Penerapan Algoritma *Route Planning* untuk Menemukan Jarak Relatif pada Sistem Penerimaan Peserta Didik Baru (PPDB) SMA Jalur Zonasi di Provinsi Riau

Mohammad Farhan Fahrezy - 13521106
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521106@std.stei.itb.ac.id

***Abstrak*—Penerimaan Peserta Didik Baru (PPDB) merupakan agenda tahunan yang terdiri dari proses pendaftaran dan seleksi untuk calon peserta didik baru di Indonesia. Salah satu jalur masuk dalam PPDB adalah jalur zonasi. Jalur tersebut menggunakan jarak domisili calon peserta didik menuju sekolah menggunakan jarak absolut. Pada penelitian ini, dilakukan penerapan algoritma *route planning* untuk menemukan jarak relatif sebagai metode alternatif untuk menemukan jarak yang lebih akurat untuk proses perankingan jalur zonasi pada PPDB SMA di Provinsi Riau.**

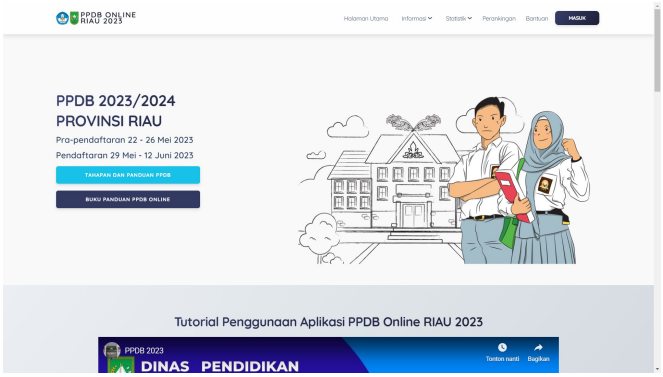
Kata Kunci*—*Route Planning, Uniform Cost Search, A*, Zonasi, PPDB, SMA,

I. PENDAHULUAN

Penerimaan Peserta Didik Baru atau yang biasa disebut PPDB merupakan agenda tahunan penerimaan peserta didik baru di setiap jenjang sekolah di Indonesia[1]. PPDB terdiri dari serangkaian proses pendaftaran dan seleksi yang dilakukan oleh pemerintahan daerah beserta sekolah negeri yang berada di daerah tersebut. Proses pendaftaran dan seleksi dapat dilakukan secara daring maupun luring, tergantung dari kapabilitas SDM pada daerah tersebut.

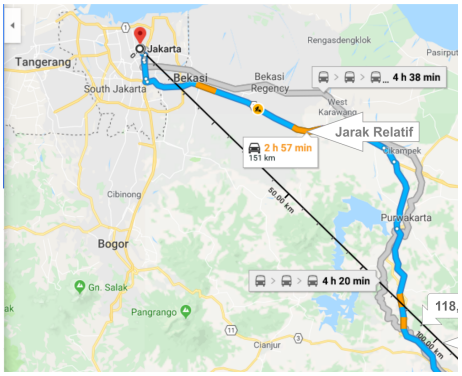
Provinsi Riau memiliki Sistem PPDB untuk SMA dan SMK yang berada di Provinsi Riau. Seluruh keberlangsungan PPDB dilakukan secara daring pada pranala https://ppdb.riau.go.id/. Pada pendaftaran untuk tingkat SMA, PPDB Provinsi Riau memiliki beberapa jalur pendaftaran yang tersedia, salah satunya adalah jalur zonasi.

Sistem zonasi merupakan salah satu kebijakan yang ditempuh Kementerian Pendidikan dan Kebudayaan (Kemendikbud) Republik Indonesia untuk menghadirkan pemerataan akses pada layanan pendidikan, serta pemerataan kualitas pendidikan nasional[2]. Jalur zonasi merupakan jalur pendaftaran yang menggunakan jarak domisili calon peserta didik dengan sekolah yang dipilih. Pemeringkatan pada jalur ini dilihat dari skor "keterjangkauan" yaitu konversi dari jarak rumah calon peserta didik dengan sekolah tujuannya. Skor tersebut dapat berbeda-beda pada tiap sekolah, tergantung dari segi demografi dan peminat pendaftar sekolah tersebut.



