Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Semester II tahun 2022/2023

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun oleh

Syarifa Dwi Purnamasari	13521018
Jauza Lathifah Annassalafi	13521030

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

DAFTAR ISI

BAB 1

DESKRIPSI PERSOALAN

1.1 ALGORITMA UCS

UCS (Uniform Cost Search) adalah algoritma pencarian jalur terpendek pada graf dengan bobot yang tidak negatif. Algoritma UCS menggunakan prinsip pencarian dengan biaya seragam, artinya pencarian dilakukan dengan mempertimbangkan biaya terkecil dari simpul awal ke simpul tujuan. Berikut adalah langkah-langkah algoritma UCS:

1. Inisialisasi
   - Tentukan simpul awal
   - Beri nilai biaya awal ke simpul awal, yaitu 0
   - Masukkan simpul awal ke dalam himpunan simpul terbuka
2. Ambil simpul dengan biaya terkecil dari himpunan simpul terbuka
3. Periksa apakah simpul tersebut merupakan simpul tujuan. Jika ya, selesaikan pencarian
4. Jika simpul bukan simpul tujuan, tambahkan simpul-simpul tetangga yang belum dieksplorasi ke dalam himpunan simpul terbuka.
5. Perbarui nilai biaya untuk setiap simpul tetangga baru yang ditambahkan. Jika biaya yang diperoleh lebih kecil dari biaya sebelumnya, perbarui nilai biaya dan jadikan simpul tetangga sebagai orangtua simpul saat ini.
6. Tandai simpul saat ini sebagai simpul dieksplorasi dan hapus dari himpunan simpul terbuka.
7. Ulangi langkah 2 sampai 6 sampai simpul tujuan ditemukan atau himpunan simpul terbuka kosong.

Algoritma UCS dapat digunakan pada graf dengan jumlah simpul dan jumlah sisi yang besar, namun algoritma ini hanya efektif pada graf dengan bobot yang tidak negatif.

1.2 ALGORITMA A*

Algoritma A* (A-star) adalah algoritma pencarian jalur terpendek pada graf yang menggunakan pendekatan heuristik. Algoritma A* mencoba menggabungkan dua metode, yaitu algoritma UCS (Uniform Cost Search) dan algoritma Greedy Best First Search. Berikut adalah langkah-langkah algoritma A*:

1. Inisialisasi
   - Tentukan simpul awal
   - Tentukan simpul tujuan
   - Beri nilai biaya awal ke simpul awal, yaitu 0
   - Masukkan simpul awal ke dalam himpunan simpul terbuka

2. Ambil simpul dengan biaya terkecil dari himpunan simpul terbuka, tetapi dengan tambahan nilai heuristik.
    - Nilai heuristik adalah perkiraan biaya dari simpul saat ini ke simpul tujuan.
    - Algoritma A* menggunakan nilai heuristik untuk memperkirakan biaya yang masih perlu ditempuh hingga mencapai simpul tujuan.
3. Periksa apakah simpul tersebut merupakan simpul tujuan. Jika ya, selesaikan pencarian.
4. Jika simpul bukan simpul tujuan, tambahkan simpul-simpul tetangga yang belum dieksplorasi ke dalam himpunan simpul terbuka.
5. Perbarui nilai biaya untuk setiap simpul tetangga baru yang ditambahkan. Jika biaya yang diperoleh lebih kecil dari biaya sebelumnya, perbarui nilai biaya dan jadikan simpul tetangga sebagai orangtua simpul saat ini.
6. Tandai simpul saat ini sebagai simpul dieksplorasi dan hapus dari himpunan simpul terbuka.
7. Ulangi langkah 2 sampai 6 sampai simpul tujuan ditemukan atau himpunan simpul terbuka kosong.

Algoritma A* dapat digunakan pada graf dengan jumlah simpul dan jumlah sisi yang besar dan lebih efisien daripada algoritma UCS. Namun, nilai heuristik harus memenuhi kriteria admissibility dan consistency agar algoritma A* menghasilkan jalur terpendek yang optimal.

1.4 Alur Kerja Program

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, penulis 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 (meter) 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.

Pada program yang dibuat penulis, input file yang diminta berupa matriks ketetanggaan dan nama titik serta koordinatnya. Matriks ketetanggaan yang dijadikan input boleh memiliki bobot ataupun tidak. Ketika matriks sudah memiliki bobot, jarak antar titik yang bertetangga akan diambil dari bobot tersebut, sedangkan jika matriks belum memiliki bobot maka bobot akan di generate melalui rumus Euclidean yang ada pada program. Setelah selesai membaca file, program akan menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek ditampilkan pada peta/graf yang akan muncul ketika program sudah menemukan path yang dilalui serta program juga akan menampilkan peta menggunakan Google Map API yang sudah di generate.

BAB 2

KODE PROGRAM

2.1 inputFile.py

```
import os
import numpy as np


# Input file validation
def inputValid():
    flag = False
    while (flag == False):
        filename = input("\t Masukkan nama file : ")
        filepath = os.path.join('..\\Tucil3_13521018_13521030\\test',
filename)
        # Check if file exists
        if not os.path.isfile(filepath):
            print("\t File tidak ditemukan.")
            flag = False
        else :
            # Read file
            with open(filepath, 'r') as f:
                matrix = []
                line = f.readline()
                row = line.strip().split()
                row = [int(value) for value in row]
```

```
                length = len(row)
                # Check if file has minimum 8 nodes
                if length < 8:
                    print("\t Input file minimum memiliki 8 node.")
                    flag = False
                else :
                    if any(value < 0 for value in row):
                        print("\t Nilai matriks tidak boleh
negatif.")
                        flag = False
                        break
                    else :
                        # Append row to matrix
                        matrix.append(row)
                        for i in range(length-1):
                            line = f.readline()
                            row = line.strip().split()
                            row = [int(value) for value in row]
                            # Check if matrix has negative value
                            if any(value < 0 for value in row):
                                print("\t Nilai matriks tidak boleh
negatif.")
                                flag = False
                                break
                            else :
```

```
                                # Append row to matrix
                                matrix.append(row)
                                flag = True
                        # Check if matrix is symmetric
                        if (flag == True):
                            matrix = np.array(matrix)
                            if not isSymmetric(matrix, len(matrix)):
                                print("\t Matriks tidak simetris.")
                                flag = False
                                continue
                            else :
                                print("\t Input file valid.")
                                # Convert line to list of node and
list of coordinates
                                node = []
                                point = []
                                for i in range(length):
                                    line = f.readline()
                                    row = line.strip().split(', ')
                                    node.append(row[0])
                                    point.append([float(row[1]),
float(row[2])])
                                # Convert matrix of adjacent to
matrix of distances
                                matrixDist = [[0 for x in
range(length)] for y in range(length)]
```

```
                                for j in range(length):
                                    for k in range(length):
                                        # Calculate distance between
two nodes
                                        distance = ((point[j][0] -
point[k][0])**2 + (point[j][1] - point[k][1])**2)**0.5
                                        distance = distance*100000
                                        distance = round(distance,2)
                                        matrixDist[j][k] = distance
                                flag = True
    for i in range (len(matrix)):
        for j in range (len(matrix)):
            if matrix[i][j] == 1:
                matrix[i][j] = matrixDist[i][j]
    return node, point, matrix, matrixDist


# Check if matrix is symmetric
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


# Print list
```

```python
def printList(list):
    for i in range(len(list)):
        print(list[i])


# Print matrix
def printMatrix(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            print(matrix[i][j], end=' ')
        print()


# get node : point dictionary
def arrKoordinat(node, point):
    arr = {}
    length = len(node)
    for i in range(length):
        arr[node[i]] = point[i]
    return arr


# matrix of path
def matrixPath(path, node, cost):
    mat = [[0 for i in range(len(node))] for j in range(len(node))]
    for i in range(len(path)-1):
        mat[node.index(path[i])][node.index(path[i+1])] = cost[i]
```

```
        mat[node.index(path[i+1])][node.index(path[i])] = cost[i]

    return mat
```

