**IF2211**
**Strategi Algoritma**

# LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA

# IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

**Kelas Mahasiswa K03**

**Dosen: Ir. Rila Mandala, M.Eng., Ph.D.**

**Diajukan sebagai tugas kecil Mata Kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023**


**Disusun Oleh:**

| | |
|---|---|
| **Muhamad Salman Hakim Alfarisi** | **13521010** |
| **Eunice Sarah Siregar** | **13521013** |

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2023**

# DAFTAR ISI

# BAB I

# PENDAHULUAN

## 1.1 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 antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

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.

## 1.2 Teori Singkat

Algoritma Uniform Cost Search (UCS) adalah pencarian rute yang menggunakan biaya pencarian terendah untuk menemukan jalur dari titik awal ke titik tujuan. Algoritma UCS merupakan salah satu jenis algoritma pencarian dari uniform search karena tidak mempertimbangkan ruang pencarian. Sedangkan algoritma A* merupakan salah satu contoh algoritma informed search algorithm dengan melihat kombinasi nilai dari *path*-nya dengan nilai *heuristic*.

# BAB II

# IMPLEMENTASI PROGRAM

## 2.1 Algoritma UCS

```
import heapq

def ucs(graph, startString, goalString, nodes):
    start = nodes.index(startString)
    goal = nodes.index(goalString)
    queue = [(0, start, [])]
    visited = set()
    while queue:
        (cost, currNode, path) = heapq.heappop(queue)
        if currNode == goal:
            return path + [currNode], cost
        if currNode not in visited:
            visited.add(currNode)
            for neighbor in range(len(graph[currNode])):
                if graph[currNode][neighbor] != 0:
                    newCost = cost + graph[currNode][neighbor]
                    heapq.heappush(queue, (newCost, neighbor, path
+ [currNode]))
    return None, None
```

### 2.2 Algoritma A*

```
import heapq
import numpy as np

def aStar(graph, startString, goalString, nodes, coordinates):
    start = nodes.index(startString)
    goal = nodes.index(goalString)
    costQueue = [(0, start, [])]
    prioQueue = [(straightLine(coordinates[start],
coordinates[goal]), start)]
    visited = set()
    while costQueue:
        (cost, currNode, path) = heapq.heappop(costQueue)
        if currNode == goal:
            return path + [currNode], cost
        if currNode not in visited:
            visited.add(currNode)
            for neighbor in range(len(graph[currNode])):
                if graph[currNode][neighbor] != 0:
                    newCost = cost + graph[currNode][neighbor]
                    newPath = path + [currNode]
                    heapq.heappush(costQueue, (newCost, neighbor,
newPath))
                    heuristicCost =
straightLine(coordinates[neighbor], coordinates[goal])
                    heapq.heappush(prioQueue, (newCost +
heuristicCost, neighbor))
    return None, None

def straightLine(coordinate1, coordinate2):
    return np.linalg.norm(np.array(coordinate1) -
np.array(coordinate2))
```

### 2.3 main.py

```
from readFile import *
from ucs import *
from astar import *
from visualize import *
import sys
import time

def main():
    while True:
```

```
        print("Welcome to the Path Finder!")
        print("Please choose the algorithm you want to use:")
        print("1. Uniform Cost Search")
        print("2. A* Search")
        print("Press any key to exit program")
        choice = input("Your choice: ")
        if choice == '1':
            while True:
                filename = input("Please enter the filename: ")
                nodes, graph, coordinates = readFile(filename)
                if nodes is None and graph is None:
                    continue
                print("The map is: ")
                print(nodes)
                start = input("Please enter the start node: ")
                goal = input("Please enter the goal node: ")
                if start not in nodes or goal not in nodes:
                    print("Start or goal node not found!")
                    continue
                path, cost = ucs(graph, start, goal, nodes)
                if path is None and cost is None:
                    print("No path found!")
                    continue
                else:
                    for i in range(len(path)):
                        path[i] = nodes[path[i]]
                    printPath(path, cost)
                    show = input("Do you want to see the graph?
(y/n): ")
                    if show == 'y':
                        showGraph(nodes, graph, path, coordinates)
                        print("Thank you for using our program!")
                break
        elif choice == '2':
            while True:
                filename = input("Please enter the filename: ")
                nodes, graph, coordinates = readFile(filename)
                if nodes is None and graph is None:
                    continue
                print("The map is: ")
                print(nodes)
                start = input("Please enter the start node: ")
                goal = input("Please enter the goal node: ")
                if start not in nodes or goal not in nodes:
                    print("Start or goal node not found!")
                    continue
                path, cost = aStar(graph, start, goal, nodes,
coordinates)
                if path is None and cost is None:
                    print("No path found!")
```

```
                    continue
                else:
                    for i in range(len(path)):
                        path[i] = nodes[path[i]]
                    printPath(path, cost)
                    show = input("Do you want to see the graph?
(y/n): ")
                    if show == 'y':
                        showGraph(nodes, graph, path, coordinates)
                        print("Thank you for using our program!")
                break
        elif (choice != '1' and choice != '2'):
                print("Bye!")
                time.sleep(2)
                sys.exit()


if __name__ == "__main__":
    main()
```

