

# **Membandingkan Algoritma Pencarian Jalur A\* 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

Membandingkan Algoritma Pencarian Jalur A\* 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)

Menyetujui,

Pembimbing I

Pembimbing II (Jika Ada)

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 *Membandingkan Pencarian Jalur A\* dan Dijkstra Pada Lima Belas Peta Berbasis Heksagonal*.
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**

(Style: abstrak) Jumlah kata pada abstrak adalah 150-200 kata. Gunakanlah style abstrak. Pada prinsipnya, abstrak akan memberikan rangkuman dari laporan Anda secara keseluruhan. Jika seseorang membaca abstrak, maka pembaca akan tahu apa keseluruhan isi laporan dari latar belakang sampai simpulan dan saran. Abstrak disarankan untuk memuat hal-hal berikut ini yaitu pokok masalah yang dibahas (Mengapa melakukan KP/ TA dengan topik ini?); tujuan pembahasan (Apa yang ingin dicapai?); teori yang digunakan, sumber data; metode dan teknik penelitian (metode adalah cara menganalisis/ memecahkan sedangkan teknik penelitian adalah cara pengumpulan data); temuan ilmiah/ jawaban pembahasan/ hasil dan simpulan yang dicapai.

Kata kunci: frase 1, frase 2, dst. (maksimum 6, urutkan abjad)



## ABSTRACT

(Style: Abstract) Jumlah kata pada abstract adalah 150-200 kata. Abstract Bahasa Inggris harus dibuat sesuai dengan abstrak Bahasa Indonesia-nya. Harap menggunakan English grammar yang baik dan benar. Tidak disarankan menggunakan Google Translate karena belum tentu sesuai dengan kalimat yang dimaksud. Namun Google Translate bisa membantu Anda untuk mencari kata-kata yang Anda tidak tahu Bahasa Inggris-nya.

Keywords: phrase 1, phrase 2, etc. (max 6 phrases, sorted alphabetically)



# DAFTAR ISI

<b>LEMBAR PENGESAHAN</b>	<b>1</b>
<b>PERNYATAAN ORISINALITAS LAPORAN PENELITIAN</b>	<b>2</b>
<b>PERNYATAAN PUBLIKASI LAPORAN PENELITIAN</b>	<b>3</b>
<b>PRAKATA</b>	<b>4</b>
<b>ABSTRAK</b>	<b>5</b>
<b>ABSTRACT</b>	<b>6</b>
<b>DAFTAR ISI</b>	<b>7</b>
<b>DAFTAR GAMBAR</b>	<b>7</b>
<b>DAFTAR TABEL</b>	<b>7</b>
<b>DAFTAR NOTASI/ LAMBANG</b>	<b>7</b>
<b>DAFTAR SINGKATAN</b>	<b>8</b>
<b>DAFTAR ISTILAH</b>	<b>8</b>
<b>BAB 1 PENDAHULUAN</b>	<b>8</b>
1.1 Latar Belakang	8
1.2 Rumusan Masalah	2
1.3 Tujuan Pembahasan	3
1.4 Ruang Lingkup	3
1.5 Sumber Data	4
1.6 Sistematika Penyajian	4
<b>BAB 2 KAJIAN TEORI</b>	<b>4</b>
2.1 Algoritma Penjelajahan Graf	5
2.1.1 Breadth First Search	6
2.1.2 A-Bintang	9
2.1.2.1 Fungsi Heuristik	11
<b>BAB 3 ANALISIS DAN RANCANGAN SISTEM</b>	<b>12</b>
3.1 Tipe-Tipe Node	12
3.2 Simulasi Algoritma	14
3.2.1 A-Bintang	17
3.2.2 Breadth First Search Saja	19
3.2.3 A-Bintang Dengan Pengacau Biaya	23
3.2.4 BFS Dengan Pengacau Biaya	27
3.3 Pemrograman Node	32
3.4 Pemrograman Peta Grid	36
3.5 Pemrograman Algoritma	38
<b>BAB 4 IMPLEMENTASI</b>	<b>39</b>
4.1 Sub Topik Bab 4	39
4.2 Sub Topik Bab 4	39

4.3 Sub Topik Bab 4	39
<b>BAB 5 PENGUJIAN</b>	<b>39</b>
5.1 Sub Topik Bab 5	39
5.2 Pengujian Black Box	39
5.3 Pengujian White Box	39
5.4 User Acceptance Test	39
5.5 Benchmarking	39
<b>BAB 6 SIMPULAN DAN SARAN</b>	<b>39</b>
6.1 Simpulan	39
6.2 Saran	40
<b>DAFTAR PUSTAKA</b>	<b>41</b>



## DAFTAR GAMBAR

Gambar 3.1.1 Node Standar, Tembok, Tujuan.	9
Gambar 3.1.2 Node Awal, Node Saat Ini, Node Kuning.	10
Gambar 3.2.1 Algoritma A-Bintang	17
Gambar 3.2.2 Algoritma Dijkstra	21
<b>BAB 4 IMPLEMENTASI</b>	<b>22</b>
4.1 Sub Topik Bab 4	22
4.2 Sub Topik Bab 4	22
4.3 Sub Topik Bab 4	22
<b>BAB 5 PENGUJIAN</b>	<b>23</b>
5.1 Peta 01	23
5.1.1 Dijkstra	23
5.1.2 A-Bintang	23
<b>BAB 6 SIMPULAN DAN SARAN</b>	<b>25</b>
6.1 Simpulan	25
6.2 Saran	25
<b>DAFTAR PUSTAKA</b>	<b>26</b>

## DAFTAR TABEL

Tabel 2.1 Contoh Pembuatan Tabel

5



## DAFTAR NOTASI / LAMBANG

Jenis	Notasi/ Lambang	Nama	Arti
BPMN 2.0	Isi dengan gambar notasinya		
DFD			
ERD			
UML			
Dst.			

Referensi:

Notasi/ Lambang BPMN 2.0 dari Object Management Group [1]

Notasi/ Lambang DFD dari ....

Notasi/ Lambang ERD dari ....

Dst.



## DAFTAR SINGKATAN

ERD	Entity Relationship Diagram
MMJar	Multi Media Jaringan
RPL	Rekayasa Perangkat Lunak
SI	Sistem Informasi
UML	Unified Modelling Language
Dst ...	



## 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 ini dengan menggambar grid, di mana setiap simpul ditempatkan pada koordinat bidang area. Layar komputer dapat dilihat sebagai grid bidang area, untuk setiap piksel yang ditempatkan pada koordinat itu [5]. Menggambar bidang area pada layar komputer ini dapat dilakukan dengan menggambar grid heksagonal untuk setiap bidang area tersebut [6]. Godot Game Engine merupakan salah satu alat yang dapat digunakan untuk menggambar grid graf pada bidang area.