2.2 visualisasi.py

```
import networkx as nx

import matplotlib.pyplot as plt
```

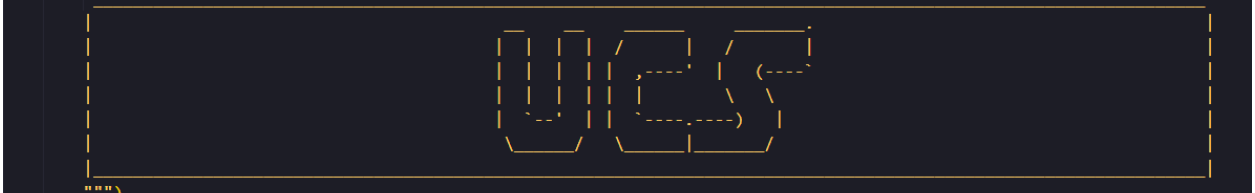
```python
from inputFile import *


# membuat graph
def createGraph(node, path, matrix, string):
    graph = nx.Graph()
    if (string == "initial"):
        for i in range(len(node)):
            graph.add_node(node[i])
    elif (string == "ucs_aStar"):
        for i in range(len(path)):
            graph.add_node(path[i])
    for i in range(len(node)):
        for j in range(len(node)):
            if matrix[i][j] != 0:
                graph.add_edge(node[i], node[j], weight=matrix[i][j])
    return graph


# menampilkan graph
def showGraph(graph, arr, finalGraph, string, path, cost):
    position = nx.spring_layout(graph)
    position = arr
    if (string == "initial"):
```

```
        nx.draw(graph, position, with_labels=True,
node_color='green', node_size=300, edge_color='grey', width=2,
font_size=8)

        labels = nx.get_edge_attributes(graph, 'weight')

        nx.draw_networkx_edge_labels(graph, position,
edge_labels=labels, font_size=8, label_pos=0.5, rotate=False,
font_color='black', font_weight='bold')

        fig = plt.gcf()

        fig.suptitle('Initial Graph', fontsize=16,
fontweight='bold')

    elif (string == "ucs"):

        edge_colors = ['red' if edge in finalGraph.edges else 'grey'
for edge in graph.edges()]

        nx.draw(graph, position, with_labels=True,
node_color='green', node_size=300, edge_color=edge_colors, width=2,
font_size=8)

        labels = nx.get_edge_attributes(graph, 'weight')

        nx.draw_networkx_edge_labels(graph, position,
edge_labels=labels, font_size=8, label_pos=0.5, rotate=False,
font_color='black', font_weight='bold')

        for i in range(len(finalGraph.nodes)):

            nx.draw_networkx_nodes(finalGraph, position,
nodelist=[list(finalGraph.nodes)[i]], node_color='blue',
node_size=500)

        fig = plt.gcf()

        fig.suptitle('UCS Graph', fontsize=16, fontweight='bold')

        fig.text(0.02, 0.06, 'Path: ' + str(path), fontsize=5,
fontweight='bold', color='black', bbox=dict(facecolor='white',
edgecolor='black', boxstyle='round,pad=0.5'))
```

```
        fig.text(0.02, 0.02, 'Cost: ' + str(cost), fontsize=5,
fontweight='bold', color='black', bbox=dict(facecolor='white',
edgecolor='black', boxstyle='round,pad=0.5'))

    elif (string == "aStar"):

        edge_colors = ['red' if edge in finalGraph.edges else 'grey'
for edge in graph.edges()]

        nx.draw(graph, position, with_labels=True,
node_color='green', node_size=300, edge_color=edge_colors, width=2,
font_size=8)

        labels = nx.get_edge_attributes(graph, 'weight')

        nx.draw_networkx_edge_labels(graph, position,
edge_labels=labels, font_size=8, label_pos=0.5, rotate=False,
font_color='black', font_weight='bold')

        for i in range(len(finalGraph.nodes)):

            nx.draw_networkx_nodes(finalGraph, position,
nodelist=[list(finalGraph.nodes)[i]], node_color='blue',
node_size=500)

        fig = plt.gcf()

        fig.suptitle('A* Graph', fontsize=16, fontweight='bold')

        fig.text(0.02, 0.06, 'Path: ' + str(path), fontsize=5,
fontweight='bold', color='black', bbox=dict(facecolor='white',
edgecolor='black', boxstyle='round,pad=0.5'))

        fig.text(0.02, 0.02, 'Cost: ' + str(cost), fontsize=5,
fontweight='bold', color='black', bbox=dict(facecolor='white',
edgecolor='black', boxstyle='round,pad=0.5'))

    plt.show()
```

```
# menampilkan ascii art
def art(string):
    if (string == "SHORT PATH"):
        print("""
```

```
    elif (string == "UCS"):
        print("""
```



```
    elif (string == "A*"):
        print("""
```

```
    elif (string == "line"):
        print("""
        _____________________________________________________________________________________________________________
        """)
```

2.3 aStar.py

```
from queue import PriorityQueue
import copy
from visualisasi import *
from inputFile import *
from apiMap import *


def aStar(node, matrix, matrixDist, start, goal):
    # mengubah nama node menjadi index
    start = node.index(start)
    goal = node.index(goal)
    # mengambil matrix distance yang berisi jarak garis lurus dari simpul goal
    mat = matrixDist[goal]
    queue = PriorityQueue()
    path = []
    listChild = []
    if (start == goal):
        path.append(0+mat[start], start, child, 0)
        dist = 0
    else :
        tempPath = []
        tempChild = []
        child = 0
```

```
        found = False
        temp = (0+mat[start], start, child, 0)
        listChild.append(temp[2])
        gn = matrix[start][temp[1]]
        while not (found):
            child += 1
            # kondisi jika sudah sampai ke goal
            if (temp[1] == goal):
                found = True
                path.append(temp)
                dist = temp[3]
            else :
                # menginisialisasi node yang sedang dikunjungi
                curr = temp[1]
                path.append(temp)
                tempGn = gn
                # memasukkan node yang belum dikunjungi dan
bertetangga dengan curr ke queue
                for i in range(len(node)):
                    gn = tempGn
                    if (matrix[curr][i] > 0):
                        gn += matrix[curr][i]
                        queue.put((gn + mat[i], i, child, gn))
                temp1 = copy.copy(temp)
```

```
                temp = queue.get()
                # mengganti node jika sudah pernah dikunjungi
                for i in range(len(path)):
                    if (temp[1] == path[i][1]):
                        temp = queue.get()
                gn = temp[3]
                tempChild1 = tempChild.copy()
                tempPath1 = tempPath.copy()
                idx = temp[2]
                if (len(tempChild1) != 0):
                    idx = tempChild[0]
                # mengecek kondisi backtracking
                if (temp[2] <= temp1[2] or matrix[temp[1]][temp1[1]]
== 0):
                    removeChild = []
                    removePath = []
                    tempChild = []
                    tempPath = []
                    for i in range(len(listChild)):
                        if (listChild[i] >= idx):
                            t = listChild[i]
                            removeChild.append(t)
                            tempChild.append(t)
                            tt = path[i]
```

```
                        removePath.append(tt)
                        tempPath.append(tt)
                for i in range(len(removeChild)):
                    listChild.remove(removeChild[i])
                    path.remove(removePath[i])
                if (len(tempChild1) != 0):
                    for k in range(len(tempChild1)):
                        listChild.append(tempChild1[k])
                    for l in range(len(tempPath1)):
                        path.append(tempPath1[l])
            listChild.append(temp[2])
    return path, dist
```