**2.4 visualize.py**

```
import networkx as nx
import matplotlib.pyplot as plt
import gmplot
import webbrowser
import os

def printPath(path, cost):
    if path is None and cost is None:
        print("Tidak ditemukan path menuju goal node!")
    else:
        print("Path: ", end='')
        for i in range(len(path)):
            if i == len(path) - 1:
                print(path[i])
            else:
                print(path[i], end=' -> ')
        print("Cost: ", cost)

def showGraph(nodes, graph, path, coordinates):
    print("Showing graph...")
    showMap(nodes, graph, path, coordinates)
    webbrowser.open(os.getcwd() + "\\bin\\map.html")

    # membuat graph
```

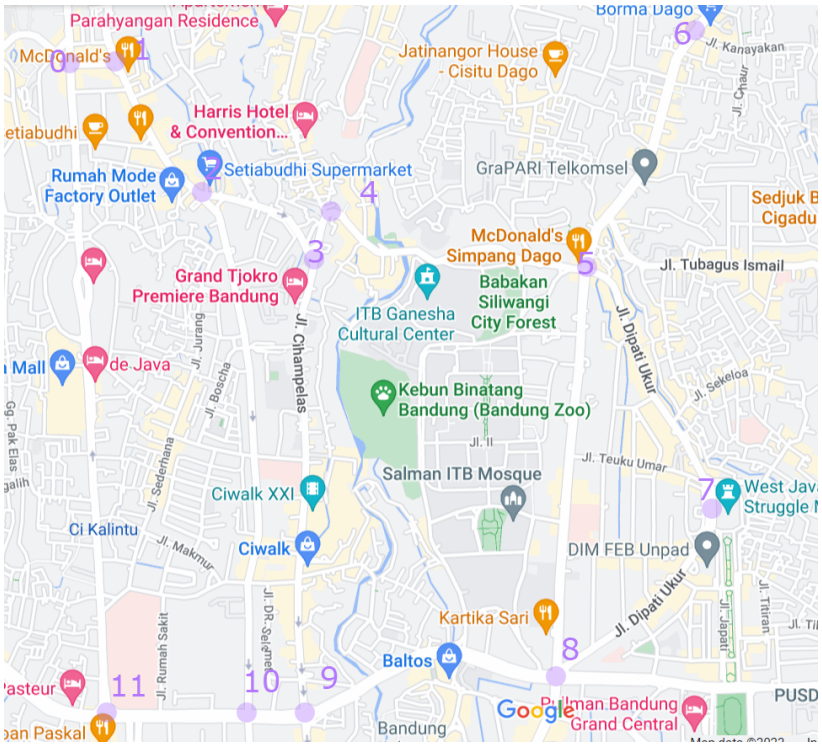
```
    G = nx.Graph()
    for i in range(len(nodes)):
        G.add_node(nodes[i])
        for j in range(i+1, len(nodes)):
            if graph[i][j] != 0:
                G.add_edge(nodes[i], nodes[j], weight=graph[i][j])

    # mengatur warna dan label untuk nodes
    nodeColors = []
    nodeLabels = {}
    for node in G.nodes():
        if node in path:
            nodeColors.append('#d3b2d9')
        else:
            nodeColors.append('#b2d3d9')
        nodeLabels[node] = node

    # mengatur warna dan label untuk edges
    edgeColors = []
    edgeLabels = {}
    for u, v, data in G.edges(data=True):
        if (u in path and v in path):
            edgeColors.append('#5e2d61')
        else:
            edgeColors.append('black')
        edgeLabels[(u,v)] = data['weight']

    # plotting graph dengan warna dan label
    pos = {nodes[i]: coordinates[i][::-1] for i in
range(len(nodes))}
    nx.draw_networkx_nodes(G, pos, node_color=nodeColors,
node_size=500)
    nx.draw_networkx_edges(G, pos, edge_color=edgeColors)
    nx.draw_networkx_labels(G, pos, nodeLabels, font_size=8,
font_weight='bold')
    nx.draw_networkx_edge_labels(G, pos, edgeLabels, font_size=8,
font_weight='bold')
    plt.show()

def showMap(nodes, graph, path, coordinates):
    avgLat = sum([coordinates[i][0] for i in
range(len(coordinates))]) / len(coordinates)
    avgLng = sum([coordinates[i][1] for i in
range(len(coordinates))]) / len(coordinates)
    gmap = gmplot.GoogleMapPlotter(avgLat, avgLng,
apikey="AIzaSyB5UAh67qqEWkt8i2VH6AMD3KJgIdx4vNI", use_api=True,
map_type="roadmap", zoom=14, title="Shortest Path")

    # Plotting semua nodes
    for i, node in enumerate(nodes):
```

```
        lat, lng = coordinates[i]
        gmap.marker(lat, lng, title=node)

    # Plotting semua edges
    for i in range(len(graph)):
        for j in range(i+1, len(graph)):
            if graph[i][j] != 0:
                lat1, lng1 = coordinates[i]
                lat2, lng2 = coordinates[j]
                if (nodes[i] in path and nodes[j] in path):
                    gmap.plot([lat1, lat2], [lng1, lng2], 'red',
edge_width=5)
                else:
                    gmap.plot([lat1, lat2], [lng1, lng2], 'blue',
edge_width=5)

    # Menampilkan map dalam file html
    gmap.draw("bin/map.html")
```

