

# **LAPORAN TUGAS KECIL**

**3 IF2211 STRATEGI**

**ALGORITMA**

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



Oleh:

Fazel Ginanda	13521098
Satria Octavianus Nababan	13521168

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2021**

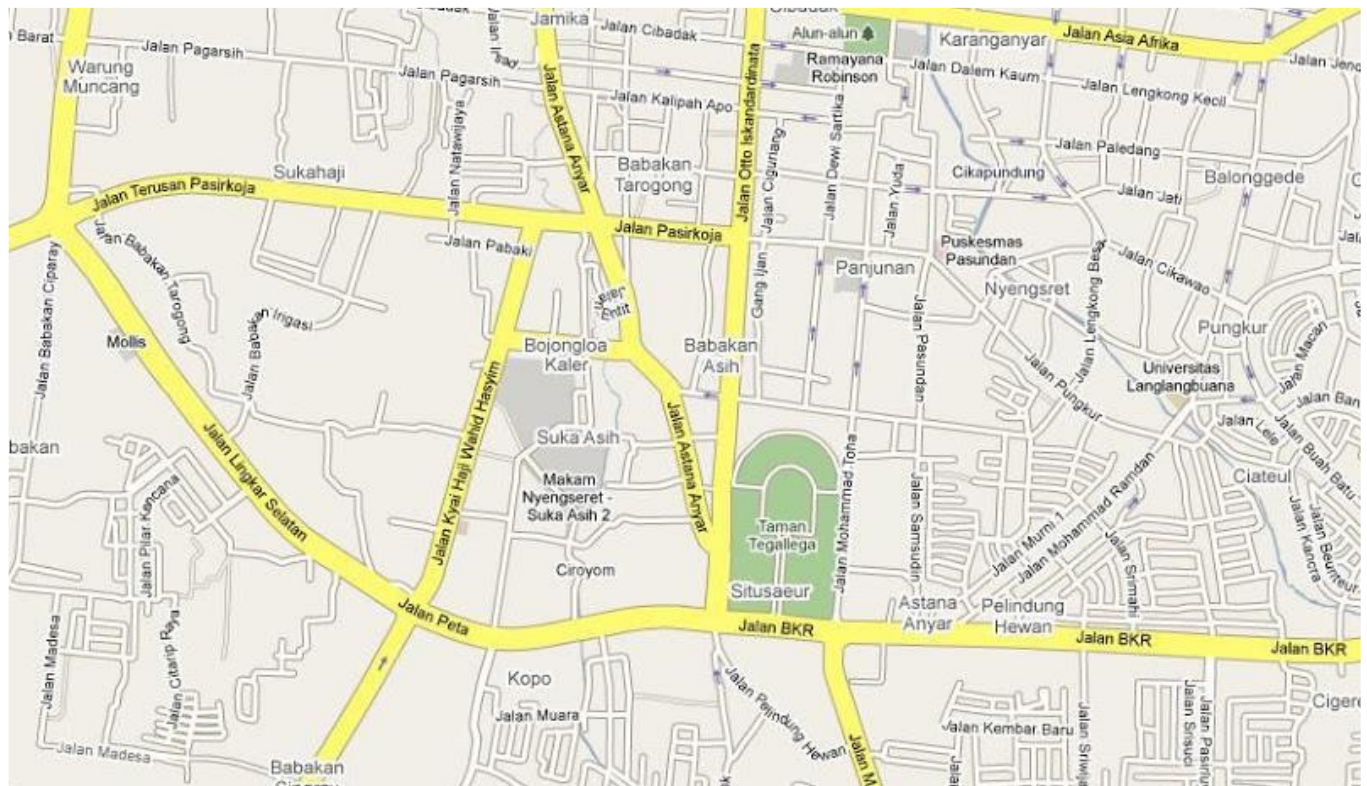
# Daftar Isi

<b>BAB 1 .....</b>	<b>3</b>
<b>DESKRIPSI PERSOALAN .....</b>	<b>3</b>
<b>BAB II.....</b>	<b>4</b>
<b>KODE PROGRAM .....</b>	<b>4</b>
Coordinate.py.....	4
Graph.py .....	4
LinkedList.py .....	5
Node.py.....	6
AStar.py .....	6
UCS.py.....	8
Utility.py .....	9
main.py .....	10
<b>BAB III .....</b>	<b>12</b>
<b>PENGUJIAN .....</b>	<b>12</b>
itb.txt.....	12
alunalunbandung.txt.....	14
Buahbatu.txt.....	15
gedebage.txt .....	17
<b>BAB IV.....</b>	<b>19</b>
<b>LAMPIRAN .....</b>	<b>19</b>
1. <i>Link</i> Kode Program .....	19
2.     Lampiran.....	19
<b>BAB V .....</b>	<b>20</b>
<b>KESIMPULAN DAN SARAN.....</b>	<b>20</b>
1.     Kesimpulan .....	20
2.     Saran .....	20
<b>Referensi .....</b>	<b>21</b>

## BAB 1

### DESKRIPSI PERSOALAN

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 Tampilan Google Maps

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.

