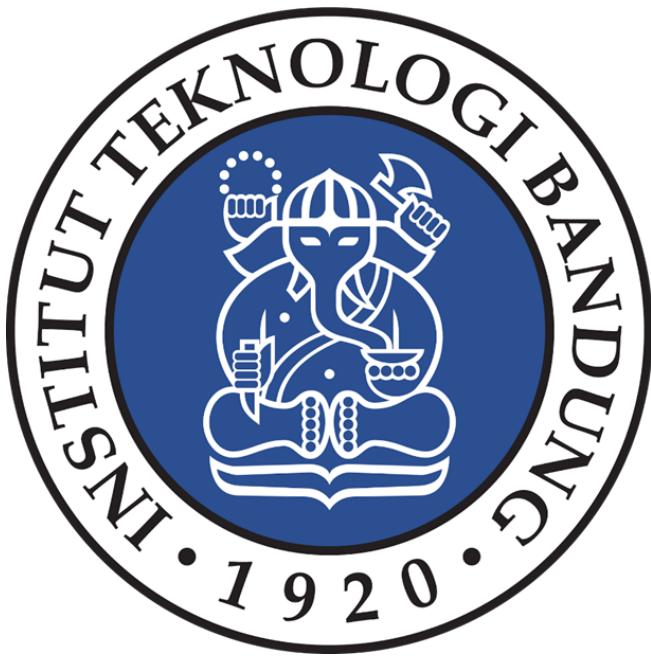


TUGAS KECIL 03

IMPLEMENTASI ALGORITMA UCS DAN A*

UNTUK MENENTUKAN LINTASAN TERPENDEK

IF2211 – STRATEGI ALGORITMA



Disusun oleh:

Muhammad Fadhil Amri 13521066

Hanif Muhammad Zhafran 13521157

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

Daftar Isi

Daftar Isi	2
Bab I : Deskripsi Persoalan	3
Bab II: Algoritma UCS dan A*	4
2.1 Overview Algoritma UCS	4
2.2 Penerapan Algoritma UCS	4
2.3 Overview Algoritma A*	4
2.4 Penerapan Algoritma A*	5
Bab III: Source Code Program	6
3.1 index.html	6
3.2 Coordinate.js	7
3.3 Node.js	7
3.4 Route. js	10
3.5 PrioQueue.js	10
3.6 IO.js	12
3.7 AStar.js	14
3.8 UCS.js	16
3.9 Map.js	17
Bab IV: Contoh Masukan dan Keluaran	22
Bab V: Simpulan	32
Lampiran	33

Bab I : 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.

Bab II: Algoritma UCS dan A*

2.1 Overview Algoritma UCS

Uniform-Cost Search merupakan algoritma pencarian tanpa informasi (uninformed search) yang menggunakan biaya kumulatif terendah untuk menemukan jalur dari node sumber ke node tujuan. Algoritma ini beroperasi di sekitar ruang pencarian berbobot terarah untuk berpindah dari node awal ke salah satu node akhir dengan biaya akumulasi minimum. Algoritma Uniform-Cost Search masuk dalam algoritma pencarian uninformed search atau blind search karena bekerja dengan cara brute force, yaitu tidak mempertimbangkan keadaan node atau ruang pencarian. Algoritma ini umumnya digunakan untuk menemukan jalur dengan biaya kumulatif terendah dalam graph berbobot di mana node diperluas sesuai dengan biaya traversalnya dari node root. Biasanya algoritma Uniform-Cost Search diimplementasikan dengan menggunakan priority queue di mana prioritasnya adalah menurunkan biaya operasi. Uniform-Cost Search juga dapat disebut sebagai varian dari algoritma Dijkstra. Hal ini karena pada uniform cost search, alih-alih memasukkan semua simpul ke dalam antrian prioritas (priority queue), kita hanya menyisipkan node sumber, lalu memasukkan satu per satu bila diperlukan. Di setiap iterasi, kita memeriksa apakah item sudah dalam antrian prioritas (menggunakan array yang dikunjungi). Jika ya, kita melakukan kunci penurunan, jika tidak, kita memasukkan item tersebut ke dalam antrian.

2.2 Penerapan Algoritma UCS

1. Inisialisasi PRIORITY QUEUE (Priority Queue berdasarkan Cost => *cumulative distance*)
2. Letakkan simpul awal pada PRIORITY QUEUE
3. Lakukan langkah-langkah berikut sampai ditemukan node goal:
 - o *Dequeue* PRIORITY QUEUE
 - o Evaluasi *currentNode* hasil *dequeue*
 - o Jika *currentNode* adalah goal, pencarian dihentikan dan route menuju goal diperoleh beserta *cost*-nya
 - o Jika current node bukan goal, *enqueue* PRIORITY QUEUE dengan semua route menuju setiap tetangga *currentNode* yang masih bisa di-expand
 - o Tetangga *currentNode* yang bisa di-expand adalah node yang belum pernah dikunjungi,

2.3 Overview Algoritma A*

Algoritma A* (A Star) adalah algoritma pencarian yang digunakan untuk menemukan jalur terpendek antara titik awal dan akhir. Algoritma ini sering digunakan untuk penjelajahan peta guna menemukan jalur terpendek yang akan diambil. A* awalnya dirancang sebagai masalah penjelajahan graph (graph traversal), untuk membantu robot agar dapat menemukan arahnya sendiri. A* saat ini masih

tetap menjadi algoritma yang sangat populer untuk graph traversal. Algoritma A* mencari jalur yang lebih pendek terlebih dahulu, sehingga menjadikannya algoritma yang optimal dan lengkap. Algoritma yang optimal akan menemukan hasil yang paling murah dalam hal biaya untuk suatu masalah, sedangkan algoritma yang lengkap menemukan semua hasil yang mungkin dari suatu masalah. Aspek lain yang membuat A* begitu powerful adalah penggunaan graph berbobot dan nilai heuristic dalam penerapannya. Graph berbobot menggunakan angka untuk mewakili biaya pengambilan setiap jalur atau tindakan. Ini berarti bahwa algoritma dapat mengambil jalur dengan biaya paling sedikit, dan menemukan rute terbaik dari segi jarak dan waktu. Sementara itu, nilai heuristic yang merupakan penyebab A* tergolong ke dalam *informed search* dapat mengurangi node yang dikunjungi secara signifikan karena nilai heuristic dapat mengarahkan penelusuran ke arah yang lebih dekat menuju goal. Adapun kelemahan utama dari algoritma ini adalah kompleksitas ruang dan waktunya. Algoritma A* membutuhkan banyak ruang untuk menyimpan semua kemungkinan jalur dan banyak waktu untuk menemukannya.

2.4 Penerapan Algoritma A*

Algoritma A* dijalankan setelah nilai heuristic untuk setiap node telah di-set. Berikut langkah-langkah penerapan algoritma A* pada persoalan ini.

1. Inisialisasi OPEN LIST (Priority Queue berdasarkan Cost)
2. Letakkan simpul awal pada OPEN LIST
3. Inisialisasi CLOSE LIST
4. Lakukan langkah-langkah berikut sampai ditemukan node goal:
 - *Dequeue* OPEN LIST
 - Evaluasi *currentNode* hasil *dequeue*
 - Jika *currentNode* adalah goal, pencarian dihentikan dan route menuju goal diperoleh beserta *cost*-nya
 - Jika current node bukan goal, *enqueue* OPEN LIST dengan semua route menuju setiap tetangga *currentNode* yang masih bisa di-expand
 - Tetangga *currentNode* yang bisa di-expand adalah node yang belum pernah dikunjungi, atau *currentNode* yang sudah pernah dikunjungi tetapi memiliki $cost \leq cost$ route menuju node yang ada pada CLOSE LIST
 - *Cost* route berasal dari *cost* path menuju node (*cumulative distance*) + nilai heuristic dari node
 - Tambahkan *currentNode* ke dalam CLOSE LIST

Bab III: Source Code Program

Program ditulis menggunakan bahasa pemrograman Javascript node.js v16.13.2 dan HTML 5 pada sistem operasi Windows 11.

3.1 index.html

```
src > index.html > {} "index.html"
1  <html>
2
3  <head>
4      <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
5          integrity="sha256-kLaT2GO5pIechnsozB+fInD+UyjE2L1fWPgU04xyI=" crossorigin="" />
6      <script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
7          integrity="sha256-WBkxX0wTeyKc1OHuHtc-12uENFpDZ9YFd5Hf+D7ewM=" crossorigin=""></script>
8      <link rel="stylesheet" href="styles.css" />
9      <script type="module">
10         import { toggleUCS, toggleAStar, toggleAdd, toggleDelete, togglePath, toggleSearch, saveFileMap, resetMapState, plotToMap } from './Map.js';
11         import { readInputFile } from './IO.js';
12
13         document.querySelector('#UCS').addEventListener('click', toggleUCS);
14         document.querySelector('#Astar').addEventListener('click', toggleAStar);
15         document.querySelector('#Save').addEventListener('click', saveFileMap);
16         document.querySelector('#Add').addEventListener('click', toggleAdd);
17         document.querySelector('#Path').addEventListener('click', togglePath);
18         document.querySelector('#Delete').addEventListener('click', toggleDelete);
19         document.querySelector('#Search').addEventListener('click', toggleSearch);
20         document.querySelector('#Reset').addEventListener('click', resetMapState);
21         document.querySelector('#Plot').addEventListener('click', plotToMap);
22         document.querySelector('#graphinput').addEventListener('input', readInputFile);
23     </script>
24
25 </head>
26
27 <body>
28     <div id="map"></div>
29     <div class="content">
30         <div class="c1">
31             <h3>Path Planning</h3>
32         </div>
33         <div class="c2">
34             <form>
35                 <br>
36                 <div>
37                     <h3>Algorithm</h3>
38
39                     <input type="radio" id="UCS" name="algo" value="UCS" checked="checked">
40                     <label for="UCS">UCS</label>
41
42                     <input type="radio" id="Astar" name="algo" value="Astar">
43                     <label for="Astar">A*</label>
44
45                 </div>
46             <br>
47             <div class="inputtxt">
48                 <input type="text" value="52.27064, 106.51545, 52.27064, 106.51545" />
49             </div>
50         </form>
51     </div>
52 </body>
```

```

45     <div class="inputtxt">
46         <h3>Input File:</h3>
47         <input type="file" id="graphinput" name="graphinput" accept=".txt">
48         <br>
49         <button type="button" id="Plot" name="Plot" accept=".txt">Plot Data</button>
50     </div>
51     <div class="outputtxt">
52         <h3>Output to File:</h3>
53         <button type="button" id="Save" name="Save">Save File</button>
54     </div>
55     <br>
56     <br>
57     <div>
58         <h3>Cursor Modes:</h3>
59         <input type="radio" id="Add" name="mode" value="Add" checked="checked">
60         <label for="Add">Marker Mode</label><br>
61         <input type="radio" id="Path" name="mode" value="Path">
62         <label for="Path">Path Mode</label><br>
63         <input type="radio" id="Delete" name="mode" value="Delete">
64         <label for="Delete">Delete Mode</label><br>
65         <input type="radio" id="Search" name="mode" value="Search">
66         <label for="Search">Search Mode</label>
67     </div>
68     <br>
69     <br>
70     <button type="button" id="reset" name="reset">Reset State</button>
71
72 </div>
73 </div>
74 </body>
75
76 </html>
```

Ln 10, Col 119 Spaces:4 UTF-8 CR/LF HTML Ø Port:55

File ini berisi main program yang merupakan program untuk *website local* yang memiliki fungsionalitas untuk menerima input dari file atau dari map, pemilihan node awal dan tujuan, pemilihan algoritma pencarian rute yang akan digunakan (UCS atau A*), dan output hasil pencarian berupa rute dan *cost*-nya.