# BAB III

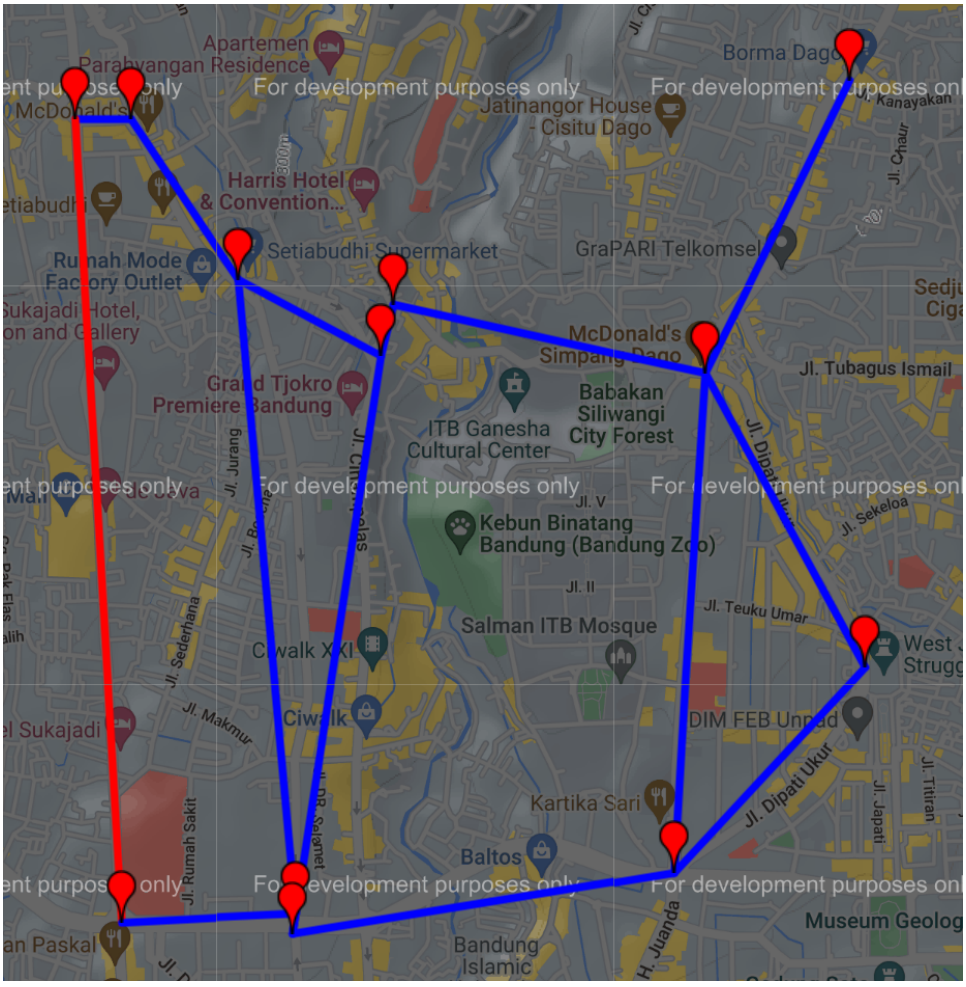
# HASIL

## A. Test Case Map bandungatas

| Map |
| --- |
|   |
| **.txt** |
| UNPAR, MCD UNPAR, Setia Budi Market, Grand Tjokro, Cihampelas, MCD Dago, Borma Dago, UNPAD DU, Baltos, Ciwalk, Boscha, Pasteru<br>0 157 0 0 0 0 0 0 0 0 0 2410<br>157 0 598 0 0 0 0 0 0 0 0 0<br>0 598 0 438 0 0 0 0 0 0 1950 0<br>0 0 438 0 192 0 0 0 0 1700 0 0<br>0 0 0 192 0 500 0 0 0 0 0 0<br>0 0 0 0 500 0 1000 1030 1501 0 0 0<br>0 0 0 0 0 1000 0 0 0 0 0 0<br>0 0 0 0 0 1030 0 0 860 0 0 0<br>0 0 0 0 0 1501 0 860 0 960 0 0<br>0 0 0 1700 0 0 0 0 960 0 227 0 |