2.4 ucs.py

```
from visualisasi import *

from inputFile import *

# sort by a cost value

def sortByCost(dict):

    sorted_tuples = sorted(dict.items(), key=lambda
x:-list(x[1].values())[0], reverse=False)

    sorted_dict = {t[0]:t[1] for t in sorted_tuples}

    return sorted_dict


# Uniform Cost Search

def ucs(matrix, start, goal, node):

    visited = []

    queue = {}

    path = []

    queue.update({(0, start) : {'': 0}})

    path.append(start)

    visited.append(start)

    j = 0

    while queue:

        queue = sortByCost(queue)

        path = queue.popitem()

        currNode = path[0][1]

        visited.append(currNode)
```

```
        currPath = list(path[1].keys())[0]
        currCost = list(path[1].values())[0]
        for i in range(len(matrix)):
            if matrix[node.index(currNode)][i] != 0:
                if node[i] not in visited:
                    queue.update({(j, node[i]) : {currPath + "," +
node[i] : currCost + matrix[node.index(currNode)][i]}})
        j += 1
        if currNode == goal:
            break
    cost = list(path[1].values())[0]
    path = (start + list(path[1].keys())[0]).split(',')
    return path, cost
```

2.5 apiMap.py

```
import plotly.graph_objects as go


def printGraph(node, point, path, matrix):
    pointY = []
    pointX = []
    for i in range(len(point)):
        pointX.append(point[i][0])
        pointY.append(point[i][1])


    pathX = []
    pathY = []
    for i in range(len(path)):
        idx = node.index(path[i])
        pathX.append(point[idx][0])
        pathY.append(point[idx][1])


    fig = go.Figure(go.Scattermapbox(
        lon = pointY,
        lat = pointX
        ))


    for i in range(len(matrix)):
        for j in range(0,i):
            if (matrix[i][j] > 0):
```

```
                adjacentX = []
                adjacentY = []
                adjacentX.append(point[j][0])
                adjacentY.append(point[j][1])
                adjacentX.append(point[i][0])
                adjacentY.append(point[i][1])
                fig.add_trace(go.Scattermapbox(
                    mode = "markers+lines",
                    lon = adjacentY,
                    lat = adjacentX,
                    marker = {'size': 10, 'color' : 'rgb(0, 255, 0)'}))


fig.add_trace(go.Scattermapbox(
    mode = "markers+lines",
    lon = pathY,
    lat = pathX,
    marker = {'size': 10, 'color' : 'rgb(255, 0, 0)'}))


max_x, max_y = maxPoint(point)
min_x, min_y = minPoint(point)


fig.update_layout(
    margin ={'l':0,'t':0,'b':0,'r':0},
    mapbox = {
        'center': {'lon': min_y, 'lat': min_x},
```

```
            'style': "stamen-terrain",
            'center': {'lon': max_y, 'lat': max_x},
            'zoom': 16})

    fig.show()


def maxPoint(point):
    max_x = point[0][0]
    max_y = point[0][1]
    for i in range(len(point)):
        if (point[i][0] > max_x):
            max_x = point[i][0]
        if (point[i][1] > max_y):
            max_y = point[i][1]
    return max_x, max_y


def minPoint(point):
    min_x = point[0][0]
    min_y = point[0][1]
    for i in range(len(point)):
        if (point[i][0] < min_x):
            min_x = point[i][0]
        if (point[i][1] < min_y):
            min_y = point[i][1]
    return min_x, min_y
```

2.6 main.py

```
from inputFile import *

from visualisasi import *

from aStar import *

from ucs import *

from apiMap import *


if __name__ == "__main__":

    art("SHORT PATH")    # menampilkan ascii art

    # validasi intup txt

    node, point, matrix, matrixDist = inputValid()

    print("\t Menampilkan Graph...")

    showGraph(createGraph(node, 0, matrix, "initial"),
arrKoordinat(node, point), 0, "initial", 0, 0)

    art("line")

    print("\t Area :")

    for i in range(len(node)):

        print("\t -", node[i])

    start = str(input("\t Masukkan titik awal : "))

    goal = str(input("\t Masukkan titik tujuan : "))

    while (start not in node or goal not in node or start == goal):

        if (start == goal and start in node and goal in node):

            print("\t Titik tidak valid. Titik awal tidak boleh sama
dengan titik tujuan.")

        else:
```

```
            print("\t Titik tidak valid. Masukkan titik yang ada di
area.")
        start = str(input("\t Masukkan titik awal : "))
        goal = str(input("\t Masukkan titik tujuan : "))

    # UCS
    art("UCS")
    pathUCS, costUCS = ucs(matrix, start, goal, node)
    if (start not in pathUCS or goal not in pathUCS):
        print("\t Tidak ada jalur yang dapat ditempuh.")
    else:
        print("\t Path : ", end="")
        strpath = ""
        for i in range(len(pathUCS) - 1):
            strpath = strpath + pathUCS[i] + " -> "
            print(pathUCS[i], end=" -> ")
        print(goal)
        print("\t Jarak : ", costUCS)
        print("\t Menampilkan Graph...")
        # membuat cost untuk graph
        cost = []
        for i in range(len(pathUCS)):

cost.append(matrixDist[node.index(pathUCS[i])][node.index(goal)])
        # membuat matrix yang menghubungkan path
```

```
        mat = matrixPath(pathUCS, node, cost)

        showGraph(createGraph(node, 0, matrix, "initial"),
arrKoordinat(node, point), createGraph(node, pathUCS, mat,
"ucs_aStar"), "ucs", strpath + goal, costUCS)


    # A*

    art("A*")

    pathA, costA = aStar(node, matrix, matrixDist, start, goal)

    for i in range(len(pathA)):

        pathA[i] = node[pathA[i][1]]

    print("\t Path : ", end="")

    strpathA = ""

    for i in range(len(pathA)):

        if (i == len(pathA) - 1):

            print(pathA[i])

        else:

            strpathA = strpathA + pathA[i] + " -> "

            print(pathA[i], end=" -> ")

    print("\t Jarak : ", costA)

    print("\t Menampilkan Graph...")


    cost = []

    for i in range(len(pathA)):


cost.append(matrixDist[node.index(pathA[i])][node.index(goal)])
```

```
    mat = matrixPath(pathA, node, cost)

    showGraph(createGraph(node, 0, matrix, "initial"),
arrKoordinat(node, point), createGraph(node, pathA, mat,
"ucs_aStar"), "aStar", strpathA + goal, costA)


    printGraph(node, point, pathUCS, matrix)

    printGraph(node, point, pathA, matrix)
```