Gambar 1.1 Tampilan Laman Utama Situs PPDB 2023/2024 Provinsi Riau
Sumber: https://ppdb.riau.go.id/

Penentuan jarak domisili calon peserta didik dengan sekolah tujuan ditentukan menggunakan metode jarak absolut. Jarak absolut merupakan jarak linear antara dua titik pada permukaan bumi. Pada penelitian ini, akan dilakukan penerapan algoritma *route planning,* yaitu algoritma *Unified Cost Search* (UCS) dan algoritma A* untuk menentukan jarak relatif dari domisili calon peserta didik dengan sekolah tujuannya.



Gambar 1.2 Perbandingan Jarak Absolut dan Jarak Relatif
Sumber: https://www.google.com/maps/

II. TEORI DASAR

A. Algoritma *Route Planning*

Algoritma *route planning* atau perencanaan rute merupakan suatu metode yang digunakan untuk mencari rute yang paling optimal antara dua atau lebih titik dalam suatu ruang yang terdiri dari *node* atau lokasi tertentu. Tujuan dari algoritma ini adalah mencari rute terpendek, tercepat, atau paling efisien berdasarkan kriteria tertentu.

Algoritma ini dapat diklasifikasikan menjadi dua kategori utama, yaitu *uninformed seacrh* dan *informed search.* Algoritma *uninformed search* atau yang biasa disebut *blind search*, merupakan kategori algoritma *route planning* yang mencari rute optimal tanpa ada informasi khusus mengenai lingkungan atau masalah saat mencari rute. Algoritma ini hanya bergantung pada informasi *node* atau lokasi yang diberikan tanpa ada metode heuristik. Beberapa contoh algoritma yang termasuk ke *uninformed search* adalah Algoritma *breadth-first search* (BFS), *depth-first search* (DFS), *depth-limited search* (DLS), *iterative deepening search* (IDS), dan *uniform cost search* (UCS).

Sedangkan algoritma *informed search* atau juga dapat disebut *heuristic search*, merupakan kategori algoritma yang mencari rute optimal menggunakan informasi tambahan (heuristik) untuk membantu algoritma mendapatkan rute yang lebih optimal. Informasi heuristik pada algoritma ini dapat berupa jarak absolut berupa jarak antara dua titik pada garis lurus. Beberapa contoh algoritma yang termasuk ke *informed search* adalah *greedy best first search* dan *A-star* (A*)

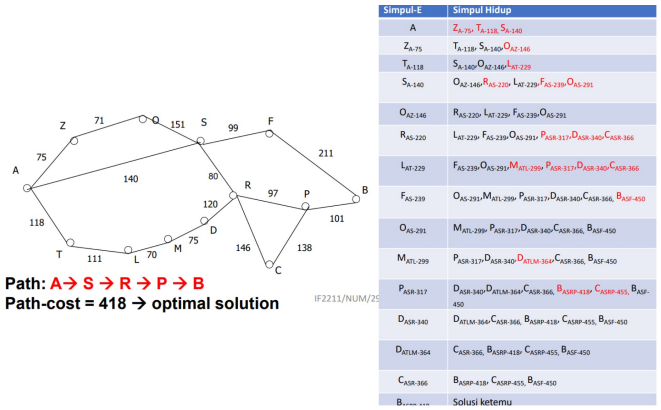
B. Algoritma *Uniform Cost Search* (UCS)

Algoritma *uniform cost search* (UCS) adalah algoritma *route planning* yang mencari rute dengan biaya minimum dari *node* awal ke *node* tujuan. UCS menggunakan pendekatan *greedy*, di mana simpul berikutnya yang akan dijadikan simpul hidup dipilih berdasarkan biaya yang paling rendah. Algoritma ini menggunakan asumsi nilai sisi adalah non-negatif. Penentuan biaya pada algoritma UCS dilakukan menggunakan persamaan berikut.

$$g(n) = \text{path cost from root to } n$$

Algoritma UCS dapat diterapkan dengan menggunakan struktur data seperti *priority queue* untuk mengoptimalkan pencarian simpul dengan biaya g(n) terendah. Setiap simpul dalam *priority queue* diurutkan berdasarkan biayanya, sehingga simpul dengan biaya terendah akan dipilih terlebih dahulu.

Keuntungan dari algoritma UCS adalah kemampuannya untuk menemukan solusi optimal, jika ada. Namun, algoritma ini dapat menghabiskan banyak waktu dan memori jika graf yang digunakan memiliki banyak *node* atau jika terdapat rute dengan biaya yang sangat tinggi.