0 0 1950 0 0 0 0 0 0 227 0 502
2410 0 0 0 0 0 0 0 0 0 502 0
-6.878250868322604,107.59617516817069
-6.878250868322604,107.59772012054502
-6.882639315099353,107.60068127926249
-6.88470570560123,107.60462949088577
-6.8832997089183445,107.60497281363561
-6.88519567318329,107.61362025551486
-6.8771857051969665,107.61761138253057
-6.8932481112060815,107.61801907833657
-6.898850652667312,107.6127404910264
-6.900533532435832,107.60220477404731
-6.899958371706619,107.60226914700256
-6.9002139987935625,107.59748408617651

**Map Hasil**



**IF2211**
**Strategi Algoritma**

## UCS

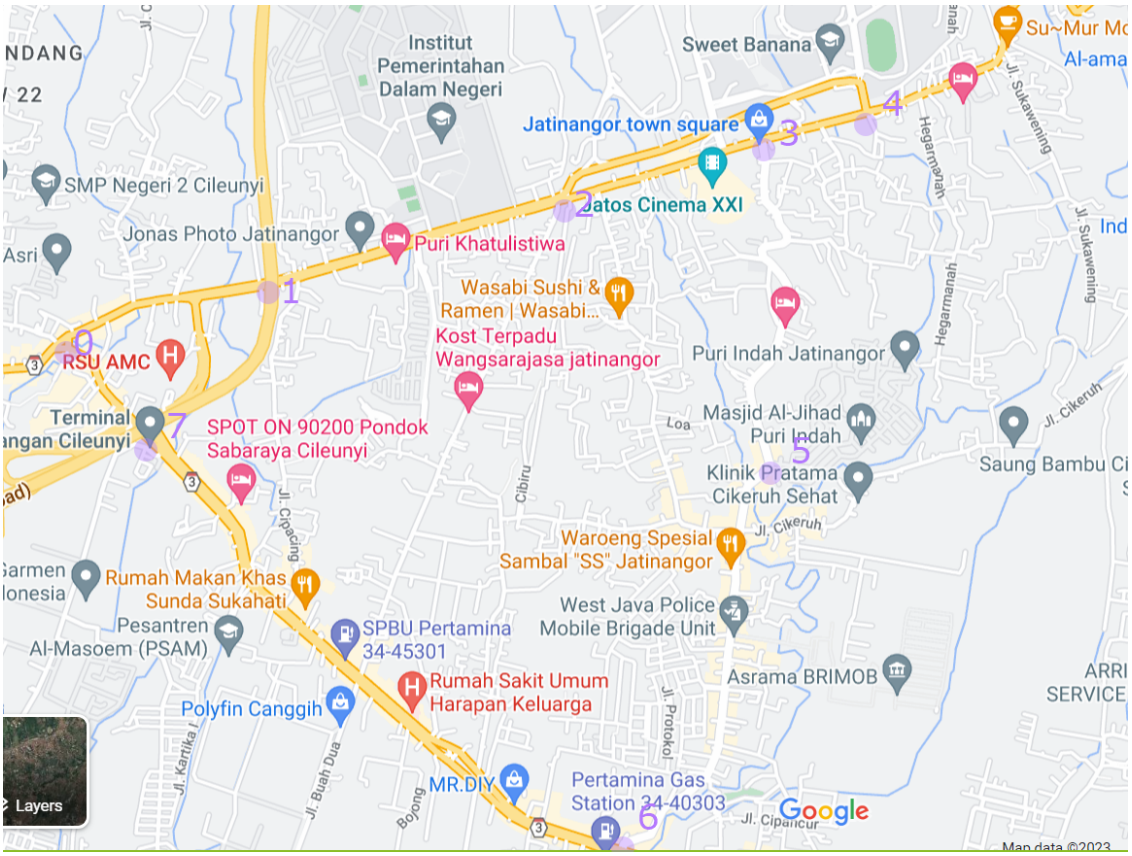
```
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 1
Please enter the filename: bandungatas
The map is:
['UNPAR', 'MCD UNPAR', 'Setia Budi Market', 'Grand Tjokro', 'Cihampelas', 'MCD Dago', 'Borma Dago', 'UNPAD DU', 'Baltos', 'Ciwalk', 'Boscha', 'Pasteru']
Please enter the start node: UNPAR
Please enter the goal node: Pasteru
Path: UNPAR -> Pasteru
Cost:  2410
Do you want to see the graph? (y/n): y
Showing graph...
```

## A*

```
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 2
Please enter the filename: bandungatas
The map is:
['UNPAR', 'MCD UNPAR', 'Setia Budi Market', 'Grand Tjokro', 'Cihampelas', 'MCD Dago', 'Borma Dago', 'UNPAD DU', 'Baltos', 'Ciwalk', 'Boscha', 'Pasteru']
Please enter the start node: UNPAR
Please enter the goal node: Pasteru
Path: UNPAR -> Pasteru
Cost:  2410
```