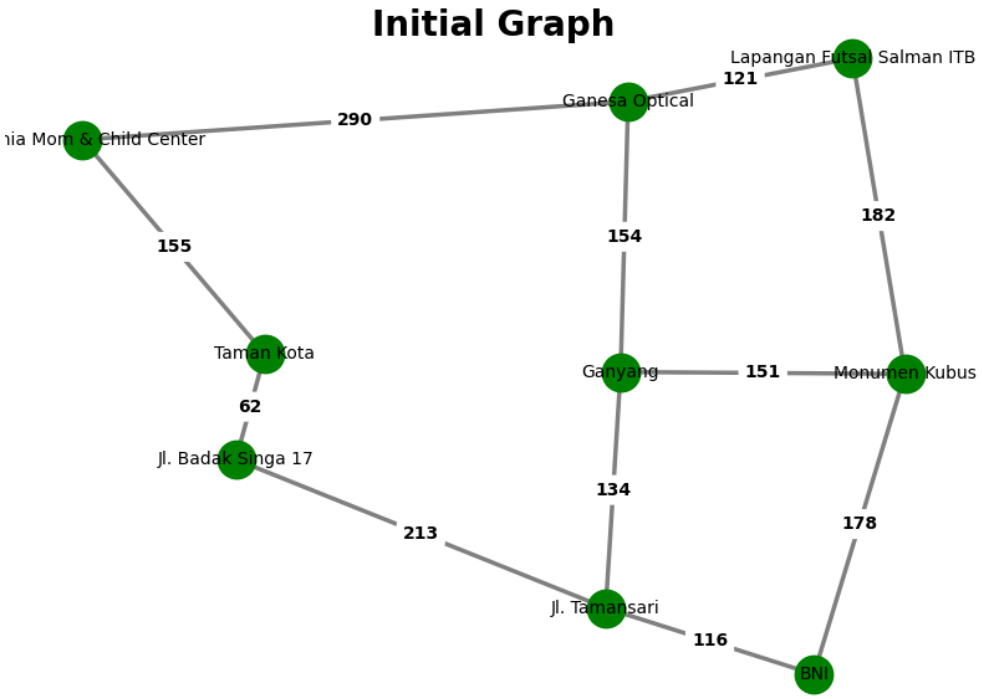
BAB 3

TEST CASE

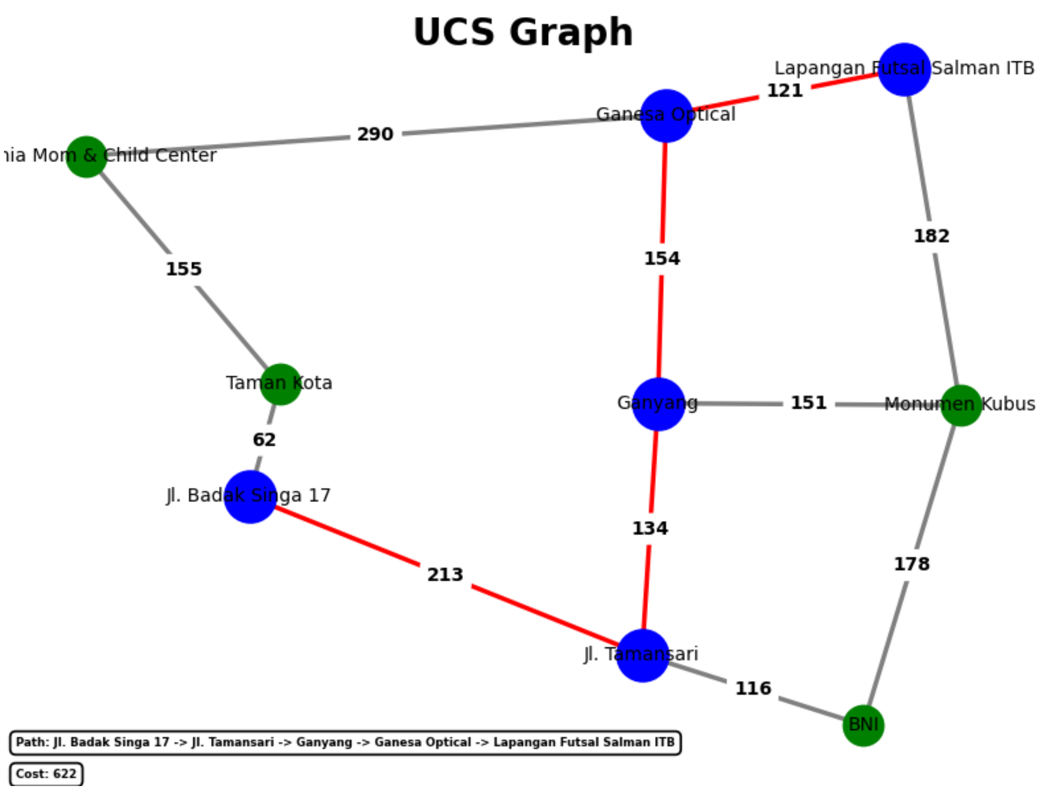
3.1 Test Case 1 : Peta jalan sekitar kampus ITB/Dago/Bandung Utara gor

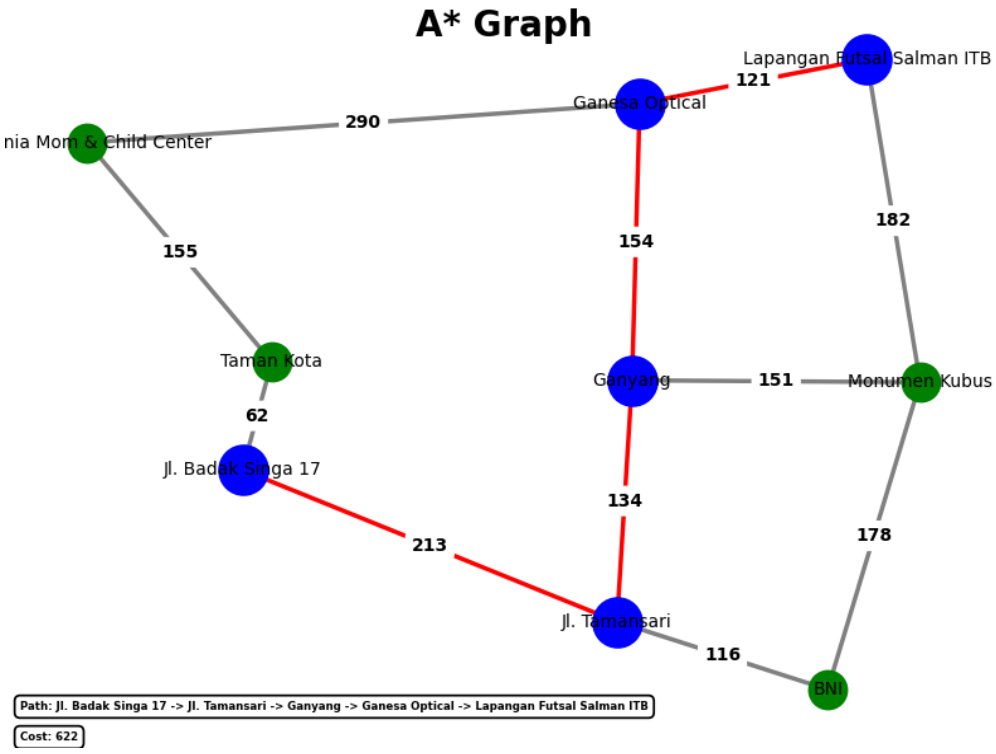
graf2.txt

0 1 1 0 1 0 0 0 0
1 0 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0 0
0 0 1 0 1 0 0 0 1
1 0 0 1 0 1 0 0 0
0 1 0 0 1 0 1 0 0
0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 1 0 1
0 0 0 1 0 0 0 1 0
Monumen Kubus, -6.893266, 107.610145
BNI, -6.893751, 107.608431
Lapangan Futsal Salman ITB, -6.893546, 107.611947
Ganesa Optical, -6.894737, 107.611700
Ganyang, -6.894777, 107.610156
Jl. Tamansari, -6.894856, 107.608809
Jl. Badak Singa 17, -6.896820, 107.609656
Taman Kota, -6.896667, 107.610259
Galenia Mom & Child Center, -6.897634, 107.611482



```
Masukkan nama file : graf2.txt
Input file valid.
Menampilkan Graph...

------------------------------------------------------------------------------------------------------------------

Area :
- Monumen Kubus
- BNI
- Lapangan Futsal Salman ITB
- Ganesa Optical
- Ganyang
- Jl. Tamansari
- Jl. Badak Singa 17
- Taman Kota
- Galenia Mom & Child Center
Masukkan titik awal : Jl. Badak Singa 17
Masukkan titik tujuan : Lapangan Futsal Salman ITB
```