## BAB II

### KODE PROGRAM

#### Coordinate.py

Class *coordinate* adalah struktur data yang dapat digunakan untuk merepresentasikan posisi atau titik dalam suatu sistem koordinat, dan memungkinkan program untuk dengan mudah mengakses dan memanipulasi data posisi atau titik tersebut.

```
class Coordinate:
    def __init__(self,latitude=0.0,longitude=0.0):
        self.__latitude = latitude
        self.__longitude = longitude

    def __str__(self):
        return "(" + str(self.__latitude) + ", " + str(self.__longitude) + ")"

    def __eq__(self, other):
        return ((self.getLatitude() == other.getLatitude()) and (self.getLongitude() == other.getLongitude()))
```

#### Graph.py

Class *graph* adalah sebuah struktur data yang digunakan untuk merepresentasikan kumpulan titik (*node*) maupun simpul serta hubungan (*edge*) antara mereka. Pada *class* ini juga terdapat juga representasi matriks ketetangaan yang memiliki bobot.

```
from Node import *
import networkx as nx
import matplotlib.pyplot as plt
from typing import Any

class Graph:
    def __init__(self, nodes, adjMatrix):
        self.__nodes = nodes
        self.__adjMatrix = adjMatrix

    def getNode(self,index:int) -> Node:
        return self.__nodes[index]

    def getAdjMatrix(self) -> Any:
        return self.__adjMatrix

    def getListNode(self) -> Any:
        return self.__nodes

    def displayAdjList(self):
        adjMatrix = self.getAdjMatrix()
        for i in range(len(adjMatrix)):
            print(i, end="")
```

```

        for j in range(len(adjMatrix[i])):
            if(adjMatrix[i][j] != 0):
                print(" --> " + str(j), end = ")
        print()

def normalizeGraph(self):
    graph = nx.Graph()
    for i in range(len(self.getListNode())):
        graph.add_node(i, pos=(self.getNode(i).getCoordinate()[0], self.getNode(i).getCoordinate()[1]))
    for j in range(len(self.getListNode())):
        for k in range(len(self.getListNode())):
            if(self.getAdjMatrix()[j][k]!=0):
                graph.add_edge(j, k, weight = float(self.getAdjMatrix()[j][k]) )
    return graph

def drawInputGraph(self,outputName:str):
    graph = self.normalizeGraph()
    pos=nx.get_node_attributes(graph,'pos')
    nx.draw(graph,pos,with_labels=True, font_weight='bold')
    labels = nx.get_edge_attributes(graph, 'weight')
    nx.draw_networkx_edge_labels(graph, pos, edge_labels=labels)
    plt.savefig("../test/"+outputName)
    plt.show()

def drawOutputGraph(self,Path:Any):
    graph = self.normalizeGraph()
    pos=nx.get_node_attributes(graph,'pos')
    nx.draw(graph,pos,with_labels=True, font_weight='bold')
    labels = nx.get_edge_attributes(graph, 'weight')
    nx.draw_networkx_edge_labels(graph, pos, edge_labels=labels)
    listedge = []
    for i in range(len(Path)-1):
        edge = (Path[i],Path[i+1])
        listedge.append(edge)
    nx.draw_networkx_edges(graph,pos,edgelist = listedge,edge_color="tab:red")

```

## LinkedNode.py

Class linked node adalah sebuah struktur data yang digunakan untuk merepresentasikan sebuah node dalam suatu linked list. Linked list adalah salah satu struktur data yang berguna untuk menyimpan dan mengelola kumpulan data dengan urutan tertentu.

```

class LinkedNode:
    def __init__(self, index, parent=None, path_cost=0):
        self.index = index
        self.parent = parent
        self.path_cost = path_cost

```

```
def __lt__(self, other):  
    return self.path_cost < other.path_cost
```

