

IF2211 Strategi Algoritma

**PENCARIAN RUTE TERPENDEK DENGAN ALGORITMA
*UNIFORM COST SEARCH DAN A****

Laporan Tugas Kecil 3

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2022/2023



Oleh

Hosea Nathanael Abetnego 13521057

Chiquita Ahsanunnisa 13521129

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

DAFTAR ISI

DAFTAR ISI	1
BAB I	
DESKRIPSI MASALAH	3
1.1 Permasalahan Pencarian Rute Terdekat	3
1.2 Spesifikasi Program	4
1.2.1 Input dan Output	4
1.2.3 Bonus	4
BAB II	
LANDASAN TEORI	5
2.1 Jarak Euclidean	5
2.2 Algoritma	5
2.3 Algoritma Uniform Cost Search (UCS)	5
BAB III	
IMPLEMENTASI	1
3.1 Pemetaan Permasalahan menjadi Graf	1
3.2 Pencarian Rute Terdekat dengan Algoritma Uniform Cost Search (UCS)	1
3.3 Pencarian Rute Terdekat dengan Algoritma A*	1
3.4 Implementasi Algoritma	1
3.4.1 Struktur Data	1
BAB IV	
UJI COBA	1
4.1 Spesifikasi Komputer Uji Coba	1
4.2 Hasil Uji Coba	1
4.2.1 Uji Coba dengan Masukan File	1
BAB V	
SIMPULAN DAN SARAN	1
5.1 Simpulan	1
5.2 Saran	1
5.3 Komentar	1
LAMPIRAN	1
Lampiran 1 Link Repository GitHub	1
Lampiran 2 Tabel Check List Poin	1
DAFTAR REFERENSI	1

BAB I

DESKRIPSI MASALAH

1.1 Permasalahan Pencarian Rute Terdekat

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 1.1.1 Ilustrasi Permasalahan

(Sumber: Dokumentasi Penulis)

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 Spesifikasi Program

Program yang dibangun memanfaatkan algoritma *uniform cost search* dan *A** untuk mencari rute terdekat. Program ditulis dalam bahasa pemrograman C, C++, Java, Python, C#, Golang, Javascript, Ruby, Typescript, atau Kotlin. Program dapat berbasis aplikasi *desktop*, web, ataupun *mobile*. Berikut adalah detail dari spesifikasi program yang harus dibuat.

1.2.1 *Input* dan *Output*

Program memiliki spesifikasi sebagai berikut.

1. Program menerima *input file* graf (direpresentasikan sebagai matriks ketetanggan berbobot) dengan jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta atau 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. Program memiliki antarmuka dengan GUI atau *command line*.

1.2.3 Bonus

Spesifikasi bonus dari program adalah penggunaan 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.

BAB II

LANDASAN TEORI

2.1 Jarak *Euclidean*

Pada ruang dua dimensi, jarak *euclidean* antara dua titik $p(p_1, p_2)$ dan $q(q_1, q_2)$ dinyatakan dengan notasi $d(p, q)$ yang dapat dihitung dengan rumus di bawah ini.

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

2.2 Algoritma

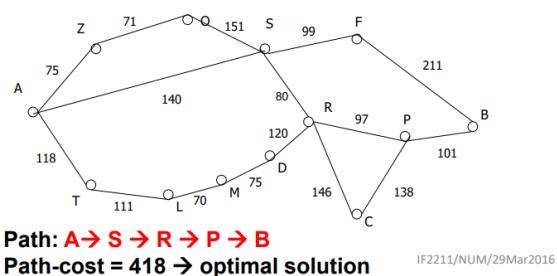
Algoritma adalah urutan langkah-langkah yang jelas dan terstruktur dalam menyelesaikan suatu permasalahan.

2.3 Algoritma *Uniform Cost Search* (UCS)

Algoritma *uniform cost search* (UCS) adalah algoritma pencarian dengan mengutamakan *cost* terendah untuk sampai tujuan. Algoritma ini merupakan jenis algoritma *uninformed search* dan bekerja pada pencarian berbobot dan berarah. UCS bekerja dengan mengutamakan ekspansi pada suatu *node* yang memiliki *cost* atau bobot terkecil terlebih dahulu yang biasanya dilakukan dengan *priority queue* dengan prioritiunya adalah jalur dengan bobot terkecil.

Uniform Cost Search (UCS)

- BFS & IDS find path with fewest steps (A-S-F-B)
- If steps ≠ cost, this is not relevant (to optimal solution)
- How can we find the shortest path (measured by sum of distances along path)?
- $g(n) = \text{path cost from root to } n$



IF2211/NUM/29Mar2016

Simpul-E	Simpul Hidup
A	Z _{A-75} , T _{A-118} , S _{A-140}
Z _{A-75}	T _{A-118} , S _{A-140} , O _{AZ-146}
T _{A-118}	S _{A-140} , O _{AZ-146} , L _{AT-229}
S _{A-140}	O _{AZ-146} , R _{AS-220} , L _{AT-229} , F _{AS-239} , O _{AS-291}
O _{AZ-146}	R _{AS-220} , L _{AT-229} , F _{AS-239} , O _{AS-291}
R _{AS-220}	L _{AT-229} , F _{AS-239} , O _{AS-291} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366}
L _{AT-229}	F _{AS-239} , O _{AS-291} , M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366}
F _{AS-239}	O _{AS-291} , M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366} , B _{ASF-450}
O _{AS-291}	M _{ATL-299} , P _{ASR-317} , D _{ASR-340} , C _{ASR-366} , B _{ASF-450}
M _{ATL-299}	P _{ASR-317} , D _{ASR-340} , D _{ATLM-364} , C _{ASR-366} , B _{ASF-450}
P _{ASR-317}	D _{ASR-340} , D _{ATLM-364} , C _{ASR-366} , B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
D _{ASR-340}	D _{ATLM-364} , C _{ASR-366} , B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
D _{ATLM-364}	C _{ASR-366} , B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
C _{ASR-366}	B _{ASRP-418} , C _{ASRP-455} , B _{ASF-450}
B _{ASRP-418}	Solusi ketemu

Gambar 2.3.1 Ilustrasi Algoritma

(Sumber: <https://informatika.stei.itb.ac.id/>)

Algoritma UCS memiliki skema umum sebagai berikut.

1. Ekspan semua *node* yang bersisian dengan *root node* dan urutkan dengan prioritas bobot terkecil.
2. Ekspan *node* dengan bobot terkecil tersebut, menandai *node* tersebut sudah dikunjungi, dan mengurutkan *node* yang telah diekspan sesuai dengan prioritasnya.
3. Jika menemukan rute ekspan yang mengarah ke sebuah *node* yang telah dikunjungi, rute tersebut diabaikan.
4. Lakukan langkah 2 dan 3 hingga menemukan *goal node*.

