

# **Pemanfaatan Algoritma BFS dan DFS dalam Pencarian *Recipe* pada Permainan Little Alchemy 2**

Laporan Tugas Besar 2  
IF2211 Strategi Algoritma



**Disusun Oleh Kelompok 17 (Elemental) :**

12821046	Fardhan Indrayesa
13523051	Ferdinand Gabe Tua Sinaga
13523108	Henry Filberto Shenelo

**Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2025**

# DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>Bab 1</b>	
<b>Deskripsi Tugas.....</b>	<b>2</b>
<b>Bab 2</b>	
<b>Landasan Teori.....</b>	<b>4</b>
2.1. Traversal Graf.....	4
2.2. Breadth First Search (BFS).....	4
2.3. Depth First Search (DFS).....	5
2.4. Aplikasi Web.....	6
2.5. Kompleksitas Algoritma.....	6
<b>Bab 3</b>	
<b>Analisis Pemecahan Masalah.....</b>	<b>7</b>
3.1. Langkah Pemecahan Masalah.....	7
3.2. Proses Pemetaan Masalah.....	8
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web.....	8
3.4. Contoh Ilustrasi Kasus.....	9
<b>Bab 4</b>	
<b>Implementasi dan Pengujian.....</b>	<b>13</b>
4.1.1. DFS.....	13
4.1.2. BFS.....	16
4.1.3. Aplikasi Web.....	19
4.2.1. Satu elemen.....	20
4.2.2. Banyak Elemen.....	22
<b>Bab 5</b>	
<b>Kesimpulan dan Saran.....</b>	<b>25</b>
a. Kesimpulan.....	25
b. Saran.....	25
c. Refleksi.....	25
<b>PEMBAGIAN TUGAS.....</b>	<b>26</b>
<b>LAMPIRAN.....</b>	<b>27</b>
<b>DAFTAR PUSTAKA.....</b>	<b>28</b>

# Bab 1

## Deskripsi Tugas



Gambar 1.1 Little Alchemy 2  
(Sumber: <https://www.thegamer.com>)

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Recloak yang dirilis pada tahun 2017, permainan ini bertujuan untuk membuat 720 elemen dari 4 elemen dasar yang tersedia, yaitu *air*, *earth*, *fire*, dan *water*. Permainan ini merupakan sekuel dari permainan sebelumnya yakni Little Alchemy 1 yang dirilis tahun 2010.

Mekanisme dari permainan ini adalah pemain dapat menggabungkan kedua elemen dengan melakukan drag and drop, jika kombinasi kedua elemen valid, akan memunculkan elemen baru, jika kombinasi tidak valid maka tidak akan terjadi apa-apa. Permainan ini tersedia di web browser, Android atau iOS.

Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk menyelesaikan permainan Little Alchemy 2 ini dengan menggunakan **strategi Depth First Search dan Breadth First Search**.

Komponen-komponen dari permainan ini antara lain:

1. Elemen dasar

Dalam permainan Little Alchemy 2, terdapat 4 elemen dasar yang tersedia yaitu water, fire, earth, dan air, 4 elemen dasar tersebut nanti akan di-combine menjadi elemen turunan yang berjumlah 720 elemen.



Gambar 1.2 Elemen dasar pada Little Alchemy 2

2. Elemen turunan

Terdapat 720 elemen turunan yang dibagi menjadi beberapa *tier* tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki *recipe* yang terdiri atas elemen lainnya atau elemen itu sendiri.

3. *Combine Mechanism*

Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

## Bab 2

### Landasan Teori

#### 2.1. Traversal Graf

Traversal graf adalah aktivitas untuk melakukan pencarian solusi persoalan yang direpresentasikan dengan graf. Dalam traversal graf, terdapat algoritma yang dapat digunakan untuk mengunjungi simpul-simpul di dalam graf dengan cara yang sistematis, dengan asumsi graf terhubung. Terdapat dua jenis algoritma pencarian solusi berbasis graf, yaitu

- Algoritma pencarian tanpa informasi (*uninformed search/blind search*)

Pada jenis algoritma ini, tidak ada informasi tambahan yang disediakan. Contoh algoritma yang termasuk dalam jenis ini, yaitu *Depth First Search* (DFS), *Breadth First Search* (BFS), *Depth Limited Search*, *Iterative Deepening Search*, dan *Uniform Cost Search*.

- Algoritma pencarian dengan informasi (*informed search*)

Pada jenis algoritma ini, pencariannya berbasiskan heuristik, sehingga kita dapat mengetahui *non-goal state* yang mungkin mendekati solusi daripada *non-goal state* yang lain. Contoh algoritma ini, yaitu *Best First Search* dan algoritma  $A^*$ .

#### 2.2. *Breadth First Search* (BFS)

*Breadth First Search* adalah algoritma fundamental traversal grafik yang dilakukan dengan mengunjungi node pada level  $n$  terlebih dahulu sebelum mengunjungi node-node pada level  $n+1$ . Algoritma ini dapat digunakan untuk mendeteksi siklus dalam graf berarah maupun tak berarah, mencari jalur terpendek dalam sebuah graf tak berbobot, dan masalah lainnya.

Algoritma *Breadth First Search* (BFS) bekerja dengan cara memasukkan simpul awal ke dalam antrian. Selanjutnya, simpul pertama dalam antrian diambil untuk diperiksa. Jika simpul tersebut merupakan solusi, maka pencarian selesai dan hasil dikembalikan. Jika bukan, maka semua simpul tetangganya yang belum dikunjungi dimasukkan ke dalam antrian. Proses ini diulang hingga antrian kosong. Jika seluruh simpul telah diperiksa dan tidak ditemukan solusi, maka pencarian berakhir dengan hasil bahwa solusi tidak ditemukan.

Kelebihan algoritma BFS:

- Tidak akan mengalami jalan buntu, karena BFS mengeksplorasi semua kemungkinan jalur secara sistematis.
- Jika terdapat satu solusi, BFS pasti menemukannya.

- Jika terdapat lebih dari satu solusi, BFS akan menemukan solusi dengan tingkat kedalaman (level) paling minimum terlebih dahulu (solusi optimal dalam hal jumlah langkah).