## Node.py

Class *Node* merepresentasikan simpul yang terdapat pada graf, terdiri atas 2 buah objek yaitu name yang bertipe *integer* dan *coordinate*.

```
from Coordinate import *  
from math import *  
class Node:  
    def __init__(self, name: int, coordinate:Coordinate):  
        self.__name = name  
        self.__coordinate = coordinate  
  
    def getName(self):  
        return self.__name  
  
    def getLatitude(self):  
        return self.__latitude  
  
    def getLongitude(self):  
        return self.__longitude  
  
    def setLatitude(self, latitude):  
        self.__latitude = latitude  
  
    def setLongitude(self, longitude):  
        self.__longitude = longitude  
  
    def getCoordinate(self):  
        return self.__coordinate  
  
    def setCoordinate(self, coordinate):  
        self.__coordinate = coordinate  
  
    def __str__(self):  
        return "Node " + self.__name.__str__() + ": " + self.__coordinate.__str__()  
  
    def __eq__(self, other):  
        return ((self.getName() == other.getName()) and (self.getCoordinate() == other.getCoordinate()))
```

## AStar.py

Algoritma A\* adalah algoritma pencarian jalur terpendek yang digunakan untuk mencari jalur terpendek

antara dua titik dalam graf atau peta. Algoritma ini menggabungkan metode Dijkstra dengan heuristik(*haversine*) untuk mempercepat pencarian jalur terpendek.

```
import heapq
from ListNode import *
from Graph import *

def aStar(graph:Graph, coordinate,startNode:int, goalNode:int,outputFileName:str):
    liveNodes = []
    heapq.heappush(liveNodes, ListNode(startNode))

    visitedNodes = set()

    while liveNodes:
        expandNode = heapq.heappop(liveNodes)

        if expandNode.index == goalNode:
            path = []
            while expandNode:
                path.append(expandNode.index)
                expandNode = expandNode.parent
            path.reverse()
            return showPath(graph, path, startNode, goalNode,outputFileName)

        visitedNodes.add(expandNode.index)

        for i in range(len(graph.getAdjMatrix())):
            if graph.getAdjMatrix()[expandNode.index][i] != 0 and i not in visitedNodes:
                radius = 6371
                lat1 = radians(coordinate[i][0])
                lat2 = radians(coordinate[goalNode][0])
                dLat = radians(lat2 - lat1)
                dLon = radians(coordinate[goalNode][1] - coordinate[i][1])
                a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
                c = 2*asin(sqrt(a))
                distance = radius * c
                new_path_cost = expandNode.path_cost + graph.getAdjMatrix()[expandNode.index][i] + distance
                heapq.heappush(liveNodes, ListNode(i, expandNode, new_path_cost))
    return displayPathNotFound()

def displayPathNotFound():
    print("Tidak ada lintasan dari simpul asal ke simpul tujuan")

def showPath(graph:Graph,path:Any, startNode:int, goalNode:int,outputFileName:str):
    print(f"Lintasan terpendek dari simpul {startNode} ke {goalNode} adalah ",end="")
    i = 0
    while(i < len(path)):
```

```

    print(path[i],end="")
    i = i + 1
    if (i < len(path)):
        print(" --> ", end="")
    print(f" dengan panjang lintasan sebesar {sum(graph.getAdjMatrix()[path[i-1]][path[i]] for i in range(1, len(path)))}")
    graph.drawOutputGraph(path)
    plt.savefig("../test/"+outputFileName)
    plt.show()

```

## UCS.py

Algoritma *Uniform Cost Search* (UCS) adalah algoritma pencarian jalur terpendek yang menggunakan pendekatan berbasis biaya, di mana setiap langkah memiliki nilai biaya yang berbeda-beda. Algoritma UCS mencari jalur terpendek dari titik awal ke titik tujuan dengan mempertimbangkan biaya setiap langkah dalam graf.

```

import heapq
from ListNode import *
from Graph import *
import matplotlib.pyplot as plt

def uniformCostSearch(graph:Graph, startNode:int, goalNode:int,outputFileName:str):
    liveNodes = []
    heapq.heappush(liveNodes, ListNode(startNode))
    visitedNodes = set()

    while liveNodes:
        expandNode = heapq.heappop(liveNodes)
        if expandNode.index == goalNode:
            path = []
            while expandNode:
                path.append(expandNode.index)
                expandNode = expandNode.parent
            path.reverse()
            return showPath(graph, path, startNode, goalNode, outputFileName)

        visitedNodes.add(expandNode.index)

    for i in range(len(graph.getAdjMatrix())):
        if graph.getAdjMatrix()[expandNode.index][i] != 0 and i not in visitedNodes:
            new_path_cost = expandNode.path_cost + graph.getAdjMatrix()[expandNode.index][i]
            heapq.heappush(liveNodes, ListNode(i, expandNode, new_path_cost))
    return displayPathNotFound()

def displayPathNotFound():
    print("Tidak ada lintasan dari simpul asal ke simpul tujuan")