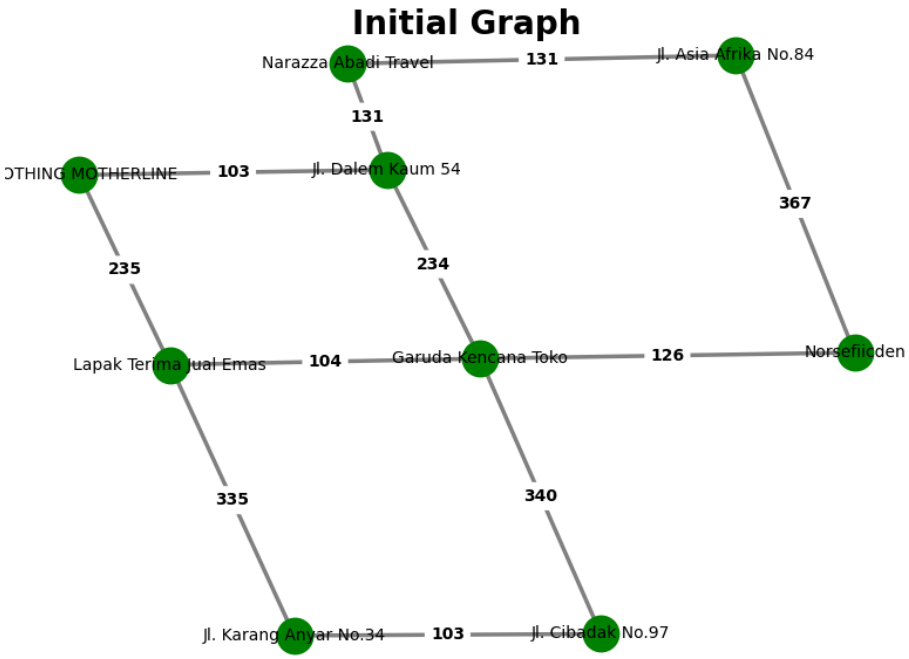

A* Graph
Lapangan Futsal Salman ITB
121
Ganesa Optical
290
nia Mom & Child Center
182
154
155
Taman Kota
Ganyang
151
Monumen Kubus
62
Jl. Badak Singa 17
134
213
178
Jl. Tamansari
116
BNI
Path: Jl. Badak Singa 17 -> Jl. Tamansari -> Ganyang -> Ganesa Optical -> Lapangan Futsal Salman ITB
Cost: 622

3.2 Test Case 2 : Peta jalan sekitar Alun-alun Bandung

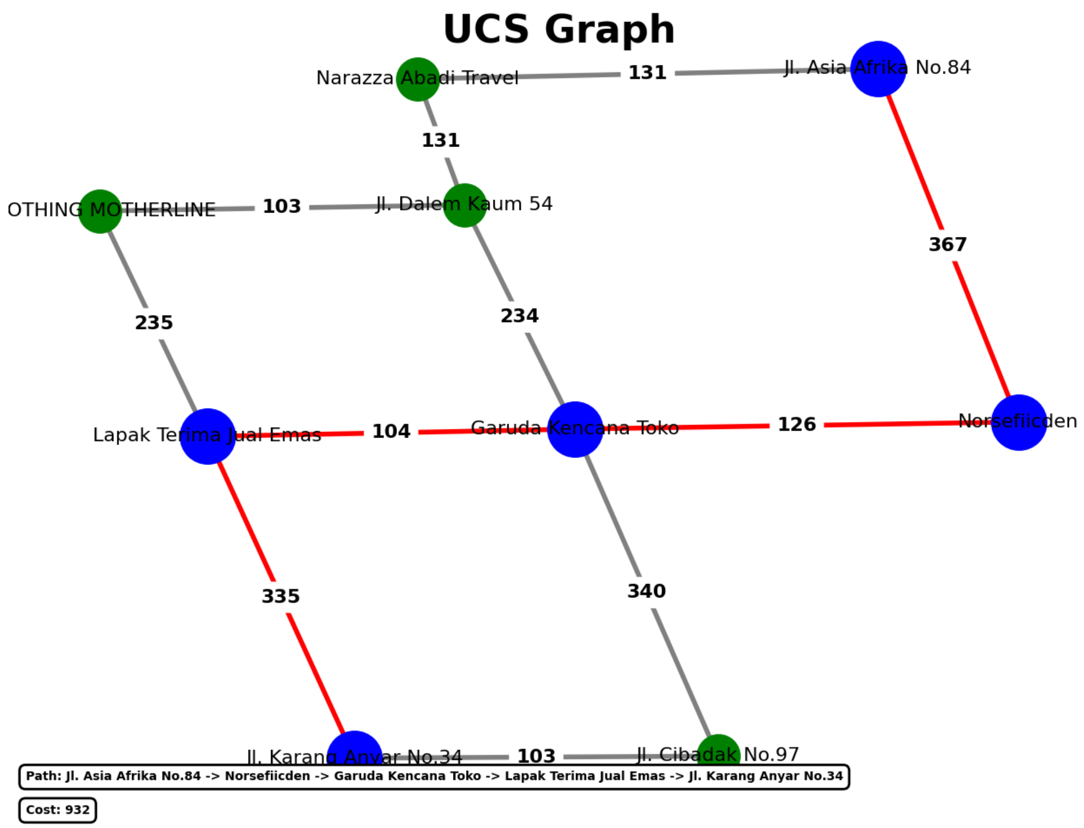
graf3.txt

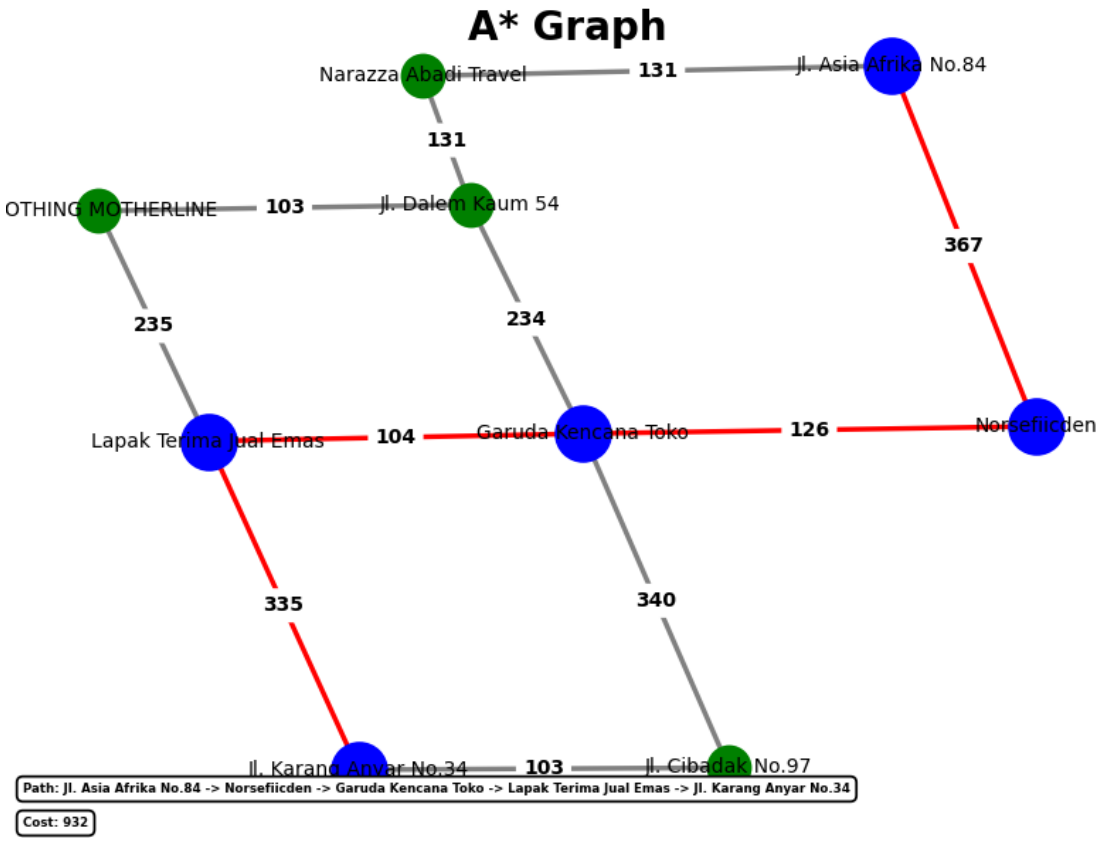
0	126	0	0	367	0	0	0	0
126	0	234	0	0	0	104	340	0
0	234	0	131	0	103	0	0	0
0	0	131	0	131	0	0	0	0
367	0	0	131	0	0	0	0	0
0	0	103	0	0	0	235	0	0
0	104	0	0	0	235	0	0	335
0	340	0	0	0	0	0	0	103
0	0	0	0	0	0	335	103	0