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.....	Solusi ketemu

Gambar 2.2.1 Algoritma UCS
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

C. Algoritma A* (A-star)

Algoritma A* adalah algoritma *route planning* yang yang digunakan untuk mencari rute dengan biaya minimum antara *node* awal dan *node* tujuan pada suatu graf berbobot. Algoritma ini termasuk ke dalam algoritma *informed search* yang menggunakan informasi tambahan (heuristik) dalam mengevaluasi *node* yang akan digunakan.

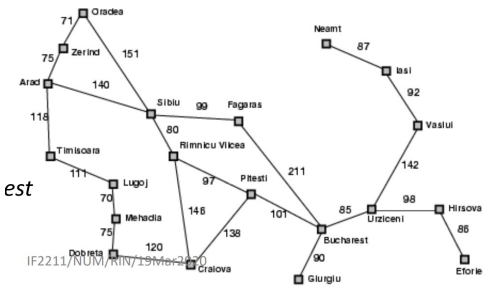
Ide dari algoritma ini adalah dengan menghindari melakukan pengecekan kedalam suatu *node* yang sudah "mahal". Algoritma ini menggunakan fungsi evaluasi sebagai berikut.

$$f(n) = g(n) + h(n)$$

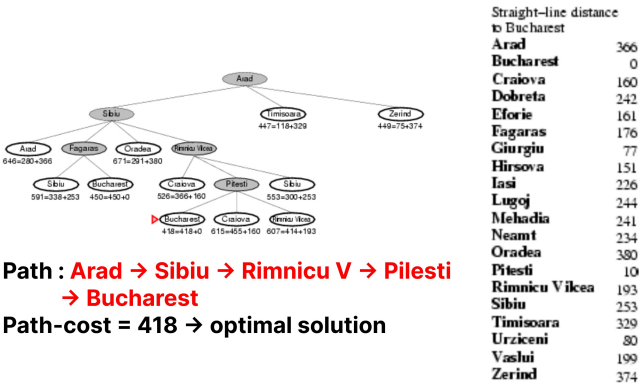
dengan,

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost from n to goal
$f(n)$ = estimated total cost of path through n to goal

Algoritma ini menggabungkan biaya g(n) yang telah dilewati dan nilai heuristik h(n) yaitu jarak lurus yang diperlukan untuk mencapai tujuan.



Gambar 2.3.1 Representasi Peta Romania pada Graf Berbobot
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf



Gambar 2.3.2 Algoritma A* (kiri), Jarak Heuristik (kanan)
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

Fungsi heuristik harus *admissible* yaitu tidak melebih-lebihkan biaya sebenarnya dan idealnya konsisten sehingga nilai f(n) tidak menjadi lebih rendah daripada biaya sebenarnya.

D. Persamaan Haversine

Persamaan Haversine digunakan untuk menghitung jarak antara dua titik pada suatu permukaan bola menggunakan koordinat latitude dan longitude[5]. Persamaan ini merupakan persamaan turunan dari persamaan *spherical law of cosines*.

$$\cos(A)\sin(b)\sin(c) + \cos(b)\cos(c)$$

Persamaan Haversine adalah sebagai berikut

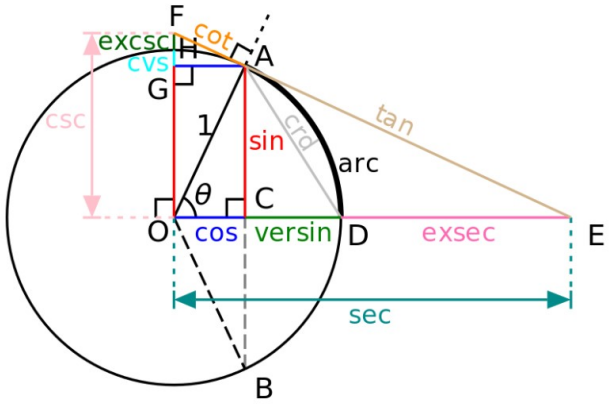
$$a = sin^2((\varphi_B - \varphi_A)/2) + cos(\varphi_A) * cos(\varphi_B) * sin^2((\lambda_B - \lambda_A)/2)$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

dengan ,

φ = koordinat latitude
λ = koordinat longitude
R = radius bumi = 6371000 m
d = jarak antara dua koordinat (m)