Berdasarkan skema di atas, algoritma UCS memiliki kompleksitas waktu asimtotik:

$$O(b^{(1 + C/\varepsilon)}) = b^{1 + C/\varepsilon} / (b - 1)$$

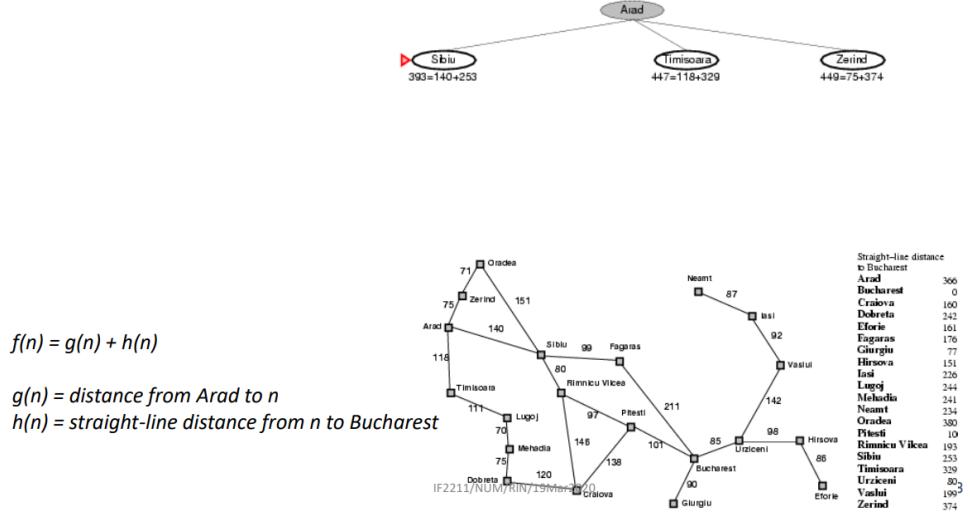
dengan $O(b^{(1 + C/\varepsilon)})$ adalah kompleksitas waktu asimtotik, b adalah faktor ekspansi, C adalah banyak langkah yang diambil, dan ε merupakan langkah menuju *goal node*.

Algoritma UCS tidak selalu menghasilkan rute paling optimal. Hal ini disebabkan karena UCS hanya memedulikan *cost* atau bobot terkecil dan tidak peduli berapa banyak langkah yang diambil untuk mencari *goal*-nya, sehingga yang terjadi terkadang pencarian akan bertemu kembali dengan *node* yang sudah pernah diekspan dan membuang-buang waktu atau rute yang diambil memiliki jarak lebih jauh.

2.4 Algoritma A*

Algoritma A* adalah algoritma pencarian dengan memanfaatkan prinsip dari algoritma UCS tetapi dengan modifikasi tambahan yaitu memprediksi jarak *node* selanjutnya ke *goal node*. Algoritma A* merupakan jenis *informed search* dan bekerja pada pencarian berbobot dan berarah. A* juga memanfaatkan *priority queue* sama seperti UCS, tetapi menambahkan jarak antara *node* tersebut dengan *goal node* sebagai konsiderasi prioritasnya.

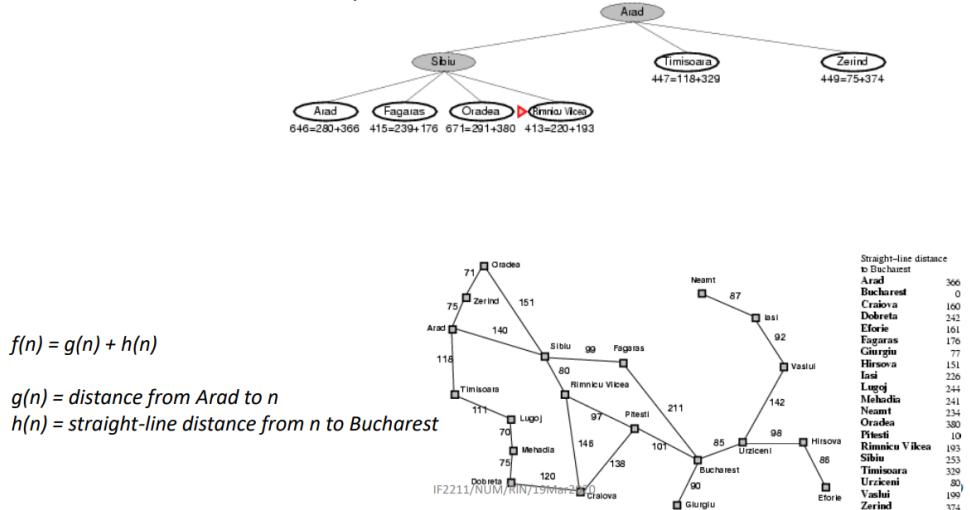
A* search example



Gambar 2.3.1 Ilustrasi Algoritma bag. 1

(Sumber: <https://informatika.stei.itb.ac.id/>)

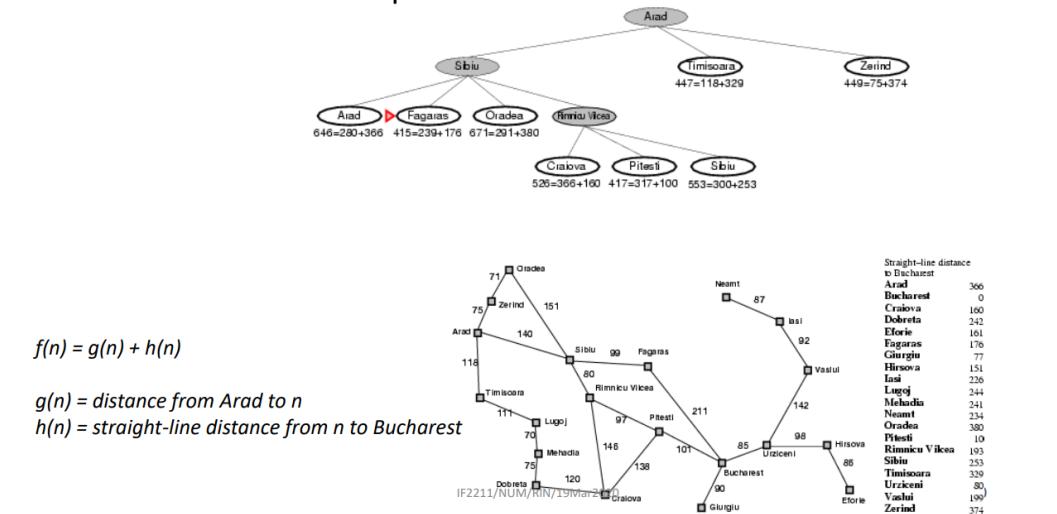
A* search example



Gambar 2.3.1 Ilustrasi Algoritma bag. 2

(Sumber: <https://informatika.stei.itb.ac.id/>)

A* search example



Gambar 2.3.1 Ilustrasi Algoritma bag. 3

(Sumber: <https://informatika.stei.itb.ac.id/>)

Algoritma A* memiliki skema umum sebagai berikut.

1. Ekspan semua node dari root dan urutkan dengan prioritas bobot dan prediksi jarak ke goal node terkecil.
2. Ekspan node dengan prioritas terkecil tersebut dan urutkan kembali node yang bersisian tersebut dengan prioritas seperti sebelumnya.
3. Lakukan langkah 2 hingga menemukan goal node.

Berdasarkan skema di atas, algoritma A* memiliki kompleksitas waktu asimtotik:

$$O(b^d) = b^d$$

dengan $O(b^d)$ adalah kompleksitas waktu asimtotik, b adalah faktor ekspansi, dan d adalah kedalaman goal node dari root node.

