

Perbandingan Algoritma Pencarian Jalur A-Bintang dan Dijkstra Pada Lima Belas Peta Berbasis Heksagonal Menggunakan Godot Game Engine

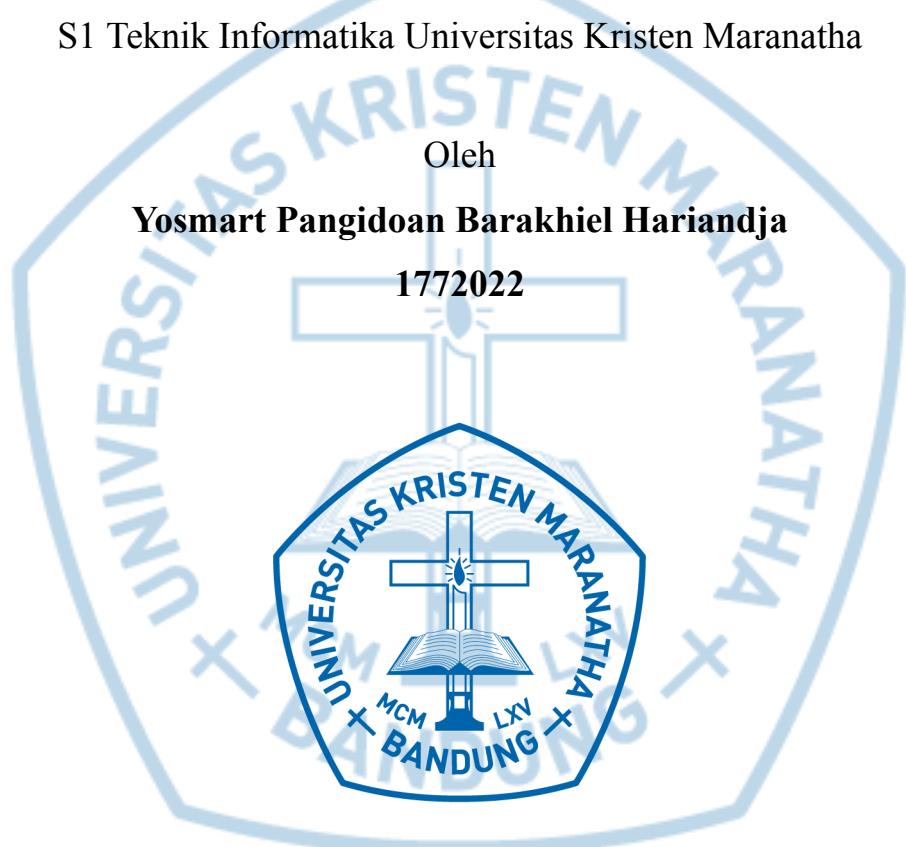
TUGAS AKHIR

Diajukan untuk Memenuhi Persyaratan Akademik dalam
Menyelesaikan Pendidikan pada Program Studi
S1 Teknik Informatika Universitas Kristen Maranatha

Oleh

Yosmart Pangidoan Barakhiel Hariandja

1772022



**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN MARANATHA
BANDUNG**

2023

LEMBAR PENGESAHAN

Perbandingan Algoritma Pencarian Jalur A-Bintang dan Dijkstra Pada Lima Belas Peta Berbasis Heksagonal Menggunakan Godot Game Engine

**Dengan ini, saya menyatakan bahwa
isi CD ROM Laporan Penelitian sama dengan hasil revisi akhir**

Bandung, Tanggal Bulan Tahun

(Yosmart Pangidoan Barakhiel) (1772022)

Erico Darmawan H., S.Kom, M.T. **Nama dan Gelar Dosen**
NIK: 720247 **NIK: NIK Dosen**

Pengaji I **Pengaji II (Jika Ada)**

Nama dan Gelar Dosen	Nama dan Gelar Dosen
NIK: NIK Dosen	NIK: NIK Dosen

Mengetahui,
Ketua Program Studi Teknik Informatika

**Julianti Kasih, S.E., M.Kom. Ka. Prodi
NIK: 720281**

PERNYATAAN ORISINALITAS LAPORAN PENELITIAN

Dengan ini, saya yang bertanda tangan di bawah ini:

Nama : Yosmart Pangidoan Barakhiel Hariandja
NRP : 1772022
Fakultas/ Program Studi : Teknologi Informasi / Teknik Informatika

Menyatakan bahwa laporan penelitian ini adalah benar merupakan hasil karya saya sendiri dan bukan duplikasi dari orang lain.

Apabila pada masa mendatang diketahui bahwa pernyataan ini tidak benar adanya, saya bersedia menerima sanksi yang diberikan dengan segala konsekuensinya.

Demikian pernyataan ini saya buat.

Bandung, Tanggal Bulan Tahun

Yosmat Pangidoan Barakhiel Hariandja

NRP: 1772022

PERNYATAAN PUBLIKASI LAPORAN PENELITIAN

Saya yang bertanda tangan di bawah ini:

Nama : Yosmart Pangidoan Barakhiel Hariandja

NRP : 1772022

Fakultas / Program : Teknologi Informasi / Teknik Informatika
Studi

Dengan ini, saya menyatakan bahwa:

1. Demi perkembangan ilmu pengetahuan, saya menyetujui untuk memberikan kepada Universitas Kristen Maranatha Hak Bebas Royalti non eksklusif (*Non Exclusive Royalty Free Right*) atas laporan penelitian saya yang berjudul *Perbandingan Algoritma Pencarian Jalur A-Bintang dan Dijkstra Pada Lima Belas Peta Berbasis Heksagonal Menggunakan Godot Game Engine*.
2. Universitas Kristen Maranatha Bandung berhak menyimpan, mengalihmediakan / mengalihformatkan, mengelola dalam bentuk pangkalan data (*database*), mendistribusikannya, serta menampilkannya dalam bentuk *softcopy* untuk kepentingan akademis tanpa perlu meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/ pencipta.
3. Saya bersedia dan menjamin untuk menanggung secara pribadi tanpa melibatkan pihak Universitas Kristen Maranatha Bandung, segala bentuk tuntutan hukum yang timbul atas pelanggaran hak cipta dalam karya ilmiah saya ini.

Demikian pernyataan ini saya buat dengan sebenarnya dan untuk dapat dipergunakan sebagaimana mestinya.

Bandung, Tanggal Bulan Tahun

Yosmart Pangidoan Barakhiel

NRP: 1772022

PRAKATA

Isi prakata harus terstruktur, dengan saran isi urutan sebagai berikut:

1. Ucapan syukur kepada Tuhan Yang Maha Esa
2. Penjelasan mengenai adanya tugas karya ilmiah, tujuan subjektif. Contoh: untuk gelar S1/ D3 untuk syarat kelulusan
3. Penjelasan pelaksanaan pembimbing karya ilmiah. Contoh: satu kalimat tentang judul
4. Intro tentang arahan, bimbingan, bantuan dalam penyusunan karya ilmiah (Ucapan terima kasih kepada dosen pembimbing)
5. Ucapan terima kasih kepada pihak-pihak dimulai dari unit tertinggi (Dekan, Ketua Program Studi, Koordinator KP/ TA, dosen-dosen sampai dengan rekan-rekan mahasiswa). Ucapan terima kasih kepada dosen wajib dilengkapi dengan gelarnya.
6. Pernyataan keterbukaan terhadap kritik dan saran
7. Harapan. Contoh: dengan adanya penelitian ini diharapkan
8. Kata mutiara, dll.

Bandung, tanggal bulan tahun

Yosmart
Hariandja

Pangidoan Barakhiel

ABSTRAK

Penelitian ini bertujuan untuk membandingkan kinerja dua algoritma pencarian jalur, yaitu Algoritma A-Bintang dan Algoritma Dijkstra, dalam konteks aplikasi pengujian pencarian jalur berbasis peta heksagonal. Penerapan algoritma ini dilakukan pada lima belas peta berbasis heksagonal menggunakan Godot Game Engine.

Pada setiap peta, dilakukan uji coba menggunakan Algoritma A-Bintang dan Algoritma Dijkstra untuk menemukan jalur optimal antara dua titik yang ditentukan. Godot Game Engine dipilih sebagai platform implementasi untuk menguji kinerja algoritma pencarian jalur. Dengan menggunakan Godot, penelitian ini dapat secara realistik mereplikasi lingkungan permainan yang memanfaatkan peta heksagonal.

Data dikumpulkan dengan mengukur kecepatan tempuh, biaya perjalanan, jumlah node yang dilalui, dan jumlah node yang dikunjungi oleh kedua algoritma. Dengan memperhatikan variabel-variabel tersebut, penelitian ini akan mengidentifikasi tidak hanya efisiensi dari segi waktu eksekusi, tetapi juga efektivitas jalur yang dihasilkan dari kedua algoritma.

Harapannya, hasil penelitian ini memberikan pemahaman yang lebih mendalam mengenai kelebihan dan kekurangan masing-masing algoritma dalam konteks pengujian pencarian jalur.

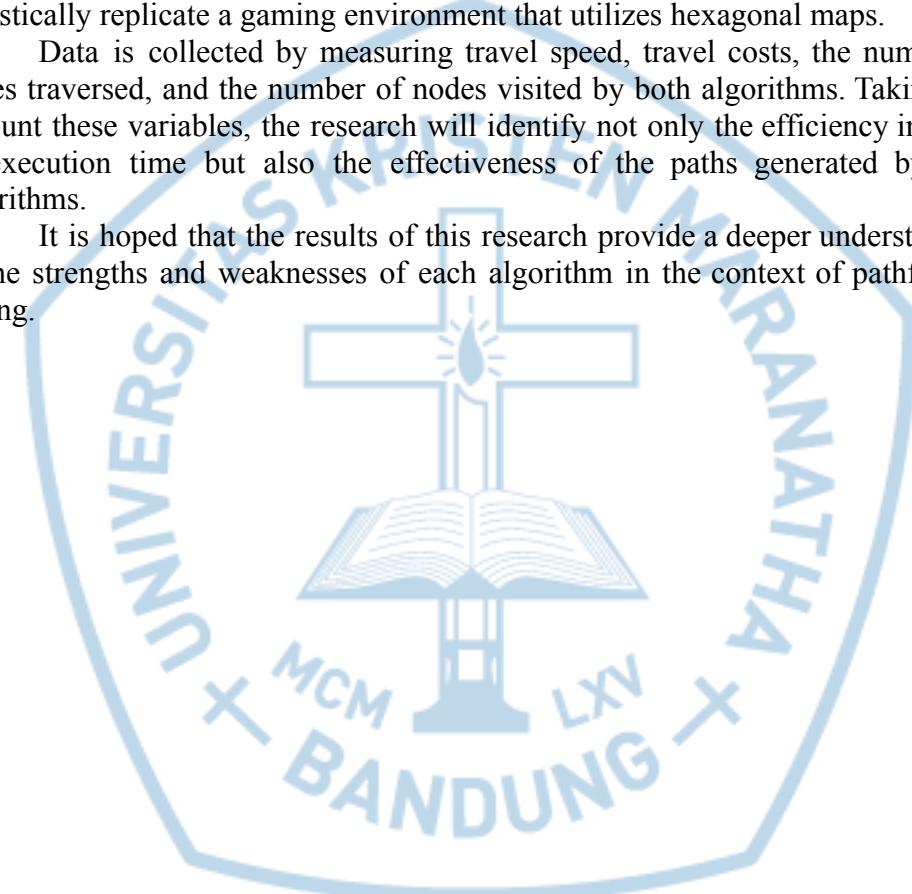
ABSTRACT

This research aims to compare the performance of two pathfinding algorithms, namely the A-Star Algorithm and the Dijkstra Algorithm, in the context of pathfinding testing applications based on hexagonal map. The implementation of these algorithms is carried out on fifteen hexagonal map-based using the Godot Game Engine.

On each map, experiments are conducted using the A-Star Algorithm and the Dijkstra Algorithm to find the optimal path between two specified points. Godot Game Engine is chosen as the implementation platform to test the performance of the pathfinding algorithms. By using Godot, this research can realistically replicate a gaming environment that utilizes hexagonal maps.

Data is collected by measuring travel speed, travel costs, the number of nodes traversed, and the number of nodes visited by both algorithms. Taking into account these variables, the research will identify not only the efficiency in terms of execution time but also the effectiveness of the paths generated by both algorithms.

It is hoped that the results of this research provide a deeper understanding of the strengths and weaknesses of each algorithm in the context of pathfinding testing.



DAFTAR ISI

LEMBAR PENGESAHAN	1
PERNYATAAN ORISINALITAS LAPORAN PENELITIAN	2
PERNYATAAN PUBLIKASI LAPORAN PENELITIAN	3
PRAKATA	4
ABSTRAK	5
ABSTRACT	6
DAFTAR ISI	7
DAFTAR GAMBAR	9
DAFTAR TABEL	11
DAFTAR ISTILAH	13
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan Pembahasan	3
1.4 Ruang Lingkup	3
BAB 2 KAJIAN TEORI	4
2.1 Pencarian Jalur	4
2.2 Graf	4
2.3 Node/Simpul	5
2.4 Edge/Sisi	6
2.5 Dijkstra's Algorithm	6
2.6 A-Bintang	7
BAB 3 ANALISIS DAN RANCANGAN SISTEM	9
3.1 Pengantar	9
3.2 Tipe-Tipe Node	11
3.3 Tipe-Tipe List	11
3.4 Simulasi Algoritma	11
3.4.1 A-Bintang	18
3.4.2 Dijkstra	22
BAB 4 IMPLEMENTASI	23
4.1 Pengantar	23
4.2 Map	25
4.3 Node	26
4.4 Graph	29
4.5 Algoritma	32
4.6 Catatan Hasil	33
BAB 5 PENGUJIAN	34

5.1 Pengantar	34
5.2 Perbandingan Kecepatan Tempuh	37
5.3 Perbandingan Biaya Perjalanan	40
5.4 Perbandingan Node Dilalui	43
5.5 Perbandingan Node Dikunjungi	46
BAB 6 SIMPULAN DAN SARAN	48
6.1 Simpulan	48
6.2 Saran	48
DAFTAR PUSTAKA	49
LAMPIRAN A TABEL UJIAN	
LAMPIRAN B GAMBAR TEKS MAP	



DAFTAR GAMBAR

Gambar 3.2.1 Node Standar, Tembok, Tujuan.	10
Gambar 3.2.2 Node Awal, Node Saat Ini, Node Kuning.	11
Gambar 3.4.1 Algoritma A-Bintang	18
Gambar 3.4.2 Algoritma Dijkstra	22
Gambar 4.2.1 Map Data Text	23
Gambar 4.2.2 Text ke File	24
Gambar 4.2.3 File ke Map	25
Gambar 4.3 Kode Node	28
Gambar 4.4 Kode Graph	29
Gambar 4.5 Kode Algoritma	31
Gambar 4.5 Kode Priority Enqueue	32
Gambar 4.6 Kode Priority Enqueue	33
Gambar 5.2 Fungsi Perhitungan Kecepatan Tempuh	34
Gambar 5.3 Program Mengetahui Biaya Perjalanan	37
Gambar 5.4 Program Mengetahui Banyaknya Node Yang Dilalui	41
Gambar 5.5 Mengetahui Jumlah Node Yang Dikunjungi	44

LAMPIRAN B GAMBAR TEKS MAP

DAFTAR TABEL

Tabel 5.2 Perbandingan Kecepatan Tempuh	37
Tabel 5.3 Perbandingan Biaya Perjalanan	40
Tabel 5.4 Perbandingan Node Dilalui	43
Tabel 5.5 Perbandingan Node Dikunjungi	46

LAMPIRAN A TABEL UJIAN



DAFTAR ISTILAH

Game Engine	Kerangka perangkat lunak yang dirancang utamanya untuk pengembangan permainan video dan umumnya mencakup pustaka kode yang relevan serta program-program pendukung.
Godot Game Engine.	Game Engine bernama Godot. Popular karena salah satu pemrograman yang digunakan hampir serupa dengan bahasa pemrograman Python.
GDScript	Bahasa pemrograman Godot Game Engine.
Pathfinding	Proses menemukan jalur dari satu titik ke titik lainnya.
Node	Unit dasar dari pembentukan graf: sebuah graf terdiri dari sebuah himpunan simpul dan sebuah himpunan edges.
Edges	Koneksi antara simpul satu dan lainnya.
G-value	Biaya dari node pertama di simpul ke node yang sedang dicari.
H-value	Biaya dari node yang sedang dicari kepada node tujuan.
F-value	G-value ditambah h-value.
array	Tipe data untuk menyimpan himpunan data.

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Ketika area yang ditentukan untuk ditempuh tidak jelas jalannya, sebuah agen diperlukan untuk merencanakan dan mempertimbangkan rangkaian tindakan: membentuk suatu jalur menuju area yang ditentukan itu. Agen seperti ini disebut agen pencarian jalur atau pathfinding [1]. Pathfinding adalah proses menemukan jalur dari satu titik ke titik lainnya [2]. Jalur ini dilakukan pada sebuah graf - struktur data non-linear yang terdiri dari simpul (vertices) dan sisi-sisi (edges) [3]. Agen pencarian berbasis graf ini perlu mengetahui lokasi-lokasi yang ditentukan dan juga hubungan antara lokasi yang satu itu dengan yang lainnya [4].

Salah satu cara untuk mengetahui hubungan antar lokasi adalah dengan menggambar grid, di mana setiap simpul ditempatkan pada koordinat bidang area. Layar komputer dapat dianggap sebagai grid bidang area, dengan setiap piksel ditempatkan pada koordinat tertentu [5]. Untuk menggambarkan bidang area pada layar komputer, dapat dilakukan dengan menggambar grid heksagonal untuk setiap bidang area [6]. Karena setiap piksel pada layar komputer dapat dimanipulasi, menggunakan program seperti Godot Game Engine memungkinkan programmer untuk mengubah piksel-piksel ini sehingga membentuk grid heksagonal sesuai dengan pilihan area yang diinginkan. Dalam konteks penelitian yang akan dilakukan, dibutuhkan pembentukan bidang-bidang area ini untuk membentuk peta medan. Manipulasi bidang area ini akan dijadikan sebagai dasar untuk menciptakan 15 peta medan yang berbeda, di mana nantinya proses pencarian jalur akan dijalankan.