3.2 Coordinate.js

File ini berisi kelas Coordinate yang merepresentasikan koordinate dari sebuah *node* yang ada pada graph

```

src > js Coordinate.js > ...
...
1  // Class yang merepresentasikan koordinat node pada graph
2
3  // Attribute:
4  // - x: koordinat x
5  // - y: koordinat y
6
...
7  class Coordinate {
8      constructor(x, y) {
9          this.x = x;
10         this.y = y;
11     }
12
13     getX() { return this.x; }
14     getY() { return this.y; }
15     setX(x) { this.x = x; }
16     setY(y) { this.y = y; }
17 }
18
19 module.exports = { Coordinate };
```

3.3 Node.js

File ini berisi kelas Node yang merepresentasikan sebuah node yang ada pada graph

```
src > Node.js > ...
You, 2 minutes ago | 2 authors (Muhammad Fadhil Amri and others)
1 // Class yang merepresentasikan node pada graph
2 // Attribute:
3 // - name: nama node
4 // - heuristic: nilai heuristik dari node
5 // - neighbour: array yang berisi node tetangga dan jaraknya
6 // - visited: boolean yang menandakan apakah node sudah dikunjungi atau belum
7 // - position: koordinat node
8
9 const Coordinate = require('./Coordinate.js').coordinate;
10
You, 14 hours ago | 2 authors (Muhammad Fadhil Amri and others)
11 class Node {
12     // ctore
13     constructor(name, heuristic) {
14         this.name = name;
15         this.heuristic = heuristic;
16         this.neighbour = [];
17         this.visited = false;
18         this.position = new Coordinate(0, 0);
19     }
20
21     // getter
22     getName() { return this.name; }
23     getHeuristic() { return this.heuristic; }
24     isVisited() { return this.visited; }
25     getNeighbours() {
26         return this.neighbour;
27     }
28
29     getNeighbourNode(i) {
30         return this.neighbour[i][0];
31     }
32
33     getNeighbourNodeByName(nodeName) {
34         let i = 0;
35         let found = false;
36         while ((i < this.neighbour.length) && (!found)) {
37             if (this.neighbour[i][0].getName() == nodeName) {
38                 found = true;
39             } else {
40                 i++;
41             }
42         }
43         if (found) {
44             return this.neighbour[i][0];
45         } else {
46             return null;
47         }
48     }
49
50     getNeighbourDistance(i) {
51         return this.neighbour[i][1];
52     }
53
54     getNeighbourDistanceByName(nodeName) {
55         let i = 0;
56         let found = false;
57         while ((i < this.neighbour.length) && (!found)) {
58             if (this.neighbour[i][0].getName() == nodeName) {
59                 found = true;
60             } else {
61                 i++;
62             }
63         }
64         if (found) {
65             return this.neighbour[i][1];
66         } else {
67             return null;
68         }
69     }
```

```

71     getNeighbourCount() {
72         return this.neighbour.length;
73     }
74
75     getPosition() {
76         return this.position;
77     }
78
79     // Setter
80     setVisited() {
81         this.visited = true;
82     }
83
84     setHeuristic(heuristic) {
85         this.heuristic = heuristic;
86     }
87
88 +    setPosition(x, y) {
89         this.position.setX(x);
90         this.position.setY(y);
91     }
92
93     resetVisited() {
94         this.visited = false;
95     }
96
97     // Method untuk menambahkan tetangga
98     addNeighbour(node, distance) {
99         if (this.neighbour.length == 0) {
100             this.neighbour.push([node, distance]);
101         } else {
102             // Lakukan insertion sort, terurut menaik berdasarkan (jarak + heuristik)
103             let i = 0;
104             let found = false;
105             while ((i < this.neighbour.length) && (!found)) {
106                 if ((this.neighbour[i][1] + this.neighbour[i][0].getHeuristic()) > (distance + node.getHeuristic())) {
107                     this.neighbour.splice(i, 0, [node, distance]);
108
109                     found = true;
110                 } else {
111                     i++;
112                 }
113             }
114             if (!found) {
115                 // insert di akhir
116                 this.neighbour.push([node, distance]);
117             }
118         }
119
120
121     // Method untuk mengecek apakah node memiliki tetangga dengan nama tertentu
122     isNeighbour(nodeNames) {
123         if (this.neighbour.length == 0) {
124             return false;
125         } else {
126             let i = 0;
127             let found = false;
128             while ((i < this.neighbour.length) && (!found)) {
129                 if (this.neighbour[i][0].getName() == nodeNames) {
130                     found = true;
131                 } else {
132                     i++;
133                 }
134             }
135         }
136         return found;
137     }
138 }
139
140
141 module.exports = { Node };

```

3.4 Route.js

File ini berisi kelas Route yang merepresentasikan rute hingga currentNode (node yang akan diperiksa pada saat expand)

```
src > Route.js > ...
  ...
1 // Description: class Route
2 // Attribute:
3 // - currentNode: node yang sedang ditempati/ expandNode
4 // - path: array yang berisi nodeName yang sudah dilewati
5 // - cost: nilai cost dari rute yang sudah dilewati menuju currentNode
6 > Muhammad Fadhil Amri, 3 days ago | 1 author (Muhammad Fadhil Amri)
7 class Route {
8     constructor(node, path, cost) {
9         this.currentNode = node;
10        this.path = path;
11        this.cost = cost;
12    }
13    getCurrentNode() { return this.currentNode; }
14    getPath() { return this.path; }
15    getCost() { return this.cost; }
16    addPath(nodeName) {
17        this.path.push(nodeName);
18    }
19    addCost(cost) {
20        this.cost += cost;
21    }
22    isPathExist(nodeName) {
23        let i = 0;
24        let found = false;
25        while ((i < this.path.length) && (!found)) {
26            if (this.path[i] == nodeName) {
27                found = true;
28            } else {
29                i++;
30            }
31        }
32        return found;
33    }
34 }
35 > module.exports = { Route };
36
```

3.5 PrioQueue.js

File ini berisi kelas PrioQueue yang merupakan sebuah struktur data queue dengan urutan berdasarkan suatu prioritas, pada persoalan ini prioritas yang digunakan adalah cost dari Route.

```

src > PrioQueue.js > ...
You, 21 seconds ago | 2 authors (Muhammad Fadhil Amri and others)
1 // class PrioQueue adalah class yang digunakan untuk membuat queue dengan prioritas
2 // berdasarkan cost yang dimiliki, terurut menaik
3 // Attribute:
4 // - items: object yang berisi item-item yang ada di queue => bertipe Route
5 // - headIndex: index dari item yang paling depan
6

You, 21 seconds ago | 2 authors (Muhammad Fadhil Amri and others)
7 class PrioQueue {
8     // ctor
9     constructor() {
10         this.items = {};
11         this.headIndex = 0;
12         this.tailIndex = 0;
13     }
14
15     // Method
16 + enqueue(item) {
17     // Jika kosong, langsung masukkan route pada queue
18     if (this.isEmpty()) {
19         this.items[this.tailIndex] = item;
20         this.tailIndex++;
21     } else {
22         // Jika tidak kosong Lakukan insertion sort, terurut menaik berdasarkan cost
23         let i = this.headIndex;
24         let found = false;
25         while ((i < this.tailIndex) && (!found)) {
26             if ((this.items[i].getCost() + this.items[i].getCurrentNode().getHeuristic())
27                 > (item.getCost() + item.getCurrentNode().getHeuristic())) {
28                 found = true;
29             } else {
30                 i++;
31             }
32         }
33         if (!found) {
34             // Masukkan item pada index tailIndex
35             this.items[this.tailIndex] = item;
36
37             } else {
38                 // Masukkan item pada index i
39                 for (let j = i; j < this.tailIndex; j++) {
40                     this.items[j + 1] = this.items[j];
41                 }
42                 this.items[i] = item;
43             }
44             this.tailIndex++;
45         }
46     dequeue() {
47         const item = this.items[this.headIndex];
48         delete this.items[this.headIndex];
49         this.headIndex++;
50         return item;
51     }
52     peek() {
53         return this.items[this.headIndex];
54     }
55
56     // isEmpty function
57     isEmpty() {
58         // return true if the queue is empty.
59         return this.items.length == 0;
60     }
61     print() {
62         let str = "";
63         for (let i = this.headIndex; i < this.tailIndex; i++) {
64             str += this.items[i].getCurrentNode().getName() + ", " + (this.items[i].getCost()
65             + this.items[i].getCurrentNode().getHeuristic()).toString() + "; ";
66         }
67         console.log(str);
68     }
69 }
70
71 module.exports = { PrioQueue };

```

3.6 IO.js

File ini berisi fungsi-fungsi untuk pemrosesan input/output informasi dari eksternal ke internal program dan sebaliknya (*barrier*)

```
src > JS IO.js > ...
You, 15 seconds ago | 2 authors (Muhammad Fadhil Amri and others)
1 const fs = require('fs');
2 const Node = require('./Node.js').Node;
3 const Coordinate = require('./Coordinate.js').Coordinate;
4 /*
5   FORMAT FILE INPUT
6
7   Terdapat dummy elemen baris 0 kolom 0 yang berisi string bernama "simpul"
8   Nama simpul terletak di kolom pertama dan baris pertama
9   Nilai jarak antar simpul terletak di kolom ke-i dan baris ke-j, i != 0 dan j != 0
10  setiap elemen matriks dalam satu baris dipisahkan dengan koma
11  setiap baris dipisahkan oleh newLine (\r\n)
12  baris terakhir berisi koordinat dari setiap simpul
13  format koordinat adalah (x,y) dengan x dan y merupakan bilangan bulat, setiap koordinat simpul dipisahkan dengan ;
14  urutan koordinat sesuai urutan nama simpul
15
16  Edge yang tidak ada ditandai dengan nilai 0
17
18  Contoh:
19  Simpul, Padang, Jakarta, Bandung, Surabaya, Medan
20  Padang, 0, 1000, 2000, 3000, 4000
21  Jakarta, 1000, 0, 1000, 2000, 3000
22  Bandung, 2000, 1000, 0, 1000, 2000
23  Surabaya, 3000, 2000, 1000, 0, 1000
24  Medan, 4000, 3000, 2000, 1000, 0
25  (0,0);(1000,0);(2000,0);(3000,0);(4000,0)
26
27 */
28
29 // Fungsi untuk membaca file txt input yang berbentuk matriks ketetanggan dan mengembalikan matrix of string
30 function readInputFile(fileName) {
31   let matrix = [];
32   let text = fs.readFileSync('../test/' + fileName, { encoding: 'utf-8', flag: 'r' });
33
34   // Ubah text menjadi array of lines
35   let lines = text.split("\r\n");
36 }
```

```

38 // Ubah array of lines menjadi matrix of string
39 + for (let i = 0; i < (lines.length - 1); i++) {
40     let arrElemen = lines[i].split(",");
41     arrElemen = arrElemen.map(elemen => { return elemen.trim() });
42     matrix.push(arrElemen);
43 }
44 √ if (lines.length > 0) {
45     matrix.push(lines[lines.length - 1].split(";"));
46 }
47
48 return matrix;
49 }
50
51
52 /*
53  * Fungsi untuk membaca matrice of string dan mengembalikan informasi dari map berupa matrices of float,
54  * array of coordinates, dan array of string
55  */
56
57 Fungsi ini juga akan mengecek apakah matriks yang dibaca adalah matriks yang valid
58 Jika matriks tidak valid, maka akan melakukan throw error
59 Jika matriks valid, maka akan mengembalikan matrices of float dan array of string
60 Array of Coordinate merupakan array yang berisi koordinat dari setiap simpul
61 Array of string merupakan array yang berisi nama dari setiap simpul
62 Matrices of float merupakan matriks yang berisi nilai jarak dari setiap simpul ke setiap simpul lainnya,
63 index sesuai index nama simpul
64 */
65 function generateMapInfo(matrix) {
66     if (matrix.length == 0) {
67         throw "Matrix is empty";
68     } else if (matrix.length != (matrix[0].length + 1)) {
69         throw "Matrix doesn't have valid dimension";
70     } else {
71         // Inisialisasi variabel
72         let mapInfo = [];
73         let nodeNames = [];
74         let adjacencyMatrix = [];

```

```

75             let coordinates = [];
76
77             // Membaca nama simpul dari matriks
78             for (let i = 1; i < matrix[0].length; i++) {
79                 nodeNames.push(matrix[i][0]);
80             }
81
82             // Membaca matriks ketetanggan berbobot
83             for (let i = 1; i < (matrix.length - 1); i++) {
84                 let row = [];
85                 for (let j = 1; j < matrix[i].length; j++) {
86                     row.push(parseFloat(matrix[i][j]));
87                 }
88                 adjacencyMatrix.push(row);
89             }
90
91             // Membaca koordinat dari setiap simpul
92             let lastRow = matrix[matrix.length - 1];
93             for (let i = 0; i < lastRow.length; i++) {
94                 let coordinate = lastRow[i].split(",");
95                 let x = parseInt(coordinate[0].substring(1));
96                 let y = parseInt(coordinate[1].substring(0, coordinate[1].length - 1));
97                 coordinates.push(new Coordinate(x, y));
98             }
99
100            // Memasukkan nama simpul dan matriks ketetanggan ke dalam mapInfo
101            mapInfo.push(nodeNames);
102            mapInfo.push(adjacencyMatrix);
103            mapInfo.push(coordinates);
104
105        return mapInfo;
106    }
107
108    /*
109     * Fungsi untuk mengecek apakah node dengan nama nodeName sudah ada di dalam graph
110     */