A-Bintang dan algoritma Dijkstra merupakan agen algoritma pathfinding. Algoritma Dijkstra mengatasi tindakan pencarian jalur ini dengan menentukan biaya edges terbaik 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, di mana 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]:

- Waktu perjalanan jalur.
- Jarak panjang dilalui pada jalur yang ditemukan.
- Jumlah node dilalui pada jalur yang ditemukan.
- Jumlah node dikunjungi secara keseluruhan saat pencarian jalur.

Sehingga masalah yang diteliti adalah sebagai berikut:

1. Bagaimana cara mengetahui parameter-parameter yang didapatkan pada lima belas peta yang diberikan?
2. Bagaimana cara menghitung rata-rata untuk setiap parameter pada lima belas peta yang diberikan?
3. Bagaimana cara mengevaluasi kualitas hasil pencarian jalur pada lima belas peta yang diberikan?

## 1.3 Tujuan Pembahasan

Agar menjawab masalah yang ditemukan, metode yang dilakukan adalah sebagai berikut:

1. Untuk memeriksa parameter waktu, mencari yang tercepat. Untuk parameter jarak panjang, mencari yang terendah. Untuk jumlah node yang dilalui, mencari yang terkecil. Untuk jumlah node yang dikunjungi, mencari yang terkecil pula.
2. Mengganti-ganti posisi node awal dan node tujuan pada kelima belas peta untuk mengumpulkan sejumlah data pada satu peta tersebut, yang nantinya akan dirata-ratakan.
3. Analisis visualisasi jalur yang ditemukan oleh algoritma A-Bintang dan Dijkstra.

## 1.4 Ruang Lingkup

Berikut pembatasan dari penelitian ini:

- Penelitian ini hanya akan berfokus pada perbandingan antara algoritma pencarian A\* dan Dijkstra.
- Implementasi kedua algoritma A\* 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.

## 1.5 Sumber Data

-

## 1.6 Sistematika Penyajian

-

## 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\* (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\* (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 heuristik ( $h$ -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 ( $\infty$ ). 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].

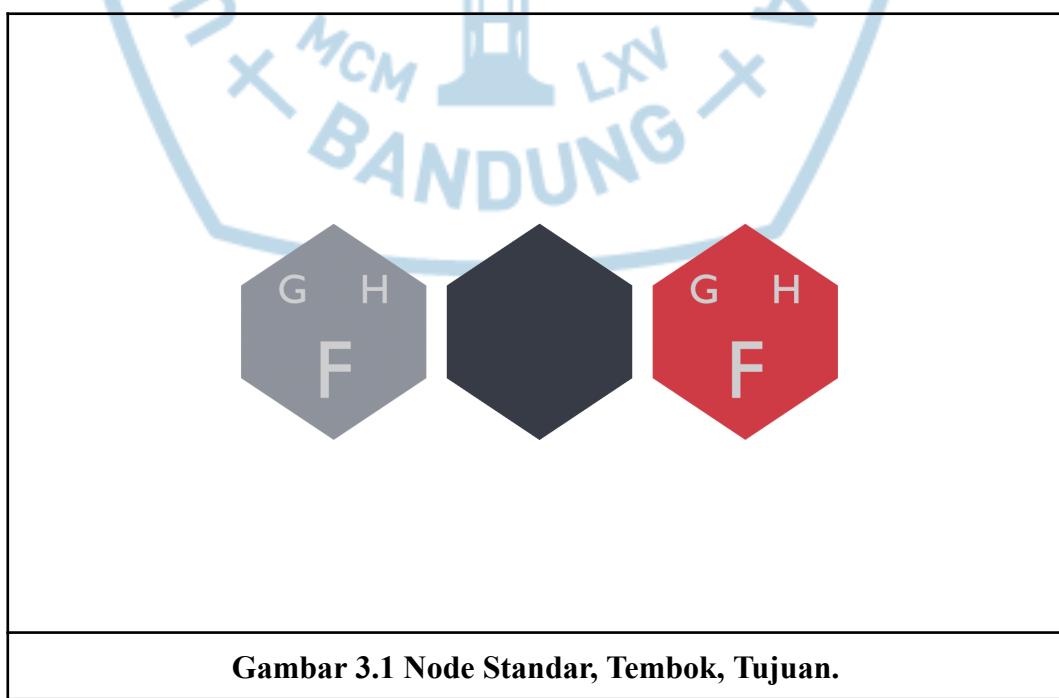


## BAB 3

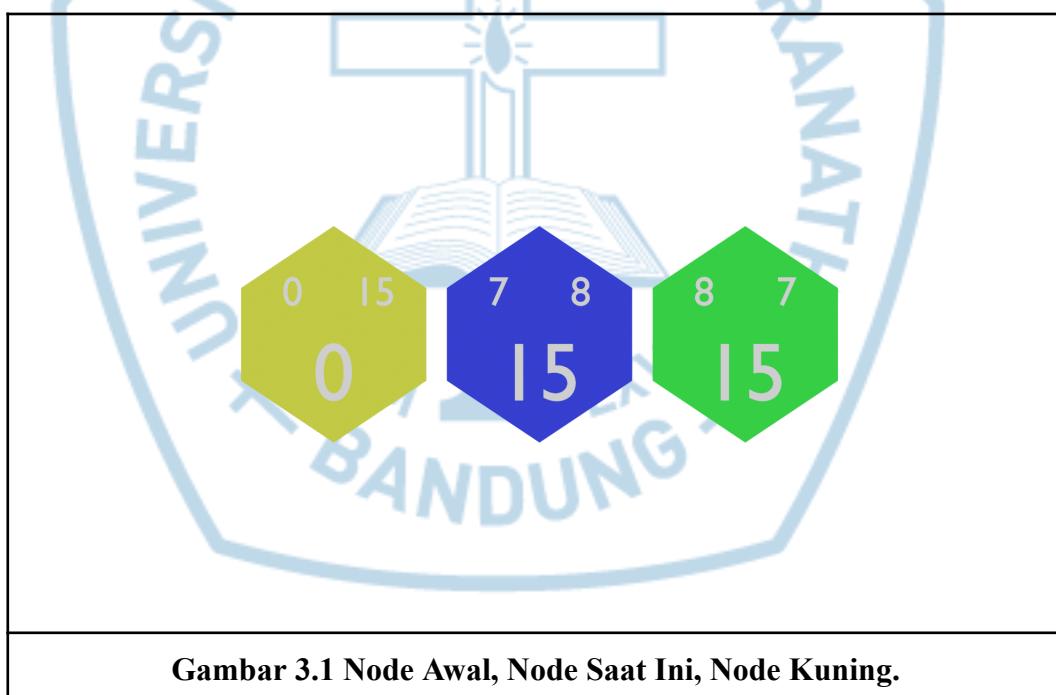
# ANALISIS DAN RANCANGAN SISTEM

### 3.1 Tipe-Tipe Node

Perancangan sistem menggunakan beberapa tipe node. Sebagai contoh, pada **Gambar 3.1.1**, terdapat tiga node dengan warna abu-abu, hitam, dan merah. 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 3.1.1** menunjukkan node pada grid yang belum diproses atau diatur, maka **Gambar 3.1.2** menggambarkan node yang sedang atau sudah diproses dan diatur. Node berwarna kuning adalah node awal pertama yang dipilih sebagai posisi awal pencarian jalur algoritma, di mana 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 menggunakan algoritma A-Bintang, 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.



