

Tugas Kecil 3 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

**Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan
Terpendek**

Disusun oleh:

Shelma Salsabila 13521115

Ferindya Aulia Berlianty 13521161



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

2023

DAFTAR ISI

DAFTAR ISI	1
DAFTAR GAMBAR	2
BAB 1 DESKRIPSI TUGAS	3
BAB 2 LANDASAN TEORI	6
2.1 Penjelasan Algoritma UCS (<i>Uniform Cost Search</i>)	6
2.2 Penjelasan Algoritma A*	8
BAB 3 IMPLEMENTASI	10
3.1 Algoritma UCS (<i>Uniform Cost Search</i>)	10
3.2 Algoritma A*	11
3.3 Algoritma menghitung distance	13
3.4 Source Code dengan Bahasa Python	14
BAB 4 PENGUJIAN	22
4.1 Test case untuk Peta jalan sekitar kampus ITB/Dago/Bandung Utara	22
4.2 Test case untuk Peta jalan sekitar Alun-alun Bandung	26
4.3 Test case untuk Peta jalan sekitar Buahbatu atau Bandung Selatan	30
4.4 Test Case Untuk Peta Jalan Sebuah Kawasan di Kota Garut	34
4.5 Test Case Untuk Peta Jalan Sebuah Kawasan di Kota Klaten	38
4.6 Test Case untuk Kasus Error	42
BAB V CARA MENJALANKAN PROGRAM	44
BAB VI KESIMPULAN DAN SARAN	49
5.1 Kesimpulan	49
5.2 Saran	49
REFERENSI	50
LAMPIRAN	50

DAFTAR GAMBAR

Gambar 1.1 Ilustrasi peta dalam Google Map

Gambar 2.1 Ilustrasi gambar simpul serta rute antara simpul untuk diproses dengan algoritma UCS

Gambar 3.1 Tampilan source code pencarian.py

Gambar 3.2 Tampilan source code AStar.py

Gambar 3.3 Tampilan source code helper.py

Gambar 3.4 Tampilan source code Visualizer.py

Gambar 3.5 Tampilan source code gui.py

Gambar 4.1 Output shortest path, visited nodes, distance sum algoritma UCS untuk file itb.txt

Gambar 4.2 Visualisasi graph menggunakan algoritma UCS untuk file itb.txt

Gambar 4.3 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file itb.txt

Gambar 4.4 Visualisasi graph menggunakan algoritma ASTAR untuk file itb.txt

Gambar 4.5 Output shortest path, visited nodes, distance sum algoritma UCS untuk file alun2.txt

Gambar 4.6 Visualisasi graph menggunakan algoritma UCS untuk file alun2.txt

Gambar 4.7 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file alun2.txt

Gambar 4.8 Visualisasi graph menggunakan algoritma ASTAR untuk file alun2.txt

Gambar 4.9 Output shortest path, visited nodes, distance sum algoritma UCS untuk file buahbatu.txt

Gambar 4.10 Visualisasi graph menggunakan algoritma UCS untuk file buahbatu.txt

Gambar 4.11 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file buahbatu.txt

Gambar 4.12 Visualisasi graph menggunakan algoritma ASTAR untuk file buahbatu.txt

Gambar 4.13 Output shortest path, visited nodes, distance sum algoritma UCS untuk file garut.txt

Gambar 4.14 Visualisasi graph menggunakan algoritma UCS untuk file garut.txt

Gambar 4.15 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file garut.txt

Gambar 4.16 Visualisasi graph menggunakan algoritma ASTAR untuk file garut.txt

Gambar 4.17 Output shortest path, visited nodes, distance sum algoritma UCS untuk file klaten.txt

Gambar 4.18 Visualisasi graph menggunakan algoritma UCS untuk file klaten.txt

Gambar 4.19 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file klaten.txt

Gambar 4.20 Visualisasi graph menggunakan algoritma ASTAR untuk file klaten.txt

Gambar 4.21 TestCase untuk file belum diinputkan

Gambar 4.22 TestCase start node salah diinputkan

Gambar 5.1 Tampilan GUI

Gambar 5.2 Tampilan GUI saat memilih file

Gambar 5.3 Tampilan GUI setelah memilih file

Gambar 5.4 Tampilan GUI saat memasukkan Start Node dan Goal Node

Gambar 5.5 Visualisasi Graph

Gambar 5.6 Tampilan Shortest Path dan Visited Nodes

Gambar 5.7 Tampilan Shortest Path, Visited Nodes, dan Distance Sum

BAB 1

DESKRIPSI 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 cara lainnya yang disediakan oleh Google Map.



Gambar 1.1 Ilustrasi peta dalam Google Map

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
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: 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.

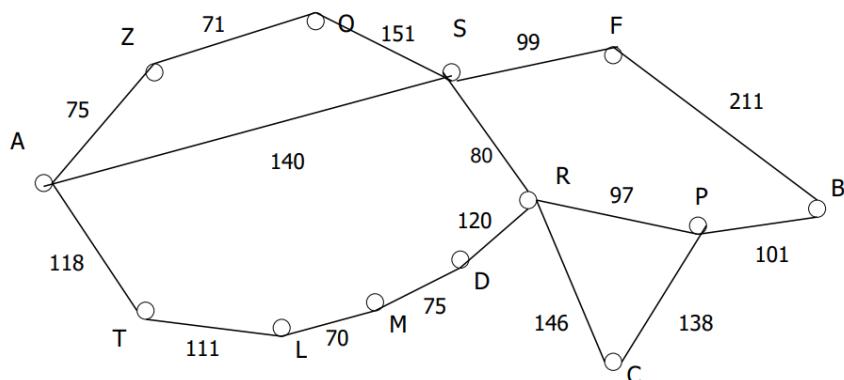
BAB 2

LANDASAN TEORI

2.1 Penjelasan Algoritma UCS (*Uniform Cost Search*)

Algoritma Uniform Cost Search (UCS) merupakan sebuah algoritma uninformed search yang dapat digunakan untuk menentukan rute. Algoritma menerima data berisi simpul data serta rute antara simpul yang diberikan sebuah skor. Simpul-simpul dalam data beserta rutennya dapat digambarkan agar isi dari data lebih mudah dipahami.

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 dimana node diperluas sesuai dengan biaya traversalnya dari node root. Biasanya algoritma *Uniform-Cost Search* diimplementasikan dengan menggunakan priority queue di mana prioritasnya adalah menurunkan biaya operasi. *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. Di setiap iterasi, periksa apakah item sudah dalam antrian prioritas (menggunakan array yang dikunjungi). Jika ya, lakukan kunci penurunan, jika tidak, masukkan item tersebut ke dalam antrian.



Gambar 2.1 Ilustrasi gambar simpul serta rute antara simpul untuk diproses dengan algoritma UCS

Sebelum menggunakan algoritma UCS, pengguna harus mengidentifikasi terlebih dahulu simpul awal serta simpul yang akan dicari. Algoritma UCS akan memeriksa simpul yang memiliki rute dari simpul awal terlebih dahulu serta bobot dari rute simpul tersebut. Untuk menentukan simpul yang akan diperiksa selanjutnya, diperlukan *priority queue* berdasarkan bobot terkecil yang ada. Simpul selanjutnya yang diperiksa diambil dari awal *priority queue* dan dihapus dari *priority queue* setelah diperiksa. Simpul tersebut akan diperiksa seperti simpul awal dengan bobot yang didapatkan oleh simpul merupakan bobot total dari simpul awal sampai simpul tersebut. Simpul-simpul dalam data yang diberikan akan terus ditelusuri berdasarkan *priority queue* yang dibuat hingga simpul yang dicari telah diperiksa. Ketika simpul yang dicari telah ditelusuri oleh program, maka *priority queue* akan dikosongkan sehingga proses pencarian berhenti.

Simpul-E	Simpul Hidup
A	Z _{A-75} , T _{A-118} , S _{A-140}
Z _{A-75}	T _{A-118} , S _{A-140} , O _{AZ-146}
T _{A-118}	S _{A-140} , O _{AZ-146} , L _{AT-229}
S _{A-140}	O _{AZ-146} , R _{AS-220} , L _{AT-229} , F _{AS-239} , O _{AS-291}
O _{AZ-146}	R _{AS-220} , L _{AT-229} , F _{AS-239} , O _{AS-291}
R _{AS-220}	L _{AT-229} , F _{AS-239} , O _{AS-291} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366}
L _{AT-229}	F _{AS-239} , O _{AS-291} , M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366}
F _{AS-239}	O _{AS-291} , M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366} , B _{ASF-450}
O _{AS-291}	M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366} , B _{ASF-450}
M _{ATL-299}	P _{ASR-317} , D _{ASR-340} , D _{ATLM-364} , C _{ASR-366} , B _{ASF-450}
P _{ASR-317}	D _{ASR-340} , D _{ATLM-364} , C _{ASR-366} , B _{ASRP-418} , D _{ASR-340} , C _{ASRP-455} , B _{ASF-450}
D _{ASR-340}	D _{ATLM-364} , C _{ASR-366} , B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
D _{ATLM-364}	C _{ASR-366} , B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
C _{ASR-366}	B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
B _{ASRP-418}	Solusi ketemu

Tabel 2.1 Contoh proses data gambar 2.1 menggunakan algoritma UCS

Algoritma UCS memiliki kelebihan dalam mencari rute karena hasil rute yang dicari akan selalu menjadi hasil yang optimal karena algoritma mencari simpul dengan bobot terkecil terlebih dahulu. Kekurangan dari algoritma UCS yaitu penggunaan memori dan proses yang relatif besar dibandingkan dengan algoritma *search* yang lain, khususnya algoritma *informed search* yang dapat menghasilkan rute optimal dengan proses yang lebih sedikit.

2.2 Penjelasan Algoritma A*

Algoritma A* (A Star) adalah algoritma pencarian yang digunakan untuk menemukan jalur terpendek antara titik awal dan akhir. Algoritma ini sering digunakan untuk penjelajahan peta guna menemukan jalur terpendek yang akan diambil. A* awalnya 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. Adapun kelemahan utama dari algoritma ini adalah kompleksitas ruang dan waktunya. Algoritma A* membutuhkan banyak ruang untuk menyimpan semua kemungkinan jalur dan banyak waktu untuk menemukannya. A* menggunakan Best First Search (BFS) dan menemukan jalur dengan biaya terkecil (least-cost path) dari node awal (initial node) yang diberikan ke node tujuan (goal node). Algoritma ini menggunakan fungsi heuristik jarak ditambah biaya (biasa dinotasikan dengan $f(n)$) untuk menentukan urutan di mana search-nya melalui node-node yang ada pada tree. 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 terakhir

Algoritma A* menemukan jalur terpendek antara dua node dalam sebuah graph. Algoritma ini mirip dengan algoritma Dijkstra, tetapi lebih canggih karena mempertimbangkan biaya setiap sisi (*edge*) dalam graph. Biaya tepi (*edge cost*) biasanya ditentukan oleh panjangnya atau ukuran jarak lainnya, seperti waktu atau uang.

BAB 3

IMPLEMENTASI

3.1 Algoritma UCS (*Uniform Cost Search*)

Algoritma UCS yang kami gunakan untuk mencari lintasan terpendek (*shortest path*) adalah sebagai berikut:

1. Program membaca file input yang didalamnya terdiri dari simpul apa saja serta kordinat dari tiap titik di google maps. Serta, tidak lupa matriks ketetanggaan dari semua simpul.
2. Kemudian semua simpul itu dimasukkan kedalam struktur dict beserta dengan tetangganya dan juga jarak diantara mereka yang nantinya kita akan sebut dengan fn.
3. Adapun menentukan jarak, bagi rute antara simpul, akan ditentukan jarak antara 2 simpul tersebut menggunakan sebuah fungsi eucleidianDistance yang terdapat dalam file helper.py. Adapun informasi kordinat latitude dan longitude didapat dari aplikasi google maps.
4. Setelah itu program akan meminta dari user untuk menentukan simpul awal dan simpul akhir pada proses pencarian ini.
5. Setiap simpul tetangga yang disimpan bersamaan dengan jarakya tadi dicari yang nilai fn yang paling kecil akan dikunjungi terlebih dahulu.
6. Kemudian ketika mengunjungi sebuah simpul akan dianalisis lagi tetangganya dan diperhatikan fn dari setiap tetangga lalu diambil simpul dengan nilai fn terkecil. Secara lebih jelas akan dijelaskan dalam sebuah kasus. Dalam contoh kasus ini hanya digunakan empat simpul untuk mempermudah visualisasi.

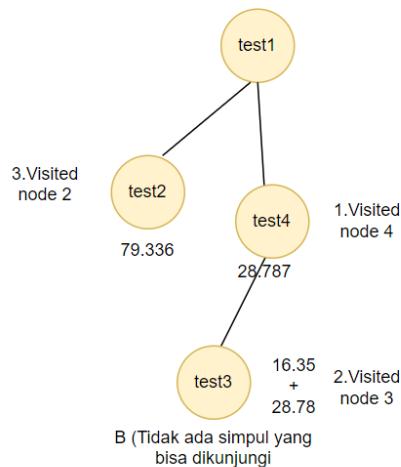
UCS
Input file
test1 test2 test3 test4
test1 4.89 11.67
test2 7.88 90.98
test3 9.90 23.65
test4 10.00 40.00
MATRIKS
0 1 0 1

1	0	0	0
1	0	0	1
0	0	1	0

1. Dipanggil fungsi readfile maka akan menghasilkan suatu dictionary sebagai berikut

```
{ {'test1': {'test2': 79.366, 'test4': 28.787}, 'test2': {'test1': 79.366}, 'test3': {'test1': 12.985, 'test4': 16.35}, 'test4': {'test3': 16.35}}}
```

2. Atau dalam graf dapat digambarkan sebagai berikut



3. Artinya shortpath yang didapat adalah test1 -> test2

3.2 Algoritma A*

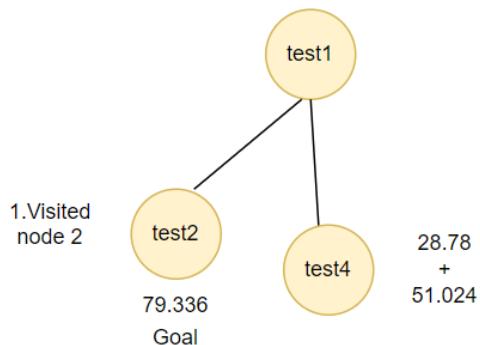
Pada dasarnya antara algoritma UCS dan algoritma A* yang dibuat tidak berbeda jauh. Hanya saja ada nilai gn yaitu nilai dari simpul yang dianalisis ke simpul goal yang dipertimbangkan untuk mencari shortpath dengan A*. Adapun Algoritma A* yang kami gunakan untuk mencari lintasan terpendek (*shortest path*) adalah sebagai berikut:

1. Program membaca file input yang didalamnya terdiri dari simpul apa saja serta kordinat dari tiap titik di google maps. Serta, tidak lupa matriks ketetanggaan dari semua simpul.
2. Kemudian semua simpul itu dimasukkan kedalam struktur dict beserta dengan tetangganya dan juga jarak diantara mereka yang nantinya kita akan sebut dengan fn.
3. Adapun menentukan jarak, bagi rute antara simpul, akan ditentukan jarak antara 2 simpul tersebut menggunakan sebuah fungsi eucleidianDistance yang terdapat dalam

file helper.py. Adapun informasi kordinat latitude dan longitude didapat dari aplikasi google maps.

4. Setelah itu program akan meminta dari user untuk menentukan simpul awal dan simpul akhir pada proses pencarian ini.
5. Setiap simpul tetangga yang disimpan bersamaan dengan jaraknya tadi dicari yang nilai fn ditambah nilai antara simpul itu ke goal atau katakan gn yang paling kecil akan dikunjungi terlebih dahulu.
6. Kemudian ketika mengunjungi sebuah simpul akan dianalisis lagi tetangganya dan diperhatikan fn+gn dari setiap tetangga lalu diambil simpul dengan nilai fn terkecil. Secara lebih jelas akan dijelaskan dalam sebuah kasus. Dalam contoh kasus ini hanya digunakan empat simpul untuk mempermudah visualisasi.