```

```

111  ✓function positionInGraph(graph, nodeName) {
112    ✓  for (let i = 0; i < graph.length; i++) {
113      ✓    if (graph[i].getName() == nodeName) {
114        ✓      return i;
115      }
116    }
117    ✓  return -1;
118  }
119
120
121  /* 
122   Fungsi untuk menghasilkan graf dari mapInfo
123   graf yang dihasilkan berupa list of node dengan nilai heuristic θ => general bisa digunakan untuk semua algoritma
124 */
125
126  ✓function generateGraph(mapInfo) {
127    let graph = [];
128    let nodeNames = mapInfo[0];
129    let adjacencyMatrix = mapInfo[1];
130    let coordinates = mapInfo[2];
131
132    // Membuat node untuk setiap simpul
133    ✓for (let i = 0; i < nodeNames.length; i++) {
134      let node = new Node(nodeNames[i], 0);
135
136      // Menambahkan tetangga ke node
137      ✓for (let j = 0; j < adjacencyMatrix[i].length; j++) {
138        ✓  if (adjacencyMatrix[i][j] != 0) {
139          ✓    // Jika node dengan nama nodeNames[j] sudah ada di dalam graph,
140          ✓    // maka tambahkan nodeNames[j] sebagai tetangga node nodenames[i]
141          ✓    let position = positionInGraph(graph, nodeNames[j]);
142          ✓    if (position != -1) {
143              ✓      node.addNeighbour(graph[position], adjacencyMatrix[i][j]);
144              ✓      // Jika node i belum menjadi tetangga nodeNames[j], maka tambahkan nodeNames[i] sebagai tetangga
145              ✓      if (!graph[position].isNeighbour(nodeNames[i])) {
146                  ✓        graph[position].addNeighbour(node, adjacencyMatrix[i][j]);
147
148                  ✓      }
149                }
150    +  }
151    ✓  graph.push(node);
152  }
153
154  // Menambahkan koordinat ke setiap node
155  ✓for (let i = 0; i < graph.length; i++) {
156    ✓  graph[i].setPosition(coordinates[i].getX(), coordinates[i].getY());
157  }
158
159  return graph;
160}
161
162 module.exports = { readInputFile, generateMapInfo, generateGraph, positionInGraph };

```

3.7 AStar.js

File ini berisi fungsi-fungsi untuk menjalankan algoritma A* dalam melakukan pencarian rute dari node awal menuju node tujuan.

```

src > js AStar.js > ...
You, 3 seconds ago | 2 authors (Muhammad Fadhil Amri and others)
1 const PrioQueue = require('./PrioQueue.js').PrioQueue;
2 const Route = require('./Route.js').Route;
3
4 /*
5   Fungsi untuk set nilai heuristic untuk setiap node di dalam graph, sesuai goal.
6   Nilai heuristic diperoleh dari euclidean distance dari node ke goal
7 */
8 function setGraphHeuristic(graph, goal) {
9   // Set heuristic untuk setiap node
10  for (let i = 0; i < graph.length; i++) {
11    let heuristicValue = 0;
12    // Jika bukan node bukan goal, maka hitung euclidean distance dan set sebagai heuristic value
13    if (graph[i].getName() != goal.getName()) {
14      heuristicValue = Math.sqrt(Math.pow(graph[i].getPosition().getX() - goal.getPosition().getX(), 2) +
15        Math.pow(graph[i].getPosition().getY() - goal.getPosition().getY(), 2));
16    }
17    graph[i].setHeuristic(heuristicValue);
18  }
19}
20
21 /*
22   Fungsi untuk mengecek apakah pruning diperlukan
23 */
24 function isNeedPruning(currentRoute, newNode, closed_list) {
25   // Jika closed_list kosong, maka pruning tidak diperlukan
26   if (closed_list.length == 0) {
27     return false;
28   }
29   // Jika closed_list tidak kosong, maka cek apakah ada route yang memiliki cost lebih besar dari currentRoute
30   let isPruningNeeded = false;
31   for (let i = 0; i < closed_list.length; i++) {
32     if (closed_list[i].getCurrentNode().getName() == newNode.getName()) {
33       let currentCost = currentRoute.getCost() + newNode.getHeuristic() +
34         currentRoute.getCurrentNode().getNeighbourDistanceByName(newNode.getName());
35
36       if (currentCost > closed_list[i].getCost()) {
37         isPruningNeeded = true;
38       }
39       break;
40     }
41   }
42   return isPruningNeeded;
43 }
44
45 /*
46   prekondisi: graph sudah memiliki nilai heuristic
47   Fungsi untuk menghasilkan jalur dari start ke goal menggunakan algoritma A*
48   fungsi mengembalikan Route object saat sampai ke goal
49
50 */
51 function runAStarAlgorithm(startNode, goal) {
52   // Inisialisasi open_list, closed_list dan Route
53   let open_list = new PrioQueue();
54   let closed_list = [];
55   let route = new Route(startNode, [startNode.getName()], 0);
56
57   // Masukkan route ke dalam open_list
58   open_list.enqueue(route);
59
60   // Lakukan perulangan sampai open_list kosong atau sampai goal ditemukan
61   while (!open_list.isEmpty()) {
62     // Ambil route dengan cost terkecil
63     let currentRoute = open_list.dequeue();
64     closed_list.push(currentRoute);
65     let currentNode = currentRoute.getCurrentNode();
66     currentNode.setVisited();
67
68     // Jika currentNode adalah goal, maka return currentRoute
69     if (currentNode.getName() == goal) {
70       return currentRoute;
71     }

```

```

72     // Jika currentNode bukan goal, maka expand currentNode
73     let neighbours = currentNode.getNeighbours();
74     for (let i = 0; i < neighbours.length; i++) {
75       let neighbour = neighbours[i][0];
76       let neighbourName = neighbour.getName();
77
78       // Jika neighbour belum pernah dilewati
79       if (!currentRoute.isPathExist(neighbourName)) {
80
81         // Jika neighbour belum pernah dikunjungi atau pruning tidak diperlukan
82         if (!isNeedPruning(currentRoute, neighbour, closed_list)) {
83           let newPath = JSON.parse(JSON.stringify(currentRoute.getPath()));
84           let newRoute = new Route(neighbour, newPath, currentRoute.getCost());
85           newRoute.addPath(neighbourName);
86           newRoute.addCost(neighbour.getNeighbourDistanceByName(currentNode.getName()));
87           open_list.enqueue(newRoute, newRoute.getCost());
88         }
89       }
90     }
91   }
92   return route;
93 }
94
95 module.exports = { runStarAlgorithm, setGraphHeuristic };

```

3.8 UCS.js

File ini berisi fungsi untuk menjalankan algoritma UCS dalam melakukan pencarian rute dari node awal menuju node tujuan.

```

1 You, 2 seconds ago | 2 authors (You and others)
2 const PrioQueue = require('./PrioQueue.js').PrioQueue;
3 const Route = require('./Route.js').Route;
4
5 /**
6  * Fungsi untuk menghasilkan jalur dari start ke goal menggunakan algoritma ucs
7  * fungsi mengembalikan Route object saat sampai ke goal,
8  * jika tidak ditemukan maka akan mengembalikan route dengan path kosong
9 */
10 function runUCSAlgorithm(startNode, goal) {
11   // Inisialisasi prioQueue dan Route
12   let prioQueue = new PrioQueue();
13   let route = new Route(startNode, [startNode.getName()], 0);
14   // Masukkan route awal ke dalam prioQueue
15   prioQueue.enqueue(route);
16
17   // Lakukan perulangan sampai prioQueue kosong
18   while (!prioQueue.isEmpty()) {
19     // Ambil route dengan cost terkecil
20     let currentRoute = prioQueue.dequeue();
21     let currentNode = currentRoute.getCurrentNode();
22     currentNode.setVisited();
23
24     // Jika currentNode adalah goal, maka return currentRoute
25     if (currentNode.getName() == goal) {
26       return currentRoute;
27     }
28
29     // Jika currentNode bukan goal, maka masukkan semua neighbour yang belum dikunjungi ke dalam prioQueue
30     let neighbours = currentNode.getNeighbours();
31     for (let i = 0; i < neighbours.length; i++) {
32
33       let neighbour = neighbours[i][0];
34       let neighbourName = neighbour.getName();

```

```

36     if (!neighbour.isVisited()) {
37         let newPath = JSON.parse(JSON.stringify(currentRoute.getPath()));
38         let newRoute = new Route(neighbour, newPath, currentRoute.getCost());
39         newRoute.addPath(neighbourName);
40         newRoute.addCost(neighbour.getNeighbourDistanceByName(currentNode.getName()));
41         prioQueue.enqueue(newRoute, newRoute.getCost());
42     }
43 }
44 return route;
45
46 }
47
48 module.exports = { runUCSAlgorithm };

```

3.9 Map.js

File ini berisi fungsionalitas dari tampilan pada web.

```

src > Map.js > routeColoring > color
1 import { runAStarAlgorithm, setGraphHeuristic } from './AStar.js';
2 import { runUCSAlgorithm } from './UCS.js';
3 import { generateGraph, generateMapInfo } from './IO.js';
4 import { matrixData } from './IO.js';
5
6 export var map = L.map('map').setView([-6.89140361736227, 107.61032928111902], 17);
7
8 L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
9     maxZoom: 19,
10    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
11 }).addTo(map);
12
13 export var markers = [];
14 export var paths = [];
15 export var pair = [];
16 export var algo = "UCS";
17 export var mode = "Add";
18 export var idMarker = 0;
19
20 export function toggleUCS(e) {
21     algo = "UCS";
22     console.log(algo);
23 }
24 export function toggleAStar(e) {
25     algo = "A*";
26     console.log(algo);
27 }
28 export function toggleAdd(e) {
29     mode = "Add";
30     pair.length = 0;
31     console.log(mode);
32 }
33 export function togglePath(e) {
34     mode = "Path";
35     console.log(mode);
36 }
37 export function toggleDelete(e) {
38     mode = "Delete";
39     pair.length = 0;
40     console.log(mode);
41 }
42 export function toggleSearch() {
43     mode = "Search";
44     pair.length = 0;
45     console.log(mode);
46 }

```

```

41     routeColoring();
42 }
43 export function toggleSearch() {
44     mode = "Search";
45     pair.length = 0;
46     console.log(mode);
47 }
48 export function addMarker(lat, lng, id = idMarker) {
49     var marker = L.marker([lat, lng], {
50         name: id.toString(),
51         title: id.toString()
52     })
53     marker.addTo(map);
54     marker.on('click', onMarkerClick);
55     markers.push(marker);
56     idMarker++;
57 }
58
59 export function removeMarker(target) {
60     for (var i = 0; i < markers.length; i++) {
61         if (target == markers[i]) {
62             markers.splice(i, 1);
63             map.removeLayer(target);
64         }
65     }
66 }
67
68
69 export function addPath(marker1, marker2) {
70     var polyline = L.polyline([marker1.getLatlng(), marker2.getLatlng()], { color: 'yellow' }).addTo(map);
71     polyline.on('click', onLineClick);
72     var length = (marker1.getLatlng()).distanceTo(marker2.getLatlng());
73     var center = polyline.getCenter();
74     // Create a DivIcon with text content
75     var textIcon = L.divIcon({
76         className: 'text-icon',
77         html: length.toFixed(2),
78     });
79     // Create a marker with the text icon and add it to the map at the polyline's center
80     var lengthDisplay = L.marker(center, { icon: textIcon }).addTo(map);
81
82     paths.push({ 'markers': [marker1, marker2], 'length': length.toFixed(2), 'line': polyline, 'lengthDisplay': lengthDisplay })
83 }
84
85 export function removePath(target) {
86     for (var i = 0; i < paths.length; i++) {

