Laporan Tugas Kecil 3
IF2211 Strategi Algoritma
Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun Oleh :

Yanuar Sano Nur Rasyid	13521110
Muhammad Zaki Amanullah	13521146

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023

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.

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. 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

2. Kode Program

2.1 Struktur Data (Graph.py)

```python
import os
from haversine import haversine, Unit

class Graph:
    def __init__ (self):
        self.n : int = 0              # number of nodes
        self.nodes : list = []         # list of nodes

        # dictionary of coordinates (key = node, value = (lat, long))
        self.coords : dict = {}

        self.adj : list = []           # adjecency matrix
        self.weighted : list = []      # weighted adjecency matrix

    # read input from file (without coordinates)
    def read_input(self, file_name: str):···

    # read input from file (with coordinates)
    def read_input_coords(self, file_name: str): ···

    # calculate the weighted adjecency matrix
    def calculate_weighted(self): ···

    # getters
    def get_nodes(self) -> list:···
    def get_adj(self) -> list:···
    def get_coords(self) -> dict:···
    def get_weighted(self) -> list:···
```

Format input terdiri dari jumlah simpul kemudian diikuti dengan nama simpul serta koordinat simpul yang terdiri dari *latitude* dan *longitude* kemudian input diakhiri dengan matrks ketetanggaan dengan indeks input sesuai dengan urutan simpul yang dimasukkan. Perhitungan matriks berbobot pada algoritma bukan dihitung dari input yang terdapat dalam file melainkan dihitung dari koordinat tiap simpul yang terdapat dalam file.

2.2 Algoritma UCS (UCS.py)

```
import heapq
from Input import Graph

> def count_distance(path, weighted) -> float: ···

> def ucs(graph, start, goal) -> tuple: ···
```

```
def count_distance(path, weighted):
    distance = 0
    for i in range(len(path) - 1):
        distance += weighted[path[i]][path[i + 1]]
    return distance
```

```
def ucs(graph, start, goal) -> tuple:
    """
    Finds the shortest path from start to goal using Uniform Cost Search algorithm.

    :param graph: the weighted adjacency matrix of the graph
    :param start: the starting node
    :param goal: the goal node
    :return: the shortest path as a list of nodes
    """

    # Create a dictionary to store the parent node of each visited node
    parent = {}

    # Create a dictionary to store the cost of the path from the starting node to each visited node
    cost = {}

    # Initialize the cost of the starting node to 0
    cost[start] = 0
```

```
        # Add the starting node to the priority queue with cost 0
        heapq.heappush(pq, (0, start))

        # Loop until the priority queue is empty
        while pq:
            # Pop the node with the smallest cost from the priority queue
            node_cost, node = heapq.heappop(pq)

            # If the goal node is reached, return the shortest path
            if node == goal:
                path = [node]
                while node in parent:
                    node = parent[node]
                    path.append(node)
                return (path[::-1], count_distance(path[::-1], graph))
```

```
        # Loop through the neighbors of the current node
        for neighbor, weight in enumerate(graph[node]):
            # Ignore nodes that is not the neighbor or have already been visited with a lower cost
            if weight == 0 or neighbor in cost and cost[neighbor] <= node_cost + weight:
                continue

            # Update the cost and parent of the neighbor
            cost[neighbor] = node_cost + weight
            parent[neighbor] = node

            # Add the neighbor to the priority queue with its new cost
            heapq.heappush(pq, (cost[neighbor], neighbor))

    # If the goal node cannot be reached, return None
    return None
```

Terdapat dua buah fungsi di dalam modul UCS.py, yaitu count_distance() dan ucs(). Fungsi count_distance() menerima dua buah parameter, list of path dan weighted adjacency matrix. Fungsi ini akan mengembalikan float berupa jarak dari titik awal ke titik akhir.

Fungsi ucs menerima sebuah weighted adjacency matrix, node awal, dan node akhir. Langkah pertama yang dilakukan adalah menginisialisasi dictionary parent yang berisi parent dari setiap node agar dapat dilakukan backtracking saat path yang optimal sudah ditemukan. Selanjutnya inisialisasi dictionary cost dengan key berupa node dan value berupa jarak dari node tersebut ke node awal. Kita inisialisasikan pula cost[start] = 0. Kita juga perlu menginisialisasikan priority queue pq yang akan digunakan untuk mengetahui node mana yang perlu di telusuri selanjutnya.