## B. Test Case Map Jatinangor

**Map**

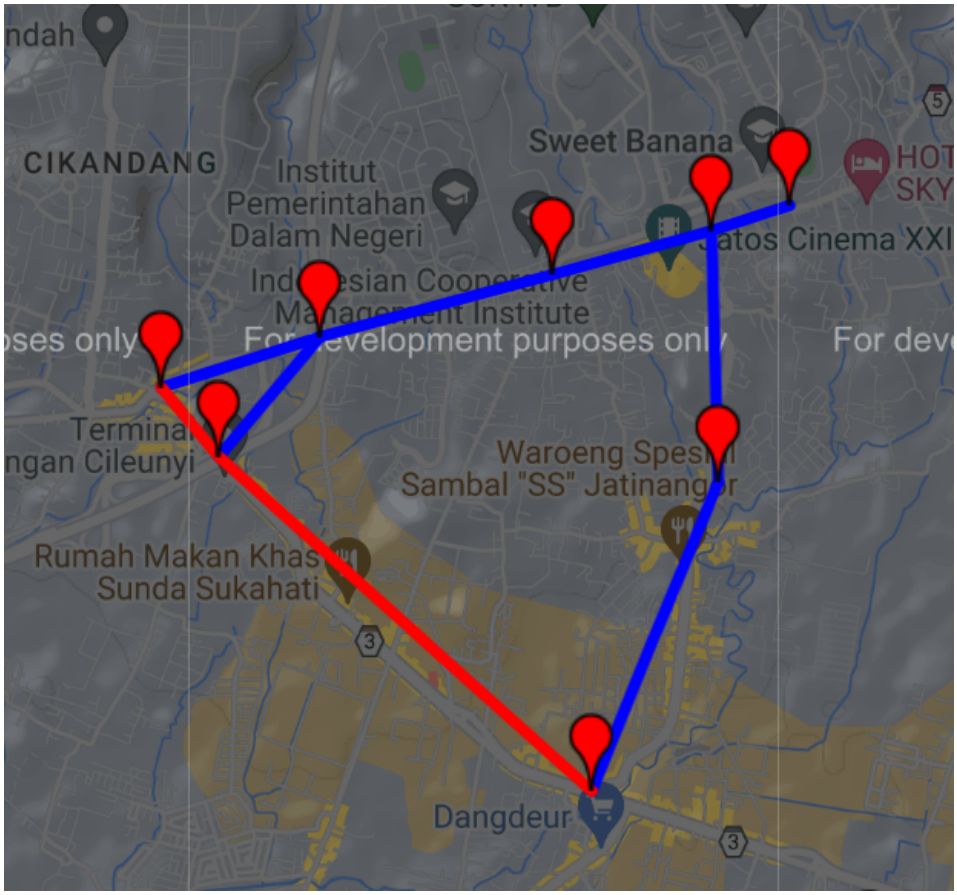



**.txt**

Cibiru, Selamat Datang Jatinangor, Jatos, Pertigaan Sayang, MCD, Komplek Puri, Pertamina Rancaekek, Terminal Cileunyi
0 680 0 0 0 0 0 440
680 0 990 0 0 0 0 690
0 990 0 676 0 0 0 0
0 0 676 0 357 980 0 0
0 0 0 357 0 0 0 0
0 0 0 980 0 0 1320 0
0 0 0 0 0 1320 0 2000
440 690 0 0 0 0 2000 0
-6.939113312357726,107.75284943505343
-6.9372388649458365,107.75883612522708

-6.934938396504598,107.76746210941278
-6.933319541611385,107.77338442702734
-6.932403608109727,107.77628121259522
-6.9426065808216375,107.77362046138575
-6.954002123926927,107.7689641466255
-6.941666793576368,107.75498745682222