```

```

src > js Mapjs > ↗ routeColoring > ↗ color
84
85 export function removePath(target) {
86     for (var i = 0; i < paths.length; i++) {
87         if (target == paths[i].line) {
88             // console.log("deleted");
89             map.removeLayer(target);
90             map.removeLayer(paths[i].lengthDisplay);
91             paths.splice(i, 1);
92             break;
93         }
94     }
95 }
96
97 export function onMapClick(e) {
98     if (mode == "Add") {
99         addMarker(e.latlng.lat, e.latlng.lng, idMarker);
100    }
101 }
102
103 export function onMarkerClick(e) {
104     if (mode == "Path") {
105
106         if (pair.length == 0) {
107             pair.push(e.target);
108         } else if (pair.length == 1) {
109             if (e.target != pair[0]) {
110                 pair.push(e.target);
111             }
112         }
113
114         if (pair.length == 2) {
115             addPath(pair[0], pair[1]);
116             pair.length = 0;
117         }
118     } else if (mode == "Delete") {
119         removeMarker(e.target);
120     } else if (mode == "Search") {
121         if (pair.length == 0) {
122             pair.push(e.target);
123         } else if (pair.length == 1) {
124             if (e.target != pair[0]) {
125                 pair.push(e.target);
126                 routeColoring();
127             }
128         }

```

```

128     }
129   }
130 }
131 }
132
133 export function onLineClick(e) {
134   if (mode == "Delete") {
135     removePath(e.target);
136   }
137 }
138
139 export function resetMapState() {
140   markers.length = 0;
141   paths.length = 0;
142   pair.length = 0;
143   idMarker = 0;
144   map.eachLayer(function (layer) {
145     if (!(layer instanceof L.TileLayer)) {
146       map.removeLayer(layer);
147     }
148   });
149 }
150
151 export function generateMapMatrix() {
152   var matrix = []
153   var names = []
154   const hashmapIdx = new Map();
155   // console.log("Start mapping")
156   for (var i = 0; i <= markers.length; i++) {
157     if (i == 0) {
158       names.push("Simpul");
159     } else {
160       names.push(markers[i - 1].options.name);
161       hashmapIdx.set(markers[i - 1].options.name, i);
162     }
163   }
164   // console.log(names);
165   // console.log(markers);
166   // console.log(paths);
167   // console.log(matrix);
168
169   // Initialize matrix
170   for (var i = 0; i < names.length; i++) {
171     matrix.push(names.slice());
172     if (i != 0) {
173       for (var j = 0; j < names.length; j++) {
174         if (j == 0) {
175           matrix[i][0] = matrix[0][i];
176         } else {
177           matrix[i][j] = '0';
178         }
179       }
180     }
181   }
182   // console.log(matrix);
183
184   for (var i = 0; i < paths.length; i++) {
185     var idxMarker1 = hashmapIdx.get(paths[i].markers[0].options.name);
186     var idxMarker2 = hashmapIdx.get(paths[i].markers[1].options.name);
187     console.log(idxMarker1, idxMarker2);
188     matrix[idxMarker1][idxMarker2] = paths[i].length.toString();
189     matrix[idxMarker2][idxMarker1] = paths[i].length.toString();
190   }
191
192   // console.log(matrix);
193   var coordinates = []
194   for (var i = 0; i < markers.length; i++) {
195     coordinates.push(`(${markers[i].getLatLang().lat},${markers[i].getLatLang().lng})`);
196   }
197   matrix.push(coordinates.slice());
198   // console.log(matrix);
199   return matrix;
200 }
201
202 export function saveFileMap() {
203   var matrix = generateMapMatrix();
204   var text = "";
205   for (var i = 0; i < matrix.length; i++) {
206     var first = true;
207     for (var el of matrix[i]) {
208       if (first) {
209         first = false;
210       } else if (i != matrix.length - 1) {
211         text += ", ";
212       } else {
213         text += ",";
214       }
215     }

```

```

170     for (var i = 0; i < names.length; i++) {
171       matrix.push(names.slice());
172       if (i != 0) {
173         for (var j = 0; j < names.length; j++) {
174           if (j == 0) {
175             matrix[i][0] = matrix[0][i];
176           } else {
177             matrix[i][j] = '0';
178           }
179         }
180       }
181     }
182     // console.log(matrix);
183
184     for (var i = 0; i < paths.length; i++) {
185       var idxMarker1 = hashmapIdx.get(paths[i].markers[0].options.name);
186       var idxMarker2 = hashmapIdx.get(paths[i].markers[1].options.name);
187       console.log(idxMarker1, idxMarker2);
188       matrix[idxMarker1][idxMarker2] = paths[i].length.toString();
189       matrix[idxMarker2][idxMarker1] = paths[i].length.toString();
190     }
191
192     // console.log(matrix);
193     var coordinates = []
194     for (var i = 0; i < markers.length; i++) {
195       coordinates.push(`(${markers[i].getLatLang().lat},${markers[i].getLatLang().lng})`);
196     }
197     matrix.push(coordinates.slice());
198     // console.log(matrix);
199     return matrix;
200   }
201
202 export function saveFileMap() {
203   var matrix = generateMapMatrix();
204   var text = "";
205   for (var i = 0; i < matrix.length; i++) {
206     var first = true;
207     for (var el of matrix[i]) {
208       if (first) {
209         first = false;
210       } else if (i != matrix.length - 1) {
211         text += ", ";
212       } else {
213         text += ",";
214       }
215     }

```

```
src > js Map.js > routeColoring > color
215     text += el;
216 }
217 if (i != matrix.length - 1) {
218     text += "\r\n";
219 }
220 }
221 var filename = "mapfile.txt";
222 var blob = new Blob([text], { type: "text/plain;charset=utf-8" });
223 var url = window.URL.createObjectURL(blob);
224 var element = document.createElement('a');
225 element.setAttribute('href', url);
226 element.setAttribute('download', filename);
227 document.body.appendChild(element);
228 element.click();
229 document.body.removeChild(element);
230 window.URL.revokeObjectURL(url);
231 }
232
233 export function plotToMap() {
234     // console.log(matrixData);
235     resetMapState();
236     var mapInfo = generateMapInfo(matrixData);
237     var nodeNames = mapInfo[0];
238     var adjacencyMatrix = mapInfo[1];
239     var coordinates = mapInfo[2];
240     for (var i = 0; i < nodeNames.length; i++) {
241         addMarker(coordinates[i].getX(), coordinates[i].getY(), nodeNames[i]);
242     }
243     console.log(nodeNames);
244     console.log(adjacencyMatrix);
245     for (var i = 0; i < nodeNames.length; i++) {
246         for (var j = i + 1; j < nodeNames.length; j++) {
247             if (adjacencyMatrix[i][j] != 0) {
248                 for (var el of markers) {
249                     if (el.options.name == nodeNames[i] || el.options.name == nodeNames[j]) {
250                         pair.push(el);
251                     }
252                 }
253                 addPath(pair[0], pair[1]);
254                 pair.length = 0;
255             }
256         }
257     }
258 }
259 }
260 }
```

```

258 }
259 }
260 export function routeColoring() {
261     resetColoring();
262     var matrixMap = generateMapMatrix();
263     var mapInfo = generateMapInfo(matrixMap);
264     var graph = generateGraph(mapInfo);
265     console.log(mapInfo);
266     for (var i = 0; i < graph.length; i++) {
267         if (pair[0].options.name == graph[i].getName()) {
268             var startNode = graph[i];
269         }
270     }
271     for (var i = 0; i < graph.length; i++) {
272         if (pair[1].options.name == graph[i].getName()) {
273             var goalNode = graph[i];
274         }
275     }
276 }
277 if (algo == "A*") {
278     setGraphHeuristic(graph, goalNode);
279     var route = runAStarAlgorithm(startNode, goalNode.getName());
280 } else if (algo == "UCS") {
281     var route = runUCSAlgorithm(startNode, goalNode.getName());
282 }
283
284 if (route.path.length <= 1) {
285     alert("No path found.");
286 } else {
287     console.log(route);
288     console.log(paths);
289     for (var i = 0; i < route.path.length - 1; i++) {
290         for (var p of paths) {
291             if ((route.path[i] == p.markers[0].options.name && route.path[i + 1] == p.markers[1].options.name) ||
292                 (route.path[i] == p.markers[1].options.name && route.path[i + 1] == p.markers[0].options.name)) {
293                 p.line.setStyle({ color: 'red' });
294             }
295         }
296     }
297     var popup = L.popup()
298         .setLatLng(pair[1].getLatLng())
299         .setContent(`Distance from node ${startNode.getName()} to node ${goalNode.getName()} = ${route.cost} meters.`)
300         .openOn(map);
301     }
302 }
303

```

```

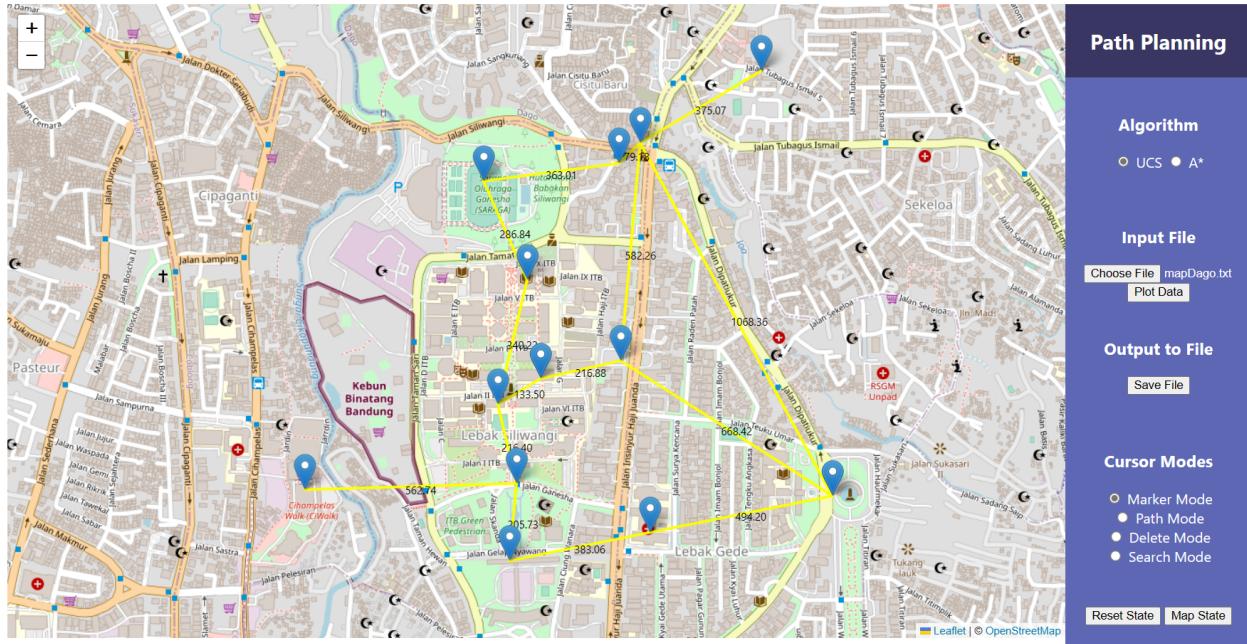
296     }
297 }
298 var popup = L.popup()
299     .setLatLng(pair[1].getLatLng())
300     .setContent(`Distance from node ${startNode.getName()} to node ${goalNode.getName()} = ${route.cost} meters.`)
301     .openOn(map);
302 }
303
304 }
305
306 export function resetColoring() {
307     for (var p of paths) {
308         p.line.setStyle({ color: 'yellow' });
309     }
310 }
311
312 map.on('click', onMapClick);
313
314 // export { toggleUCS, toggleAStar, toggleAdd, togglePath, toggleDelete, toggleSearch, addMarker, removeMarker, addPath, removePath, onMapClick, onMarkerClick, on
315