Dilakukan iterasi sampai priority queue kosong dengan langkah-langkah setiap iterasi sebagai berikut. Langkah pertama yang dilakukan adalah pop node dengan cost yang paling rendah menggunakan method heappop dalam library heapq yang mengembalikan tuple node_cost, node. Dilakukan pengecekan apakah node tersebut merupakan node tujuan. Apabila node tersebut merupakan node tujuan, dilakukan langkah sebabai berikut. Kita akan menginisalisasikan variabel path yang akan berisi list of node dengan elemen pertama goal node. Setelah itu kita iterasi parent node dari goal node lalu parent node dari parent dari goal node dan seterusnya. Setelah itu dikembalikan list of path yang di balik urutannya serta costnya. Apabila node bukan merupakan goal node, akan dilakukan iterasi terhadap semua neighbor dari node tersebut. Apabila weight adalah 0 yang artinya kedua node tidak bertetangga iterasi akan dilanjutkan. Begitu pula apabila neighbor sudah ada di cost dan cost dari neighbor kurang dari sama dengan cost dari neighbor yang akan dikunjungi yang artinya neighbor sudah pernah dikunjungi dengan cost yang lebih baik. Apabila tidak memenuhi kedua kriteria tersebut, kita akan menambahkan key value pair dengan key adalah indeks node dan value adalah node_cost ditambah weight ke dalam dictionary cost. Kita juga akan menambahkan key value pair dengan key adalah indeks neighbor dengan value indeks node parent ke dalam dictionary

parent. Apabila iterasi sudah selesai tetapi goal belum ditemukan akan dikembalikan value none.

2.3 Algoritma A* (AStar.py)

```python
import heapq
import math
from Input import Graph

def distance(coord1, coord2) -> float: ...

def a_star(weighted_matrix, coords, start, goal) -> tuple: ...
```

```python
def distance(coord1, coord2) -> float:
    """
    Calculate the distance between two 2-dimensional coordinates

    :param coord1: coordinates for point 1
    :param coord2: coordinates for point 2
    :return: Euclidean distance between two coordinates
    """

    return haversine(coord1, coord2)
```

```python
def a_star(weighted_matrix, coords, start, goal) -> tuple:
    """
    Find the shortest path from start to goal using A* algorithm

    Parameters:
    weighted_matrix: 2D list that represents the weighted adjacency matrix of the graph
    coords: a dictionary mapping each node to their respective (x, y) coordinate
    start: the start node
    goal: the goal node

    Returns:
    (path, distance): tuple where path is the list of nodes in the shortest path
    """
    queued_node = [(0, start)]
    discovered = set()

    # Initialize the g_value (distance from start to current node) of each node with infinity (not explored yet)
    g_value = {node: float("inf") for node in range(len(weighted_matrix))}
    g_value[start] = 0

    # Initialize f_value (distance from start + straight line distance between current node and goal)
    f_value = {node: float("inf") for node in range(len(weighted_matrix))}
    f_value[start] = distance(coords[start], coords[goal])

    # Dictionary of the parents of each node
    parent = {}
```

```
    while queued_node:
        cur_node = heapq.heappop(queued_node)[1]

        # If goal found then backtrack through the parent dictionary and return the result
        if cur_node == goal:
            path = [cur_node]
            while cur_node in parent:
                cur_node = parent[cur_node]
                path.append(cur_node)
            # Return the path and the cost of the path (g_value)
            return (path[::-1], g_value[goal])

        discovered.add(cur_node)

        for neighbor in range(len(weighted_matrix)):
            if (weighted_matrix[cur_node][neighbor]) > 0 and neighbor not in discovered:
                test_g_value = g_value[cur_node] + weighted_matrix[cur_node][neighbor]

                # To prevent repetition of the same node, we only add this to the prio_queue if it has a better g_value than th
                if test_g_value < g_value[neighbor]:
                    parent[neighbor] = cur_node
                    g_value[neighbor] = test_g_value

                    # g_value is added to the straight line distance between the coordinates
                    f_value[neighbor] = g_value[neighbor] + distance(coords[neighbor], coords[goal])
                    heapq.heappush(queued_node, (f_value[neighbor], neighbor))

    return None
```

Terdapat dua buah fungsi di dalam modul AStar.py yaitu distance() dan a_star(). Fungsi distance() menerima dua buah koordinat bujur dan lintang dan mengembalikan jarak dari kedua koordinat tersebut dalam kilometer menggunakan haversine formula.

Fungsi a_star() menerima empat buah parameter yaitu weighted adjacency matrix, list koordinat, indeks awal, dan indeks akhir lalu mengembalikan tuple yang berisikan path dan jarak yang ditempuh. Langkah pertama yang dilakukan adalah inisialisasi struktur data. Priority queue diisikan dengan (0, start) yang merupakan sebuah tuple dengan nilai pertama adalah f_value dan nilai kedua adalah indeks node awal. Selanjutnya dibuat sebuah set of discovered nodes untuk mengurangi redundansi. Selanjutnya diinisialisasikan sebuah dictionary of g_values yang mirip dengan cost pada algoritma ucs merupakan jarak dari node awal ke node tersebut dimana semua value diisi dengan infinity kecuali untuk key node awal yang diisi dengan nilai nol. Setelah itu kita inisialisasi dictionary of f_values. f_value adalah g_value dijumlahkan dengan straight line distance menggunakan koordinat.