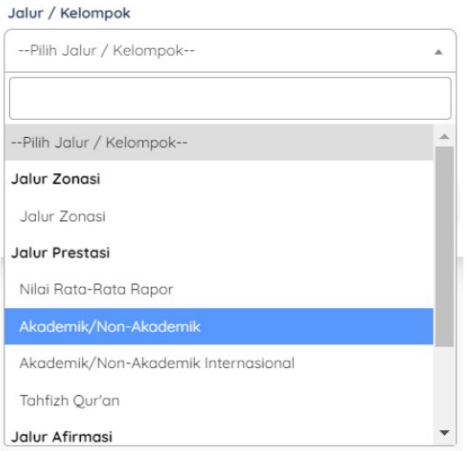


Gambar 2.4.1 Persamaan Haversine
Sumber:
https://community.esri.com/legacyfs/online/375715_Circle-trig6_svg.png

E. Penerimaan Peserta Didik Baru di Provinsi Riau

Penerimaan Peserta Didik Baru atau yang biasa disebut PPDB merupakan agenda tahunan penerimaan peserta didik baru di setiap jenjang sekolah di Indonesia[1]. PPDB terdiri dari serangkaian proses pendaftaran dan seleksi yang dilakukan oleh pemerintahan daerah beserta sekolah negeri yang berada di daerah tersebut.

Provinsi Riau memiliki Sistem PPDB untuk SMA yang berada di Provinsi Riau. Seluruh keberlangsungan PPDB dilakukan secara daring pada pranala https://ppdb.riau.go.id/. Pada pendaftaran untuk tingkat SMA, PPDB Provinsi Riau memiliki beberapa jalur pendaftaran yang tersedia, seperti pada gambar 2.5.1.



Gambar 2.5.1 Jalur Pendaftaran PPDB 2023/2024 SMA Provinsi Riau
Sumber:
https://ppdb.riau.go.id/file/manual_book_siswa_ppdb.pdf

PPBD SMA di Provinsi Riau menggunakan system "*knock-out*" atau sistem gugur yang artinya jika calon peserta didik yang mendaftar di SMA tertentu dengan jalur tertentu "jatuh" dari pemeringkatan, maka ia dinyatakan tidak lolos untuk sekolah dengan jalur masuk tersebut. Namun, tiap peserta didik dapat mendaftarkan ulang ke sekolah yang berbeda. Hasil pemeringkatan dibuka secara umum pada situs PPDB Provinsi Riau.

Diantara beberapa jalur yang tersedia, ada jalur zonasi yang merupakan jalur masuk yang menggunakan jarak domisili calon peserta didik menuju sekolah tujuan sebagai kriteria penilaian. Pemeringkatan pada jalur ini dilihat dari skor "keterjangkauan" yaitu konversi dari jarak rumah calon peserta didik dengan sekolah tujuannya. Skor tersebut dapat berbeda untuk tiap sekolah, tergantung dari segi demografi dan peminat pendaftar sekolah tersebut.

III. PENERAPAN ALGORITMA *ROUTE PLANNING*

Algoritma *route planning* yang akan digunakan pada penerapan system PPDB adalah algoritma UCS dan A*. Untuk melakukan percobaan, penulis membuat sebuah *web-app* menggunakan framework React Js dengan menggunakan bahasa pemrograman Javasript. Untuk *source code* dari program yang digunakan, dapat diakses pada https://github.com/farhanfahreezy/Tucil3_UCS_Astar.

A. Implementasi Persamaan Haversine

Persamaan haversine digunakan untuk menghitung jarak antara dua lokasi. Input yang dibutuhkan pada fungsi ini adalah koordinat latitude dan koordinat longitude dari kedua lokasi. Luaran yang diberikan pada fungsi ini adalah jarak dalam meter.

```
export function getDistance(lat1, lon1, lat2, lon2) {
    const earthRadius = 6371000; // meter
    const dLat = toRadians(lat2 - lat1);
    const dLon = toRadians(lon2 - lon1);
    const a =
        Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(toRadians(lat1)) * Math.cos(toRadians(lat2)) *
        Math.sin(dLon / 2) * Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    const distance = earthRadius * c;
    return distance;
}
```

Gambar 3.1.1 Implementasi Persamaan Haversine pada Fungsi getDistance()
Sumber: Dokumentasi Pribadi