ASTAR
Input file test1 test2 test3 test4 test1 4.89 11.67 test2 7.88 90.98 test3 9.90 23.65 test4 10.00 40.00 MATRIKS 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0
1. Dipanggil fungsi readfile maka akan menghasilkan suatu dictionary sebagai berikut <code>{}{'test1': {'test2': 79.366, 'test4': 28.787}, 'test2': {'test1': 79.366}, 'test3': {'test1': 12.985, 'test4': 16.35}, 'test4': {'test3': 16.35}}}</code> nilai fn Adapun nilai gn yang diperoleh adalah sebagai berikut <code>{'test1': 79.366, 'test2': 0.0, 'test3': 67.36, 'test4': 51.024}</code> , nilai ini akan dijumlahkan dengan nilai fn yang paling kecil akan dikunjungi paling awal. 2. Atau dalam graf dapat digambarkan sebagai berikut



3. Artinya shortpath yang didapat adalah test1 -> test2

3.3 Algoritma menghitung distance

Algoritma ini menghitung jarak antara simpul awal dan simpul sebelumnya lalu disimpan di variabel `distance_sum`. Proses penjumlahan berhenti setelah semua simpul di `shortPath` terkunjungi.

3.4 Source Code dengan Bahasa Python

a. pencarian.py



```
from helper import *

# Pencarian dengan algoritma UCS
def UCS(loadDictionary, start, goal):
    try:
        # Penanganan eksepsi untuk simpul start dan goal
        if start not in loadDictionary or goal not in loadDictionary:
            raise ValueError("Start atau goal tidak ada dalam graf")

        # Inisialisasi variabel
        weight = {start: 0}
        simpulnotVisited = []
        visited = []
        curr = start
        simpulnotVisited.append([curr, weight[curr]])
        isTrue = True

        # Algoritma UCS
        while isTrue:
            curr = min(simpulnotVisited, key=lambda x: x[1])
            hasVisited = curr[0]
            #print(curr)
            visited.append(curr)
            simpulnotVisited.remove(curr)
            if visited[-1][0] == goal:
                isTrue = False

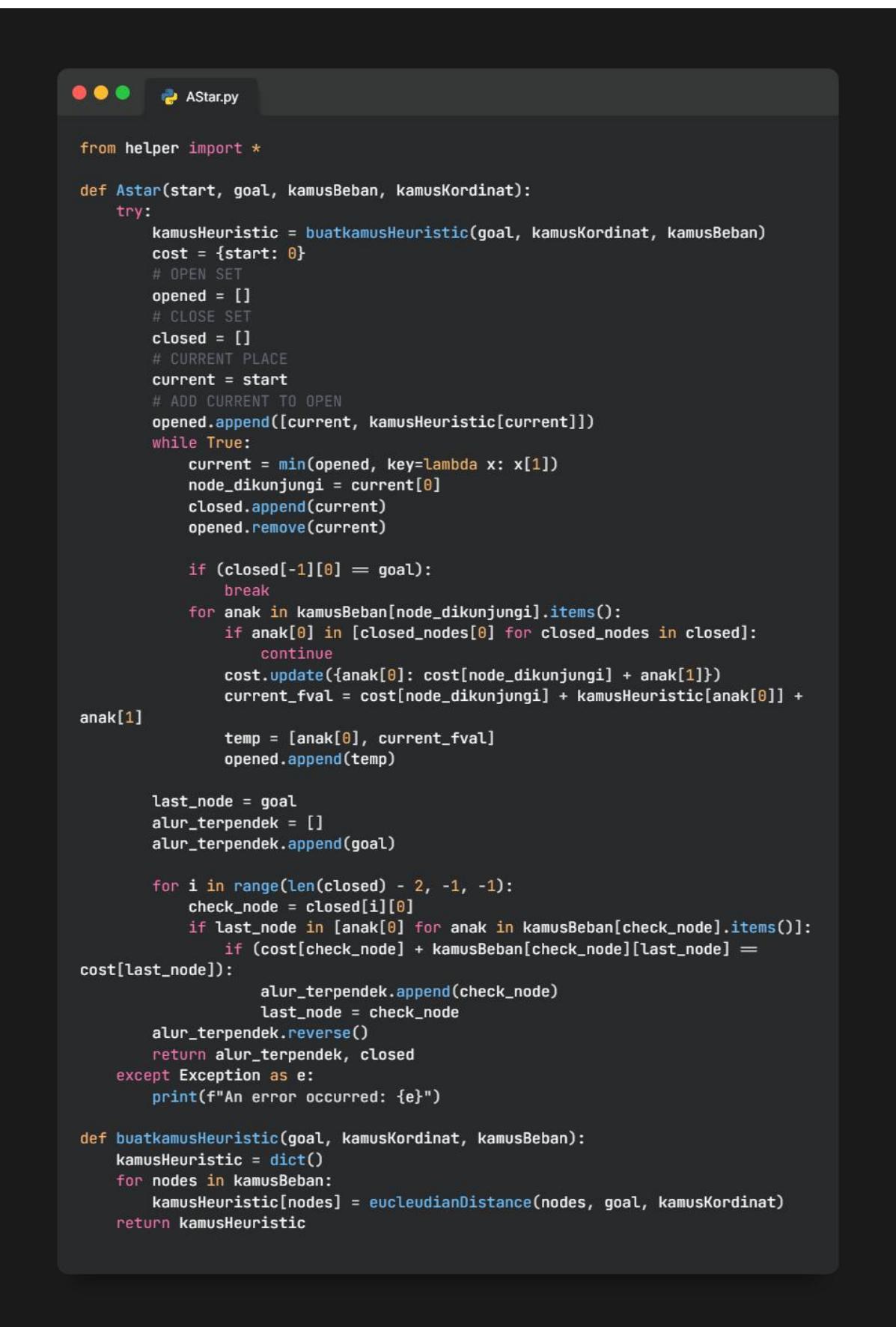
            # Mengecek apakah simpul sudah dikunjungi, jika sudah akan dilewat
            for i in loadDictionary[hasVisited].items():
                if i[0] in [j[0] for j in visited]:
                    continue
                weight[i[0]] = weight[hasVisited] + i[1]
                simpulnotVisited.append([i[0], weight[i[0]]])

        # Membangun jalur terpendek
        shortPath = []
        lastPath = goal
        shortPath.append(goal)
        for i in range(len(visited) - 2, -1, -1):
            check = visited[i][0]
            if lastPath in [k[0] for k in loadDictionary[check].items()]:
                if weight[check] + loadDictionary[check][lastPath] == weight[lastPath]:
                    shortPath.append(check)
                    lastPath = check
        shortPath.reverse()
        return shortPath, visited

    except ValueError as e:
        print("Error: " + str(e))

    except:
        print("Error : Coba Pastikan file dapat dibaca")
```

b. AStar.py



```
from helper import *

def Astar(start, goal, kamusBeban, kamusKordinat):
    try:
        kamusHeuristic = buatkamusHeuristic(goal, kamusKordinat, kamusBeban)
        cost = {start: 0}
        # OPEN SET
        opened = []
        # CLOSE SET
        closed = []
        # CURRENT PLACE
        current = start
        # ADD CURRENT TO OPEN
        opened.append([current, kamusHeuristic[current]])
        while True:
            current = min(opened, key=lambda x: x[1])
            node_dikunjungi = current[0]
            closed.append(current)
            opened.remove(current)

            if (closed[-1][0] == goal):
                break
            for anak in kamusBeban[node_dikunjungi].items():
                if anak[0] in [closed_nodes[0] for closed_nodes in closed]:
                    continue
                cost.update({anak[0]: cost[node_dikunjungi] + anak[1]})
                current_fval = cost[node_dikunjungi] + kamusHeuristic[anak[0]] +
anak[1]
                temp = [anak[0], current_fval]
                opened.append(temp)

        last_node = goal
        alur_terpendek = []
        alur_terpendek.append(goal)

        for i in range(len(closed) - 2, -1, -1):
            check_node = closed[i][0]
            if last_node in [anak[0] for anak in kamusBeban[check_node].items()]:
                if (cost[check_node] + kamusBeban[check_node][last_node] ==
cost[last_node]):
                    alur_terpendek.append(check_node)
                    last_node = check_node
        alur_terpendek.reverse()
        return alur_terpendek, closed
    except Exception as e:
        print(f"An error occurred: {e}")

def buatkamusHeuristic(goal, kamusKordinat, kamusBeban):
    kamusHeuristic = dict()
    for nodes in kamusBeban:
        kamusHeuristic[nodes] = euclideanDistance(nodes, goal, kamusKordinat)
    return kamusHeuristic
```

c. helper.py

```
import os
from collections import defaultdict
from os.path import dirname, abspath

def euclideanDistance(nodeAwal, nodeAkhir, coordinateDictionary):
    distance = round(((coordinateDictionary[nodeAwal]["lat"] -
    coordinateDictionary[nodeAkhir]["lat"]) ** 2
    + (coordinateDictionary[nodeAwal]["long"] - coordinateDictionary[nodeAkhir]
    ["long"]) ** 2) ** (0.5), 3)
    return distance

def readfile(namaFile):
    try:
        #inisialisasi awal
        coordinate = defaultdict(dict)
        weightDictionary = defaultdict(dict)

        #Baca File
        lokasiFile = namaFile

        with open(lokasiFile, "r") as f:
            #Mendapatkan list Seluruh Simpul
            listofSimpul = f.readline().strip().split(" ")
            nOfSimpul = len(listofSimpul)

            #Membentuk kamus kordinat
            location = f.readlines()
            kordinate = location[:location.index("MATRIKS\n")]
            for elem in kordinate:
                node = elem.strip().split(" ")
                coordinate[node[0]]["lat"] = float(node[1])
                coordinate[node[0]]["long"] = float(node[2])

            #Mengurus Adjacent Matriks
            mat = []
            Adjacent = location[location.index("MATRIKS\n") + 1:]
            for elemenAdjacency in Adjacent:
                elemen = elemenAdjacency.strip().split(" ")
                elemen = list(map(int, elemen))
                mat.append(elemen)

            for i in range(nOfSimpul):
                for j in range(nOfSimpul):
                    if mat[i][j] == 1:
                        weightDictionary[listofSimpul[i]][listofSimpul[j]] =
euclideanDistance(
                            listofSimpul[i], listofSimpul[j], coordinate
                        )

    return weightDictionary, coordinate

except FileNotFoundError:
    print("File tidak ditemukan. Silakan periksa kembali nama file yang
dimasukkan.")
    return None
```

d. Visualizer.py



```
Visualizer.py

import matplotlib.pyplot as plt
import networkx as nx
from helper import *
from pencarian import *
from AStar import *

def membuatGraph(start, goal, alur_terpendek, visited, kamusBeban,
kamusKoordinat):
    try:
        # Inisialisasi Graph
        G = nx.Graph()

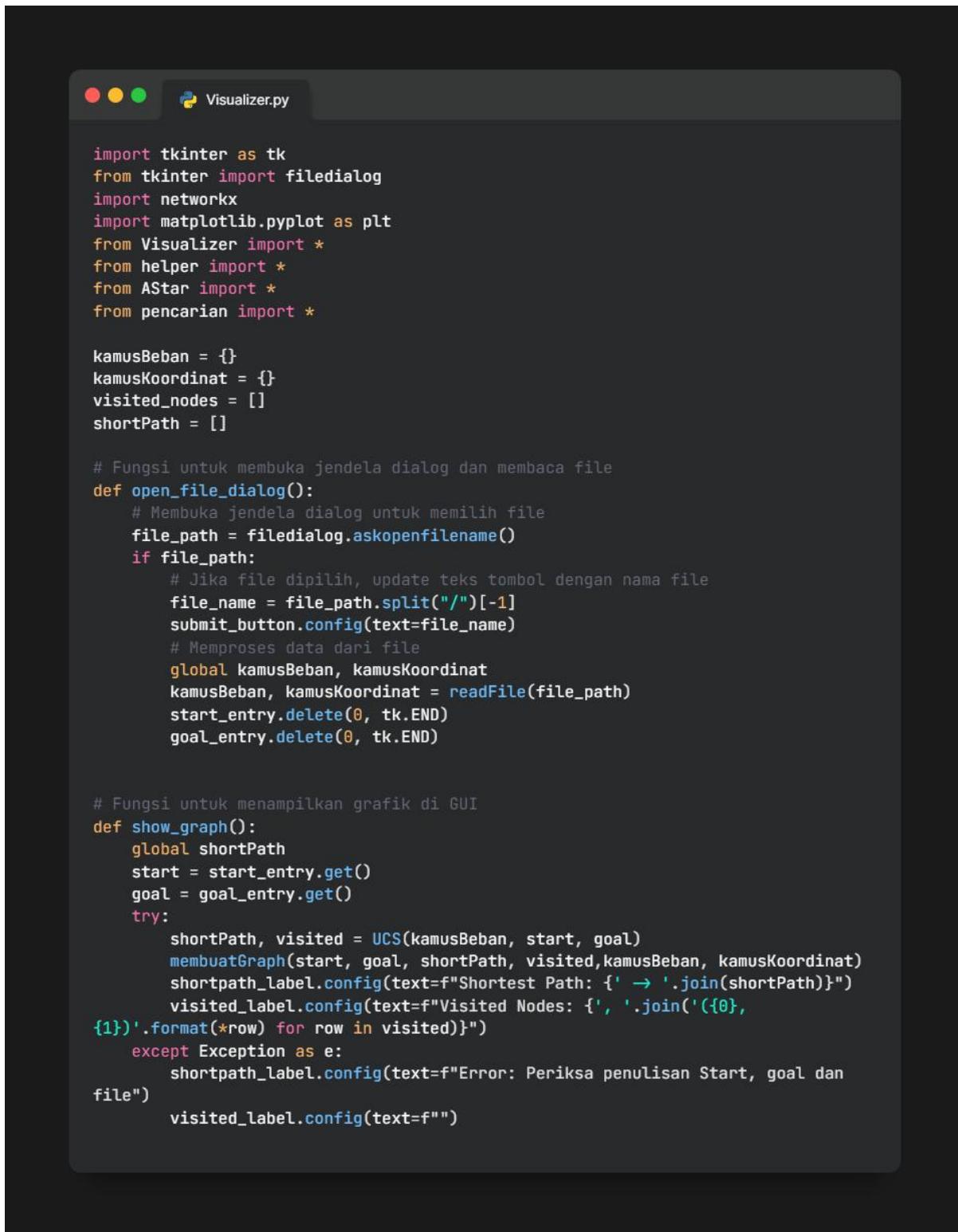
        # Bentuk semua nodes
        for nodes in kamusBeban:
            G.add_node(nodes, pos=(kamusKoordinat[nodes]['lat'],
kamusKoordinat[nodes]['long']))

        # Bentuk semua edges
        for nodes in kamusBeban:
            for children in kamusBeban[nodes]:
                G.add_edge(nodes, children, weight=kamusBeban[nodes][children])

        # Animasi penjelajahan semua node visited
        fig, ax = plt.subplots()
        for i in range(len(visited)):
            node = visited[i]
            node_color = ["blue" if n == node or n in alur_terpendek else "red"
for n in G.nodes]
            nx.draw(G, nx.get_node_attributes(G, 'pos'), node_color=node_color,
with_labels=True, node_size=1200, ax=ax)
            ax.set_title(f"Visiting {node}")
            fig.canvas.draw()
            plt.pause(0.5)

        plt.show()
    except Exception as e:
        print(f"Terjadi kesalahan: {e}")
```

e. gui.py



The screenshot shows a dark-themed code editor window titled "Visualizer.py". The code is written in Python and defines a class with methods for opening files, displaying graphs, and managing visited nodes.

```
import tkinter as tk
from tkinter import filedialog
import networkx
import matplotlib.pyplot as plt
from Visualizer import *
from helper import *
from AStar import *
from pencarian import *

kamusBeban = {}
kamusKoordinat = {}
visited_nodes = []
shortPath = []

# Fungsi untuk membuka jendela dialog dan membaca file
def open_file_dialog():
    # Membuka jendela dialog untuk memilih file
    file_path = filedialog.askopenfilename()
    if file_path:
        # Jika file dipilih, update teks tombol dengan nama file
        file_name = file_path.split("/")[-1]
        submit_button.config(text=file_name)
        # Memproses data dari file
        global kamusBeban, kamusKoordinat
        kamusBeban, kamusKoordinat = readFile(file_path)
        start_entry.delete(0, tk.END)
        goal_entry.delete(0, tk.END)

# Fungsi untuk menampilkan grafik di GUI
def show_graph():
    global shortPath
    start = start_entry.get()
    goal = goal_entry.get()
    try:
        shortPath, visited = UCS(kamusBeban, start, goal)
        membuatGraph(start, goal, shortPath, visited, kamusBeban, kamusKoordinat)
        shortpath_label.config(text=f"Shortest Path: {' → '.join(shortPath)}")
        visited_label.config(text=f"Visited Nodes: {', '.join(['{0}, {1}'.format(*row) for row in visited])}")
    except Exception as e:
        shortpath_label.config(text=f"Error: Periksa penulisan Start, goal dan file")
    visited_label.config(text="")
```