Kekurangan algoritma BFS:

- Membutuhkan memori yang besar karena harus menyimpan semua simpul pada level tertentu sebelum melanjutkan ke level berikutnya.
- Proses pencarian bisa memakan waktu lama, terutama jika solusi berada di level yang sangat dalam, karena BFS harus memeriksa semua simpul pada level sebelumnya terlebih dahulu.

### 2.3. *Depth First Search (DFS)*

Algoritma *Depth First Search* (DFS) adalah algoritma pencarian yang menelusuri jalur secara mendalam dengan memulai dari simpul awal, lalu terus bergerak ke simpul anak paling kiri hingga mencapai simpul terdalam sebelum kembali menelusuri jalur lain.

Cara kerja algoritma DFS adalah dengan memasukkan simpul akar ke dalam sebuah tumpukan. Selanjutnya, ambil simpul paling atas dari tumpukan untuk diperiksa. Jika simpul tersebut merupakan solusi, maka pencarian selesai dan hasil dikembalikan. Jika bukan, maka semua simpul tetangganya yang belum dikunjungi dimasukkan ke dalam tumpukan. Proses ini terus diulang hingga semua simpul telah diperiksa. Jika tumpukan kosong dan solusi tidak ditemukan, maka pencarian berakhir dengan menyatakan bahwa tidak ada solusi.

Kelebihan algoritma DFS:

- DFS hanya menyimpan simpul-simpul pada jalur pencarian yang sedang aktif, sehingga penggunaan memorinya relatif kecil.
- Dalam beberapa kasus, DFS dapat secara tidak sengaja menemukan solusi lebih cepat tanpa harus menjelajahi seluruh ruang kemungkinan.

Kekurangan algoritma DFS:

- Jika struktur pohon pencarian memiliki kedalaman yang sangat besar atau bahkan tak terbatas, DFS tidak menjamin akan menemukan solusi (tidak bersifat *complete*).
- Ketika terdapat lebih dari satu solusi yang identik namun berada pada tingkat kedalaman berbeda, DFS tidak menjamin menemukan solusi terbaik atau terdangkal (tidak bersifat *optimal*).

## 2.4. Aplikasi Web

Aplikasi web adalah program yang dijalankan melalui internet dan dapat diakses langsung menggunakan browser tanpa perlu diunduh atau diinstal terlebih dahulu. Aplikasi ini bekerja dengan cara memproses permintaan pengguna melalui website, dan biasanya menggunakan API (Application Programming Interface) untuk menghubungkan bagian-bagian di dalamnya.

Secara umum, aplikasi web terdiri dari dua bagian utama, yaitu frontend dan backend. Frontend adalah bagian yang dilihat dan digunakan langsung oleh pengguna. Ia bertugas menampilkan informasi dan menerima input dari pengguna. Backend adalah bagian yang bertugas mengelola logika aplikasi, memproses data, dan berinteraksi dengan basis data atau layanan eksternal.

Agar frontend dan backend dapat berkomunikasi, digunakanlah API. API menerima permintaan dari frontend, kemudian backend memproses permintaan tersebut, dan hasilnya dikirim kembali ke frontend untuk ditampilkan kepada pengguna. Dengan cara ini, aplikasi web dapat berfungsi secara efisien dan responsif.

## 2.5. Kompleksitas Algoritma

Kompleksitas waktu algoritma DFS dalam notasi Big O adalah  $O(b^m)$  dengan  $m$  adalah maksimum kedalaman dari ruang pencarian. Keuntungannya adalah dalam kompleksitas ruang yaitu  $O(bm)$ .

Kompleksitas waktu algoritma BFS dalam notasi Big O adalah  $O(b^d)$  dengan  $d$  adalah kedalaman dari solusi terbaik. Kekurangannya adalah kompleksitas ruangnya juga  $O(b^d)$ . Akan tetapi, solusi yang ditemukan pada BFS dijamin merupakan solusi optimal (*recipe* terpendek).

## **Bab 3**

### **Analisis Pemecahan Masalah**

#### **3.1. Langkah Pemecahan Masalah**

Berikut merupakan algoritma dalam menemukan resep dari elemen target pada permainan Little Alchemy 2:

DFS:

1. Tentukan elemen target, sebagai *root*.
2. Cari resep-resep yang dapat membentuk elemen target tersebut.
3. Gunakan algoritma pencarian DFS untuk menjelajahi semua kemungkinan resep
4. Dimulai dari root (elemen target), akan dicari resep-resep yang dapat membentuk elemen tersebut.
5. Ambil salah satu resep dan cari resep dari masing-masing komponen tersebut yang dilakukan dan disimpan dalam struktur data stack.
6. Setiap stack menyimpan path dari target hingga titik tersebut.
7. Lakukan secara terus menerus hingga semua elemen menjadi elemen dasar (stack kosong). Artinya, mendapatkan 1 solusi.
8. Pencarian berhenti ketika semua kemungkinan sudah dijelajahi (semua stack kosong) atau jumlah solusi maksimum telah ditemukan (1 atau banyak).

BFS:

1. Tentukan elemen target, sebagai *root*.
2. Cari resep-resep yang dapat membentuk elemen target tersebut.
3. Gunakan algoritma pencarian BFS untuk menjelajahi semua kemungkinan resep
4. Dimulai dari root (elemen target), akan dicari resep-resep yang dapat membentuk elemen tersebut.
5. Eksplorasi setiap resep-resep tersebut tiap levelnya yang disimpan dalam struktur data queue.
6. Setiap queue menyimpan path dari target hingga titik tersebut.
7. Lakukan secara terus menerus hingga semua elemen menjadi elemen dasar (queue kosong). Artinya, mendapatkan 1 solusi.
8. Pencarian berhenti ketika semua kemungkinan sudah dijelajahi (semua queue kosong) atau jumlah solusi maksimum telah ditemukan (1 atau banyak).