Setelah dilakukan inisialisasi, dilakukan iterasi sampai priority queue kosong dengan langkah sebagai berikut. Pertama kita pop node pertama di dalam prio queue. Apabila node tersebut adalah node akhir, maka dilakukan langkah sebagai berikut. Kita iinisialisasikan sebuah list of node yang akan berisi path yang dilalui. Kita akan mencari parent dari node akhir, parent dari parent dari node akhir dan seterusnya semberi menambahkannya ke dalam path. Setelah itu kita kembalikan tuple dengan nilai pertama berupa list of path dan nilai kedua berupa g_value dari node akhir. Apabila node yang di-pop dari queue bukan merupakan node akhir maka dilakukan langkah-langkah sebagai berikut. Pertama kita tambahkan node tersebut ke dalam set of discovered nodes. Selanjutnya akan dilakukan iterasi dari setiap node yang adjacent dengan node tersebut dan node yang belum terdapat dalam set of discovered node. Setiap iterasi akan dilakukan

langkah sebagai berikut. Pertama kita hitung g_value dari tetangga dengan menambahkan g_value dari cur_node dengan weight dari node tetangga dan memasukkannya ke dalam dictionary of g_values. Untuk menghindari repetisi kita hanya akan menambahkan sebuah node jika node tersebut memiliki g_value atau cost yang lebih rendah dari yang sudah ada. Apabila lebih rendah kita juga perlu menghitung f_value dari node tersebut dengan menambahkannya dengan straight_line_path yang dihitung dari koordinat.

Semua *source code* dapat diakses melalui github dengan link yang terdapat pada lampiran.

3. Uji Kasus

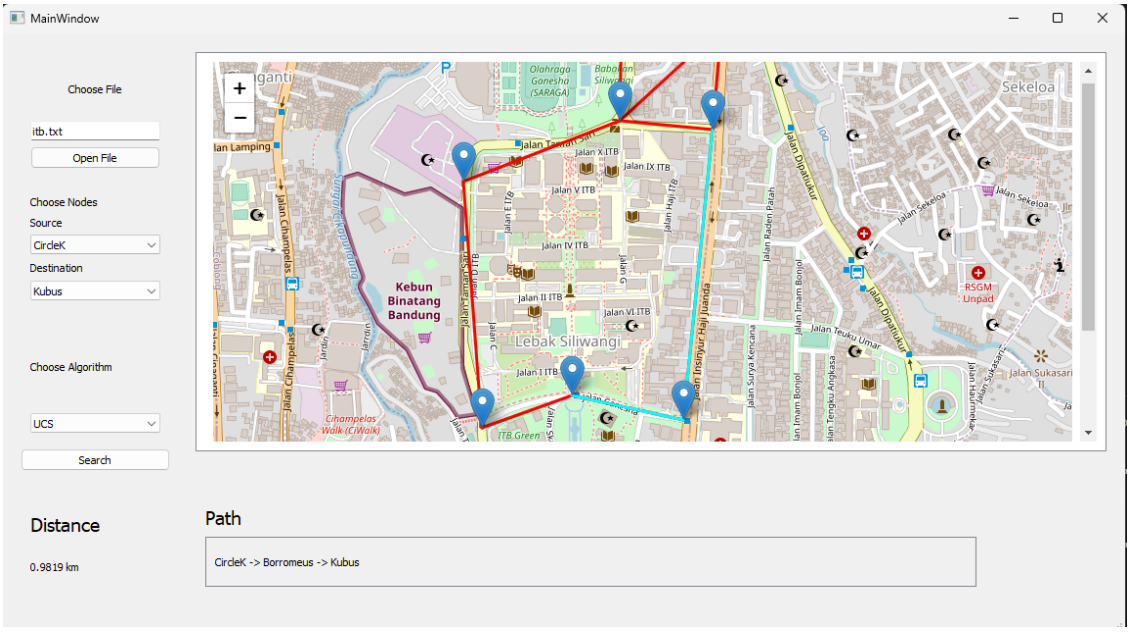
1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara

Input

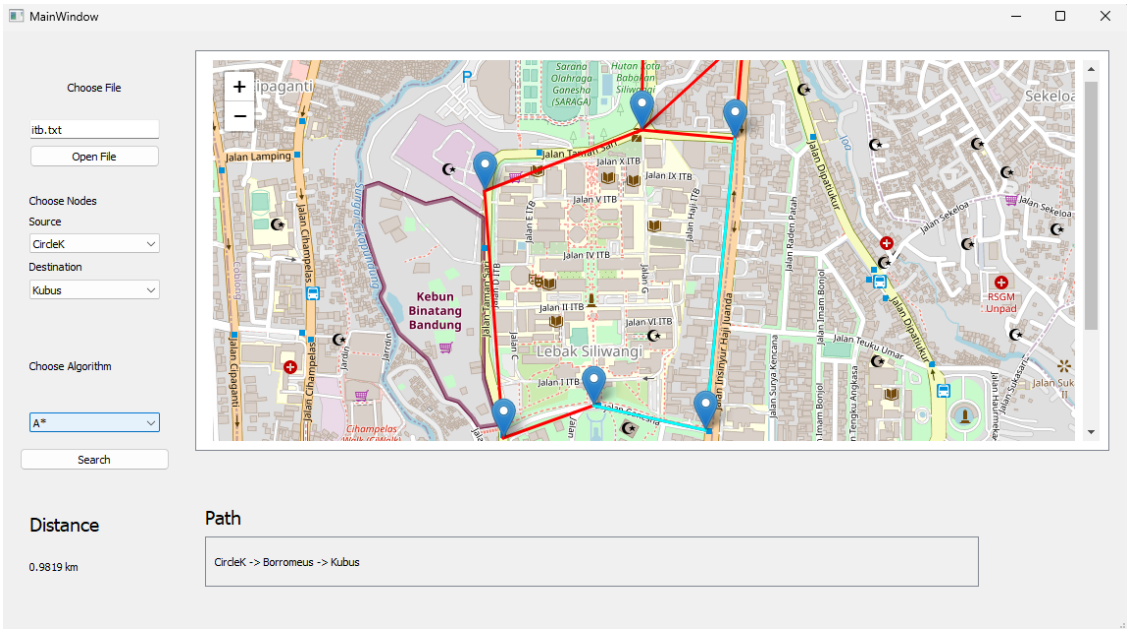
```
itb.txt
8
SegitigaDayangSumbi -6.887221 107.611479
Batan -6.888555 107.608039
Bonbin -6.893890 107.608442
Kubus -6.893188 107.610418
Borromeus -6.893731 107.612864
CircleK -6.887422 107.613518
PosPolisi -6.885217 107.613692
Baksil -6.884972 107.611512
0 1 0 0 0 1 1 1
1 0 1 0 0 0 0 0
0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 1 0 0
1 0 0 0 1 0 1 0
1 0 0 0 0 1 0 1
1 0 0 0 0 0 1 0
```