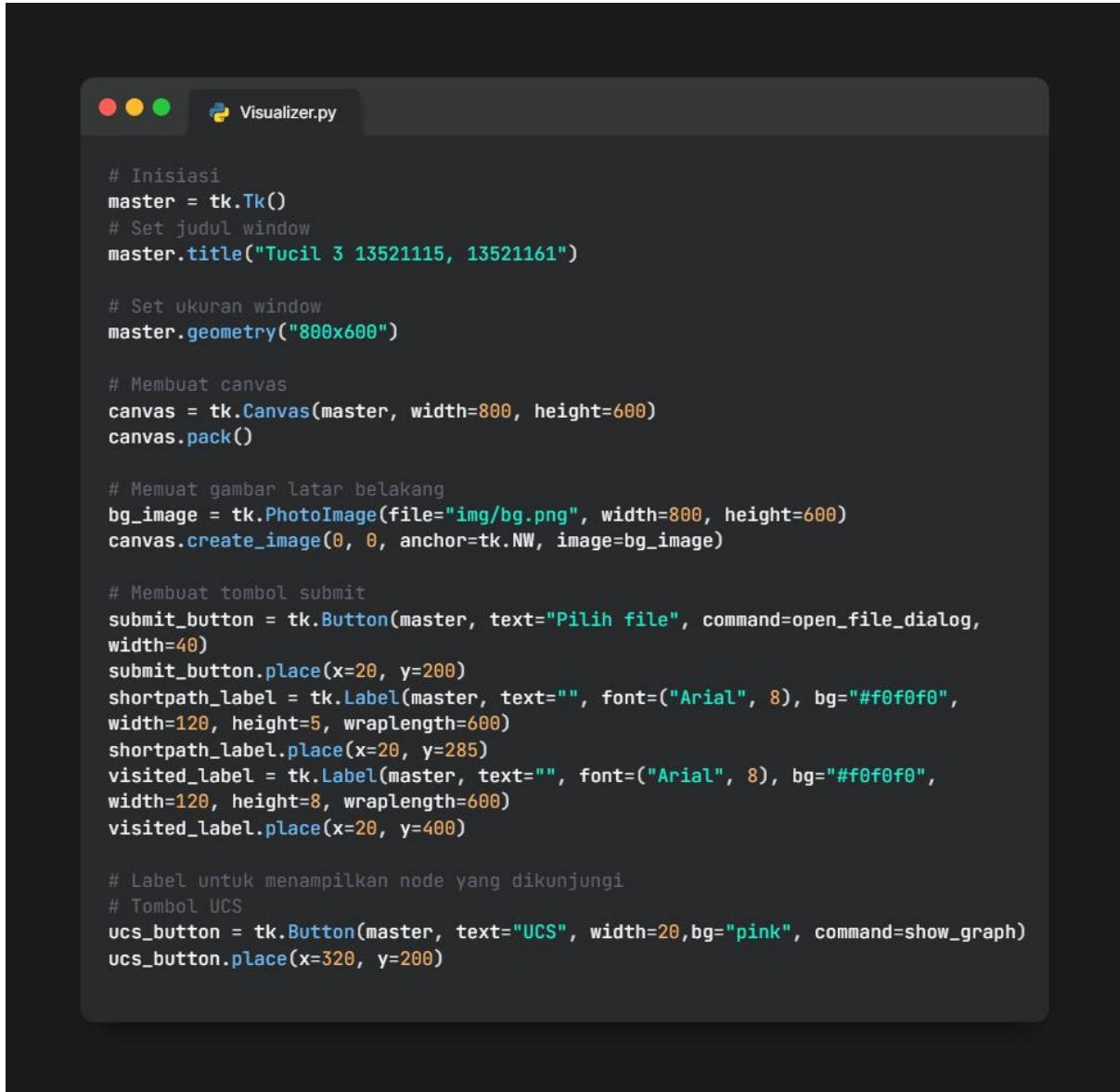
```
# Update label untuk menampilkan jalur terpendek
def showASTAR_graph():
    global shortPath
    start = start_entry.get()
    goal = goal_entry.get()
    try:
        # Memanggil fungsi Astar untuk mendapatkan jalur terpendek dan node yang dikunjungi
        shortPath, visited = Astar(start, goal, kamusBeban, kamusKoordinat)

        # Menampilkan grafik dengan memanggil fungsi membuatGraph dari modul Visualizer
        membuatGraph(start, goal, shortPath, visited, kamusBeban, kamusKoordinat)

        # Menampilkan hasil jalur terpendek dan node yang dikunjungi di GUI
        shortpath_label.config(text=f"Shortest Path: {' → '.join(shortPath)}")
        visited_label.config(text=f"Visited Nodes: {', '.join('{0}, {1}'.format(*row) for row in visited)}")
    except Exception as e:
        # Menampilkan pesan error jika terjadi kesalahan saat proses pencarian jalur terpendek
        shortpath_label.config(text=f"Error: Periksa penulisan Start, goal dan file")
        visited_label.config(text=f"")

def disstance_sum():
    distance_sum = 0
    for i in range(len(shortPath) - 1):
        distance_sum += kamusBeban[shortPath[i]][shortPath[i + 1]]
    return distance_sum

# Fungsi untuk menampilkan distance sum
def show_distance_sum():
    try:
        distance_sum = disstance_sum()
        distance_sum_label.config(text=f"{distance_sum}")
    except Exception as e:
        distance_sum_label.config(text=f"Error: {e}")
```



```
# Inisiasi
master = tk.Tk()
# Set judul window
master.title("Tucil 3 13521115, 13521161")

# Set ukuran window
master.geometry("800x600")

# Membuat canvas
canvas = tk.Canvas(master, width=800, height=600)
canvas.pack()

# Memuat gambar latar belakang
bg_image = tk.PhotoImage(file="img/bg.png", width=800, height=600)
canvas.create_image(0, 0, anchor=tk.NW, image=bg_image)

# Membuat tombol submit
submit_button = tk.Button(master, text="Pilih file", command=open_file_dialog,
width=40)
submit_button.place(x=20, y=200)
shortpath_label = tk.Label(master, text="", font=("Arial", 8), bg="#f0f0f0",
width=120, height=5, wraplength=600)
shortpath_label.place(x=20, y=285)
visited_label = tk.Label(master, text="", font=("Arial", 8), bg="#f0f0f0",
width=120, height=8, wraplength=600)
visited_label.place(x=20, y=400)

# Label untuk menampilkan node yang dikunjungi
# Tombol UCS
ucs_button = tk.Button(master, text="UCS", width=20, bg="pink", command=show_graph)
ucs_button.place(x=320, y=200)
```

```
# Tombol A*
astar_button = tk.Button(master, text="A*", width=20, bg="pink",
command=showASTAR_graph)
astar_button.place(x=480, y=200)

# Label start
start_label = tk.Label(master, text="Start Node", font=("Arial", 12),
bg="#f0f0f0")
start_label.place(x=20, y=230)

# Field start
start_entry = tk.Entry(master, font=("Arial", 12))
start_entry.place(x=20, y=255, width=200)

# Label goal
goal_label = tk.Label(master, text="Goal Node", font=("Arial", 12), bg="#f0f0f0")
goal_label.place(x=320, y=230)

# Field goal
goal_entry = tk.Entry(master, font=("Arial", 12))
goal_entry.place(x=320, y=255, width=200)

# Label untuk menampilkan distance sum
distance_sum_label = tk.Label(master, text="", font=("Arial", 12), bg="#f0f0f0",
width=80)
distance_sum_label.place(x=20, y=560)

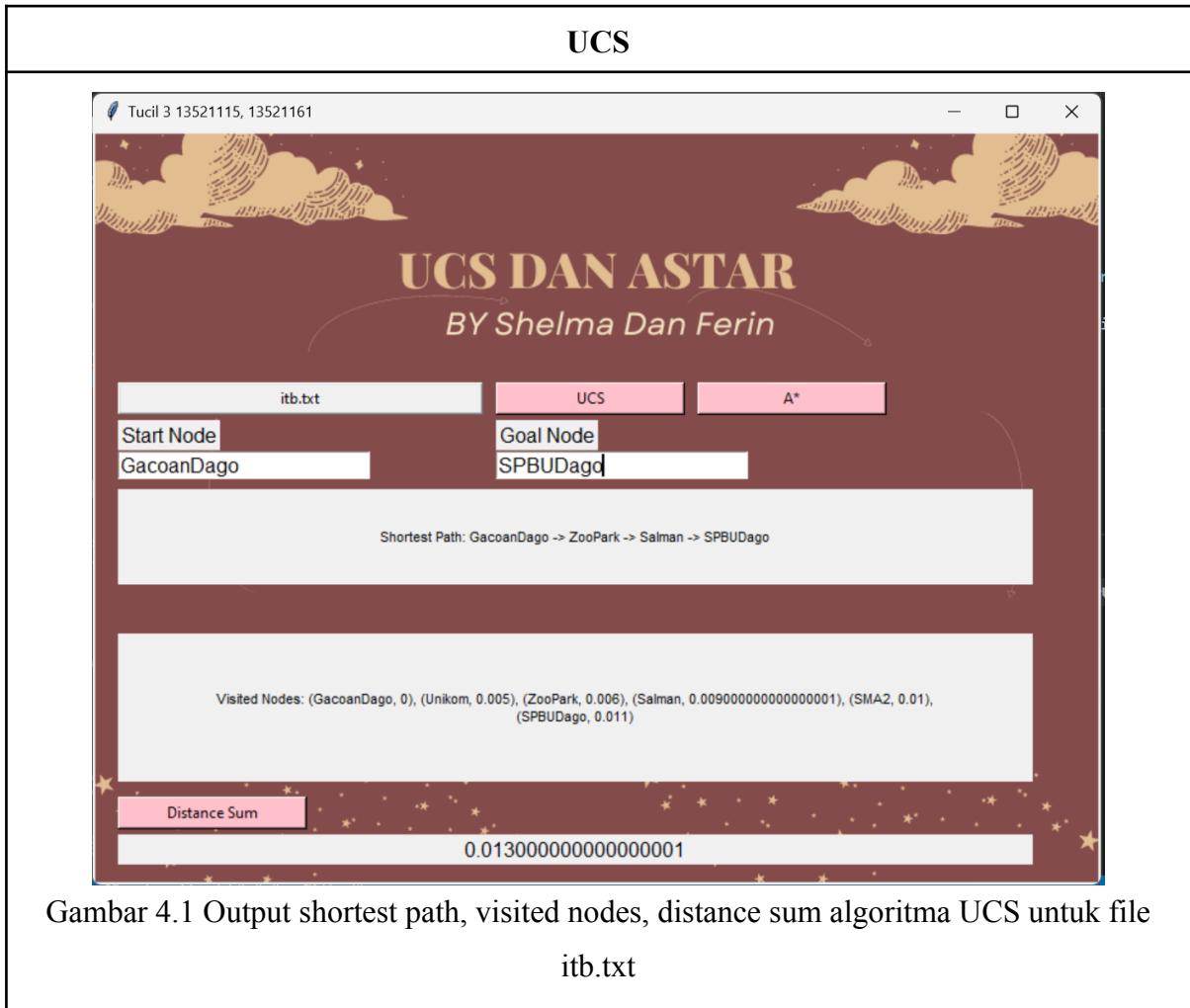
# Tombol untuk menampilkan distance sum
distance_sum_button = tk.Button(master, text="Distance Sum", width=20, bg="pink",
command=show_distance_sum)
distance_sum_button.place(x=20, y=530)

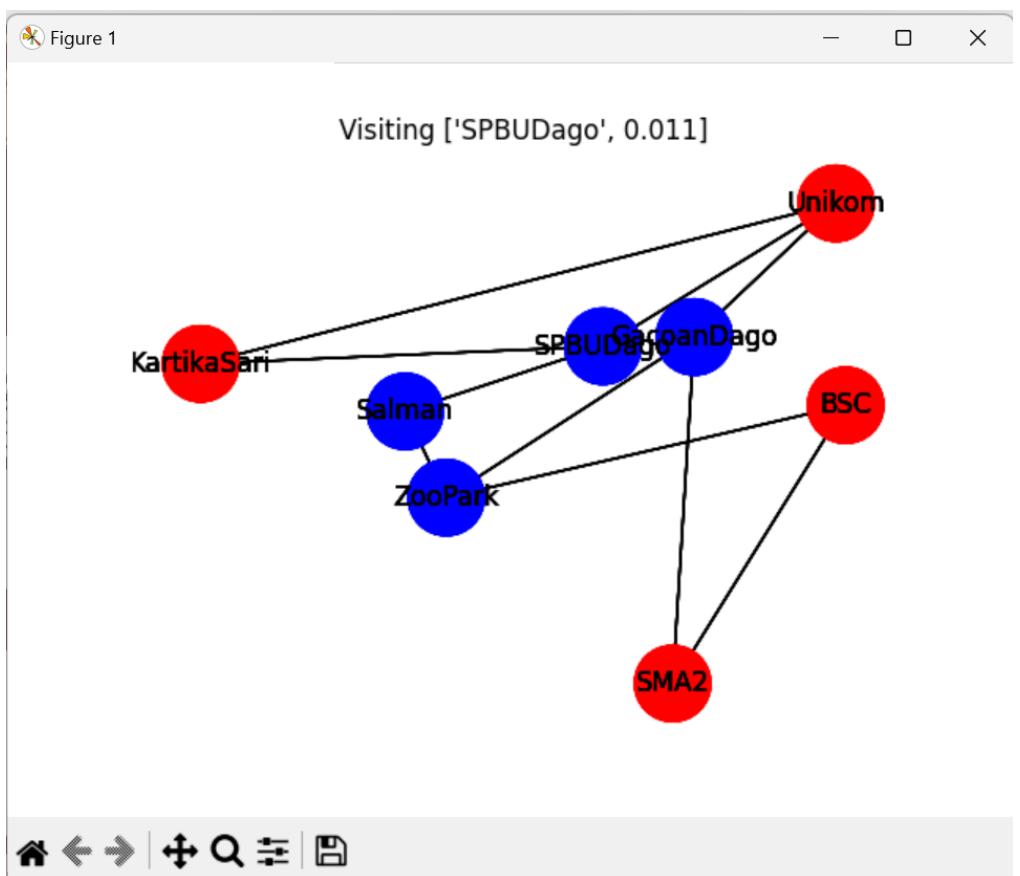
# Menjalankan event loop
tk.mainloop()
```

BAB 4

PENGUJIAN

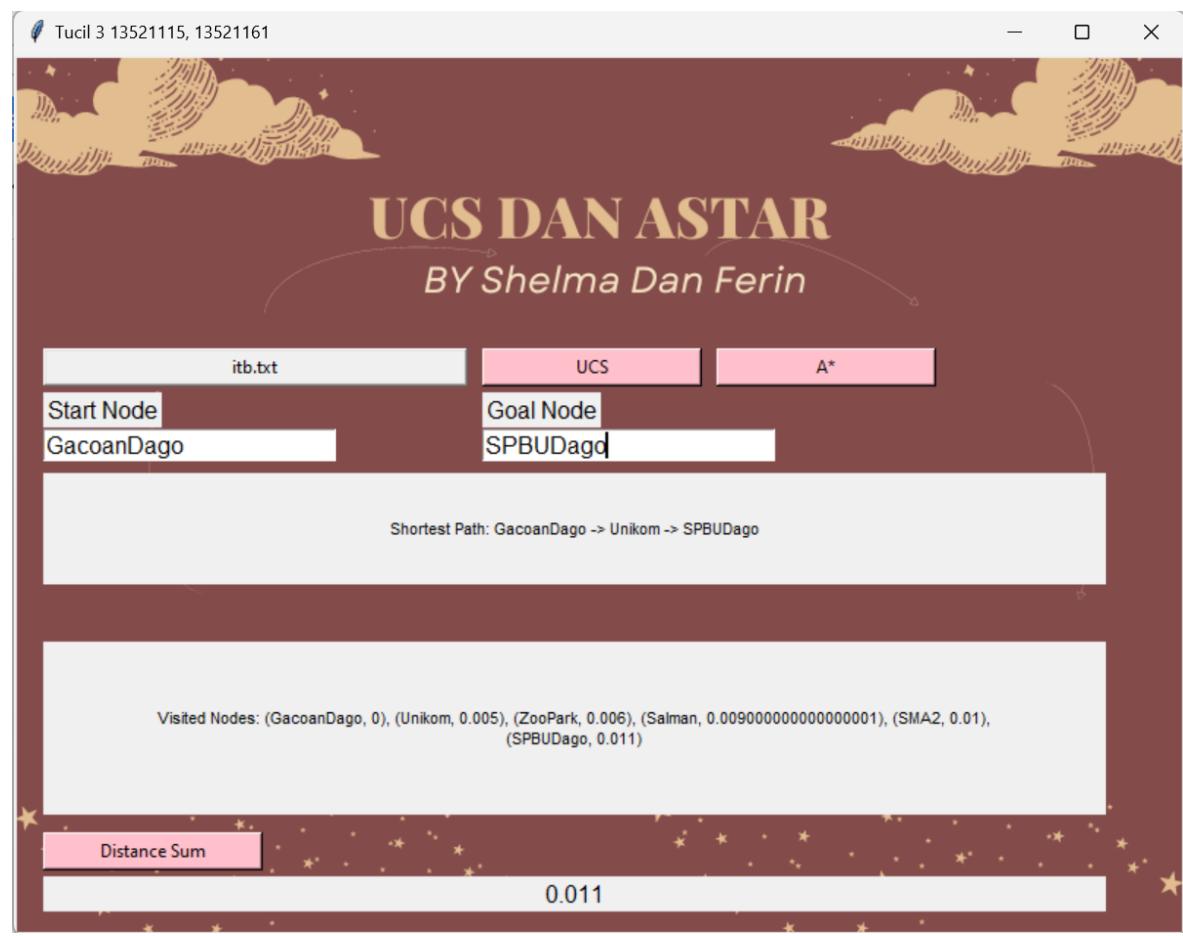
4.1 Test case untuk Peta jalan sekitar kampus ITB/Dago/Bandung Utara



