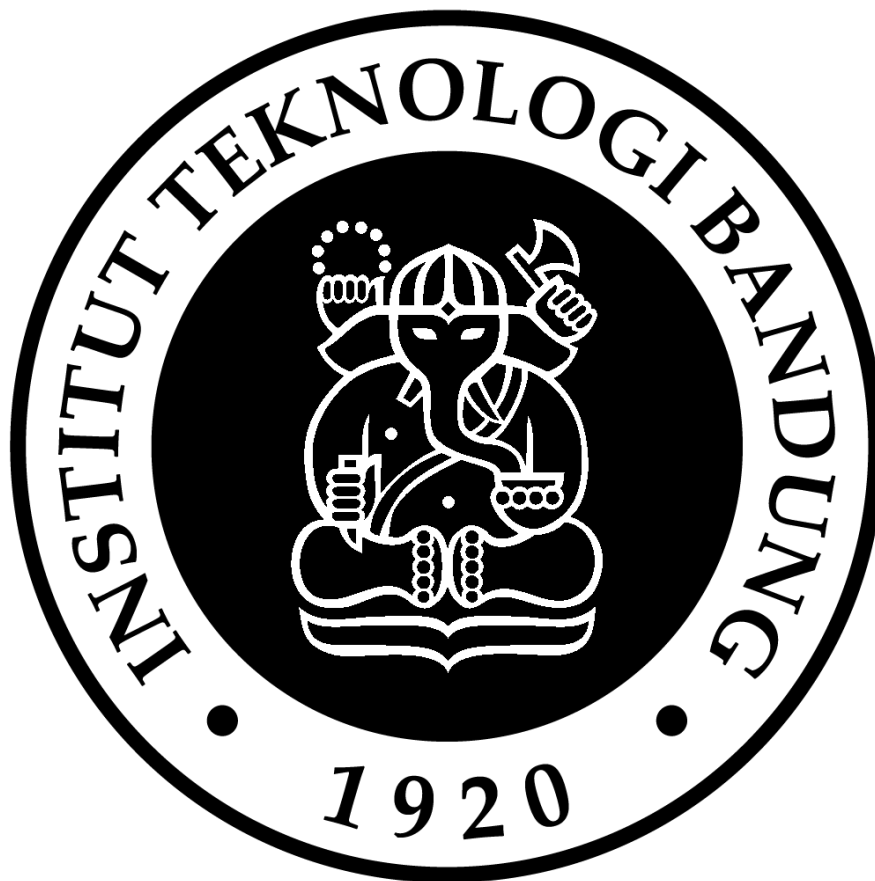


LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA SEMESTER II
2022-2023

PENENTUAN LINTASAN TERPENDEK DENGAN
ALGORTIMA UCS DAN A*

Disusun oleh :

1. Fahrian afdholi 13521031
2. Tobias Natalio Sianipar 13521090



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG 2022

1. DESKRIPSI MASALAH

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.

2. METODE PENYELESAIAN

2.1 Penjelasan Algoritma A* dan UCS

Algoritma UCS (Uniform Cost Search) adalah salah satu algoritma pencarian jalur terpendek yang digunakan dalam kecerdasan buatan untuk mencari jalur terpendek antara dua titik dalam graf atau peta berbobot. Algoritma UCS bekerja dengan cara mengeksplorasi node atau simpul yang memiliki biaya terendah terlebih dahulu dan kemudian memperluas pencarian ke simpul tetangga dengan biaya yang lebih rendah atau sama dengan biaya saat ini. Algoritma ini terus memperluas pencarian hingga mencapai simpul tujuan atau menemukan jalur terpendek dari simpul awal ke simpul tujuan. Keuntungan dari algoritma UCS adalah efektivitasnya dalam menemukan jalur terpendek dengan biaya minimum, namun kekurangannya adalah kompleksitas waktu yang tinggi karena perlu memeriksa setiap simpul pada setiap tingkat kedalaman dalam graf.

