning

Route/ Path planning adalah proses pencarian komprehensif untuk menentukan metode yang paling efisien untuk mencapai suatu tujuan. Beberapa jenis route/ path planning antara lain mencakup

1. Dynamic route planning yang membuat berbagai kemungkinan rute berdasarkan faktor eksternal yang dapat mempengaruhi efisiensi rute.
2. Multi-stop route planning yang membuat rute seefisien mungkin dapat melewati setiap titik yang telah direncanakan.
3. Open path routing berfokus dalam menentukan rute yang paling cepat dijangkau dari berbagai tujuan yang telah direncanakan.
4. Closest path routing berfokus untuk menentukan rute dengan jarak terdekat dari titik awal.

Pada tugas besar ini, route/ path planning yang akan diselesaikan berfokus pada penyelesaian *closest path routing*. Permodelan yang digunakan menggunakan konsep graf  $G = (V, E)$  dengan  $n$  nodes dan  $m = \Theta(n)$  edges yang tiap edge nya memiliki berat/ cost nonnegatif  $w(u,v)$ . *Closest path routing* akan melakukan pengecekan antara titik asal  $s$  menuju titik tujuan  $g$  dengan menentukan berat/ cost paling minimal  $d(s, g)$ .

Beberapa algoritma yang biasa digunakan dalam penentuan route/ path planning antara lain adalah UCS/ Uniform Cost Search (Dijkstra's Algorithm), Priority Queues, Greedy Best First Search, Bidirectional Search, dan Geometric Goal Directed Search (A\*/ A star). Di antara beberapa algoritma ini, implementasi pada tugas ini akan berfokus pada algoritma UCS dan algoritma A\*.

##### 2.1.1. Algoritma UCS

*Uniform-Cost Search* merupakan algoritma pencarian tanpa informasi (uninformed search) yang menggunakan biaya kumulatif terendah untuk menemukan jalur dari node sumber ke node tujuan.

Algoritma ini beroperasi di sekitar ruang pencarian berbobot terarah untuk berpindah dari node awal ke salah satu node akhir dengan biaya akumulasi minimum. Algoritma Uniform-Cost Search masuk dalam algoritma pencarian uninformed search atau blind search karena bekerja dengan cara brute force, yaitu tidak mempertimbangkan keadaan node atau ruang pencarian. Algoritma ini umumnya digunakan untuk menemukan jalur dengan biaya kumulatif terendah dalam graph berbobot di mana node diperluas sesuai dengan biaya traversalnya dari node root.

Uniform-Cost Search juga dapat disebut sebagai varian dari algoritma Dijkstra. Hal ini karena pada uniform cost search, alih-alih memasukkan semua simpul ke dalam antrian prioritas (priority queue), kita hanya menyisipkan node sumber, lalu memasukkan satu per satu bila diperlukan.

Cara kerja algoritma uniform-cost search secara singkat sebagai berikut:

1. Masukkan node root ke dalam priority queue
2. Ulangi langkah berikut saat antrian (queue) tidak kosong:
  - a. Hapus elemen dengan prioritas tertinggi
  - b. Jika node yang dihapus adalah node tujuan, cetak total biaya (cost) dan hentikan algoritma
  - c. Jika tidak, enqueue semua child dari node saat ini ke priority queue, dengan biaya kumulatifnya dari root sebagai prioritas

Di sini node root adalah node awal untuk jalur pencarian, dan priority queue tetap untuk mempertahankan jalur dengan biaya paling rendah untuk dipilih pada traversal berikutnya. Jika 2 jalur memiliki biaya traversal yang sama, node diurutkan berdasarkan abjad. Time complexity pada algoritma uniform cost search dapat dirumuskan:

$$O(b(1 + C / \varepsilon))$$

Dengan  $b$  adalah branching factor,  $C$  adalah biaya optimal,  $\varepsilon$  adalah biaya setiap langkah.

### 2.1.2. Algoritma A\*

Algoritma A\* (A Star) adalah algoritma pencarian yang digunakan untuk menemukan jalur terpendek antara titik awal dan akhir. Algoritma ini digunakan untuk penjelajahan peta guna menemukan jalur terpendek yang akan diambil.

A\* dirancang sebagai masalah penjelajahan graph (graph traversal), untuk membantu robot agar dapat menemukan arahnya sendiri. A\* saat ini masih tetap menjadi algoritma yang sangat populer untuk graph traversal. Algoritma A\* mencari jalur yang lebih pendek terlebih dahulu, sehingga menjadikannya algoritma yang optimal dan lengkap. Algoritma yang optimal akan menemukan hasil yang paling murah dalam hal biaya untuk suatu masalah, sedangkan algoritma yang lengkap menemukan semua hasil yang mungkin dari suatu masalah.

Aspek lain yang membuat A\* begitu powerful adalah penggunaan graph berbobot dalam penerapannya. Graph berbobot menggunakan angka untuk mewakili biaya pengambilan setiap jalur atau tindakan. Ini berarti bahwa algoritma dapat mengambil jalur dengan biaya paling sedikit, dan menemukan rute terbaik dari segi jarak dan waktu.

Notasi yang dipakai oleh algoritma A\* adalah sebagai berikut:

$$f(n) = g(n) + h(n)$$

Dengan  $f(n)$  = biaya estimasi terendah,  $g(n)$  = biaya dari node awal ke node n,  $h(n)$  = perkiraan biaya dari node n ke node akhir. Cara kerja algoritma A\* search secara singkat sebagai berikut:

1. Inisialisasi OPEN LIST
2. Letakkan simpul awal pada OPEN LIST
3. Inisialisasi CLOSE LIST
4. Ikuti langkah-langkah berikut sampai OPEN LIST tidak kosong:
  5. Temukan simpul dengan  $f$  terkecil pada OPEN LIST dan beri nama "Q".
  6. Hapus Q dari OPEN LIST.
  7. Generate delapan turunan Q dan tetapkan Q sebagai induknya.
  8. Untuk setiap keturunan:
    - a. Jika menemukan penerus adalah tujuannya, pencarian dihentikan
    - b. Jika tidak, hitung  $g$  dan  $h$  untuk penerusnya.  
 $penerus.g = q.g + \text{jarak yang dihitung antara penerus dan } q$ .  
 $\text{suksesor.h} = \text{jarak terhitung antara suksesor dan tujuan}$ .  
 $penerus.f = penerus.g + penerus.h$
    - c. Lewati penerus ini jika node dalam daftar OPEN dengan lokasi yang sama tetapi nilai  $f$  lebih rendah dari pengantinya.
    - d. Lewati penerusnya jika ada simpul dalam CLOSE LIST dengan posisi yang sama dengan penerusnya tetapi nilai  $f$  lebih rendah; jika tidak, tambahkan simpul ke ujung OPEN LIST (untuk loop).
  9. Push Q ke dalam CLOSE LIST dan akhiri loop sementara.

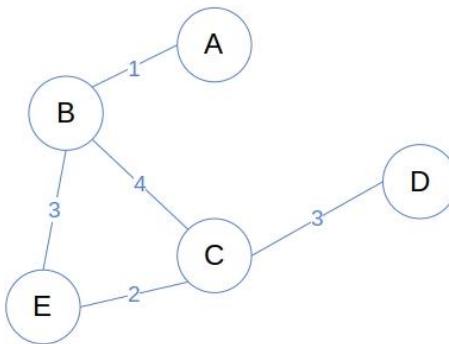
## 2.2. Peta dan Graf

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan.

Simpul pada graph disebut dengan verteks ( $V$ ), sedangkan sisi yang menghubungkan antar verteks disebut edge ( $E$ ). Pasangan  $(x,y)$  disebut sebagai edge, yang menyatakan bahwa simpul  $x$  terhubung ke simpul  $y$ .

Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk) dan penentuan jarak dan titik pada peta.

Penentuan jarak dan titik pada peta tergolong sebagai weighted graph. Weighted graph adalah jenis graph yang cabangnya diberi label bobot berupa bilangan numerik. Pemberian label bobot pada edge biasanya digunakan untuk memudahkan algoritma dalam menyelesaikan masalah.



**Gambar 2.1 Weighted Graph**

Sumber: [baeldung. com](http://baeldung.com)

Contoh implementasinya misalkan kita ingin menyelesaikan masalah dalam mencari rute terpendek dari lokasi A ke lokasi D, namun kita juga dituntut untuk mempertimbangkan kepadatan lalu lintas, panjang jalan dll. Untuk masalah seperti ini, kita bisa mengasosiasikan sebuah edge  $e$  dengan bobot  $w(e)$  berupa bilangan riil. Nilai bobot ini bisa apa saja yang relevan untuk masalah yang dihadapi: misalnya jarak, kepadatan, durasi, biaya, probabilitas, dan sebagainya.

## BAB 3

### ALGORITMA A\* DAN UCS

Pada penyelesaian program ini, digunakan algoritma A\* dan UCS. Algoritma A\* dan UCS memanfaatkan fungsi euclidean dalam menentukan jarak antar titik. Fungsi euclidean adalah sebagai berikut  $d_{euclidean} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ . Selain itu, algoritma ini dibuat untuk menyelesaikan pencarian jarak pada objek dengan tipe graph. Objek dengan tipe graph memiliki detail sebagai berikut:

```

class Node
    private ID : String
    private Name : String
    private x : float
    private y : float
    private Neighbors : array of Node

    { Attribut A* }
    private g : float
    private h : float
    private f : float
    private Parent : Node

    { Attribut UCS }
    private Checked : boolean

    public procedure addNeighbor(Node, Node)

end of class

class Edge
    private Source : Node
    private Target : Node

end of class

class Graph
    private n : Integer
    private nodes : array of Node
    private edges : array of Edge

    public procedure addNode(Graph, Node)
    public procedure addEdge(Graph, Edge)

end of class

```

Selanjutnya algoritma A\* memiliki *pseudocode* sebagai berikut:

```
function A* (start, end: Node)
    → solution: array of Node, distance: Integer

{Inisialisasi nilai g, h, f}
start.g ← 0
start.h ← euclidean(start, end)
start.f ← start.g + start.h

{Inisialisasi array queue solusi}
solusi ← start

{Iterasi pencarian path selama solusi masih ditemukan}
while solusi tidak kosong do
    current ← solusi.pop()

    {Cek apakah solusi sudah ditemukan}
    if current adalah end then
        distance ← current.f
        → solusi, distance

    {Jika belum, cek node tetangga}
    iterate neighbor in current.neighbors:
        temp_g ← current.g + euclidean(current, neighbor)

        {Cek apakah solusi saat ini lebih optimal daripada solusi
         tersimpan}
        if temp_g lebih kecil dari g then
            Perbarui g, h, f
            Perbarui parent tetangga
            solusi ← neighbor

    { jika tidak ada solusi }
    → NULL, NULL
```

Dan algoritma UCS memiliki *pseudocode* sebagai berikut:

```
function UCS (start, end: Node)
    → path: array of Node, distance: Integer

{Inisialisasi queue}
queue ← (0, start, [start])

{Iterasi pencarian path selama solusi masih ditemukan}
while queue tidak kosong do
    (distance, current, path) ← queue
```

```
current.checked ← True

{Cek apakah current adalah goal}
if current adalah end then
    → path, distance

{jika tidak, cek tetangga yang belum dicek}
iterate neighbor in current.neighbors:
    if not neighbor.checked then
        new_dist ← distance + euclidean(neighbor, current)
        queue.push((new_dist, neighbor, path.push(neighbor)))

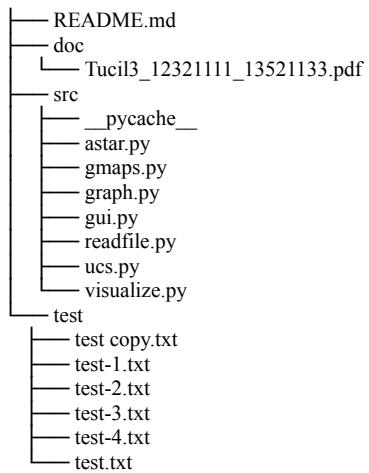
{ jika tidak ada solusi }
→ NULL, NULL
```

Algoritma-algoritma ini selanjutnya akan diimplementasikan dengan menggunakan bahasa pemrograman Python dan library terkait yang dapat dimanfaatkan.