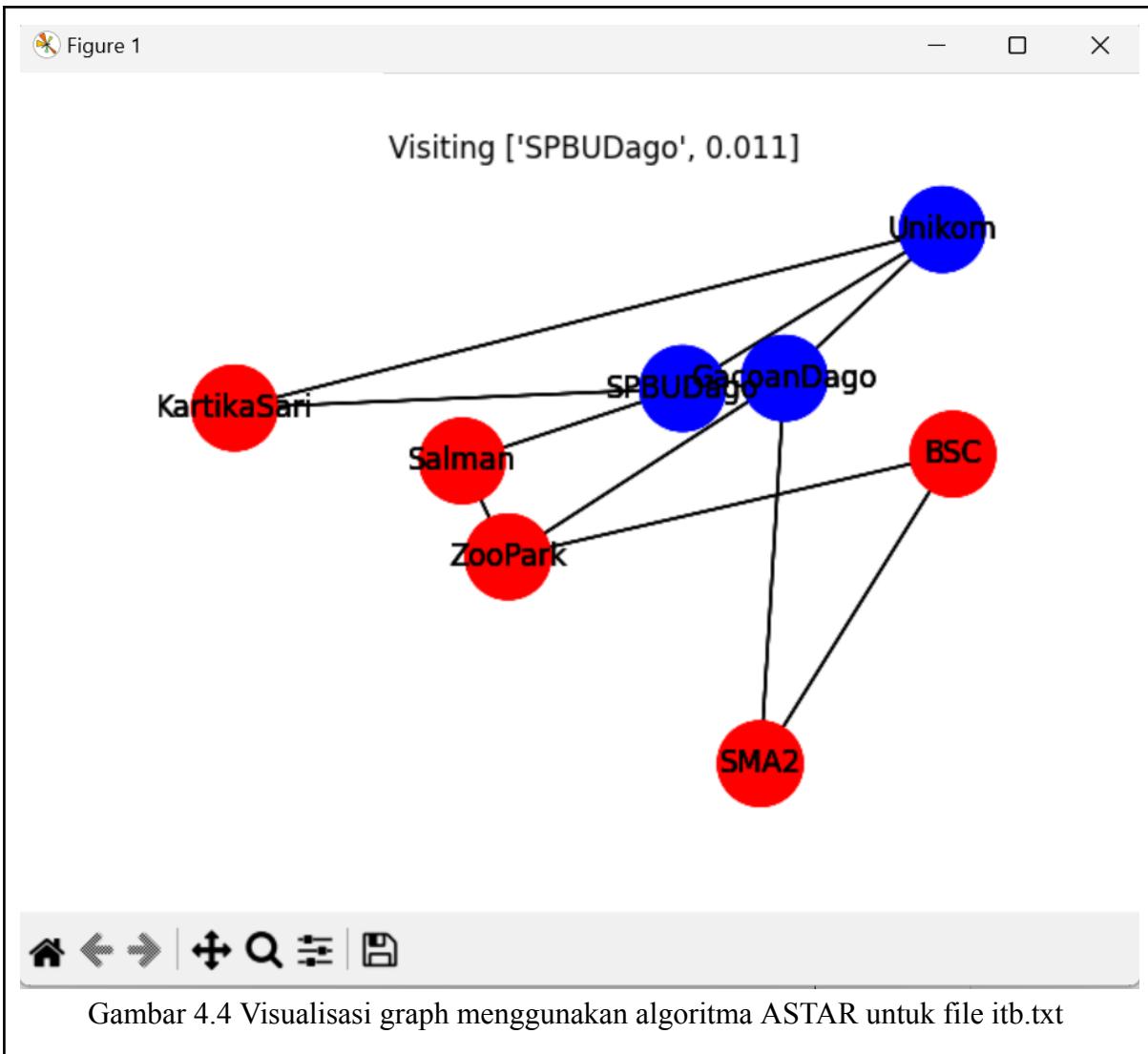


Gambar 4.2 Visualisasi graph menggunakan algoritma UCS untuk file itb.txt

ASTAR



Gambar 4.3 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file itb.txt



Dari hasil implementasi ini dapat disimpulkan bahwa ASTAR lebih baik dari UCS hal ini bisa dilihat dari banyaknya simpul yang dikunjungi oleh UCS lebih banyak dari ASTAR serta jarak yang didapat juga lebih kecil ASTAR dibanding UCS.

4.2 Test case untuk Peta jalan sekitar Alun-alun Bandung

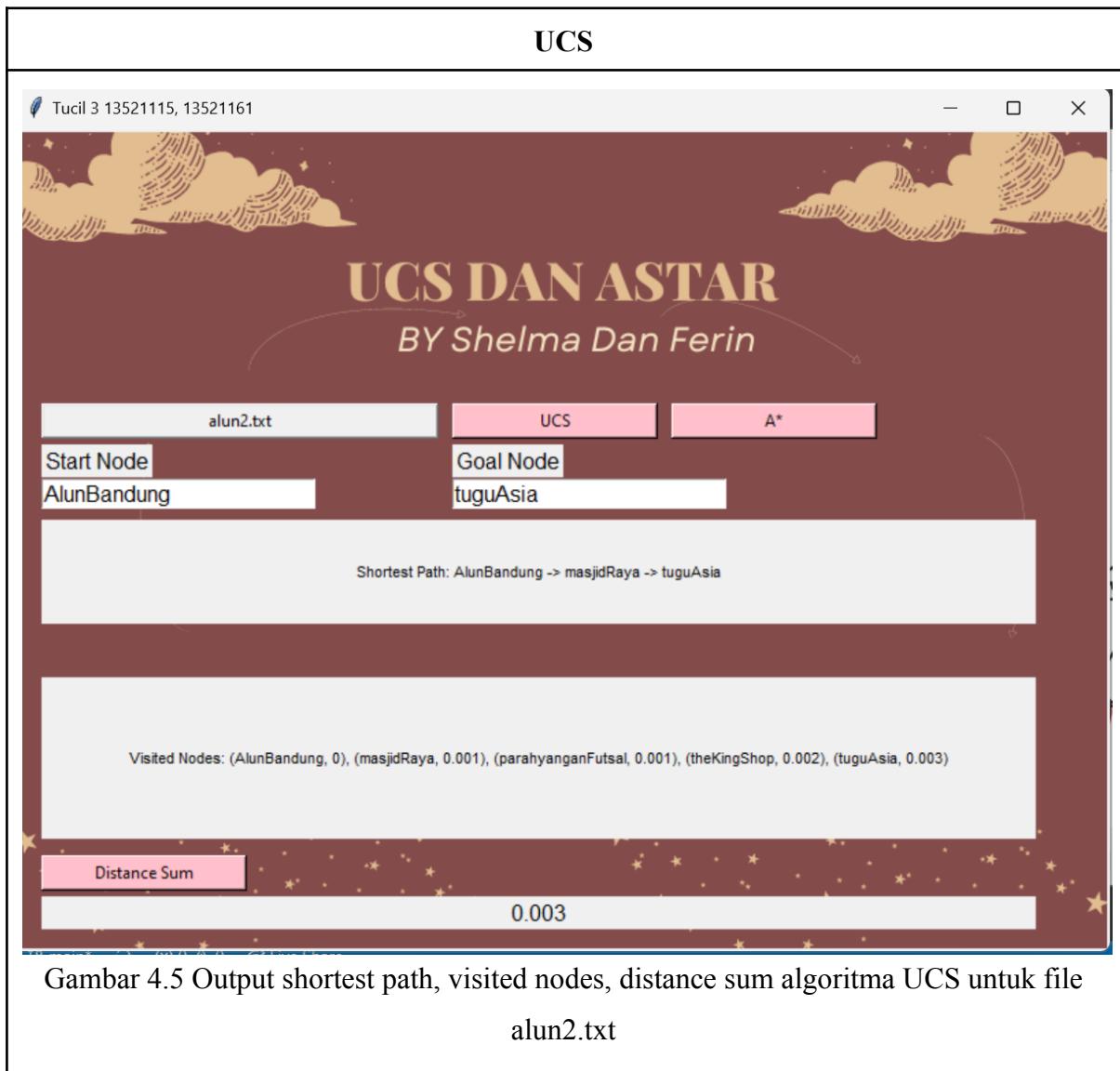
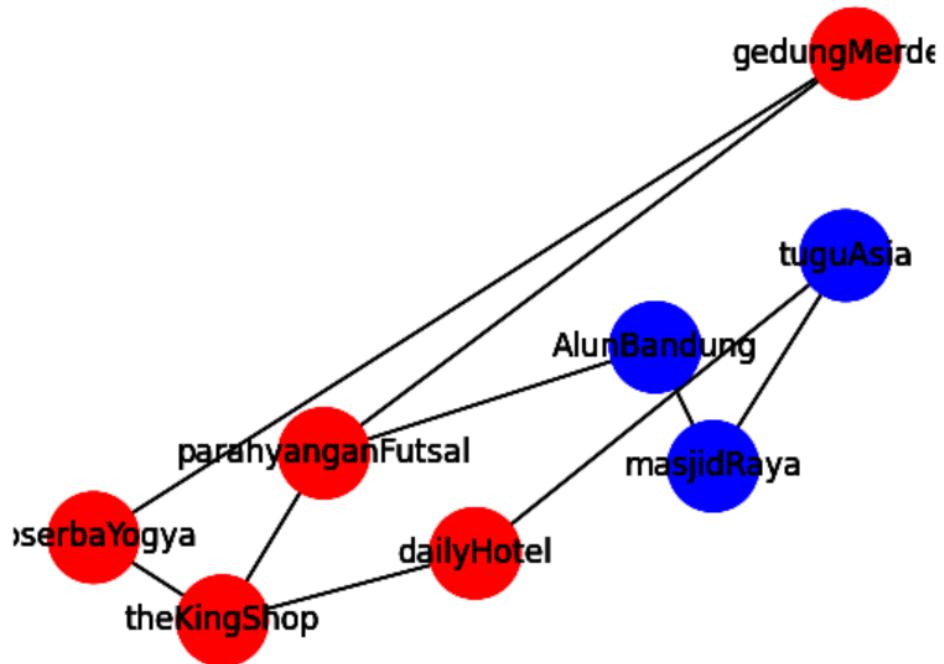


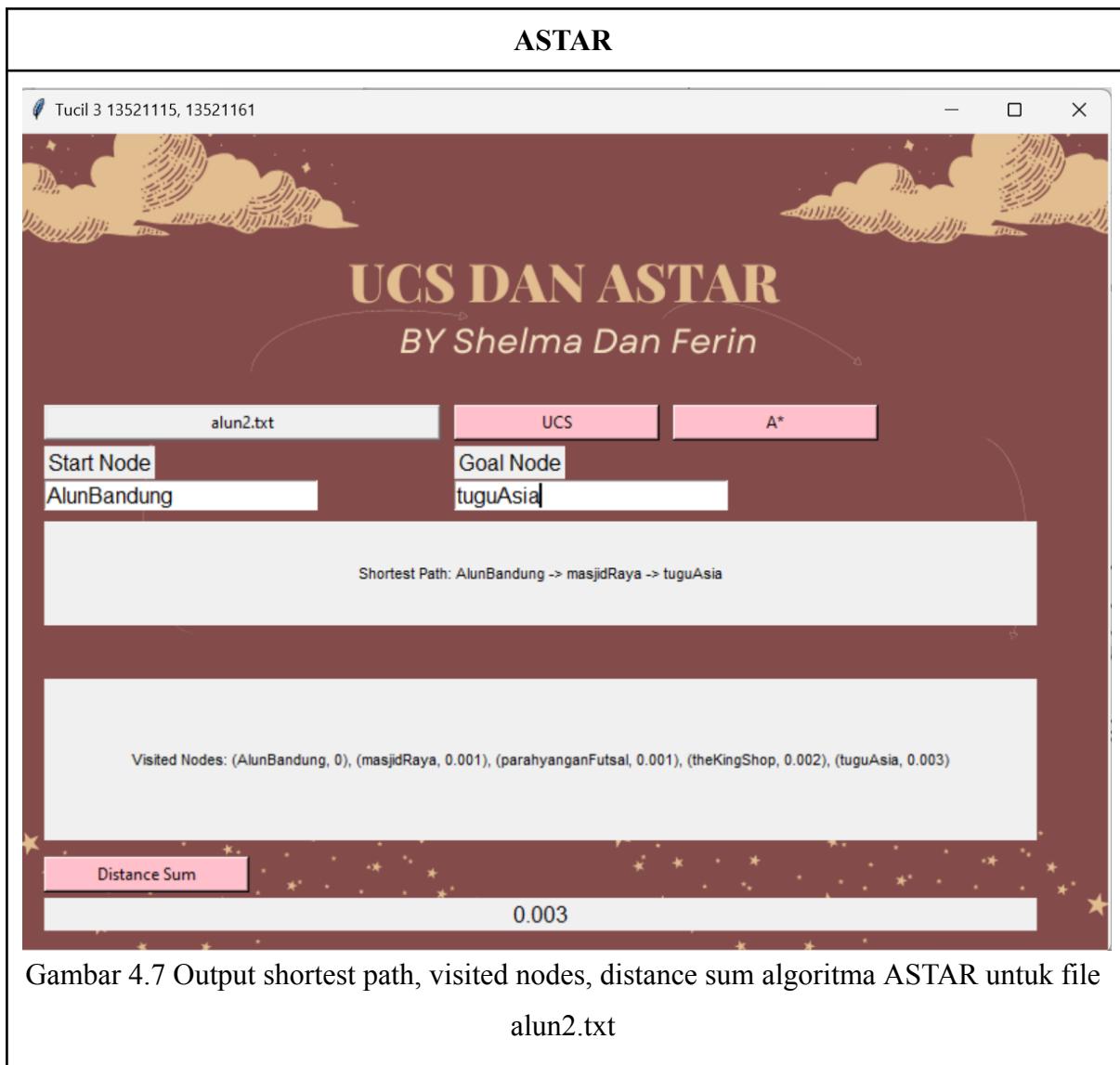


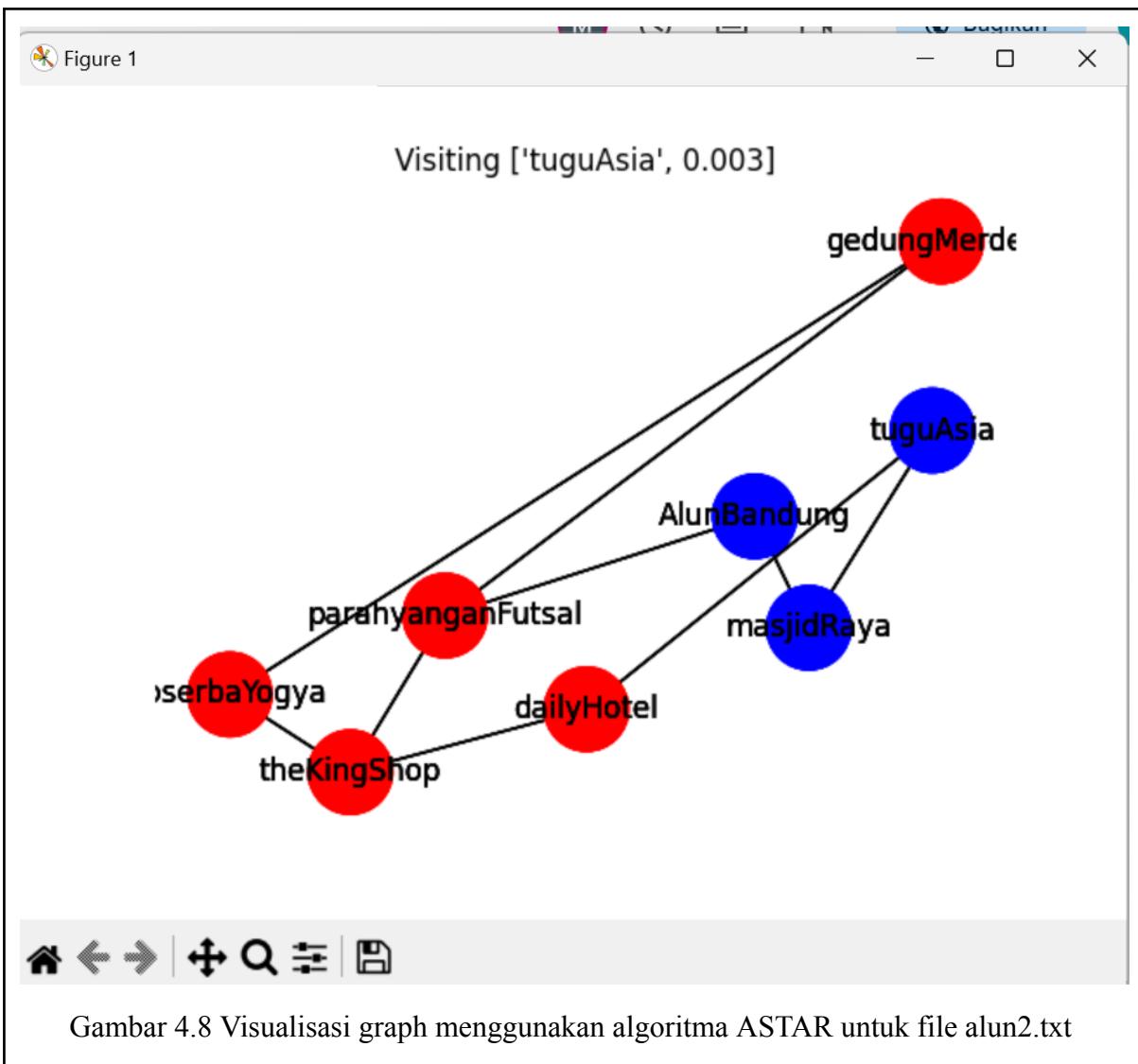
Figure 1

Visiting ['tuguAsia', 0.003]