### 3.2 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.

### 3.3 Simulasi Algoritma

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

- **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.

### 3.3.1 A-Bintang

Berikut penjelasan proses simulasi algoritma pencarian jalur A-Bintang:

#### Langkah 1.

Seperti pada **Gambar 3.2.1**, 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. Dimana 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.

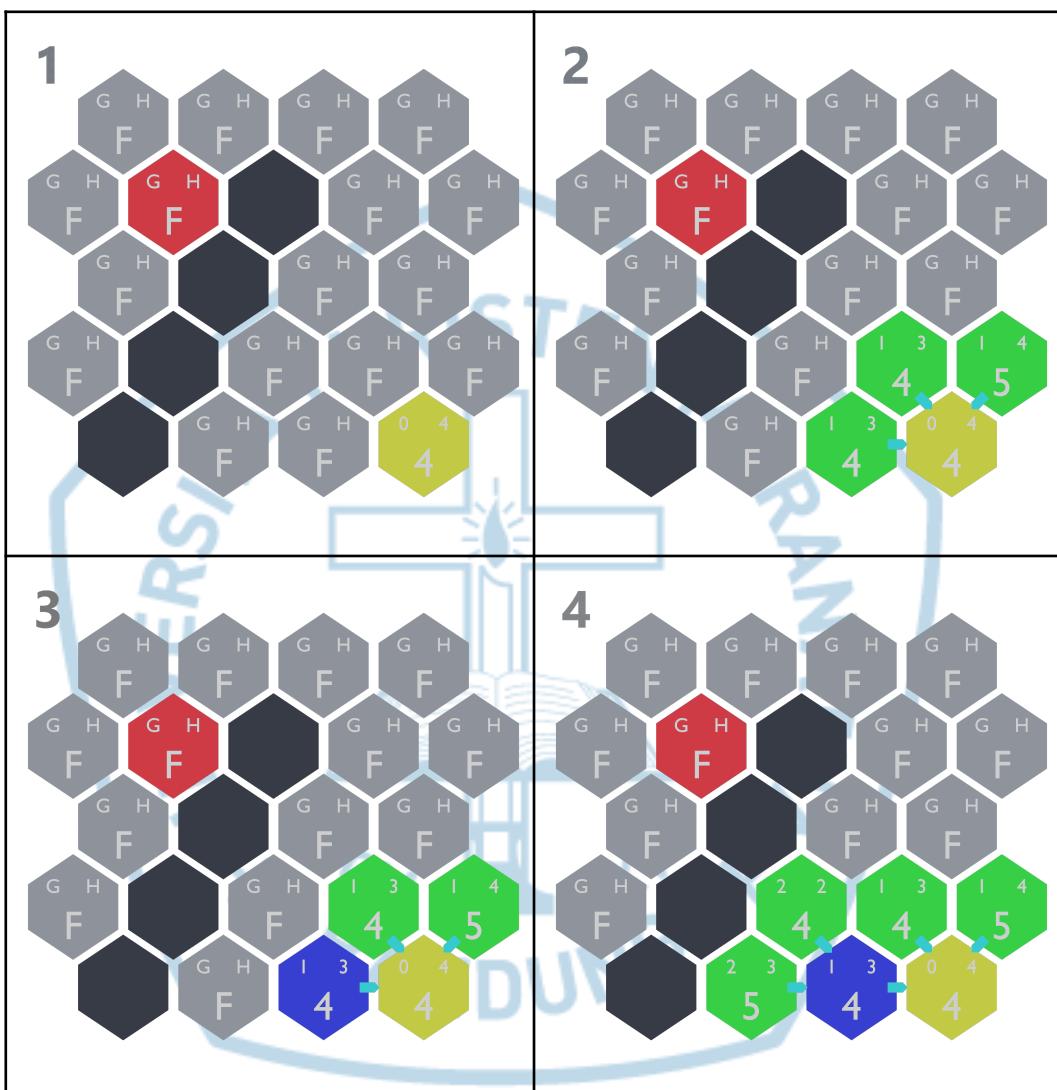
#### **Langkah 9 - 19.**

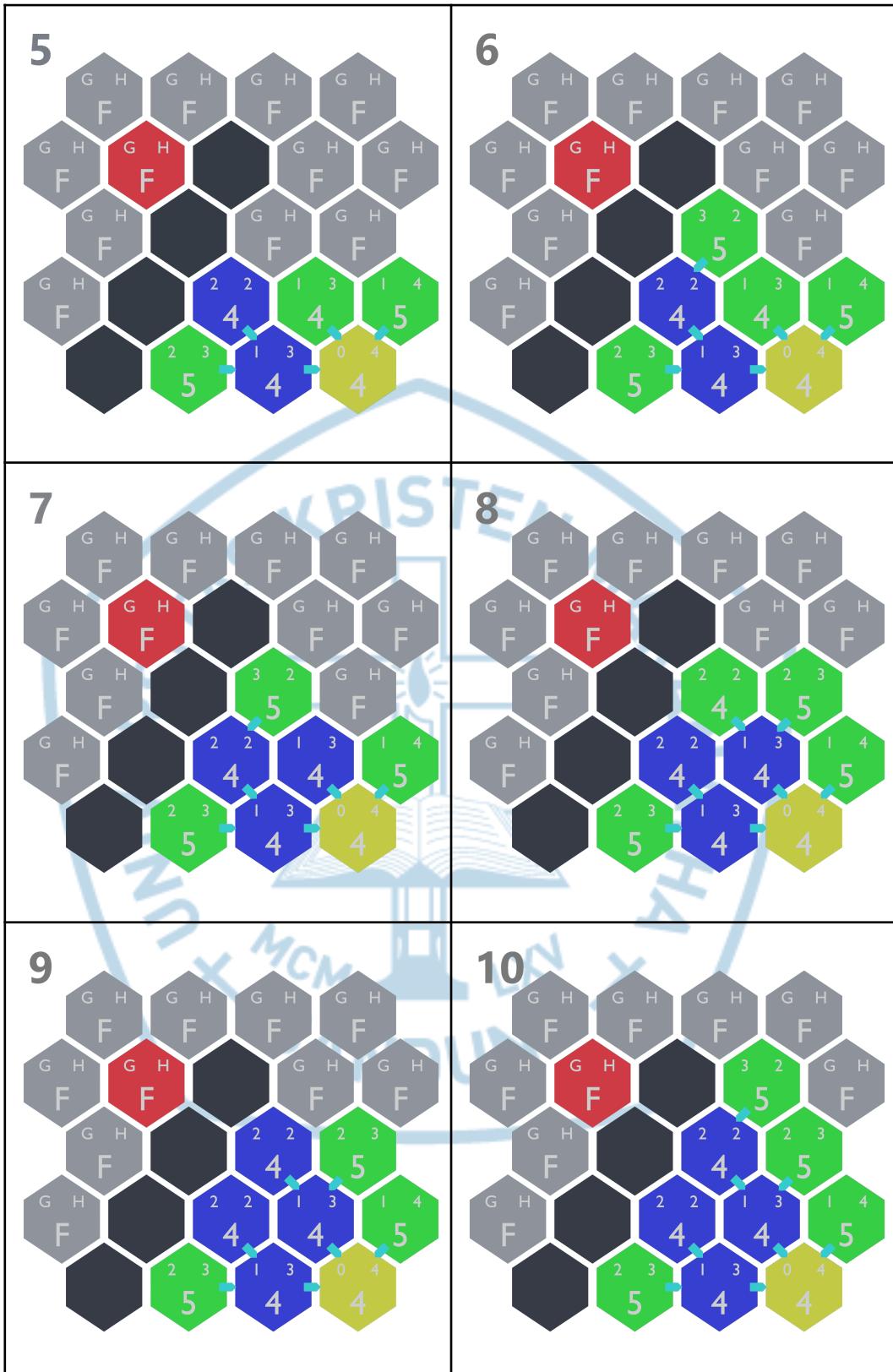
Dari langkah kesembilan hingga kesembilan belas sama proses pencarian jalurnya. Terjadi pengulangan **Langkah 1-5**, ditambah dengan **Langkah 8**.