## **BAB 4**

### **IMPLEMENTASI**

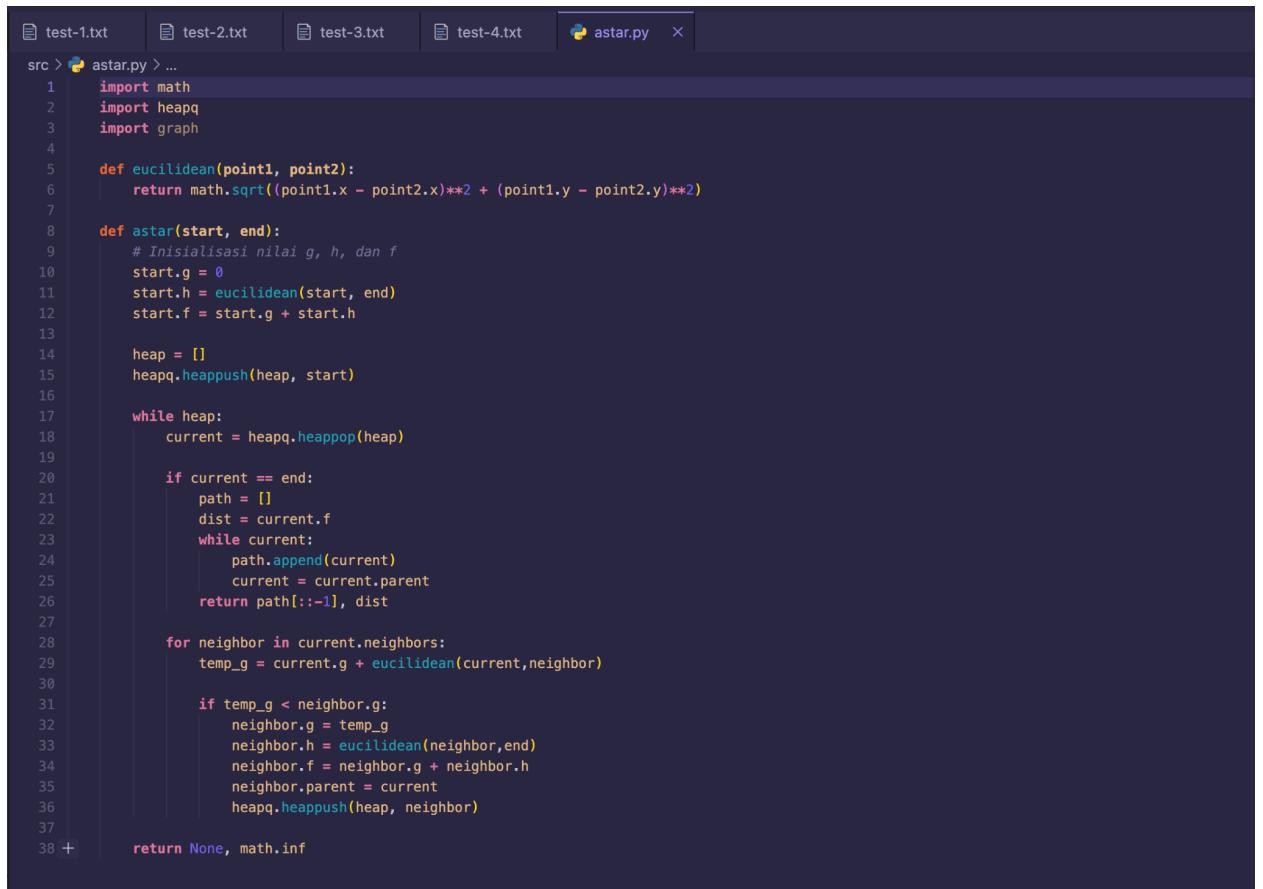
Algoritma-algoritma ini diimplementasikan dengan menggunakan bahasa pemrograman Phyton dan ditampilkan dengan memanfaatkan fitur GUI yang disediakan oleh library tkinter pada Python. Struktur data pada program yang dibuat adalah sebagai berikut:



Pada folder **src** terdapat source code program dengan detail sebagai berikut:

a. **astar.py**

File ini berisi fungsi **astar(start, end)** dan **euclidean(node1, node2)** yang memanfaatkan library **math** dan **heapq** serta melakukan pemanggilan metode pada file **graph.py**.



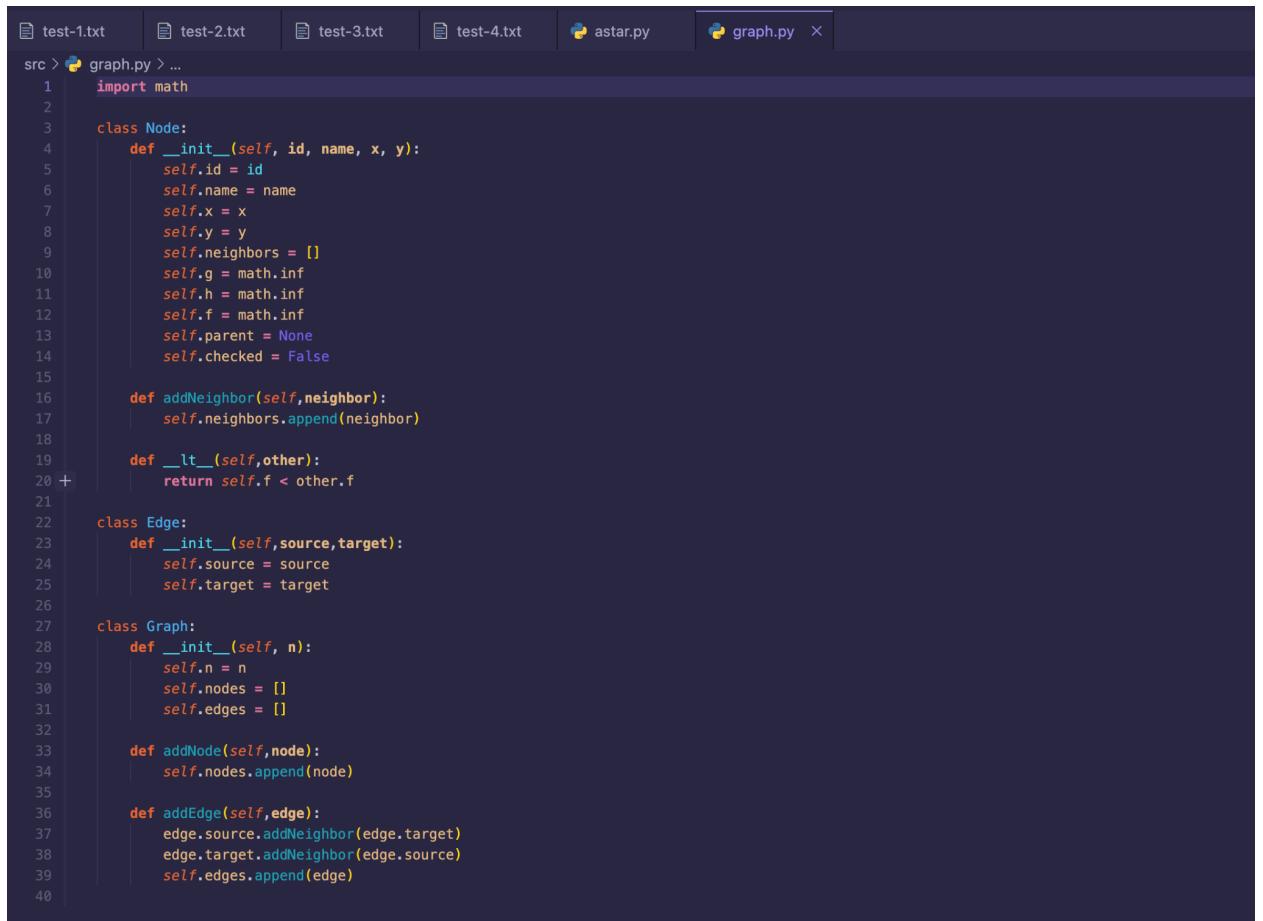
```
test-1.txt test-2.txt test-3.txt test-4.txt astar.py
src > astar.py > ...
1 import math
2 import heapq
3 import graph
4
5 def euclidean(point1, point2):
6     return math.sqrt((point1.x - point2.x)**2 + (point1.y - point2.y)**2)
7
8 def astar(start, end):
9     # Inisialisasi nilai g, h, dan f
10    start.g = 0
11    start.h = euclidean(start, end)
12    start.f = start.g + start.h
13
14    heap = []
15    heapq.heappush(heap, start)
16
17    while heap:
18        current = heapq.heappop(heap)
19
20        if current == end:
21            path = []
22            dist = current.f
23            while current:
24                path.append(current)
25                current = current.parent
26            return path[::-1], dist
27
28        for neighbor in current.neighbors:
29            temp_g = current.g + euclidean(current, neighbor)
30
31            if temp_g < neighbor.g:
32                neighbor.g = temp_g
33                neighbor.h = euclidean(neighbor, end)
34                neighbor.f = neighbor.g + neighbor.h
35                neighbor.parent = current
36                heapq.heappush(heap, neighbor)
37
38    return None, math.inf
```

Gambar 3.1.1

File **astar.py**

b. graph.py

File ini berisi kelas **Node**, **Edge**, dan **Graph** yang memanfaatkan library **math**.



```
src > graph.py > ...
1     import math
2
3     class Node:
4         def __init__(self, id, name, x, y):
5             self.id = id
6             self.name = name
7             self.x = x
8             self.y = y
9             self.neighbors = []
10            self.g = math.inf
11            self.h = math.inf
12            self.f = math.inf
13            self.parent = None
14            self.checked = False
15
16        def addNeighbor(self,neighbor):
17            self.neighbors.append(neighbor)
18
19        def __lt__(self,other):
20            return self.f < other.f
21
22    class Edge:
23        def __init__(self,source,target):
24            self.source = source
25            self.target = target
26
27    class Graph:
28        def __init__(self, n):
29            self.n = n
30            self.nodes = []
31            self.edges = []
32
33        def addNode(self,node):
34            self.nodes.append(node)
35
36        def addEdge(self,edge):
37            edge.source.addNeighbor(edge.target)
38            edge.target.addNeighbor(edge.source)
39            self.edges.append(edge)
40
```

Gambar 3.1.2

File graph.py

c. **gui.py**

File ini berisi kelas **App** yang memanfaatkan library **customtkinter**, **tkinter**, dan **matplotlib**.