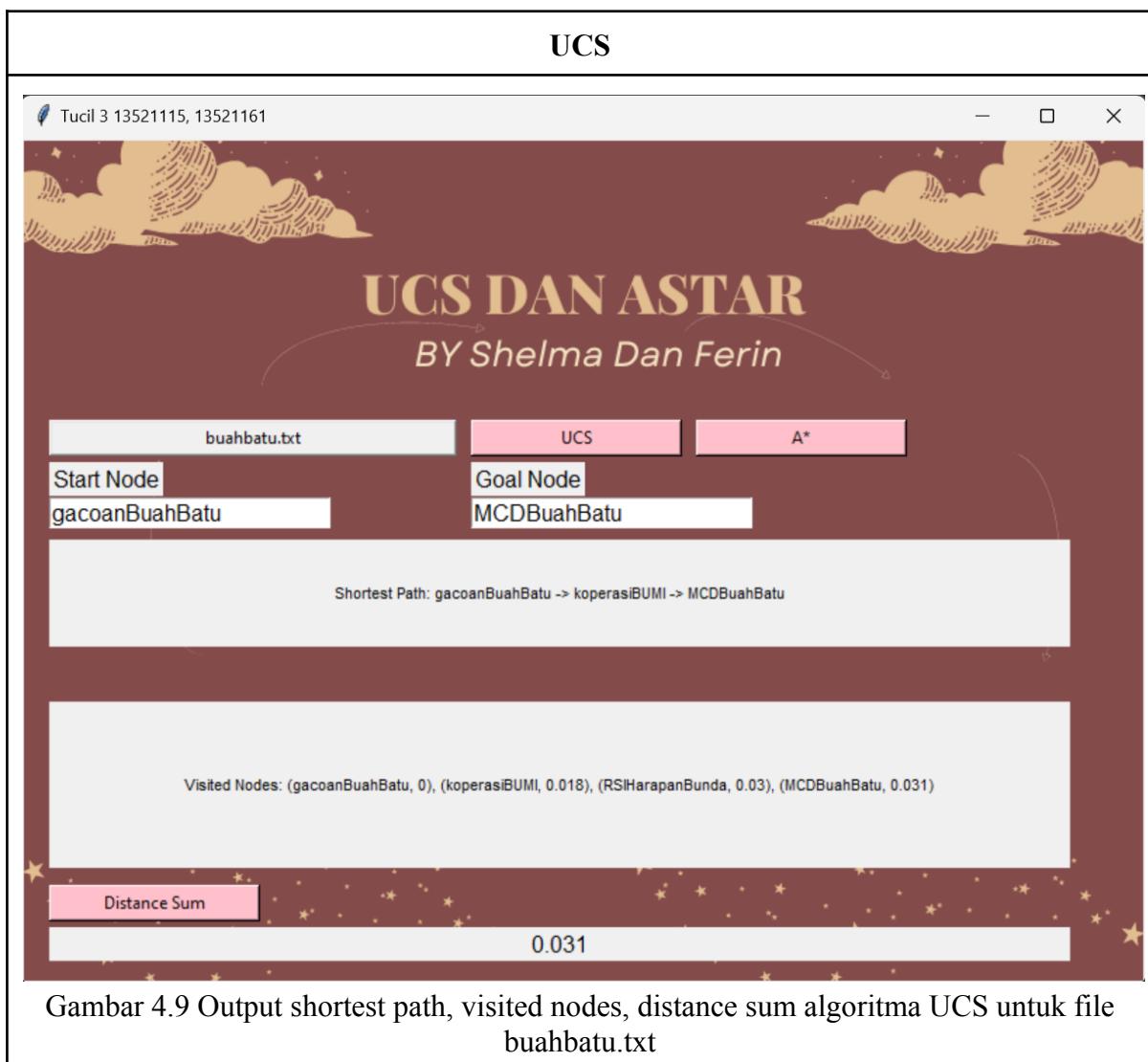
Gambar 4.6 Visualisasi graph menggunakan algoritma UCS untuk file alun2.txt

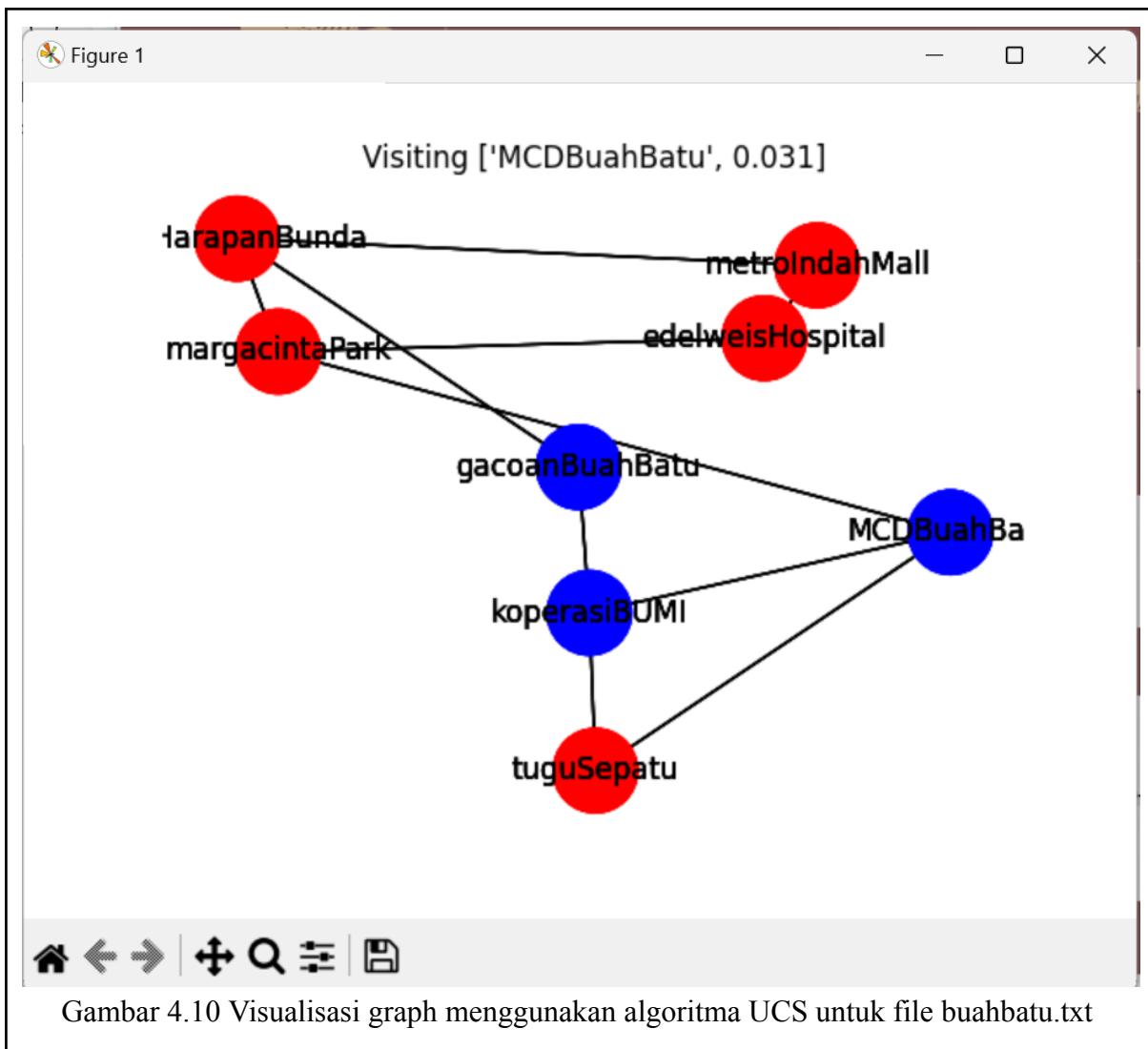




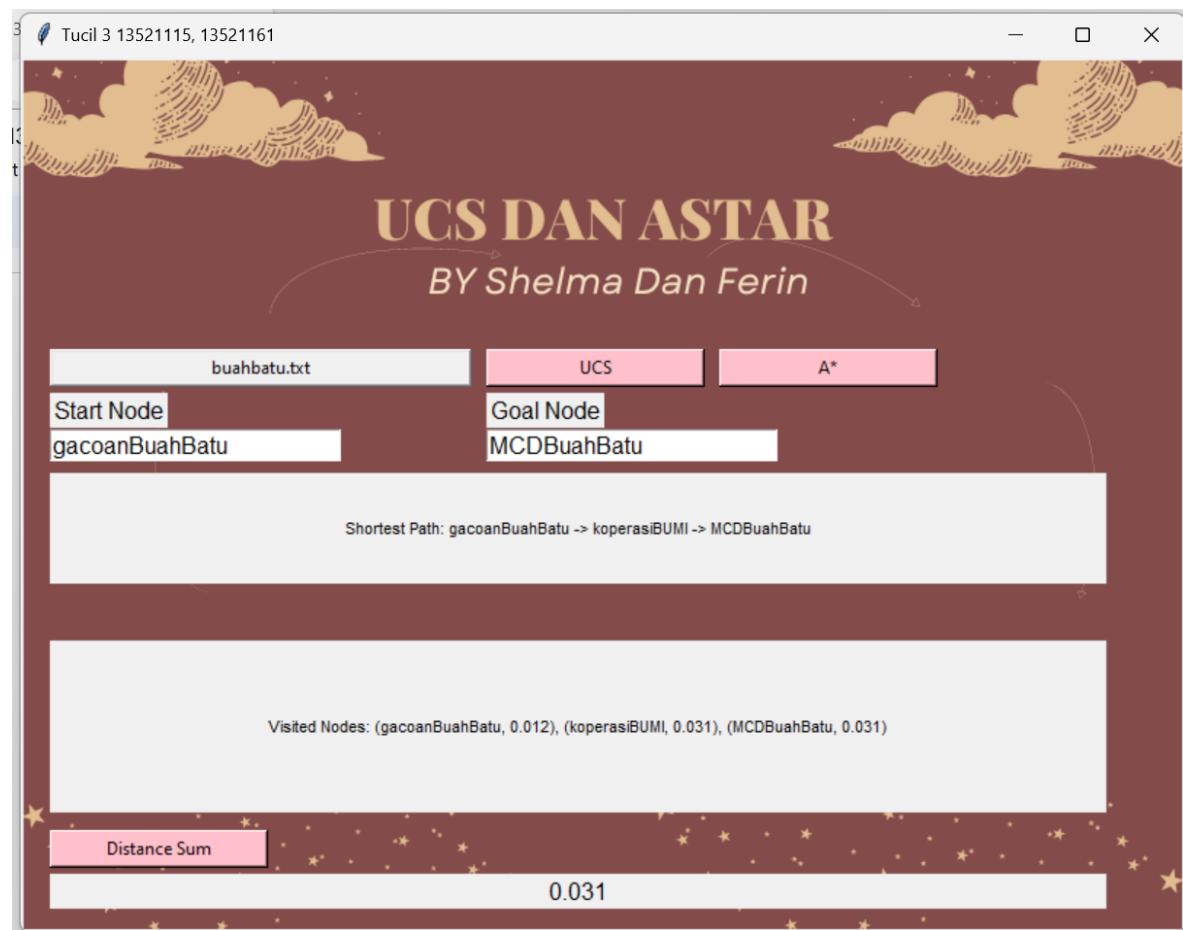
Dari hasil implementasi ini dapat disimpulkan bahwa ASTAR sama dengan UCS hal ini bisa dilihat dari banyaknya simpul yang dikunjungi oleh UCS sama dari ASTAR serta jarak yang didapat juga sama antara ASTAR dan UCS.