```

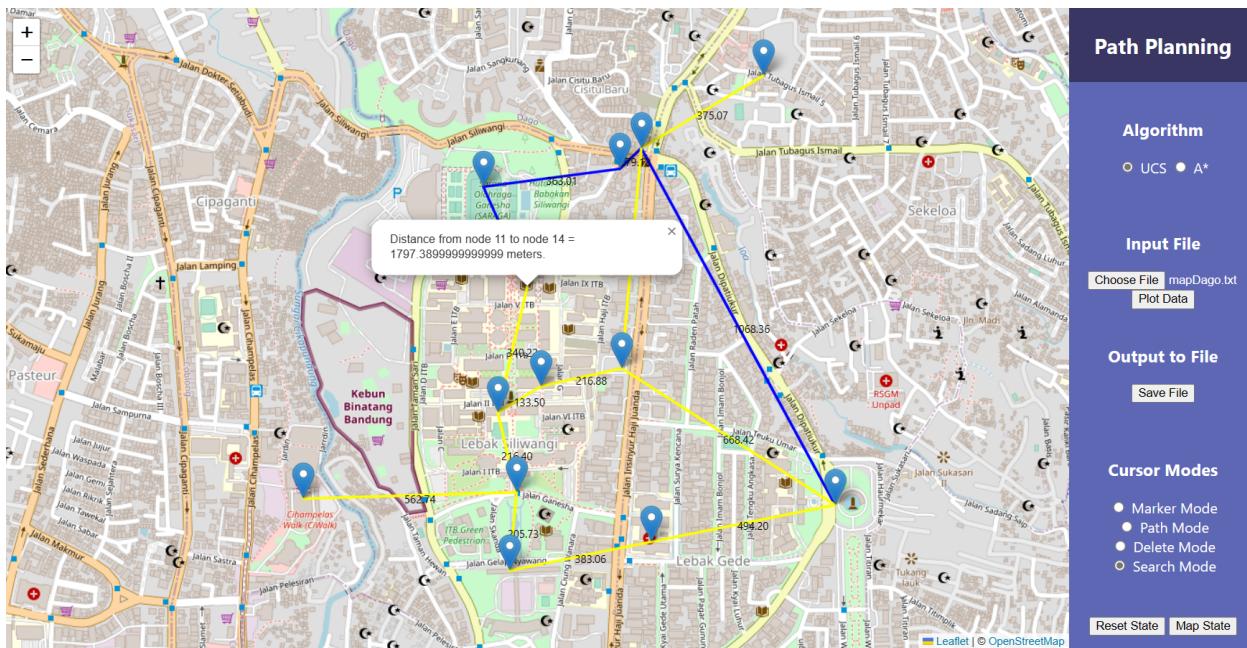
Bab IV: Contoh Masukan dan Keluaran

4.1 Map Sekitar ITB/ Bandung Utara/ Dago

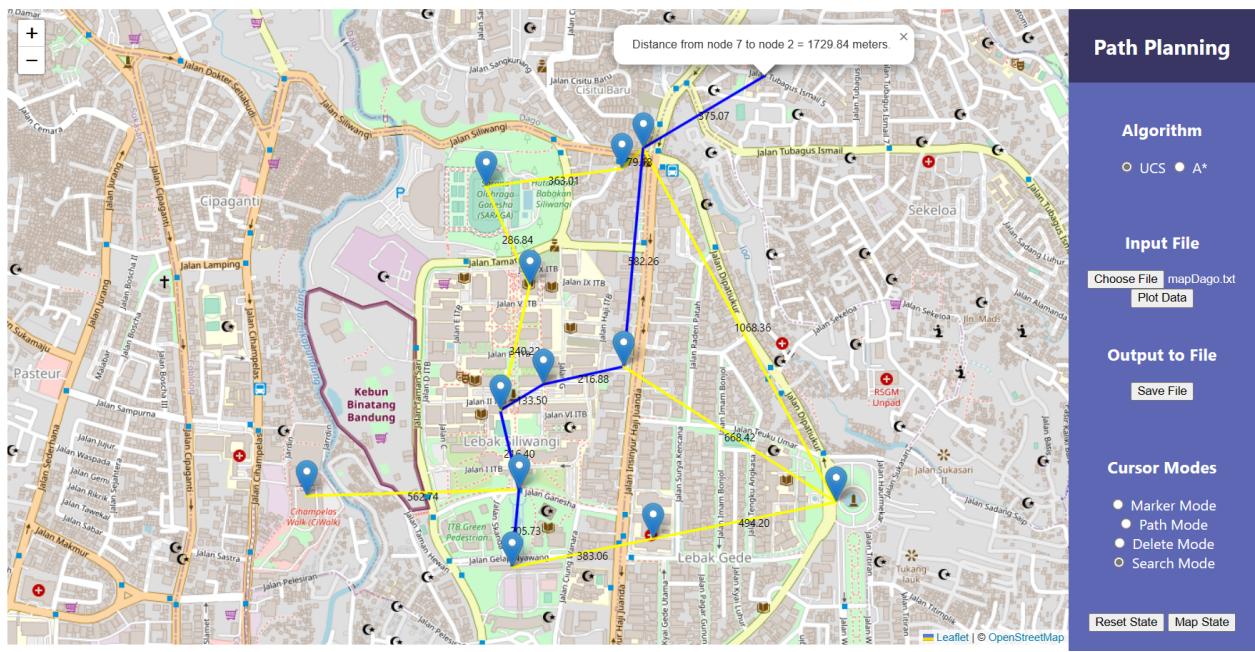
4.1.1 Input



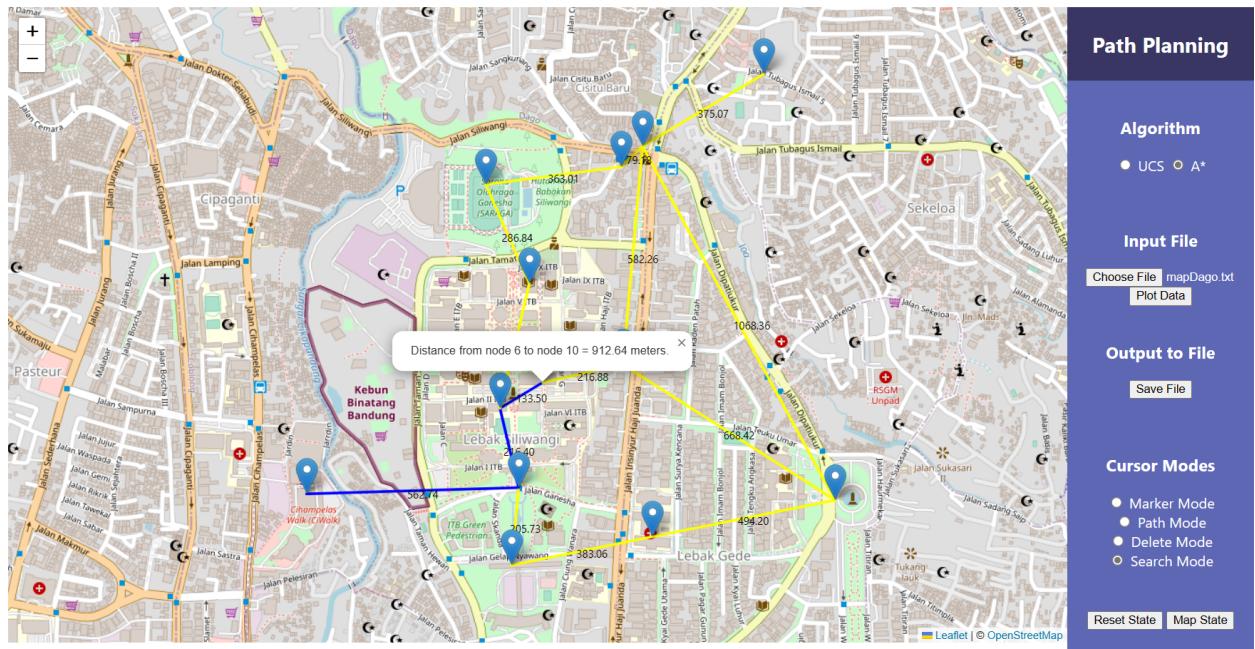
4.1.2 Start: Unpad, Goal: Perpustakaan Pusat ITB (UCS Algorithm)



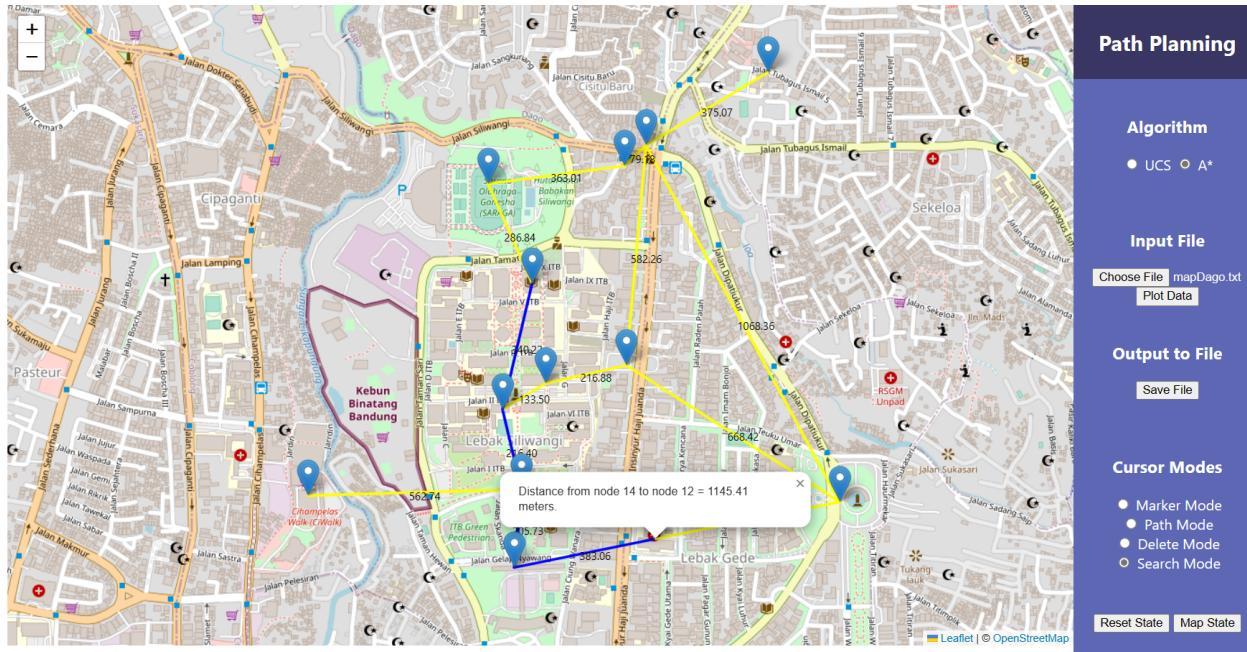
4.1.3 Start: Kantin Gelap Nyawang , Goal: Kostan Hanif (UCS Algorithm)