B. Implementasi Algoritma *Uniform Cost Search* (UCS)

Algoritma UCS menggunakan struktur *prio queue* untuk menentukan *node* selanjutnya yang akan digunakan untuk dijadikan *node* hidup. Implementasi dari kelas *prio queue* dapat dilihat pada gambar 3.2.1 dan implementasi dari algoritma UCS dapat dilihat pada gambar 3.2.2.

```
export class PrioQueue{
    constructor(){
        this.queue = [];
    }

    enqueue(node,step,path){
        var queueElement = {
            node,
            step,
            path
        };
        var added = false;
        for (var i = 0; i < this.queue.length; i++) {
            if (queueElement.step < this.queue[i].step) {
                this.queue.splice(i, 0, queueElement);
                added = true;
                break;
            }
        }
        if (!added) {
            this.queue.push(queueElement);
        }
    }

    dequeue(){
        return this.queue.shift();
    }

}
```

Gambar 3.2.1 Implementasi Kelas *Prio Queue*
Sumber: Dokumentasi Pribadi

```
export function searchPathUCS(start,finish,nodes,weightedAdjacencyMatrix){
    // Inisiasi PriorityQueue dan activeNode
    const pQueue = new PrioQueue();
    let activeNode = {
        node : start,
        step : 0,
        path : [start.id]
    };

    // Masukkan activeNode ke queue
    pQueue.enqueue(activeNode.node,activeNode.step,activeNode.path);

    console.time("ucsTIME");
    // Mencari jawaban dengan algoritma UCS
    while(activeNode.node != finish){
        activeNode = doTheThing(pQueue,finish,weightedAdjacencyMatrix,nodes);
    }
    console.timeEnd("ucsTIME");
    return activeNode;
}

function doTheThing(pQueue,finish,weightedAdjacencyMatrix,nodes){
    // Algoritma UCS

    // Mengeluarkan isi queue paling depan
    let activeNode = pQueue.dequeue();

    if(activeNode.node == finish){
        // Mengecek jika activeNode merupakan finish
        return activeNode;
    } else {
        // Mengecek semua adjacent node
        for(let i = 0;i<weightedAdjacencyMatrix[0].length;i++){
            if(weightedAdjacencyMatrix[activeNode.node.id-1][i]!=0){
                let id = i+1;
                if(!activeNode.path.includes(id)){
                    // Megecek agar pencarian tidak mundur kebelakang
                    let newPath = activeNode.path.slice();
                    let newStep = activeNode.step + weightedAdjacencyMatrix[activeNode.node.id-1][i];
                    newPath.push(id);

                    // Memasukkan elemen baru ke queue
                    pQueue.enqueue(getNodesById(id,nodes),newStep,newPath);
                }
            }
        }
    }
    return activeNode;
}
```

Gambar 3.2.2 Implementasi Algoritma UCS
Sumber: Dokumentasi Pribadi

C. Implementasi Algoritma A*

Jarak heuristik yang digunakan pada algoritma A* adalah jarak antara setiap *node* ke *node* tujuan. Pada implementasinya, digunakan fungsi getDistance() untuk mendapatkan jarak antara dua *node*. Implementasi jarak heuristik dapat dilihat pada gambar 3.3.1.

```
function getHeuristicDistance(nodes,finish){
    const hashMap = {};
    for(let i = 0;i<nodes.length;i++){
        let id = i+1;
        let check = getNodesById(id,nodes);
        if(check!=finish){
            hashMap[id] = getDistance(check.lat,check.lon,finish.lat,finish.lon);
        } else {
            hashMap[id] = 0;
        }
    }
    return hashMap;
}
```

Gambar 3.3.1 Implementasi Fungsi getHeuristicDistance()
Sumber: Dokumentasi Pribadi