### 3.2. Proses Pemetaan Masalah

Mapping elemen-elemen dari permasalahan Little Alchemy 2 berdasarkan algoritma graph traversal adalah sebagai berikut:

- a. Simpul: Setiap elemen dalam permainan
- b. Sisi: Elemen (simpul) *parent* dapat dibentuk oleh elemen (simpul) *child*-nya.
- c. Root: elemen target (yang ingin dicari resepnya)
- d. Leaf: elemen dasar (Fire, Earth, Water, Air)

Jumlah elemen: terdapat 720 elemen yang dapat ditemukan di dalam permainan, masing-masing dengan berbagai kemungkinan kombinasi. Jumlah kombinasi per elemen: Setiap elemen dapat digabungkan dengan beberapa elemen lainnya, menghasilkan tree yang besar dan kompleks, yang menyebabkan banyaknya jalur dan solusi yang mungkin.

### 3.3. Fitur Fungsional dan Arsitektur Aplikasi Web

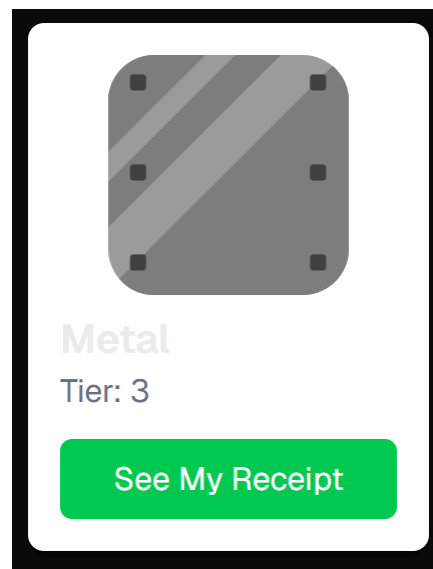
Fitur Fungsional dari Aplikasi Web adalah sebagai berikut:

1. Melakukan pencarian resep menggunakan algoritma BFS
2. Melakukan pencarian resep menggunakan algoritma DFS
3. Terdapat search bar untuk mencari elemen yang diinginkan atau kumpulan elemen yang berada pada tier yang sama.
4. Tampilan resep dalam bentuk *tree*.
5. Pemilihan banyak solusi yang diinginkan

Website ini dibangun dengan arsitektur yang memisahkan antara bagian tampilan (frontend) dan bagian logika pemrosesan (backend). Pada sisi frontend, digunakan Next.js, sebuah framework berbasis React yang mendukung *server-side rendering* dan *static generation*. Frontend ini berfungsi sebagai antarmuka pengguna untuk menangkap input, seperti pilihan algoritma, nilai *max solusi* dan resep yang ingin dicari, lalu mengirimkan permintaan ke backend melalui HTTP request. Di sisi lain, backend dikembangkan menggunakan bahasa Go (Golang) untuk menangani *concurrent processing*, sehingga mampu menjalankan algoritma seperti DFS atau BFS secara lebih efisien dibandingkan iterasi biasa yang berjalan secara linear. Backend menyediakan beberapa endpoint yang menerima input dari frontend, memproses data sesuai algoritma yang dipilih, dan menghasilkan file JSON berisi hasil pencarian berdasarkan parameter yang diberikan. File JSON ini kemudian dikirim kembali ke frontend dan divisualisasikan menggunakan D3.js dalam bentuk struktur tree yang interaktif dan mudah dipahami oleh pengguna.

### 3.4. Contoh Ilustrasi Kasus

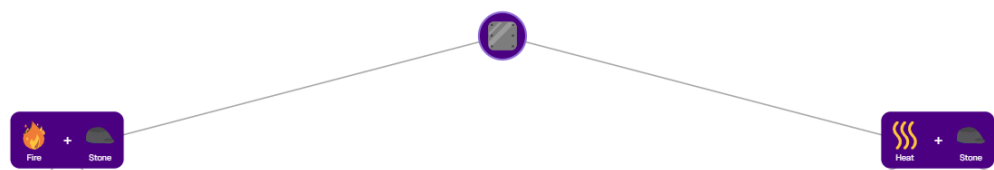
Untuk memperjelas analisis pemecahan masalah, akan diilustrasikan kasus beserta penyelesaiannya. Misalkan kasus yang diinginkan adalah mencari Metal.



Gambar 3.1.1 Elemen Target (Metal)

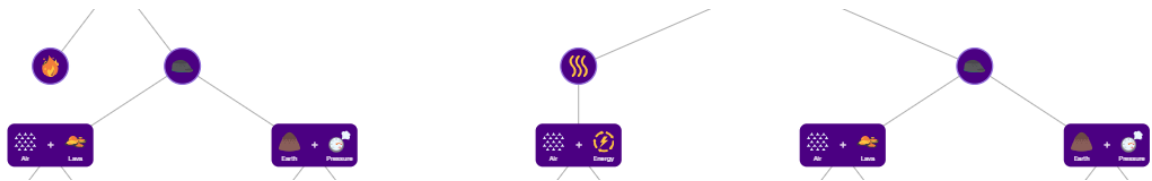
#### a. BFS

Ilustrasi proses pencarian resep menggunakan BFS dapat dibayangkan sebagai eksplorasi pohon pencarian resep, di mana akar (root) dari pohon adalah target yang ingin dicapai. Namun, pohon ruang status yang dibentuk hanya sekadar untuk keperluan visualisasi. Pada kenyataannya, struktur data yang digunakan dalam proses BFS adalah *queue*. Semua hasil kombinasi baru akan dimasukkan ke dalam antrian, dan BFS akan memproses kombinasi berdasarkan urutan antrian tersebut. Setiap elemen turunan dari elemen *parent*-nya akan dimasukkan ke dalam *queue*. Hasilnya adalah setiap elemen pada *depth* tertentu akan dieksplorasi terlebih dahulu sebelum menuju *depth* selanjutnya.