Dipanggil pula metode dan kelas yang berada pada file **astar.py**, **ucs.py**, **graph.py**, **visualize.py**, dan **readfile.py** untuk selanjutnya dipanggil dalam program utama **main** yang memanggil kelas **App** dengan prosedur mainloop.

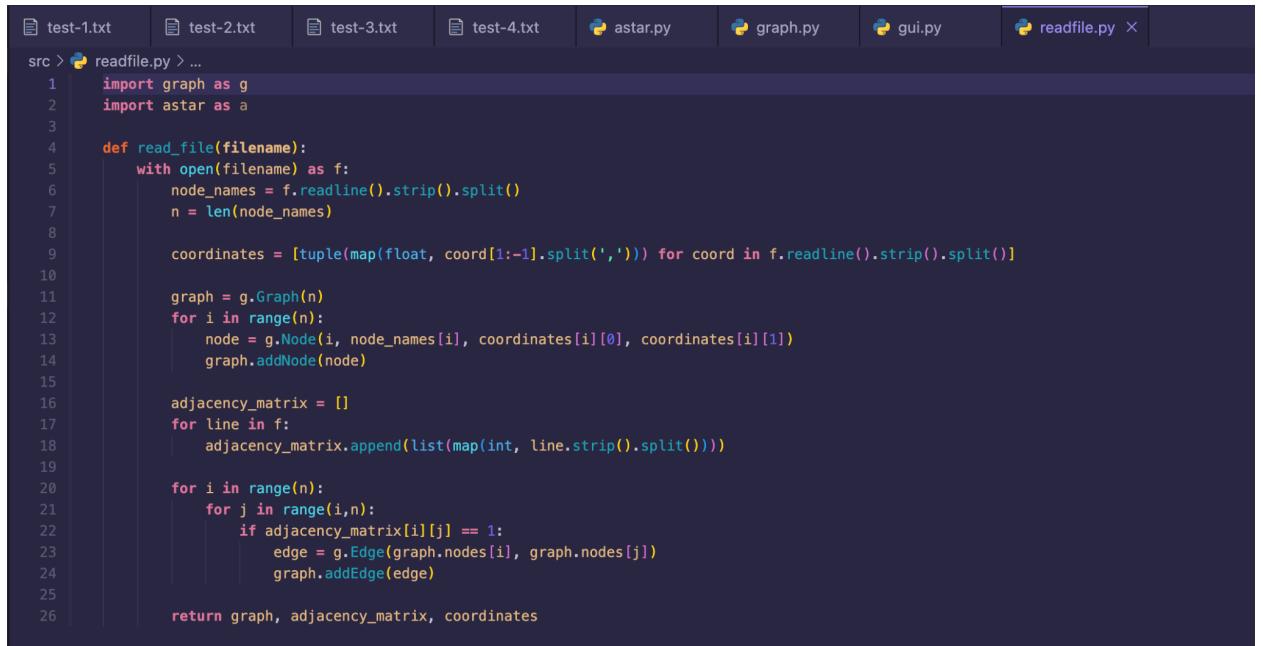
```
1 import customtkinter
2 import tkinter as tk
3 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
4 from matplotlib.figure import Figure
5 import astar
6 import ucs
7 import graph as g
8 import copy
9 import visualize as v
10 import readfile as r
11 from tkinter import filedialog
12
13 class App(customtkinter.CTk):
14     def __init__(self):
15         super().__init__()
16
17         # set background color
18         App.configure(self,fg_color="#C1CEFE")
19         self.geometry("1900x1200")
20         self.title("Shortest Path")
21
22         # top frame
23         frame = customtkinter.CTkFrame(master=self,
24                                         width=1050,
25                                         height=125,
26                                         corner_radius=10,
27                                         fg_color = "#AODDF",
28                                         border_color="white",
29                                         border_width=1)
30         frame.pack(padx=10, pady=10)
31
32         text_var = tk.StringVar(value="Shortest Path with A* and UCS")
33         label = customtkinter.CTkLabel(master=frame,
34                                         textvariable=text_var,
35                                         font=("MV Boli",33),
36                                         corner_radius=8)
37         label.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
38
39
40         # Frames
41         frame1 = customtkinter.CTkFrame(self, fg_color="#7189FF", width=300)
42         frame1.pack(fill="both",side="left", padx=25, pady=80)
43
```

**Gambar 3.1.3**

**File gui.py**

d. **readfile.py**

Pada file ini dibuat metode **read\_file(filename)** yang akan mengkonversi txt file menjadi graf, adjacency\_matrix, dan titik-titik koordinat.



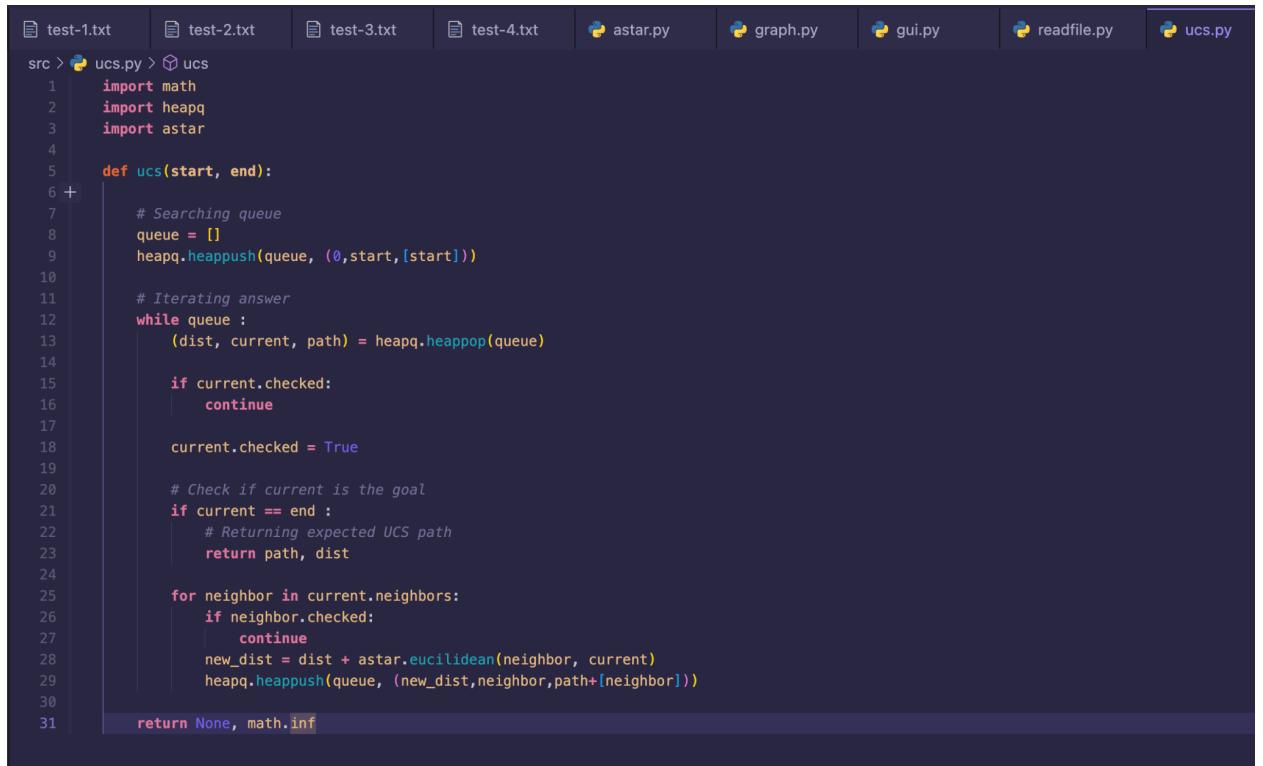
```
src > 🐍 readfile.py > ...
1     import graph as g
2     import astar as a
3
4     def read_file(filename):
5         with open(filename) as f:
6             node_names = f.readline().strip().split()
7             n = len(node_names)
8
9             coordinates = [tuple(map(float, coord[1:-1].split(','))) for coord in f.readline().strip().split()]
10
11            graph = g.Graph(n)
12            for i in range(n):
13                node = g.Node(i, node_names[i], coordinates[i][0], coordinates[i][1])
14                graph.addNode(node)
15
16            adjacency_matrix = []
17            for line in f:
18                adjacency_matrix.append(list(map(int, line.strip().split())))
19
20            for i in range(n):
21                for j in range(i,n):
22                    if adjacency_matrix[i][j] == 1:
23                        edge = g.Edge(graph.nodes[i], graph.nodes[j])
24                        graph.addEdge(edge)
25
26        return graph, adjacency_matrix, coordinates
```

**Gambar 3.1.4**

**File readfile.py**

e. ucs.py

File ini berisi fungsi **ucs(start, end)** yang memanfaatkan library **math** dan **heapq** serta metode euclidean yang terdapat pada file **astar.py**.