4.3 Test case untuk Peta jalan sekitar Buahbatu atau Bandung Selatan

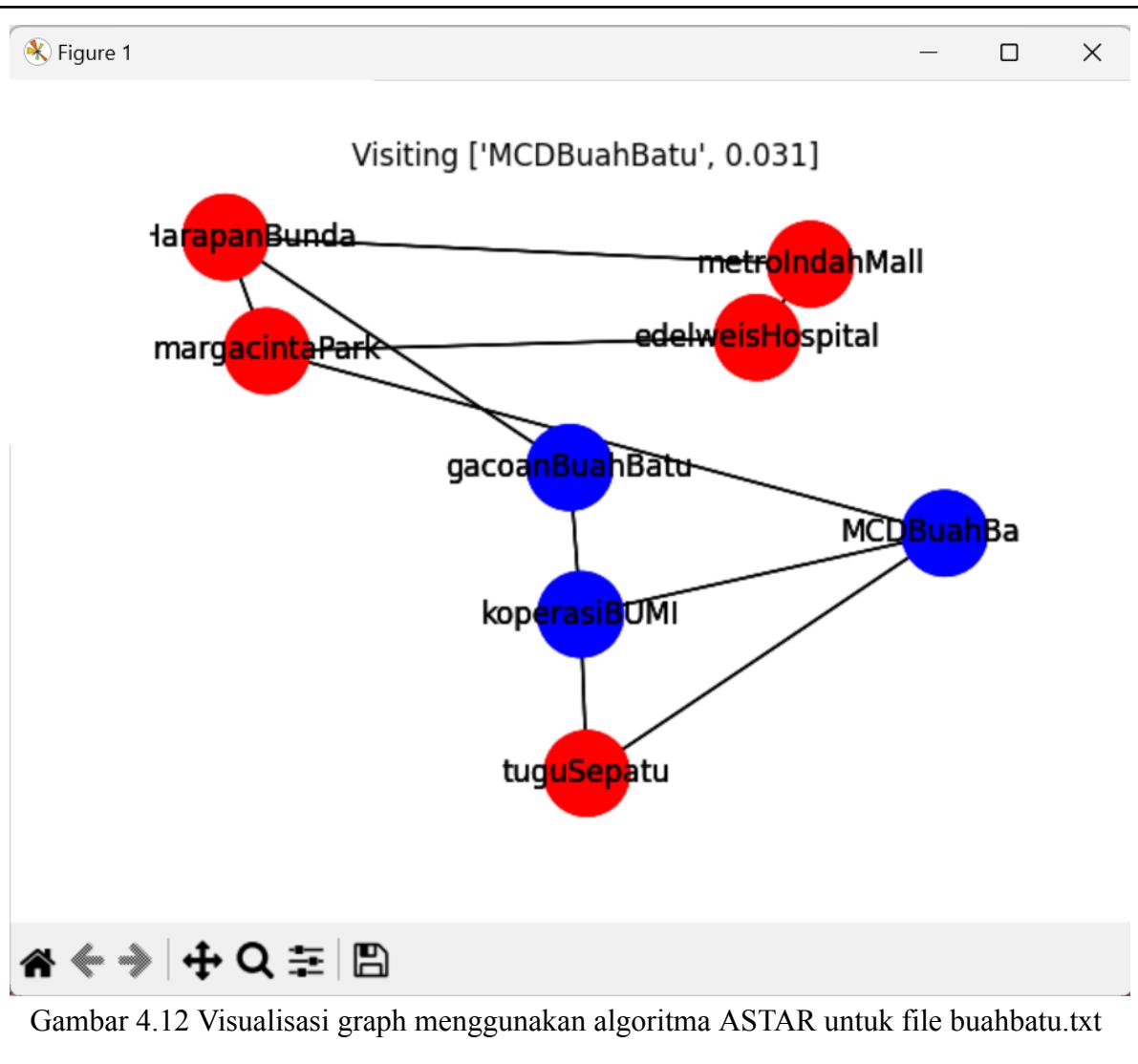




ASTAR

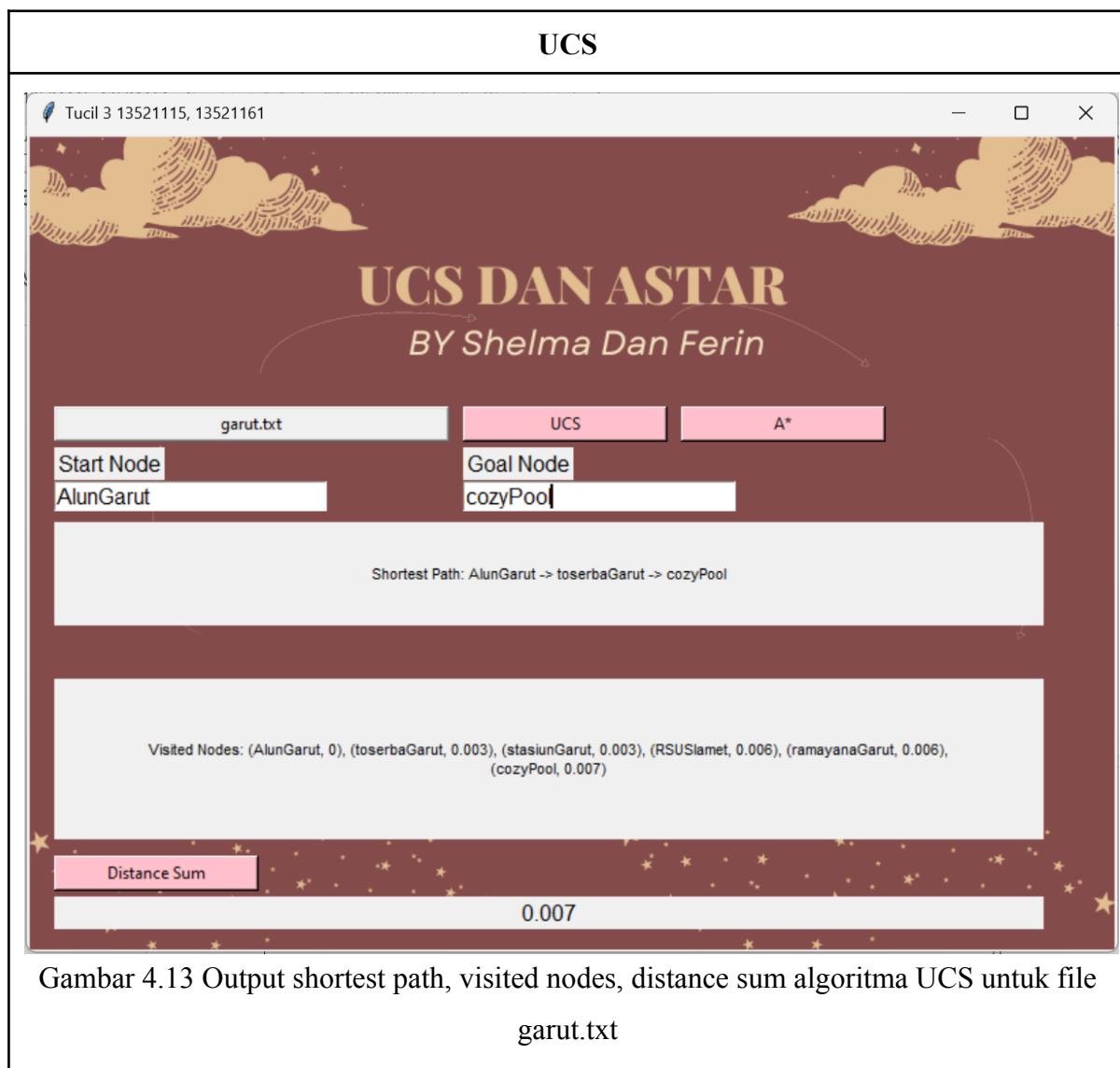


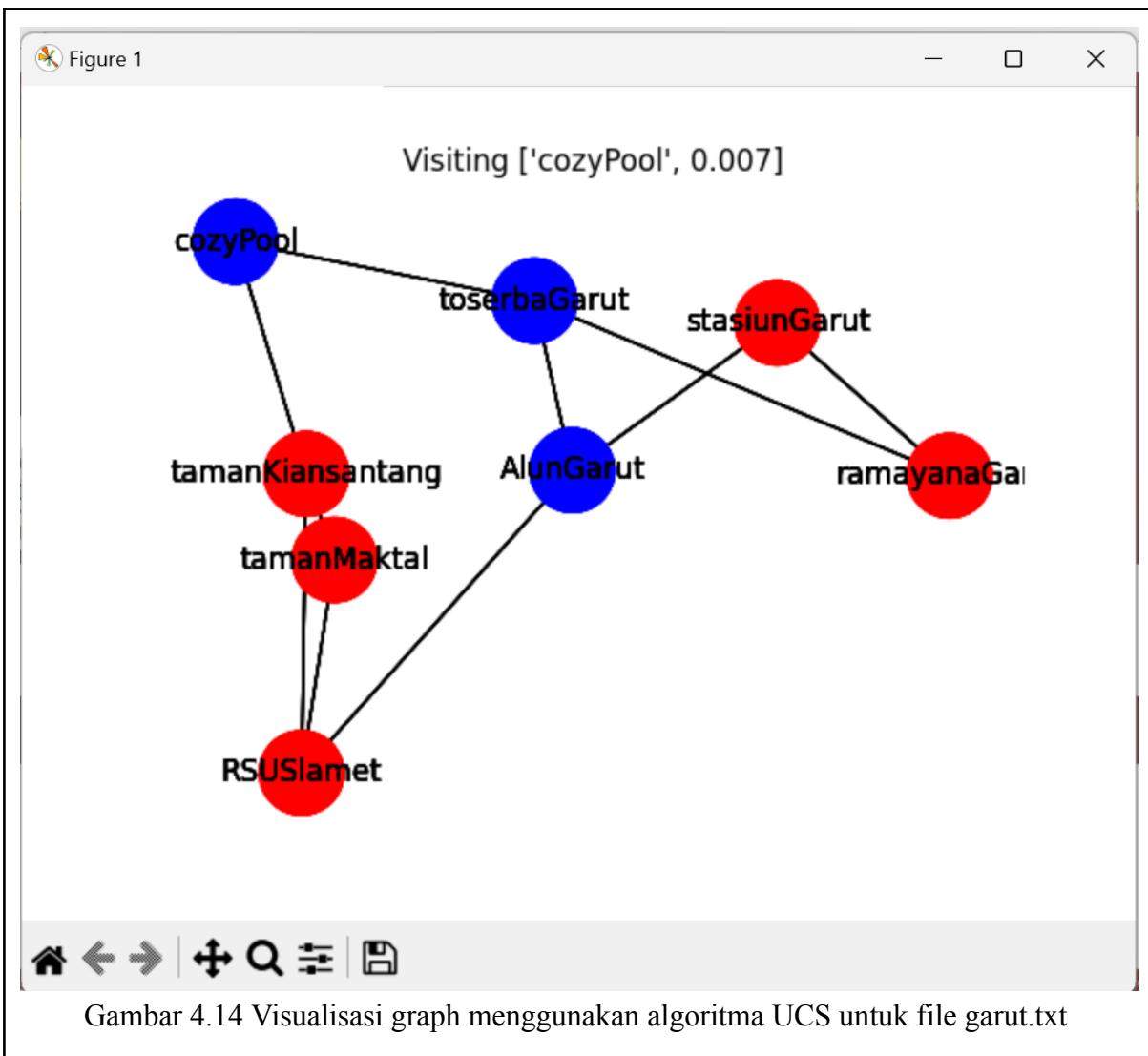
Gambar 4.11 Output shortest path, visited nodes, distance sum algoritma ASTAR untuk file buahbatu.txt



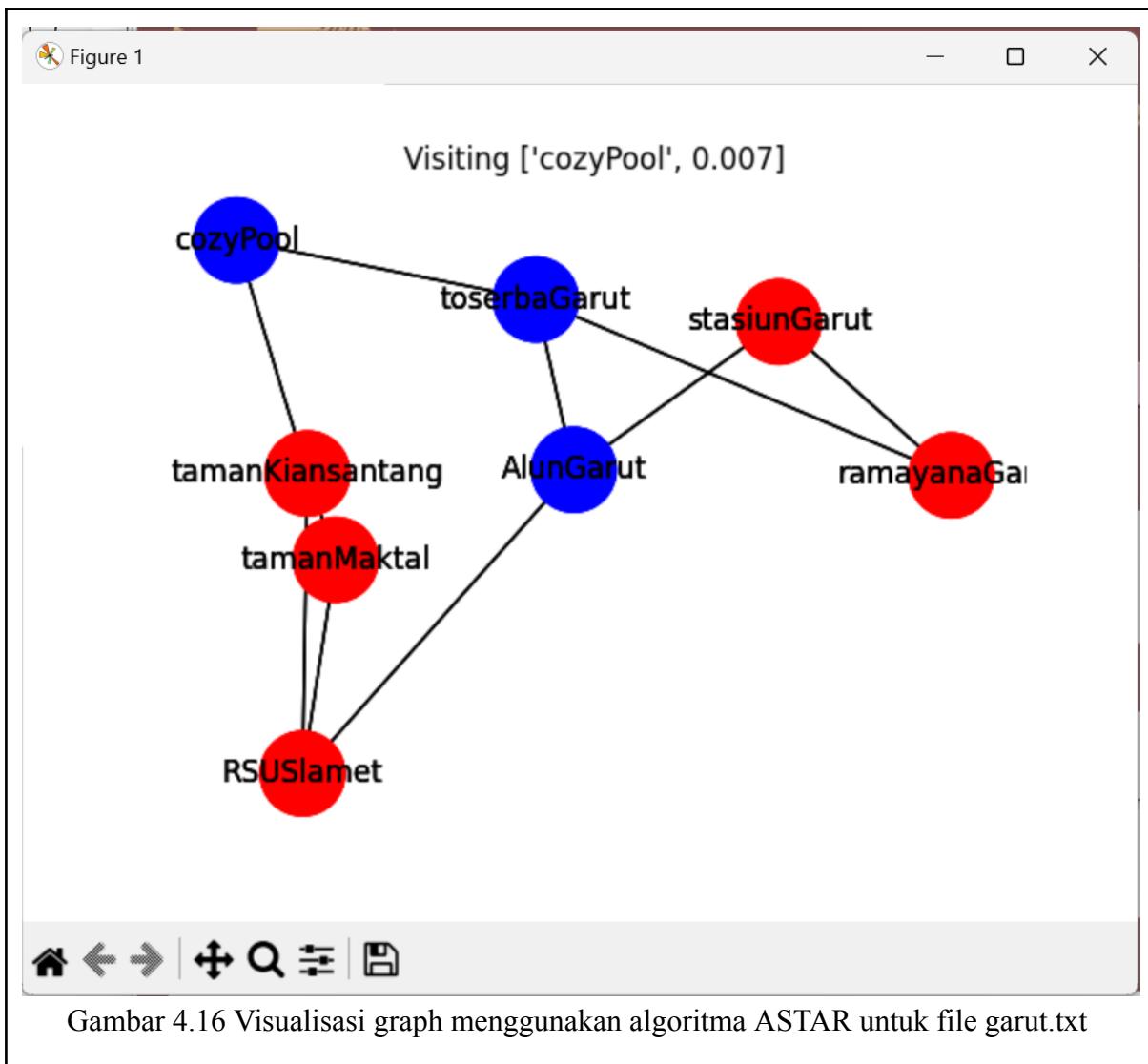
Dari hasil implementasi ini dapat disimpulkan bahwa ASTAR lebih baik dari UCS hal ini bisa dilihat dari banyaknya simpul yang dikunjungi oleh UCS lebih banyak dari ASTAR walaupun jarak yang mereka tempuh sama. karena melewati shortpath yang sama.