```



```

def showPath(graph:Graph,path:Any, startNode:int, goalNode:int, outputFileName:str):
    print(f"Lintasan terpendek dari simpul {startNode} ke {goalNode} adalah ",end="")
    i = 0
    while(i < len(path)):
        print(path[i],end="")
        i = i + 1
        if (i < len(path)):
            print(" --> ", end="")
    print(f" dengan panjang lintasan sebesar {sum(graph.getAdjMatrix()[path[i-1]][path[i]] for i in range(1, len(path)))}")
    graph.drawOutputGraph(path)
    plt.savefig("../test/"+outputFileName)
    plt.show()

```

## Utility.py

```

def read_file(filename):
    file = filename.read().splitlines()
    banyak = int(file[0])
    matriks = []
    koordinat = []
    matriksToFloat = [[0 for i in range(banyak)] for j in range(banyak)]
    CoorToFloat = [[0 for i in range(2)] for j in range(banyak)]
    for i in range(banyak):
        line = file[i+2]
        matriks.append(line)
        koor = file[i+2+banyak]
        koordinat.append(koor)
        matriksToFloat[i] = [float(x) for x in matriks[i].split()]
        CoorToFloat[i] = [float(x) for x in koordinat[i].split()]
        if(isSquareMatrix(matriksToFloat) and isSymmetricMatrix(matriksToFloat) and
isNumberEqualsMatrix(matriksToFloat)):
            return file,matriksToFloat,CoorToFloat
        else:
            raise Exception

def isSquareMatrix(matrix:list[list[float]]) -> bool:
    square = True
    rowCount = len(matrix)
    columnCount = len(matrix[0])
    if(rowCount != columnCount):
        square = False
    else:
        i = 0
        while(i < len(matrix) and square):
            if (len(matrix[i]) != columnCount):
                square = False

```

```

        i = i + 1
    return square

def isNumberEqualsMatrix(number:int,matrix:list[list[int]]) -> bool:
    if(number == len(matrix)):
        return True
    else:
        return False

def isSymmetricMatrix(matrix:list[list[float]]) -> bool:
    symmetric = True
    i = 0
    while (i < len(matrix) and symmetric):
        j = 0
        while(j < len(matrix[i]) and symmetric):
            if(matrix[i][j] != matrix[j][i]):
                symmetric = False
            j = j + 1
        i = i + 1
    return symmetric

```

## main.py

```

from Node import *
from Graph import *
from UCS import *
from AStar import *
from Utility import *

if __name__ == "__main__":
    print("=====")
    print("PROGRAM PENCARIAN LINTASAN TERDEKAT")
    print("=====")
    print()
    filename = input("Masukkan nama file: ")
    try:
        file = open("../test/" + filename)
        try:
            listName, adjMatrix, listCoordinate = read_file(file)
        except Exception as e:
            print("Input graf tidak valid")
            exit()
        listNode = []
        for i in range (len(listCoordinate)):
            newNode = Node(i,listCoordinate[i])
            listNode.append(newNode)

```

```

inputGraph = Graph(listNode, adjMatrix)
print("Visualisasi graf dengan representasi list ketetanggaan:")
inputGraph.displayAdjList()
graphInputName = filename.split(".")[0] + ".png"
graphUCSName = filename.split(".")[0] + "PathUCS.png"
graphAStarName = filename.split(".")[0] + "PathAStar.png"
inputGraph.drawInputGraph(graphInputName)

try:
    startNode = int(input("Masukkan simpul asal: "))
except IndexError as e:
    print("Simpul asal tidak ada pada graf")
    exit()

try:
    goalNode = int(input("Masukkan simpul tujuan: "))
except IndexError as e:
    print("Simpul tujuan tidak ada pada graf")
    exit()

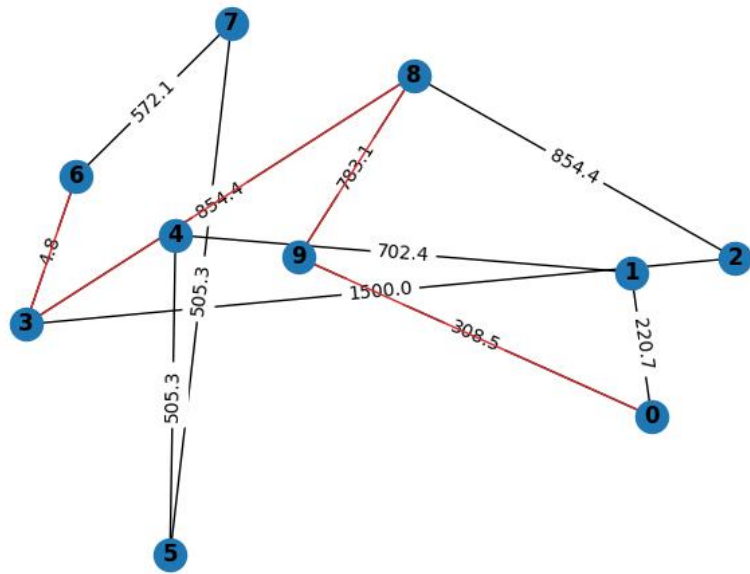
print("Pilih metode yang ingin digunakan:")
print("1. UCS")
print("2. A*")
method = int(input("Masukkan pilihan metode (1 atau 2): "))
if method == 1:
    uniformCostSearch(inputGraph, startNode, goalNode, graphUCSName)
elif method == 2:
    aStar(inputGraph, listCoordinate, startNode, goalNode, graphAStarName)
else:
    print("Pilihan salah")
except FileNotFoundError as e:
    print("File tidak ditemukan pada folder test")
    exit()
except ValueError as e:
    print("Input graf tidak valid")
    exit()