**Map Hasil**




**UCS**

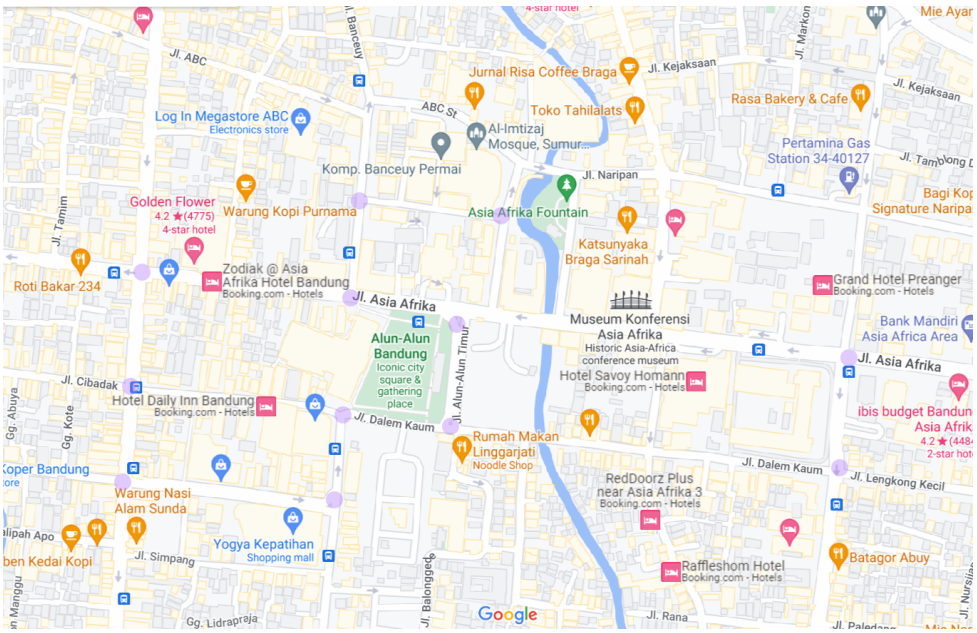
```
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 1
Please enter the filename: jatinewyork
The map is:
['Cibiru', 'Selamat Datang Jatinangor', 'Jatos', 'Pertigaan Sayang', 'MCD', 'Komplek Puri', 'Pertamina Rancaekek', 'Terminal Cileunyi']
Please enter the start node: Cibiru
Please enter the goal node: Pertamina Rancaekek
Path: Cibiru -> Terminal Cileunyi -> Pertamina Rancaekek
Cost:  2440
Do you want to see the graph? (y/n): y
Showing graph...
```

**A***

```
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 2
Please enter the filename: jatinewyork
The map is:
['Cibiru', 'Selamat Datang Jatinangor', 'Jatos', 'Pertigaan Sayang', 'MCD', 'Komplek Puri', 'Pertamina Rancaekek', 'Terminal Cileunyi']
Please enter the start node: Cibiru
Please enter the goal node: Pertamina Rancaekek
Path: Cibiru -> Terminal Cileunyi -> Pertamina Rancaekek
Cost:  2440
```

## C. Test Case Map Alun-alun

**Map**



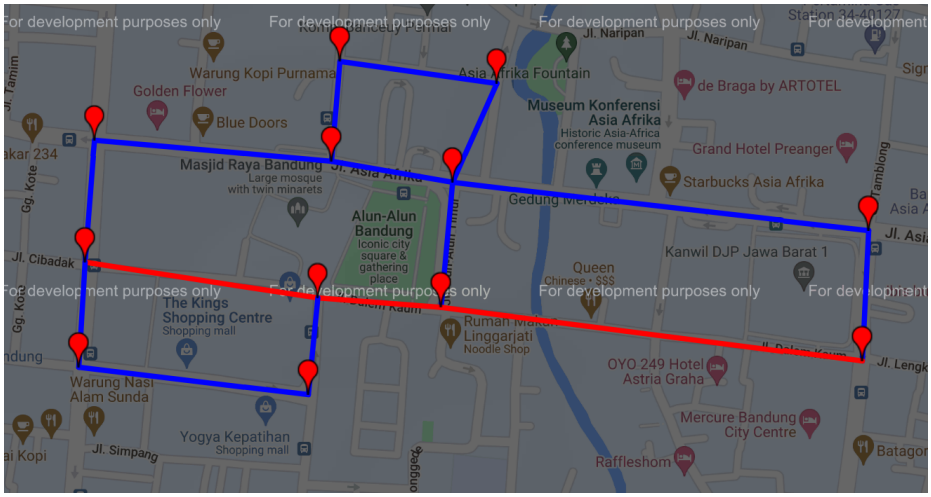

**.txt**

Roti Bakar 234, Masjid Raya Bandung, Pertigaan Asia Afrika, Museum KAA, Belakang Factory, Pertigaan Banceu, Dalem Kaum, Alun-alun, Belakang Masjid Raya, Cibadak, Warung Nasi Alam, Yogya Kepatihan

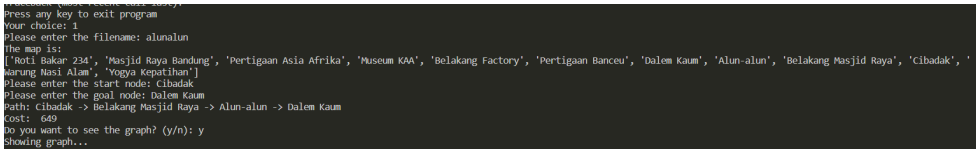
0 264 0 0 0 0 0 0 0 141 0 0
264 0 148 0 0 120 0 0 0 0 0 0
0 148 0 231 148 0 0 140 0 0 0 0
0 0 231 0 0 0 144 0 0 0 0 0
0 0 148 0 0 182 0 0 0 0 0 0
0 120 0 0 182 0 0 0 0 0 0 0