Norsefiicden, -6.920823, 107.604098
Garuda Kencana Toko, -6.922086, 107.604024
Jl. Dalem Kaum 54, -6.922401, 107.606344
Narazza Abadi Travel, -6.922533, 107.607648
Jl. Asia Afrika No.84, -6.921224, 107.607752
CLOTHING MOTHERLINE, -6.923438, 107.606281
Lapak Terima Jual Emas, -6.923132, 107.603948
Jl. Cibadak No.97, -6.921679, 107.600640
Jl. Karang Anyar No.34, -6.922713, 107.600619



```
SHORT PATH

Masukkan nama file : graf3.txt
Input file valid.
Menampilkan Graph...

________________________________________________________________________________

Area :
- Norsefiicden
- Garuda Kencana Toko
- Jl. Dalem Kaum 54
- Narazza Abadi Travel
- Jl. Asia Afrika No.84
- CLOTHING MOTHERLINE
- Lapak Terima Jual Emas
- Jl. Cibadak No.97
- Jl. Karang Anyar No.34
Masukkan titik awal : Jl. Asia Afrika No.84
Masukkan titik tujuan : Jl. Karang Anyar No.34
```



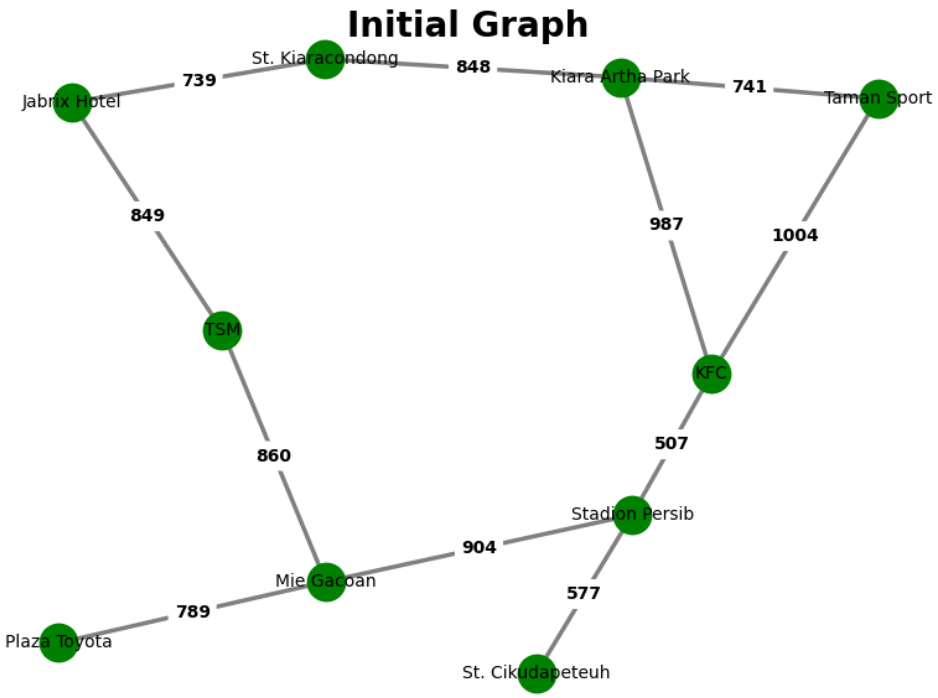

A* Graph
Narazza Abadi Travel
131
Jl. Asia Afrika No.84
131
OTHING MOTHERLINE
103
Jl. Dalem Kaum 54
367
235
234
Lapak Terima Jual Emas
104
Garuda Kencana Toko
126
Norsefiicden
335
340
Jl. Karang Anyar No.34
103
Jl. Cibadak No.97
Path: Jl. Asia Afrika No.84 -> Norsefiicden -> Garuda Kencana Toko -> Lapak Terima Jual Emas -> Jl. Karang Anyar No.34
Cost: 932


Jalan ABC

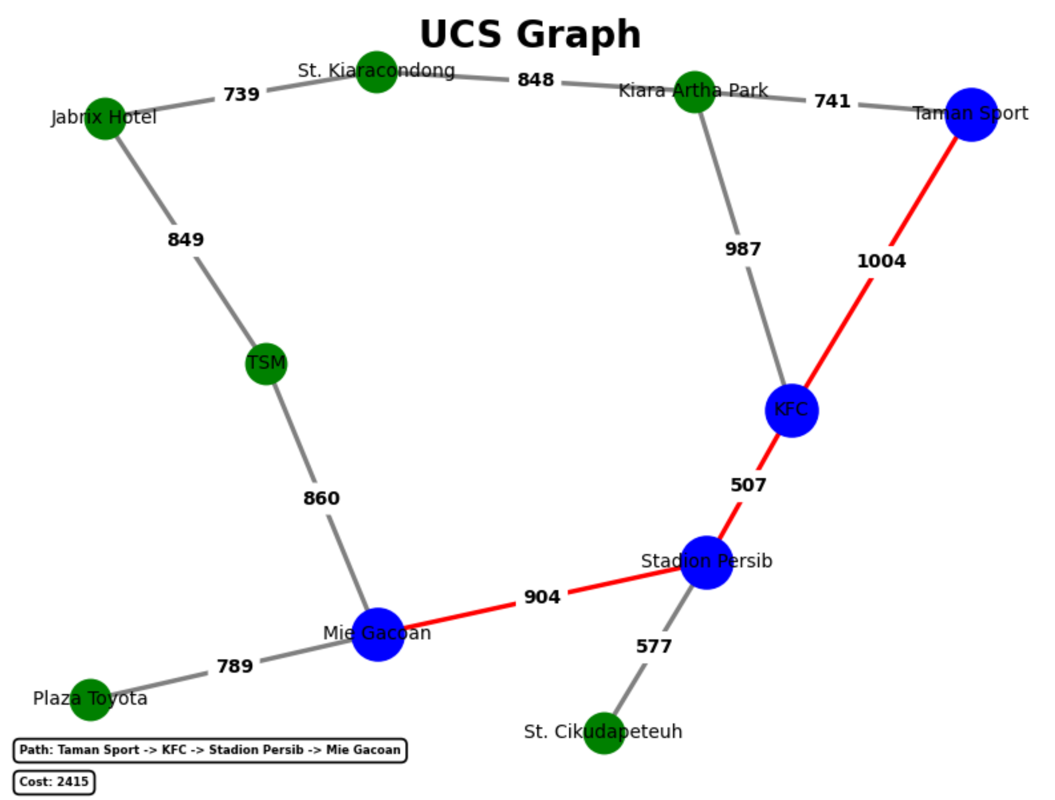
3.3 Test Case 3 : Peta jalan sekitar Buahbatu atau Bandung Selatan