Algoritma A* akan menghasilkan rute paling optimal jika estimasi jarak dari sebuah node ke goal node tidak menaksir nilai yang berlebih. Akan tetapi, jika tidak menaksir nilai yang berlebih, A* menjadi lebih efektif karena tidak melakukan ekspansi berlebih ke node yang tidak diperlukan.

BAB III

IMPLEMENTASI

3.1 Pemetaan Permasalahan menjadi Graf

text

3.2 Pencarian Rute Terdekat dengan Algoritma *Uniform Cost Search* (UCS)

Berikut adalah skema algoritma pencarian rute terdekat dengan algoritma UCS yang diimplementasikan.

1. skema

Berikut adalah *pseudocode* untuk pencarian rute terdekat dengan algoritma UCS.

```
Me.resetAllVisisted
Dim queue = New PriorityQueue
Dim startPath = New Path
startPath.add(start)
queue.enqueue(startPath.clone)

While Not queue.isEmpty
    Dim currPath = queue.dequeue
    Dim currNum = currPath.getLast

    If (currNum = finish) Then
        Me.nodes(currNum).setVisisted(true)
        console.log(Search, finished)
        console.log(Result:= Result:, {, currPath.toString})
        Return currPath

    ElseIf Not Me.nodes(currNum).isVisisted Then
        Dim neighbors = Me.getUnvisitedNeighbors(currNum)
        neighbors.forEach(num => {
            Dim childPath = currPath.clone
            Dim currGnDiff = Me.mat[currPath.getLast][num].getDist
            childPath.add(num, currGnDiff)
            queue.enqueue(childPath)
        })
    End If
End While

Throw New Error("Could not find path")
```

Algoritma ini menggunakan *priority queue* untuk menyimpan *list* urutan ekspansi *node* dengan prioritas bobot (*currGnDiff*) ke node terakhir dalam *list* yang paling kecil. Di bagian awal, *list* diinisialisasi dengan *root node*. Kemudian dilakukan iterasi terhadap elemen dari *queue* tersebut. *Path* yang akan diekspansi diambil dari *queue* dan diambil elemen terakhirnya. Dilakukan pengecekan jika *node* tersebut merupakan *goal node*, jika ya maka rute (*path*) tersebut merupakan hasilnya, jika tidak maka dilakukan ekspansi pada *node* tersebut untuk *node-node* bersisian yang belum dikunjungi. Dilakukan iterasi terus menerus hingga

menemukan *goal node* atau *queue* habis yang berarti tidak ada rute menuju *goal node*.

3.3 Pencarian Rute Terdekat dengan Algoritma A*

Berikut adalah skema algoritma pencarian rute terdekat dengan algoritma A* yang diimplementasikan.

1. skema

Berikut adalah *pseudocode* untuk pencarian rute terdekat dengan algoritma A*.

```
Me.resetAllVisisted
Dim queue = New PriorityQueue
Dim startPath = New Path
startPath.add(start)
queue.enqueue(startPath.clone)

While Not queue.isEmpty
    Dim currPath = queue.dequeue
    Dim currNum = currPath.getLast

    If Me.nodes(currNum).isVisited Then
        continue
    End If

    Me.nodes(currNum).setVisited(true)

    If (currNum = finish) Then
        Return currPath
    Else
        Dim neighbors = Me.getUnvisitedNeighbors(currNum)
        neighbors.forEach(num => {
            Dim childPath = currPath.clone
            Dim currGnDiff = Me.mat[currPath.getLast][num].getDist
            Dim currHnDiff =
                Me.nodes(num).getCoordinate.getDistance(Me.nodes(finish).getCoordinate)
            childPath.add(num, currGnDiff, currHnDiff)
            queue.enqueue(childPath)
        })
    End If
End While

Throw New Error("Could not find path")
```

Algoritma ini juga memanfaatkan *priority queue* untuk menyimpan *list* urutan ekspansi *node* dengan prioritas bobot (*currGnDiff*) ke *node* tersebut dan estimasi jarak *node* tersebut ke *goal node* (*currHnDiff*). Sama seperti UCS, dilakukan inisialisasi terlebih dahulu dengan *queue* berisi *path* dengan *start node*. Dilakukan iterasi untuk setiap elemen pada *queue* dengan mengambil elemen terakhir dari *list* yang diambil dari *queue*. Cek jika *node* tersebut sudah pernah dikunjungi, jika ya maka lanjut ke elemen pada *queue* selanjutnya. Jika elemen tersebut belum pernah dikunjungi dan merupakan *goal node*, maka rute (*path*)

tersebut adalah hasilnya, jika tidak semua *node* tetangga dari *node* tersebut yang belum dikunjungi, diekspan dan dimasukkan ke *priority queue*. Iterasi terus dilakukan hingga ditemukan *goal node* atau *queue* habis yang berarti tidak ada rute menuju *goal node*.

3.4 Implementasi Algoritma

Algoritma yang telah dipaparkan pada bagian sebelumnya diimplementasikan pada bahasa pemrograman TypeScript (.ts)

3.4.1 Struktur Data

3.4.1.1 GraphMatrix

```
// nodes menyimpan map dengan key adalah id node dan
// value adalah objek Node
private nodes: { [key: number]: Node } = {};

// Matriks adjacency
private mat: Edge[][] = [];

// Panjang list / banyak titik
private length: number;

// Konstruktor kelas
constructor(filePath: string) { }

// Mengubah status visited semua node menjadi false
resetAllVisisted(): void { }

// Memperoleh node yang bersisian dengan node i yang
// belum dikunjungi
getUnvisitedNeighbors(i: number): Array<number> { }

// Memulai algoritma pencarian menggunakan UCS dengan
// start sebagai start node dan finish sebagai goal node
searchUCS(start: number, finish: number): Path { }
```

```

// Memulai algoritma pencarian menggunakan A* dengan
start sebagai start node dan finish sebagai goal node
searchAStar(start: number, finish: number): Path { }

// Method menampilkan node dan matriks adjacency yang
telah dibentuk
toString(): string { }

// Memperoleh array posisi dari Path yang dihasilkan
getPolyline(path: Path): Array<Position> { }

// Memperoleh nama jalan yang diambil
getPolylineName(path: Path): string { }

// Memperoleh array number yang merupakan node untuk
ditampilkan
getPolylineArr(path: Path): Array<Array<number>> { }

```

3.4.1.2 Priority Queue

```

// Penyimpanan Queue
private queue: Array<Path> = [];

// Konstruktor kelas
constructor() {}

// Memasukkan elemen ke dalam list dengan prioritas bobot
path terkecil
enqueue(path: Path): void { }

// Mengeluarkan elemen dengan prioritas terkecil
dequeue(): Path { }

// Mengembalikan true jika queue kosong
isEmpty(): boolean { }

// Mengembalikan panjang queue
getLength(): number { }

// mengembalikan elemen ke-i dari queue
getAt(i: number): Path { }

```

```
// Menampilkan queue  
toString(): string { }
```

3.4.1.3 Path

```
// List yang menyimpan rute node  
private pathList: Array<number> = [];  
  
// cost dari rute yang ada di pathList  
private Gn: number = 0;  
  
// estimasi jarak node terakhir di list ke goal node  
private Hn: number = 0;  
  
// Konstruktor kelas  
constructor() {}  
  
// Memasukkan node ke pathList dengan perhitungan Gn dan  
Hn  
add(node: number, GnDiff: number = 0, newHn: number =  
0): void {}  
  
// Mengembalikan elemen terakhir pada pathList  
getLast(): number {}  
  
// Mengembalikan elemen ke-i pada pathList  
getAt(index: number): number {}  
  
// Memperoleh prioritas dari path (Gn + Hn)  
getPrio(): number {}  
  
// Memperoleh panjang dari list  
getPathLength(): number {}  
  
// Meng-copy list  
clone(): Path {}  
  
// Mengembalikan path dalam bentuk string  
toString(): string {}
```