0 0 0 144 0 0 0 245 0 0 0 0
0 0 140 0 0 0 245 0 139 0 0 0
0 0 0 0 0 0 0 139 0 265 0 115
141 0 0 0 0 0 0 0 265 0 117 0
0 0 0 0 0 0 0 0 0 117 0 255
0 0 0 0 0 0 0 0 115 0 255 0
-6.920814530300131,107.60406512310648
-6.921027543484346,107.6064791111338
-6.9212405565724335,107.60771292723665
-6.921730486310206,107.61195081732905
-6.920239394222596,107.60816353833508
-6.920015730003211,107.6065649418192
-6.923051163939521,107.61188644431498
-6.922507982458791,107.6075949100442
-6.922412126838457,107.60633963627
-6.922050005430406,107.6039685635854
-6.923115067602032,107.60390419057131
-6.923391983372905,107.60624307674891

**Map Hasil**



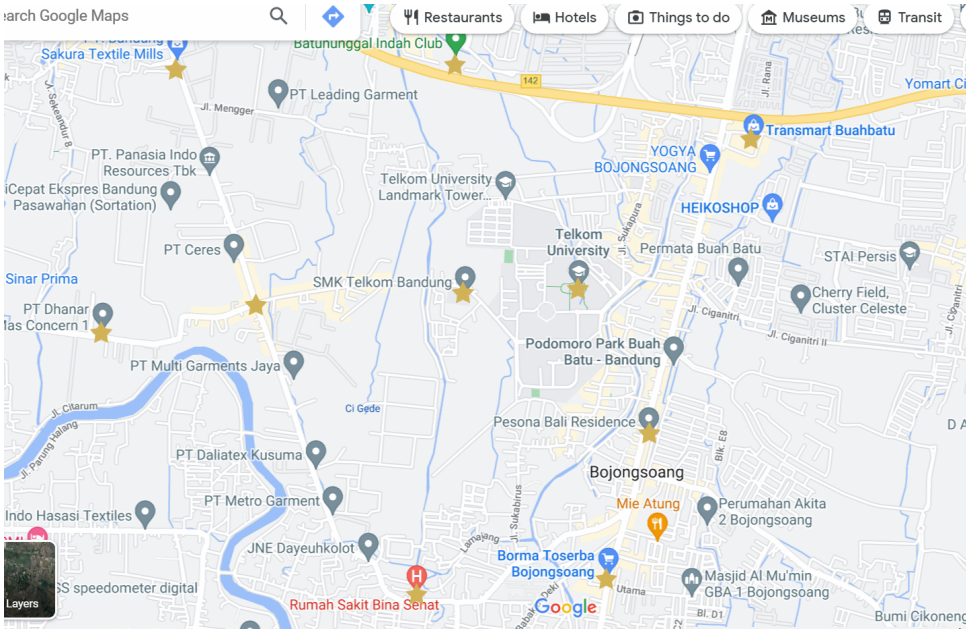

**UCS**




**A***

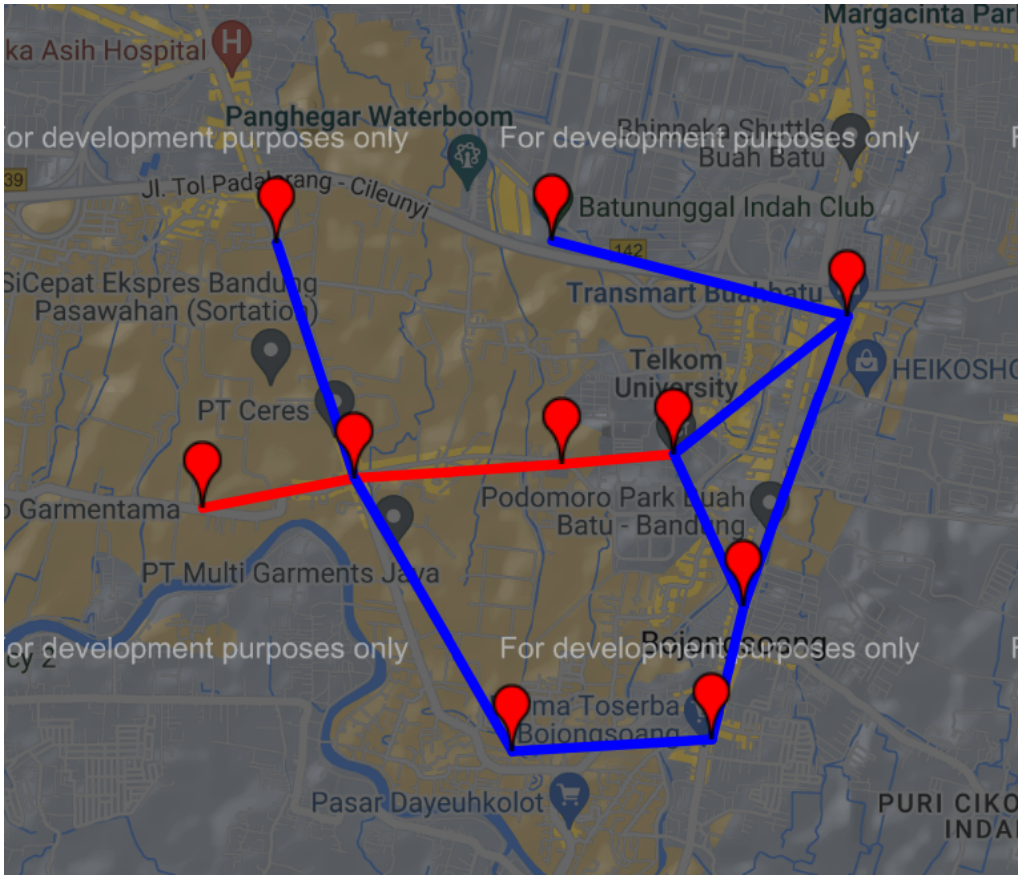
```
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 2
Please enter the filename: alunalun
The map is:
['Roti Bakar 234', 'Masjid Raya Bandung', 'Pertigaan Asia Afrika', 'Museum KAA', 'Belakang Factory', 'Pertigaan Banceu', 'Dalem Kaum', 'Alun-alun', 'Belakang Masjid Raya', 'Cibadak', 'Warung Nasi Alam', 'Yogya Kepatihan']
Please enter the start node: Cibadak
Please enter the goal node: Dalem Kaum
Path: Cibadak -> Belakang Masjid Raya -> Alun-alun -> Dalem Kaum
Cost:  649
```

## D. Test Case Map Buah Batu

| Map |
| --- |
|  |
| **.txt** |
| Dhanar Mas, Sakura Textile, Perempatan, Batununggal Indah, SMK Telkom, RS Bina Sehat, Borma Bojongsoang, Pesona Bali, TEL-U, Transmart<br>0 0 740 0 0 0 0 0 0<br>0 0 1200 0 0 0 0 0 0<br>740 1200 0 0 984 1510 0 0 0<br>0 0 0 0 0 0 0 0 1400<br>0 0 984 0 0 0 0 535 0<br>0 0 1510 0 0 0 900 0 0<br>0 0 0 0 0 900 0 670 0<br>0 0 0 0 0 0 670 0 765 1460<br>0 0 0 0 535 0 0 765 0 1000<br>0 0 0 1400 0 0 0 1460 1000 0 |

-6.974966142193127,107.61136450689669
-6.963635006348128,107.61454024215404
-6.973730806103597,107.61784472343533
-6.963592407576357,107.62638487635711
-6.973134435790849,107.62681402977024
-6.98535987536215,107.62466826270449
-6.984848713555631,107.63325133096757
-6.979140702105112,107.63462462188969
-6.97266585858392,107.63157763265626
-6.9667873047220406,107.63908781738647

**Map Hasil**




**UCS**

```
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 1
Please enter the filename: buahbatu
The map is:
['Dhanar Mas', 'Sakura Textile', 'Perempatan', 'Batununggal Indah', 'SMK Telkom', 'RS Bina Sehat', 'Borma Bojongsoang', 'Pesona Bali', 'TEL-U', 'Transmart']
Please enter the start node: Dhanar Mas
Please enter the goal node: TEL-U
Path: Dhanar Mas -> Perempatan -> SMK Telkom -> TEL-U
Cost:  2259
Do you want to see the graph? (y/n): y
Showing graph...
```

**A***

```
Do you want to see the graph? (y/n): n
Welcome to the Path Finder!
Please choose the algorithm you want to use:
1. Uniform Cost Search
2. A* Search
Press any key to exit program
Your choice: 2
Please enter the filename: buahbatu
The map is:
['Dhanar Mas', 'Sakura Textile', 'Perempatan', 'Batununggal Indah', 'SMK Telkom', 'RS Bina Sehat', 'Borma Bojongsoang', 'Pesona Bali', 'TEL-U', 'Transmart']
Please enter the start node: Dhanar Mas
Please enter the goal node: TEL-U
Path: Dhanar Mas -> Perempatan -> SMK Telkom -> TEL-U
Cost:  2259
```

# BAB IV

# PENUTUP

A. Kesimpulan

Algoritma UCS dan A* merupakan algoritma untuk mencari jarak terpendek dengan hasil optimum global. Berdasarkan hasil pengerjaan, disimpulkan bahwa algoritma A* memiliki efisiensi lebih besar dikarenakan UCS merupakan blind search yang tidak mengetahui jarak simpul awal dan simpul tujuan, sedangkan algoritma A* informed search yang memberikan data mengenai jarak simpul awal ke simpul yang ada pada peta dengan nilai *heuristic*.

B. Komentar dan Saran

Melalui tugas kecil ini, kami menjadi lebih mengetahui tentang algoritma A* dan UCS dalam kehidupan sehari-hari. Selain itu, pengetahuan kami mengenai penggunaan API juga bertambah. Menjadi tahu bahwa penggunaan API sangat penting dalam pengembangan tugas ini agar hasil akhir lebih akurat.

# LAMPIRAN

Pranala Github : https://github.com/eunicesarah/Tucil3_13521010_13521013.git

Checklist :

| 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 | ✔ | ✔ |