```

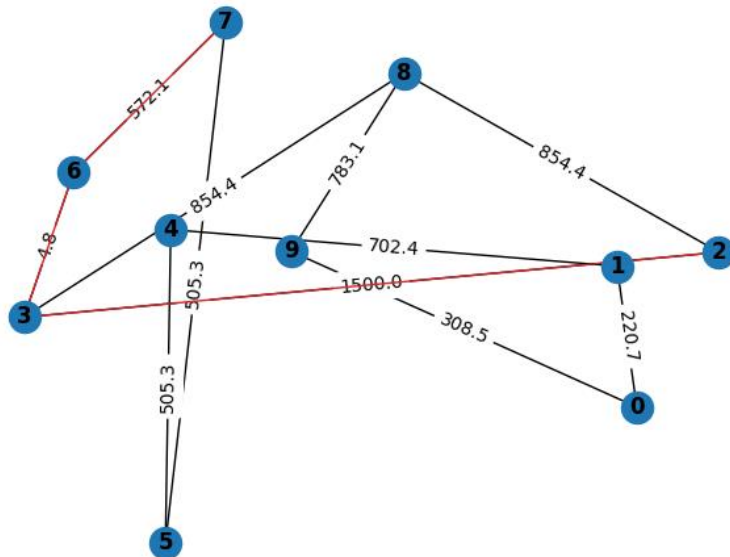
## BAB III PENGUJIAN

itb.txt		
Tampilan file input program		<pre> 10 0 220.7 0 0 0 0 0 0 0 308.5 220.7 0 0 0 702.4 0 0 0 0 0 0 0 0 1500 0 0 0 0 854.4 0 0 0 1500 0 0 0 4.8 0 0 0 0 702.4 0 0 0 505.3 0 0 0 0 0 0 0 0 505.3 0 0 505.3 0 0 0 0 0 4.8 0 0 0 572.1 0 0 0 0 0 0 0 505.3 572.1 0 0 0 0 0 0 854.4 0 0 0 0 783.1 308.5 0 0 0 0 0 0 0 783.1 0 -6.886972 107.611525 -6.887387 107.613478 -6.885252 107.613671 -6.899789 107.612770 -6.896719 107.613985 -6.896830 107.609637 -6.898747 107.614781 -6.895570 107.616866 -6.891842 107.616152 -6.894204 107.613684 </pre>
Tampilan awal graf		

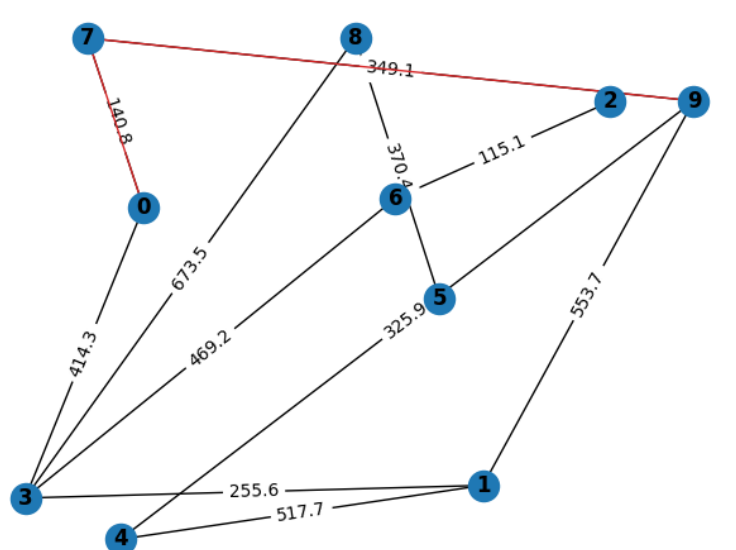
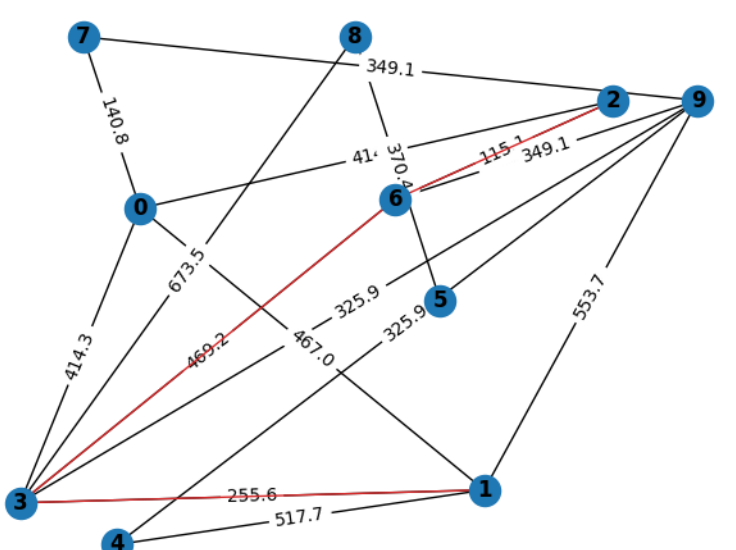
Path graf hasil UCS  
dengan simpul asal 0  
dan simpul tujuan 6.