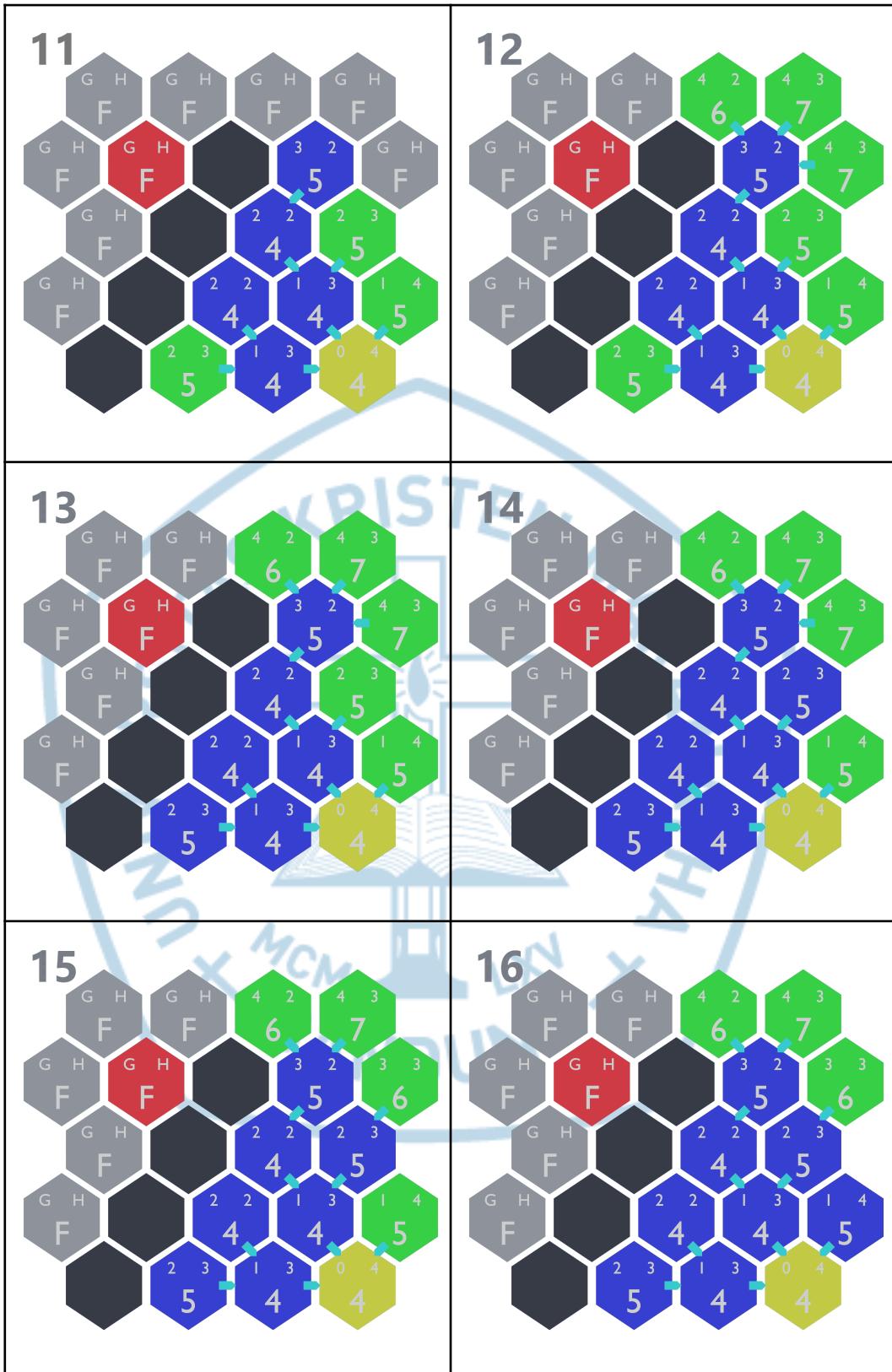
#### **Langkah 20.**

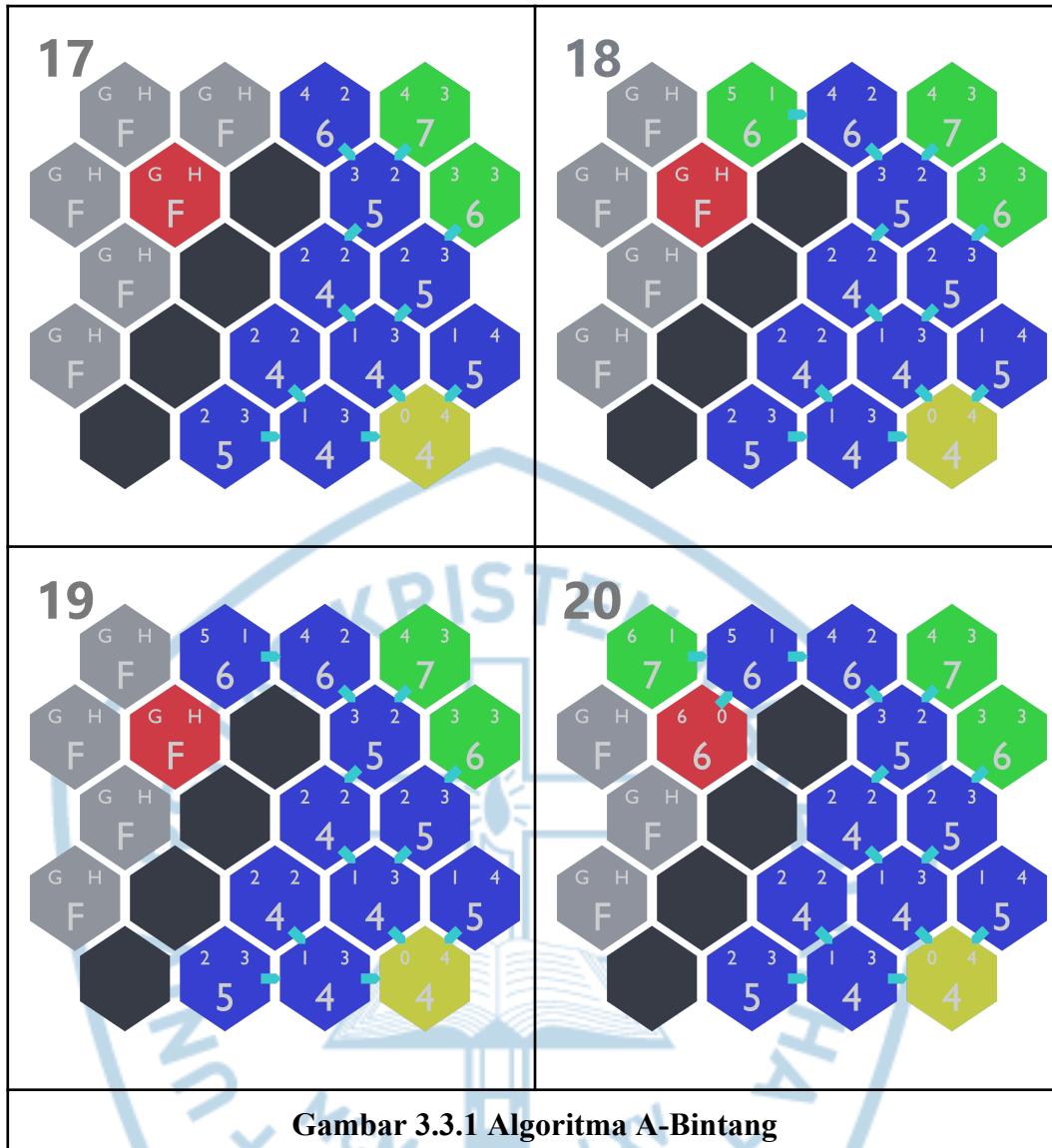
Pada **Langkah 20**, meskipun node merah tujuan telah ditemukan dan algoritma selesai melakukan pencarian jalur.

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.









Gambar 3.3.1 Algoritma A-Bintang