Luaran dari fungsi getHeuristicDistance() adalah sebuah *hash-map* yang berisi pasangan *key* = id dan *value* = jarak heuristik. Jarak heuristik akan digunakan pada fungsi evaluasi f(n) pada algoritma A*. Implementasi algoritma A* dapat dilihat pada gambar 3.3.2.

```
export function searchPathAstar(start,finish,nodes,weightedAdjacencyMatrix){
    // Inisiasi Hashmap
    const heuristicDist = getHeuristicDistance(nodes,finish);
    const pQueue = new PrioQueue();
    let activeNode = {
        node : start,
        step : heuristicDist[start.id],
        path : [start.id]
    };
    // Masukkan activeNode ke queue
    pQueue.enqueue(activeNode.node,activeNode.step,activeNode.path);

    console.time("astrTIME");
    // Mencari jawaban dengan algoritma UCS
    while(activeNode.node != finish){
        activeNode = doTheThing(pQueue,finish,weightedAdjacencyMatrix,nodes,heuristicDist);
    }
    console.timeEnd("astrTIME");
    return activeNode;
}

function doTheThing(pQueue,finish,weightedAdjacencyMatrix,nodes,heuristicDist){
    // Algoritma A*

    // Mengeluarkan isi queue paling depan
    let activeNode = pQueue.dequeue();

    if(activeNode.node == finish){
        // Mengecek jika activeNode merupakan finish
        return activeNode;
    } else {
        // Mengecek semua adjacent node
        let activeNoHeuristic = activeNode.step - heuristicDist[activeNode.node.id];
        for(let i = 0;i<weightedAdjacencyMatrix[0].length;i++){
            if(weightedAdjacencyMatrix[activeNode.node.id-1][i]!=0){
                let id = i+1;
                if(!activeNode.path.includes(id)){
                    // Megecek agar pencarian tidak mundur kebelakang
                    let newPath = activeNode.path.slice();
                    let newStep = activeNoHeuristic +
                                  weightedAdjacencyMatrix[activeNode.node.id-1][i]
                                  + heuristicDist[id];
                    newPath.push(id);

                    // Memasukkan elemen baru ke queue
                    pQueue.enqueue(getNodesById(id,nodes),newStep,newPath);
                }
            }
        }
    }
    return activeNode;
}
```

Gambar 3.3.2 Implementasi Algoritma A*
Sumber: Dokumentasi Pribadi

D. Menjalankan Program

Program menerima sebuah input file *.txt yang merupakan representasi peta dalam bentuk *node-node*. Tiap *node* harus diberikan koordinat latitude dan longitude beserta matriks ketetanggannya. Berikut merupakan format input untuk file *.txt.

```
{jumlah_node}
{nama_node1} {lattitude_node1},{longitude_node1}
{nama_node2} {lattitude_node2},{longitude_node2}
.
.
{nama_nodeN} {lattitude_nodeN},{longitude_nodeN}
{matriks_ketetanggaan_node_1}
{matriks_ketetanggaan_node_2}
.
.
{matriks_ketetanggaan_node_N}
```

Agar dapat menjalankan program, komputer *client* harus sudah terinstall node.js versi terbaru. Berikut merupakan cara menjalankan program.

1. *Clone* repository atau unduh sebagai ZIP ke komputer
2. Buka *root directory* dan jalankan `npm run dev` pada terminal
3. Buka `http://localhost:XXXX/` yang muncul pada terminal untuk membuka website
4. Klik `Choose File` untuk memilih file *.txt
5. Pilih `Origin ID` dan `Destination ID` pada bagian *dropdown*
6. Pilih algoritma yang akan digunakan
7. Klik `Search`

Setelah tombol Search diklik, program akan memberikan luaran berupa jarak relatif dalam meter, rute yang dilalui, dan waktu eksekusi program.

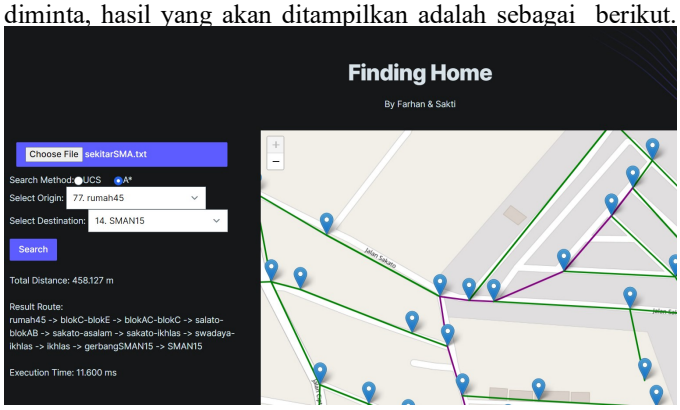
E. Implementasi Menggunakan Input Peta SMAN 15 Pekanbaru

Pada percobaan kali ini, akan dilakukan percobaan penentuan jarak relatif menggunakan peta SMAN 15 Pekanbaru. Berikut adalah isi dari file sekitarSMA.txt

```
112
ciptaKarya-sakato 0.444845,101.390014
rumah1 0.444433,101.390722
sakato-ikhlas 0.443833,101.391829
ciptaKarya-swadaya 0.443976,101.390176
.
.
pertigaan2Pendidikan 0.442884,101.395774
rumah62 0.443804,101.397009
rumah63 0.443569,101.397299
rumah64 0.443954,101.389554
0 1 0 1 0 0 0 0 0 0 . . .
.
.
0 0 0 1 0 0 0 0 0 0 . . .
```

Isi file dipersingkat karena jumlah node terlalu banyak (112 nodes), file bisa diakses di https://github.com/farhanfahreezy/Tucil3_UCS_Astar/blob/main/test/sekitarSMA.txt

Untuk mendapatkan informasi mengenai jarak relatif dari domisili calon peserta didik menuju sekolah, dapat melihat tata cara pada bagian D. Setelah mengikuti seluruh tahapan yang diminta, hasil yang akan ditampilkan adalah sebagai berikut.