4.1.4 Start: Ciwalk, Goal: Labtek VIII (A* Algorithm)

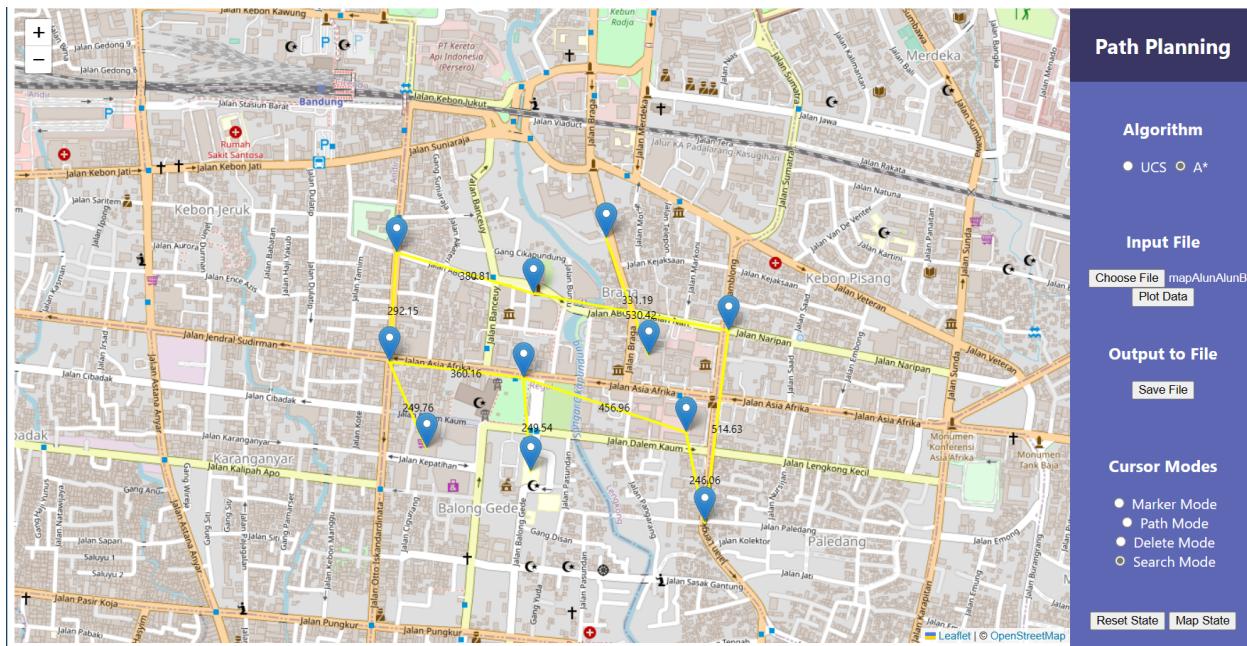


4.1.5 Start: Perpustakaan ITB, Goal: RS Borromeus (A* Algorithm)

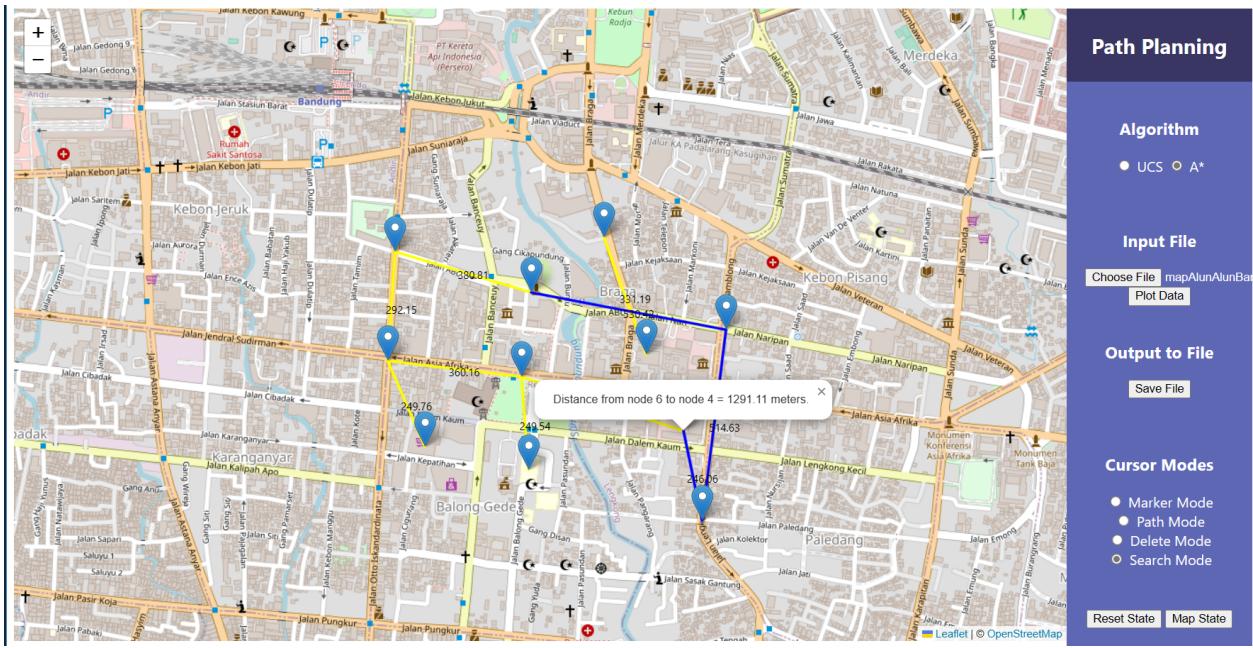


4.2 Map Sekitar Alun-Alun Bandung

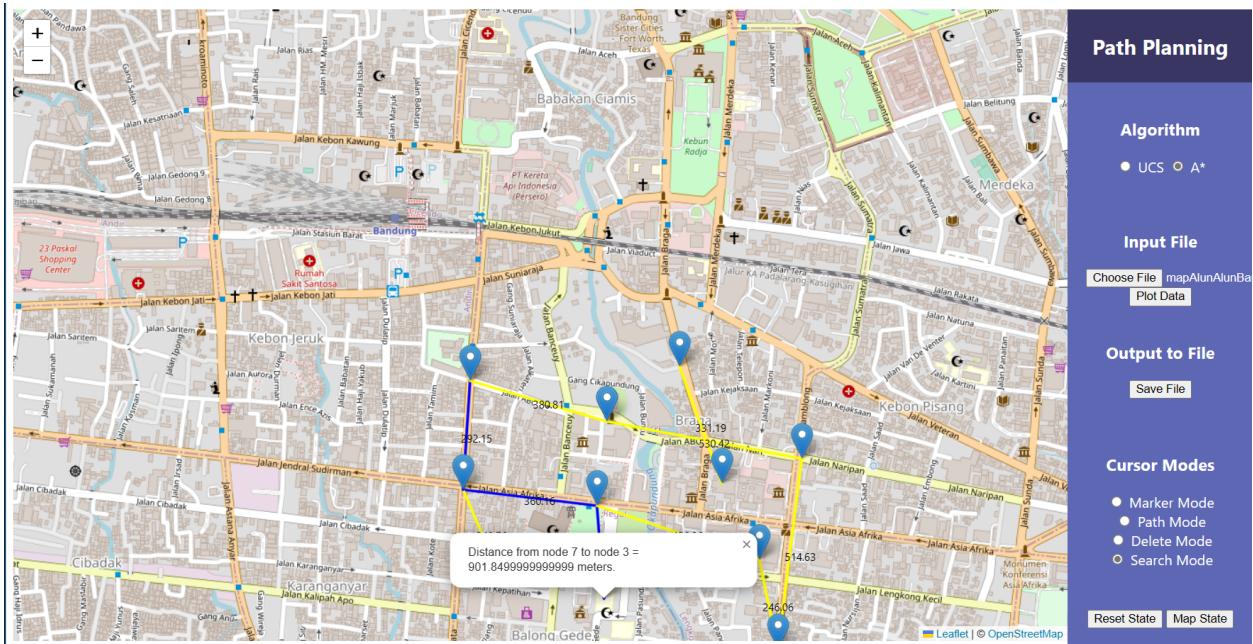
4.2.1 Input



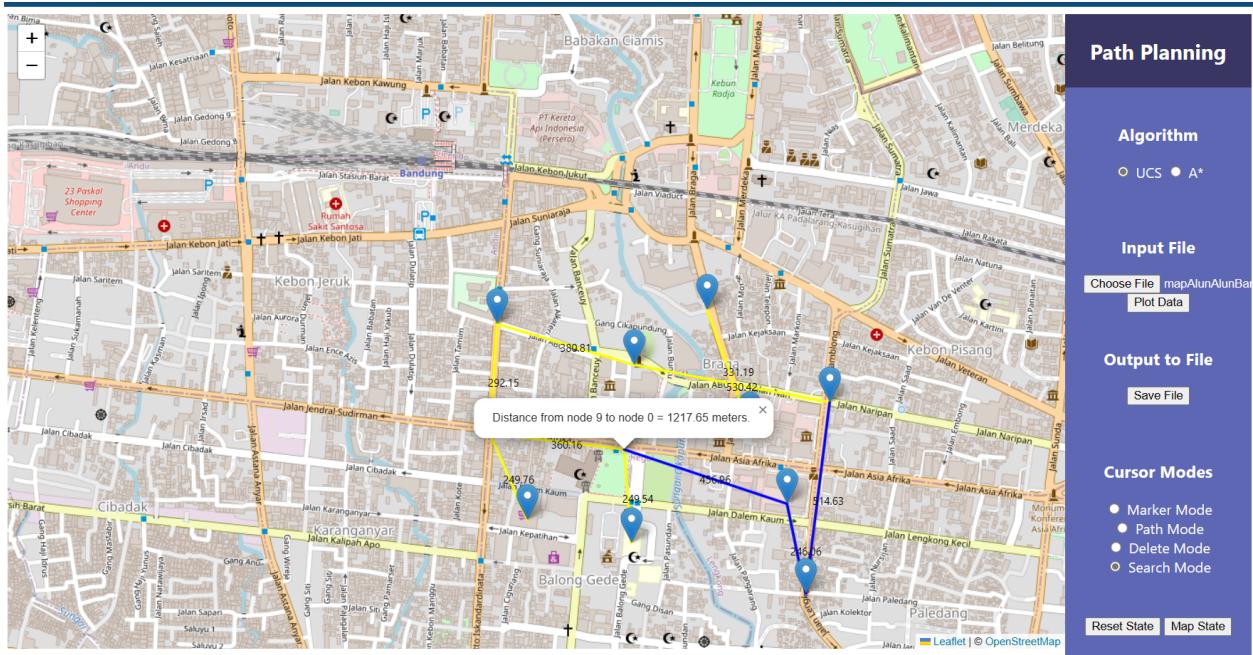
4.2.2 Start: Monumen Penjara Bung Karno, Goal: Gedung Keuangan (A* Algorithm)



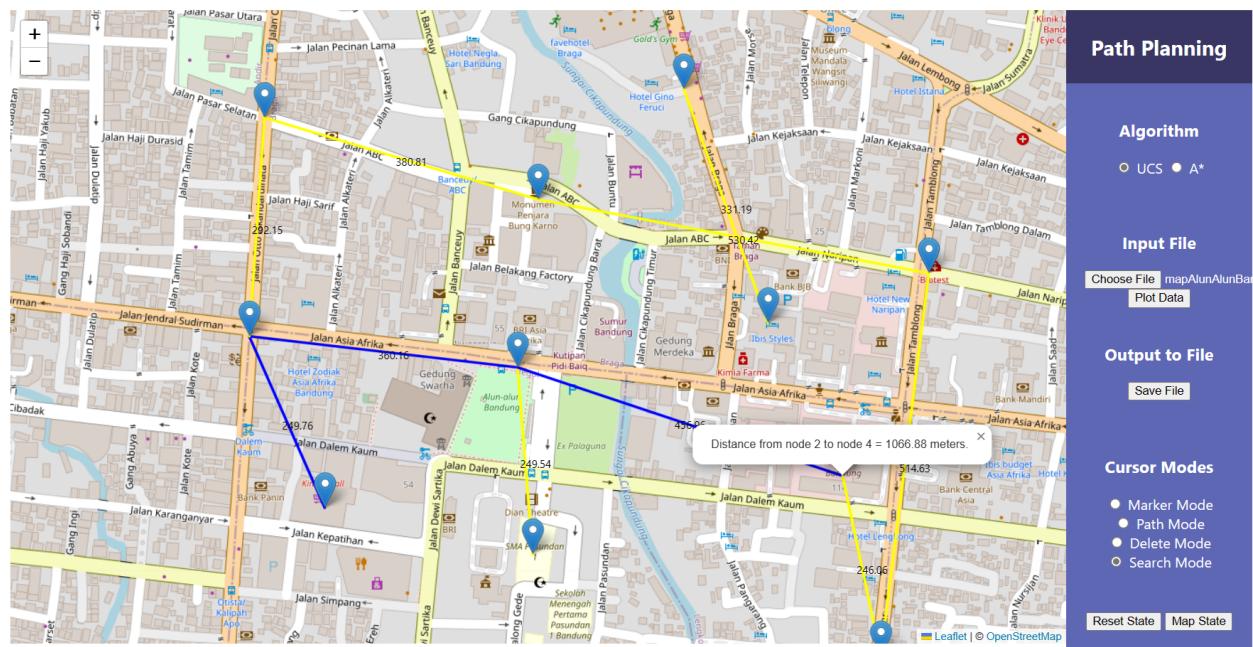
4.2.3 Start: Pasar Selatan, Goal: SMA Pasundan 1 (A* Algorithma)