A-Bintang dan algoritma Dijkstra merupakan agen algoritma pathfinding. Algoritma Dijkstra mengatasi tindakan pencarian jalur ini dengan menentukan biaya edges termurah dari node awal ke node yang sedang dicari [1]. Biaya edges adalah biaya pergerakan atau biaya aksi saat pencarian jalur melalui suatu node [1][7]. Algoritma A-Bintang merupakan perkembangan dari algoritma Dijkstra,

yang tidak hanya menentukan biaya terbaik dari node awal ke node yang sedang dicari (g -value), tetapi juga menggunakan biaya estimasi dari node yang sedang dicari ke node tujuan (h -value) [1]. Fungsi penentuan biaya estimasi pada grid graf berbentuk heksagonal, tempat setiap aksi atau gerakan memungkinkan enam arah pergerakan, adalah Manhattan Distance [8].

1.2 Rumusan Masalah

Untuk membandingkan antara algoritma Dijkstra dan algoritma A-Bintang, parameter yang dapat dibandingkan meliputi [9]:

- Kecepatan Tempuh: Waktu yang diperlukan oleh algoritma untuk menemukan jalur optimal.
- Biaya Perjalanan: Biaya yang harus dikeluarkan oleh algoritma saat melewati bidang area.
- Node Dilalui: Jumlah simpul atau node yang dilewati oleh algoritma selama proses pencarian jalur.
- Node Dikunjungi: Jumlah simpul atau node yang dikunjungi oleh algoritma selama proses pencarian jalur.

Masalah yang diteliti adalah sebagai berikut:

1. Kecepatan Tempuh: Bagaimana menentukan parameter kecepatan tempuh terbaik dengan memprioritaskan waktu yang paling cepat dalam pencarian jalur?
2. Biaya Perjalanan: Bagaimana menentukan parameter biaya perjalanan terbaik dengan mempertimbangkan biaya yang paling rendah dalam pencarian jalur?
3. Node Dilalui: Bagaimana menentukan parameter node yang sudah dilalui terbaik dengan memprioritaskan jumlah yang paling kecil dalam pencarian jalur?
4. Node Dikunjungi: Bagaimana menentukan parameter node yang sudah dikunjungi terbaik dengan memprioritaskan jumlah yang paling kecil dalam pencarian jalur?

1.3 Tujuan Pembahasan

Dalam upaya menjawab masalah yang dihadapi, metode yang diterapkan adalah sebagai berikut:

1. Kecepatan Tempuh: Pencarian parameter kecepatan tempuh terbaik dilakukan dengan mempertimbangkan waktu yang paling cepat.
2. Biaya Perjalanan: Pencarian parameter biaya perjalanan terbaik dilakukan dengan mempertimbangkan biaya yang paling rendah.
3. Node Dilalui: Pencarian parameter node yang sudah dilalui terbaik dilakukan dengan mempertimbangkan jumlah yang paling kecil.
4. Node Dikunjungi: Pencarian parameter node yang sudah dikunjungi terbaik dilakukan dengan mempertimbangkan jumlah yang paling kecil.

1.4 Ruang Lingkup

Berikut pembatasan dari penelitian ini:

- Penelitian ini hanya akan berfokus pada perbandingan antara algoritma pencarian A-Bintang dan Dijkstra.
- Implementasi kedua algoritma A-Bintang dan Dijkstra dilakukan pada Godot Game Engine.
- Pengujian dilakukan pada lima belas sampel peta yang sudah dibentuk sebelumnya.
- Parameter pengujian akan mencakup waktu eksekusi, panjang jalur, node yang dilalui, dan node yang dikunjungi.
- Metrik performa yang digunakan untuk menyelesaikan permasalahan pertama dan kedua adalah evaluasi kuantitatif.
- Interpretasi visual dilakukan pada empat awal dan akhir pada peta, yaitu: atas-kiri, atas-kanan, bawah-kanan, dan bawah-kiri.

BAB 2

KAJIAN TEORI

2.1 Pencarian Jalur

Algoritma pencarian jalur, juga dikenal sebagai algoritma pencarian graf, adalah algoritma komputasi yang dirancang untuk menemukan jalur atau solusi dari satu titik ke titik lain dalam graf [10][11]. Graf ini biasanya merepresentasikan struktur data yang terdiri dari node (simpul) yang terhubung oleh edge (sisi atau hubungan) [11]. Tujuannya adalah untuk menemukan jalur atau urutan node yang menghubungkan dua node tertentu, yaitu node awal dan node tujuan, dengan mempertimbangkan berbagai kendala, biaya, atau aturan tertentu yang mungkin ada [1].

Beberapa contoh pencarian jalur termasuk [1]:

- **Breadth-First Search (BFS):** Algoritma ini menjelajahi graf secara berlapis.
- **Depth-First Search (DFS):** Algoritma ini menjelajahi graf secara vertikal, mencoba satu cabang terlebih dahulu sebelum kembali.
- **Dijkstra's Algorithm:** Algoritma ini mencari jalur terpendek berdasarkan biaya g-value terendah.
- **A-Bintang:** Algoritma ini menggabungkan biaya sejauh ini (g-value) dan nilai heuristik (h-value) untuk memilih jalur terbaik.

2.2 Graf

Graf dalam konteks algoritma pencarian jalur mengacu pada representasi visual atau matematis dari kumpulan simpul (node) dan sisi (edge) di antara simpul-simpul tersebut [4][12]. Graf digunakan untuk merepresentasikan struktur ruang atau lingkungan yang ingin dijelajahi [4].

Simpul dalam graf berbentuk grid biasanya mewakili lokasi atau titik dalam lingkungan yang ingin dijelajahi, sedangkan sisi (edge) antara

simpul-simpul merepresentasikan keterhubungan atau jarak antara lokasi-lokasi tersebut [5][13]. Algoritma pencarian jalur, seperti algoritma Dijkstra dan algoritma A-Bintang, bekerja pada graf ini untuk menemukan jalur terpendek atau jalur optimal dari satu simpul ke simpul lainnya.

Graf dapat memiliki berbagai atribut tambahan pada simpul dan sambungan, seperti bobot pada sambungan, yang mewakili biaya atau jarak antara simpul-simpul [4][14].

2.3 Node/Simpul

Node dalam konteks algoritma pencarian jalur merujuk pada titik-titik individual dalam ruang pencarian yang sedang dijelajahi [7]. Ruang pencarian adalah representasi abstrak bidang tiga dimensi dari semua kemungkinan keadaan atau konfigurasi yang dapat diakses oleh algoritma pencarian dalam upaya untuk menemukan jalur atau solusi tertentu. Sebuah node biasanya direpresentasikan sebagai struktur data atau objek yang menyimpan informasi penting tentang titik atau keadaan tertentu dalam ruang pencarian [7]. Informasi ini dapat mencakup posisi, biaya atau g-value, heuristik atau h-value (jika digunakan), dan referensi ke node tetangga (anak-anak atau tetangga terhubung) [1].

G-Value (Biaya Sejauh Ini): G-value adalah jarak atau biaya total yang diperlukan dari node awal ke node saat ini.

H-Value (Biaya Heuristik): H-Value adalah perkiraan jarak atau biaya dari node saat ini ke node tujuan.

Node Tetangga: Node tetangga adalah kandidat node yang akan dijelajahi selanjutnya oleh algoritma.

Node Awal dan Node Tujuan: Node awal adalah titik mulai dalam ruang pencarian, sedangkan node tujuan adalah tujuan akhir yang ingin dicapai oleh algoritma.

Node Saat Ini: Node saat ini adalah node yang sedang dievaluasi atau dianalisis oleh algoritma.

Node Bermedan: Node yang memiliki extra biaya dari biaya normalnya.

2.4 Edge/Sisi

Sisi atau Edge, dalam konteks algoritma pencarian jalur, adalah koneksi atau hubungan antara dua node dalam graf atau peta yang digunakan. Sisi atau edge pada pencarian jalur, memiliki atribut biaya atau jarak, yang menggambarkan seberapa biaya atau seberapa jauh jarak langkah tersebut.

2.5 Dijkstra's Algorithm

Algoritma Dijkstra, yang dinamai sesuai dengan nama matematikawan Belanda Edsger W. Dijkstra, adalah algoritma pencarian jalur atau rute yang digunakan untuk menemukan jalur terpendek dari satu node (simpul) ke semua node lain dalam graf yang berbobot [16].

Berikut adalah inti dari algoritma Dijkstra [15]:

1. **Inisialisasi:** Mulai dari node awal pada graf. Biaya dari node awal ke dirinya sendiri adalah 0, dan biaya ke node lainnya yang belum diinisialisasi atau dihitung dijadikan tak terhingga (infinity).
2. **Perluas Jalur:** Selama pencarian, algoritma memperluas jalur dari node awal ke node tetangga yang belum dieksplorasi. Saat memperluas jalur, algoritma akan melakukan perhitungan biaya g-value. Jika biaya yang dihitung lebih rendah dari biaya sebelumnya, maka biaya tetangga diperbarui.
3. **Pilih Node Terpendek:** Dijkstra kemudian memilih node dengan biaya terendah dari node-node perluasan itu, dan menjadikannya sebagai node saat ini.
4. **Tandai Node Sebagai Dieksplorasi:** Setelah biaya terpendek ke node saat ini ditemukan, node tersebut ditandai sebagai "dieksplorasi" sehingga tidak akan dievaluasi lagi di pengulangan algoritma berikutnya.
5. **Ulangi Langkah 2-4:** Proses ini diulangi untuk setiap node, hingga semua node sudah dieksplorasi atau ketika node tujuan sudah ditemukan.

Hasil dari algoritma Dijkstra adalah sebuah pohon jalur terpendek dari node awal ke semua node lain dalam graf [15]. Hasil ini juga mencakup pohon jalur pencarian terhadap node tujuan yang telah ditentukan.

2.6 A-Bintang

A* (dibaca "A-Bintang") adalah algoritma pencarian jalur yang digunakan untuk menemukan jalur efektif dari satu simpul ke simpul tujuan dalam graf berbobot. Algoritma ini memiliki kesamaan dengan algoritma Dijkstra dalam penggunaan biaya sejauh ini (g -value), perbedaannya adalah juga mempertimbangkan biaya estimasi dari node yang dicari ke node tujuan (h -value) untuk meningkatkan efisiensi dan kemampuan dalam menemukan jalur terpendek dalam waktu yang lebih singkat [17].

Inti dari algoritma A-Bintang adalah sebagai berikut [17]:

1. **Inisialisasi:** Inisialisasi dimulai dari simpul awal dalam graf. Biaya dari simpul awal ke dirinya sendiri adalah 0, sementara biaya ke simpul lain yang belum diinisialisasi atau dihitung diatur menjadi tak terhingga (∞). Selain itu, pertimbangkan juga nilai heuristik (h -value) dari simpul awal ke simpul tujuan.
2. **Perluas Jalur:** Selama pencarian, algoritma memperluas jalur dari simpul awal ke simpul tetangga yang belum dieksplorasi. Saat memperluas jalur, algoritma akan melakukan perhitungan biaya g -value ditambah h -value. Jika biaya yang dihitung lebih rendah dari biaya sebelumnya, maka biaya tetangga diperbarui.
3. **Pilih Node Terbaik:** A-Bintang kemudian memilih node dengan nilai f -value (g -value + h -value) terendah sebagai node saat ini.
4. **Tandai Node Sebagai Dieksplorasi:** Setelah node saat ini sudah dipilih, node tersebut ditandai sebagai "dieksplorasi" sehingga tidak akan dievaluasi lagi di pengulangan algoritma berikutnya.
5. **Ulangi Langkah 2-4:** Proses ini diulangi hingga node tujuan ditemukan.

Hasil dari algoritma A-Bintang adalah jalur efektif dari simpul awal ke simpul tujuan dalam graf berbobot. Keefektifan ini, dilihat dari kemampuan algoritma A-Bintang untuk memotong jalur yang tidak akan menghasilkan solusi yang lebih baik dari yang telah ditemukan, dikarenakan adanya biaya estimasi h-value yang memandu pencarian jalur [17].

2.7 Menghitung Jarak

Pengukuran jarak adalah skor objektif yang merangkum perbedaan relatif antara dua objek dalam suatu domain masalah [19]. Dalam penelitian ini, node dianggap sebagai objek itu sendiri, dan graf yang mencakup node-node ini dianggap sebagai domainnya. Untuk melakukan pencarian jalur, langkah pertama yang perlu dicari atau dihitung adalah perbedaan relatif antara node-node tersebut. Oleh karena itu, perhitungan jarak perlu dilakukan.

Dua formula pengukuran jarak yang umum digunakan dalam algoritma pencarian jalur adalah sebagai berikut:

- Jarak Euclidean: $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$.
- Jarak Manhattan: $\text{abs}((x1 - x2) + (y1 - y2))$.

Jarak Euclidean digunakan saat menghitung jarak antara dua baris data yang memiliki nilai numerik, seperti nilai floating point atau integer [19]. Gunakan Jarak Manhattan sebagai gantinya untuk dua vektor dalam ruang fitur berupa bilangan bulat, ketika vektor-vektor tersebut menggambarkan objek pada grid yang seragam, seperti papan catur atau blok-blok kota [19]. Karena perbedaan setiap node tetap konstan, seperti pada papan catur, hanya saja menggunakan blok-blok segi enam, penelitian ini menggunakan metode Jarak Manhattan pada kedua algoritma pencarian jalur.

BAB 3

ANALISIS DAN RANCANGAN SISTEM

3.1 Pengantar

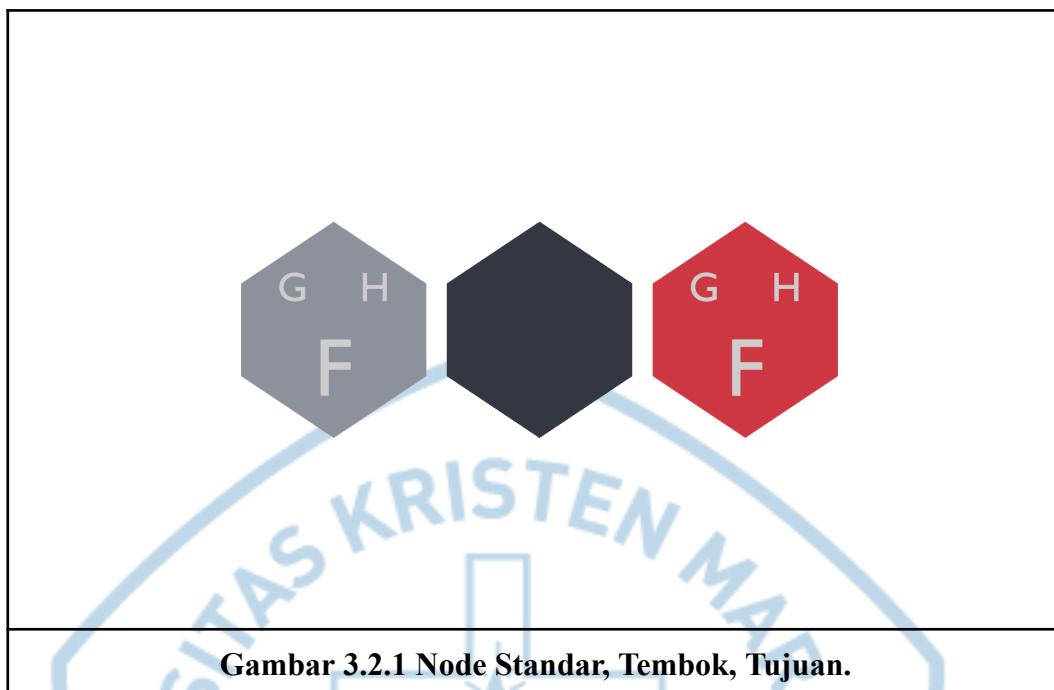
Bab ini membahas rancangan sistem serta analisis yang akan diimplementasikan dalam penelitian "Perbandingan Algoritma Pencarian Jalur A-Bintang dan Dijkstra Pada Lima Belas Peta Berbasis Heksagonal Menggunakan Godot Game Engine." Bab ini akan eksplorasi konsep-konsep dasar terkait dengan algoritma pencarian jalur, seperti node, list, dan simulasi algoritmanya.

3.2 Tipe-Tipe Node

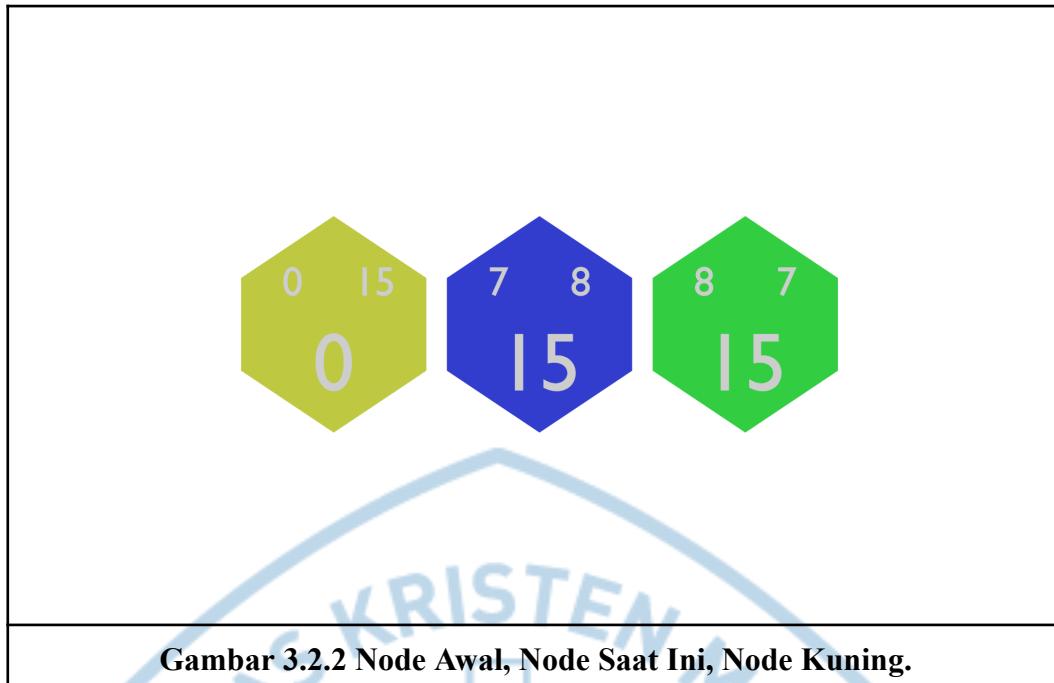
Perancangan sistem memanfaatkan beberapa jenis node. Sebagai contoh, pada **Gambar 3.2.1**, terdapat tiga kategori node dengan warna yang berbeda: abu-abu, hitam, dan merah. Node berwarna abu-abu menandakan bahwa node tersebut belum ditemukan oleh algoritma selama proses pencarian jalur berlangsung, sementara node berwarna hitam menunjukkan bahwa node tersebut tidak dapat dilalui oleh algoritma dalam pencarian jalur. Node berwarna merah menjadi penanda bahwa node tersebut merupakan tujuan akhir bagi algoritma dalam pencarian jalur hingga mencapai node tersebut.