Gambar 3.1.2 BFS pada Depth 1

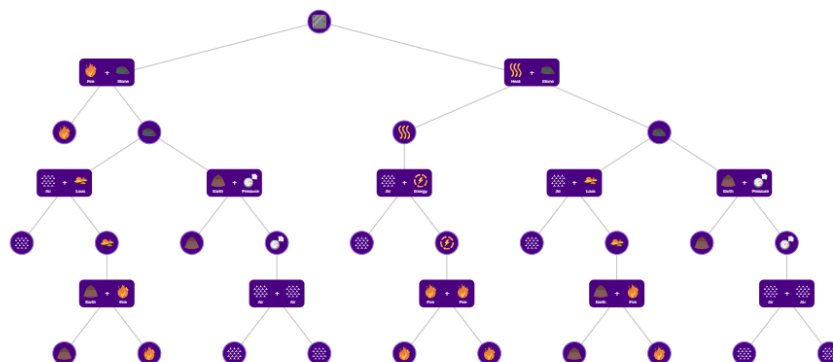
Pada gambar di atas, terlihat bahwa BFS dapat dibentuk oleh dua *recipe* yaitu (Fire + Stone) dan (Heat + Stone). Keduanya akan dieksplorasi dan dicari anak-anak dari tiap elemen pada kedalaman 1 tersebut.



Gambar 3.1.3 BFS pada Depth 2

Hal ini akan dilakukan secara terus menerus hingga elemen-elemen tersebut mencapai bentuk elemen dasar (Fire, Earth, Water, Air). Jika sampai pada titik tersebut, akan dihitung sebagai satu solusi. Pencarian akan berhenti apabila mencapai jumlah solusi/resep yang diinginkan sehingga ada *node* yang tidak dikunjungi.

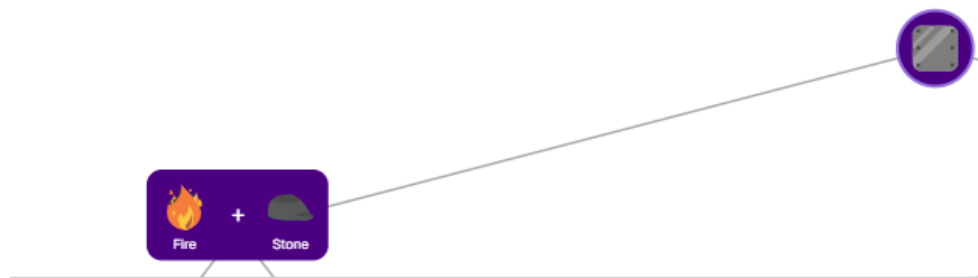
Pada akhirnya, untuk 4 solusi akan membentuk ruang pohon pencarian seperti:



Gambar 3.1.4 Ruang Pohon Pencarian BFS

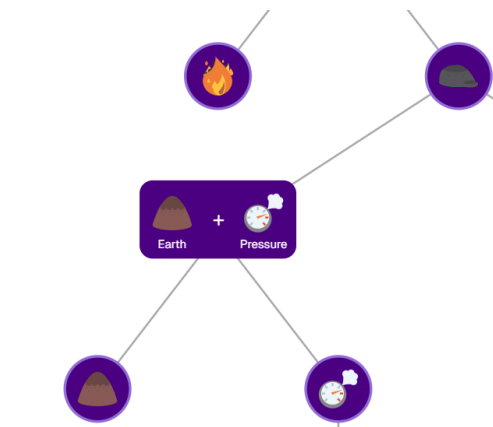
## b. DFS

Pada kasus DFS, sama seperti BFS, tidak digunakan struktur data *tree* secara langsung pada saat memproses tetapi menggunakan struktur *stack*. Semua hasil kombinasi baru akan dimasukkan ke dalam *stack*, dan DFS akan memproses kombinasi dari elemen paling terakhir yang masuk. Akibatnya, DFS akan mengeksplorasi satu resep terlebih dahulu secara mendalam hingga mencapai elemen dasar baru kemudian melakukan *backtracking* menuju resep yang lain. Setiap resep akan dieksplorasi hingga *depth* terdalam baru *backtracking*.



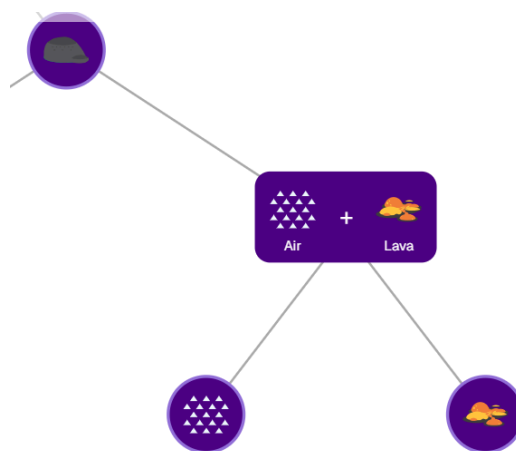
Gambar 3.1.5. Penelusuran Resep Pertama dengan DFS

Pada kasus DFS, yang pertama kali dieksplor hanyalah resep (Fire + Stone). Kemudian dilanjutkan dengan eksplorasi elemen Fire hingga dasar (sudah dasar sebenarnya) dan dilanjutkan Stone hingga mencapai elemen dasar.



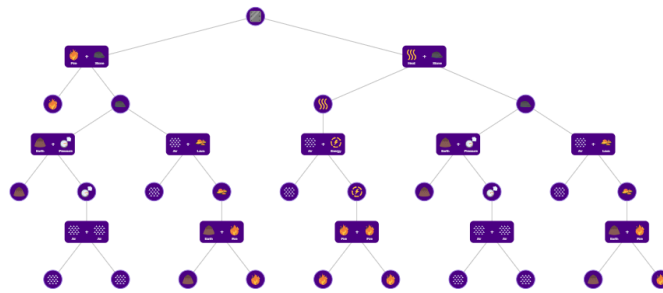
Gambar 3.1.6 Ilustrasi DFS

kemudian baru dilanjutkan dengan resep kedua dari *stone*



Gambar 3.1.7. Ilustrasi DFS (2)

Hasil ruang pencarian DFS adalah sebagai berikut



Gambar 3.1.8 Ruang Pencarian DFS (2)

Hasil yang didapatkan bisa jadi sama ataupun berbeda, tergantung pada urutan traversal dari DFS dan BFS serta jumlah solusi yang dicari, Akan tetapi, pada umumnya, BFS akan *visit* lebih banyak node dari DFS karena harus mengeksplor semua elemen pada setiap *depth*-nya.