4.2.4 Start: BioTest, Goal: Alun-Alun Kota Bandung (UCS Algorithm)

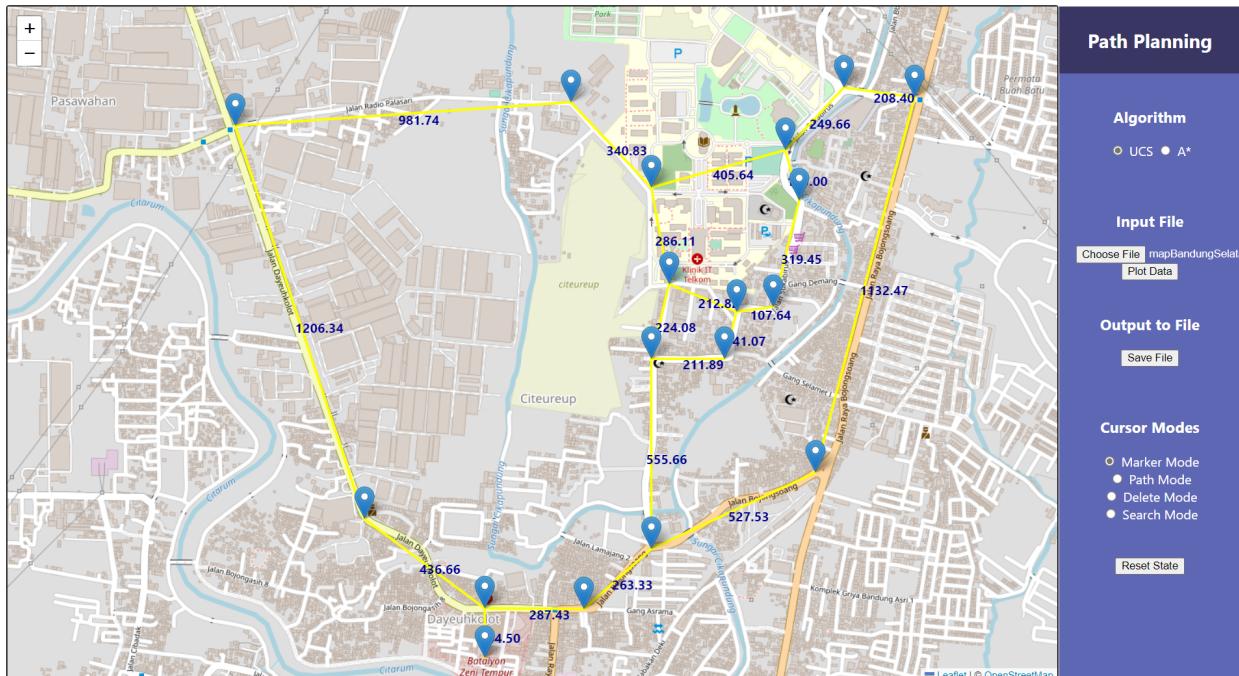


4.2.5 Start: King's Mall, Goal: Gedung Keuangan (UCS Algorithm)

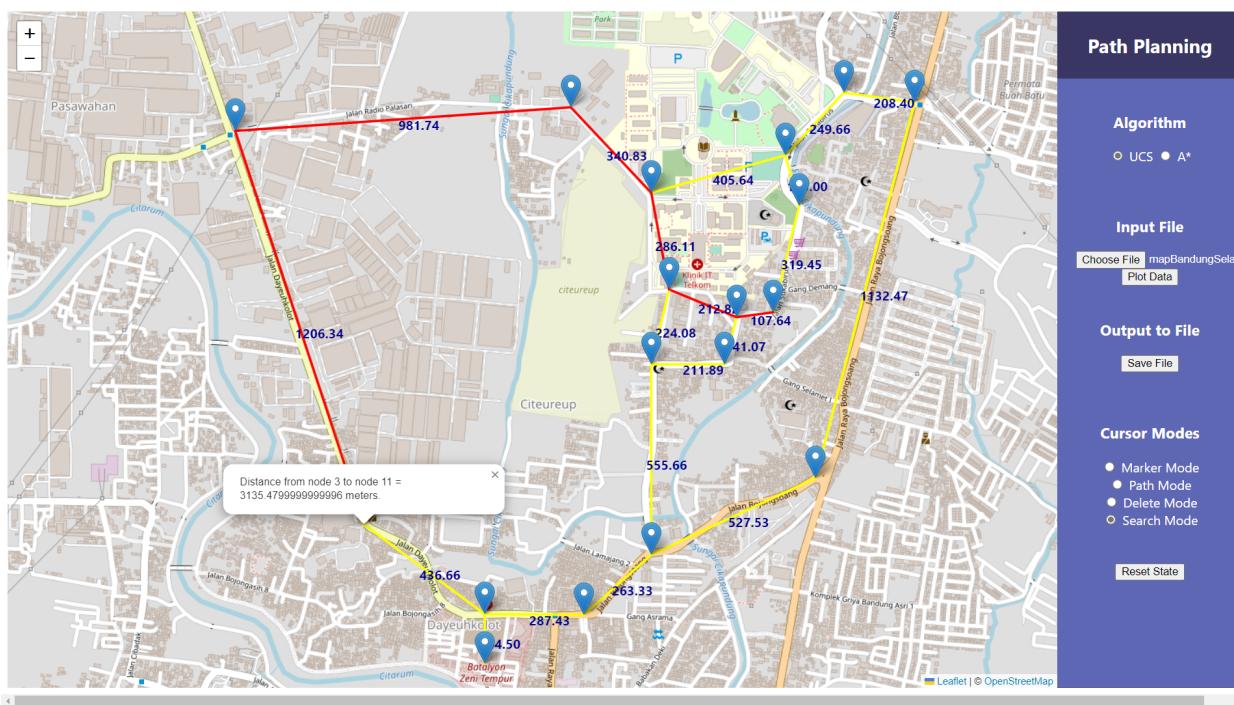
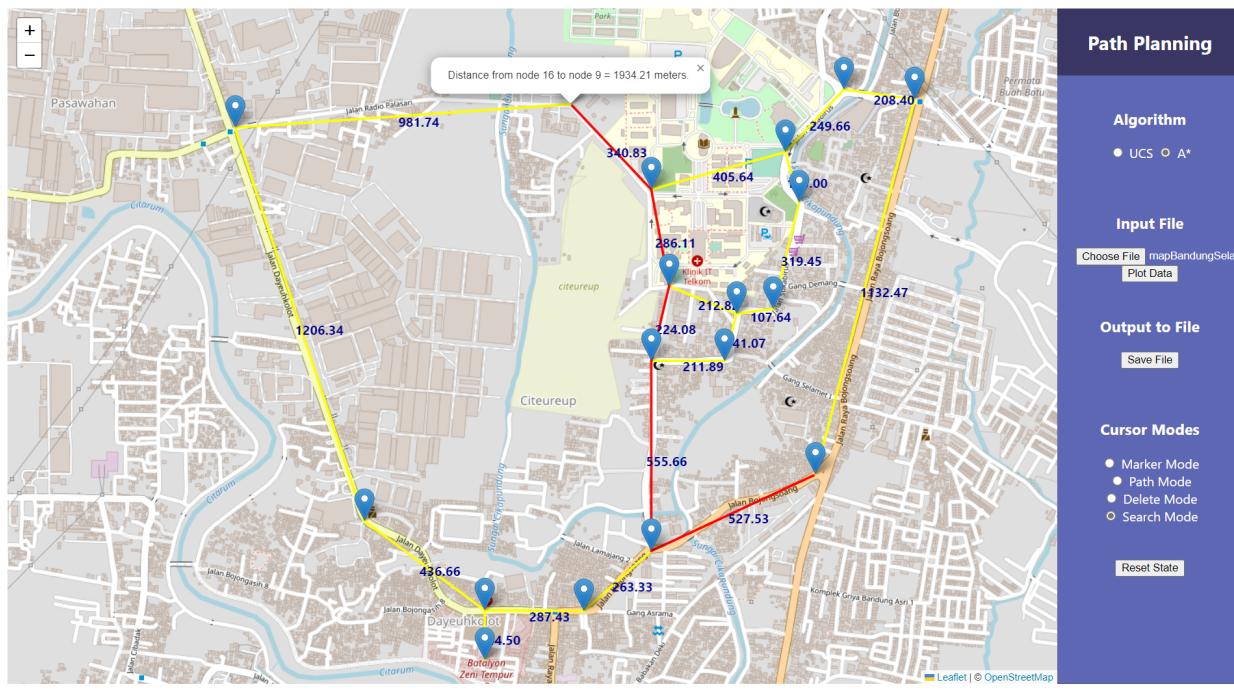