Path hasil A\* dengan  
simpul asal 7 dan  
simpul tujuan 2.

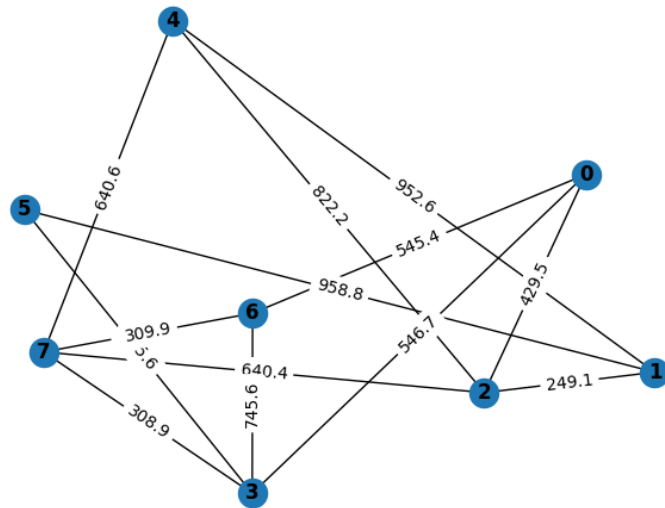


alunalunbandung.txt		
Tampilan file input program	<pre> 10 0 0 0 414.3 0 0 0 140.8 0 0 0 0 0 255.6 517.7 0 0 0 0 553.7 0 0 0 0 0 0 115.1 0 0 0 0 255.6 0 0 0 0 0 0 0 0 0 517.7 0 0 0 0 0 0 0 325.9 0 0 0 0 0 0 0 0 370.4 0 0 0 115.1 469.2 0 0 0 0 0 0 140.8 0 0 0 0 0 0 0 0 349.1 0 0 0 673.5 0 370.4 0 0 0 0 0 0 0 325.9 0 0 349.1 0 0 -6.922516 107.607628 -6.920806 107.604088 -6.920165 107.608976 -6.923112 107.603927 -6.922633 107.603401 -6.921024 107.606464 -6.921248 107.607741 -6.922798 107.609783 -6.921448 107.609789 -6.919744 107.608984 </pre>	
Tampilan awal graf		

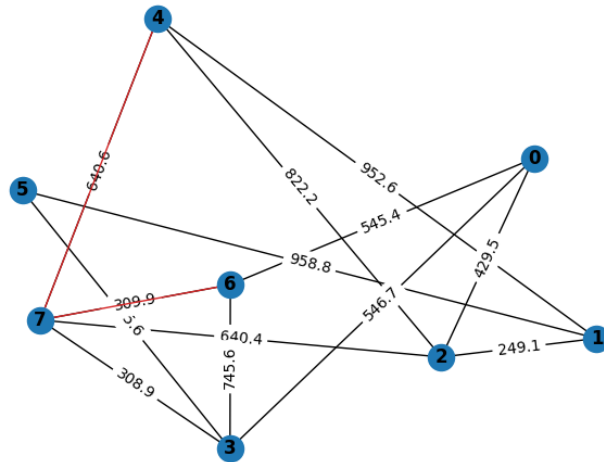
<p>Path hasil UCS dengan simpul asal 9 dan simpul tujuan 0</p>	
<p>Path hasil A* dengan simpul asal 1 dan simpul tujuan 2</p>	