Algoritma A* (A-star) adalah salah satu algoritma pencarian jalur terpendek yang menggunakan strategi pencarian heuristik untuk mencari jalur terpendek antara dua titik dalam graf atau peta berbobot. Algoritma A* menggabungkan informasi biaya dan jarak dari simpul awal ke simpul tujuan dalam upaya untuk memperkirakan biaya total yang diperlukan untuk mencapai simpul tujuan. Algoritma ini menggabungkan biaya aktual dari simpul awal ke simpul saat ini (biaya g) dengan perkiraan biaya tersisa dari simpul saat ini ke simpul tujuan (biaya h) untuk memberikan nilai fungsi $f = g + h$ yang digunakan untuk memilih simpul yang akan diperluas selanjutnya. Algoritma A* memiliki keuntungan dari efisiensi dan akurasi dalam menemukan jalur terpendek, dan sering digunakan dalam berbagai aplikasi seperti navigasi, game, dan robotika. Namun, pemilihan fungsi heuristik yang tepat dapat mempengaruhi kinerja algoritma A*.

Pada algoritma UCS, simpul dievaluasi berdasarkan biaya aktual dari simpul awal ke simpul saat ini. Algoritma ini bekerja dengan mengeksplorasi simpul dengan biaya terendah terlebih dahulu, tanpa memperhatikan jarak dari simpul tersebut ke simpul tujuan. Algoritma UCS efektif dalam menemukan jalur terpendek dengan biaya minimum, tetapi memiliki kompleksitas waktu yang tinggi karena perlu memeriksa setiap simpul pada setiap tingkat kedalaman dalam graf.

Sementara itu, algoritma A* menggabungkan informasi biaya dan jarak dari simpul awal ke simpul tujuan dalam upaya untuk memperkirakan biaya total yang diperlukan untuk mencapai simpul tujuan. Algoritma A* menghitung fungsi $f = g + h$, di mana g adalah biaya aktual dari simpul awal ke simpul saat ini, dan h adalah perkiraan biaya tersisa dari simpul saat ini ke simpul tujuan. Algoritma A* bekerja dengan mengeksplorasi simpul dengan nilai f terkecil terlebih dahulu.

Keuntungan dari algoritma A* adalah efisiensi dan akurasi dalam menemukan jalur terpendek, terutama jika fungsi heuristik yang tepat digunakan. Namun, algoritma ini dapat menjadi tidak efektif jika heuristik tidak akurat atau jika graf memiliki banyak simpul dengan nilai f yang sama. Sedangkan, keuntungan dari algoritma UCS adalah efektivitasnya dalam menemukan jalur terpendek dengan biaya minimum, tetapi kekurangannya adalah kompleksitas waktu yang tinggi.

2.2 Penerapan Algoritma A* dan UCS

Dalam penerapan algoritma A* pada pencarian jarak terdekat, kami membuat sebuah fungsi A* yang memiliki parameter matriks integer yang berisi simpul ketetanggaan yang memiliki bobot. pada fungsi tersebut akan memiliki return jalur terpendek yang memiliki tipe array satu dimensi berisi nama simpul. dalam fungsi tersebut akan mengolah matriks ketetanggaan dengan cara mencari tetangga dari si simpul awal lalu menyimpan jumlah jalan yang sudah ditempuh dan menambahkannya pada jarak yang ditempuh, lalu menghitung jarak yang sedang ditempuh dengan jarak yang ingin dituju. setelah itu bandingkan bobot simpul mana yang paling minimum yang akan dipilih selanjutnya sampai tujuan akhir. UCS hampir sama seperti A* hanya saja hanya menghitung jumlah jejak yang pernah ditempuh saja dan tidak menjumlahkannya dengan jarak yang ingin dituju.