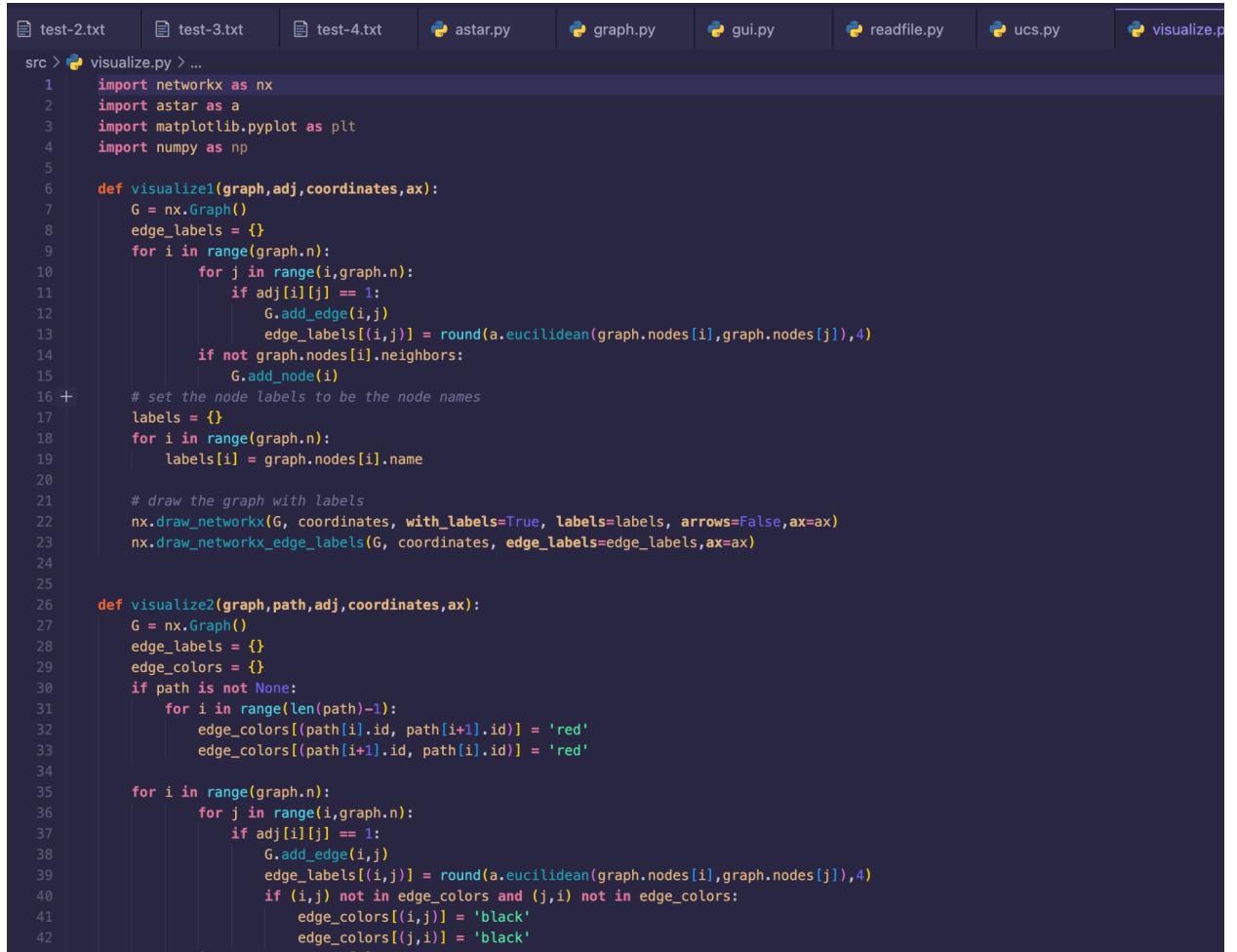
```
src > ucs.py > ucs
1 import math
2 import heapq
3 import astar
4
5 def ucs(start, end):
6     # Searching queue
7     queue = []
8     heapq.heappush(queue, (0,start,[start]))
9
10    # Iterating answer
11    while queue :
12        (dist, current, path) = heapq.heappop(queue)
13
14        if current.checked:
15            continue
16
17        current.checked = True
18
19        # Check if current is the goal
20        if current == end :
21            # Returning expected UCS path
22            return path, dist
23
24        for neighbor in current.neighbors:
25            if neighbor.checked:
26                continue
27            new_dist = dist + astar.euclidean(neighbor, current)
28            heapq.heappush(queue, (new_dist,neighbor,path+[neighbor]))
29
30
31 return None, math.inf
```

Gambar 3.1.5

File ucs.py

#### f. visualize.py

File ini berisi fungsi **visualize1(graph,adj,coordinates,ax)** dan **visualize2(graph,adj,coordinates,ax)** sebagai variansi visualisasi dengan label jarak dan dengan pewarnaan rute. Visualisasi memanfaatkan library **networkx**, **matplotlib**, dan **numpy** serta metode euclidean yang terdapat pada file **astar.py**.



```

1  import networkx as nx
2  import astar as a
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  def visualize1(graph,adj,coordinates,ax):
7      G = nx.Graph()
8      edge_labels = {}
9      for i in range(graph.n):
10          for j in range(i,graph.n):
11              if adj[i][j] == 1:
12                  G.add_edge(i,j)
13                  edge_labels[(i,j)] = round(a.euclidean(graph.nodes[i],graph.nodes[j]),4)
14              if not graph.nodes[i].neighbors:
15                  G.add_node(i)
16  # set the node labels to be the node names
17  labels = {}
18  for i in range(graph.n):
19      labels[i] = graph.nodes[i].name
20
21  # draw the graph with labels
22  nx.draw_networkx(G, coordinates, with_labels=True, labels=labels, arrows=False, ax=ax)
23  nx.draw_networkx_edge_labels(G, coordinates, edge_labels=edge_labels, ax=ax)
24
25
26  def visualize2(graph,path,adj,coordinates,ax):
27      G = nx.Graph()
28      edge_labels = {}
29      edge_colors = {}
30      if path is not None:
31          for i in range(len(path)-1):
32              edge_colors[(path[i].id, path[i+1].id)] = 'red'
33              edge_colors[(path[i+1].id, path[i].id)] = 'red'
34
35          for i in range(graph.n):
36              for j in range(i,graph.n):
37                  if adj[i][j] == 1:
38                      G.add_edge(i,j)
39                      edge_labels[(i,j)] = round(a.euclidean(graph.nodes[i],graph.nodes[j]),4)
40                      if (i,j) not in edge_colors and (j,i) not in edge_colors:
41                          edge_colors[(i,j)] = 'black'
42                          edge_colors[(j,i)] = 'black'

```

**Gambar 3.1.6**

**File visualize.py**

Pada folder **doc** terdapat lampiran laporan ini. Sedangkan pada folder **test** terdapat sejumlah test case. Test case yang kami gunakan memiliki spesifikasi sebagai berikut:

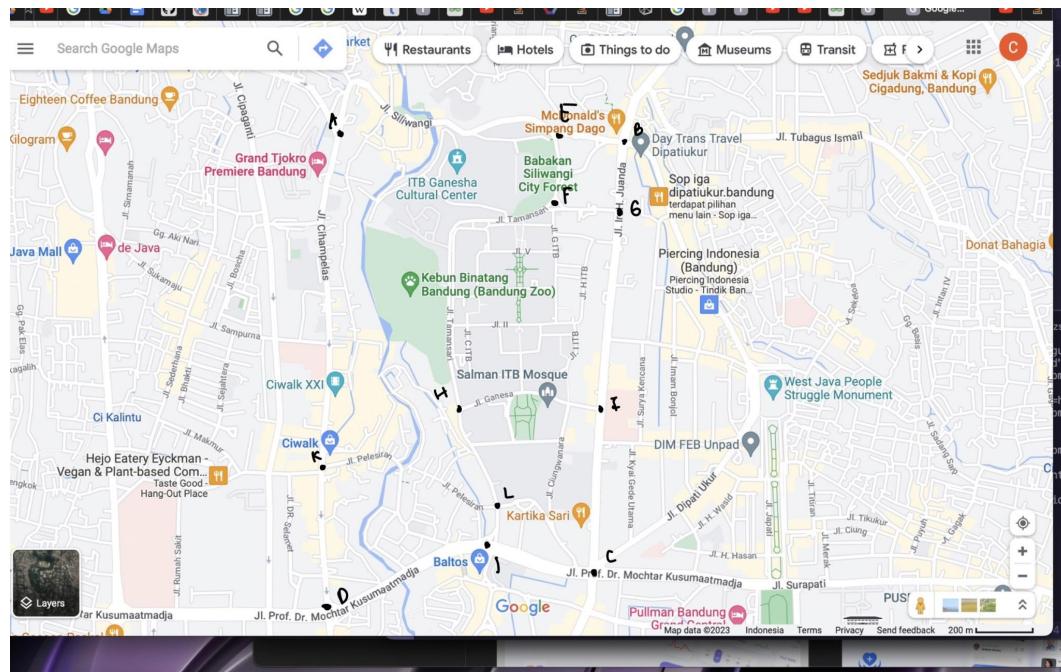
1. Untuk peta/graf dengan n buah simpul, file txt akan terdiri atas  $(n + 2)$  baris dan n kolom.
2. Baris pertama berisi nama-nama dari simpul
3. Baris kedua berisi koordinat dari simpul dengan format  $(x,y)$

4. Baris ketiga sampai baris ke-(n+2) berisi matriks ketetanggaan, dengan nilai 1 menunjukkan ketetanggaan, dan nilai 0 menunjukkan tidak adanya sisi antar simpul. Jalan diasumsikan dapat dilalui secara 2 arah, sehingga matriks ketetanggaan bersifat simetris.

Test case yang kami lampirkan memiliki detail sebagai berikut:

1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara

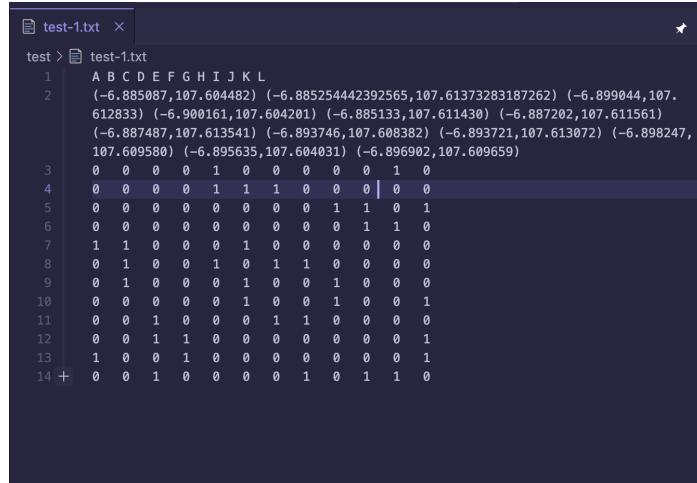
Dari peta sebagai berikut, dipilih 12 node yang masing-masing ditandai dalam gambar peta yang merepresentasikan sebagai berikut:



Gambar 3.2.1.a

Gambar Peta Sekitar Kampus ITB

Kemudian dihasilkan file `test-1.txt` dengan detail sebagai berikut:



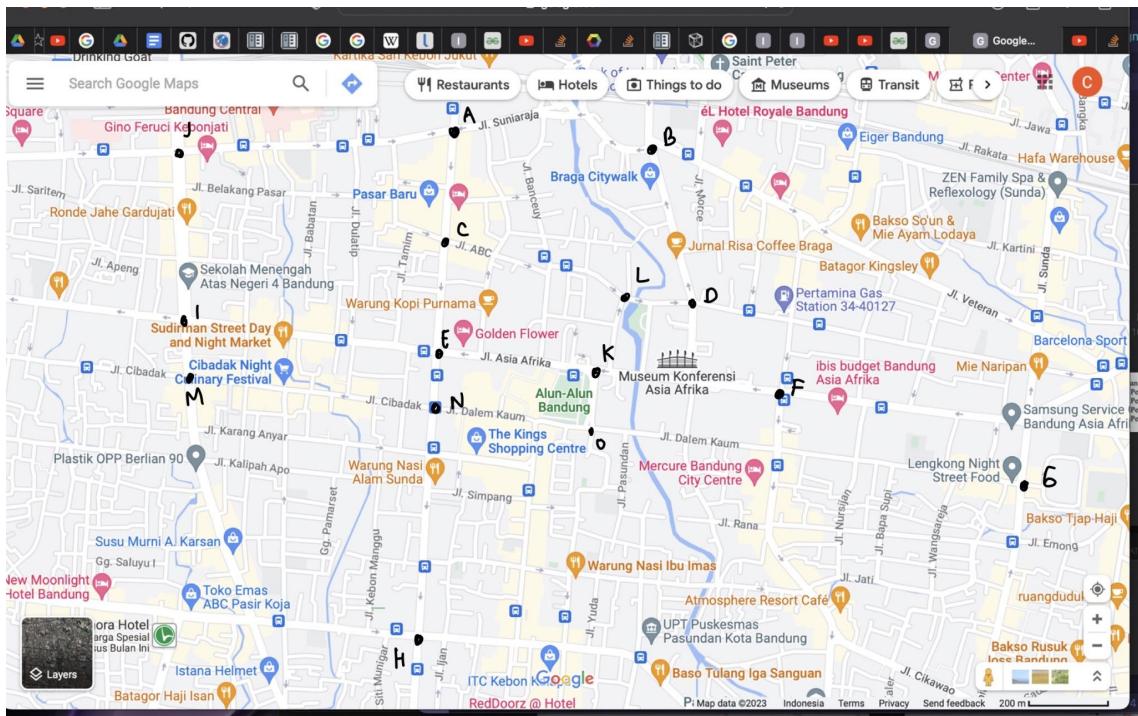
```
test > test-1.txt
test A B C D E F G H I J K L
1 (-6.885087,107.604482) (-6.885254442392565,107.61373283187262) (-6.899044,107.
2 612833) (-6.900161,107.604201) (-6.885133,107.611430) (-6.887202,107.611561)
(-6.887487,107.613541) (-6.893746,107.608382) (-6.893721,107.613072) (-6.898247,
107.609580) (-6.895635,107.604031) (-6.896902,107.609659)
3 0 0 0 0 1 0 0 0 0 0 0 1 0
4 0 0 0 0 1 1 1 0 0 0 | 0 0
5 0 0 0 0 0 0 0 0 0 0 1 1 0
6 0 0 0 0 0 0 0 0 0 0 1 1 0
7 1 1 0 0 0 1 0 0 0 0 0 0 0
8 0 1 0 0 0 1 0 1 1 0 0 0 0
9 0 1 0 0 0 1 0 0 0 1 0 0 0
10 0 0 0 0 0 1 0 0 0 1 0 0 1
11 0 0 1 0 0 0 0 1 1 0 0 0 0
12 0 0 1 1 0 0 0 0 0 0 0 0 1
13 1 0 0 1 0 0 0 0 0 0 0 0 1
14 + 0 0 1 0 0 0 0 1 0 1 1 0
```

Gambar 3.2.1.b

File test-1.txt

## 2. Peta jalan sekitar Alun-alun Bandung

Dari peta sebagai berikut, dipilih 15 node yang masing-masing ditandai dalam gambar peta yang merepresentasikan sebagai berikut:



Gambar 3.2.2.a

Gambar Peta Sekitar Alun-alun Bandung

Kemudian dihasilkan file `test-2.txt` dengan detail sebagai berikut:

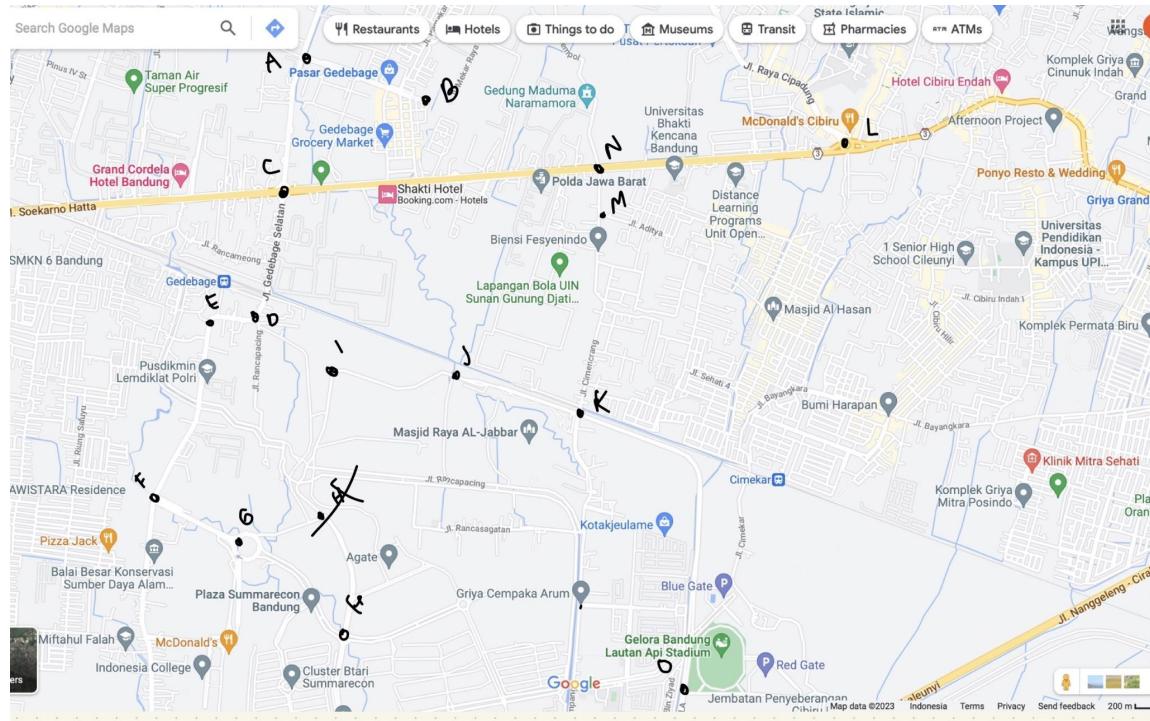
```
test > test-2.txt
1 A B C D E F G H I J K L M N 0
2 (-6.915813,107.604525) (-6.916174945368406,107.60885945682983) (-6.
918198586623415,107.6042675150454) (-6.919689685156928,107.60986796740868) (-6.
920818656628165,107.60413876901404) (-6.9216068049227095,107.6119708192539) (-6.
923798075004712,107.61745464326363) (-6.9272032225304825,107.60367036706238)
(-6.920049927439903,107.59830787985136) (-6.916359813219521,107.5981860051437)
(-6.921335411275937,107.60772270101911) (-6.919671843292535,107.6083777757285)
(-6.921426151179148,107.59842975455984) (-6.922084993328369,107.60401420875438)
(-6.922556811634229,107.60763878543729)
3 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0
4 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0
5 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
6 +
7 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0
8 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1
9 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1
10 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
11 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0
12 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
13 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1
14 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0
15 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1
16 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
17 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1
```

**Gambar 3.2.2.b**

**File test-2.txt**

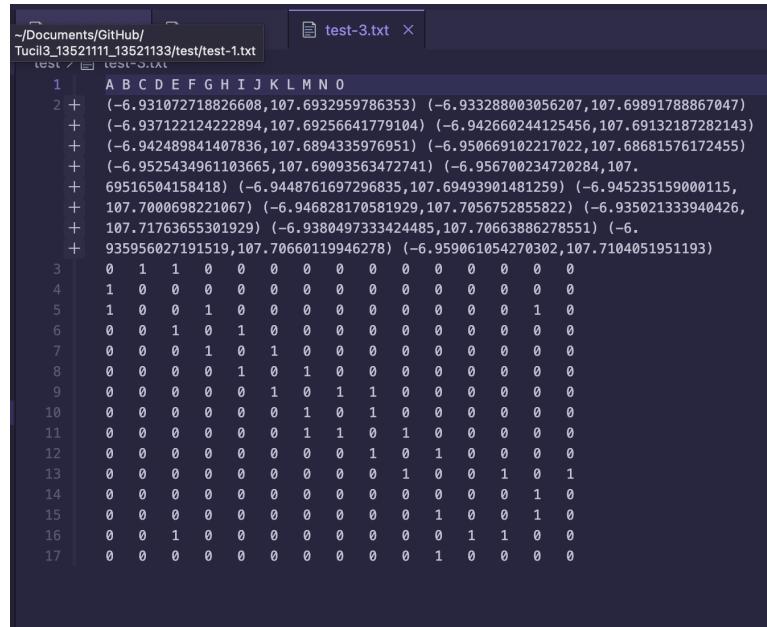
### 3. Peta jalan sekitar Buahbatu atau Bandung Selatan

Dari peta sebagai berikut, dipilih 15 node yang masing-masing ditandai dalam gambar peta yang merepresentasikan sebagai berikut:



**Gambar 3.2.3.a**  
**Gambar Peta Sekitar Bandung Selatan**

Kemudian dihasilkan file `test-3.txt` dengan detail sebagai berikut:



The screenshot shows a terminal window with the following details:

- Path: ~/Documents/GitHub/Tucil3\_13521111\_13521133/test/test-1.txt
- File: test-3.txt
- Content:

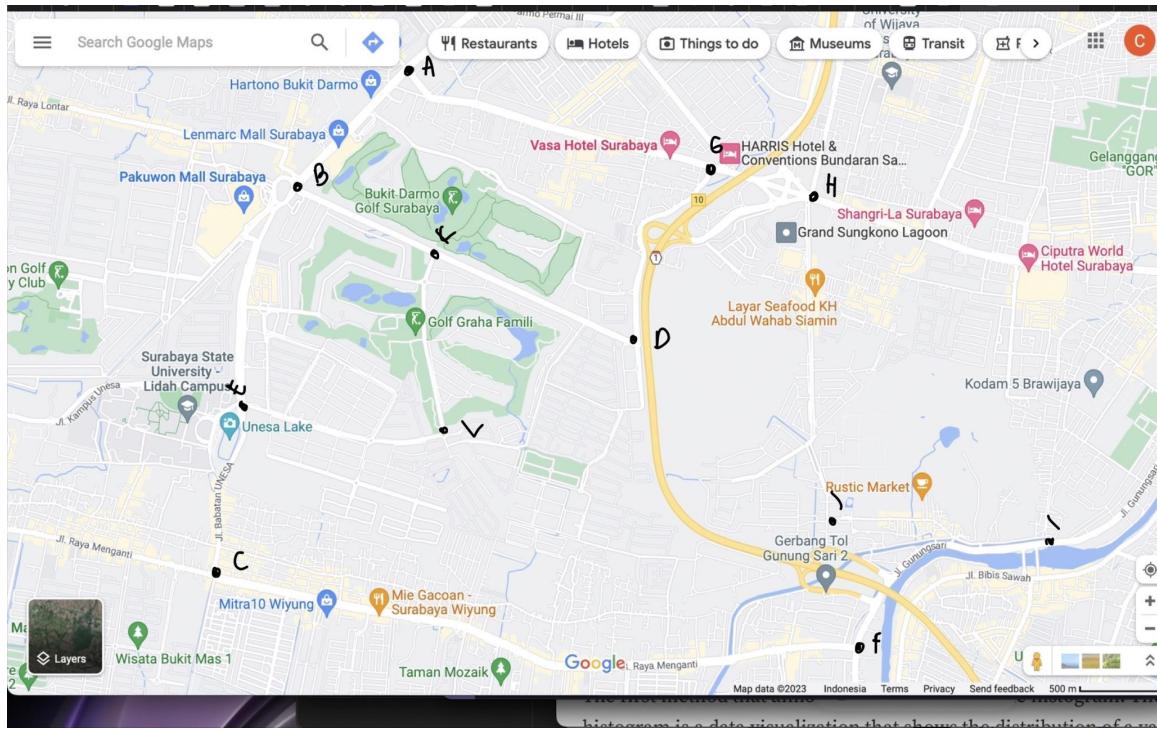
```
1   A B C D E F G H I J K L M N O
2 + (-6.931072718826608,107.6932959786353) (-6.933288003056207,107.69891788867047)
+ (-6.937122124222894,107.69256641779104) (-6.942660244125456,107.69132187282143)
+ (-6.942489841407836,107.6894335976951) (-6.950669102217022,107.68681576172455)
+ (-6.9525434961103665,107.69093563472741) (-6.956700234720284,107.
+ 69516504158418) (-6.9448761697296835,107.69493901481259) (-6.945235159000115,
+ 107.7000698221067) (-6.946828170581929,107.7056752855822) (-6.935021333940426,
+ 107.71763655301929) (-6.9380497333424485,107.7063886278551) (-6.
+ 935956027191519,107.70660119946278) (-6.959061054270302,107.7104051951193)
3  0   1   1   0   0   0   0   0   0   0   0   0   0   0   0
4  1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5  1   0   0   1   0   0   0   0   0   0   0   0   0   0   1
6  0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
7  0   0   0   1   0   1   0   0   0   0   0   0   0   0   0
8  0   0   0   0   1   0   1   0   0   0   0   0   0   0   0
9  0   0   0   0   0   1   0   1   1   0   0   0   0   0   0
10 0   0   0   0   0   0   1   0   1   0   1   0   0   0   0
11 0   0   0   0   0   0   0   1   1   0   1   0   0   0   0
12 0   0   0   0   0   0   0   0   0   1   0   1   0   0   0
13 0   0   0   0   0   0   0   0   0   0   1   0   0   0   1
14 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
15 0   0   0   0   0   0   0   0   0   0   0   1   0   0   1
16 0   0   1   0   0   0   0   0   0   0   0   0   1   1   0
17 0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
```