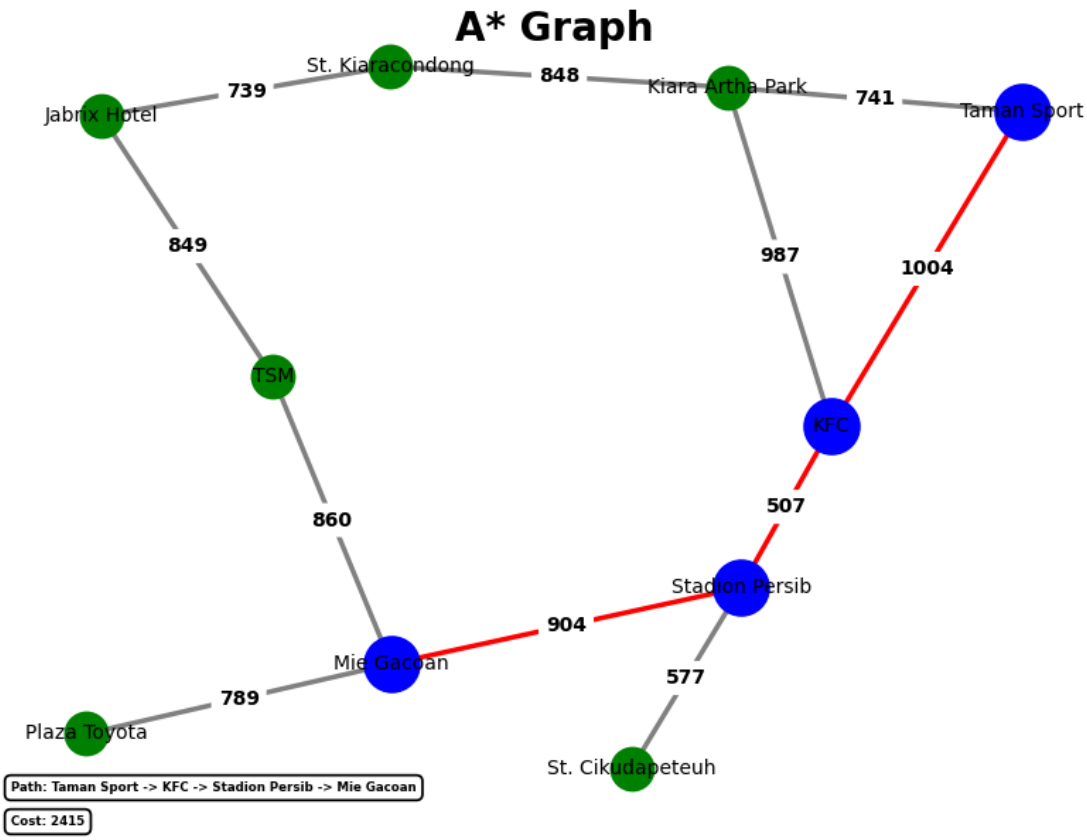
graf4.txt

0 1 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 1
1 0 0 1 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0
0 0 0 1 0 1 0 1 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 1 0 1
0 1 0 0 0 0 0 0 1 0
St. Kiaracondong, -6.924553425378429, 107.64445579682963
Kiara Artha Park, -6.9160919669451015, 107.64386913703503
Jabrix Hotel, -6.931818653404809, 107.64306750082548
TSM, -6.927510412327061, 107.63574637977118
Mie Gacoan, -6.924540250235155, 107.62767286864126
Plaza Toyota, -6.932191032147432, 107.62570745767948
St. Cikudapeteuh, -6.91850239447493, 107.62473245601261
Stadion Persib, -6.915756663252279, 107.62981717366264
KFC, -6.913477, 107.634349
Taman Sport, -6.908711, 107.643190




```
Masukkan nama file : graf4.txt
Input file valid.
Menampilkan Graph...

-----------------------------------------------------------------------------------------------

Area :
- St. Kiaracondong
- Kiara Artha Park
- Jabrix Hotel
- TSM
- Mie Gacoan
- Plaza Toyota
- St. Cikudapeteuh
- Stadion Persib
- KFC
- Taman Sport
Masukkan titik awal : Taman Sport
Masukkan titik tujuan : Mie Gacoan
```



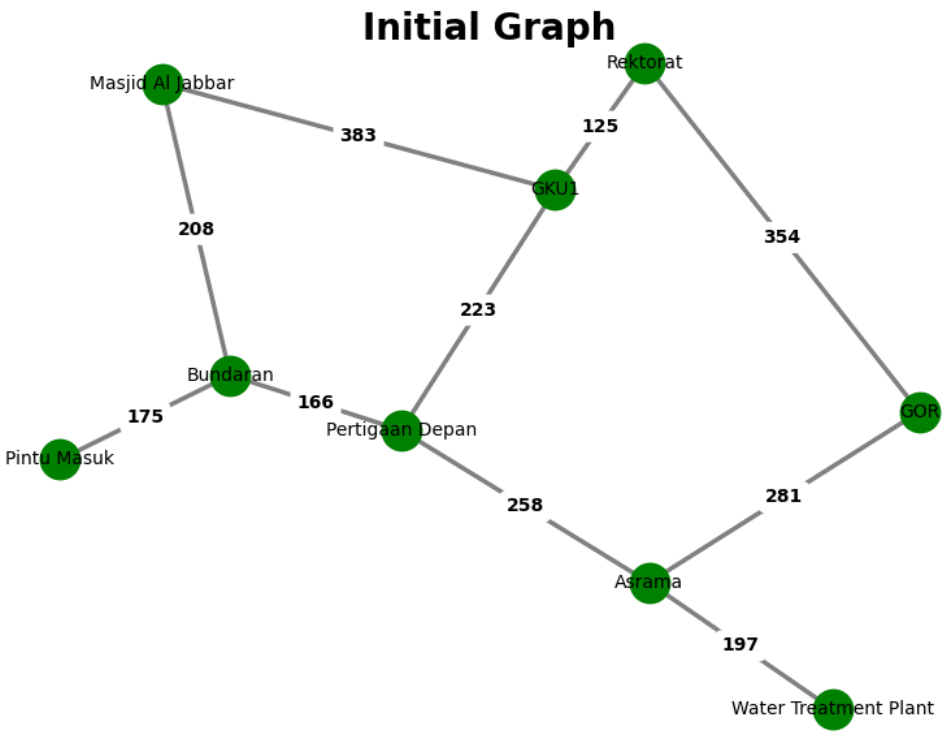

A* Graph
St. Kiaracondong
Jabnix Hotel
Kiara Artha Park
Taman Sport
TSM
KFC
Stadion Persib
Mie Gacoan
Plaza Toyota
St. Cikudapeteuh
739
848
741
849
987
1004
860
507
904
789
577
Path: Taman Sport -> KFC -> Stadion Persib -> Mie Gacoan
Cost: 2415