### 3.3.2 Dijkstra

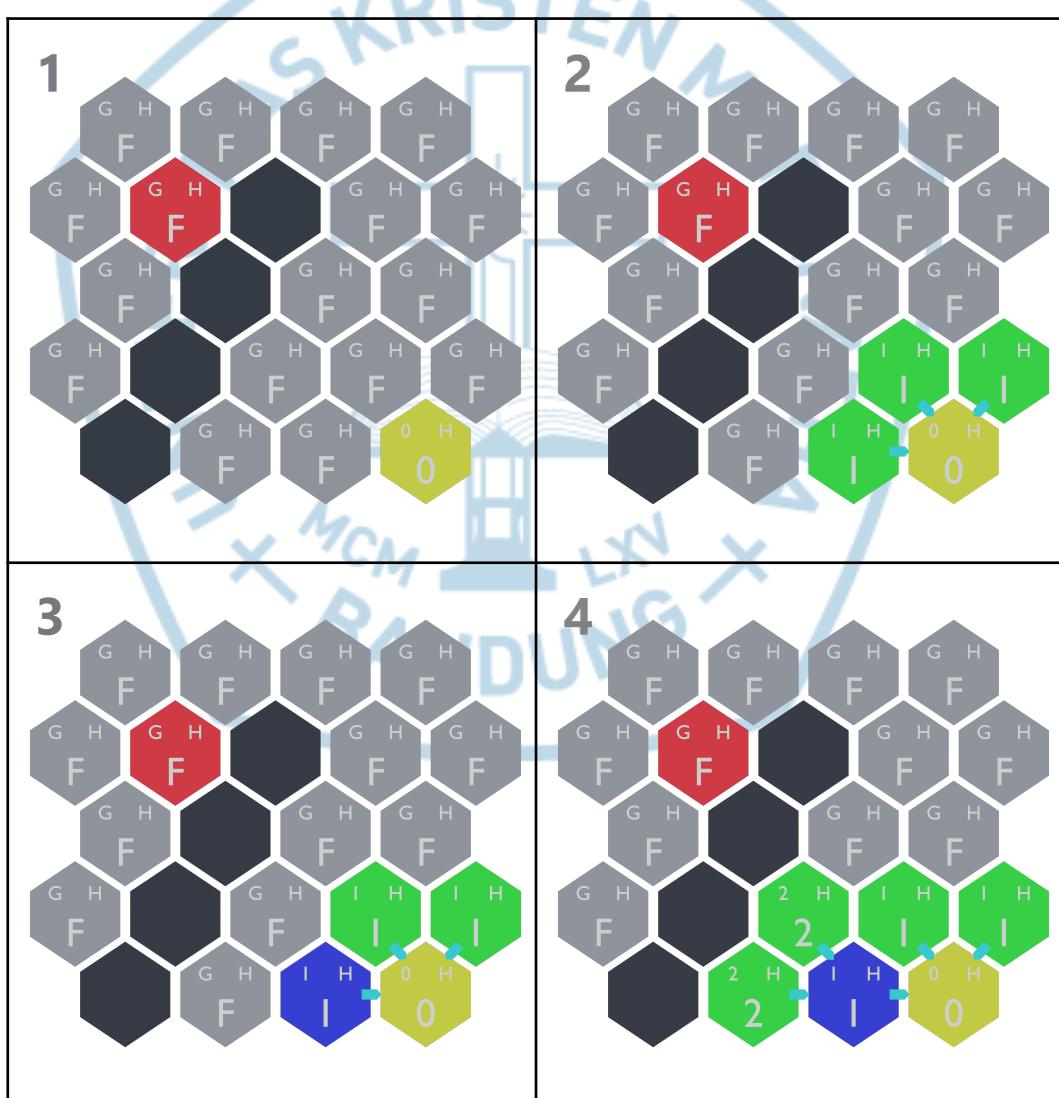
Berbeda dengan A-Bintang yang tidak memproses dua node hijau tetangga, Dijkstra tidak meninggalkan satupun node pada open list selama proses pencarian jalur, kecuali pada **Langkah 21** di mana **kondisi kedua** berlaku, yaitu node merah tujuan berada pada urutan pertama dalam open list.