## Bab 4

### Implementasi dan Pengujian

#### 4.1. Implementasi

##### 4.1.1. DFS

Struct method: WorkerPool, Stack

Struct: Step, Node, TreeNode

Fungsi: NewWorkerPool, PrintTree, UniqueElementAfterTime, Contains, ContainsElementInPath, copyStack, IsBasicElement, BuildTree, DFS, PrintStack, SaveResultToJSON

Struct	
Step	Struct untuk menyimpan langkah kombinasi.
Attribute	
Result (string)	Hasil kombinasi elemen
Components (slice of string)	Bahan yang dibutuhkan untuk membentuk elemen Result.

Struct	
Node	Node job yang akan dikirim ke worker
Attribute	
stack (Stack)	Stack yang berada dalam node saat ini

Struct	
TreeNode	Representasi elemen dalam bentuk pohon
Attribute	
Result (string)	Hasil kombinasi elemen-elemen yang berada di Components

Components (slice of TreeNode)	Bahan-bahan yang diperlukan untuk membentuk elemen Result.
--------------------------------	--

Struct	
WorkerPool	Kumpulan worker untuk menjalankan algoritma BFS secara paralel.
Attribute	
jobs (channel of Node)	Setiap worker mendapatkan Node untuk diproses secara BFS
wg (wait group)	Wait group untuk memastikan semua worker selesai bekerja.
results (channel of dfs.TreeNode)	Channel hasil pencarian
Count (int32)	Jumlah solusi yang ditemukan
maxSolutions (int)	Maksimal solusi yang diambil
done (channel of struct)	Sinyal untuk menghentikan semua worker
activeJobs (int32)	Worker yang aktif
jobsSubmitted (int32)	Total job yang sudah dikirim
mu (Mutex)	Untuk memeriksa apakah queue kosong untuk setiap job
Fungsi dan Prosedur	
Start	Untuk menjalankan semua worker dalam workerpool
worker	Fungsi worker yang mengambil dan memproses Node.
processNode	Proses satu Node BFS (mencari bahan/resep dari suatu elemen)

Submit	Mengirim Node ke antrean worker
Wait	Menunggu semua worker selesai
Stop	Menghentikan worker pool
GetResults (slice of dfs.TreeNode)	Mengambil seluruh hasil solusi dari channel.
isEmpty (bool)	Untuk mengecek, apakah antrean job kosong.

Struct	
Stack	Representasi status DFS saat ini dari elemen target
Attribute	
Elements (list of string)	Elemen yang harus diproses
Path (list of Step)	Urutan langkah pembuatan elemen
Visited (map from string to bool)	Elemen yang sudah ditelusuri di Path ini
Fungsi dan Prosedur	
push	Menambahkan elemen dan Path ke Stack

Fungsi dan Prosedur Lainnya	
DFS	Prosedur pencarian resep secara DFS
NewWorkerPool (struct of WorkerPool)	Membuat WorkerPool baru
PrintTree	Untuk membuat pohon yang akan di-import ke teks
UniqueElementsAfterTime (int)	Untuk menghitung elemen unik setelah



	Time muncul
SaveResultsToJSON (error)	Menyimpan hasil solusi dalam bentuk JSON
Contains (bool)	Memeriksa apakah suatu kumpulan elemen mengandung elemen tertentu.
ContainsElementInPath (bool)	Memeriksa, apakah mengandung elemen tertentu di path.
copyStack (Stack)	Membuat salinan baru dari Stack
IsBasicElement (bool)	Untuk memeriksa apakah suatu elemen merupakan elemen dasar
BuildTree (TreeNode)	Membuat pohon solusi dari path langkah
PrintStack	Mencetak isi Stack

#### 4.1.2. BFS

Struct method: WorkerPool, DepthTracker

Struct: Queue, Node

Fungsi: NewWorkerPool, copyQueue, NewDepthTracker, BFS, PrintQueue, SaveResultsToJSON

berikut merupakan penjelasan dari struktur data, fungsi, dan prosedur yang digunakan

Struct	
Queue	Struct untuk menyimpan data antrian elemen, path, dan elemen yang sudah dikunjungi.
Attribute	
Elements (list of string)	Elemen yang masih harus diproses
Path (list of dfs.Step)	Jalur langkah dari hasil ke komponen
Visited (map from string to int)	Kumpulan elemen yang sudah dikunjungi

Struct	
Node	Node job yang akan dikirim ke worker
Attribute	
queue (struct of Queue)	Antrian yang berada dalam Node saat ini
depth (int)	Jalur langkah dari hasil ke komponen

Struct	
DepthTracker	Struct untuk menjaga agar BFS tetap berjalan di kedalaman yang sama terlebih dahulu.
Attribute	
currentDepth (int)	Kedalaman saat ini
nodesAtDepth (map from string to int32)	Node pada kedalaman tertentu
processingDepth (map from string to int32)	Kedalaman yang sedang diproses
Lock	Pengunci untuk mengakses struct
Fungsi dan Prosedur	
GetCurrentDepth (int)	Atribut untuk mendapatkan kedalaman BFS saat ini.
AddNodeAtDepth	Menambah jumlah node pada kedalaman tertentu.
StartProcessingAtDepth	Menandai bahwa proses pada node sudah dimulai
FinishProcessingAtDepth	Menandai bahwa proses pada node sudah selesai.

Struct
--------

WorkerPool	Kumpulan worker untuk menjalankan algoritma BFS secara paralel.
<b>Attribute</b>	
jobs (channel of Node)	Setiap worker mendapatkan Node untuk diproses secara BFS
wg (wait group)	Wait group untuk memastikan semua worker selesai bekerja.
results (channel of dfs.TreeNode)	Channel hasil pencarian
Count (int32)	Jumlah solusi yang ditemukan
maxSolutions (int)	Maksimal solusi yang diambil
done (channel of struct)	Sinyal untuk menghentikan semua worker
activeJobs (int32)	Worker yang aktif
jobsSubmitted (int32)	Total job yang sudah dikirim
depthTracker (struct of DepthTracker)	Untuk melacak kedalaman BFS
mu (Mutex)	Untuk memeriksa apakah queue kosong untuk setiap job
<b>Fungsi dan Prosedur</b>	
Start	Untuk menjalankan semua worker dalam workerpool
worker	Fungsi worker yang mengambil dan memproses Node.
processNode	Proses satu Node BFS (mencari bahan/resep dari suatu elemen)
Submit	Mengirim Node ke antrean worker
Wait	Menunggu semua worker selesai
Stop	Menghentikan worker pool

GetResults (slice of dfs.TreeNode)	Mengambil seluruh hasil solusi dari channel.
isQueueEmpty (bool)	Untuk mengecek, apakah antrean job kosong.