3.4.1.4 Node

```
// Menyimpan koordinat node
private coordinate: Position;

// Informasi keterkunjungan node
private visited: boolean = false;

// Informasi nama node
private name: string;

// Informasi id node
private id: number;

// Informasi banyak node yang telah dibentuk
private static numberOfNodes: number = 0;

// Konstruktor kelas
constructor(coordinate: Position, name: string = `Node
${Node.numberOfNodes}`) { }

// Memperoleh koordinat node dalam bentuk kelas Position
getCoordinate(): Position { }

// Mengembalikan informasi keterkunjungan node
isVisited(): boolean { }

// Mengembalikan id node
getId(): number { }

// Mengembalikan nama node
getName(): string { }

// Mengubah informasi keterkunjungan node
setVisited(visited: boolean): void { }

// Mengubah koordinat node
setCoordinate(coordinate: Position): void { }

// Mengubah nama node
setName(name: string): void { }
```

```

// Mengubah keterkunjungan node menjadi false
resetVisited(): void { }

// Mengembalikan banyak node yang telah dibentuk
static getNumOfNodes(): number { }

// Mengubah banyak node yang telah dibentuk menjadi 0
static resetNumOfNodes(): void { }

// Penampilan node
toString(): string { }

```

3.4.1.5 Position

```

// Menyimpan posisi node
private x: number;
private y: number;

// Konstruktor kelas
constructor(x: number, y: number) { }

// Mengembalikan posisi pada sumbu x
getX(): number { }

// Mengembalikan posisi pada sumbu y
getY(): number { }

// Mengubah posisi pada sumbu x
setX(x: number) { }

// Mengubah posisi pada sumbu y
setY(y: number) { }

// Memperoleh jarak euclidean distance antar node
getDistance(other: Position): number { }

// Mengembalikan posisi dalam format [number, number]
getPosArr(): [number, number] { }

// Penampilan posisi node
toString(): string { }

```

3.4.1.6 Edge

```
// Menyimpan informasi nama jalan
private street: string;

// Menyimpan informasi panjang jalan
private dist: number;

// Konstruktor kelas
constructor(street: string, dist: number) { }

// Memperoleh nama jalan
getStreet(): string { }

// Memperoleh panjang jalan
getDist(): number { }

// Mengubah nama jalan
setStreet(street: string): void { }

// Mengubah panjang jalan
setDist(dist: number): void { }

// Penampilan Edge
toString(): string { }

// Memperoleh true jika edge menghubungkan dua node
isConnected(): boolean { }
```

BAB IV

UJI COBA

4.1 Spesifikasi Komputer Uji Coba

Uji coba dilakukan pada komputer dengan spesifikasi sebagai berikut.

Tabel 4.1.1 Spesifikasi Komputer Uji Coba

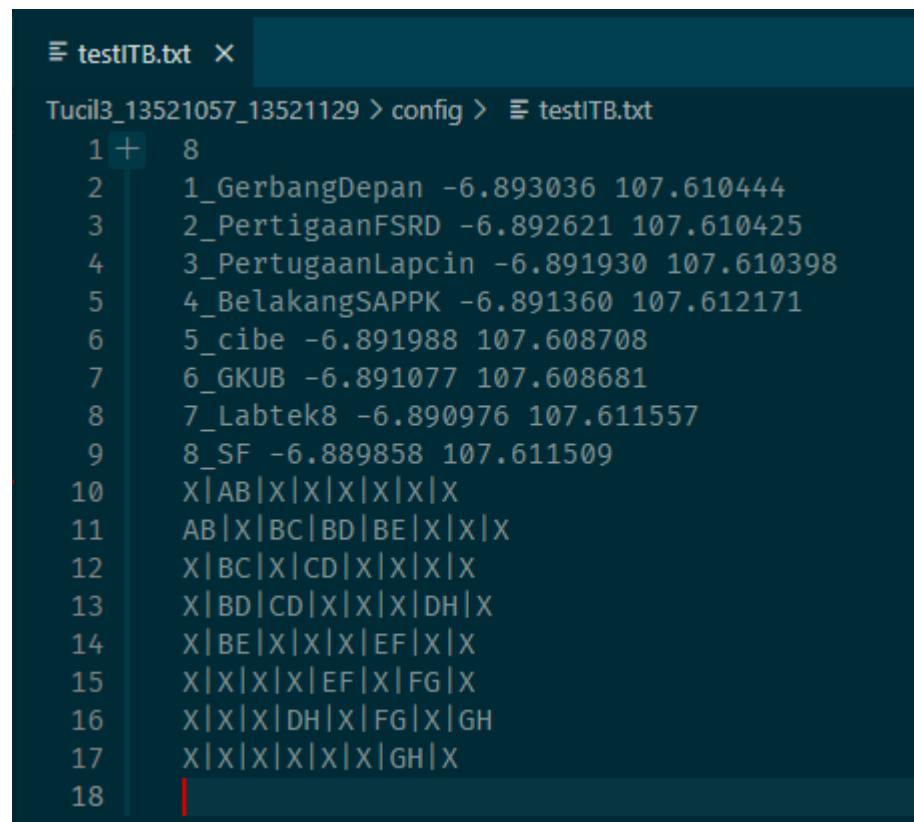
Jenis	Macbook / PC
Prosesor	2,3 GHz Dual-Core Intel Core i5
RAM	16 GB 2133 MHz DDR4
OS	macOS

4.2 Hasil Uji Coba

4.2.1 Uji Coba dengan Masukan File

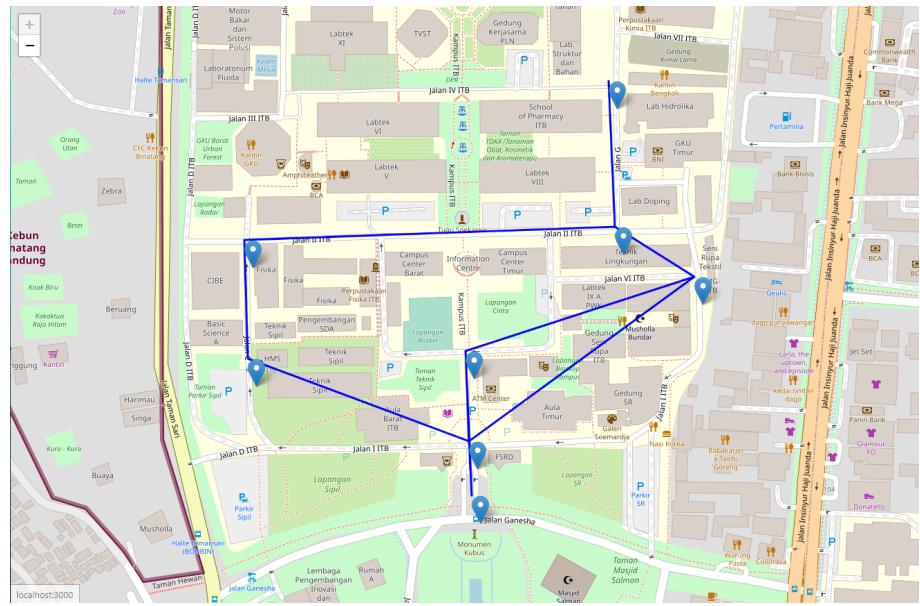
4.2.1.1 Daerah Sekitar ITB

Input:



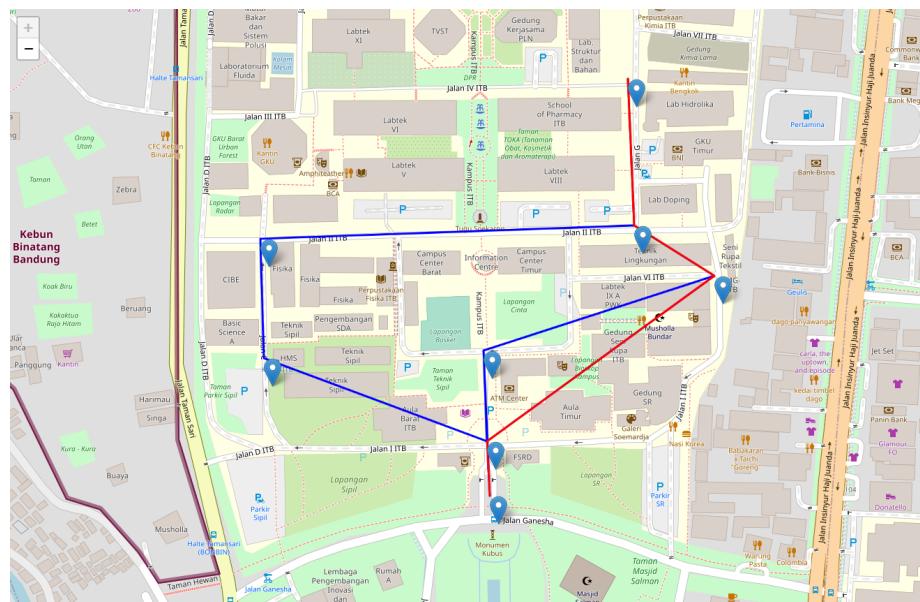
```
testITB.txt > config > testITB.txt
1 + 8
2 1_GerbangDepan -6.893036 107.610444
3 2_PertigaanFSRD -6.892621 107.610425
4 3_PertugaanLapcin -6.891930 107.610398
5 4_BelakangSAPPK -6.891360 107.612171
6 5_cibe -6.891988 107.608708
7 6_GKUB -6.891077 107.608681
8 7_Labtek8 -6.890976 107.611557
9 8_SF -6.889858 107.611509
10 X|AB|X|x|x|x|x|x
11 AB|X|BC|BD|BE|x|x|x
12 X|BC|x|CD|x|x|x|x
13 X|BD|CD|x|x|x|DH|x
14 X|BE|x|x|x|EF|x|x
15 X|x|x|x|EF|x|FG|x
16 X|x|x|DH|x|FG|x|GH
17 X|x|x|x|x|x|GH|x
18 |
```

Graf:



4.2.1.1.1 Algoritma A*

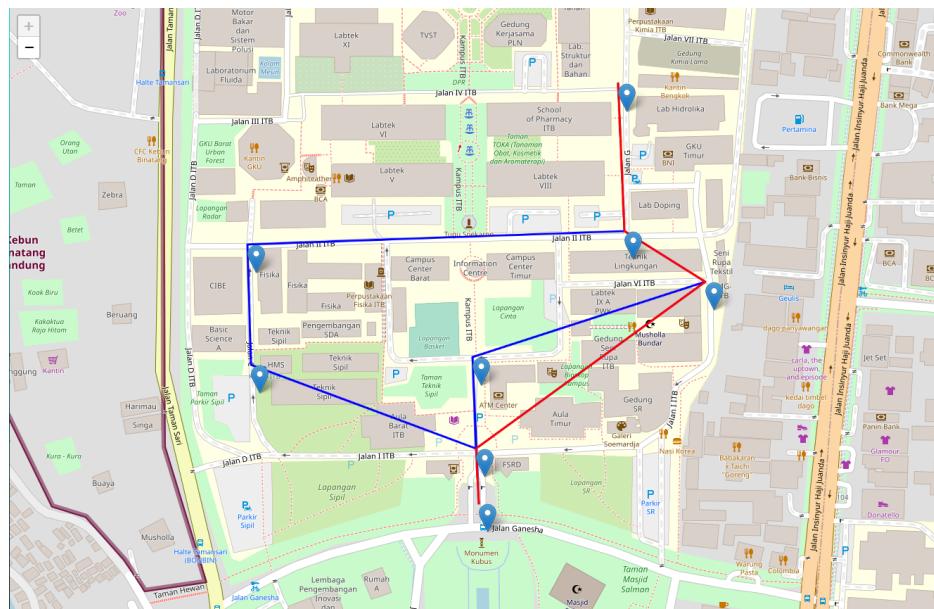
Solusi:



Path Cost : 489.08530258681196 meter

4.2.1.1.2 Algoritma UCS

Solusi:



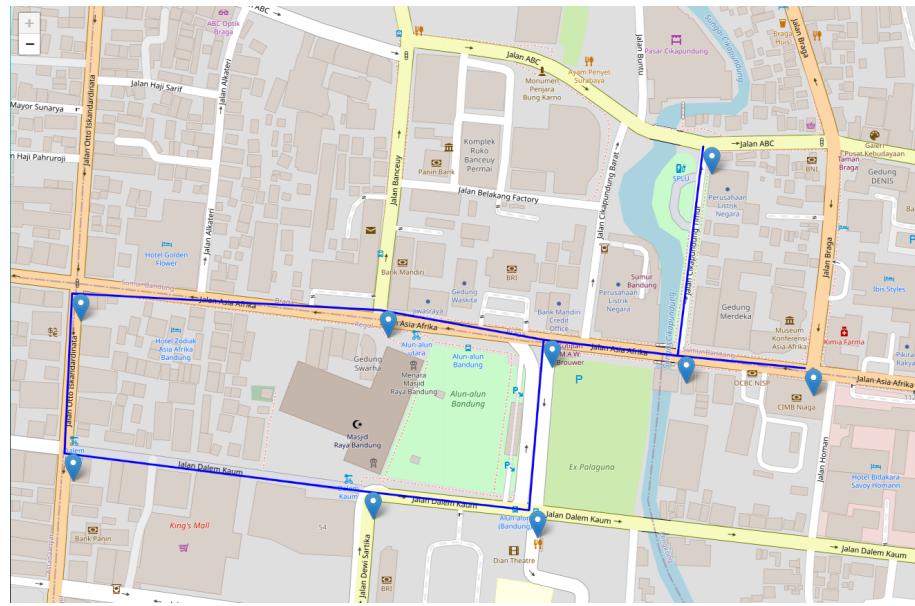
Path Cost : 489.08530258681196 meter

4.2.1.2 Daerah Alun-Alun

Input:

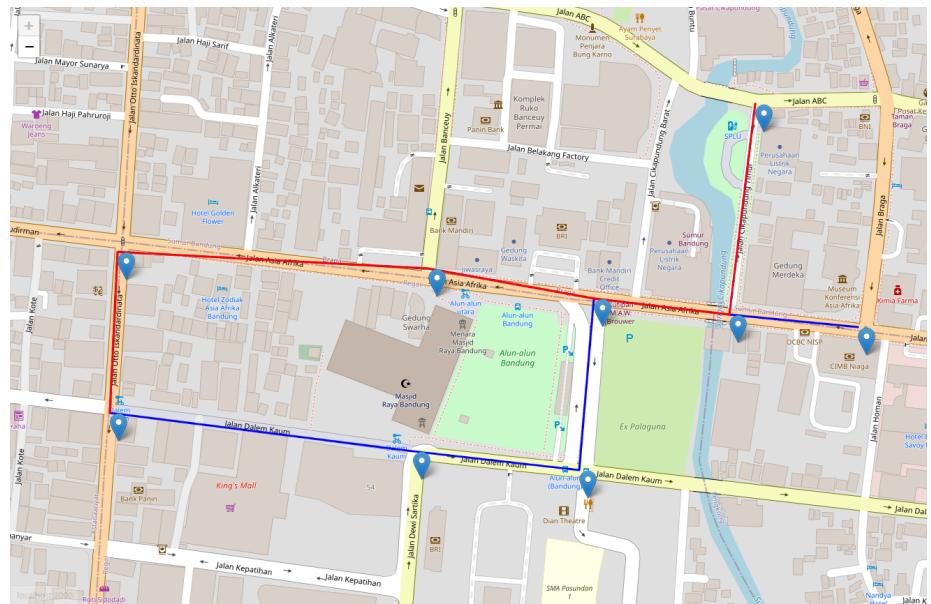
```
testAlunAlun.txt X
Tucil3_13521057_13521129 > config > testAlunAlun.txt
You, 7 seconds ago | 1 author (You)
1 9
2 1_HalteTrans -6.922095 107.604019
3 2_BRIAlunAlun -6.921223 107.607737
4 3_JembatanBelakangMuseum -6.921347 107.608779
5 4_HalteAlunAlun -6.922536 107.607622
6 5_PertigaanSatopolPP -6.922392 107.606349
7 6_KantorPos -6.920996 107.606465
8 7_Norsefiecden -6.920868 107.604078
9 8_MuseumKonferensiAsiaAfrika -6.921444 107.609762
10 9_TamanAsiaAfrika -6.919739 107.608975
11 X|X|X|X|J|X|A|X|X
12 X|X|B|D|X|E|X|X|X
13 X|B|X|X|X|X|X|H|C
14 X|D|X|X|F|X|X|X|X
15 J|X|X|F|X|X|X|X|X
16 X|E|X|X|X|X|I|X|X
17 A|X|X|X|X|I|X|X|X
18 X|X|H|X|X|X|X|X|X
19 X|X|C|X|X|X|X|X|X
20 +
```

Graf:



4.2.1.2.1 Algoritma A*

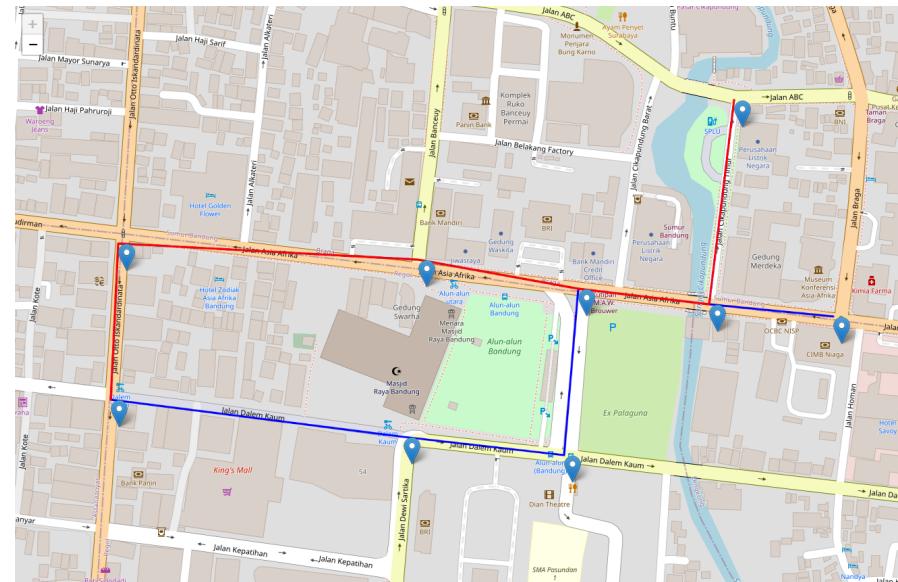
Solusi:



Path Cost : 839.0811246079234 meter

4.2.1.2.2 Algoritma UCS

Solusi:



Path Cost : 839.0811246079234 meter

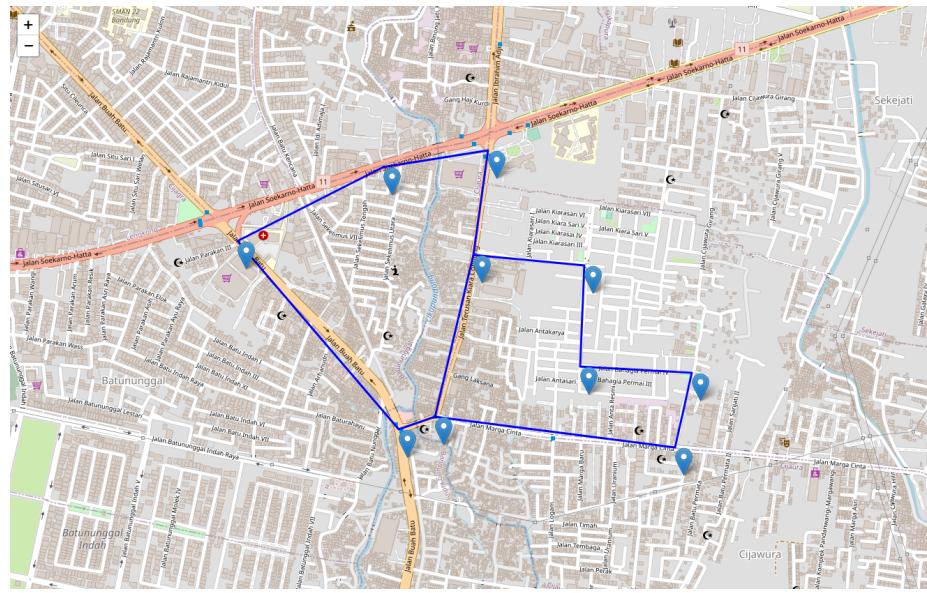
4.2.1.3 Daerah Buah Batu / Bandung Selatan