Buahbatu.txt		
Tampilan file input program	<pre> 8 0 1 2 3 4 5 6 7 0 0 429.5 546.7 0 0 545.4 0 0 0 249.1 0 952.6 958.8 0 0 429.5 249.1 0 0 822.2 0 0 640.4 546.7 0 0 0 0 745.6 308.9 0 952.6 822.2 0 0 0 0 640.6 0 958.8 0 745.6 0 0 0 0 545.4 0 0 0 0 0 309.9 0 0 640.4 308.9 640.6 0 309.9 0 -6.947938 107.650525 -6.947038 107.647215 -6.949264 107.646872 -6.952294 107.645186 -6.953332 107.653100 -6.955271 107.649935 -6.952289 107.648202 -6.955015 107.647553 </pre>	

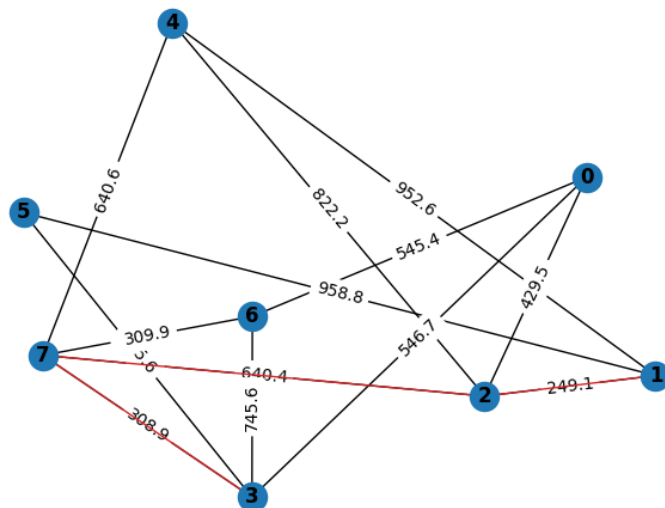
Tampilan awal graf



Path hasil UCS dengan simpul asal 6 dan simpul tujuan 4

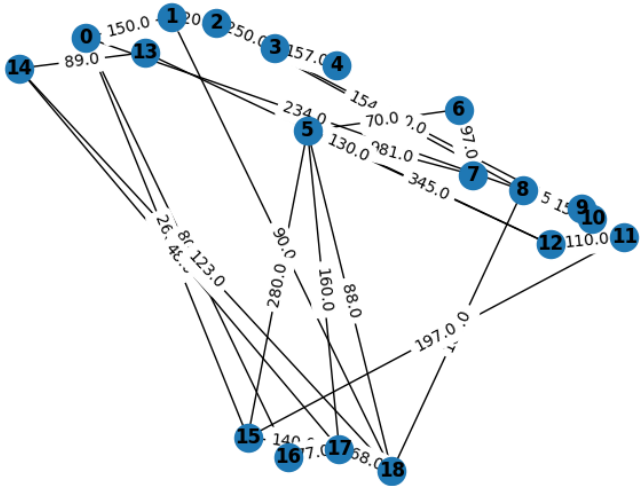
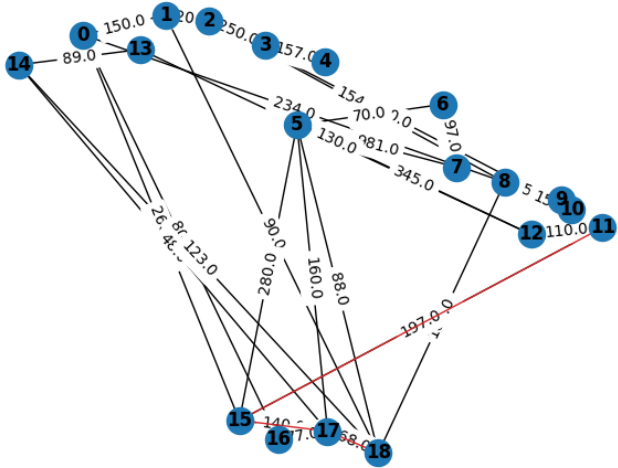
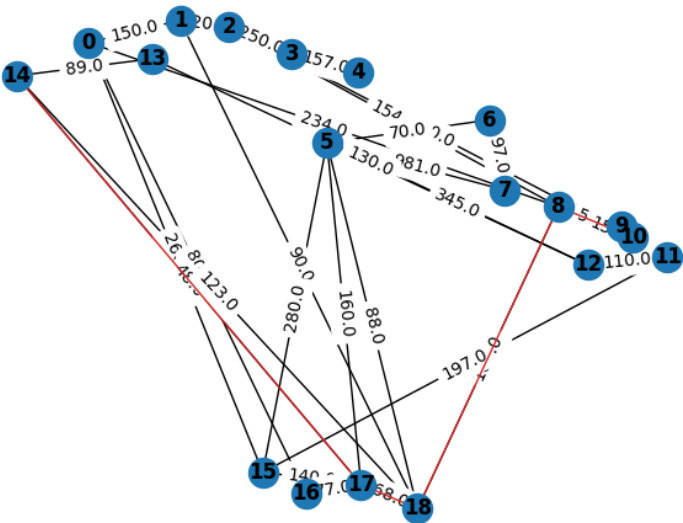