Fungsi dan Prosedur Lainnya	
BFS	Prosedur pencarian resep secara BFS
NewWorkerPool (struct of WorkerPool)	Membuat WorkerPool baru
NewDepthTracker (struct of DepthTracker)	Membuat DepthTracker baru
PrintQueue	Mencetak isi antrian dan jalur pencarian
SaveResultsToJSON (error)	Menyimpan hasil solusi dalam bentuk JSON
copyQueue (struct of Queue)	Membuat salinan baru dari Queue

Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dsb.)

#### 4.1.3. Aplikasi Web

Sesuai dengan spesifikasi yang telah ditetapkan, kami menggunakan bahasa Go (Golang) pada bagian backend untuk menjalankan proses pencarian resep menggunakan algoritma DFS (Depth-First Search) atau BFS (Breadth-First Search). Data yang digunakan untuk membentuk peta (map) elemen-elemen dalam pencarian ini diperoleh melalui proses web scraping dari situs <https://little-alchemy.fandom.com/wiki>. Data tersebut dimanfaatkan untuk membangun struktur tree, dimulai dari elemen tujuan hingga ke elemen dasar (*basic elements*), yang kemudian diproses di sisi backend. Hasil pencarian dari backend disediakan melalui endpoint API yang berjalan pada port 8080, sehingga dapat diakses oleh frontend.

Setelah data hasil pencarian dikirim dalam format JSON ke frontend, data tersebut digunakan oleh aplikasi frontend yang berjalan pada port 3000 untuk menampilkan visualisasi tree berdasarkan struktur yang dikirim dari backend. Untuk

Di sisi frontend, kami menggunakan framework React karena sifatnya yang responsif, cepat, dan mudah diimplementasikan. Seluruh komponen antarmuka pengguna seperti *search bar*, *modal*, dan elemen visual lainnya kami simpan dalam folder bernama `_components`. Tujuannya adalah untuk memisahkan logika tampilan dari bagian luar dan menjaga struktur proyek tetap modular. Semua komponen tersebut digunakan oleh file `page.tsx`, yang berfungsi sebagai *main entry point* pada aplikasi web ini, serta dikelola state-nya secara optimal untuk menciptakan pengalaman pengguna (UI/UX) yang baik dan intuitif.

#### 4.2.1. Satu elemen

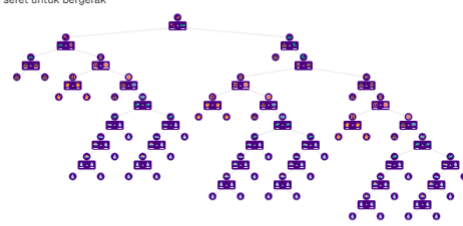
Informasi Program	Hasil									
<p>Tier 3: Chimney</p> <p>Waktu eksekusi: 25,3579 ms</p> <p>Jumlah node: 21</p>	<p>Gunakan pinch untuk zoom in/out, dan seret untuk bergerak</p> <table border="1"> <thead> <tr> <th colspan="3">Informasi Program</th> </tr> <tr> <th>Waktu Eksekusi:</th> <th>Jumlah Solusi:</th> <th>Jumlah Node:</th> </tr> </thead> <tbody> <tr> <td>25.3579ms</td> <td>1</td> <td>21</td> </tr> </tbody> </table>	Informasi Program			Waktu Eksekusi:	Jumlah Solusi:	Jumlah Node:	25.3579ms	1	21
Informasi Program										
Waktu Eksekusi:	Jumlah Solusi:	Jumlah Node:								
25.3579ms	1	21								
<p>Tier 8: Dynamite</p> <p>Waktu eksekusi: 25,6664</p> <p>Jumlah node: 194</p>	<p>Gunakan pinch untuk zoom in/out, dan seret untuk bergerak</p> <table border="1"> <thead> <tr> <th colspan="3">Informasi Program</th> </tr> <tr> <th>Waktu Eksekusi:</th> <th>Jumlah Solusi:</th> <th>Jumlah Node:</th> </tr> </thead> <tbody> <tr> <td>25.6664ms</td> <td>1</td> <td>194</td> </tr> </tbody> </table>	Informasi Program			Waktu Eksekusi:	Jumlah Solusi:	Jumlah Node:	25.6664ms	1	194
Informasi Program										
Waktu Eksekusi:	Jumlah Solusi:	Jumlah Node:								
25.6664ms	1	194								

Tier 10: Ant

Waktu eksekusi: 32,8527 ms

Jumlah node: 1631

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

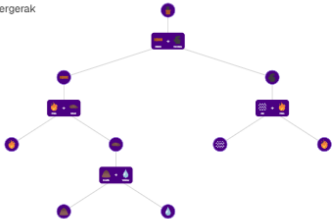
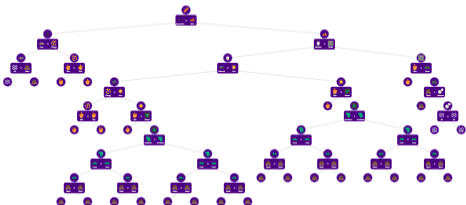
Jumlah Node:

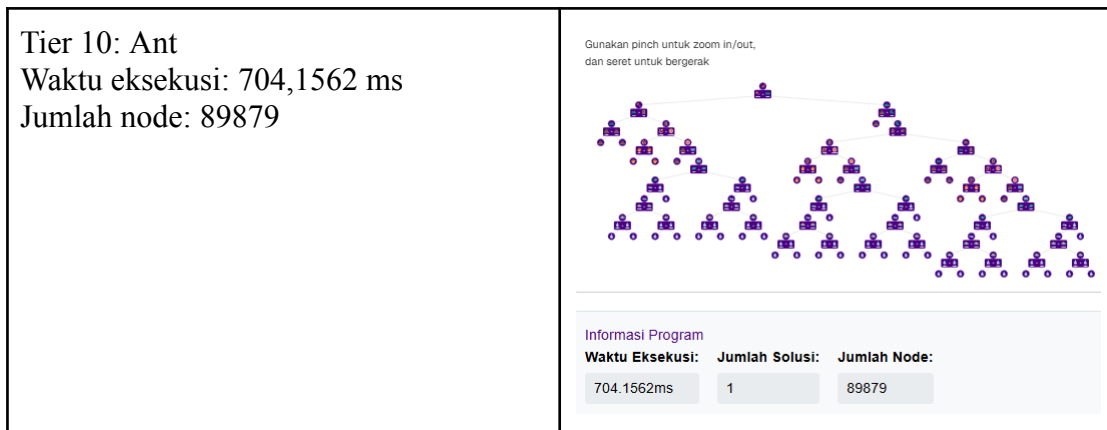
32.8527ms

1

1631

## BFS

Informasi Program	Hasil						
<p>Tier 3: Chimney</p> <p>Waktu eksekusi: 29,578 ms</p> <p>Jumlah node: 16</p>	<div><p>Gunakan pinch untuk zoom in/out, dan seret untuk bergerak</p></div> <div><p>Informasi Program</p><table><tr><td><b>Waktu Eksekusi:</b></td><td><b>Jumlah Solusi:</b></td><td><b>Jumlah Node:</b></td></tr><tr><td>29.578ms</td><td>1</td><td>16</td></tr></table></div>	<b>Waktu Eksekusi:</b>	<b>Jumlah Solusi:</b>	<b>Jumlah Node:</b>	29.578ms	1	16
<b>Waktu Eksekusi:</b>	<b>Jumlah Solusi:</b>	<b>Jumlah Node:</b>					
29.578ms	1	16					
<p>Tier 8: Dynamite</p> <p>Waktu eksekusi: 34,5653</p> <p>Jumlah node: 139</p>	<div><p>Gunakan pinch untuk zoom in/out, dan seret untuk bergerak</p></div> <div><p>Informasi Program</p><table><tr><td><b>Waktu Eksekusi:</b></td><td><b>Jumlah Solusi:</b></td><td><b>Jumlah Node:</b></td></tr><tr><td>34.5653ms</td><td>1</td><td>139</td></tr></table></div>	<b>Waktu Eksekusi:</b>	<b>Jumlah Solusi:</b>	<b>Jumlah Node:</b>	34.5653ms	1	139
<b>Waktu Eksekusi:</b>	<b>Jumlah Solusi:</b>	<b>Jumlah Node:</b>					
34.5653ms	1	139					



Berdasarkan hasil yang sudah diperoleh, terlihat bahwa pada kedua algoritma semakin tinggi tier, maka semakin lama waktu yang dibutuhkan untuk mencari resep elemen target dan semakin banyak node yang dikunjungi. Hal ini karena pada tier yang lebih tinggi, dibutuhkan banyak kombinasi dari tier-tier sebelumnya. Algoritma yang lebih unggul (cepat) dalam proses pencarian adalah algoritma DFS. Hal ini karena pada algoritma DFS, elemen tetangga yang ditinjau terlebih dahulu adalah elemen yang berada di level lebih dalam, sehingga cenderung langsung mengeksplorasi cabang kombinasi secara maksimal sebelum beralih ke cabang lainnya. Selain itu, batas kedalaman node pada permainan Little Alchemy 2 juga memiliki batas, yaitu hingga menemukan elemen dasar yang terdiri dari Earth, Air, Fire, dan Water.

Sedangkan pada algoritma BFS, pencarian dilakukan secara level-by-level atau menggunakan sistem antrian, yang menelusuri terlebih dahulu resep elemen tetangga di level kedalaman yang sama. Pada tier yang lebih tinggi, algoritma ini harus memeriksa lebih banyak node karena menelusuri terlebih dahulu semua resep di tier sebelumnya, sehingga lebih lambat daripada DFS. Kelemahan pada BFS ini adalah ketika sedang menelusuri resep suatu node (elemen), algoritma ini harus menelusuri semua node di level/kedalaman yang sama terlebih dahulu, sehingga tidak dapat menemukan elemen dasar secara langsung dalam suatu resep.