Node yang belum diatur adalah node yang belum memiliki atribut g-value, h-value, dan f-value, yang merupakan hasil pertambahan dari dua g-value dan h-value. Setiap algoritma menentukan bagaimana atribut-atribut tersebut diatur. Sebuah node dianggap sudah diatur jika atribut-atribut tersebut telah ditetapkan. Node berwarna abu-abu merupakan node standar yang belum ditemukan oleh algoritma dan sedang menunggu untuk diproses serta diatur atribut-atributnya. Node berwarna hitam merupakan node tembok yang tidak dapat dilewati oleh algoritma pencarian jalur. Node hitam ini tidak akan menerima aturan atribut karena node tembok berfungsi sebagai halangan dalam peta grid, sedangkan node berwarna merah adalah node tujuan yang ingin dicari oleh kedua algoritma. Jika

algoritma berhasil menemukan node tujuan atau node merah, maka pencarian jalur dianggap selesai.



Jika **Gambar 3.2.1** menunjukkan node pada grid yang belum diproses atau diatur, maka **Gambar 3.2.2** menggambarkan node yang sedang atau sudah diproses dan diatur. Node berwarna kuning adalah node awal pertama yang dipilih sebagai posisi awal pencarian jalur algoritma, sehingga semua node yang dicari akan memiliki induk yang akan menunjuk ke node awal kuning ini. Node berwarna biru merupakan node yang dipilih untuk nantinya proses pencarian node-node tetangga dan pencarian jalur terbaik menurut algoritma ditentukan selama proses pencarian jalur sedang dilakukan. Jika algoritma A-Bintang digunakan, node biru yang dipilih harus memiliki biaya-f terendah. Sedangkan algoritma BFS memilih node biru sebagai node pertama yang dimasukkan ke dalam frontier list. Node awal kuning memiliki sifat yang sama dengan node biru terpilih. Node berwarna hijau merupakan node tetangga dari node yang sudah dipilih. Node ini akan ditemukan saat node terpilih selesai diproses.



3.3 Tipe-Tipe List

Frontier List: Frontier list adalah daftar (biasanya dalam bentuk antrian atau tumpukan) yang menyimpan node-node yang sedang dalam proses atau yang akan dieksplorasi dalam langkah-langkah selanjutnya.

Reached List: Reached list adalah daftar yang menyimpan node-node yang sudah dieksplorasi dan diproses.

Tipe list yang digunakan pada kedua list ini adalah array list Godot. Ini merupakan struktur data yang memungkinkan penyimpanan elemen secara dinamis, memungkinkan perubahan ukuran list sesuai kebutuhan.

3.4 Simulasi Algoritma

Untuk menyederhanakan simulasi rancangan sistem, simulasi akan dilakukan dengan mengadakan beberapa kondisi agar kedua algoritma A-Bintang dan Dijkstra memiliki medan grid pencarian jalur yang sama. Berikut kondisinya adalah sebagai berikut:

- **Kondisi pertama** adalah pemilihan posisi awal atau node kuning hanya dilakukan sekali dan digunakan oleh kedua algoritma tersebut. Hal ini dilakukan agar pembaca tahu bahwa A-Bintang dan Dijkstra memulai dari posisi yang sama. Hal yang sama berlaku untuk node merah tujuan.
- **Kondisi kedua** adalah setelah memilih node biru terpilih, semua node tetangga hijau harus ditemukan terlebih dahulu. Beberapa node akan mengubah atribut f-value-nya, entah itu bertambah maupun mengurang.
- **Kondisi ketiga** adalah algoritma Dijkstra memilih node biru terpilih berdasarkan f-value. Pada algoritma Dijkstra, f-value hanya terdiri dari g-value yang merupakan jumlah langkah yang sudah dilalui. Jika ada dua node dengan g-value yang sama, node pertama yang dimasukkan ke dalam frontier list akan dipilih.
- **Kondisi keempat** adalah node hijau tetangga akan diproses ulang oleh kedua algoritma jika g-value tambah satu dari node biru terpilih ditambah dengan h-value dari node hijau tetangga, lebih kecil dari f-value dari node hijau tetangga tersebut. Dan yang diproses ulang hanya node yang bertetangga dengan node biru terpilih.
- **Kondisi kelima** adalah node biru terpilih tidak akan diproses ulang setelah dimasukkan ke dalam reached list. Reached list merupakan list tempat penyimpanan node yang sudah diproses.
- **Kondisi keenam** adalah node yang dimasukkan ke dalam frontier list diurutkan berdasarkan posisi geografisnya, dimulai dari node yang berada di sebelah barat node yang dipilih. Dengan demikian, node yang berada di sebelah kiri akan menjadi yang pertama dalam urutan pada frontier list
- **Kondisi ketujuh** adalah pemilihan node biru terpilih dari node-node yang berada di dalam frontier list dilakukan dengan mencari f-value terendah. Jika terdapat beberapa node dengan f-value yang sama, maka dipilih node dengan h-value terendah. Jika h-value juga sama, maka dipilih node dengan g-value terendah. Jika masih terdapat kesamaan, maka node biru terpilih, dipilih dari node yang pertama kali dimasukkan ke dalam frontier list berdasarkan kondisi ketujuh.

3.4.1 A-Bintang

Berikut penjelasan proses simulasi algoritma pencarian jalur A-Bintang:

Langkah 1.

Seperti pada **Gambar 3.4.1**, dalam **Langkah 1**, ditentukan node kuning sebagai node awal dan node merah sebagai node tujuan. Node kuning awal memiliki g-value yang diatur menjadi nol karena belum pernah dilangkahi, dan h-value diatur menjadi empat menggunakan fungsi estimasi heuristik manhattan distance. Manhattan distance, berfungsi untuk menghitung jarak node awal ke node tujuan. Kemudian, nilai h-value ini juga digunakan sebagai hasil perhitungan f-value, yang merupakan hasil penjumlahan g-value dan h-value.

Langkah 2.

Karena node kuning awal memiliki sifat yang sama dengan node biru terpilih, pada **Langkah 2**, node awal menjadi node terpilih pertama yang dimasukkan ke dalam reached list. Langkah selanjutnya adalah mencari node abu-abu standar di sekitar node kuning awal yang dapat ditemukan, dan mengubah node abu-abu standar tersebut menjadi node hijau tetangga serta memproses atribut biayanya. Pada contoh **Gambar 3.2.1**, dalam **Langkah 2**, terdapat tiga node hijau tetangga yang ditemukan. F-value dari node-node tersebut pada frontier list diurutkan menjadi 4, 4, 5.

Langkah 3.

Langkah 3 yang dilakukan oleh algoritma A-Bintang setelah menemukan node hijau tetangga di sekitar node kuning awal yang dapat ditemukan adalah memilih node biru terpilih untuk melanjutkan proses pencarian jalur. Dengan cara memilih node pertama yang berada dalam frontier list yang sudah diurutkan dari f-value terkecil hingga terbesar, sebagai node biru terpilih. Sesuai dengan **kondisi ketujuh**, yaitu pemasukan node ke dalam frontier list dimulai dari node yang berada di sebelah barat node terpilih, maka node dengan f-value empat yang berada di sebelah kiri node kuning awal dipilih menjadi node biru terpilih.

Langkah 4.

Setelah node biru terpilih dipilih sebagai node terbaik, langkah selanjutnya adalah melakukan proses yang serupa dengan **Langkah 2** pada node kuning awal.

Yaitu dengan mencari node abu-abu standar di sekitar node biru terpilih, mengubahnya menjadi node hijau tetangga, dan mengatur atributnya dengan menghitung g-value, h-value, dan f-value.

Langkah 5.

Namun, berbeda dengan **Langkah 3** saat memilih node hijau tetangga menjadi node biru terpilih saat node yang dipilih adalah node kuning awal, pada **Langkah 5** ini atribut yang dimiliki oleh kedua node hijau tetangga tidaklah sama persis. Salah satu node memiliki h-value sebesar dua sedangkan yang lain memiliki h-value sebesar tiga. Dengan mengikuti **kondisi kedelapan**, node yang memiliki h-value sebesar dua akan dipilih untuk node pencarian jalur berikutnya.

Langkah 6-7.

Setelah itu, dari **Langkah 6-7** mengulang **Langkah 2-3**.

Langkah 8.

Namun, pada **Langkah 8**, terjadi kasus yang sesuai dengan **kondisi kelima**. Hal ini terjadi ketika node biru yang baru saja dipilih menawarkan jalur yang lebih pendek untuk node hijau tetangga yang memiliki f-value sebesar lima. Oleh karena itu, node tersebut diproses ulang.

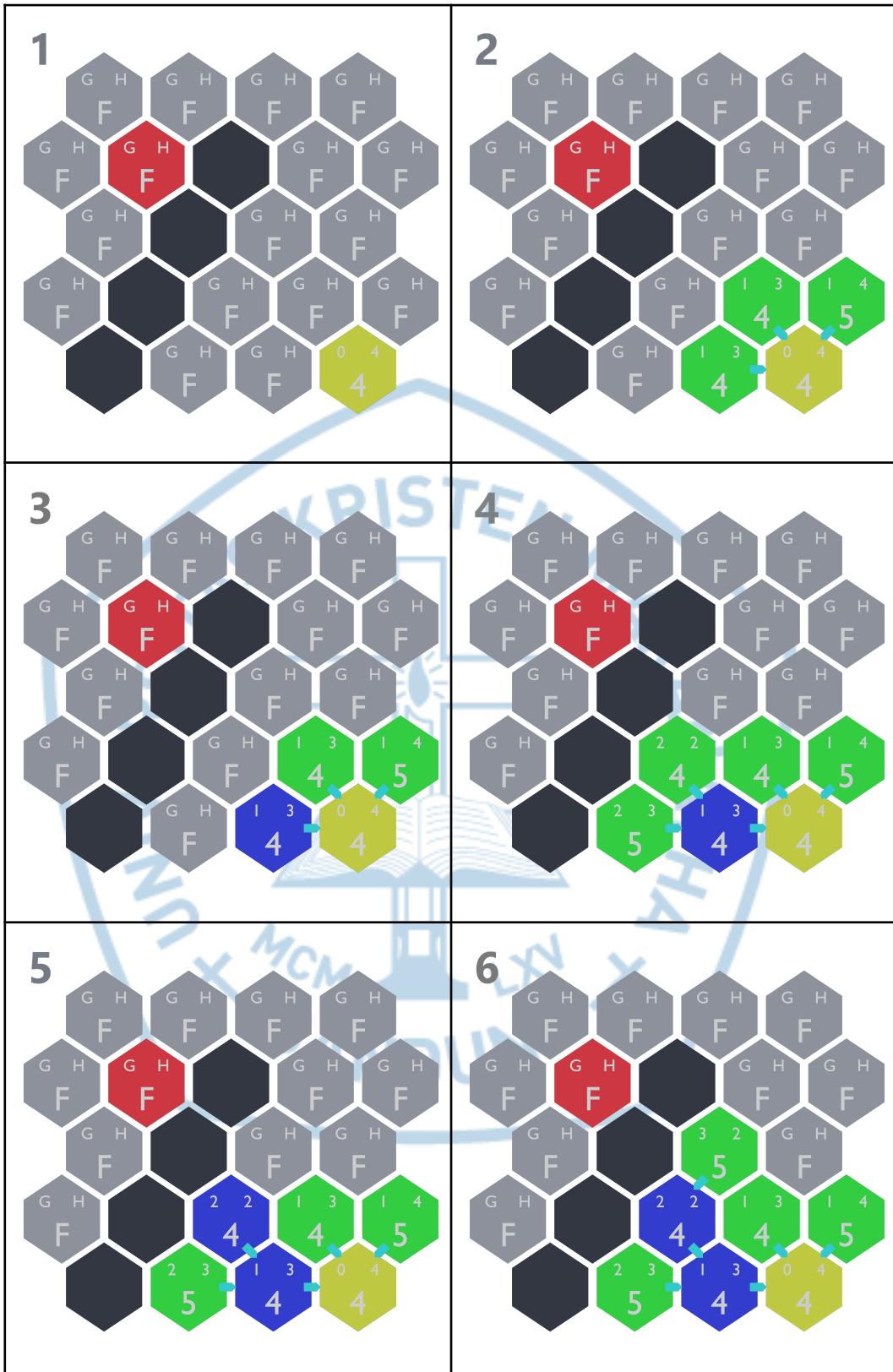
Langkah 9 - 19.

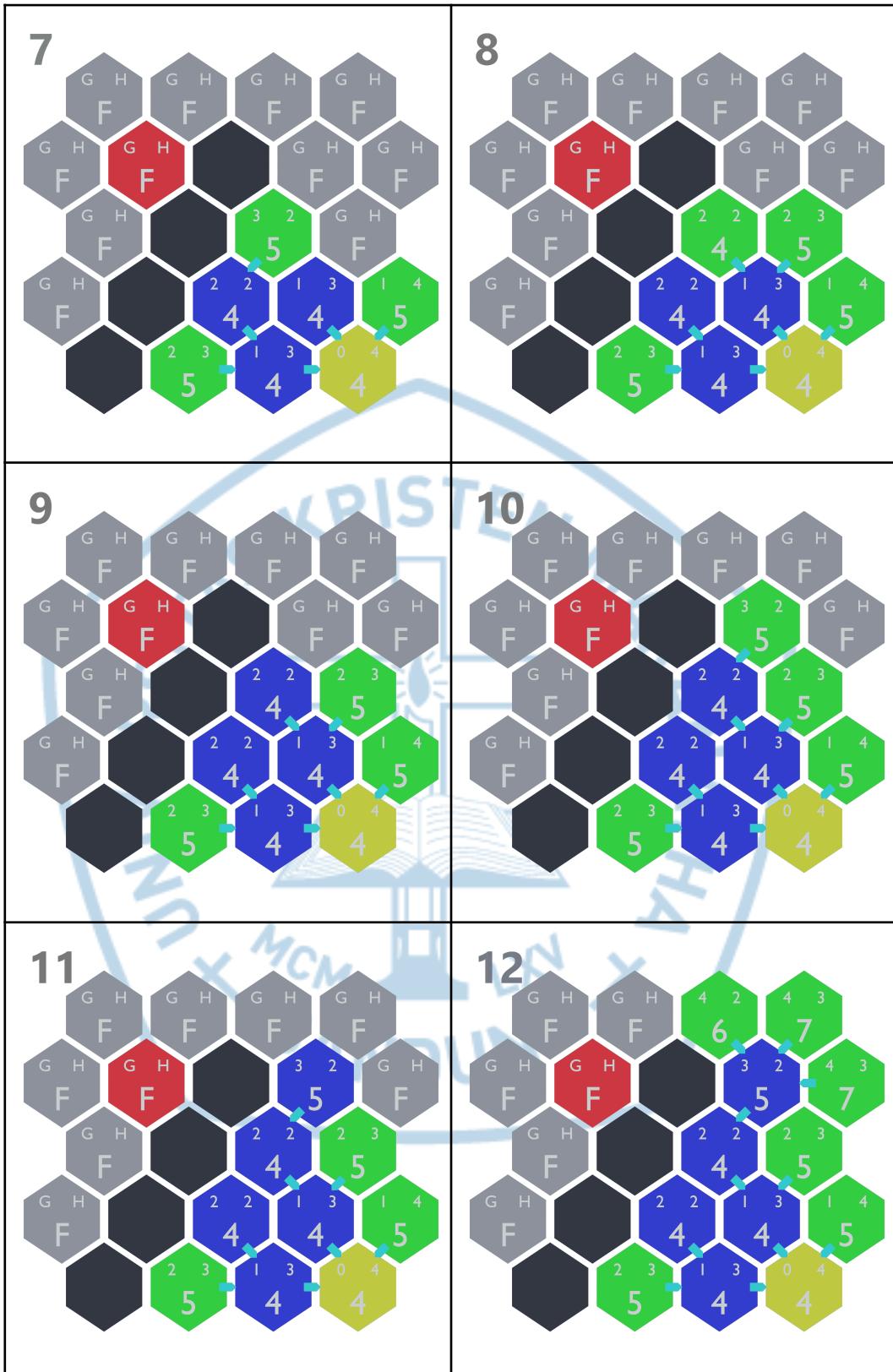
Dari langkah kesembilan hingga kesembilan belas sama proses pencarian jalurnya. Terjadi pengulangan **Langkah 1-5**, ditambah dengan **Langkah 8**.