4.4 Test Case Untuk Peta Jalan Sebuah Kawasan di Kota Garut





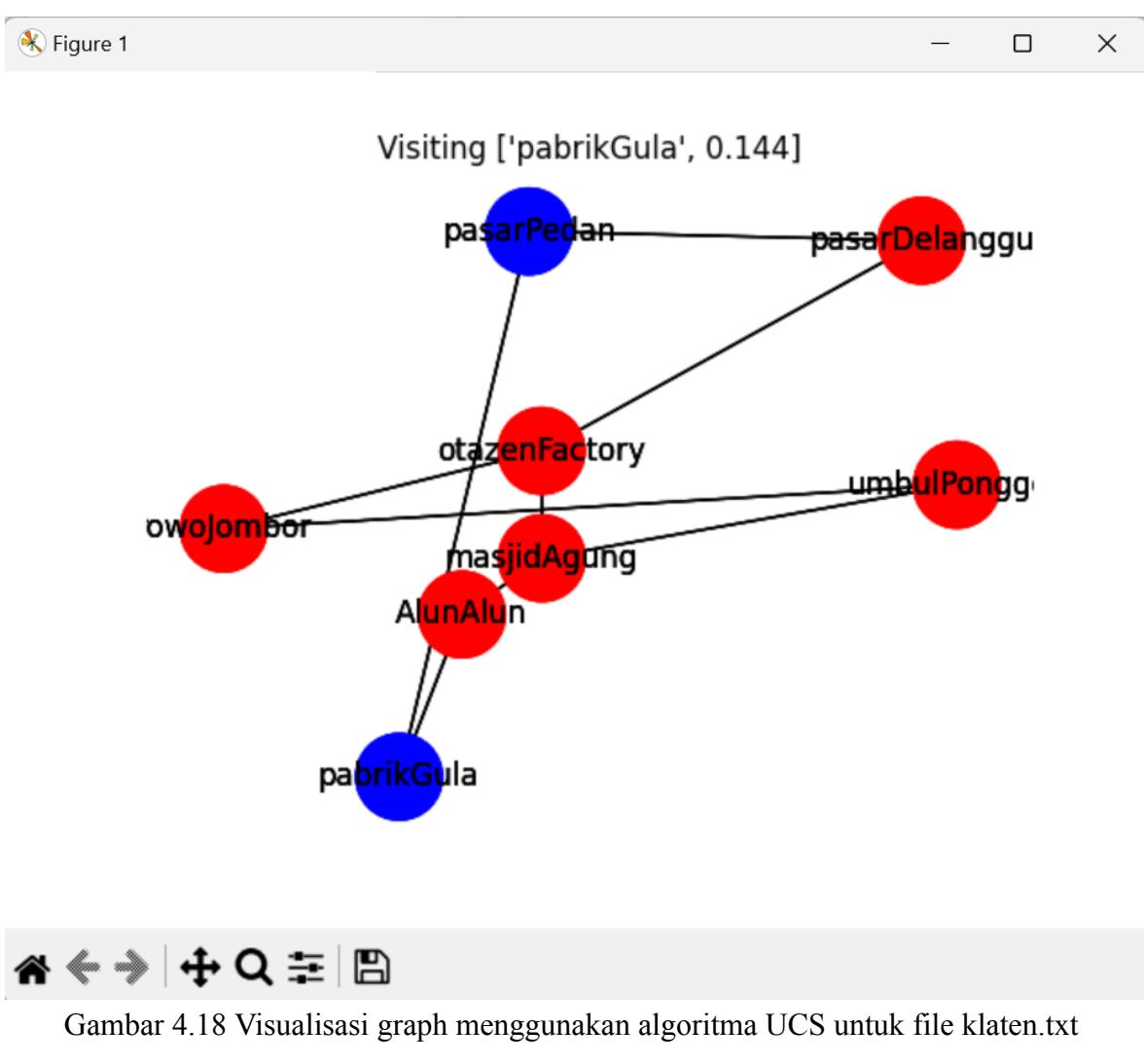




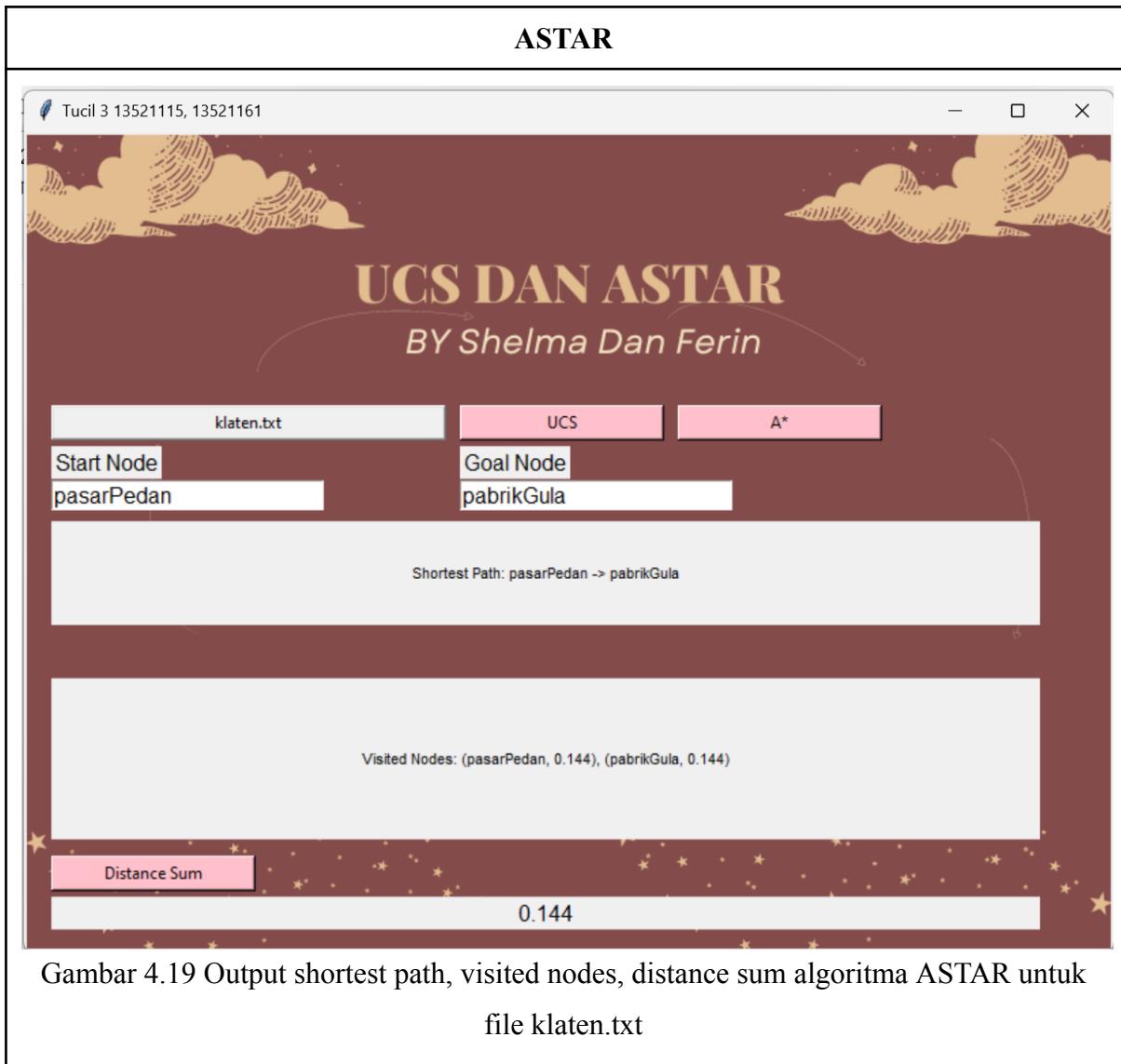
Dari hasil implementasi ini dapat disimpulkan bahwa ASTAR lebih baik dari UCS hal ini bisa dilihat dari banyaknya simpul yang dikunjungi oleh UCS lebih banyak dari ASTAR walaupun jarak yang mereka tempuh sama. karena melewati shortpath yang sama.

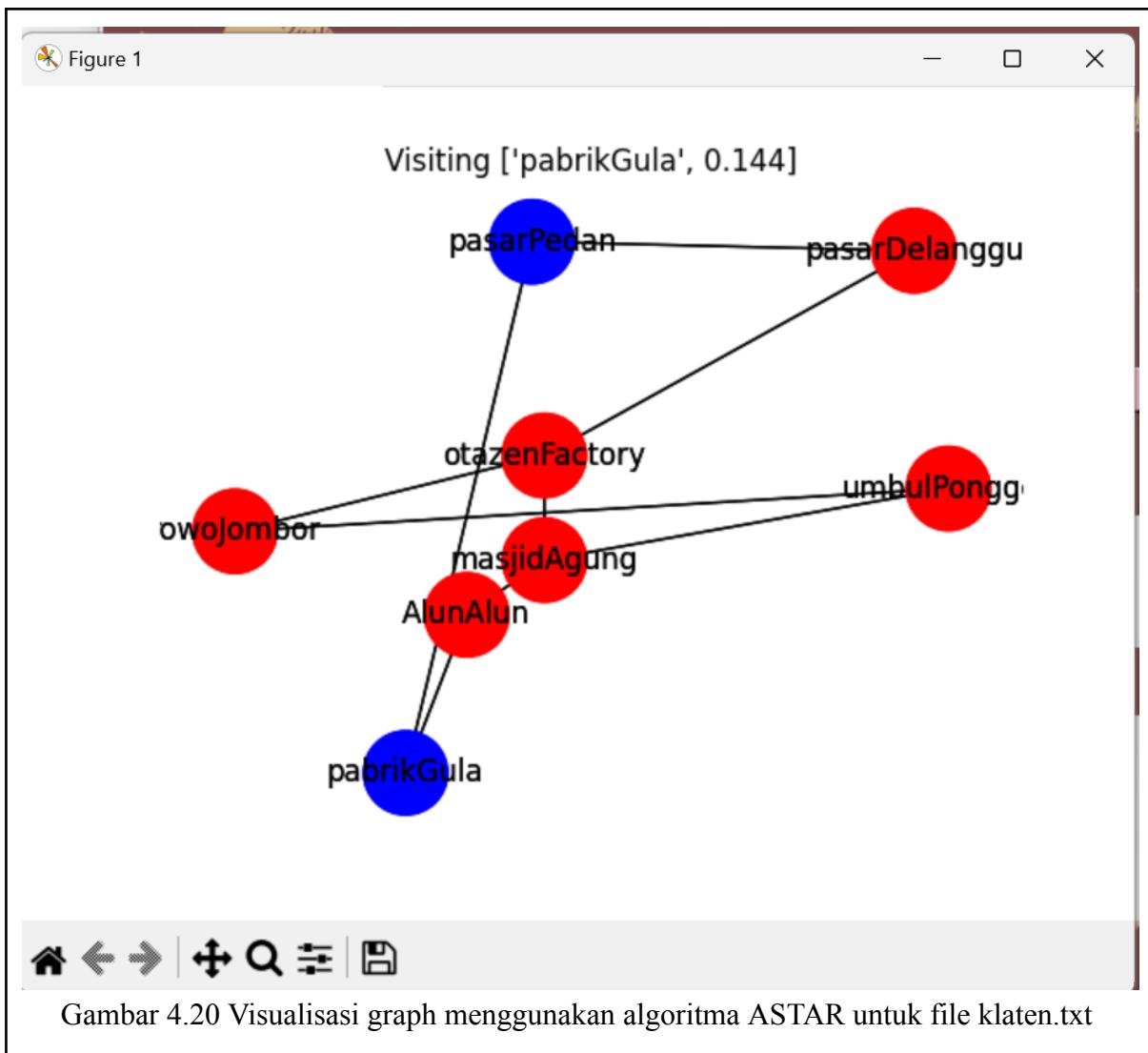
4.5 Test Case Untuk Peta Jalan Sebuah Kawasan di Kota Klaten



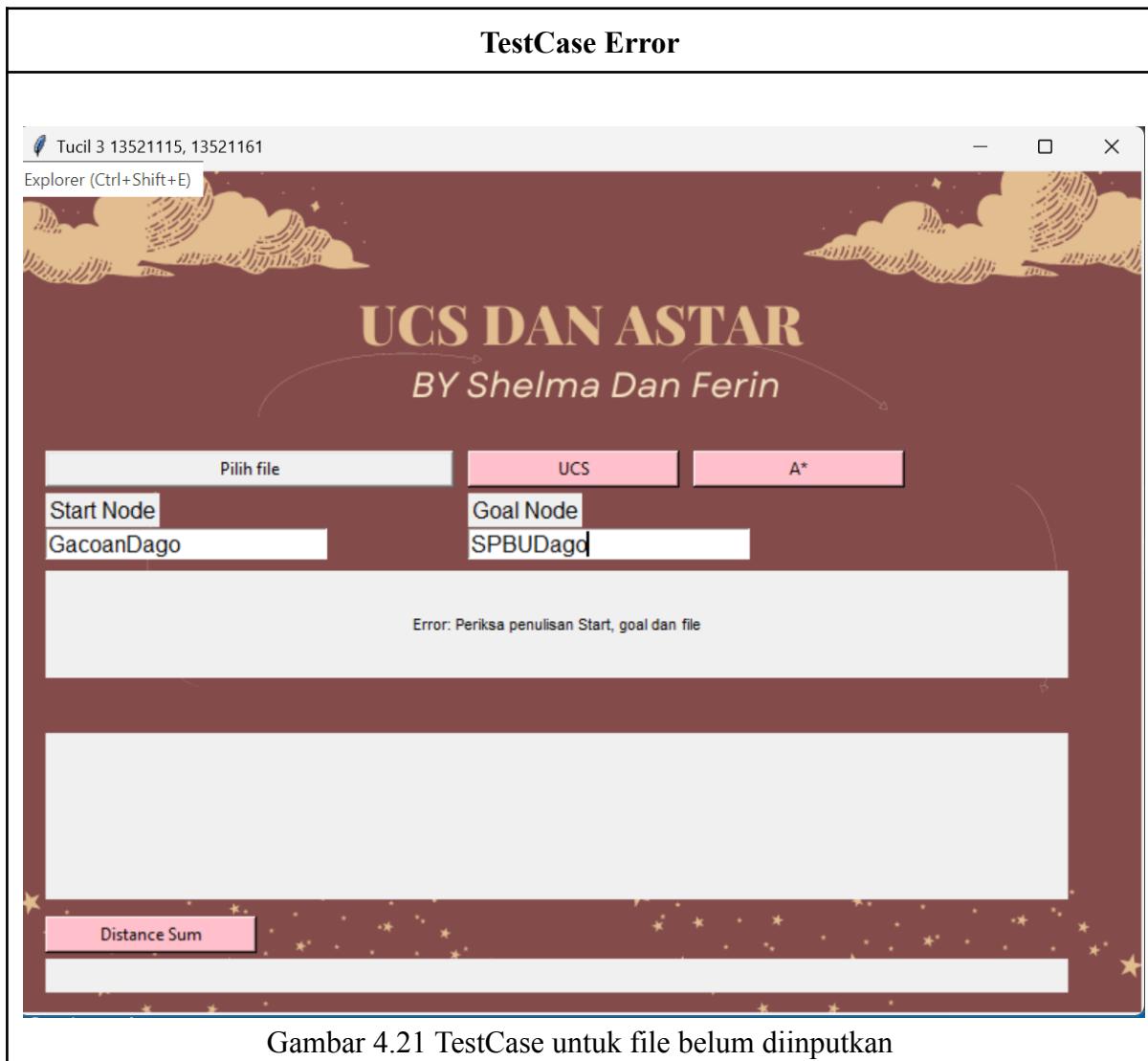


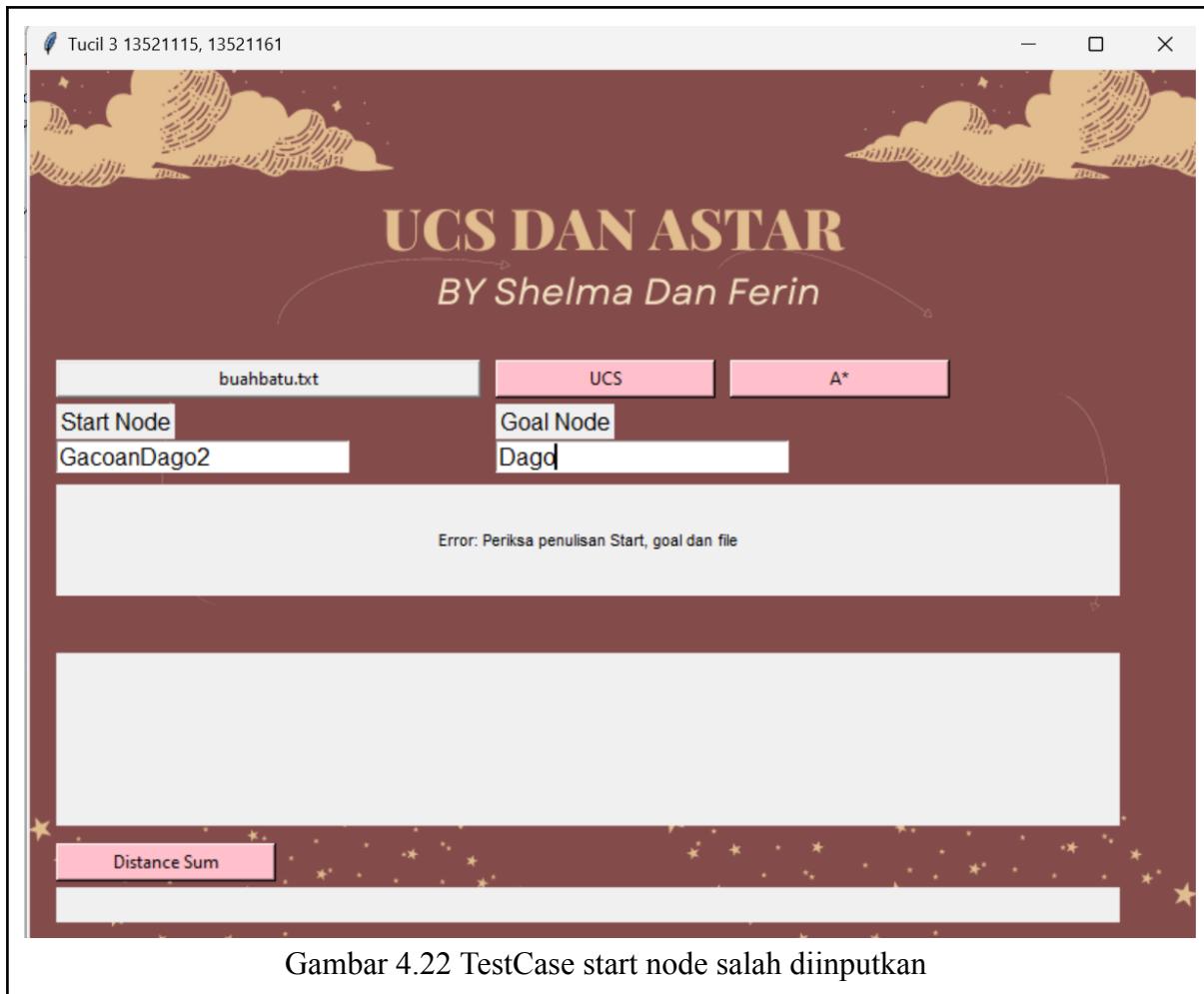
ASTAR





4.6 TestCase untuk kasus error



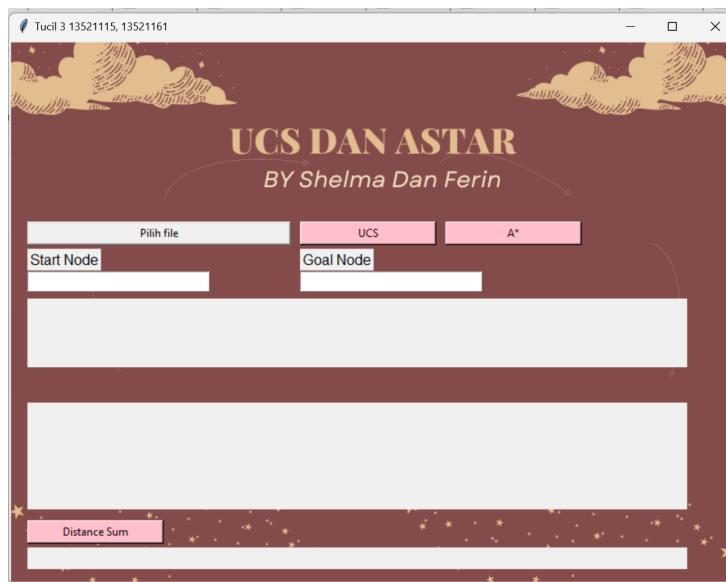


Gambar 4.22 TestCase start node salah diinputkan

BAB V

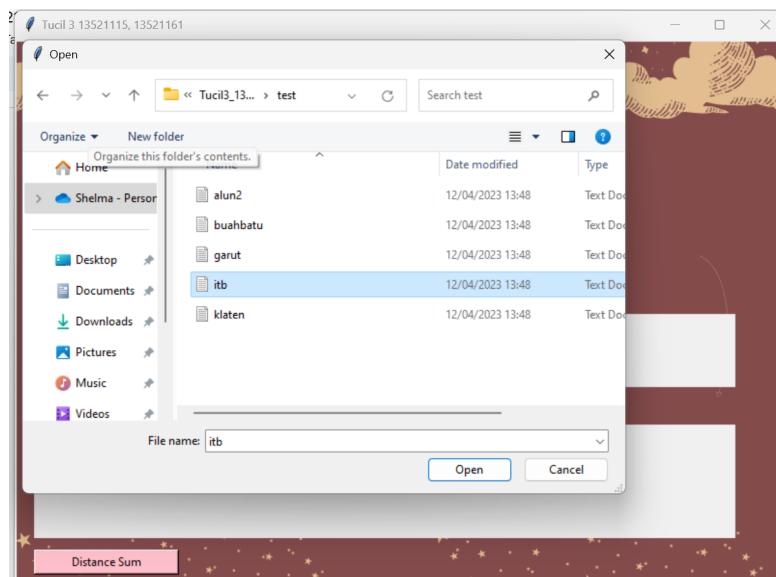
CARA MENJALANKAN PROGRAM

1. Download repository pada tautan
https://github.com/shelmasalsa17/Tucil3_13521115_13521161.git
2. Setelah itu buka di visual studio code atau code editor lainnya.
3. Masuk kedalam file gui.py lalu run setelah itu akan muncul tampilan seperti berikut.
Atau bisa langsung tekan main.exe di folder bin

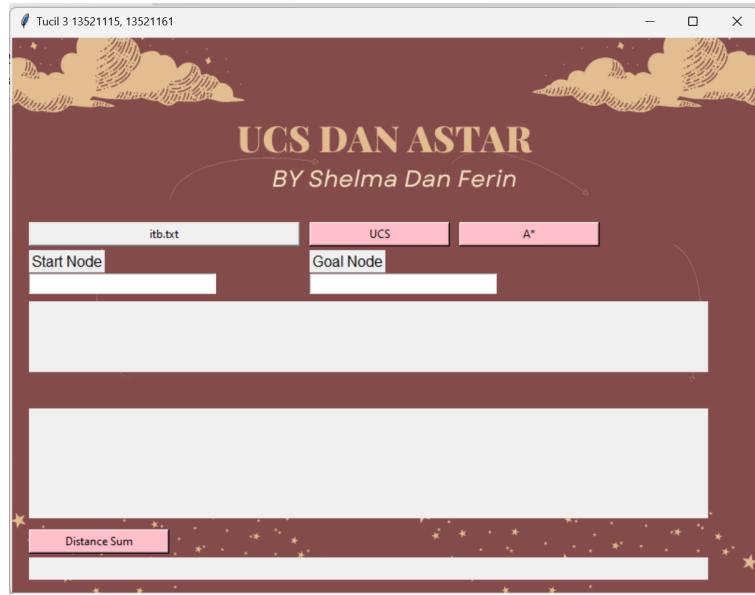


Gambar 5.1 Tampilan GUI

4. Tekan pilih file lalu pilih map yang diinginkan ada 5 pilihan file map yaitu itb, alun2, buahbatu, garut, klaten. Tinggal pergi ke file disimpannya. Misal yang dipilih adalah file itb.txt

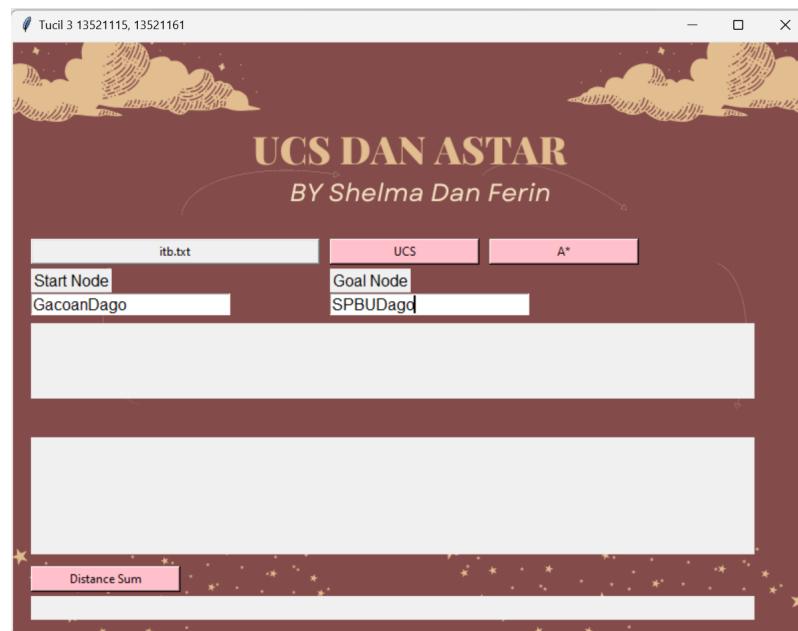


Gambar 5.2 Tampilan GUI saat memilih file



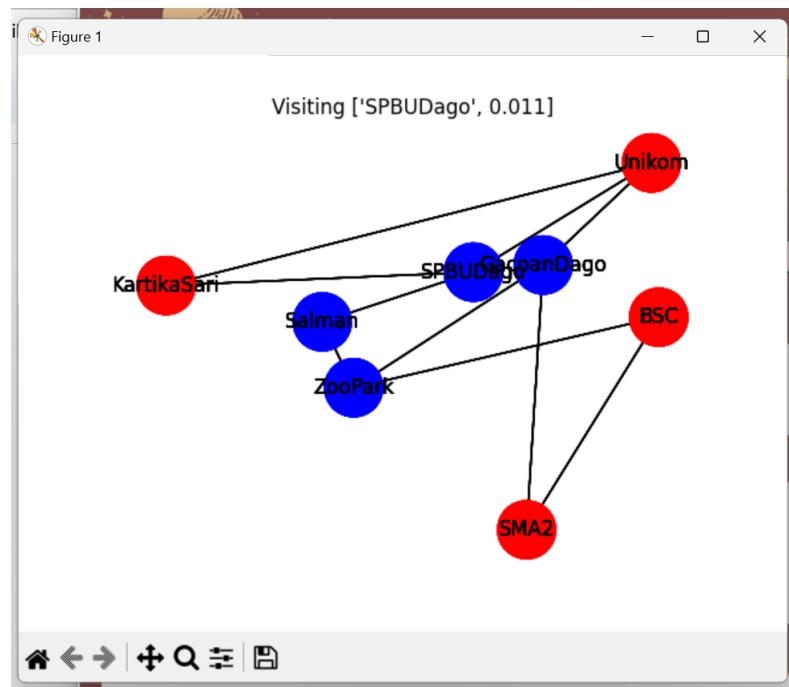
Gambar 5.3 Tampilan GUI setelah memilih file

5. Masukkan StartNode dan GoalNode. Pastikan StartNode dan GoalNode ada dalam daftar simpul file.txt yang diinputkan. Harus diperhatikan tidak boleh ada spasi setelah penulisan itu.



Gambar 5.4 Tampilan GUI saat memasukkan Start Node dan Goal Node

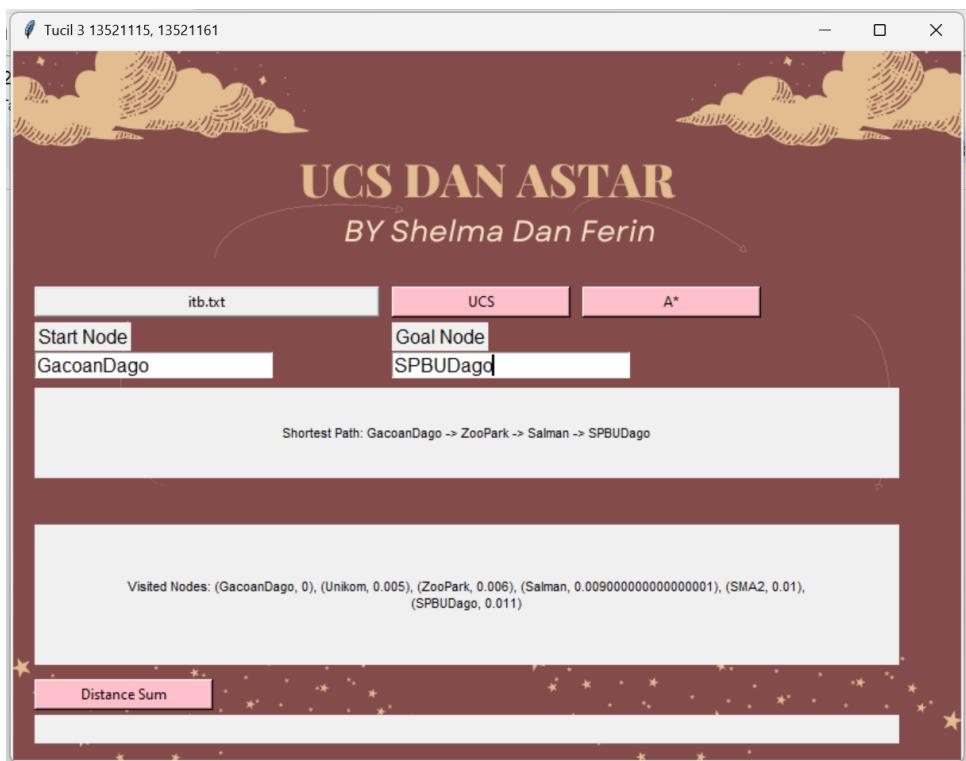
6. Setelah itu tekan algoritma pencarian yang diinginkan
7. Setelah itu akan muncul seperti ini



Gambar 5.5 Visualisasi Graph

Bagian atas akan menampilkan node node yang dikunjungi saat proses pencarian, kemudian warna simpul biru merupakan shortPath yang dihasilkan

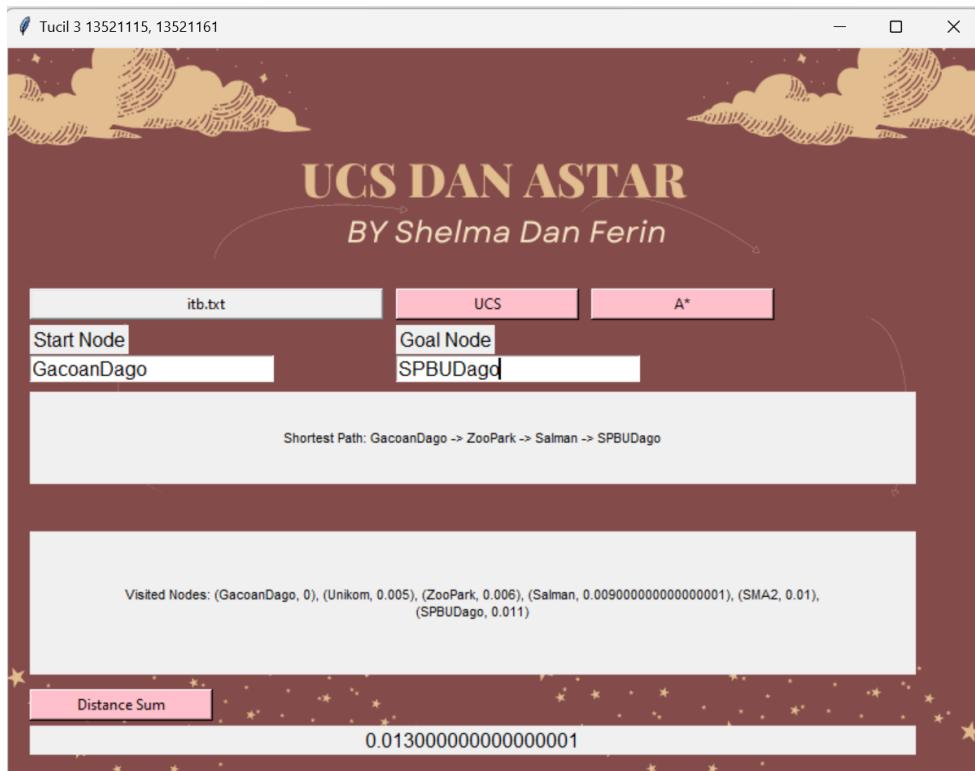
- Setelah itu tutup bagian visualizer



Gambar 5.6 Tampilan Shortest Path dan Visited Nodes
shortestPath : Menunjukkan jalur simpul terpendek yang ditempuh

VisitedNodes : Menunjukkan simpul-simpul yang dikunjungi pada proses pencarian

9. Jika ingin mengetahui jarak yang telah ditempuh maka tekan distance sum



Gambar 5.7 Tampilan Shortest Path, Visited Nodes, dan Distance Sum

10. Beberapa catatan tambahan, Seringkali error terjadi pada beberapa bagian berikut 1.

- Input file tidak sesuai, pastikan input yang diberikan sama dengan format yang telah ditulis.
- Masukan start dan end tanpa spasi harus sesuai dengan simpul yang dituliskan di file
- Tidak mengclose visualizer sehingga simpul-simpul shortPath dan visited tidak muncul
- Tidak menekan tombol "Distance Sum" sehingga nilai distance tidak keluar ataupun berubah menyesuaikan apa yang ingin di test.
- Tidak memasukkan file map
- Node-node yang bisa dimasukkan dalam start dan node

- Berikut node yang ada dalam setiap file.txt

alun2.txt	bubahbatu.txt	garut.txt	itb.txt	klaten.txt
AlunBandung	margaCintaPark	AlunGarut	GacoanDago	AlunAlun
masjidRaya	RSIHarapanBunda	toserbaGarut	ZooPark	pabrikGula
tuguAsia	metroIndahMall	ramayanaGarut	BSC	pasarPedan
parahyanganFutsal	edelweisHospital	stasiunGarut	SMA2	masjidAgung
theKingShop	gacoanBuahBatu	cozyPool	Salman	otazenFactory
dailyHotel	koperasiBUMI	tamanKianSanta ng	SPBUDago	pasarDelanggu
toserbaYogya	tuguSepatu	tamanMaktal	KartikaSari	rowoJombor
gedungMerdeka	MCDBuahBatu	RSUSlamet	Unikom	umbulPonggok

BAB VI

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil implementasi ini dapat disimpulkan bahwa pencarian shortPath dengan ASTAR lebih baik dari pencarian shortPath dengan UCS. Hal ini bisa dilihat dari salah satu test case di atas. Jarak yang ditempuh dengan ASTAR jauh lebih sedikit dibanding UCS. Adapun beberapa test case menunjukkan jarak yang ditempuh sama hanya saja node yang dikunjungi oleh ASTAR pasti jauh lebih sedikit dibanding UCS hal ini, menunjukkan kembali bahwa waktu eksekusi ASTAR jauh lebih cepat dari UCS.

5.2 Saran

Saran yang dapat kami berikan dari pengembangan aplikasi ini adalah pengembangan *graphical user interface* dapat dikembangkan menggunakan pustaka yang berbeda sehingga dapat membuat *graphical user interface* yang lebih *versatile*. Saran lainnya adalah akan lebih baik untuk mengerjakan penulisan struktur data dengan algoritma secara bersamaan karena besar kemungkinan ada kesalahpahaman antara dua orang yang mengerjakan hal tersebut.

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>

LAMPIRAN

Link Google Drive dan Repository Github

- a. Google Drive

[https://drive.google.com/drive/folders/1VUMZmaW4n_ipCE8WC_EqGR5zH53vizsz
?usp=sharing](https://drive.google.com/drive/folders/1VUMZmaW4n_ipCE8WC_EqGR5zH53vizsz?usp=sharing)

- b. Repository Github

https://github.com/shelmasalsa17/Tucil3_13521115_13521161.git

Checklist

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