Output

UCS



A*



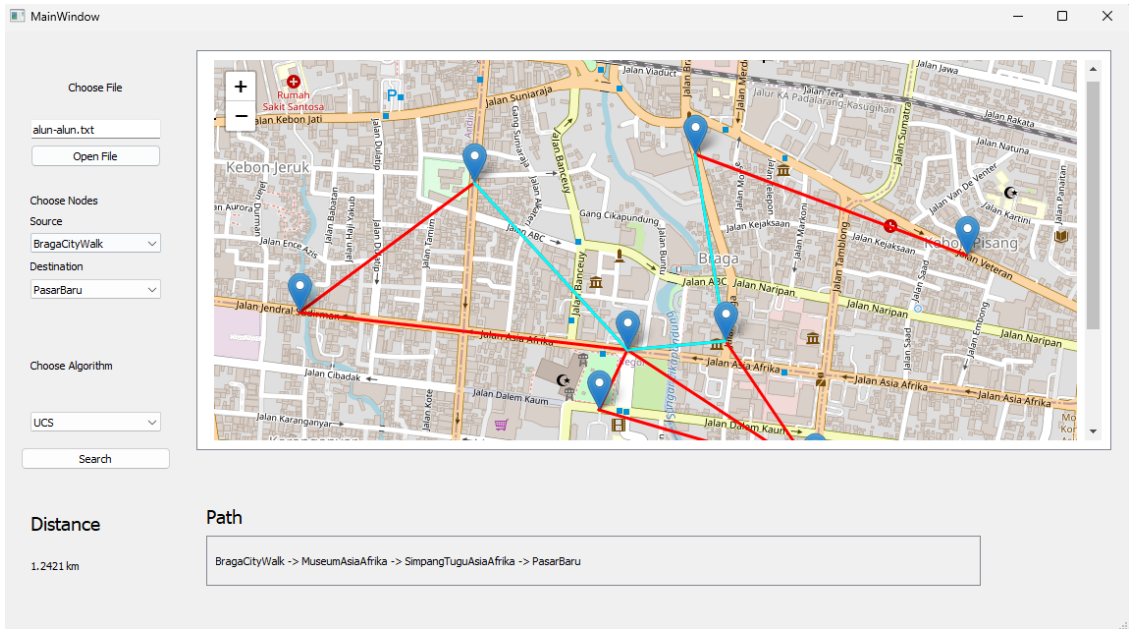
2. Peta jalan sekitar Alun-alun Bandung

Input

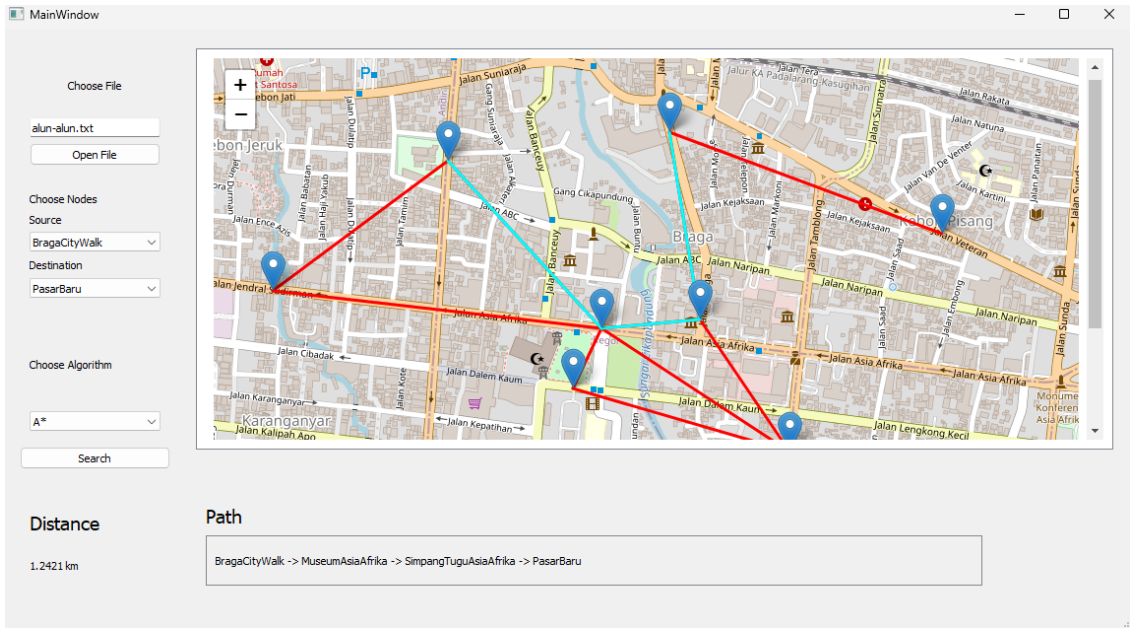
```
alun-alun.txt
8
SimpangTuguAsiaAfrika -6.921210 107.607689
MuseumAsiaAfrika -6.921023 107.609848
BragaCityWalk -6.916968 107.609178
MercureCityCentre -6.923878 107.611800
SudirmanStreetMarket -6.920395 107.600530
PendopoKota -6.922503 107.607066
PasarBaru -6.917585 107.604353
BatagorKingsley -6.919173 107.615111
0 1 0 1 1 1 1 0
1 0 1 1 0 0 0 0
0 1 0 0 0 0 0 1
1 1 0 0 0 1 0 0
1 0 0 0 0 0 1 0
1 0 0 1 0 0 0 0
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
```

Output

UCS


MainWindow
Choose File
alun-alun.txt
Open File
Choose Nodes
Source
BragaCityWalk
Destination
PasarBaru
Choose Algorithm
UCS
Search
Distance
1.2421 km
Path
BragaCityWalk -> MuseumAsiaAfrika -> SimpangTuguAsiaAfrika -> PasarBaru

A*


MainWindow
Choose File
alun-alun.txt
Open File
Choose Nodes
Source
BragaCityWalk
Destination
PasarBaru
Choose Algorithm
A*
Search
Distance
1.2421 km
Path
BragaCityWalk -> MuseumAsiaAfrika -> SimpangTuguAsiaAfrika -> PasarBaru

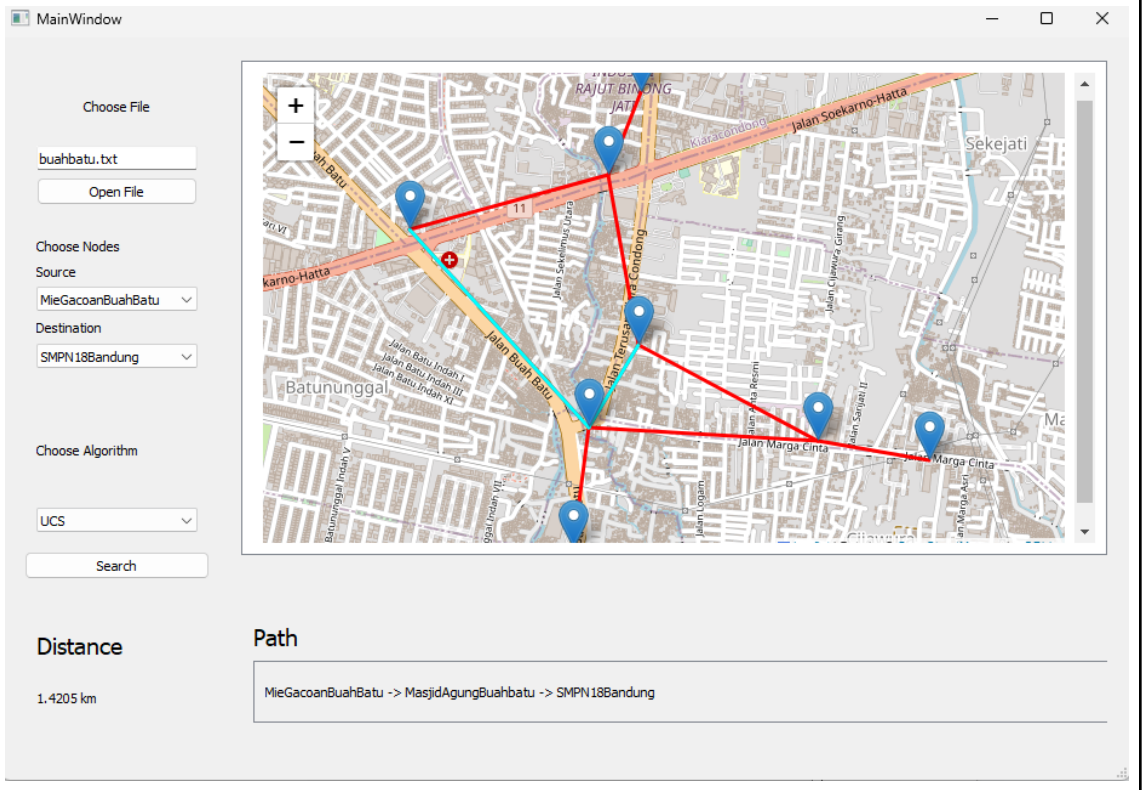
3. Peta jalan sekitar Buahbatu atau Bandung Selatan