Path hasil A\* dengan simpul asal 1 dan simpul tujuan 3





gedebage.txt		
Tampilan file input program		<pre> 1  19 2  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 3  0 150 0 0 0 0 0 0 234 0 0 0 0 0 0 0 0 0 4  150 0 120 0 270 0 0 0 0 0 0 0 0 0 0 0 0 0 5  0 120 0 250 0 0 0 0 154 0 980 0 0 0 0 0 0 0 6  0 0 250 0 157 0 0 0 0 0 0 0 0 0 0 0 0 0 7  0 270 0 157 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8  0 0 0 0 0 0 70 981 0 0 0 0 345 0 0 280 0 0 88 9  0 0 0 0 0 70 0 97 0 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 981 97 0 0 0 0 0 0 0 0 0 0 0 0 11 234 0 154 0 0 0 0 0 0 78 50 155 0 0 0 0 0 0 0 12 0 0 0 0 0 0 0 78 0 0 0 0 0 0 0 0 0 0 0 13 0 0 980 0 0 0 0 0 50 0 0 0 0 0 0 0 0 0 0 14 0 0 0 0 0 0 0 155 0 0 0 110 0 0 197 0 0 0 15 0 0 0 0 0 345 0 0 0 0 0 110 0 130 0 0 0 0 0 16 0 0 0 0 0 0 0 0 0 0 0 130 0 89 0 0 0 0 0 17 0 0 0 0 0 0 0 0 0 0 0 0 89 0 0 0 48 123 18 260 0 0 0 0 280 0 0 0 0 197 0 0 0 0 0 140 0 19 80 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 77 0 20 0 0 0 0 0 160 0 0 0 0 0 0 0 48 140 0 0 68 21 0 90 0 0 0 88 0 0 160 0 0 0 0 123 0 0 68 0 22 -6.956513 107.695354 23 -6.954175 107.697232 24 -6.952926 107.696673 25 -6.951307 107.694355 26 -6.949624 107.692826 27 -6.950412 107.686860 28 -6.946252 107.688806 29 -6.945852 107.682868 30 -6.944478 107.681452 31 -6.942869 107.679753 32 -6.942573 107.678849 33 -6.941681 107.677186 34 -6.943726 107.676529 35 -6.954890 107.693983 36 -6.958372 107.692529 37 -6.952057 107.658985 38 -6.950950 107.657178 39 -6.949538 107.657961 40 -6.948090 107.655906 </pre>

<p>Tampilan awal graf</p>	
<p>Path hasil UCS dengan simpul asal 11 dan simpul tujuan 18</p>	
<p>Path hasil A* dengan simpul asal 14 dan simpul tujuan 10</p>	

## BAB IV LAMPIRAN

### 1. *Link* Kode Program

Berikut merupakan *link* repositori Github yang mengandung *source code* program ini.

[https://github.com/satrianababan/Tucil3\\_13521098\\_13521168](https://github.com/satrianababan/Tucil3_13521098_13521168)

### 2. Lampiran

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	

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **1. Kesimpulan**

Algoritma Uniform Cost Search (UCS) dan A\* merupakan strategi yang dapat digunakan untuk mencari jalur terpendek dari suatu titik asal ke titik tujuan dan dijamin akan selalu menemukan hasil yang optimal, tetapi akan memerlukan ruang pencarian yang luas dan juga waktu yang lebih lama dibanding algoritma traversal yang telah dipelajari sebelumnya, seperti DFS dan BFS.

#### **2. Saran**

Spesifikasi tugas besar ini seharusnya dibuat dengan lebih detail sehingga dapat lebih jelas dan mahasiswa tidak mengalami kebingungan dalam pengerjaannya.

## Referensi

Munir, Rinaldi (2022). *Bahan Kuliah IF2211 Strategi*. Diakses pada 23 Januari 2023 pukul 23.10 dari sumber <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

Munir, Rinaldi (2022). *Bahan Kuliah IF2211 Strategi*. Diakses pada 23 Januari 2023 pukul 23.10 dari sumber <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>