Gambar 3.5.1 Tampilan Jalur Relatif (garis ungu)
Sumber: Dokumentasi Pribadi

Selanjutnya, lakukan iterasi terhadap keseluruhan data yang ada pada map. Disini penulis membandingkan hasil pemeringkatan jika menggunakan jarak absolut dan jika menggunakan jarak relatif.

No	Jarak Absolut		Jarak Relatif	
	Nama	Jarak (m)	Nama	Jarak (m)
1	rumah7	43.47987	rumah5	112.7483
2	rumah38	99.06038	rumah41	145.7509
3	rumah35	107.0803	rumah4	160.1559
4	rumah5	108.5543	rumah7	169.4035
5	rumah41	120.9937	rumah3	184.4488
6	rumah39	126.4202	rumah39	187.6703
7	rumah34	132.9552	rumah35	194.4984
8	rumah10	144.2618	rumah8	210.2337
9	rumah3	145.4738	rumah34	228.6074
10	rumah12	146.0321	rumah40	230.7395
11	rumah30	148.375	rumah38	244.0867
12	rumah4	153.8451	rumah42	259.5667
13	rumah47	156.9636	rumah6	298.1257
14	rumah6	163.1574	rumah33	304.1969
15	rumah9	166.3644	rumah2	315.2744
16	rumah40	167.9213	rumah1	338.0833
17	rumah8	169.9906	rumah32	343.1046
18	rumah29	170.6127	rumah31	343.7898
19	rumah33	180.2606	rumah36	358.8543
20	rumah13	182.1536	rumah37	375.3616
21	rumah28	186.3202	rumah30	397.027
22	rumah31	186.8554	rumah26	408.2386
23	rumah19	210.1377	rumah28	408.4684
24	rumah32	214.6355	rumah64	410.2475
25	rumah11	216.7371	rumah9	414.1208
26	rumah37	222.4877	rumah11	414.8226
27	rumah56	228.2288	rumah27	423.6418
28	rumah26	231.3389	rumah29	437.6231
29	rumah42	234.7688	rumah25	443.0901
30	rumah20	234.8273	rumah10	443.8891
31	rumah18	239.5102	rumah45	458.1265
32	rumah27	240.8079	rumah46	461.9502
33	rumah46	244.7592	rumah12	476.7819
34	rumah36	245.2273	rumah43	479.9175
35	rumah60	247.1504	rumah47	485.197
36	rumah25	252.5467	rumah13	509.1286
37	rumah23	253.522	rumah44	533.168
38	rumah24	256.1025	rumah15	540.3279
39	rumah14	269.2596	rumah50	546.6425
40	rumah15	269.5846	rumah56	561.3287
41	rumah2	270.9868	rumah22	568.1468
42	rumah17	286.5028	rumah16	574.2044
43	rumah1	286.9338	rumah21	587.0304
44	rumah45	293.2466	rumah14	598.1523
45	rumah22	297.2098	rumah48	603.0264
46	rumah58	301.466	rumah23	612.7832
47	rumah21	303.9826	rumah51	626.841
48	rumah16	304.8229	rumah55	628.9002
49	rumah50	305.8524	rumah60	630.253
50	rumah61	311.6749	rumah24	640.6956
51	rumah44	315.1905	rumah58	657.8747
52	rumah43	350.736	rumah61	689.6904
53	rumah55	352.5137	rumah17	697.0474
54	rumah48	355.4703	rumah49	700.2222
55	rumah64	362.0393	rumah57	736.7738
56	rumah51	380.449	rumah52	765.4127
57	rumah57	401.2862	rumah54	808.0445
58	rumah52	447.1167	rumah53	821.6847
59	rumah49	448.5987	rumah18	866.0468
60	rumah53	467.8389	rumah20	869.4759
61	rumah54	495.9991	rumah62	883.1745
62	rumah62	520.5535	rumah19	913.5821
63	rumah63	545.7341	rumah63	924.6787
AVG		**252.7773**		**486.7013**
STD		**107.557**		**210.4332**