3.4 Test Case 4 : Peta jalan sebuah kawasan di Kabupaten Jatinangor

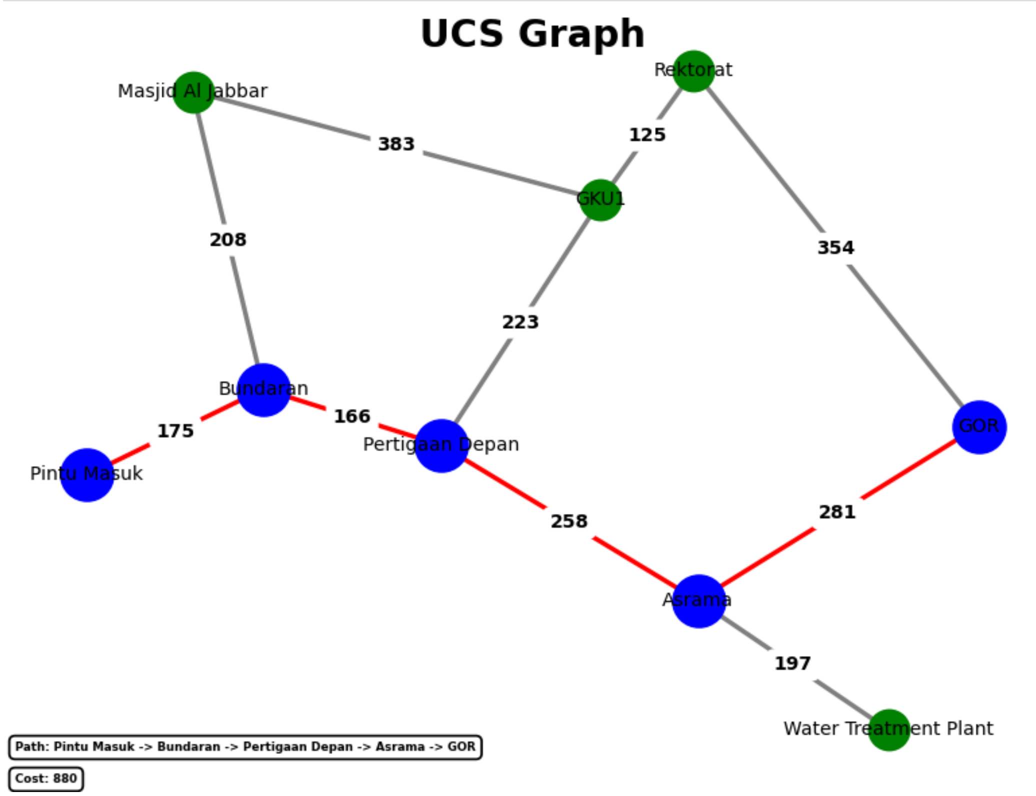
graf1.txt

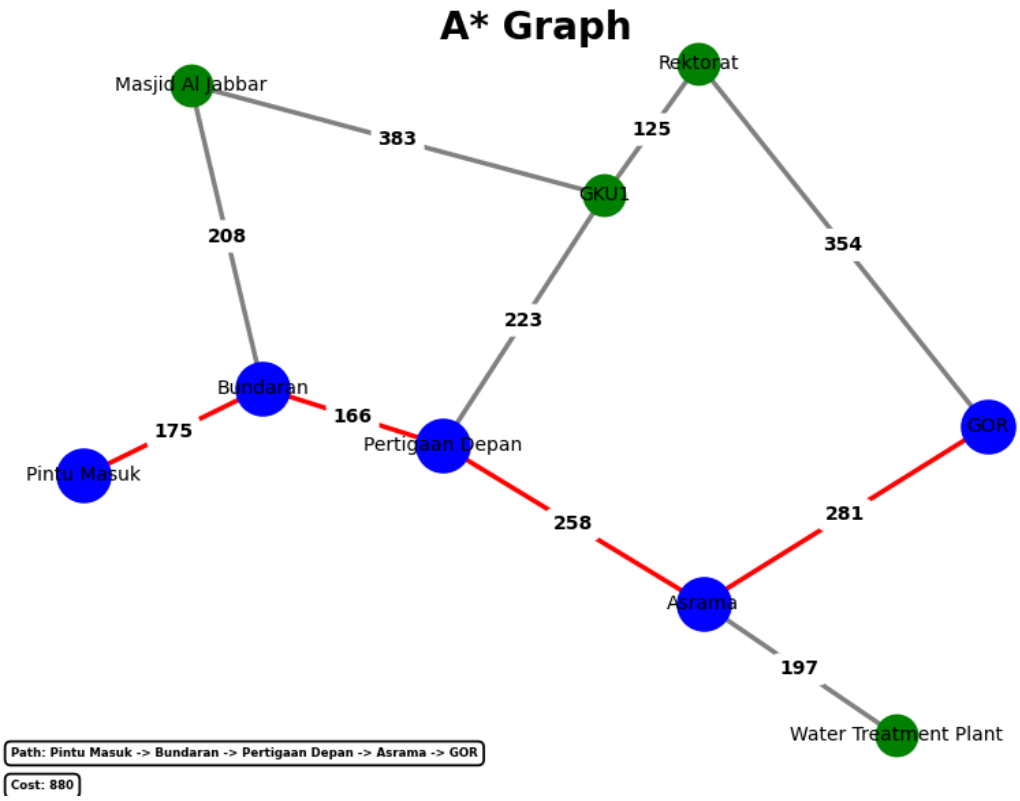
0	175	0	0	0	0	0	0	0
175	0	166	208	0	0	0	0	0
0	166	0	0	223	0	0	258	0
0	208	0	0	383	0	0	0	0
0	0	223	383	0	125	0	0	0
0	0	0	0	125	0	354	0	0
0	0	0	0	0	354	0	281	0
0	0	258	0	0	0	281	0	197
0	0	0	0	0	0	0	197	0

Pintu Masuk, -6.933273, 107.768342
Bundaran, -6.931819, 107.768872
Pertigaan Depan, -6.930354, 107.768526
Masjid Al Jabbar, -6.932403, 107.770706
GKU1, -6.929037, 107.770037
Rektorat, -6.928270, 107.770833
GOR, -6.925923, 107.768642
Asrama, -6.928236, 107.767569
Water Treatment Plant, -6.926668, 107.766774



```
Masukkan nama file : graf1.txt
Input file valid.
Menampilkan Graph...

-----------------------------------------------------------------------------------------------------------

Area :
- Pintu Masuk
- Bundaran
- Pertigaan Depan
- Masjid Al Jabbar
- GKU1
- Rektorat
- GOR
- Asrama
- Water Treatment Plant
Masukkan titik awal : Pintu Masuk
Masukkan titik tujuan : GOR
```




A* Graph
Masjid Al Jabbar
Rektorat
GKU1
Bundaran
Pertigaan Depan
Pintu Masuk
GOR
Asrama
Water Treatment Plant
383
125
208
354
223
175
166
258
281
197
Path: Pintu Masuk -> Bundaran -> Pertigaan Depan -> Asrama -> GOR
Cost: 880

BAB 4

KESIMPULAN

4.1 Kesimpulan

Berdasarkan hasil pengerjaan, lintasan terpendek dapat ditemukan dengan menggunakan algoritma UCS dan algoritma A*. Algoritma UCS (Uniform Cost Search) melakukan perhitungan dengan mempertimbangkan biaya terkecil dari simpul awal ke simpul tujuan dan atau dengan kata lain menggunakan prinsip pencarian dengan biaya seragam, sedangkan algoritma A* melakukan pencarian jalur terpendek pada graf dengan menggunakan pendekatan heuristik. Algoritma A* mencoba menggabungkan dua metode, yaitu algoritma UCS (Uniform Cost Search) dan algoritma Greedy Best First Search. Secara umum, algoritma A* lebih efisien untuk menghitung jarak terpendek antar simpul dengan syarat, nilai heuristik harus memenuhi kriteria admissibility dan consistency agar algoritma A* menghasilkan jalur terpendek yang optimal.

4.2 Komentar

Dengan adanya tugas besar 3 Strategi Algoritma ini, kami sebagai penulis merasa lebih dapat memahami algoritma UCS dan algoritma A* dengan lebih baik serta mampu mengasah kemampuan programming.

BAB 5

LAMPIRAN

4.1 Link Repository

Link : https://github.com/lostgirrlll/Tucil3_13521018_13521030

4.2 Cek List

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	✔️	✔️