**Gambar 3.2.3.b**

**File test-3.txt**

#### 4. Peta jalan sebuah kawasan di kota asal

Dari peta sebagai berikut, dipilih 12 node yang masing-masing ditandai dalam gambar peta yang merepresentasikan sebagai berikut:



**Gambar 3.2.4.a**

#### Gambar Peta Sekitar Surabaya Barat

Kemudian dihasilkan file `test-4.txt` dengan detail sebagai berikut:

```

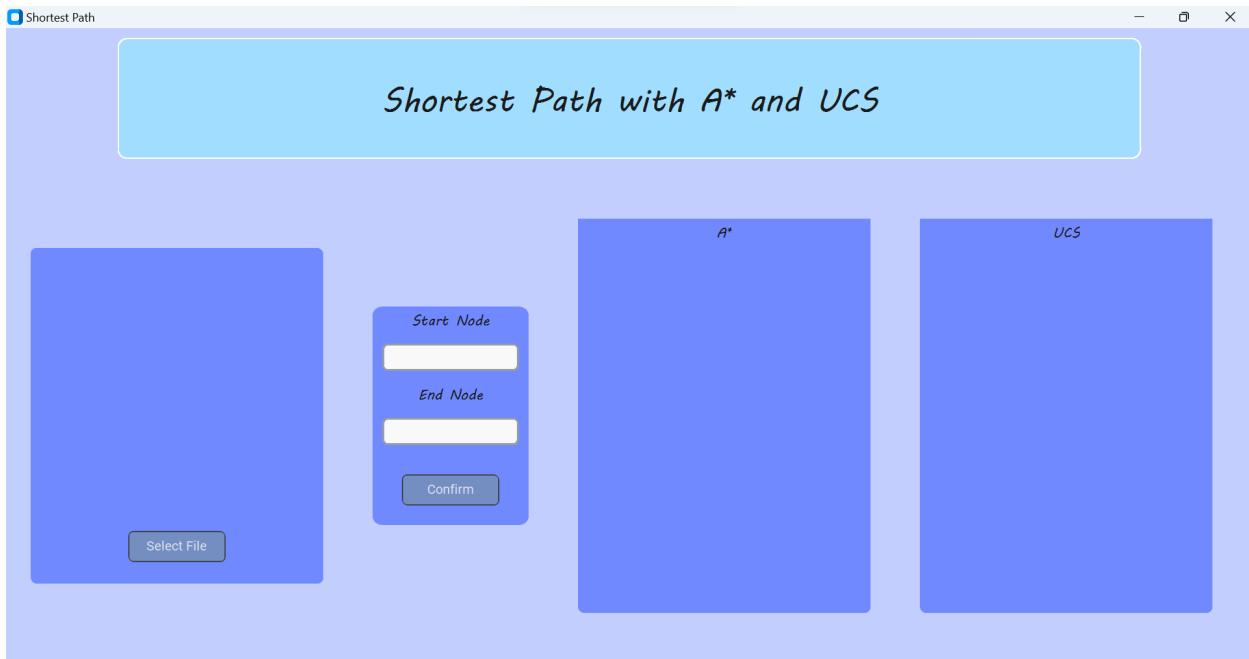
test > test-4.txt
  test-1.txt  test-2.txt  test-3.txt  test-4.txt x
test > test-4.txt
1   A B C D E F G H I J K L
2   (-7.281446742180804,112.6848356659058) (-7.287462200525647,112.6784395744414)
(-7.30884257797526,112.67402310555443) (-7.296376989143727,112.6974204860472)
(-7.299527982154259,112.6755479342228) (-7.312842606282772,112.70951034593273)
(-7.2866727617431515,112.7015402711931) (-7.288305981861206,112.7070443538573)
(-7.307170411012963,112.72034374053892) (-7.305884012776403,112.70807695007709)
(-7.291411777124504,112.68648847938053) (-7.301247588235915,112.68683973109172)
3   0 1 0 0 0 0 1 0 0 0 0 0
4   1 0 0 0 1 0 0 0 0 0 0 1 0
5   0 0 0 0 1 1 0 0 0 0 0 0 0
6   0 0 0 0 0 0 1 0 0 0 0 1 0
7   0 1 1 0 0 0 0 0 0 0 0 0 1
8   0 0 1 0 0 0 0 0 1 0 0 0 0
9   1 0 0 1 0 0 0 1 0 0 0 0 0
10  0 0 0 0 0 0 1 0 0 0 1 0 0
11  0 0 0 0 0 1 0 0 0 1 0 0 0
12 + 0 0 0 0 0 0 1 1 0 0 0 0 0
13  0 1 0 1 0 0 0 0 0 0 0 0 1
14  0 0 0 1 0 0 0 0 0 0 0 1 0

```

**Gambar 3.2.4.b**

#### File test-4.txt

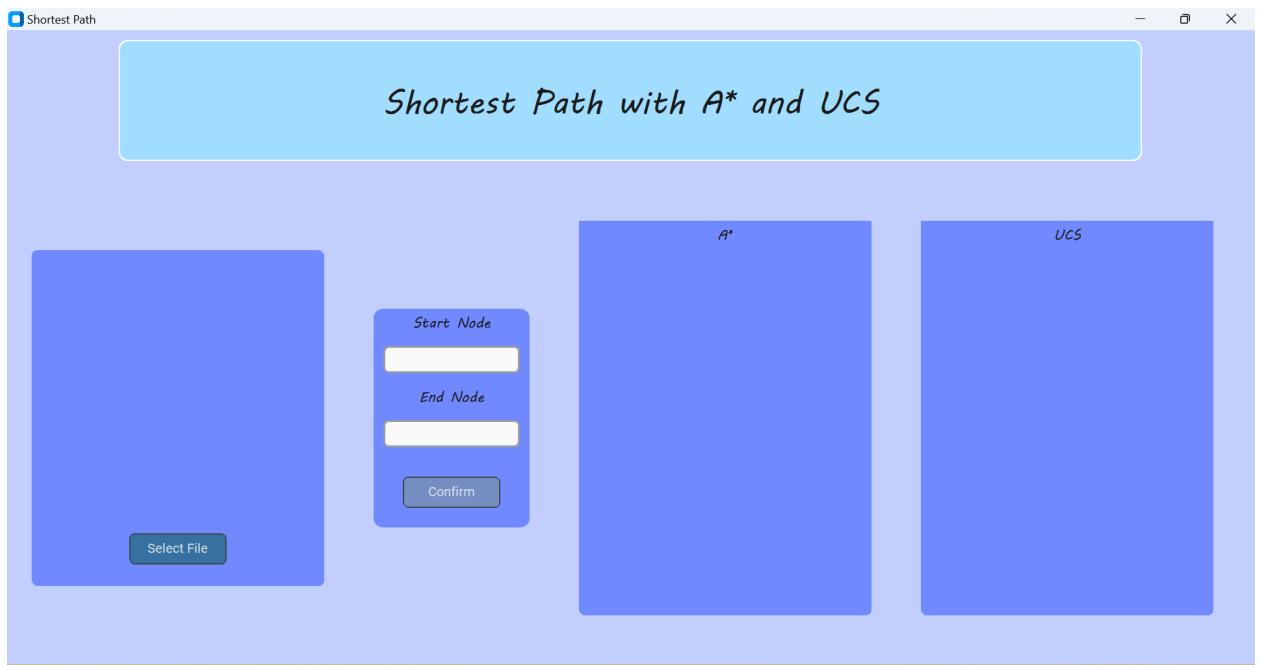
Dari implementasi tersebut, program dapat dijalankan dengan melakukan running code `python ./src/gui.py` di terminal yang mengarah pada directory folder program. Selanjutnya akan didapatkan tampilan GUI sebagai berikut (direkomendasikan untuk mengubah display settings Scale (150%) dan Display Resolution (1920x1080)):



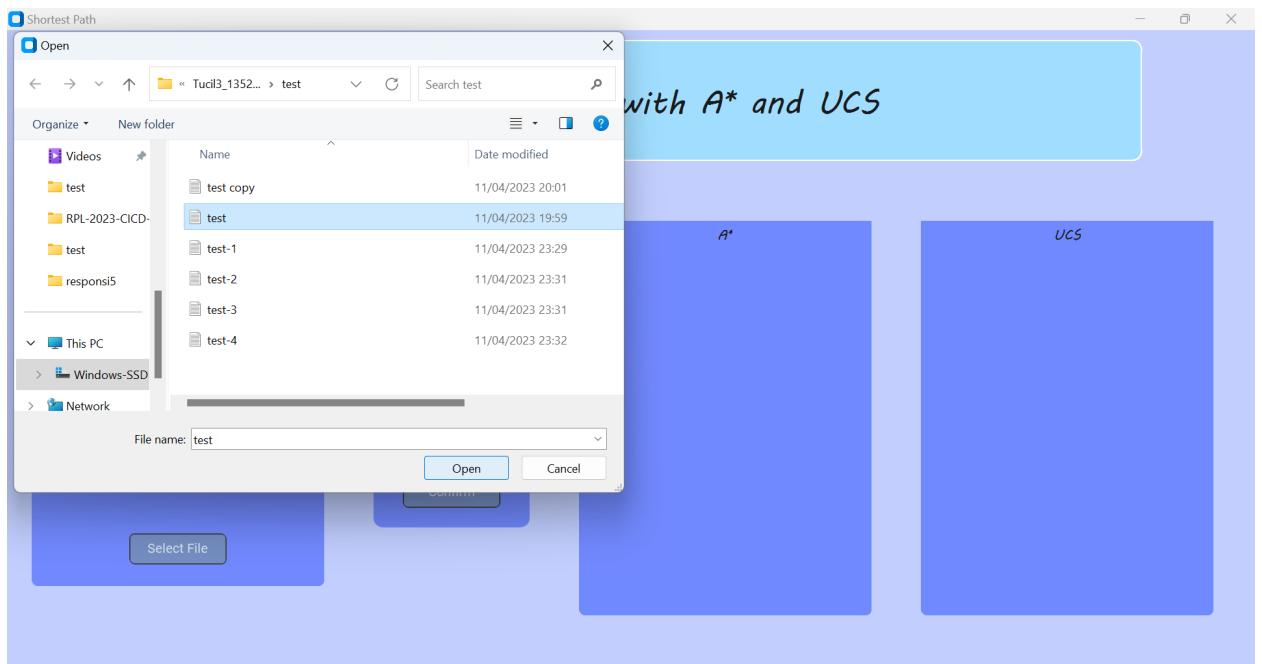
**Gambar 3.3**  
**Tampilan GUI**

Untuk mulai menjalankan pencarian lintasan tersingkat dapat mengikuti langkah-langkah sebagai berikut beserta contoh tampilan yang akan didapat oleh pengguna:

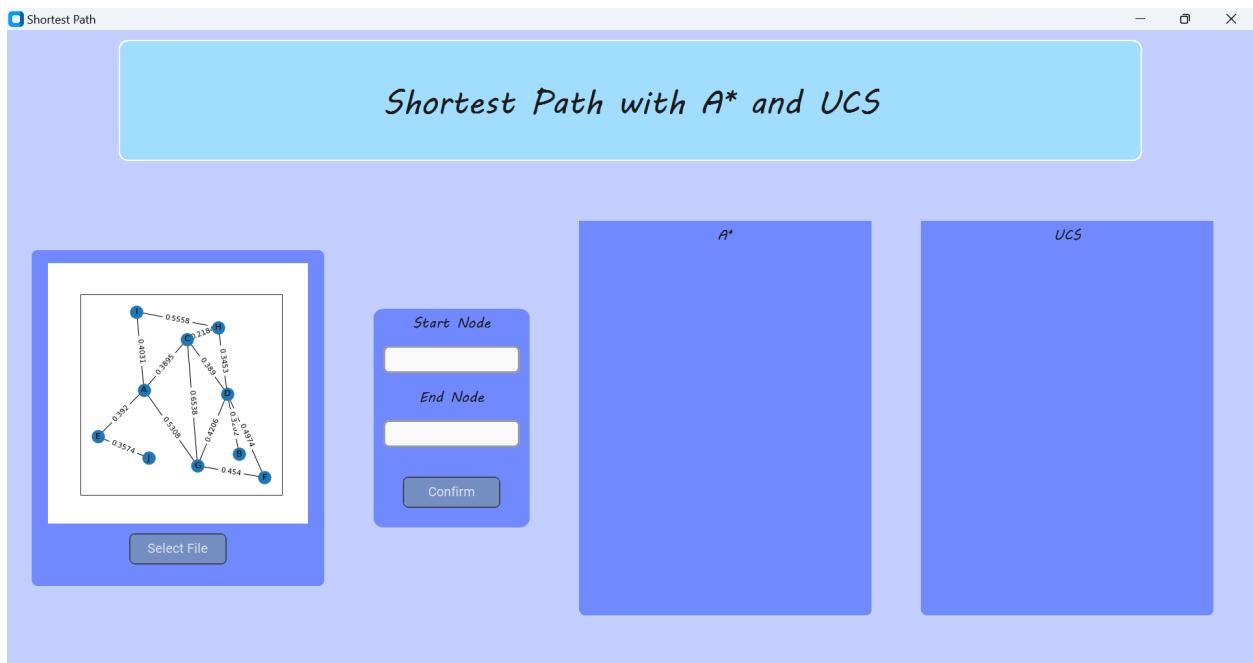
- a) Tekan button `select file`



- b) Pilih file test dengan format .txt yang akan digunakan untuk pencarian lintasan tersingkat



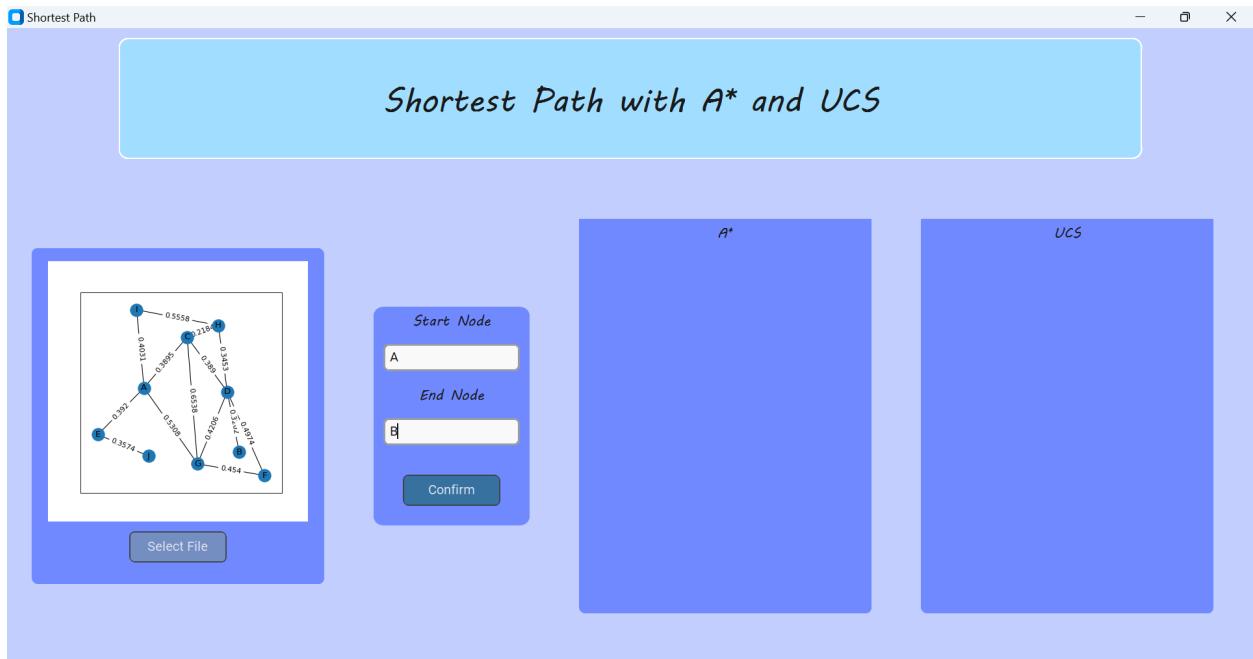
- c) Tunggu hingga ditampilkan *graph/ peta* yang merepresentasikan file



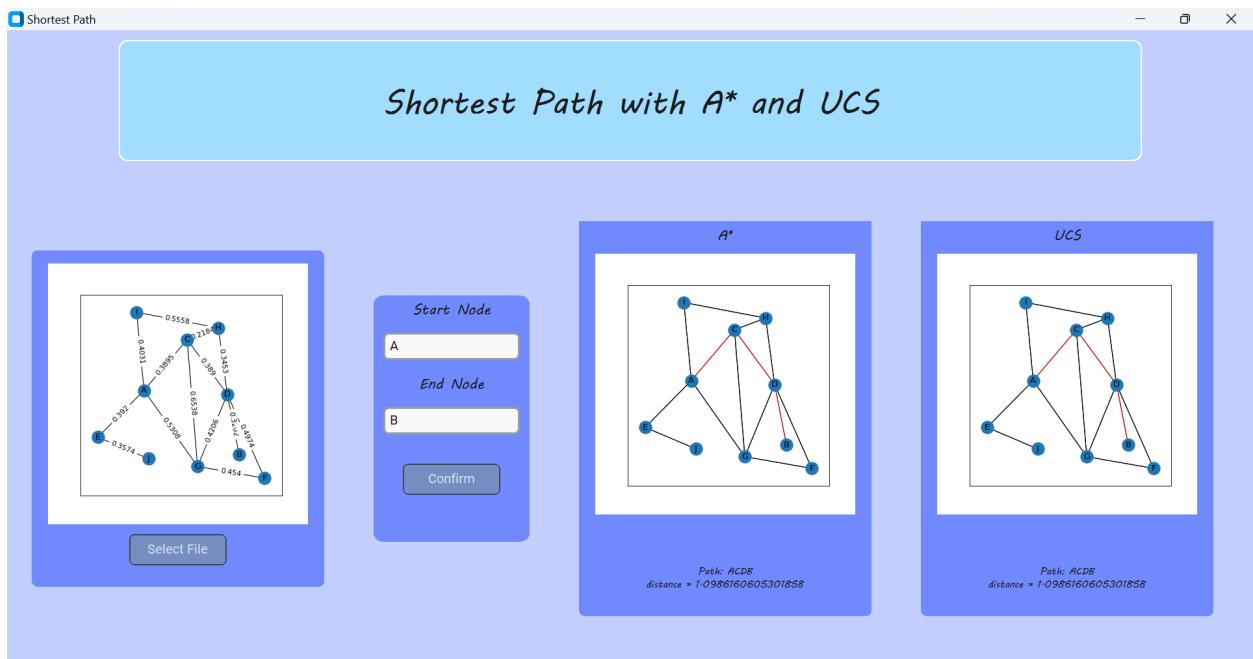
- d) Masukkan titik mula dan titik akhir pada kolom yang tersedia



- e) Tekan button 'confirm'



- f) Program akan menampilkan hasil pencarian lintasan tersingkat dengan menggunakan algoritma A\* dan UCS pada kolom di kanan kolom masukan dengan keterangan lintasan dan jarak yang ditempuh



## BAB 5

### PENGUJIAN

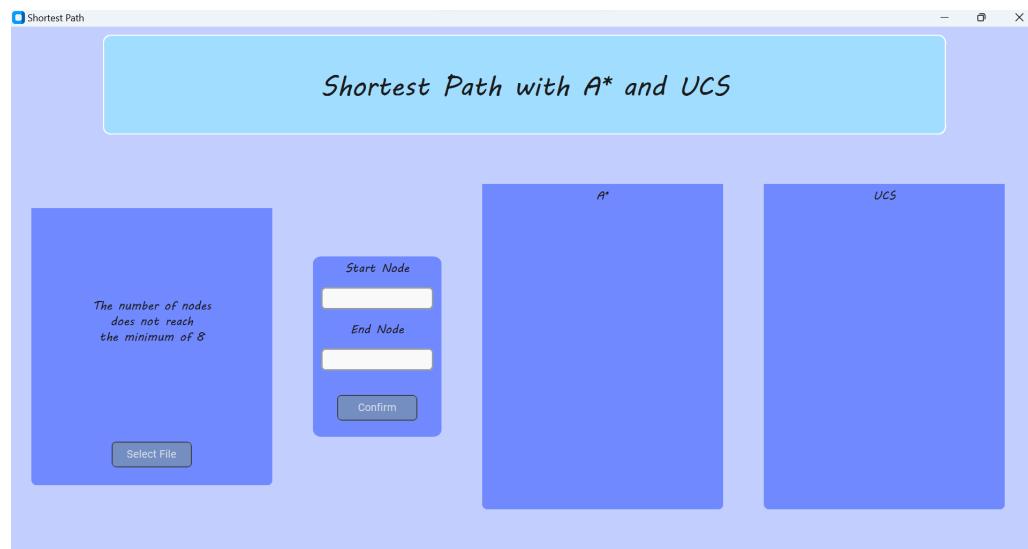
Berikut adalah langkah-langkah pengujian yang kami lakukan:

#### 5.1. Validasi Input

##### 5.3.1. Validasi file txt (jumlah simpul harus $\geq 8$ )

Input						
1	A	B	C	D	E	F
2	(0.12,0.56)	(0.76,0.23)	(0.41,0.82)	(0.68,0.54)	(-0.19,0.32)	(0.93,0.11)
3	0	0	1	0	1	0
4	0	0	0	1	0	0
5	1	0	0	1	0	0
6	0	1	1	0	0	1
7	1	0	0	0	0	0
8	0	0	0	1	0	0

Output						
						

### 5.3.2. Validasi nama simpul mulai dan simpul akhir

<b>Input</b>	
File:	<pre>1   A B C D E F G H I J 2   (0.12,0.56) (0.76,0.23) (0.41,0.82) (0.68,0.54) (-0.19,0.32) (0.93,0.11) (0.48,0.17) (0.62,0.88) (0.07,0.96) (0.15,0.21) 3   0   0   1   0   1   0   1   0   1   0 4   0   0   0   1   0   0   0   0   0   0 5   1   0   0   1   0   0   0   1   1   0   0 6   0   1   1   0   0   1   1   1   0   0 7   1   0   0   0   0   0   0   0   0   1 8   0   0   0   1   0   0   1   0   0   0 9   1   0   1   1   0   1   0   0   0   0 10  0   0   1   1   0   0   0   0   1   0 11  1   0   0   0   0   0   0   1   0   0 12  0   0   0   0   1   0   0   0   0   0</pre>
Start: A	
End: b	
<b>Output</b>	
	

## 5.2. Pengujian Test Case

### 5.3.1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara

<p>Input</p> <p>File:</p> <pre> test-1.txt test &gt; test-1.txt 1   A B C D E F G H I J K L 2   (-6.885087,107.604482) (-6.885254442392565,107.61373283187262) (-6.899044,107. 612833) (-6.900161,107.604201) (-6.885133,107.611430) (-6.887202,107.611561) (-6.887487,107.613541) (-6.893746,107.608382) (-6.893721,107.613072) (-6.898247, 107.609580) (-6.895635,107.604031) (-6.896902,107.609659) 3   0 0 0 0 1 0 0 0 0 0 1 0 4   0 0 0 0 1 1 1 0 0 0   0 0 5   0 0 0 0 0 0 0 0 0 1 1 0 1 6   0 0 0 0 0 0 0 0 0 0 1 1 0 7   1 1 0 0 0 1 0 0 0 0 0 0 0 8   0 1 0 0 1 0 1 1 0 0 0 0 0 9   0 1 0 0 0 1 0 0 0 1 0 0 0 10  0 0 0 0 0 1 0 0 0 1 0 0 1 11  0 0 1 0 0 0 1 1 0 0 0 0 0 12  0 0 1 1 0 0 0 0 0 0 0 0 1 13  1 0 0 1 0 0 0 0 0 0 0 0 1 14 + 0 0 1 0 0 0 0 1 0 1 1 0 </pre>
<p>Start: A End: C</p>
<p>Output</p> <p><b>Shortest Path with A* and UCS</b></p>

### 5.3.2. Peta jalan sekitar Alun-alun Bandung

<b>Input</b> File: <pre> test &gt; test-2.txt 1   A B C D E F G H I J K L M N O 2   (-6.915813,107.604525) (-6.916174945368406,107.60885945682983) (-6. 918198586623415,107.6042675150454) (-6.919689685156928,107.60986796740868) (-6. 920818656628165,107.60413876901404) (-6.9216068049227095,107.6119708192539) (-6. 923798075004712,107.61745464326363) (-6.9272032225304825,107.60367036706238) (-6.920049927439903,107.59830787985136) (-6.916359813219521,107.5981860051437) (-6.921335411275937,107.60772270101911) (-6.919671843292535,107.6083777757285) (-6.921426151179148,107.59842975455904) (-6.922084993328369,107.60401420875438) (-6.922556811634229,107.60763878543729) 3   0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 4   1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 5   1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 6   + 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 7   0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 8   0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 9   0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 11  0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 12  1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 13  0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 14  0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 15  0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 16  0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 17  0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 </pre>
Start: D End: H
<b>Output</b> 

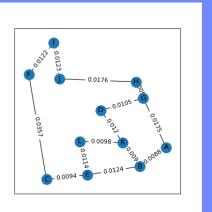
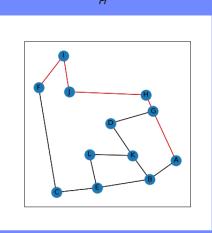
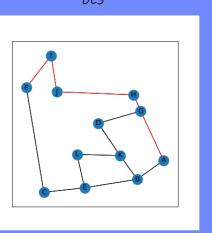
### 5.3.3. Peta jalan sekitar Buahbatu atau Bandung Selatan

Input																	
File:																	
1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
2	+	(-6.931072718826608, 107.6932959786353)	(-6.933288003056207, 107.69891788867047)														
	+	(-6.937122124222894, 107.69256641779104)	(-6.942660244125456, 107.69132187282143)														
	+	(-6.942489841407836, 107.6894335976951)	(-6.950669102217022, 107.68681576172455)														
	+	(-6.9525434961103665, 107.69093563472741)	(-6.956700234720284, 107.69516504158418)														
	+	(-6.9448761697296835, 107.69493901481259)	(-6.945235159000115, 107.7000698221067)														
	+	(-6.946828170581929, 107.7056752855822)	(-6.935021333940426, 107.71763655301929)														
	+	(-6.9380497333424485, 107.70663886278551)	(-6.95906027191519, 107.70660119946278)														
3	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0		
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
5	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0		
6	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0		
7	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0		
8	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0		
9	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0		
10	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0		
11	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0		
12	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0		
13	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1		
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		
15	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1		
16	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0		
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		

Start: L  
End: F

Output																	
Shortest Path with A* and UCS																	
<p><b>A*</b></p> <p>Path: LNCDDEF distance = 0.09131817217631176</p>																	
<p><b>UCS</b></p> <p>Path: LNCDDEF distance = 0.09131817217631176</p>																	

### 5.3.4. Peta jalan sebuah kawasan di kota asalmu

<b>Input</b> <b>File:</b> <div style="background-color: #2e3436; color: #eeeeec; padding: 5px; border-radius: 5px;"> <span style="margin-right: 10px;">test-1.txt</span> <span style="margin-right: 10px;">test-2.txt</span> <span style="margin-right: 10px;">test-3.txt</span> <span style="margin-right: 10px;">test-4.txt ×</span> </div> <pre style="font-family: monospace; background-color: #2e3436; color: #eeeeec; padding: 10px; border-radius: 5px;"> test &gt; test-4.txt 1   A B C D E F G H I J K L 2   (-7.281446742180804, 112.68483356659058) (-7.287462200525647, 112.6784395744414) (-7.308842577977526, 112.6740231055443) (-7.296376989143727, 112.6974204860472) (-7.299527982154259, 112.67554793422228) (-7.312842606282772, 112.70951034593273) (-7.2866727617431515, 112.7015402711931) (-7.288305981861206, 112.7070443538573) (-7.307170411012963, 112.72034374053892) (-7.305884012776403, 112.70807695007709) (-7.291411777124504, 112.68648847938053) (-7.301247588235915, 112.68683973109172)  3   0   1   0   0   0   0   1   0   0   0   0   0   0 4   1   0   0   0   1   0   0   0   0   0   1   0 5   0   0   0   0   1   1   0   0   0   0   0   0 6   0   0   0   0   0   0   1   0   0   0   1   0 7   0   1   1   0   0   0   0   0   0   0   0   1 8   0   0   1   0   0   0   0   0   0   1   0   0 9   1   0   0   1   0   0   0   0   1   0   0   0 10  0   0   0   0   0   0   1   0   0   1   0   0 11  0   0   0   0   0   1   0   0   0   0   1   0 12 + 0   0   0   0   0   0   0   1   1   0   0   0 13  0   1   0   1   0   0   0   0   0   0   0   1 14  0   0   0   0   1   0   0   0   0   0   1   0 </pre>
<b>Start: A</b> <b>End: F</b>
<b>Output</b> <div style="border: 1px solid #2e3436; padding: 10px; width: fit-content; margin: auto;"> <p style="text-align: center;"><b>Shortest Path with A* and UCS</b></p> <div style="display: flex; justify-content: space-around; margin-top: 20px;"> <div style="text-align: center;">  <p>Select File</p> </div> <div style="text-align: center;"> <p><b>A*</b></p>  <p>Path: AGHJIF distance = 0.065417718222927254</p> </div> <div style="text-align: center;"> <p><b>UCS</b></p>  <p>Path: AGHJIF distance = 0.065417718222927254</p> </div> </div> </div>

### 5.3. Pengujian Lainnya

#### 5.3.1. Tidak Ada Jalur

<p><b>Input</b></p> <p>File:</p> <pre> A B C D E F G H I J (0.12,0.56) (0.76,0.23) (0.41,0.82) (0.68,0.54) (-0.19,0.32) (0.93,0.11) (0.48,0.17) (0.62,0.88) (0.07,0.96) (0.15,0.21) 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 </pre> <p>Start: A End: J</p>
<p><b>Output</b></p>

## **BAB 6**

### **PENUTUP**

#### **A. Kesimpulan**

Pencarian rute terpendek dapat dibuat dengan menggunakan algoritma A\* dan UCS dengan memanfaatkan berbagai library yang berada di python serta diterapkan pada node-node realistik yang didapatkan dari peta digital terhadap garis bujur dan lintang bumi.

#### **B. Komentar dan Saran**

Dapat dikembangkan algoritma yang lebih efisien dan realistik dari dasar algoritma A\* dan UCS ini yang juga dapat mempertimbangkan berbagai faktor rute terpendek seperti topologi jalan, jarak realistik, dan keadaan trafik pada penelitian ataupun tugas yang lebih lanjut. Selain itu dengan sediaan waktu yang lebih lama, pengembangan tugas dapat lebih optimal.

## **LAMPIRAN DAN DAFTAR PUSTAKA**

### **TABEL CHECKLIST SPEK**

Poin	Ya	Tidak
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		

### PEMBAGIAN TUGAS

NIM	Nama	Tugas
13521111	Tabitha Permalla	Algoritma A*, GUI dan visualisasi, debugging
13521133	Cetta Reswara Parahita	Algoritma UCS, testing, laporan dan dokumentasi

**Link repository GitHub :**

[https://github.com/Bitha17/Tucil3\\_13521111\\_13521133](https://github.com/Bitha17/Tucil3_13521111_13521133)

**Referensi :**

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>

<https://www.geeksforgeeks.org/>

<https://i11www.iti.kit.edu/extra/publications/dssw-erpa-09.pdf>

<https://www.trivusi.web.id/>