3. SOURCE CODE DALAM BAHASA PYTHON

3.1 Astar.py

```
from queue import PriorityQueue

# masukkan arr sebagai matriks ketetanggaan origin sebagai index awal dan destination sebagai index
tujuan
def a_star(arr, origin, destination):
    rute = [origin]

    if arr[origin][destination] > 0:
        rute.append(destination)
        return rute
    else:
        temp_arr = [origin]
        prioqueue = PriorityQueue()
        prioqueue.put((abs(arr[origin][destination]), 0, temp_arr))
        count = 1
        temp = prioqueue.get()

        while arr[temp[2][len(temp[2]) - 1]][destination] != 0:

            for i in range(len(arr[temp[2][len(temp[2]) - 1]])):
                if arr[temp[2][len(temp[2]) - 1]][i] > 0:
                    temp_push = temp[2].copy()
                    temp_push.append(i)

                    temp_value = 0

                    for j in range(len(temp_push) - 1):
                        temp_value += arr[temp_push[j]][temp_push[j + 1]]

                    prioqueue.put((abs(arr[i][destination]) + temp_value, count, temp_push.copy()))
                    count += 1
                    temp_push.clear()
            if prioqueue.empty():
                break
            temp = prioqueue.get()

        rute = temp[2]
        if prioqueue.empty() and rute[len(rute)-1] != destination:
            return []
        return rute
```

3.2 CSU

```
from queue import PriorityQueue

# masukkan arr sebagai matriks ketetanggaan origin sebagai index awal dan destination sebagai index tujuan
def csu(arr, origin, destination):
    rute = [origin]

    if arr[origin][destination] > 0:
        rute.append(destination)
        return rute
    else:
        temp_arr = [origin]
        prioqueue = PriorityQueue()
        prioqueue.put((0, 0, temp_arr))
        count = 1
        temp = prioqueue.get()

        while arr[temp[2]][len(temp[2]) - 1][destination] != 0:

            for i in range(len(arr[temp[2]][len(temp[2]) - 1])):
                if arr[temp[2]][len(temp[2]) - 1][i] > 0:
                    temp_push = temp[2].copy()
                    temp_push.append(i)

                    temp_value = 0

                    for j in range(len(temp_push) - 1):
                        temp_value += arr[temp_push[j]][temp_push[j+1]]

                    prioqueue.put((temp_value, count, temp_push.copy()))
                    count += 1
                    temp_push.clear()

            if prioqueue.empty():
                break
            temp = prioqueue.get()

        rute = temp[2]
        if prioqueue.empty() and rute[len(rute) - 1] != destination:
            return []
        return rute
```

3.3 FileReader.py

```
import os

#file reader membaca file dari test
def file_reader(test):
    split_new_line = test.split("\n")

    matrix = []
    coordinate = []
    index = int(test[0])+1

    for i in range(1, index):
        coordinate.append([float(split_new_line[i][0]), float(split_new_line[i][2])])

    for i in range(index, 2*index-1):
        matrix.append(split_new_line[i].split(" "))

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            matrix[i][j] = float(matrix[i][j])

    return coordinate, matrix
```

3.4 app.py

```
from flask import Flask, render_template, request, jsonify
from algorithm.CSU import csu
from algorithm.Astar import a_star
from algorithm.FileReader import file_reader
import time

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/text-format")
def text_format():
    return render_template("text-format.html")

@app.route('/calculate', methods=['POST'])
def calculate():
    matrix = request.json['adjacencyMatrix']
    origin = int(request.json['origin'])
    destination = int(request.json['destination'])
    ucs = bool(request.json['ucs'])

    index_array = []
    # process the matrix data here
    start_time = time.time()
    if(ucs) :
        index_array = csu(matrix, origin, destination)
    else :
        index_array = a_star(matrix, origin, destination)
    end_time = time.time()

    time_in_ms = (end_time - start_time) * 1000

    # return the index array as a JSON response
    return jsonify({'index_array': index_array, 'runtime': time_in_ms})

@app.route('/process_file', methods=['POST'])
def process_file():
    content = request.get_json()['content']
    # Call your Python function with the file content
    result, result2 = file_reader(content)
    return jsonify({'result': result, 'result2': result2})

if __name__ == '__main__':
    app.run(debug=True)
```

4. HASIL EKSEKUSI PROGRAM

4.1 Hasil Eksekusi File Input bandung_utara.txt Pada Algoritma A*

Clear MapClear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

SearchGo

MapSatellite

Origin:3Destination:4

☐ UCS ☒ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1165 m.

4.2 Hasil Eksekusi File Input bandung_utara.txt Pada Algoritma UCS

Clear MapClear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

SearchGo

MapSatellite

Origin:3Destination:4

☒ UCS ☐ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1165 m.