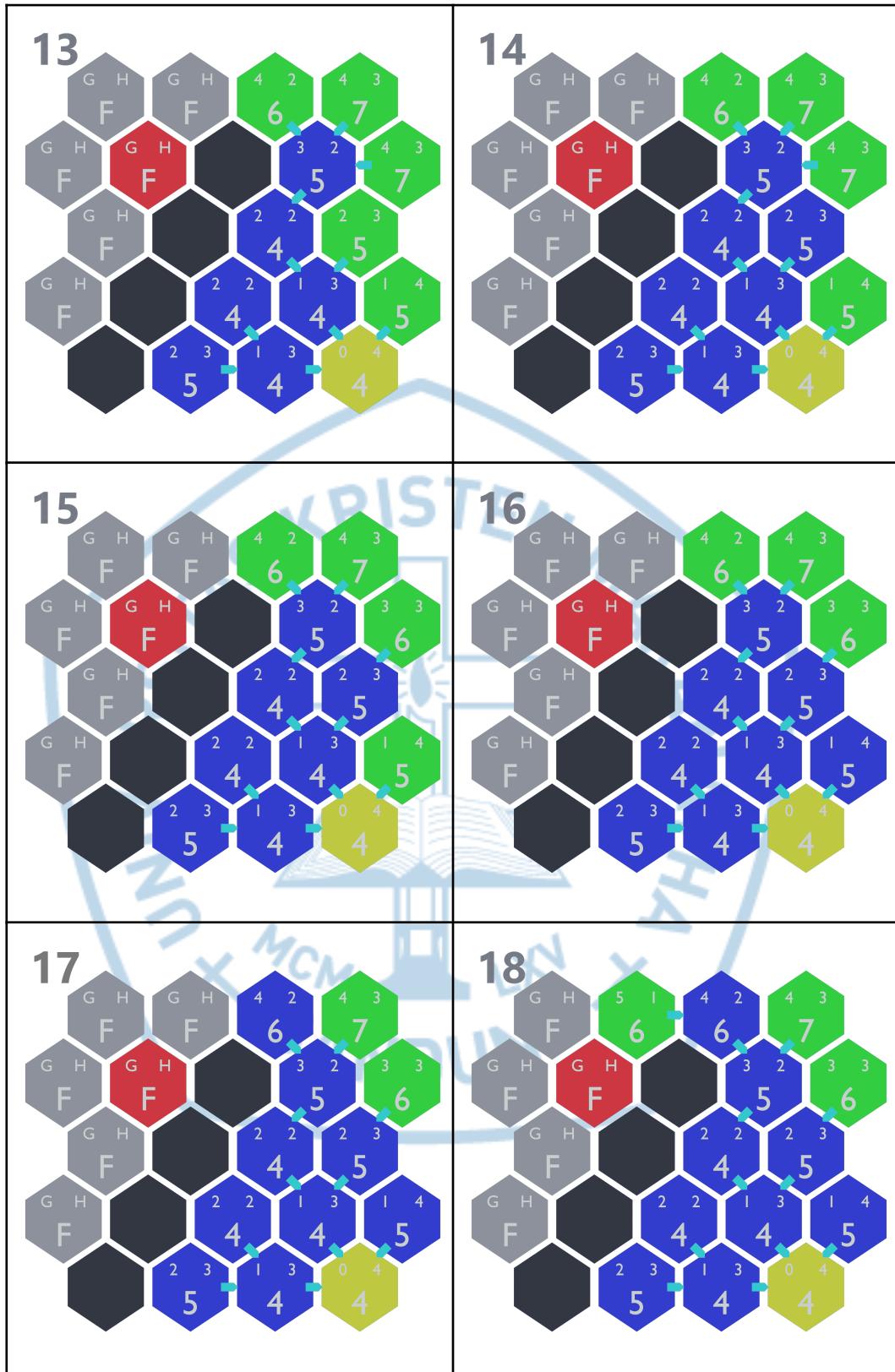
Langkah 20.

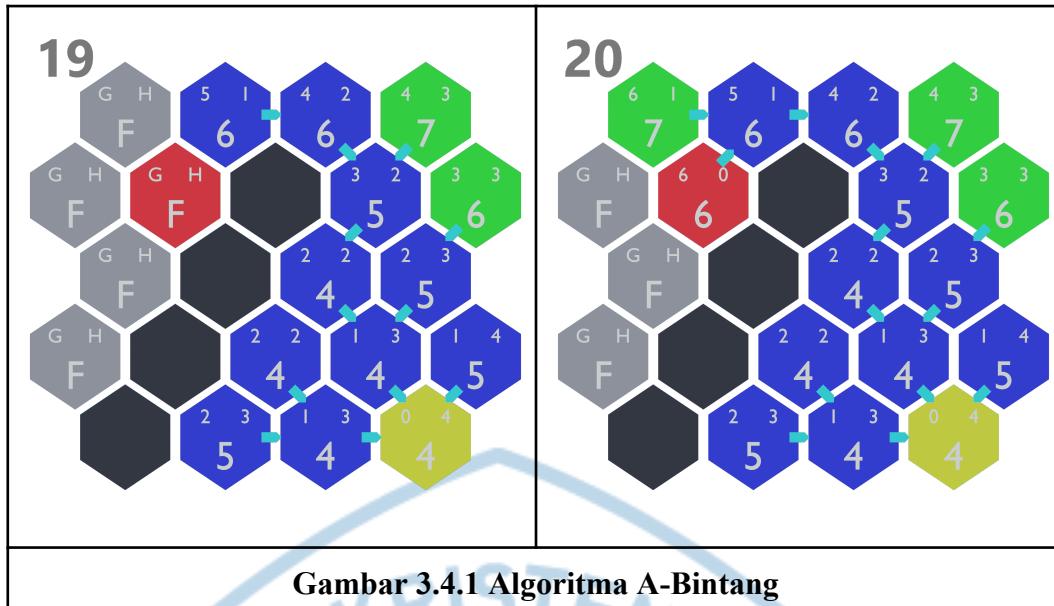
Pada **Langkah 20**, meskipun node merah tujuan telah ditemukan dan algoritma selesai melakukan pencarian jalur.

Setelah node merah tujuan ditemukan, maka algoritma dikatakan selesai dan melaksanakan penelusuran jalur kembali ke node kuning awal dengan mengikuti referensi induk dari setiap node, yang direpresentasikan dengan tanda panah biru.









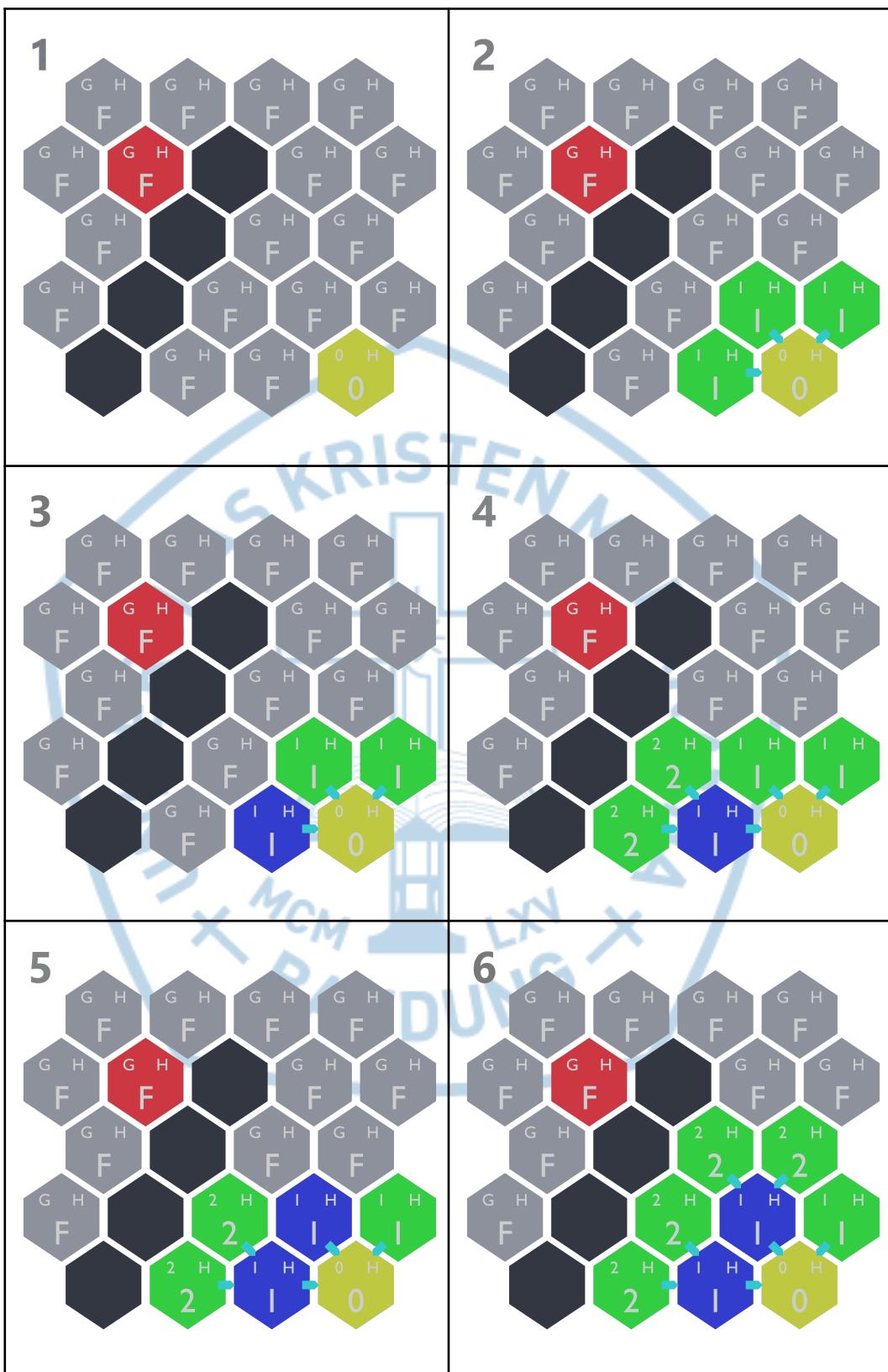
3.4.2 Dijkstra

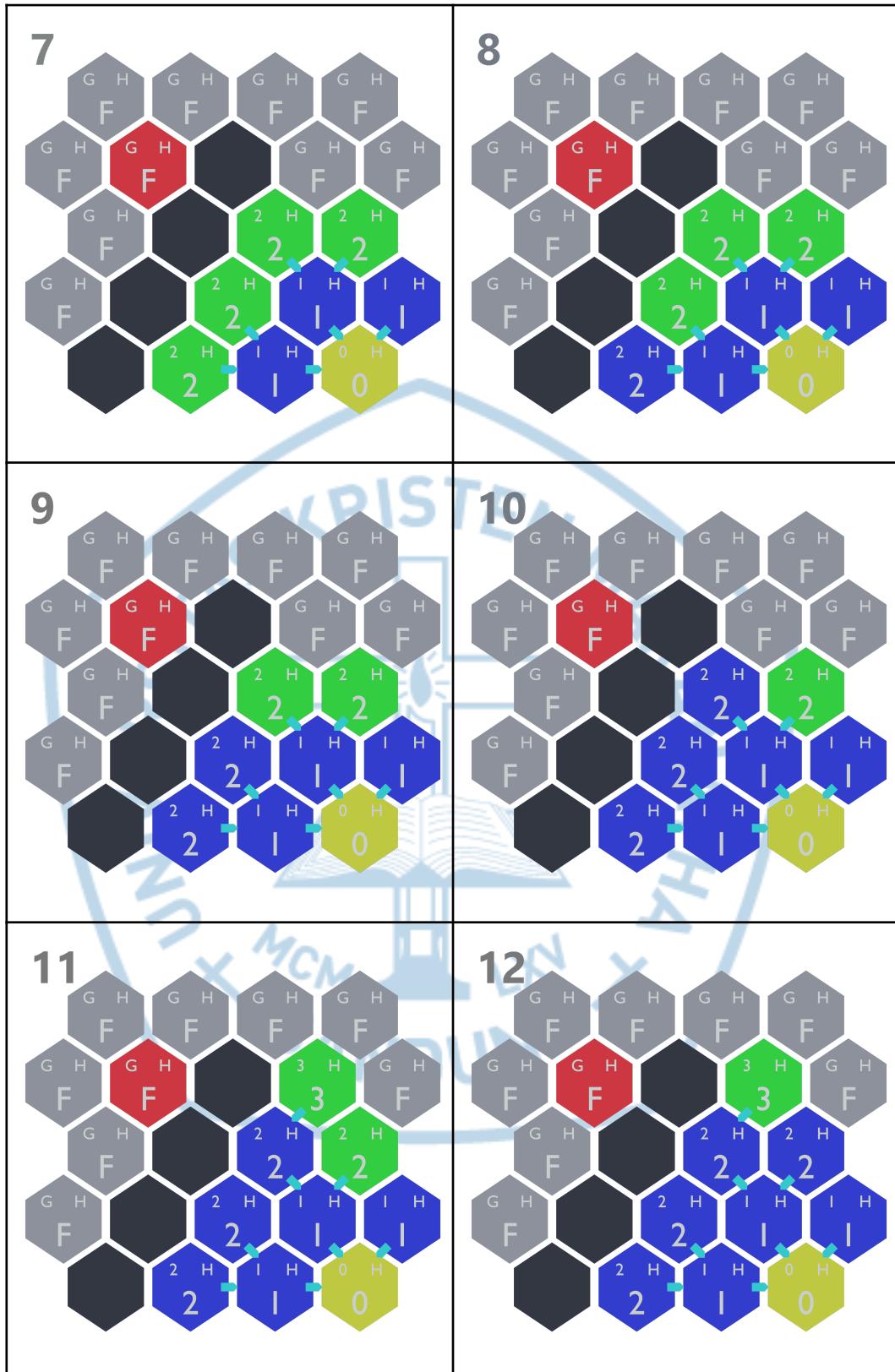
Berbeda dengan A-Bintang yang tidak memproses dua node hijau tetangga, Dijkstra tidak meninggalkan satupun node pada open list selama proses pencarian jalur, kecuali pada **Langkah 21** di mana **kondisi kedua** berlaku, yaitu node merah tujuan berada pada urutan pertama dalam open list.

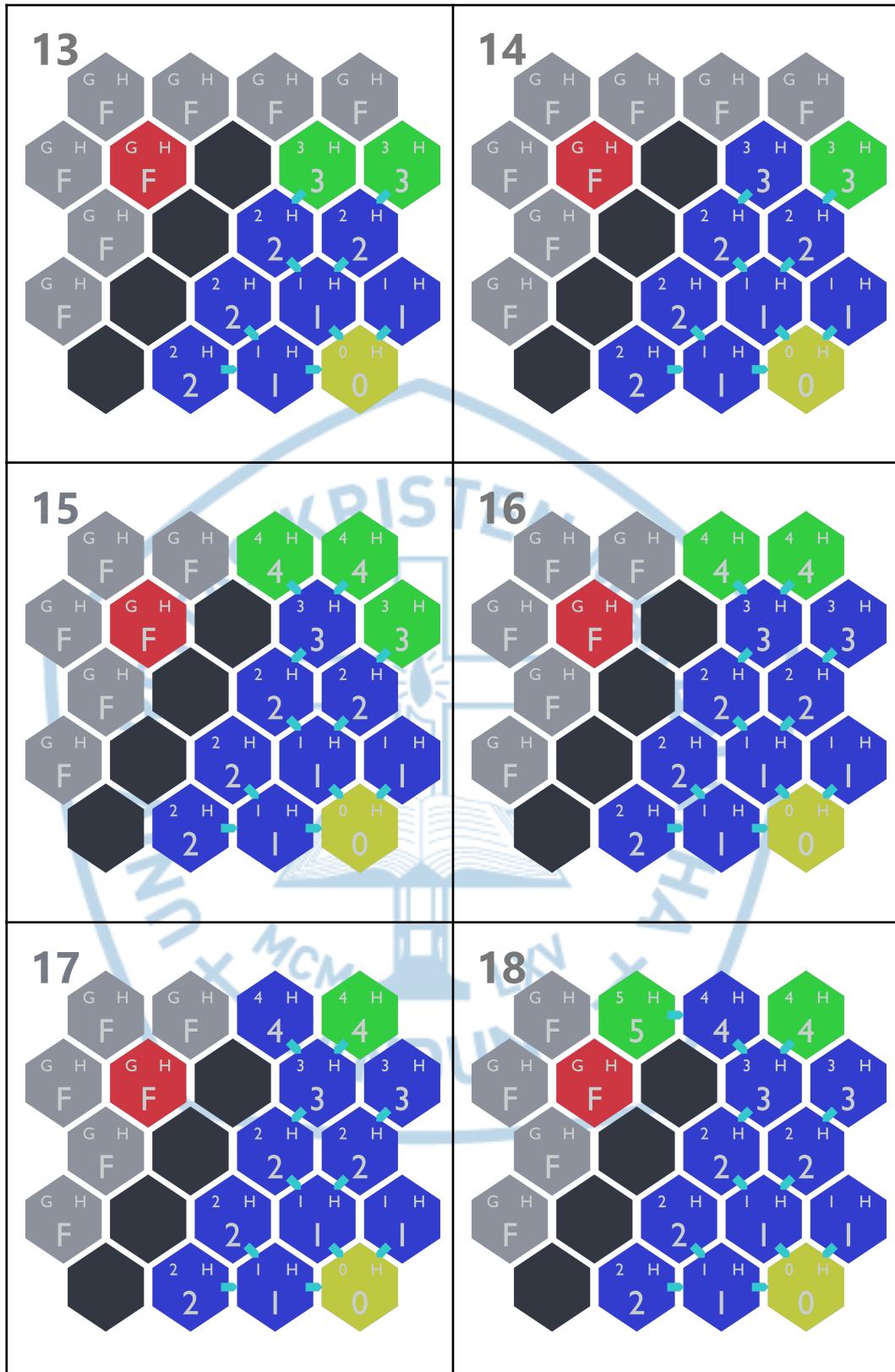
Selain itu, Dijkstra tidak menghitung h-value karena tujuan dari algoritma ini adalah mencari seluruh node hingga menemukan node merah tujuan. Oleh karena itu, h-value dari setiap node tidak diatur selama proses pencarian. Hal ini juga menyebabkan f-value dari setiap node sama dengan g-value pada setiap langkah yang dilalui.

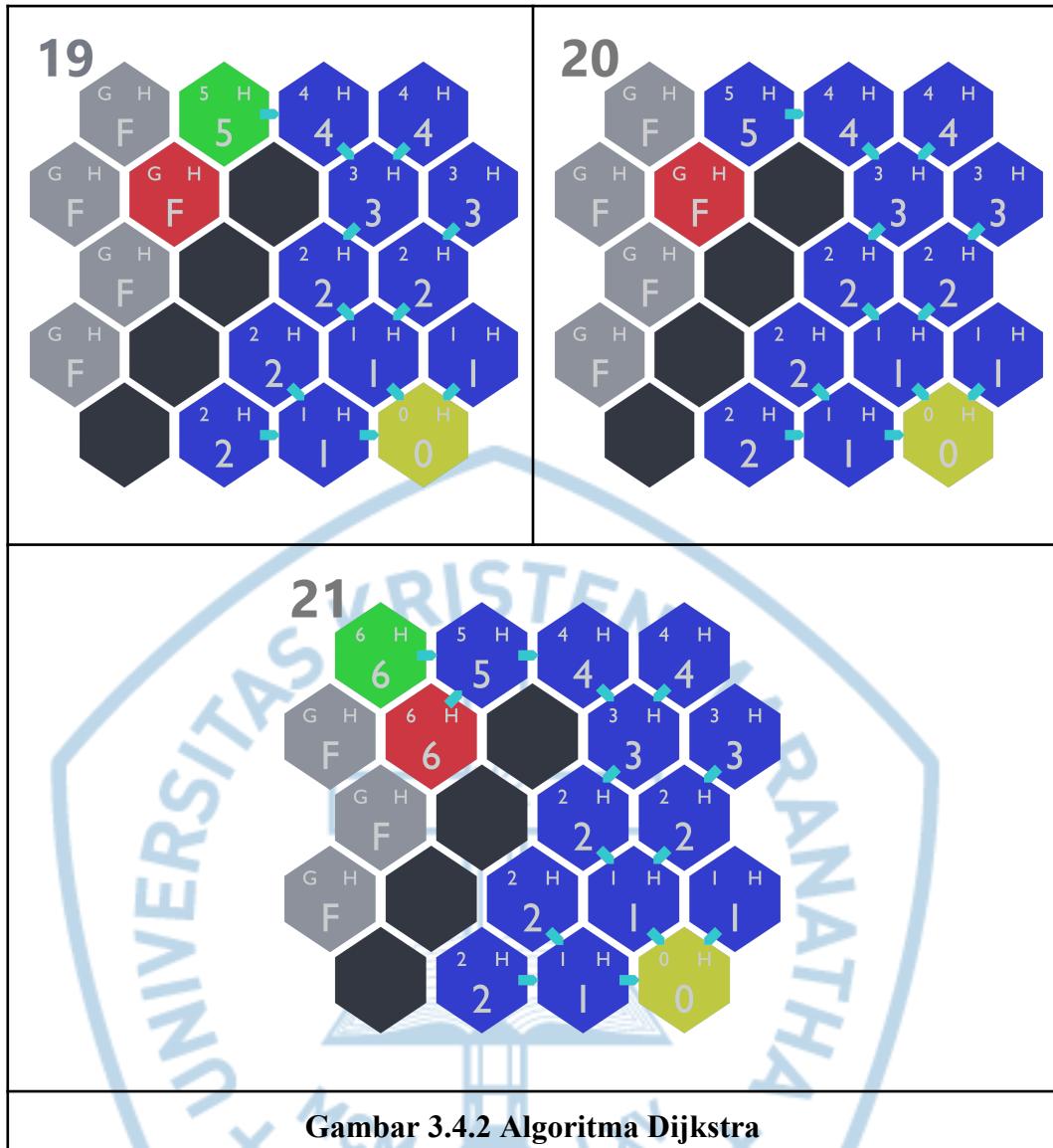
Pada **Gambar 3.4.2**, dapat dilihat bahwa **kondisi ketujuh** dalam pencarian jalur sangatlah terlihat keberadaannya. Yaitu node yang berada di urutan pertama dalam daftar node frontier list dipilih sebagai node biru terpilih untuk mencari langkah jalur selanjutnya.

Selain perbedaan yang disebutkan di atas, proses yang dilakukan untuk mencari jalur pada algoritma Dijkstra hampir sama dengan A-Bintang. Hanya di algoritma A-Bintang pencarian jalannya dituntun dengan menggunakan estimasi h-value, sedangkan Dijkstra hanya menggunakan g-value.









BAB 4

IMPLEMENTASI

4.1 Pengantar

Untuk mengimplementasikan penelitian perbandingan pencarian jalur ini, diperlukan beberapa program tersendiri yang bekerja secara bersamaan untuk membentuk satu program pencarian jalur yang kohesif. Proses dimulai dengan mengkonversi peta, yang awalnya dari file, menjadi data teks di dalam program. Selanjutnya, teks-teks ini diterjemahkan menjadi program node dengan berbagai properti dan fungsi. Setelah itu, node yang sudah diterjemahkan digabungkan menjadi satu graf yang kohesif, diurutkan, dan dapat ditampilkan dalam layar untuk membentuk peta segi enam. Peta ini dapat diproses oleh algoritma pencarian jalur ketika pengujian dilakukan untuk mencari node yang ditetapkan oleh pengguna, dan hasil pengujian dapat dicatat untuk perbandingan selanjutnya.

4.2 Map

Dalam penelitian ini, langkah awal yang perlu dilakukan adalah mengkonversi data dari peta yang berisi teks berupa angka menjadi file Godot.

1	01100001
2	0104401
3	0000022200
4	00133001
5	01100001
6	000100001

Gambar 4.2.1 Map Data Text

Seperti yang terlihat pada **Gambar 4.2.1**, awal dari peta ini berisi node-node yang hanya direpresentasikan sebagai teks berangka. Setiap angka ini memainkan peran penting dalam menentukan jenis simpul yang akan ditampilkan di layar program.

Telah dibuat lima belas data peta. Pada **Lampiran B Gambar Taks Map**, terlihat bahwa semua teks hanya terdiri dari angka tanpa menyertakan simpul terlebih dahulu.

```

34  func GetTextFromFile() -> PackedStringArray:
35    var lines:PackedStringArray = [];
36    var file:Object = FileAccess.open(textPath, FileAccess.READ);
37
38    if(file != null):
39      var text_asset:String = file.get_as_text();
40      var delimiter:String = "\n";
41      lines = text_asset.split(delimiter, false)
42    else:
43      print_debug("MAPDATA - GetTextFromFile Error: invalid file");
44
45    return lines;

```

Gambar 4.2.2 Text ke File

Setelah itu, teks yang telah diubah menjadi file .txt akan dibaca oleh Godot, di mana kode dapat ditemukan pada **Gambar 4.2.2**. Teks tersebut dimasukkan ke dalam array **lines** pada baris ke-35. Namun, karena file yang dibaca atau dibuka awalnya bukan berupa teks, maka pada baris ke-39, file tersebut diubah menjadi teks menggunakan fungsi **get_as_text()**.

Selanjutnya, teks akan dipisahkan berdasarkan **delimiter**, yaitu “\n”, sehingga array **lines** terbagi menjadi beberapa elemen. Hal ini bergantung pada baris teks data peta yang diberikan.

```

55  ↘ func TranslateMap() -> void:
56    >|   width = 0;
57    >|   height = 0;
58    >|   var lines:PackedStringArray = GetTextFromFile();
59    >|   SetDimension(lines);
60    >|
61    >|   data.resize(width);
62  ↘ >|   for column in range(self.width):
63    >|     data[column].resize(self.height);
64    >|
65  ↘ >|   for x in range(self.width):
66  ↘ >|     for y in range(self.height):
67  ↘ >|       if(lines[y].length() > x):
68    >|         data[x][y] = lines[y][x].to_int();
69  ↘ >|       else:
70    >|         data[x][y] = 0;

```

Gambar 4.2.3 File ke Map

Data yang diperoleh dari fungsi **GetTextFromFile()** berupa array elemen teks. Oleh karena itu, diperlukan fungsi untuk menerjemahkan data ini menjadi format yang dapat dibaca oleh program pembentukan peta di masa mendatang.

Seperti yang terlihat pada **Gambar 4.2.3** baris ke 56-61, data yang layak untuk diproses seharusnya berupa array dua dimensi. Variabel **height** dan **width** akan menentukan ukuran data tersebut. Setelah ukuran array dua dimensi ditentukan, seperti yang terlihat pada baris ke 65-70, **data** yang diperoleh dari fungsi **GetTextFromFile()** akan dikonversi menjadi tipe data **integer**. Data awalnya hanya berupa karakter dalam kumpulan **string**.

4.3 Node

```

1  extends Node2D
2  class_name PathNode
3
4  @export var tile:Sprite2D;
5  @export var arrow:Sprite2D;
6
7  enum Type{
8    OPEN = 0,
9    BLOCK = 1,
10   MILD = 2,
11   HARSH = 3,
12   JARRING = 4,
13 }
14
15  var index:Vector2i = Vector2i.ZERO;
16  var neighbors:Array[PathNode] = [];
17  var type:int = Type.OPEN;
18  var previous:PathNode = null;
19  var priority:int = 0;
20  var distance:int = 0;
21
22  func Reset() -> void:
23    previous = null;
24
25  func CompareTo(_other:PathNode) -> int:
26    if(priority < _other.priority):
27      return -1;
28    elif(priority > _other.priority):
29      return 1;
30    else:
31      return 0;

```

Gambar 4.3 Kode Node

Pada implementasi Node, ada beberapa hal yang perlu diperhatikan, salah satunya adalah hubungan antar tetangganya.

Jika diimplementasikan menggunakan GDScript, variabel **index** pada **Gambar 4.3** baris 15 memiliki tipe data bilangan bulat vektor. Ini memungkinkan node untuk diindekskan dan ditempatkan pada map 2 dimensi. Dengan demikian, algoritma pencarian tidak perlu mengulang dan menerka-nerka di bagian mana node yang dipilih berada.

Seperi yang terlihat pada **Gambar 4.3** baris ke-16, hubungan antara node tersebut direpresentasikan dengan **neighbors** yang memiliki tipe data array. Pada peta heksagonal, setiap node memiliki 2-6 tetangga, dan untuk merepresentasikan tetangga node, digunakan tipe data Array.

Beberapa node dalam percobaan memiliki biaya tempuh tambahan. Oleh karena itu, pada **Gambar 4.3** baris ke-17, diberikan **type** pemrograman node

untuk memudahkan pembentukan berbagai tipe node dengan biaya tempuh yang berbeda. Meskipun perbedaan antar node tidak signifikan, terdapat perbedaan pada biaya tempuh yang harus dijalani, yaitu dari biaya tempuh bernilai 1 menjadi bernilai 1-4, tergantung pada tipe yang ditentukan pada baris ke-7.

Tidak hanya itu, komponen paling penting dalam algoritma pencarian jalur adalah referensi kepada induk dari node terbaik. Oleh karena itu, dalam pemrograman node, diperlukan variabel penyimpanan referensi yang dapat dibaca dan menunjuk pada induk yang telah ditentukan. Ini dapat diwakili seperti pada **Gambar 4.3** baris ke-18, yang dapat disebut sebagai '**previous**', yang menunjukkan bahwa node terbaik sebelumnya dalam pencarian algoritma merupakan **previous** itu sendiri.

Variabel **priority** dan **distance** pada **Gambar 4.3** baris 19-20 adalah parameter-node yang memungkinkan algoritma memilih dan memilih node terbaik. Meskipun keduanya akan memiliki nilai yang berbeda, pembentukan awal atau pendefinisian awal kedua algoritma dilakukan secara bersamaan, dan menggunakan cara yang sama, sehingga kedua variabel bisa dikatakan sama kegunaannya. Perbedaannya terletak pada fakta bahwa **priority** dapat diubah selama proses algoritma dan bergantung pada keunggulan node yang diinginkan. Sebaliknya, **distance** merupakan nilai tetap yang tidak berubah selama pencarian jalur dilakukan.

4.4 Graph

```
1  extends Node2D
2  class_name GraphNodes
3
4  @export var pathNode:Resource = preload("res://Scenes/path_node.
5
6  var nodes:Array[Array];
7  var width:int;
8  var height:int;
9
10 static var oddDirections:PackedVector2Array = [
11   >I Vector2(0,1),
12   >I Vector2(1,1),
13   >I Vector2(1,0),
14   >I Vector2(1,-1),
15   >I Vector2(0,-1),
16   >I Vector2(-1,0),
17 ];
18 static var evenDirections:PackedVector2Array = [
19   >I Vector2(0,1),
20   >I Vector2(1,0),
21   >I Vector2(0,-1),
22   >I Vector2(-1,-1),
23   >I Vector2(-1,0),
24   >I Vector2(-1,1),
25 ];
26
27 func GetDistance(_start:PathNode, _end:PathNode) -> int:
28   >I var dx:int = abs(_start.index.x - _end.index.x);
29   >I var dy:int = abs(_start.index.y - _end.index.y);
30   >I return (dy + dx);
31
32 func IsWithinBounds(_x:int, _y:int) -> bool:
33   >I return (_x >= 0 && _x < width && _y >= 0 && _y < height);
34
```

```

35  ↘ func GetNeighbors(_x:int, _y:int) -> Array[PathNode]:
36    ↘|  var neighbor_nodes:Array[PathNode] = [];
37    ↘|  var directions:PackedVector2Array;
38    ↘|  if(_y % 2 == 0):
39      ↘|    directions = evenDirections;
40    ↘|  else:
41      ↘|    directions = oddDirections;
42    ↘|  for dir in directions:
43      ↘|    var newX:int = _x + int(dir.x);
44      ↘|    var newY:int = _y + int(dir.y);
45    ↘|    if(IsWithinBounds(newX, newY) && nodes[newX][newY].type != PathNode.Type.BLOCK):
46      ↘|      neighbor_nodes.push_back(nodes[newX][newY]);
47    ↘|  return neighbor_nodes;
48
49  ↘ func EstablishGraph(_width:int, _height:int, _data:Array[PackedInt32Array]) -> void:
50    ↘|  for i in range(get_child_count()):
51      ↘|    get_child(i).queue_free();
52    ↘|  width = _width;
53    ↘|  height = _height;
54    ↘|  nodes.resize(_width);
55  ↘|  for i in range(nodes.size()):
56    ↘|    nodes[i].resize(_height);
57  ↘|  for x in range(_width):
58    ↘|    for y in range(_height):
59      ↘|      var instance:Node2D = pathNode.instantiate();
60      ↘|      instance = instance as PathNode;
61      ↘|      instance.type = _data[x][y];
62      ↘|      instance.index = Vector2(x,y);
63    ↘|      if(y % 2 == 0):
64        ↘|        instance.position = Vector2(x, y) * 128 + Vector2(64, 64);
65    ↘|      else:
66        ↘|        instance.position = Vector2(x, y) * 128 + Vector2(128, 64);
67      ↘|      instance.ColorTile(MapNodes.GetColorFromNodeType(instance.type));
68      ↘|      nodes[x][y] = instance;
69      ↘|      self.add_child(instance);
70  ↘|  for x in range(_width):
71    ↘|    for y in range(_height):
72      ↘|      nodes[x][y].neighbors = GetNeighbors(x,y);
73

```

Gambar 4.4 Kode Graph

Graph pada keseluruhan **Gambar 4.4** merupakan kelas yang nantinya akan menghubungkan node-node yang telah ditetapkan. Node-node ini nantinya memiliki **neighbors**, yang nantinya akan menjadi cara algoritma pencarian jalur menemukan node sekitarnya.

Sesuai contoh yang diberikan oleh Red Blob Games dalam bagian Hexagonal Grids tentang Offset Coordinates [6], terdapat dua arah pencarian yang dibedakan berdasarkan ganjil dan genap. Hal ini dilakukan pada **Gambar 4.4** baris 10 dan 18 dengan variabel **oddDirections** dan **evenDirections**. Saat mencari tetangga-tetangga node, jika **indeks.y** dari node yang dicari adalah ganjil, arah pencarian menggunakan **oddDirections**; sebaliknya, digunakan **evenDirections**.

Pada pencarian jalur Dijkstra dan A-Bintang, diperlukan cara untuk menghitung jarak antara satu node dan node lainnya. Pada pemrograman node, terdapat properti jarak yang digunakan untuk menentukan node terbaik dalam pencarian jalur selanjutnya. Sebagaimana terlihat pada **Gambar 4.4** baris 27, terdapat fungsi yang dibentuk untuk menghitung jarak antar node, **GetDistance()**. Introduction to the A* Algorithm oleh Red Blob Games, bagian Heuristic Search, menyebutkan salah satu metode untuk menentukan jarak adalah menggunakan Manhattan Distance [18]. Formula ini melibatkan perbedaan absolut antara properti x dan y dari dua node, yang kemudian dijumlahkan.

Fungsi **IsWithinBounds()** pada **Gambar 4.4** baris 32 adalah fungsi sederhana untuk menentukan apakah node yang dicari telah melewati atau melampaui panjang dan tinggi dari map heksagonal yang telah dibuat. Fungsi ini akan digunakan oleh algoritma Dijkstra dan A-Bintang untuk menentukan node mana yang akan digunakan dari keseluruhan himpunan node yang telah ditemukan.

Untuk memenuhi himpunan tetangga pada pemrograman node, diperlukan cara untuk mencari node-node sekitar node yang dipilih. Fungsi **GetNeighbors()** pada **Gambar 4.4** baris 35 adalah implementasinya. Dengan mengulang himpunan arah tetangga yang ditentukan berdasarkan ganjil atau genapnya **indeks.y** pada node, fungsi ini mencari node-node yang mungkin menjadi tetangga. Jika ditemukan node yang berpotensi menjadi tetangga, node tersebut dimasukkan ke dalam himpunan. Himpunan ini nantinya akan digunakan untuk mendefinisikan tetangga pada pemrograman node.

Pada **Gambar 4.4** baris 49, terdapat fungsi **EstablishGraphs()** yang bertanggung jawab untuk membentuk grafnya itu sendiri. Fungsi pemrograman ini melakukan pemasangan atau perhubungan antara satu node dengan node lainnya. Seperti yang terlihat pada parameter fungsi, **_data** diambil sebagai parameter penentuan node-node himpunan. Di sini, **_data** merupakan representasi 2 dimensi dari letak node yang akan dibentuk. Dengan nilai-nilai bilangan positif 0-4, **_data** menentukan tipe node yang akan diberikan, seperti terlihat pada implementasi baris 61.

4.5 Algoritma

```

125     func ExpandFrontierDijkstra(_node:PathNode) -> void:
126         for i in range(_node.neighbors.size()):
127             if(!reachedNodes.has(_node.neighbors[i])):
128                 var distance:int = graphNodes.GetDistance(_node, _node.neighbors[i]);
129                 var sum_distance:int = distance + _node.distance + _node.neighbors[i].type;
130             if(_node.neighbors[i].distance == 0 || sum_distance < _node.neighbors[i].distance):
131                 _node.neighbors[i].previous = _node;
132                 _node.neighbors[i].distance = sum_distance;
133             if(!frontierNodes.has(_node.neighbors[i])):
134                 _node.neighbors[i].priority = _node.neighbors[i].distance;
135                 PathNode.PriorityEnqueueBinaryHeap(_node.neighbors[i], frontierNodes);
136             # Informed
137     func ExpandFrontierAStar(_node:PathNode) -> void:
138         for i in range(_node.neighbors.size()):
139             if(!reachedNodes.has(_node.neighbors[i])):
140                 var distance:int = graphNodes.GetDistance(_node, _node.neighbors[i]);
141                 var sum_distance:int = distance + _node.distance + _node.neighbors[i].type;
142                 var distance_end:int = graphNodes.GetDistance(_node.neighbors[i], endNode);
143             if(_node.neighbors[i].distance == 0 || sum_distance < _node.neighbors[i].distance):
144                 _node.neighbors[i].previous = _node;
145                 _node.neighbors[i].distance = sum_distance;
146             if(!frontierNodes.has(_node.neighbors[i])):
147                 _node.neighbors[i].priority = _node.neighbors[i].distance + distance_end;
148                 PathNode.PriorityEnqueueBinaryHeap(_node.neighbors[i], frontierNodes);

```

Gambar 4.5 Kode Algoritma

Berikut adalah kode algoritma yang akan digunakan, seperti yang terlihat pada **Gambar 4.5** baris 125-135 dan 137-148. Kedua algoritma hampir identik dalam pemrogramannya, dikarenakan penggunaan fungsi heuristik estimasi yang khas digunakan oleh A-Bintang. Jika fungsi ini, seperti pada **Gambar 4.5** baris 142 dan 147, dihilangkan, maka kedua algoritma akan memiliki cara pencarian jalur yang serupa. Seperti yang dijelaskan dalam artikel 'Introduction to the A* Algorithm' bagian Algorithms oleh Red Blob Games, algoritma A-Bintang sebenarnya hanyalah modifikasi dari algoritma Dijkstra [18].

```

33  static func PriorityEnqueueBinaryHeap(_node:PathNode, _queue:Array[PathNode]) -> void:
34    _queue.push_back(_node);
35    var index_queue:int = _queue.size() - 1;
36
37    while(index_queue > 0):
38      var parent_index:int = floor((index_queue - 1) / 2);
39
40      if(_queue[index_queue].CompareTo(_queue[parent_index]) >= 0):
41        break;
42
43      var temp_node:PathNode = _queue[index_queue];
44      _queue[index_queue] = _queue[parent_index];
45      _queue[parent_index] = temp_node;
46
47      index_queue = parent_index;

```

Gambar 4.5 Kode Priority Enqueue

Fungsi **PriorityEnqueueBinaryHeap()** pada **Gambar 4.5** merupakan fungsi yang digunakan oleh kedua algoritma untuk frontier list yang memerlukan properti selalu mengurutkan node berdasarkan prioritasnya. Dalam hal ini, node-node diurutkan secara berantai yang memiliki dua cabang, dan selalu diurutkan kembali jika ada node baru yang dimasukkan. Fungsi ini mengurutkan dengan cara membandingkan prioritas kedua node rantingnya untuk menentukan apakah prioritasnya lebih tinggi dari node itu sendiri. Jika ya, maka dilakukan pertukaran hingga mencapai ranting paling akhir, seperti yang terlihat pada baris 37 sampai 47. Meskipun node-node rantingnya dapat dikatakan tidak diurutkan secara berurutan dari yang terendah hingga yang tertinggi, namun yang penting diketahui adalah node awal dari list ini memiliki prioritas yang paling rendah. Hal ini sangat bermanfaat untuk algoritma A-Bintang dan Dijkstra yang hanya tertarik pada node awal list.

4.6 Catatan Hasil

```

12  func Establish(_log_data:Array) -> void:
13    for data in _log_data:
14      var row_instance:HBoxContainer = row.instantiate();
15      row_instance.get_child(0).text = str(data[0]);
16      row_instance.get_child(1).text = str(data[1]);
17      row_instance.get_child(2).text = str(data[2]);
18      row_instance.get_child(3).text = str(data[3]);
19      row_instance.get_child(4).text = str(data[4]);
20      row_instance.get_child(5).text = str(data[5]);
21      row_instance.get_child(6).text = str(data[6]);
22      self.get_child(2).get_child(1).get_child(0).add_child(row_instance);
23      logData = _log_data;
24
25  func _on_button_pressed() -> void:
26    self.queue_free();
27
28  func _on_button_2_pressed() -> void:
29    fileDialog.add_filter("*.txt", "Text");
30    fileDialog.current_file = "Result Log";
31    fileDialog.popup_centered();
32
33  func _on_file_dialog_file_selected(path:String) -> void:
34    var file = FileAccess.open(path, FileAccess.WRITE)
35    for item in logData:
36      file.store_line(Join(item, ","));
37    file.close();
38
39  func Join(array:Array, filler:String) -> String:
40    var rs:String = "";
41    for s in array:
42      rs += str(s)+filler;
43    return rs;
44
45  func _on_file_dialog_confirmed():
46    acceptDialog.dialog_text = "Save has been successful.";
47    acceptDialog.popup_centered();

```

Gambar 4.6 Kode Priority Enqueue

Untuk mencatat hasil pengujian kedua algoritma, diperlukan suatu program khusus yang dapat mengambil data hasil proses pencarian jalur dan menyusunnya menjadi teks yang dapat dibaca. Sebagaimana terlihat pada baris 13 sampai 23 pada **Gambar 4.6**, program ini menerima hasil proses pencarian jalur yang telah selesai berjalan, lalu menampilkannya di layar menggunakan fitur khusus Godot untuk menampilkan string. Fungsi-fungsi seperti **_on_button_pressed()** dan **_on_button_2_pressed()** bertujuan untuk merespons interaksi pengguna dan menentukan kapan hasil pengujinya akan ditampilkan di layar.

BAB 5

PENGUJIAN

5.1 Pengantar

Untuk melakukan pengujian dan menentukan algoritma mana yang terbaik, perlu adanya elemen penilaian tertentu. Kecepatan Tempuh, Biaya Perjalanan, Node Dilalui, dan Node Dikunjungi merupakan elemen-elemen penilaian yang digunakan dalam penelitian ini, yang akan dievaluasi pada enam puluh kasus uji coba yang telah ditetapkan.

Tabel yang ditandai dengan warna hijau dimaksudkan untuk menunjukkan bahwa hasil dari pengujian menunjukkan keunggulan algoritma tersebut dibandingkan dengan algoritma lainnya.

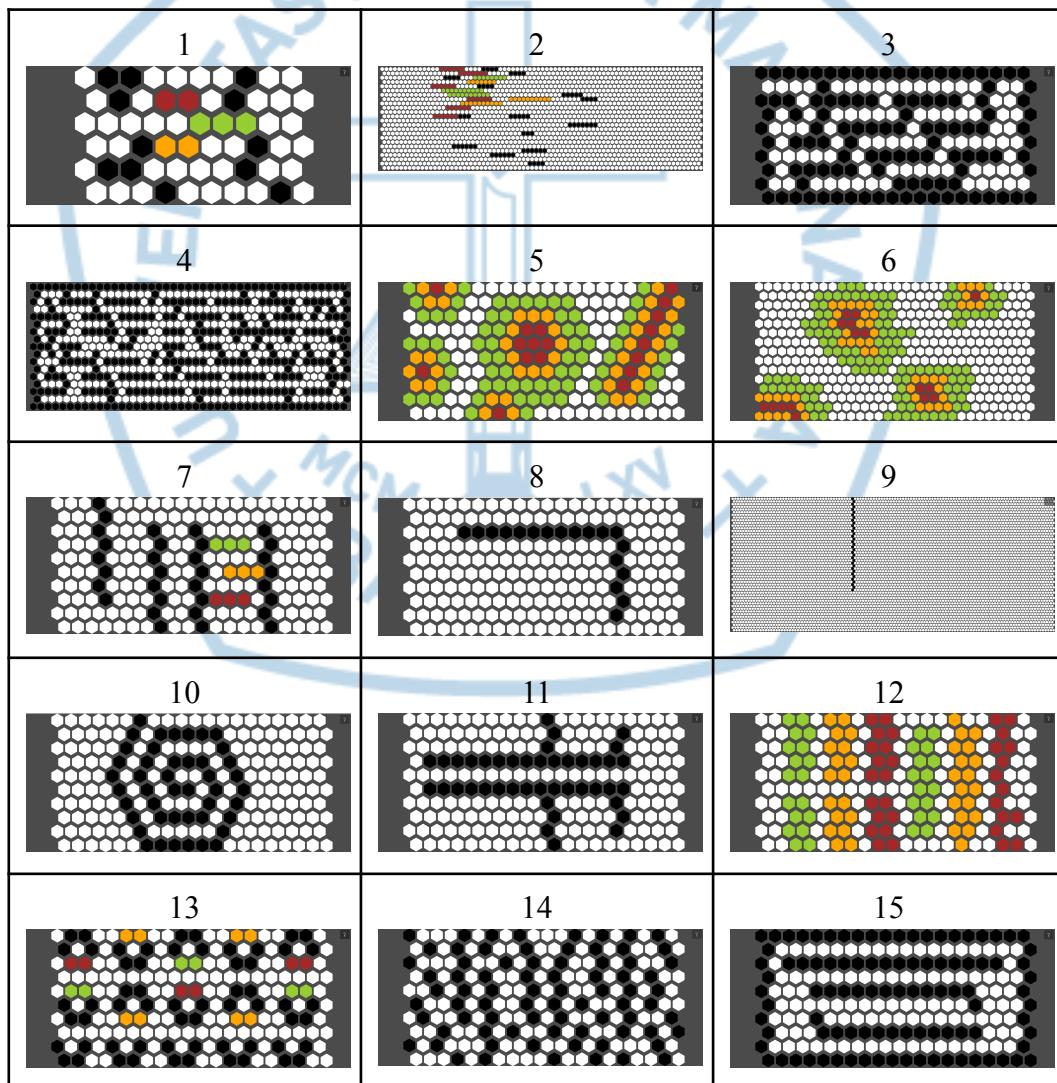
Pengujian akan dilakukan pada kelima belas peta yang telah dibentuk sebelumnya. Seperti yang terlihat pada **Gambar 5.1.1**, algoritma akan diuji pada medan map dengan karakteristik seperti ini:

1. Peta pertama diatur dengan tujuan untuk memberikan pemahaman tentang cara kerja kedua algoritma serta untuk menunjukkan jalur pencarian. Oleh karena itu, seluruh medan pada peta ini dirancang kecil dan tidak memerlukan banyak langkah untuk dilalui.
2. Peta kedua diatur dengan tujuan memberikan peta yang memiliki ukuran besar, tetapi tidak memiliki struktur atau motivasi tertentu dalam pembentukannya. Dengan kata lain, tipe-tipe node seperti node tembok tidak disusun dengan pola tertentu, melainkan ditempatkan secara acak.
3. Peta ketiga diatur dengan tujuan untuk memahami bagaimana kedua algoritma pencarian jalur menavigasi medan labirin yang hanya memiliki satu pintu masuk dan satu pintu keluar.

4. Peta keempat memiliki motivasi serupa dengan peta nomor tiga, yaitu mengevaluasi bagaimana kedua algoritma pencarian jalur menangani medan. Namun, peta keempat memiliki ukuran medan yang lebih besar dan memiliki alur tempuh yang lebih berliku-liku.
5. Peta kelima dibentuk untuk meniru peta topografi dengan pembentukan bidang area yang memiliki elevasi lebih rendah atau lebih tinggi dibandingkan area lainnya. Tujuannya adalah agar kedua algoritma dapat menavigasi peta ini dengan mempertimbangkan perbedaan elevasi, sehingga diharapkan algoritma akan menghindari area dengan elevasi yang tinggi.
6. Peta keenam memiliki motivasi serupa dengan peta nomor lima, tetapi memiliki ukuran medan yang lebih besar.
7. Peta ketujuh dibentuk dengan motivasi serupa, mengetahui dampak dari node elevasi yang lebih tinggi dibandingkan dengan node yang normal. Oleh karena itu, node elevasi hanya ditempatkan pada area tertentu dan diwujudkan dalam bentuk garis penghalang.
8. Peta kedelapan dibentuk dengan motivasi untuk memahami arah yang akan dipilih oleh algoritma selama proses pencarian jalur. Apakah akan melewati bagian atas atau bagian bawah.
9. Peta sembilan dibentuk dengan motivasi yang serupa dengan peta nomor delapan, namun dengan perluasan area medan tempuh.
10. Peta kesepuluh dibentuk dengan motivasi untuk mengetahui apakah benar algoritma A-Bintang lebih cepat ketika medan yang harus dilalui hanya memiliki satu jalur saja. Dengan demikian, pencarian jalur Dijkstra akan memiliki arah yang sama dengan A-Bintang tanpa harus mencari keseluruhan node.
11. Peta kesebelas memiliki motivasi yang mirip dengan peta nomor sepuluh, namun pada awal dan akhir pencarian jalur, area kosong disediakan agar A-Bintang dapat memanfaatkan kelebihan algoritmanya sebelum harus menjelajahi satu jalur seperti pada peta sepuluh.
12. Peta kedua belas dibentuk dengan motivasi untuk memahami bagaimana kedua algoritma ini melakukan pencarian di medan yang dapat dilalui

namun memerlukan pembayaran biaya perjalanan, menggantikan peran node tembok dengan node elevasi.

13. Peta ketiga belas dibentuk dengan motivasi untuk mengetahui bagaimana pintu masuk dan pintu keluar pada peta labirin nomor 3 dan 4 akan mempengaruhi pencarian jalur jika pintunya dihalangi oleh node elevasi.
14. Peta keempat belas memiliki motivasi yang serupa dengan peta nomor 3 dan 4, dengan perbedaan bahwa jika peta labirin sebelumnya memiliki jalur yang jelas dan dapat dilalui seperti labirin pada umumnya, peta ini memiliki halangan yang berdiagonal dan tidak teratur secara acak.
15. Peta kelima belas memiliki motivasi yang serupa dengan peta nomor 8 dan 9, dengan tambahan aspek arah yang dapat dilalui oleh algoritmanya.



Gambar 5.1.1 Peta Skenario Pengujian

5.2 Perbandingan Kecepatan Tempuh

Untuk membandingkan elemen kecepatan dari kedua algoritma ini, diperlukan fungsi untuk menghitung waktu yang telah berlalu sejak program pencarian jalur dijalankan.

```
var time_start:float = Engine.get_physics_frames();
var time_end:float = 0.0;
var fps:float = ProjectSettings.get_setting("physics/common/physics_ticks_per_second");

if(time_end == 0.0):
    > time_end = (Engine.get_physics_frames() - time_start) / fps;
```

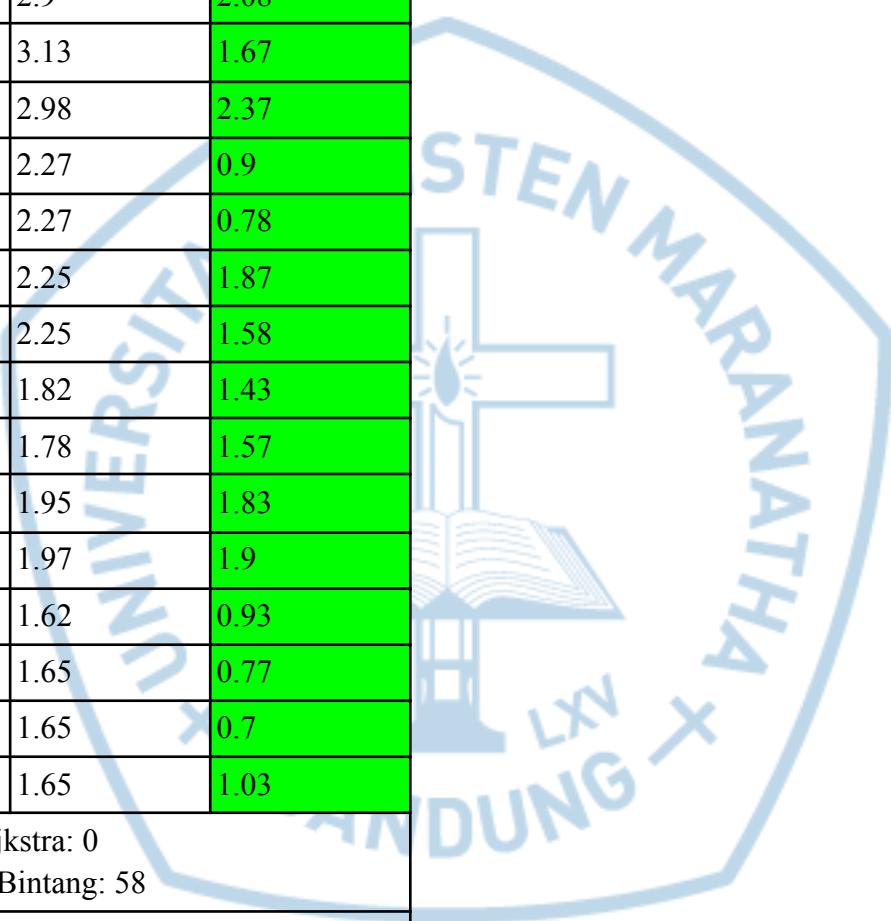
Gambar 5.2 Fungsi Perhitungan Kecepatan Tempuh

Salah satu cara untuk menghitung elemen ini, jika dilakukan dalam GDScript Godot, adalah dengan menggunakan frame per detik. Pada prinsipnya, langkah ini melibatkan mengurangkan jumlah frame saat program sedang berjalan dari jumlah frame awal yang direkam saat algoritma dijalankan. Selanjutnya, hasilnya dibagi dengan frame per detik secara keseluruhan dalam program. Hal ini tergambar pada **Gambar 5.2**.

No	Kecepatan Dijkstra	Kecepatan A-Bintang
1	0.65	0.52
2	0.58	0.38
3	0.75	0.65
4	0.73	0.48
5	29.27	9.65
6	29.25	7.7
7	29.27	3.33

8	29.25	9.57
9	1.23	1.18
10	1.25	1.23
11	1.43	1.42
12	1.28	1.18
13	5.72	2.57
14	5.9	5.13
15	5.65	3.98
16	5.88	3.05
17	2.93	2.22
18	2.93	1.58
19	3.2	2.75
20	2.65	1.13
21	7.27	3.18
22	7.42	6.1
23	7.22	4.18
24	7.25	2.65
25	2.52	1.82
26	1.83	1.25
27	1.85	1.18
28	2.52	1.8
29	2.77	1.87
30	2.48	1.08
31	2.72	1.02
32	2.67	1.58
33	54.63	27.18
34	56.67	11.68
35	36.8	1.77
36	58.13	20.37
37	0.87	0.65
38	1.72	1.72





39	1.72	1.72
40	1.17	0.85
41	2.35	1.9
42	2.35	1.58
43	2.3	1.58
44	2.15	1.73
45	3	2.23
46	2.9	2.08
47	3.13	1.67
48	2.98	2.37
49	2.27	0.9
50	2.27	0.78
51	2.25	1.87
52	2.25	1.58
53	1.82	1.43
54	1.78	1.57
55	1.95	1.83
56	1.97	1.9
57	1.62	0.93
58	1.65	0.77
59	1.65	0.7
60	1.65	1.03
Dijkstra: 0		
A-Bintang: 58		
Tabel 5.2 Perbandingan Kecepatan Tempuh		

Setelah melakukan perbandingan pada kelima belas peta yang telah diberikan, terlihat bahwa algoritma A-Bintang secara signifikan unggul dibandingkan dengan pendahulunya, yaitu Dijkstra. A-Bintang berhasil meraih nilai 58 dari total 60 pengujian yang dilakukan. Dua pengujian yang menghasilkan nilai seri terjadi pada peta yang hanya memiliki satu jalur untuk menyelesaikan

pencarian jalur. Hal ini menunjukkan bahwa keefektifan A-Bintang secara nyata lebih unggul dibandingkan dengan Dijkstra.

5.3 Perbandingan Biaya Perjalanan

Untuk membandingkan biaya perjalanan yang telah ditempuh, beberapa langkah perlu dilakukan untuk menemukan biaya tempuh ke node akhir. Node akhir ini akan menjadi acuan apakah pencarian jalur berhasil ditemukan atau tidak.

```

log += " |> Path Length = " + str(endNode.distance);

var distance:int = graphNodes.GetDistance(_node, _node.neighbors[i]);

func GetDistance(_start:PathNode, _end:PathNode) -> int:
    var dx:int = abs(_start.index.x - _end.index.x);
    var dy:int = abs(_start.index.y - _end.index.y);
    return (dy + dx);

var sum_distance:int = distance + _node.distance + _node.neighbors[i].type;

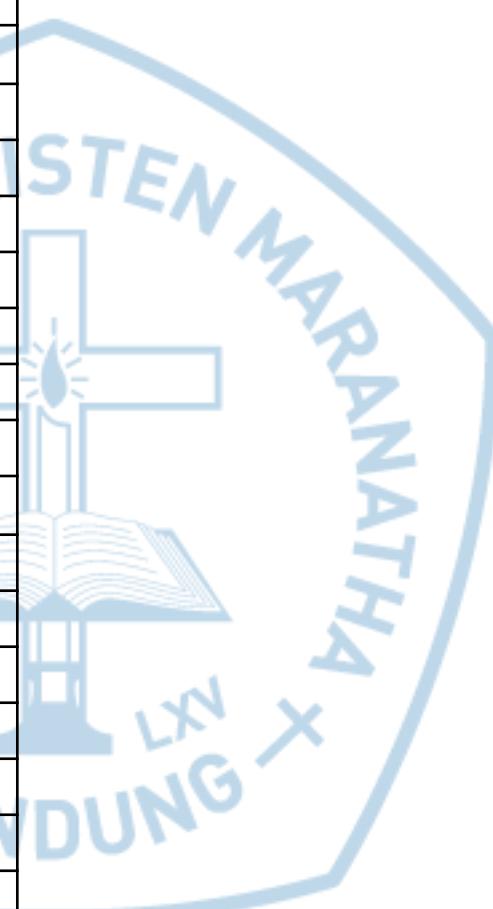
```

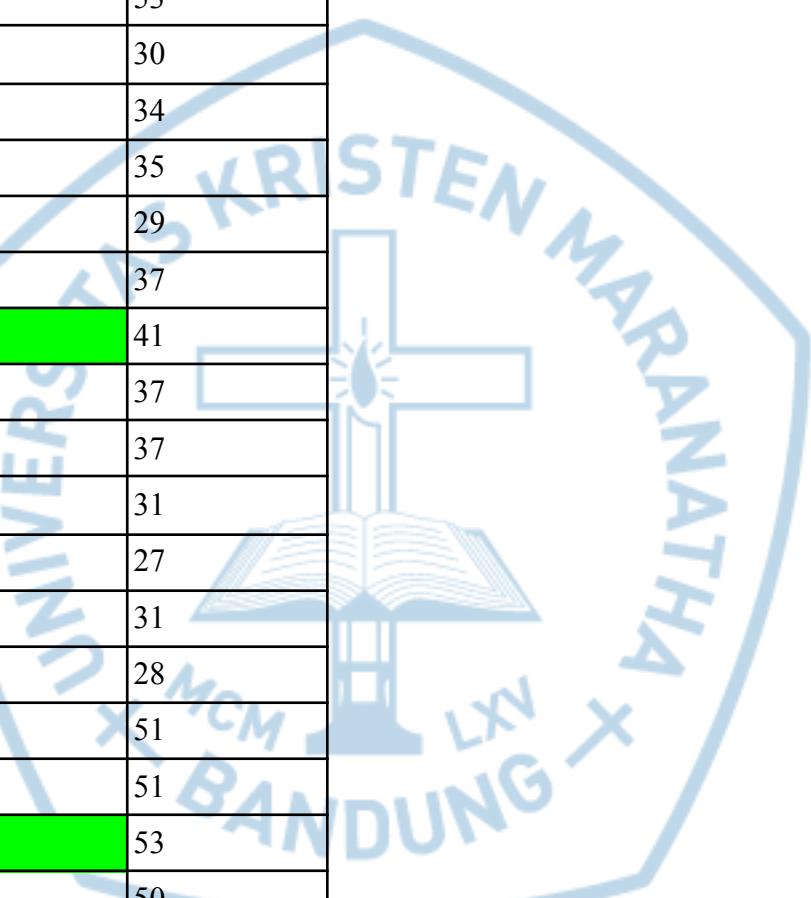
Gambar 5.3 Program Mengetahui Biaya Perjalanan

Pertama, untuk mengetahui biaya akhir dari seluruh node yang telah ditempuh, diperlukan sebuah fungsi untuk menentukan jarak dari satu node ke node berikutnya yang telah dilalui. Setelah jarak dihitung, langkah selanjutnya adalah memperhitungkan biaya medan yang harus ditempuh. Setelah semua data ditemukan, hasil akhir dari biaya kedua algoritma ini dapat dihitung dengan memperlihatkan biaya jarak perjalanan dari node akhir itu sendiri. Konsep ini dapat dilihat pada **Gambar 5.3**.

No	Biaya Dijkstra	Biaya A-Bintang
1	18	18

2	16	16
3	18	18
4	16	16
5	98	98
6	98	98
7	98	98
8	98	98
9	62	62
10	54	54
11	62	62
12	54	54
13	60	60
14	66	66
15	60	60
16	66	66
17	40	40
18	33	33
19	40	40
20	33	33
21	47	47
22	54	56
23	47	47
24	54	54
25	40	40
26	20	20
27	27	28
28	33	33
29	24	24
30	24	24
31	24	24
32	23	23





33	116	116
34	72	72
35	38	38
36	95	95
37	50	50
38	52	52
39	59	59
40	53	53
41	30	30
42	34	34
43	35	35
44	29	29
45	37	37
46	37	41
47	37	37
48	37	37
49	31	31
50	27	27
51	31	31
52	28	28
53	51	51
54	51	51
55	51	53
56	50	50
57	24	24
58	24	24
59	24	24
60	24	24
Dijkstra: 4		
A-Bintang: 0		
Tabel 5.3 Perbandingan Biaya Perjalanan		

Dalam perbandingan antara kedua algoritma ini, Dijkstra unggul dalam hal biaya perjalanan, namun kecepatan tempuh yang dimiliki oleh A-Bintang tidak dapat disaingi. Meskipun Dijkstra memiliki keunggulan 4 pada total 60 pengujian dalam hal biaya perjalanan, keefektifan A-Bintang tetap lebih unggul. Meskipun beberapa pengujian tidak menggunakan biaya medan sebagai simpul-simpul di peta yang ditempuh, A-Bintang terus menunjukkan keunggulan kecepatan, baik pada peta bermedan maupun tidak bermedan.

5.4 Perbandingan Node Dilalui

Untuk membandingkan elemen node-node yang dilalui, hal ini hanya mencakup seberapa banyak node yang ditemukan selama pencarian jalur yang telah selesai, atau jalur hasil pencarian algoritma yang berisi node-node dalam graf.

Karena elemen node-node yang dilalui adalah node-node hasil dari pencarian itu sendiri, untuk membandingkan prospek dari kedua algoritma ini, menggunakan jumlah node yang ditemukanlah caranya. Jumlah ini dapat menentukan sejauh mana langkah yang ditempuh oleh algoritma dalam mencapai node yang dicarinya, dan dapat digunakan sebagai metrik untuk membandingkan efisiensi dari kedua algoritma.

```
log += " |> Node Traveled = " + str(finishNodes.size()) ;  
  
    func ProduceFinishedPath() -> void:  
        finishNodes.push_back(endNode);  
        var currentNode:PathNode = endNode.previous;  
  
        while(currentNode != null):  
            finishNodes.push_front(currentNode);  
            currentNode = currentNode.previous;
```

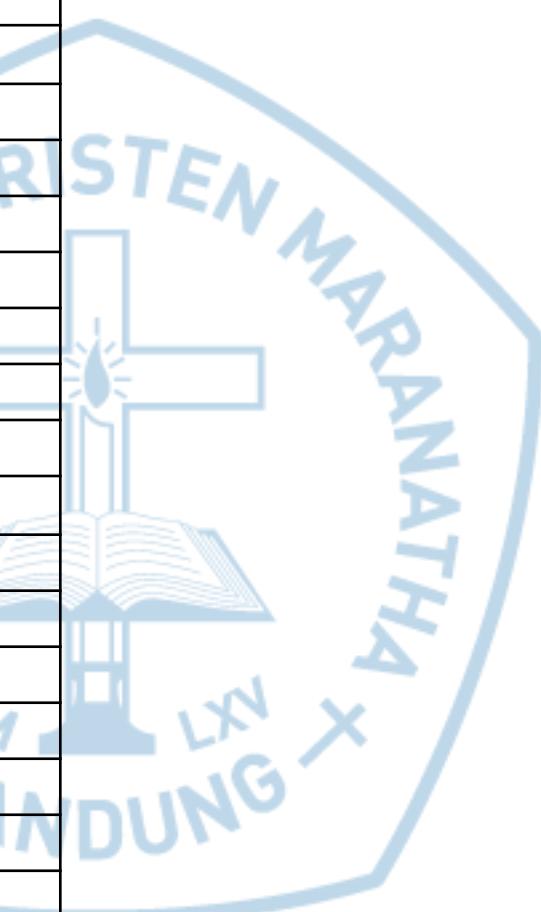
Gambar 5.4 Program Mengetahui Banyaknya Node Yang Dilalui

Fungsi menentukan jumlah node yang telah dilalui, langkah awalnya adalah melakukan pencarian jalur menggunakan algoritma yang relevan. Setelah berhasil mendapatkan hasil pencarian jalur, langkah selanjutnya adalah menghitung jumlah node yang terdapat dalam jalur tersebut. Dengan melakukan proses ini, dapat diperoleh informasi yang akurat mengenai seberapa banyak node yang telah dilalui selama proses pencarian jalur menggunakan algoritma tertentu.

Seperti yang terlihat pada **Gambar 5.4**, implementasi fungsi ini terdapat pada **ProduceFinishedPath()**. Fungsi ini bertujuan untuk menentukan jalur hasil dari pencarian jalur. Proses ini dilakukan melalui iterasi berulang, melangkah dari satu node ke node lain dalam graf, di mana setiap node memiliki induk yang merupakan node sebelumnya dalam hasil pencarian tersebut. Iterasi dilakukan hingga mencapai node awal pencarian.

No	Jumlah Node Dilalui Dijkstra	Jumlah Node Dilalui A-Bintang
1	14	17
2	14	14
3	14	17
4	12	12
5	88	90
6	87	89
7	88	99
8	87	88
9	52	52
10	46	46
11	52	52
12	46	46
13	52	54
14	59	59
15	53	53

16	59	59
17	33	26
18	26	25
19	26	32
20	25	25
21	39	39
22	40	42
23	39	41
24	40	40
25	29	30
26	16	16
27	20	21
28	24	26
29	24	24
30	24	24
31	22	22
32	21	24
33	99	105
34	72	72
35	38	38
36	88	88
37	39	39
38	43	44
39	48	48
40	45	45
41	29	29
42	31	32
43	32	33
44	28	28
45	29	28
46	27	27



47	29	28
48	27	27
49	24	23
50	25	25
51	23	24
52	22	22
53	37	37
54	36	36
55	37	37
56	36	35
57	22	23
58	22	23
59	22	23
60	23	25
Dijkstra: 20		
A-Bintang: 6		
Tabel 5.4 Perbandingan Node Dilalui		

Dalam pertandingan kali ini bisa dikatakan hasil dari pengujian dapat dikatakan imbang. Tidak seperti membandingkan kecepatan dan biaya perjalanan, seperti algoritma A-Bintang dibentuk untuk mencari lebih cepat, atau algoritma Dijkstra yang fungsinya mencari keseluruhan node hingga mendapatkan biaya tempuh terkecil. Perbandingan kali ini, kedua algoritma

5.5 Perbandingan Node Dikunjungi

Berbeda dengan membandingkan elemen-elemen lainnya, mengetahui elemen node yang sudah dikunjungi memiliki cara yang sederhana. Hal ini karena elemen ini sangat terkait dengan daftar frontier (frontier list) dalam algoritma. Oleh karena itu, jumlah node yang diekspansi sejajar dengan jumlah node yang telah dikunjungi.

```

log += " |> Node Expanded = " + str(expanded_nodes);
.
.
.
expanded_nodes += frontierNodes.size() - previous_expanded;

```

Gambar 5.5 Mengetahui Jumlah Node Yang Dikunjungi

Formula untuk mencari node-node yang sudah dikunjungi adalah dengan menyimpan hasil penambahan dari keseluruhan list frontier yang telah diproses oleh algoritma. Namun, karena adanya duplikasi node pada frontier, hasil penyimpanannya dikurangi dengan jumlah node ganda tersebut. Dengan demikian, perhitungan node-node frontier menjadi unik dalam konteks program ini. Penerapan dari formula ini dalam program dapat dilihat pada **Gambar 5.5**.

No	Jumlah Node Dikunjungi Dijkstra	Jumlah Node Dikunjungi A-Bintang
1	45	40
2	42	29
3	47	47
4	47	39
5	1758	825
6	1758	723
7	1758	349
8	1758	880
9	80	76
10	79	77
11	87	87
12	82	76
13	347	176
14	356	323
15	345	257
16	356	204

17	190	168
18	195	149
19	198	190
20	176	101
21	449	258
22	449	400
23	445	328
24	447	247
25	162	120
26	123	84
27	124	84
28	162	120
29	173	156
30	172	82
31	171	96
32	169	140
33	3351	1735
34	3546	794
35	2352	182
36	3562	1875
37	59	43
38	104	104
39	104	104
40	79	59
41	148	129
42	150	106
43	146	106
44	135	118
45	194	163
46	196	166
47	199	169



48	192	169
49	138	79
50	138	88
51	138	134
52	138	123
53	112	93
54	112	104
55	120	118
56	120	119
57	101	77
58	101	57
59	101	65
60	101	78
Dijkstra: 0		
A-Bintang: 54		
Tabel 5.3 Perbandingan Node Dikunjungi		

Sayangnya, pada pengujian perbandingan kali ini, A-Bintang menang secara mutlak dan tidak mendapatkan penentangan yang signifikan dari Dijkstra, dengan mendapatkan skor 54, sebagaimana yang terlihat dalam perbandingan kecepatan. Meskipun demikian, jika dilihat secara inti algoritma, memanglah A-Bintang akan selalu unggul dalam perbandingan jumlah node yang dikunjungi.

Karena, pokok dari algoritma ini adalah mencari jalur dengan lebih cepat dan mengurangi jumlah node yang dijelajahi, tetapi masih mempertahankan efisiensi expansi seperti pada Dijkstra, yang dapat mencari jalur terbaik. Oleh karena itu, poin yang diperoleh dari perbandingan ini menunjukkan keunggulan yang signifikan.

BAB 6

SIMPULAN DAN SARAN

6.1 Simpulan

Jika dilihat dari aspek keunggulan pada elemen pengujian, terlihat bahwa Dijkstra setara dengan A-Bintang. Hal ini dikarenakan pada Biaya Perjalanan dan Jumlah Node Dilalui, algoritma ini unggul, menyamakan skor menjadi 2-2 antara keduanya.

Namun, jika dilihat dari total skor pada setiap kasus pengujian, maka Dijkstra kalah mutlak. Pada pengujian Kecepatan Tempuh dan Jumlah Node Dikunjungi, A-Bintang mendapatkan total poin 118 dari 58 elemen Kecepatan Tempuh, 6 elemen Jumlah Node Dilalui, dan 54 elemen Jumlah Node Dikunjungi, unggul 118 poin dibanding Dijkstra. Dijkstra hanya memiliki keunggulan sebesar 24 poin, yaitu 4 poin dari elemen Biaya Perjalanan dan 20 poin dari elemen Jumlah Node Dikunjungi.

Dengan demikian, dapat ditarik kesimpulan bahwa jika peta atau lingkungan uji pencarian jalur tidak spesifik dalam mencari Biaya Perjalanan terendah atau Jumlah Node Dilalui terpendek, maka A-Bintang adalah algoritma yang cocok untuk berbagai kondisi medan pencarian jalur. Sebaliknya, Dijkstra lebih sesuai digunakan ketika tujuannya adalah mencari secara spesifik Biaya Perjalanan terendah dan Jumlah Node Dilalui terpendek.

6.2 Saran

Penelitian ini hanya melakukan perbandingan antara dua algoritma, yaitu A-Bintang dan Dijkstra. Jika penelitian ini melibatkan beberapa algoritma lain, kemungkinan hasil konklusi dari pengujian dapat berbeda. Oleh karena itu, untuk penelitian selanjutnya, disarankan untuk mengimplementasikan atau menguji algoritma-algoritma lainnya agar dapat memberikan gambaran yang lebih komprehensif dan mendalam terkait dengan perbandingan kinerja algoritma pencarian jalur.

DAFTAR PUSTAKA

- [1] S. J. Russell and P. Norvig, *Artificial intelligence : A Modern Approach, Global Edition*. London: Pearson, 2021. pp. 81, 83, 94-98, 103-107.
- [2] “Autoblocks: Full-stack monitoring, debugging, and testing,” www.autoblocks.ai. <https://www.autoblocks.ai/glossary/pathfinding> (Accessed Oct. 30, 2023).
- [3] “Graph and its representations - GeeksforGeeks,” *GeeksforGeeks*, Nov. 13, 2012. <https://www.geeksforgeeks.org/graph-and-its-representations/> (Accessed Oct. 30, 2023).
- [4] “Grids and Graphs,” www.redblobgames.com. <https://www.redblobgames.com/pathfinding/grids/graphs.html> (Accessed Oct. 30, 2023).
- [5] R. Tamassia and C. Press, *Handbook of graph drawing and visualization*. Boca Raton, Fl: Crc Press, An Imprint Of Chapman And Hall/Crc, 2013, pp. 156, 198-199.
- [6] “Red Blob Games: Hexagonal Grids,” www.redblobgames.com. <https://www.redblobgames.com/grids/hexagons/> (Accessed Oct. 30, 2023).
- [7] R. Graham, H. McCabe, and S. Sheridan, “Issue 2 Article 6 2003 Part of the Computer and Systems Architecture Commons,” *The ITB Journal The ITB Journal*, vol. 4, no. 2, 2003, doi: <https://doi.org/10.21427/D7ZQ9J>.
- [8] A. Patel, “Heuristics,” [Stanford.edu](http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html), 2019. <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html> (Accessed Oct. 30, 2023).
- [9] N. Krishnaswamy, “Comparison of Efficiency in Pathfinding Algorithms in Game Development,” Jan. 2009, pp. 19
- [10] B. Sobota, C. Szabo and J. Perhac, "Using path-finding algorithms of graph theory for route-searching in geographical information systems," 2008 6th

International Symposium on Intelligent Systems and Informatics, Subotica, Serbia, 2008, pp. 1.

- [11] Narsingh Deo, *Graph theory with applications to engineering & computer science*. Mineola, New York: Dover Publications, Inc, 2016, pp. 1-2.
- [12] “Graph and its representations - GeeksforGeeks,” *GeeksforGeeks*, Nov. 13, 2012. <https://www.geeksforgeeks.org/graph-and-its-representations/> (Accessed Oct. 30, 2023).
- [13] John Adrian Bondy and R. Murty, *Graph Theory with Applications*. London : Macmillan Press, 1976, pp. 243-246.
- [14] “Applications, Advantages and Disadvantages of Weighted Graph,” *GeeksforGeeks*, May 19, 2022. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-weighted-graph/> (Accessed Oct. 30, 2023).
- [15] “Dijkstra’s Algorithm – Explained with a Pseudocode Example,” *freeCodeCamp.org*, Dec. 01, 2022. <https://www.freecodecamp.org/news/dijkstras-algorithm-explained-with-a-pseudocode-example/> (Accessed Oct. 30, 2023).
- [16] geeksforgeeks, “What is Dijkstra’s Algorithm? | Introduction to Dijkstra’s Shortest Path Algorithm,” *GeeksforGeeks*, Mar. 10, 2023. <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/> (Accessed Oct. 30, 2023).
- [17] A. Patel, “Introduction to A*,” *Stanford.edu*, 2019. <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html> (Accessed Oct. 30, 2023).

- [18] Red Blob Games, “Red Blob Games: Introduction to A*,” *Redblobgames.com*, 2014. <https://www.redblobgames.com/pathfinding/a-star/introduction.html> (Accessed Oct. 30, 2023).
- [19] J. Brownlee, “4 Distance Measures for Machine Learning,” *Machine Learning Mastery*, Mar. 24, 2020. <https://machinelearningmastery.com/distance-measures-for-machine-learning/>



LAMPIRAN A TABEL UJIAN

Tabel Lampiran Data Terkumpul Dijkstra					
NO	Percobaan	Kecepatan	Biaya	Jumlah Node Dilalui	Jumlah Node Dikunjungi
1	Peta 1 Percobaan 1	0.65	18	14	45
2	Peta 1 Percobaan 2	0.58	16	14	42
3	Peta 1 Percobaan 3	0.75	18	14	47
4	Peta 1 Percobaan 4	0.73	16	12	47
5	Peta 2 Percobaan 1	29.27	98	88	1758
6	Peta 2 Percobaan 2	29.25	98	87	1758
7	Peta 2 Percobaan 3	29.27	98	88	1758
8	Peta 2 Percobaan 4	29.25	98	87	1758
9	Peta 3 Percobaan 1	1.23	62	52	80
10	Peta 3 Percobaan 2	1.25	54	46	79
11	Peta 3 Percobaan 3	1.43	62	52	87
12	Peta 3 Percobaan 4	1.28	54	46	82
13	Peta 4 Percobaan 1	5.72	60	52	347
14	Peta 4 Percobaan 2	5.9	66	59	356

15	Peta 4 Percobaan 3	5.65	60	53	345
16	Peta 4 Percobaan 4	5.88	66	59	356
17	Peta 5 Percobaan 1	2.93	40	33	190
18	Peta 5 Percobaan 2	2.93	33	26	195
19	Peta 5 Percobaan 3	3.2	40	26	198
20	Peta 5 Percobaan 4	2.65	33	25	176
21	Peta 6 Percobaan 1	7.27	47	39	449
22	Peta 6 Percobaan 2	7.42	54	40	449
23	Peta 6 Percobaan 3	7.22	47	39	445
24	Peta 6 Percobaan 4	7.25	54	40	447
25	Peta 7 Percobaan 1	2.52	40	29	162
26	Peta 7 Percobaan 2	1.83	20	16	123
27	Peta 7 Percobaan 3	1.85	27	20	124
28	Peta 7 Percobaan 4	2.52	33	24	162
29	Peta 8 Percobaan 1	2.77	24	24	173
30	Peta 8 Percobaan 2	2.48	24	24	172
31	Peta 8 Percobaan 3	2.72	24	22	171
32	Peta 8 Percobaan 4	2.67	23	21	169

33	Peta 9 Percobaan 1	54.63	116	99	3351
34	Peta 9 Percobaan 2	56.67	72	72	3546
35	Peta 9 Percobaan 3	36.8	38	38	2352
36	Peta 9 Percobaan 4	58.13	95	88	3562
37	Peta 10 Percobaan 1	null	null	null	49
38	Peta 10 Percobaan 2	1.72	50	43	104
39	Peta 10 Percobaan 3	1.72	59	48	104
40	Peta 10 Percobaan 4	null	null	null	49
41	Peta 11 Percobaan 1	2.35	30	29	148
42	Peta 11 Percobaan 2	2.35	34	31	150
43	Peta 11 Percobaan 3	2.3	35	32	146
44	Peta 11 Percobaan 4	2.15	29	28	135
45	Peta 12 Percobaan 1	3	37	29	194
46	Peta 12 Percobaan 2	2.9	37	27	196
47	Peta 12 Percobaan 3	3.13	37	29	199
48	Peta 12 Percobaan 4	2.98	37	27	192
49	Peta 13 Percobaan 1	2.27	31	24	138
50	Peta 13 Percobaan 2	2.27	27	25	138

51	Peta 13 Percobaan 3	2.25	31	23	138
52	Peta 13 Percobaan 4	2.25	28	22	138
53	Peta 14 Percobaan 1	1.82	51	37	112
54	Peta 14 Percobaan 2	1.78	51	36	112
55	Peta 14 Percobaan 3	1.95	51	37	120
56	Peta 14 Percobaan 4	1.97	50	36	120
57	Peta 15 Percobaan 1	1.62	24	22	101
58	Peta 15 Percobaan 2	1.65	24	22	101
59	Peta 15 Percobaan 3	1.65	24	22	101
60	Peta 15 Percobaan 4	1.65	24	23	101

Tabel Lampiran Data Terkumpul A-Bintang

No	Percobaan	Kecepatan	Biaya	Jumlah Node Dilalui	Jumlah Node Dikunjungi
1	Peta 1 Percobaan 1	0.52	18	17	40
2	Peta 1 Percobaan 2	0.38	16	14	29
3	Peta 1 Percobaan 3	0.65	18	17	47
4	Peta 1 Percobaan 4	0.48	16	12	39

5	Peta 2 Percobaan 1	9.65	98	90	825
6	Peta 2 Percobaan 2	7.7	98	89	723
7	Peta 2 Percobaan 3	3.33	98	99	349
8	Peta 2 Percobaan 4	9.57	98	88	880
9	Peta 3 Percobaan 1	1.18	62	52	76
10	Peta 3 Percobaan 2	1.23	54	46	77
11	Peta 3 Percobaan 3	1.42	62	52	87
12	Peta 3 Percobaan 4	1.18	54	46	76
13	Peta 4 Percobaan 1	2.57	60	54	176
14	Peta 4 Percobaan 2	5.13	66	59	323
15	Peta 4 Percobaan 3	3.98	60	53	257
16	Peta 4 Percobaan 4	3.05	66	59	204
17	Peta 5 Percobaan 1	2.22	40	26	168
18	Peta 5 Percobaan 2	1.58	33	25	149
19	Peta 5 Percobaan 3	2.75	40	32	190
20	Peta 5 Percobaan 4	1.13	33	25	101
21	Peta 6 Percobaan 1	3.18	47	39	258
22	Peta 6 Percobaan 2	6.1	56	42	400

23	Peta 6 Percobaan 3	4.18	47	41	328
24	Peta 6 Percobaan 4	2.65	54	40	247
25	Peta 7 Percobaan 1	1.82	40	30	120
26	Peta 7 Percobaan 2	1.25	20	16	84
27	Peta 7 Percobaan 3	1.18	28	21	84
28	Peta 7 Percobaan 4	1.8	33	26	120
29	Peta 8 Percobaan 1	1.87	24	24	156
30	Peta 8 Percobaan 2	1.08	24	24	82
31	Peta 8 Percobaan 3	1.02	24	22	96
32	Peta 8 Percobaan 4	1.58	23	24	140
33	Peta 9 Percobaan 1	27.18	116	105	1735
34	Peta 9 Percobaan 2	11.68	72	72	794
35	Peta 9 Percobaan 3	1.77	38	38	182
36	Peta 9 Percobaan 4	20.37	95	88	1875
37	Peta 10 Percobaan 1	null	null	null	49
38	Peta 10 Percobaan 2	1.72	52	44	104
39	Peta 10 Percobaan 3	1.72	59	48	104
40	Peta 10 Percobaan 4	null	null	null	49

41	Peta 11 Percobaan 1	1.9	30	29	129
42	Peta 11 Percobaan 2	1.58	34	32	106
43	Peta 11 Percobaan 3	1.58	35	33	106
44	Peta 11 Percobaan 4	1.73	29	28	118
45	Peta 12 Percobaan 1	2.23	37	28	163
46	Peta 12 Percobaan 2	2.08	41	27	166
47	Peta 12 Percobaan 3	1.67	37	28	169
48	Peta 12 Percobaan 4	2.37	37	27	169
49	Peta 13 Percobaan 1	0.9	31	23	79
50	Peta 13 Percobaan 2	0.78	27	25	88
51	Peta 13 Percobaan 3	1.87	31	24	134
52	Peta 13 Percobaan 4	1.58	28	22	123
53	Peta 14 Percobaan 1	1.43	51	37	93
54	Peta 14 Percobaan 2	1.57	51	36	104
55	Peta 14 Percobaan 3	1.83	53	37	118
56	Peta 14 Percobaan 4	1.9	50	35	119
57	Peta 15 Percobaan 1	0.93	24	23	77
58	Peta 15 Percobaan 2	0.77	24	23	57

59	Peta 15 Percobaan 3	0.7	24	23	65
60	Peta 15 Percobaan 4	1.03	24	25	78



LAMPIRAN B GAMBAR TEKS MAP

5	1 23432000000000000000234 2 23320022222200002343 3 02220022333220002343 4 00000223443220023432 5 22200223444322023432 6 33200223443220234320 7 34320022333220234320 8 3320022222202343200 9 2220023322200233200 10 00002343200000000000
6	1 000000000222000000000233320000 2 00000022222222000002234322000 3 00000022333332000002233322000 4 000000223443322000000222220000 5 000002223443322000000220000000 6 000002222344322220000000000000 7 000000022333322200000000000000 8 000000022223322200000000000000 9 00000000022222200000000000000000 10 00000000022220222222229000000 11 022220000000000223333222000000 12 222222200000000234432220000000 13 3333220000000223443220000000 14 444432220000000223333220000000 15 33334322000000222222200000000

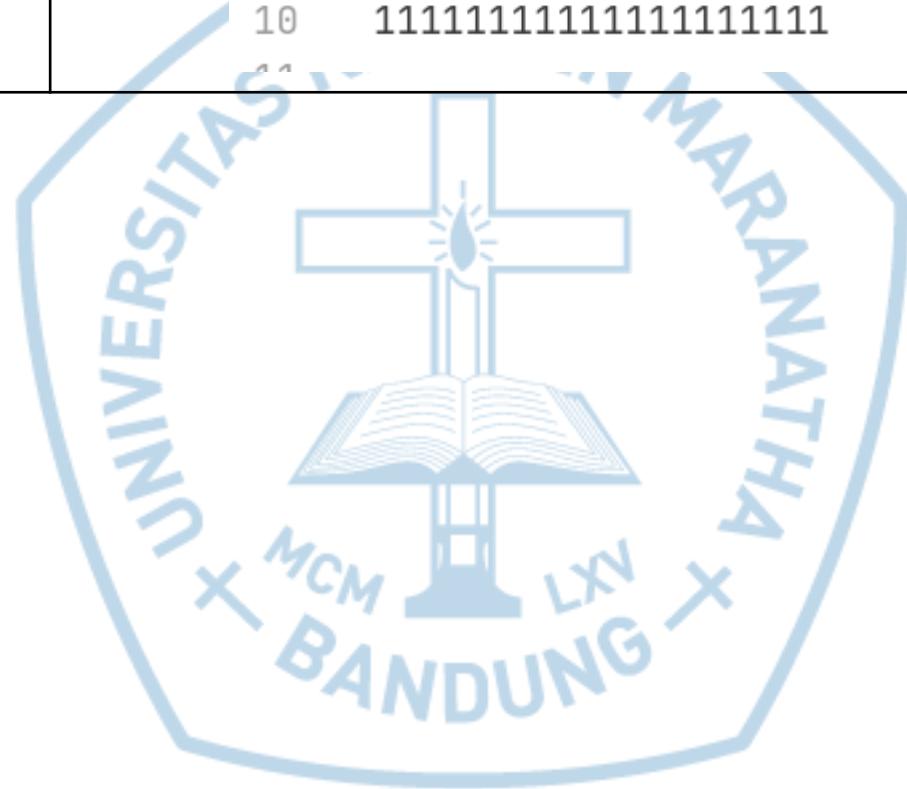
7	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">00010000000000000000</td></tr> <tr><td style="padding: 2px;">2</td><td style="padding: 2px;">00010000000000000000</td></tr> <tr><td style="padding: 2px;">3</td><td style="padding: 2px;">00010001001000010000</td></tr> <tr><td style="padding: 2px;">4</td><td style="padding: 2px;">00010001001222010000</td></tr> <tr><td style="padding: 2px;">5</td><td style="padding: 2px;">00010001001000010000</td></tr> <tr><td style="padding: 2px;">6</td><td style="padding: 2px;">00010001001033310000</td></tr> <tr><td style="padding: 2px;">7</td><td style="padding: 2px;">00010001001000010000</td></tr> <tr><td style="padding: 2px;">8</td><td style="padding: 2px;">00010001001444010000</td></tr> <tr><td style="padding: 2px;">9</td><td style="padding: 2px;">00000001001000010000</td></tr> <tr><td style="padding: 2px;">10</td><td style="padding: 2px;">00000001001000010000</td></tr> </table>	1	00010000000000000000	2	00010000000000000000	3	00010001001000010000	4	00010001001222010000	5	00010001001000010000	6	00010001001033310000	7	00010001001000010000	8	00010001001444010000	9	00000001001000010000	10	00000001001000010000
1	00010000000000000000																				
2	00010000000000000000																				
3	00010001001000010000																				
4	00010001001222010000																				
5	00010001001000010000																				
6	00010001001033310000																				
7	00010001001000010000																				
8	00010001001444010000																				
9	00000001001000010000																				
10	00000001001000010000																				
8	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">00000000000000000000</td></tr> <tr><td style="padding: 2px;">2</td><td style="padding: 2px;">00000000000000000000</td></tr> <tr><td style="padding: 2px;">3</td><td style="padding: 2px;">0000111111111110000</td></tr> <tr><td style="padding: 2px;">4</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">5</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">6</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">7</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">8</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">9</td><td style="padding: 2px;">000000000000000010000</td></tr> <tr><td style="padding: 2px;">10</td><td style="padding: 2px;">00000000000000000000</td></tr> </table>	1	00000000000000000000	2	00000000000000000000	3	0000111111111110000	4	000000000000000010000	5	000000000000000010000	6	000000000000000010000	7	000000000000000010000	8	000000000000000010000	9	000000000000000010000	10	00000000000000000000
1	00000000000000000000																				
2	00000000000000000000																				
3	0000111111111110000																				
4	000000000000000010000																				
5	000000000000000010000																				
6	000000000000000010000																				
7	000000000000000010000																				
8	000000000000000010000																				
9	000000000000000010000																				
10	00000000000000000000																				

11	1 000000000100000000 2 0000000001000010000 3 0000000001000010000 4 0111111111111110000 5 0000000000000000000000 6 0111111111111110000 7 0000000001000010000 8 0000000001000010000 9 0000000001000010000 10 0000000001000000000
12	1 00220330440000300440 2 00220330440220330400 3 00220330440220330440 4 00220330440220330400 5 00220330440220330400 6 00000000000220330400 7 00220330440220330400 8 00220330440220330440 9 00220330440220330400 10 00220330440000300440

13	<pre>1 01100330011003300110 2 10101010101010101010 3 04400110022001100440 4 00000000000000000000 5 02200110044001100220 6 10101010101010101010 7 01100330011003300110 8 00000000000000000000 9 10101010101010101010 10 11001100110011001100</pre>
14	<pre>1 10001010100010101010 2 01010100100101010010 3 10001001010100101001 4 01010100101001010100 5 00101010010100101010 6 10010100101010010100 7 01001010010100101001 8 00100101001010010101 9 10100010010010100010 10 00010100001001010010</pre>

15

- | | |
|----|----------------------|
| 1 | 11111111111111111111 |
| 2 | 10000000000000000001 |
| 3 | 10111111111111111101 |
| 4 | 10000000000000000001 |
| 5 | 10001111111111110001 |
| 6 | 1000000000000010001 |
| 7 | 10001000000000000001 |
| 8 | 10001111111111110001 |
| 9 | 10000000000000000001 |
| 10 | 11111111111111111111 |



LAMPIRAN C NAMA LAMPIRAN



RIWAYAT HIDUP PENULIS

Riwayat hidup dibuat dengan baik dan benar meliputi hal-hal berikut ini:

1. Identitas diri
2. Riwayat pendidikan
3. Riwayat pekerjaan
4. Organisasi yang pernah diikuti
5. Prestasi yang pernah diraih
6. Hasil karya yang pernah dibuat
7. Sertifikat (contohnya: SAP, CISCO, Microsoft, keikutsertaan seminar, konferensi, panitia, dll.)

Pas foto
formal resmi