Input:

```
testBuahBatu.txt X

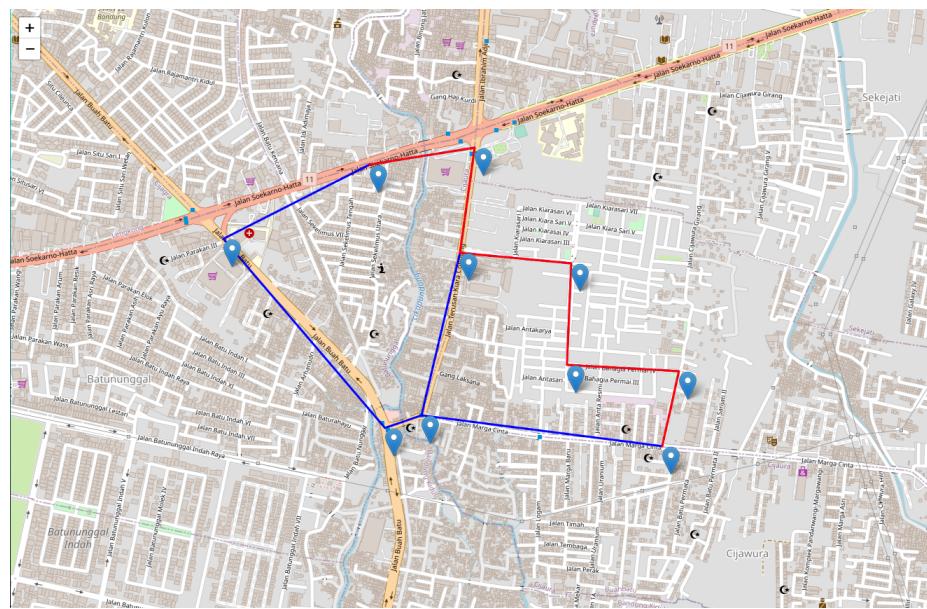
D: > Institut Teknologi Bandung > ITB STEI-G > SMT 4 > Stima > Tucil > Tucil3_
1 10
2 1_MargacintaPark -6.955034 107.647558
3 2_MasjidAgungBuahBatu -6.954105 107.640171
4 3_PasarKordon -6.954475 107.639066
5 4_RSMayapada -6.948735 107.634099
6 5_SamsatSoekarnoHatta -6.945952 107.641813
7 6_JlBahagiaPermaiIV -6.952744 107.648087
8 7_CurvasudMilanewow -6.952563 107.644643
9 8_SPDBandung -6.949475 107.644769
10 9_AmertaIndahOtsuka -6.949145 107.641368
11 10_JlnTol -6.946480 107.638600
12 X|B|X|X|X|K|X|X|X|X
13 B|X|G|X|X|X|X|X|D|X
14 X|G|X|F|X|X|X|X|X|X
15 X|X|F|X|X|X|X|X|X|E
16 X|X|X|X|X|X|X|X|X|G
17 K|X|X|X|X|X|J|X|X|X
18 X|X|X|X|X|J|X|K|X|X
19 X|X|X|X|X|X|K|X|C|X
20 X|D|X|X|I|X|X|C|X|X
21 X|X|X|E|G|X|X|X|X|X
```

Graf:



4.2.1.3.1 Algoritma A*

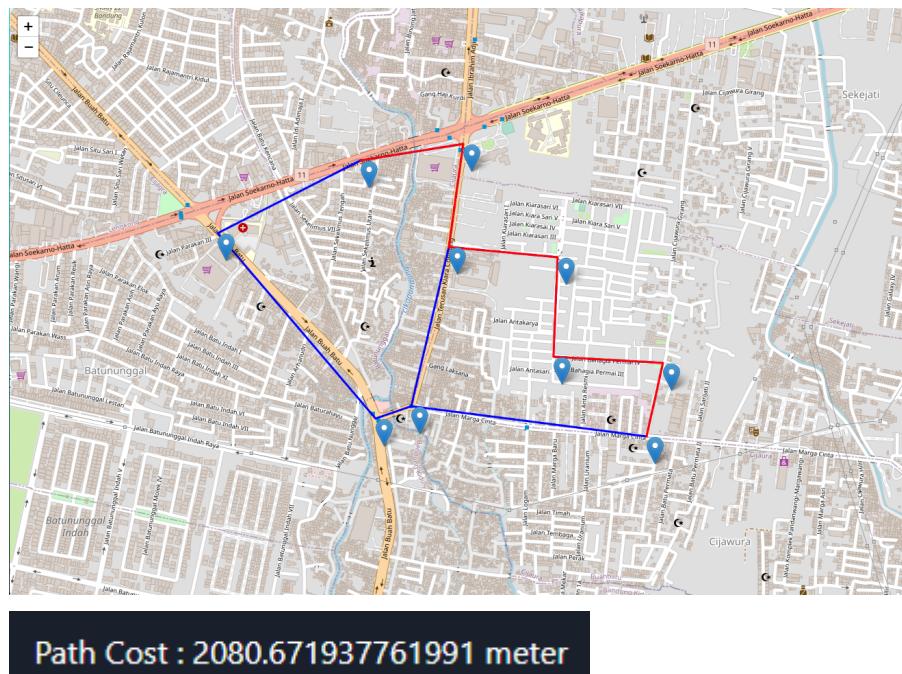
Solusi:



Path Cost : 2080.671937761991 meter

4.2.1.3.2 Algoritma UCS

Solusi:



4.2.1.4 Daerah Planet Bekasi

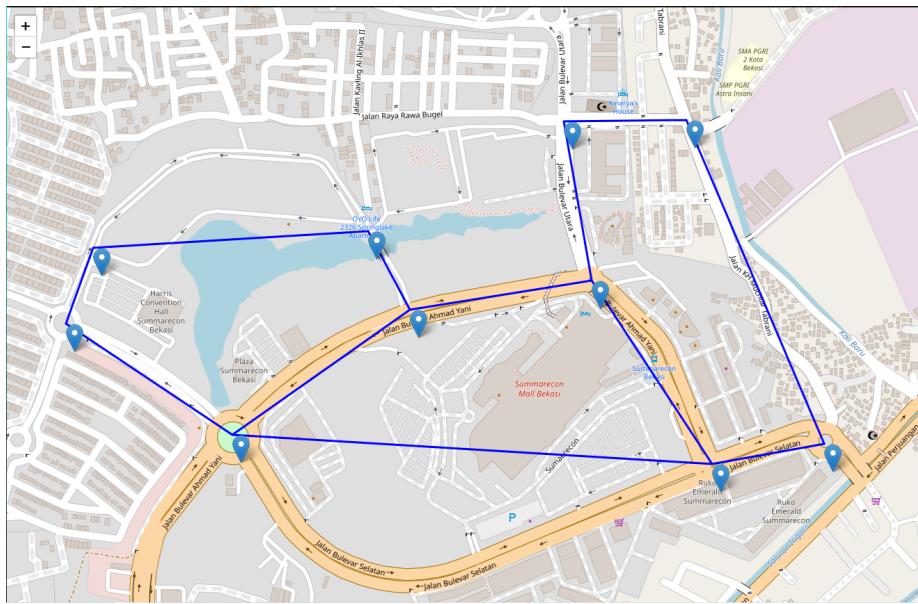
Input:

```
testPlanetBekasi.txt ×

D: > Insitut Teknologi Bandung > ITB STEI-G > SMT 4 > Stima > Tucil > Tucil

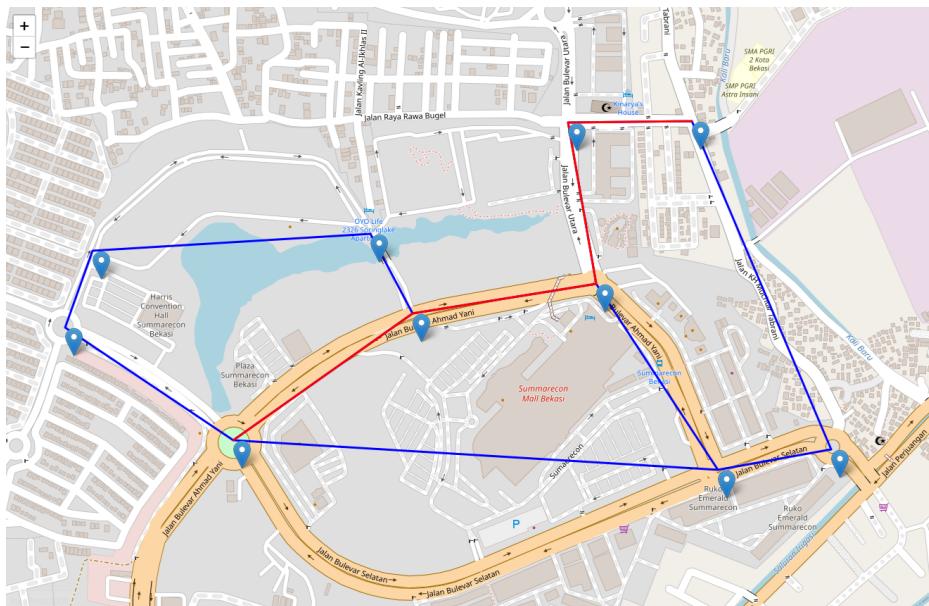
1 10
2 1_PyramidSummarecon -6.226710 106.996346
3 2_JembatanCBD -6.224790 106.999076
4 3_MasukApart -6.223579 106.998438
5 4_Lake -6.223835 106.994195
6 5_Komplek -6.225003 106.993777
7 6_BundaranPenabur -6.224335 107.001874
8 7_Sinpasa -6.227156 107.003730
9 8_BundaranKali -6.226847 107.005468
10 9_GangPenabur -6.221898 107.001445
11 10_Tytyan -6.221877 107.003333
12 X|A|X|X|B|X|C|X|X|X
13 A|X|L|X|X|F|X|X|X|X
14 X|L|X|J|X|X|X|X|X|X
15 X|X|J|X|K|X|X|X|X|X
16 B|X|X|K|X|X|X|X|X|X
17 X|F|X|X|X|X|E|X|I|X
18 C|X|X|X|X|E|X|D|X|X
19 X|X|X|X|X|X|D|X|X|G
20 X|X|X|X|X|I|X|X|X|H
21 X|X|X|X|X|X|X|G|H|X
```


Graf:



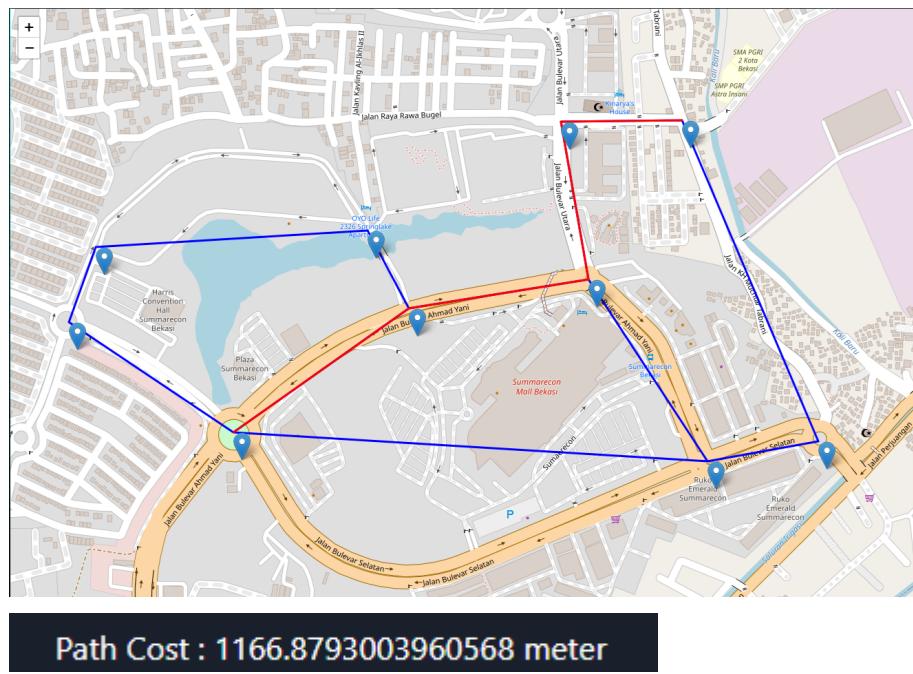
4.2.1.4.1 Algoritma A*

Solusi:



4.2.1.4.2 Algoritma UCS

Solusi:



BAB V

SIMPULAN DAN SARAN

5.1 Simpulan

Pada tugas kecil ini telah berhasil diimplementasikan algoritma UCS dan A* untuk memperoleh rute terpendek dari titik awal ke titik akhir pada peta. Kedua algoritma yang digunakan menghasilkan solusi yang optimal (jika nilai heuristic A* *admissible*) dengan jarak terkecil. Pada pencarinya, UCS dan A* memiliki kompleksitas waktu yang berbeda dengan A* bekerja lebih cepat daripada UCS karena nilai heuristic A* bisa membantu pencarian menjadi lebih cepat tetapi hanya di beberapa kondisi tertentu saja. Oleh karena itu, dapat disimpulkan bahwa algoritma A* merupakan algoritma yang lebih baik dibandingkan UCS pada kasus yang optimal.

5.2 Saran

Penulis menyarankan kepada pengembang selanjutnya untuk mengembangkan kode program yang telah dibuat pada tugas kecil ini dengan modularitas, strategi, kebersihan kode dan hal-hal lainnya dengan lebih baik. Penulis juga menyarankan kepada pengembang selanjutnya untuk melakukan eksplorasi yang lebih terhadap strategi yang digunakan maupun bahasanya.

5.3 Komentar

Pada tugas kecil ini, penulis belajar bagaimana cara mengimplementasikan algoritma UCS dan A* dalam bahasa pemrograman Typescript dengan *web-based platform*. Penulis juga belajar untuk mengatur waktu sehingga dapat mengerjakan tugas tanpa harus terburu-buru. Tidak hanya itu, penulis juga belajar seberapa pentingnya untuk mengembangkan kemampuan dalam bidang-bidang lain di dalam dunia informatika.

LAMPIRAN

Lampiran 1 Link Repository GitHub

https://github.com/ashnchiquita/Tucil3_13521057_13521129

Lampiran 2 Tabel *Check List Poin*

No.	Poin	Ya	Tidak
1.	Program dapat menerima <i>input</i> graf	✓	
2.	Program dapat menghitung lintasan terpendek dengan algoritma UCS	✓	
3.	Program dapat menghitung lintasan terpendek dengan algoritma 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	1/2	

DAFTAR REFERENSI

Munir, R. (2023). *Strategi Algoritma: Penentuan Rute (Bagian 1)*. Dilansir dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>. Diakses pada 5 April 2023.

Munir, R. (2023). *Strategi Algoritma: Penentuan Rute (Bagian 1)*. Dilansir dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>. Diakses pada 5 April 2023.