4.3 Hasil Eksekusi File Input alun-alun_bandung.txt Pada Algoritma A*

Clear Map

Clear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

Search

Go

Map

Satellite

Origin: 7 Destination: 1

☐ UCS

☒ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1105 m.

4.4 Hasil Eksekusi File Input alun-alun_bandung.txt Pada Algoritma UCS

Clear Map

Clear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

Search

Go

Map

Satellite

Origin: 7 Destination: 1

☒ UCS

☐ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1105 m.

4.5 Hasil Eksekusi File Input buah_batu.txt Pada Algoritma A*

Clear MapClear Route

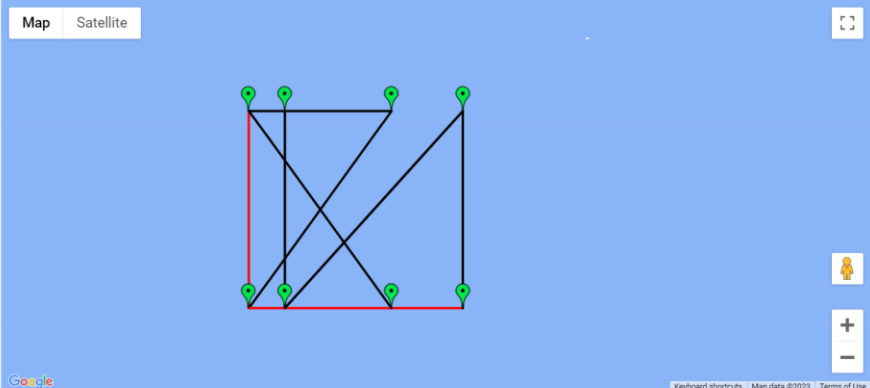
User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

SearchGo

MapSatellite



Origin:1Destination:4

☐ UCS☒ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1750 m.

4.6 Hasil Eksekusi File Input buah_batu.txt Pada Algoritma UCS

Clear MapClear Route

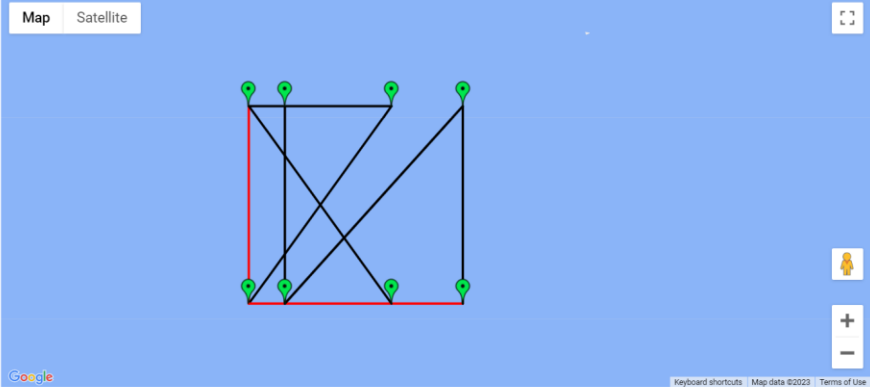
User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

Drag and drop a text file here

SearchGo

MapSatellite



Origin:1Destination:4

☒ UCS☐ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.
shortest distance traveled : 1750 m.