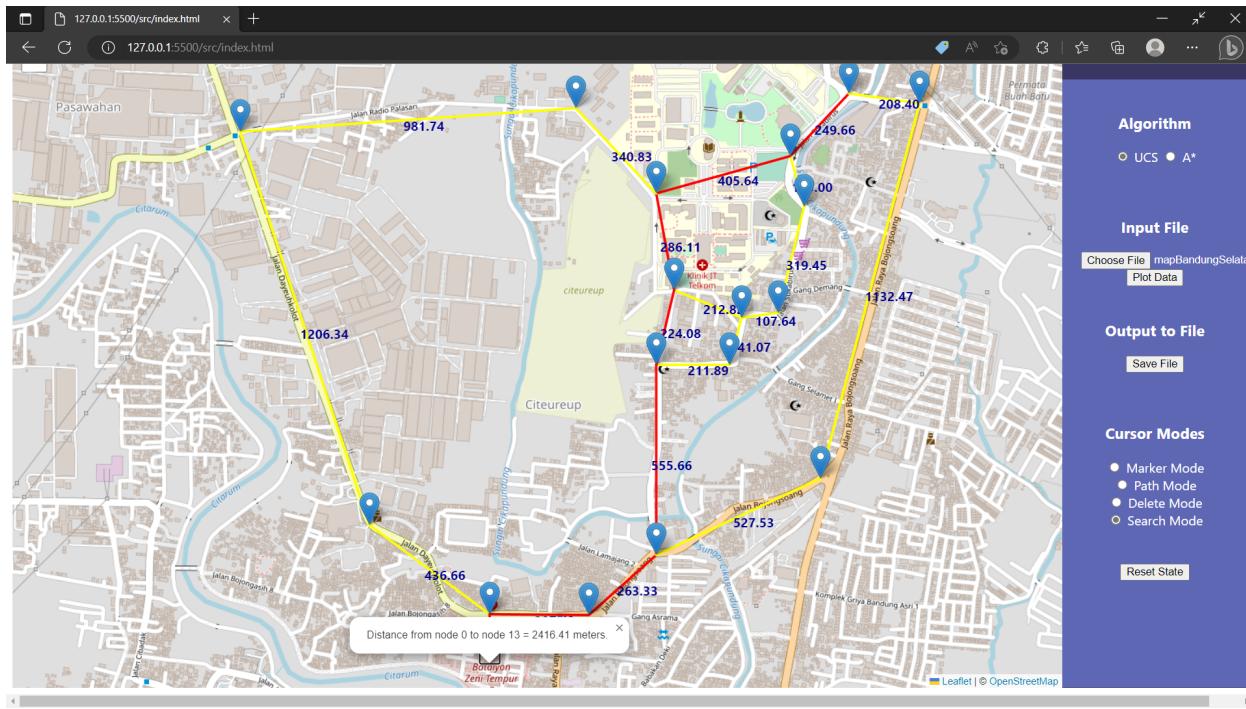
4.3 Map Sekitar Bandung Selatan

4.3.1 Input



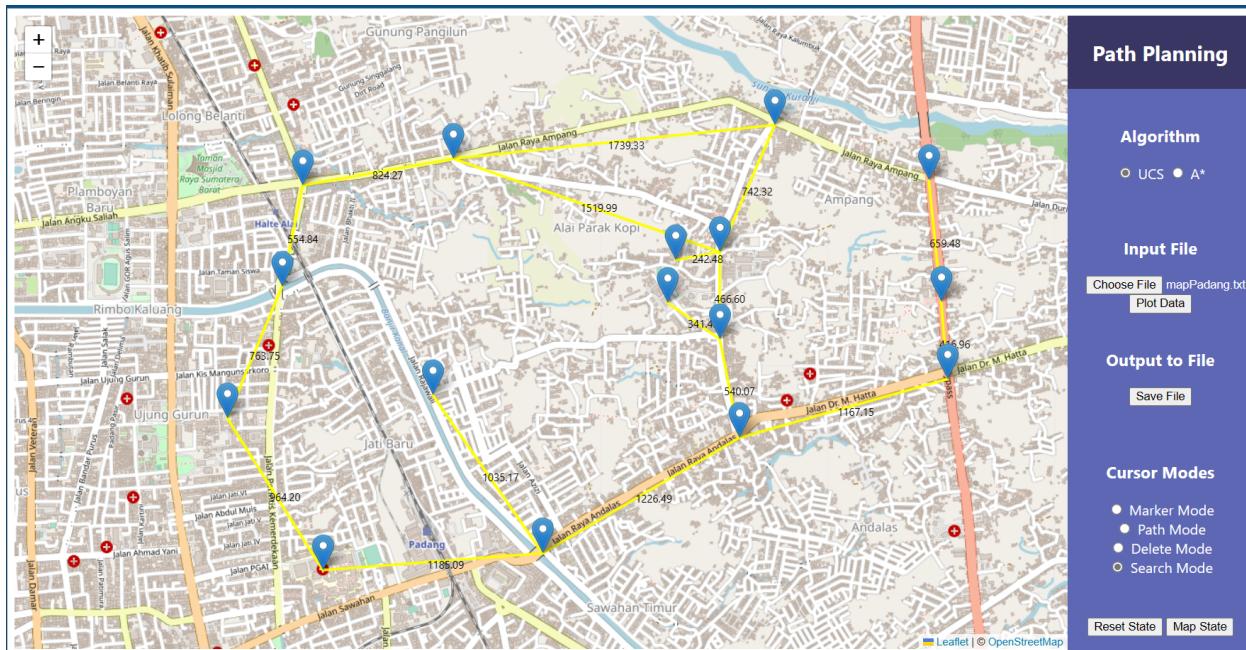
4.3.2 Output 1



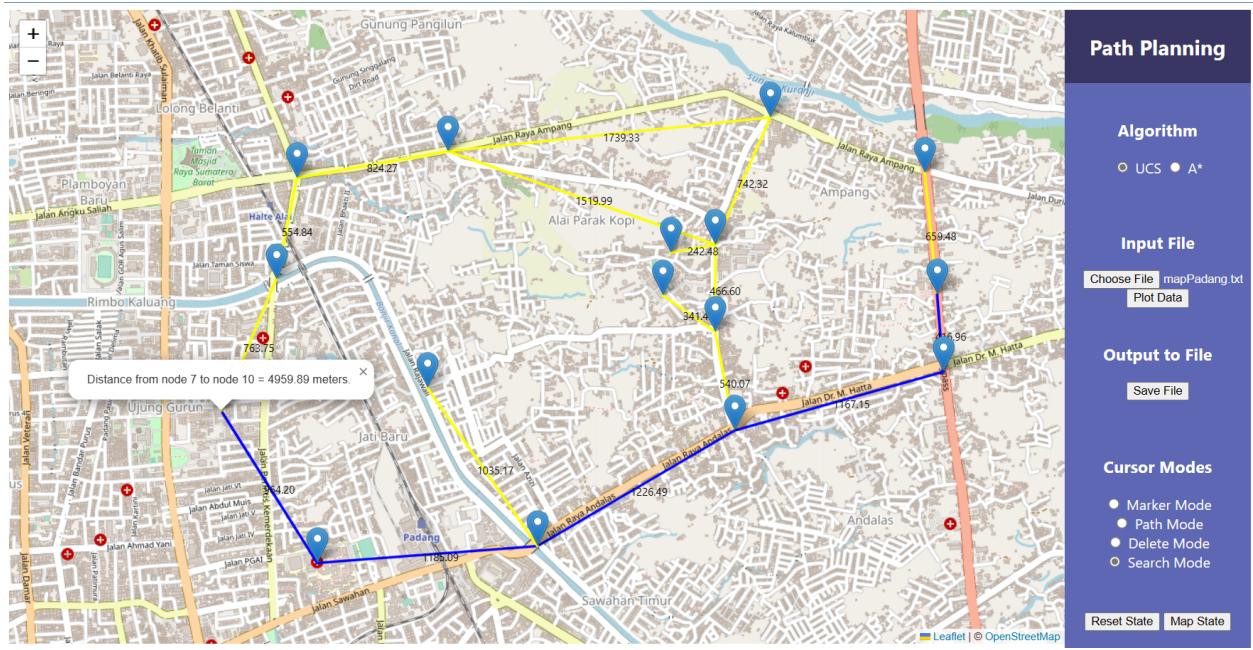


4.4 Map Sekitar Padang

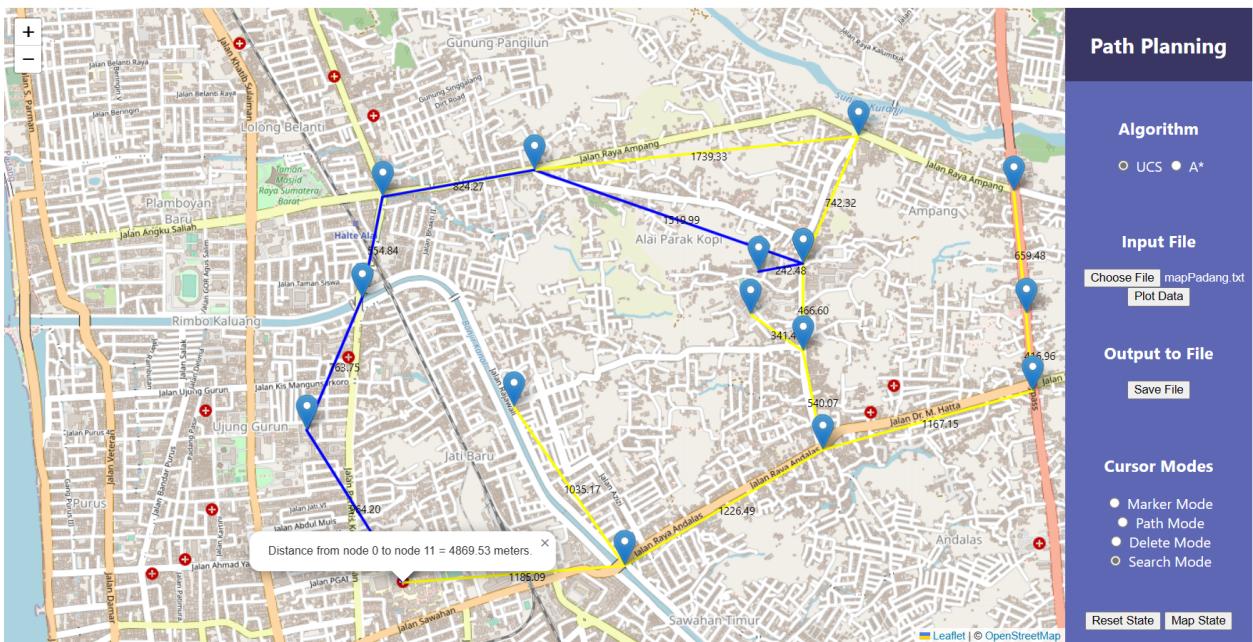
4.4.1 Input



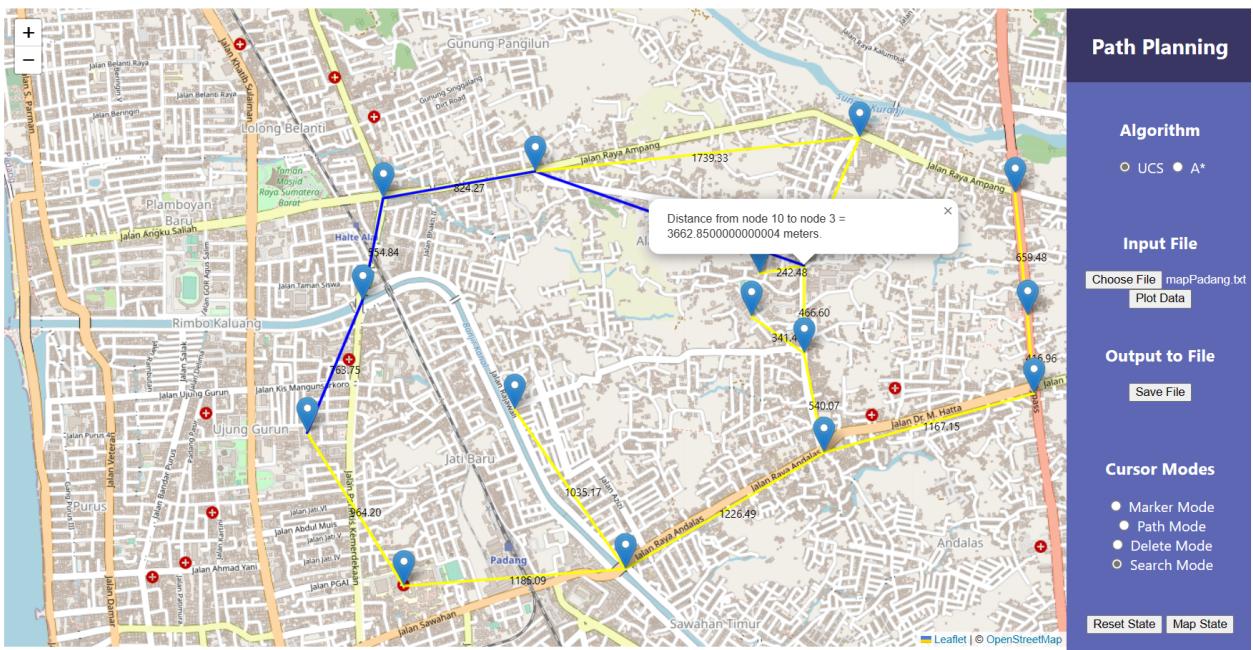
4.4.2 Start: SPBU Pasar Ambacang, Goal: SMAN 10 Padang (UCS Algorithm)



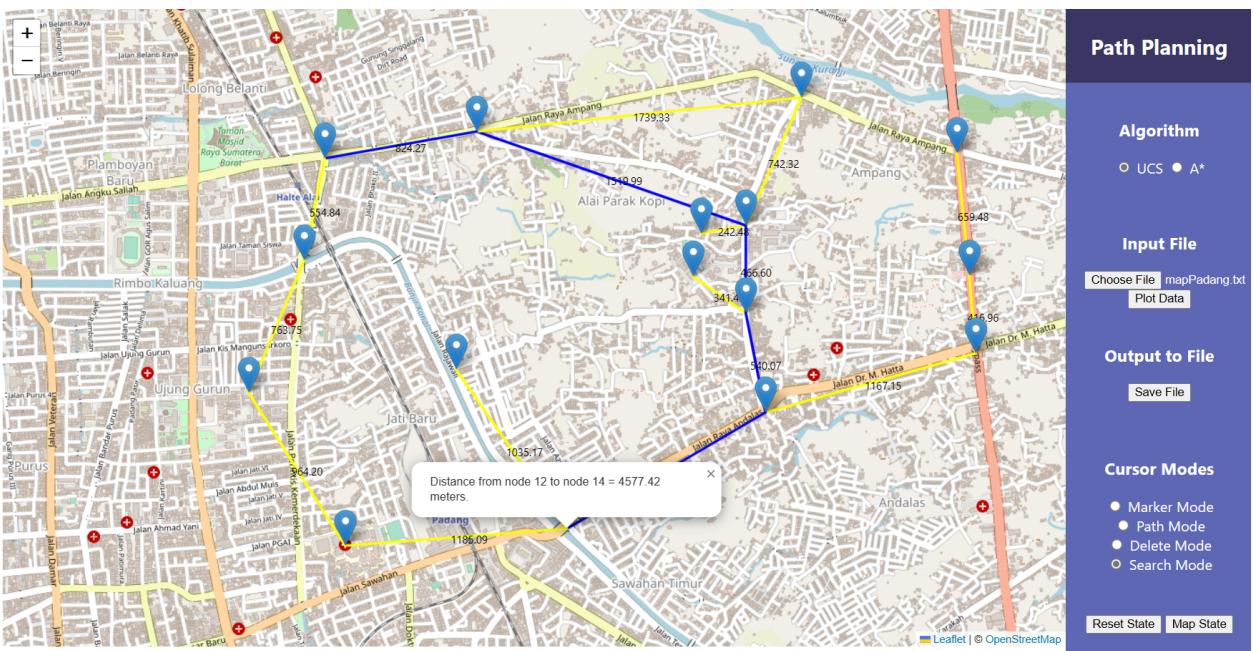
4.4.3 Start: Masjid Baiturrahman, Goal: RSUP M. Djamil (UCS Algorithm)



4.4.4 Start: SMAN 10 Padang, Goal: Rumah Fadhil (A* Algorithm)



4.4.5 Start: Pasar Alai, Goal: Simpang Haru (A* Algorithm)



Bab V: Simpulan

Algoritma UCS adalah algoritma yang termasuk *non-informed search*, sedangkan algoritma A* adalah algoritma yang termasuk *informed search*. Algoritma A* selalu bisa mencari jalur terpendek lebih cepat daripada algoritma UCS karena adanya nilai heuristic yang nantinya akan mempengaruhi prioritas pengecekan node menjadi lebih dekat pada goal. Namun, algoritma A* membutuhkan *memory space* yang lebih banyak dibandingkan dengan algoritma UCS karena harus menyimpan node yang sudah dikunjungi untuk menentukan bound pada saat mengunjungi node yang sama. Di lain sisi, algoritma UCS tidak perlu menyimpan node yang sudah dilalui karena sudah dipastikan node yang dilalui sudah berada dalam jalur yang optimal.

Lampiran

Link *repository* github : https://github.com/Mehmed13/Tucil3_13521066_13521157

Tabel check list :

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	✓	