Input

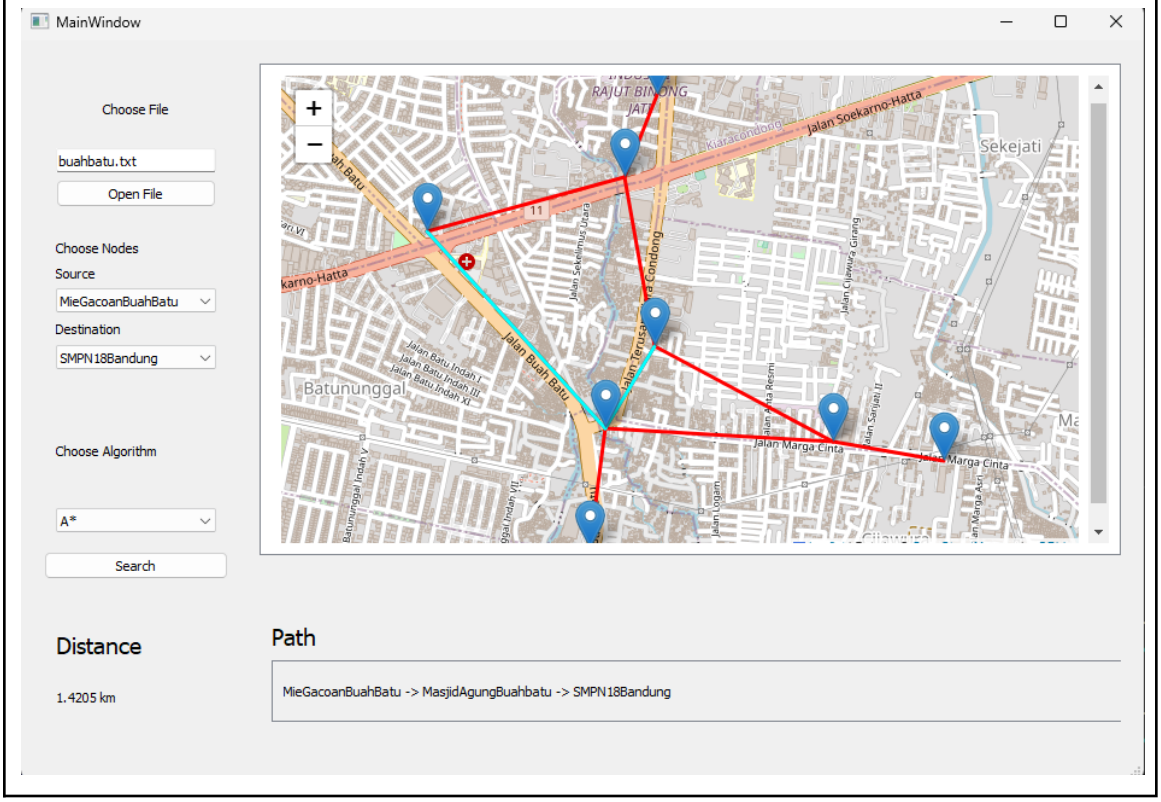
```
8
MargacintaPark -6.954881684805445 107.64794265604833
MasjidAgungBuahbatu -6.954417993825552 107.63985853174019
SMPN18Bandung -6.95150819365261 107.64157733547279
RSJPParamarta -6.945567418279151 107.64052157403196
BormaToserbaKiaracondong -6.942664038999554
107.64170393602062
StasiunBandung -6.958691117931306 107.63928778271176
BormaToserbaMargacinta -6.955572735377776
107.65189667096475
MieGacoanBuahBatu -6.947481747396867 107.63346445436977
0 1 1 0 0 0 1 0
1 0 1 0 0 1 0 1
1 1 0 1 0 0 0 0
0 0 1 0 1 0 0 1
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0
```