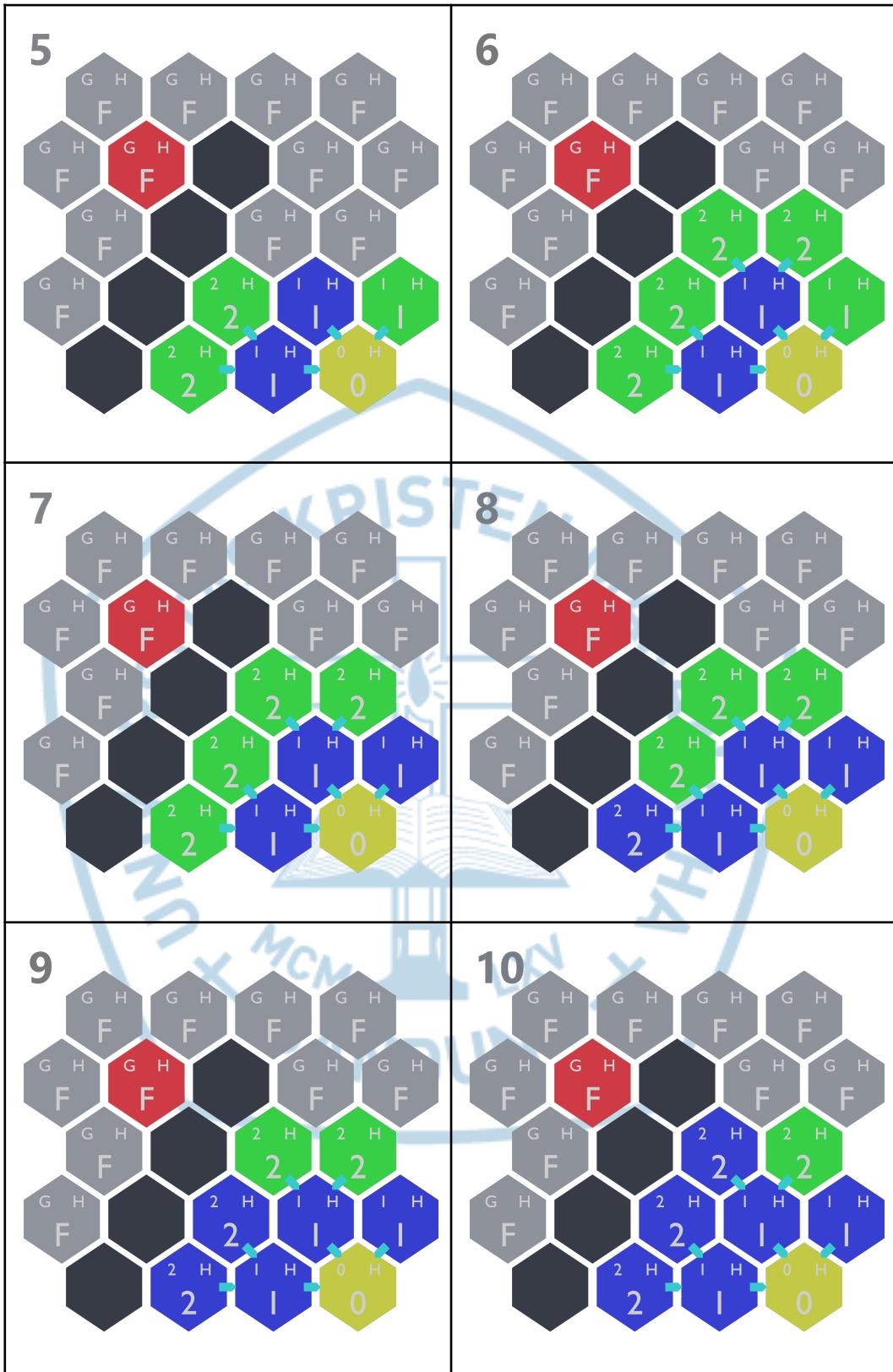
Selain itu, Dijkstra tidak menghitung h-value karena tujuan dari algoritma ini adalah mencari seluruh node hingga menemukan node merah tujuan. Oleh karena itu, h-value dari setiap node tidak diatur selama proses pencarian. Hal ini

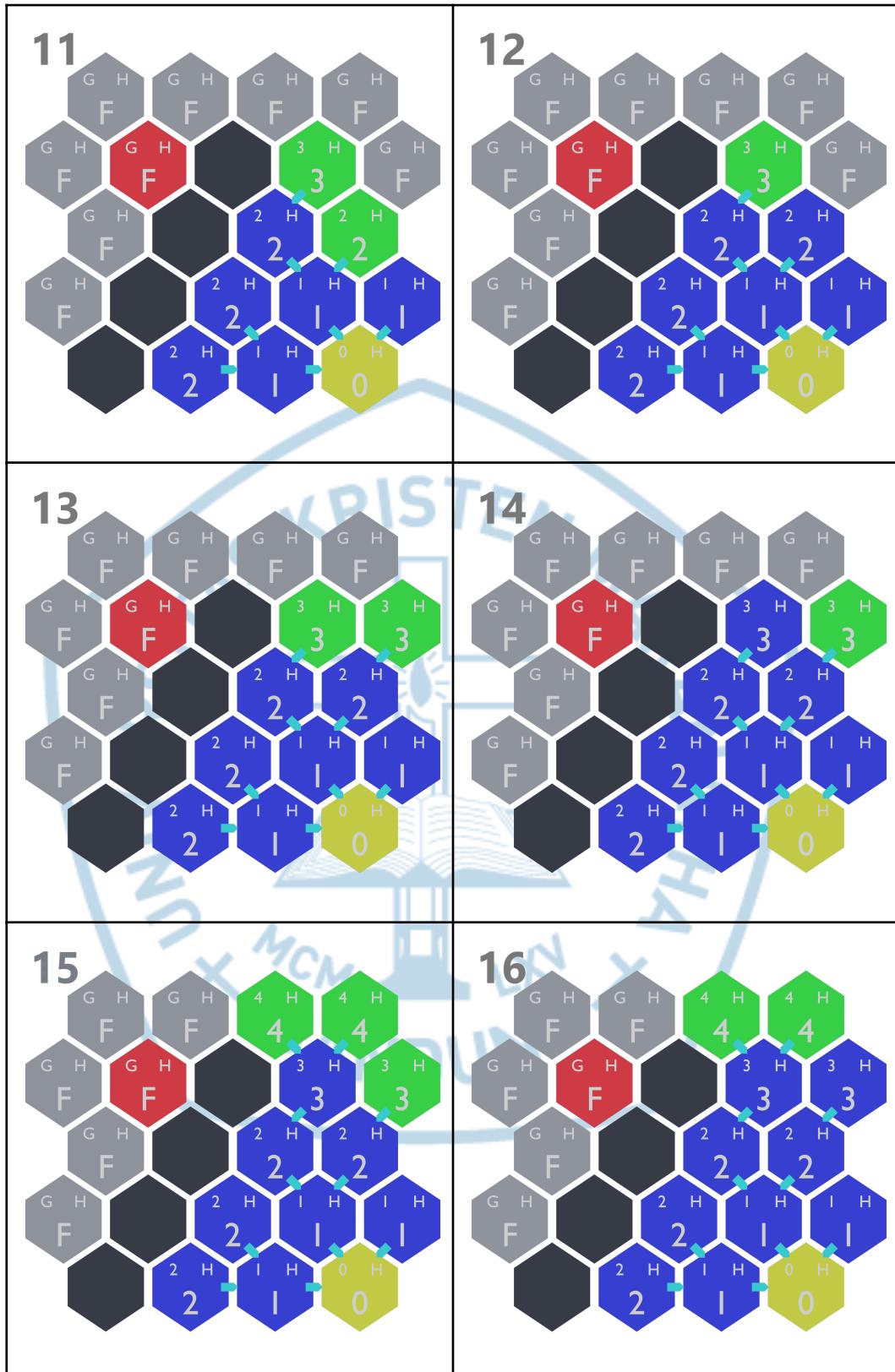
juganya menyebabkan f-value dari setiap node sama dengan g-value pada setiap langkah yang dilalui.

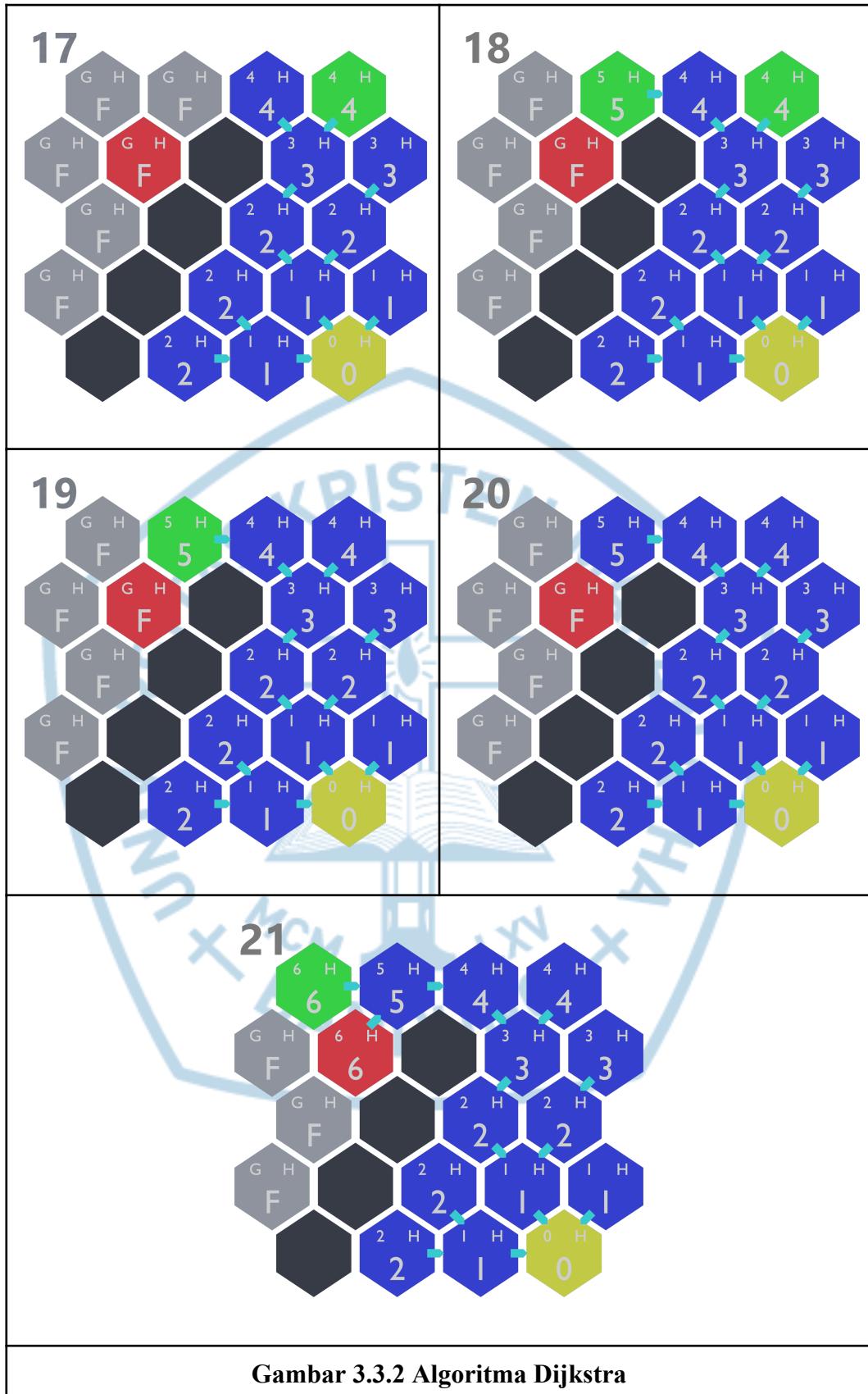
Pada **Gambar 3.2.2**, dapat dilihat bahwa **kondisi ketujuh** dalam pencarian jalur sangatlah terlihat keberadaannya. Yaitu node yang berada di urutan pertama dalam daftar node frontier list dipilih sebagai node biru terpilih untuk mencari langkah jalur selanjutnya.

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 jalurnya dituntun dengan menggunakan estimasi h-value, sedangkan Dijkstra hanya menggunakan g-value.