4.7 Hasil Eksekusi File Input kabupaten_kuningan.txt Pada Algoritma A*

Clear Map

Clear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

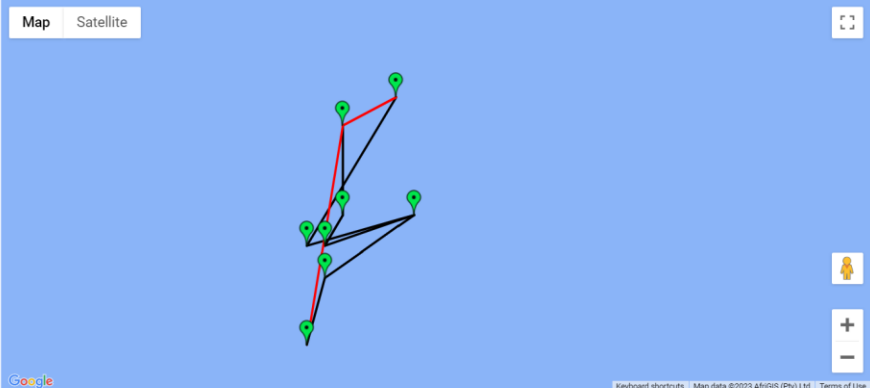
Drag and drop a text file here

Search

Go

Map

Satellite



Origin: 3 Destination: 7

☐ UCS ☒ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.

shortest distance traveled : 618 m.

4.7 Hasil Eksekusi File Input kabupaten_kuningan.txt Pada Algoritma UCS

Clear Map

Clear Route

User Guide:

- Ketik lokasi pada searchbar diatas map, lalu tekan 'Go'
- Doubleclick pada ujung jalan atau persimpangan jalan untuk menambah simpul
- Klik kanan pada simpul untuk menghapus simpul tersebut
- Klik 2 simpul untuk menambah garis ketetanggaan
- klik 2 simpul yang sudah memiliki garis untuk menghapus garis ketetanggaan
- Hover pada simpul untuk melihat ID simpul dan masukkan pada origin dan destination untuk memulai pencarian
- Pilih UCS atau A* sebagai algoritma pencarian
- tekan 'Find Path' untuk mencari jalan terdekat
- jika ingin melakukan pencarian terhadap data yang telah ada, anda dapat memasukkan file.txt dengan format pada tautan berikut [format.txt](#)
- pastikan untuk mencentang checkbox 'File input calculation' bila menggunakan file

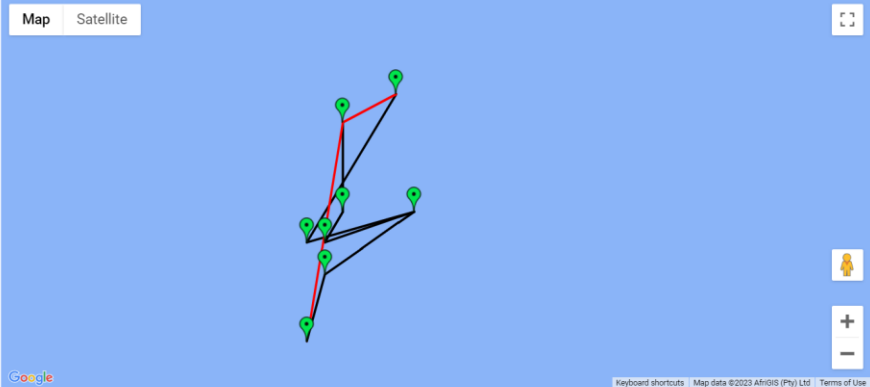
Drag and drop a text file here

Search

Go

Map

Satellite



Origin: 3 Destination: 7

☒ UCS ☐ A*

☐ File input calculation

Find Path

calculation time : 0.00000 ms.

shortest distance traveled : 618 m.

5. ALAMAT GITHUB

https://github.com/fchrgrib/Tucil3_13521031_13521090

6. KESIMPULAN DAN KOMENTAR

6.1 Kesimpulan

Dari pembuatan tugas kecil 3 strategi algoritma kami menyimpulkan jika algoritma A* dan UCS dapat mencari jarak terdekat dari titik awal dan titik akhir yang dipilih pada sebuah graf.

Jika graf memiliki jalur yang kompleks, algoritma UCS akan lebih lama dalam mengeksekusi program dibandingkan dengan algoritma A*.

6.2 Komentar

Kami harap asisten dapat lebih memperjelas spesifikasi untuk tugas tugas kedepannya karena kami merasa kebingungan saat membaca spesifikasinya.

7. KELENGKAPAN SPESIFIKASI

Program dapat menerima input graf	✓
Program dapat menghitung lintasan terpendek dengan UCS	✓
Program dapat menghitung lintasan terpendek dengan A*	✓
Program dapat menampilkan lintasan terpendek serta jaraknya	✓
Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