Tabel 3.5.1 Perbandingan antara Jarak Absolut dan Jarak Relatif
Sumber: Dokumentasi Pribadi

Pada tabel 3.5.1 dapat dilihat perbedaan yang signifikan antara jarak absolut dan jarak relatif. Dari statistic yang diberikan, rata-rata jarak domisili calon peserta didik menggunakan jarak relatif jauh lebih besar daripada jika menggunakan jarak absolut. Hal tersebut dikarenakan jarak relatif menggunakan jalur asli yang bisa dilalui oleh kendaraan.

Jarak relatif juga memiliki nilai standar deviasi yang lebih besar daripada nilai standar deviasi jarak absolut yang menandakan bahwa data jarak relatif memiliki pesebaran yang lebih merata daripada jarak absolut. Standar deviasi yang naik secara drastis ini disebabkan oleh banyaknya data rumah yang awalnya hanya mengambil garis lurus ke sekolah, sekarang harus menghitung jarak sebenarnya yang lebih jauh.

IV. KESIMPULAN

Algoritma *route planning* memiliki banyak aplikasi yang bisa digunakan pada kehidupan sehari-hari. Salah satu contohnya adalah pada sistem Penerimaan Peserta Didik Baru

(PPDB) SMA di Provinsi Riau. Algorima *route planning*, khususnya algoritma *uniform cost search* (UCS) dan algoritma A* dapat digunakan untuk mencari jarak relatif atau jarak yang benar-benar harus ditempuh peserta didik ke sekolahnya.

Jarak relatif dapat memberikan data yang lebih baik karena memiliki persebaran data yang lebih baik daripada menggunakan jarak absolut. Selain itu, penggunaan jarak relatif pada sistem zonasi PPDB SMA di Provinsi Riau juga membuka kesempatan bagi calon peserta didik untuk mendapatkan kesempatan yang sama besar dengan sesama calon peserta didik lain.

UCAPAN TERIMAKASIH

Puji Syukur kepada Tuhan Yang Maha Esa karena berkat rahmat dan karunianya penulis dapat menyelesaikan makalah Mata Kuliah IF2211 Strategi Algoritma sebagai bentuk tugas akhir dengan baik dan tepat pada waktunya. Penulis menyampaikan terima kasih kepada Dr. Nur Ulfa Maulidevi, S.T, M.Sc., selaku dosen pengajar Mata Kuliah IF2211 Kelas 02, beserta dosen penyampu mata kuliah lainnya yang juga telah membimbing kami dalam proses belajar mengajar. Penulis juga berterima kasih kepada orang tua yang telah berdoa, mendukung, dan memberikan motivasi untuk selalu belajar, sehingga penulis dapat menyelesaikan makalah ini. Tak lupa juga penulis berterima kasih kepada pembuat referensi yang penulis gunakan sehingga membantu penulis untuk menyelesaikan makalah IF2211 ini. Terakhir, penulis berterima kasih kepada semua pihak teman dan kolega yang menjadi partner diskusi dalam membantu penulis menyusun makalah ini dari awal hingga akhir.

REFERENCES

[1] G Putsanra, Dipna Videlia. "Apa Itu PPDB: Download Juknis PPDB 2022/2023 SMP dari Kemdikbud". https://tirto.id/apa-itu-ppdb-download-juknis-ppdb-2022-2023-smp-dari-kemdikbud-grV3 [Diakses 22/05/2023 01.14 WIB]

[2] Tim Komunikasi Pemerintah Kemenkominfo dan Biro Komunikasi dan Layanan Masyarakat Kemendikbud. 2018. https://www.kominfo.go.id/content/detail/13689/semua-bisa-sekolah-zonasi-untuk-pemerataan-yang-berkualitas/0/artikel_gpr [Diakses 22/05/2023 01.14 WIB]

[3] Munir, Rinaldi, 2021. Penentuan rute (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf [Diakses 22/05/2023 01.14 WIB]

[4] Munir, Rinaldi, 2021. Penentuan rute (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf [Diakses 22/05/2023 01.14 WIB]

[5] Kettle, Simon, 2017. *Distance on a sphere : The Haversine Formula*. https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128 [Diakses 22/05/2023 01.14 WIB]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Mohammad Farhan Fahrezy
13521106