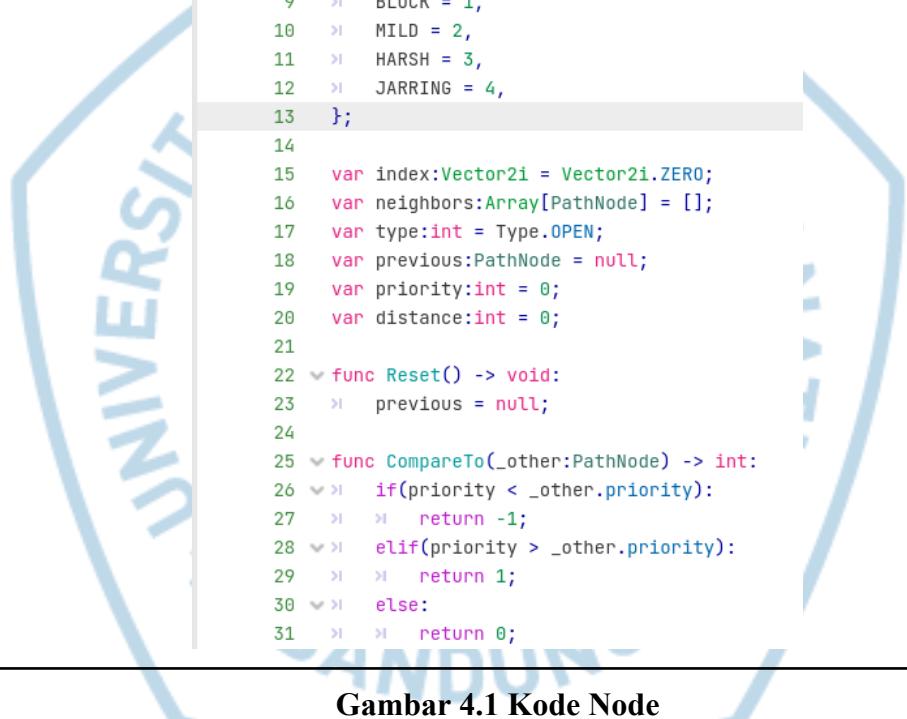




# BAB 4

## IMPLEMENTASI

### 4.1 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 4.1 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 4.1** 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.

Seperti yang terlihat pada **Gambar 4.1** 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 4.1** 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 4.1** 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 4.1** 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 kegunaanya. 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.

## 4.2 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 4.2 Kode Graph

**Graph** pada keseluruhan **Gambar 4.2** merupakan kelas yang nantinya akan menghubungkan node-node yang telah ditetapkan. Sehingga 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 4.2** 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 4.2** 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 4.2** 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 4.2** 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 4.2** 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.

### 4.3 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 4.2 Kode Algoritma

Berikut adalah kode algoritma yang akan digunakan, seperti yang terlihat pada **Gambar 4.2** 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 4.2** 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].

## BAB 5

### PENGUJIAN

#### 5.1 Perbandingan Kecepatan

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	null	null
38	1.72	1.72
39	1.72	1.72
40	null	null
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: 56		
<b>Tabel 5.1 Perbandingan Kecepatan</b>		

## 5.2 Perbandingan Biaya

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	null	null
38	50	52
39	59	59
40	null	null
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: 5		
A-Bintang: 0		
<b>Tabel 5.2 Perbandingan Biaya</b>		

### 5.3 Perbandingan Node Yang Dilalui

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	null	null
38	43	44
39	48	48
40	null	null
41	29	29
42	31	32

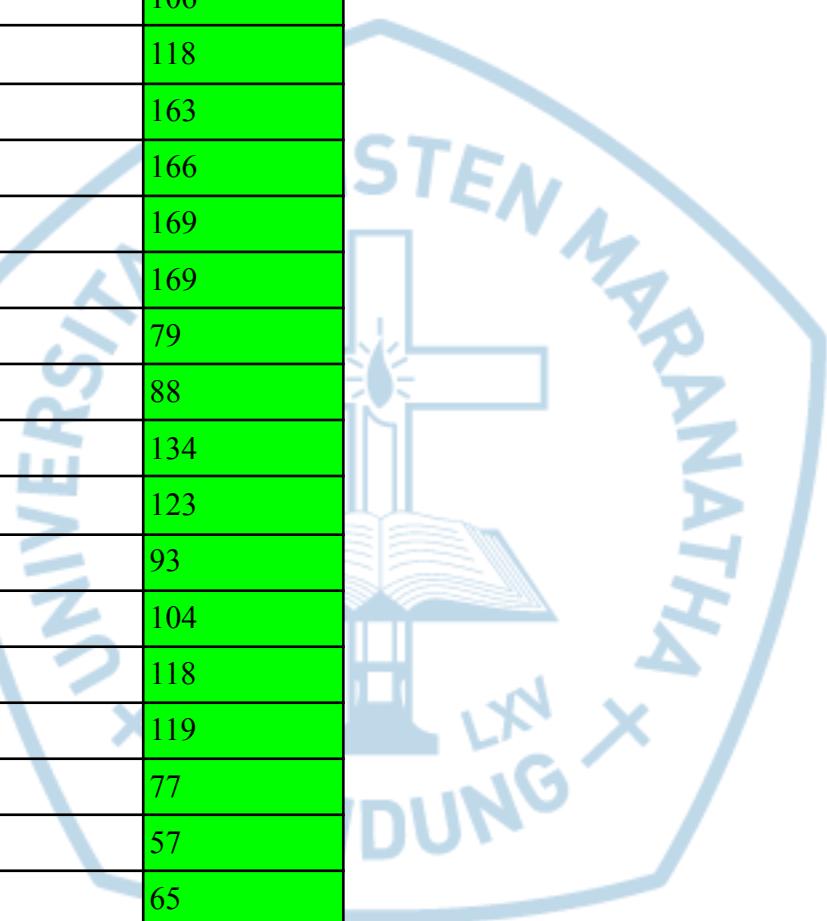
43	32	33
44	28	28
45	29	28
46	27	27
47	29	28
48	27	27
49	24	23
50	25	25
51	23	24
52	22	22
53	37	37
54	36	36
55	37	37
56	36	35
57	22	23
58	22	23
59	22	23
60	23	25
Dijkstra: 20		
A-Bintang: 6		
<b>Tabel 5.3 Perbandingan Node Yang Dilalui</b>		

#### 5.4 Perbandingan Node Yang Dikunjungi

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	49	49
38	104	104
39	104	104
40	49	49
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		
<b>Tabel 5.3 Perbandingan Node Yang Dikunjungi</b>		

## BAB 6

### SIMPULAN DAN SARAN

#### 6.1 Simpulan

Kedua algoritma memiliki hal

#### 6.2 Saran



## 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](http://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](http://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](http://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]



## LAMPIRAN A TABEL UJIAN

<b>Tabel Lampiran Data Terkumpul Dijkstra</b>					
<b>NO</b>	<b>Percobaan</b>	<b>Kecepatan</b>	<b>Biaya</b>	<b>Jumlah Node Dilalui</b>	<b>Jumlah Node Dikunjungi</b>
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
30	Peta 8 Percobaan 2	2.48		24	24
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 NAMA LAMPIRAN



## 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