Output

UCS


MainWindow
Choose File
buahbatu.txt
Open File
Choose Nodes
Source
MieGacoanBuahBatu
Destination
SMPN18Bandung
Choose Algorithm
UCS
Search
Distance
1.4205 km
Path
MieGacoanBuahBatu -> MasjidAgungBuahbatu -> SMPN18Bandung

A*


MainWindow
Choose File
buahbatu.txt
Open File
Choose Nodes
Source
MieGacoanBuahBatu
Destination
SMPN18Bandung
Choose Algorithm
A*
Search
Distance
1.4205 km
Path
MieGacoanBuahBatu -> MasjidAgungBuahbatu -> SMPN18Bandung

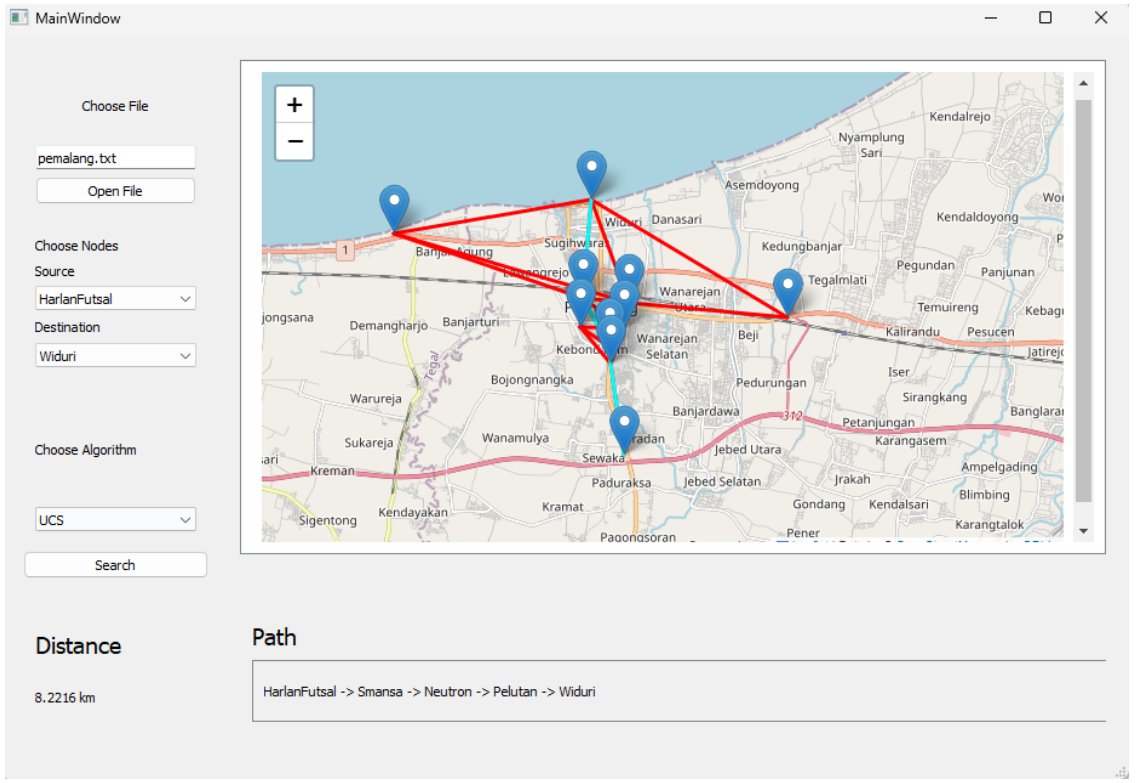
4. Peta jalan sebuah kawasan di kota asalmu

Input

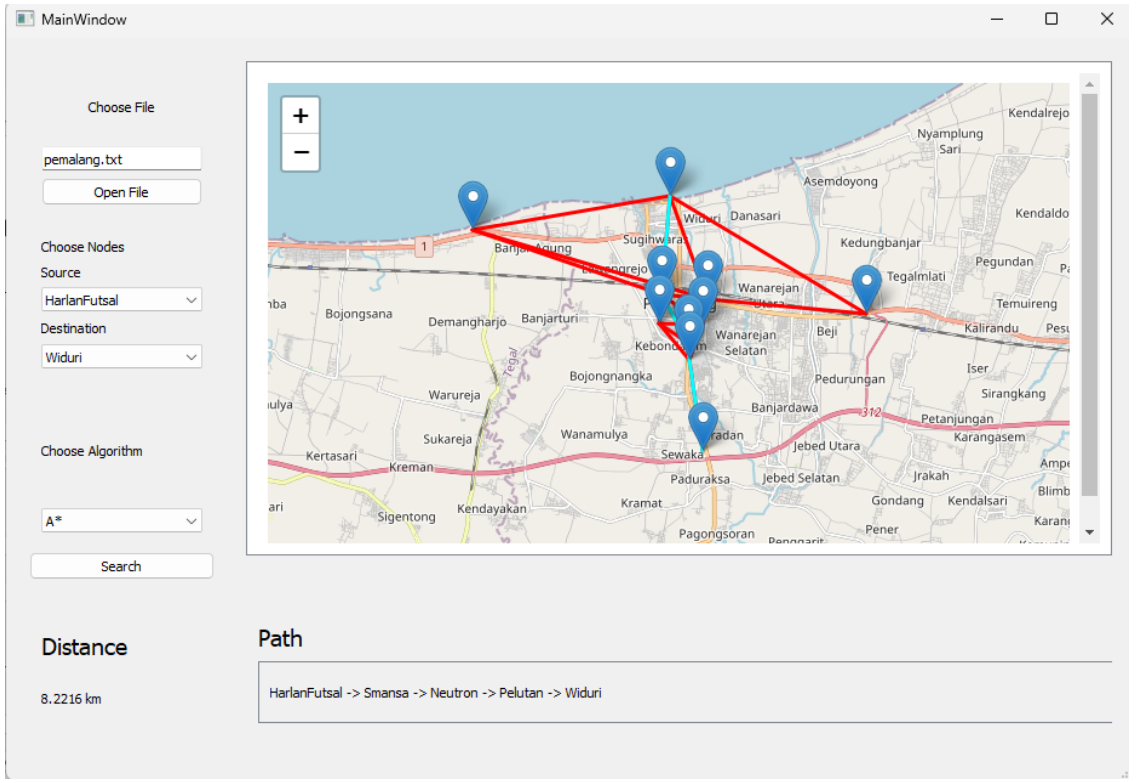
```
pemalang.txt
10
Sampurna -6.869955790040921 109.32173473650464
Smansa -6.906354688530867 109.38340089813235
Yogya -6.889210449525539 109.38830477696588
Snida -6.896528510066689 109.38707612372004
HarlanFutsal -6.931965579863931 109.3871220651295
Simed -6.893276828523623 109.43352171553332
Widuri -6.860359334704655 109.37788990547827
Sdl -6.89626159780668 109.37462444440897
Pelutan -6.887925415320945 109.37547561011165
Neutron -6.901555815768582 109.38308401571649
0 0 1 0 0 0 1 0 1 0
0 0 1 1 1 0 0 1 0 1
1 1 0 1 0 1 1 0 1 0
0 1 1 0 0 0 0 1 1 1
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0
1 0 1 0 0 1 0 0 1 0
0 1 0 1 0 0 0 0 1 1
1 0 1 1 0 0 1 1 0 1
0 1 0 1 0 0 0 1 1 0
```

Output

UCS


MainWindow
Choose File
pemalang.txt
Open File
Choose Nodes
Source
HarlanFutsal
Destination
Widuri
Choose Algorithm
UCS
Search
Distance
8.2216 km
Path
HarlanFutsal -> Smansa -> Neutron -> Pelutan -> Widuri

A*


MainWindow
Choose File
pemalang.txt
Open File
Choose Nodes
Source
HarlanFutsal
Destination
Widuri
Choose Algorithm
A*
Search
Distance
8.2216 km
Path
HarlanFutsal -> Smansa -> Neutron -> Pelutan -> Widuri

4. Kesimpulan

1. Kesimpulan

   Penentuan lintasan atau *route planning* adalah proses pencarian jalur terpendek dari suatu titik ke titik tujuan. Penentuan lintasan ini dapat diselesaikan dengan algoritma *route planning* seperti UCS (Uniform Cost Search) atau A*.

   Algoritma UCS atau Uniform Cost Search adalah algoritma pencarian rute dengan cara menghitung akumulasi *cost* dari rute yang sudah dilalui dan memilih rute selanjutnya dengan *cost* terendah, sedangkan Algoritma A* memperhitungkan biaya kumulatif ditambah dengan perhitungan *heuristic* yang memungkinkan algoritma A* akan bekerja lebih cepat dibandingkan dengan algoritma UCS. Algoritma *heuristic* yang digunakan adalah dengan mencari *straight line path* dari dua buah koordinat yang tentunya tidak akan meng overestimasi jarak sebenarnya. Sehingga dapat disimpulkan bahwa kedua buah algoritma pasti menghasilkan hasil yang optimal.

2. Saran
   1. Rumus heuristik dapat diperbaiki sehingga hasil menjadi lebih optimal
   2. Spesifikasi dapat diperjelas lagi
   3. Pemilihan *library* dapat dioptimalkan sehingga dapat menggunakannya sesuai dengan tujuan

5. Daftar Pustaka

Munir, Rinaldi. 2023. IF2211 Strategi Algoritm, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf, diakses 8 April 2023

Story, Rob. Folium 0.14.0 documentation, https://python-visualization.github.io/folium/, diakses 11 April 2023

6. Lampiran

Github : https://github.com/zakia215/Tucil3_13521110_13521146

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		✓