#### 4.2.2. Banyak Elemen

DFS

Informasi Program	Hasil
-------------------	-------

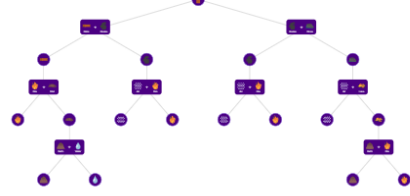
Tier 3: Chimney

Waktu eksekusi: 22,8061 ms

Jumlah node: 21

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

22.8061ms

3

21

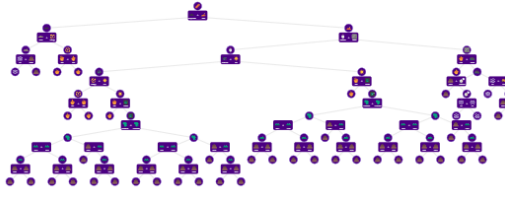
Tier 8: Dynamite

Waktu eksekusi: 23,8698 ms

Jumlah node: 167

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

23.8698ms

3

167

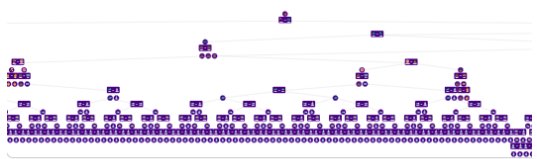
Tier 10: Ant

Waktu eksekusi: 26,7035 ms

Jumlah node: 1821

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

26.7035ms

3

1821

BFS

Informasi Program	Hasil
-------------------	-------



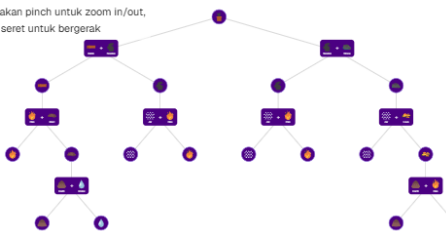
Tier 3: Chimney

Waktu eksekusi: 32,9632 ms

Jumlah node: 23

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

32.9632ms

3

23

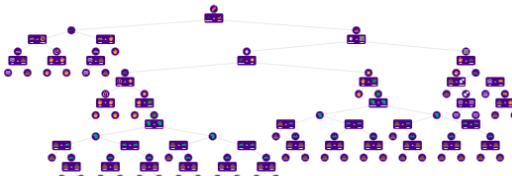
Tier 8: Dynamite

Waktu eksekusi: 38,6385 ms

Jumlah node: 148

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

38.6385ms

3

148

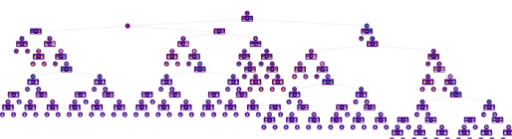
Tier 10: Ant

Waktu eksekusi: 1,1413 s

Jumlah node: 118585

Jumlah solusi: 3

Gunakan pinch untuk zoom in/out,  
dan seret untuk bergerak



Informasi Program

Waktu Eksekusi:

Jumlah Solusi:

Jumlah Node:

1.1412807s

3

118585

Tabel di atas adalah hasil yang diperoleh untuk algoritma DFS dan BFS dalam pencarian resep elemen target sebanyak 3. Berdasarkan hasil pada tabel di atas, terlihat bahwa algoritma DFS lebih unggul (cepat) dalam mencari banyak resep dibandingkan algoritma BFS. Selain itu, karena pada pencarian ini menggunakan heuristik tidak boleh ada tier elemen resep yang melebihi tier elemen targetnya, beberapa resep yang ditampilkan lebih sedikit dari jumlah solusi yang di-input oleh user. Pada pencarian banyak elemen menggunakan DFS, pencarian yang dilakukan secara mendalam akan dilewati apabila terdapat elemen dengan tier lebih tinggi.

## **Bab 5**

### **Kesimpulan dan Saran**

#### **a. Kesimpulan**

Pada tugas ini penulis telah berhasil membuat aplikasi web untuk menemukan resep dari suatu elemen pada permainan Little Alchemy 2. Algoritma yang digunakan untuk menemukan resep tersebut adalah algoritma *breadth first search* (BFS) dan *depth first search* (DFS). Berdasarkan kedua algoritma tersebut, dapat menghasilkan solusi dengan pendekatan yang berbeda. BFS akan menghasilkan *shortest path* tetapi dengan ruang pencarian yang besar, sedangkan DFS ruang pencariannya lebih kecil tetapi belum tentu optimal. Tugas ini diselesaikan dengan menggunakan bahasa pemrograman Golang dan frontend berbasis Nextjs.

#### **b. Saran**

Saran untuk kelompok ini adalah sebagai berikut.

- (1) Memulai pengerjaan tugas lebih awal agar hasil lebih maksimal
- (2) Lebih memperhatikan penggunaan *multithreading*
- (3) Komunikasi antar anggota lebih baik dan lebih bekerja sama

#### **c. Refleksi**

Refleksi yang kami peroleh dari tugas ini adalah pentingnya manajemen waktu dan komunikasi tim dalam menyelesaikan proyek berskala besar. Kami menyadari bahwa memulai pengerjaan lebih awal akan memberikan waktu yang cukup untuk mengevaluasi dan mengoptimasi hasil kerja, baik dari sisi frontend maupun backend.

Melalui tugas ini, kami mendapatkan pengalaman dalam mengembangkan aplikasi web yang menggabungkan frontend berbasis Next.js dan backend dengan bahasa Go. Kami juga belajar mengenai implementasi algoritma pencarian seperti BFS dan DFS serta memahami kelebihan serta keterbatasan masing-masing algoritma. Tugas ini juga memberikan wawasan baru dalam penerapan konsep pemrograman seperti multithreading, yang dapat meningkatkan performa sistem apabila digunakan dengan tepat.

## PEMBAGIAN TUGAS

NIM	Nama	Pembagian Tugas
12821046	Fardhan Indrayesa	Backend, Laporan
13523051	Ferdinand Gabe Tua Sinaga	Frontend, Scraping
13523108	Henry Filberto Shenelo	Backend, Scraping

## LAMPIRAN

Tautan *repository* GitHub : [https://github.com/henry204xx/Tubes2\\_Elemental](https://github.com/henry204xx/Tubes2_Elemental)

Tautan video : <https://youtu.be/TgtR0VTBphA>

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi dapat memperoleh data recipe melalui scraping.	✓	
3	Algoritma Depth First Search dan Breadth First Search dapat menemukan recipe elemen dengan benar.	✓	
4	Aplikasi dapat menampilkan visualisasi recipe elemen yang dicari sesuai dengan spesifikasi.	✓	
5	Aplikasi mengimplementasikan multithreading.	✓	
6	Membuat laporan sesuai dengan spesifikasi.	✓	
7	Membuat bonus video dan diunggah pada Youtube	✓	
8	Membuat bonus algoritma pencarian Bidirectional.		✓
9	Membuat bonus Live Update.		✓
10	Aplikasi di-containerize dengan Docker	✓	
11	Aplikasi di-deploy dan dapat diakses melalui internet.	✓	

## DAFTAR PUSTAKA

GeeksforGeeks. (n.d.). *Breadth First Search or BFS for a Graph*.  
<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

Tamara, D. (2023, July 30). *BFS (Breadth First Search): Pengertian, Kekurangan, Kelebihan, dan Contohnya*. Medium.  
<https://medium.com/@defytamara2610/bfs-breadth-first-search-pengertian-kekurangan-kelebihan-dan-contohnya-775a7d808fbc>

Tamara, D. (2023, July 30). *DFS (Depth First Search): Pengertian, Kekurangan, Kelebihan, dan Contohnya*. Medium.  
<https://medium.com/@defytamara2610/dfs-depth-first-search-pengertian-kekurangan-kelebihan-dan-contohnya-2b9b1eee2b3d>

Munir, R. (2024). *BFS dan DFS (Bagian 2)* [Lecture slides]. Program Studi Teknik Informatika, STEI ITB.  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf)