

# **LAPORAN TUGAS BESAR 2**

## **IF2211 STRATEGI ALGORITMA**

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



**Disusun Oleh:**

**“ian”**

Aloisius Adrian Stevan Gunawan                    13523054

Grace Evelyn Simon                                  13523087

Karol Yangqian Poetracahya                        13523093

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

**2025**

# **DAFTAR ISI**

<b>BAB 1</b>	<b>3</b>
<b>DESKRIPSI TUGAS</b>	<b>3</b>
<b>BAB 2</b>	<b>6</b>
<b>LANDASAN TEORI</b>	<b>6</b>
2.1 Traversal Graf	6
2.2 Algoritma BFS (Breadth-First Search)	8
2.3 Algoritma DFS (Depth-First Search)	9
2.4 Frontend	10
2.4 Backend	10
2.5 Docker	12
<b>BAB 3</b>	<b>13</b>
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>13</b>
3.1 Langkah-langkah Pemecahan Masalah	13
3.1.1 Akuisisi dan Representasi Data Elemen dan Resep	13
3.2 Proses Pemecahan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS	14
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun	16
3.4 Contoh Ilustrasi Kasus	18
<b>BAB 4</b>	<b>21</b>
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>21</b>
4.1 Struktur Data	21
4.2 Fungsi dan Prosedur Program	23
4.3 Tata Cara Penggunaan Program	68
4.4 Hasil Pengujian	72
4.5 Analisis Hasil Pengujian	77
<b>BAB 5</b>	<b>78</b>
<b>KESIMPULAN, SARAN, DAN REFLEKSI</b>	<b>78</b>
5.1 Kesimpulan	78
5.2 Saran	79
5.3 Refleksi	79
<b>LAMPIRAN</b>	<b>81</b>
Tautan Repository Github	81
Tautan Video	81
Hasil Akhir Tugas Besar	81
<b>LAMPIRAN</b>	<b>82</b>

## **DAFTAR GAMBAR**

Gambar 1. Elemen Dasar pada Little Alchemy 2	4
Gambar 2. Contoh Visualisasi Recipe Elemen	5
Gambar 3. Contoh Penggunaan Algoritma BFS/DFS	7
Gambar 4. Ilustrasi Alur Kerja Docker	12
Gambar 5. Penerimaan Request User dari Frontend	18
Gambar 6. Visualisasi Pohon Resep oleh Frontend	20
Gambar 7. Interface Frontend	69
Gambar 8. Interface Frontend	70
Gambar 9. Interface Frontend	70
Gambar 10. Interface Frontend	71
Gambar 11. Interface Frontend	71
Gambar 12. Interface Frontend	71
Gambar 13. Hasil Pengujian 1	72
Gambar 14. Hasil Pengujian 2	73
Gambar 15. Hasil Pengujian 3	73
Gambar 16. Hasil Pengujian 4	74
Gambar 17. Hasil Pengujian 5	74
Gambar 18. Hasil Pengujian 6	75
Gambar 19. Hasil Pengujian 7	75
Gambar 20. Hasil Pengujian 8	76
Gambar 21. Hasil Pengujian 9	76

# **BAB 1**

## **DESKRIPSI TUGAS**

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* (DFS) dan *Breadth First Search* (BFS). 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. Elemen Dasar pada Little Alchemy 2

## 2. Elemen turunan

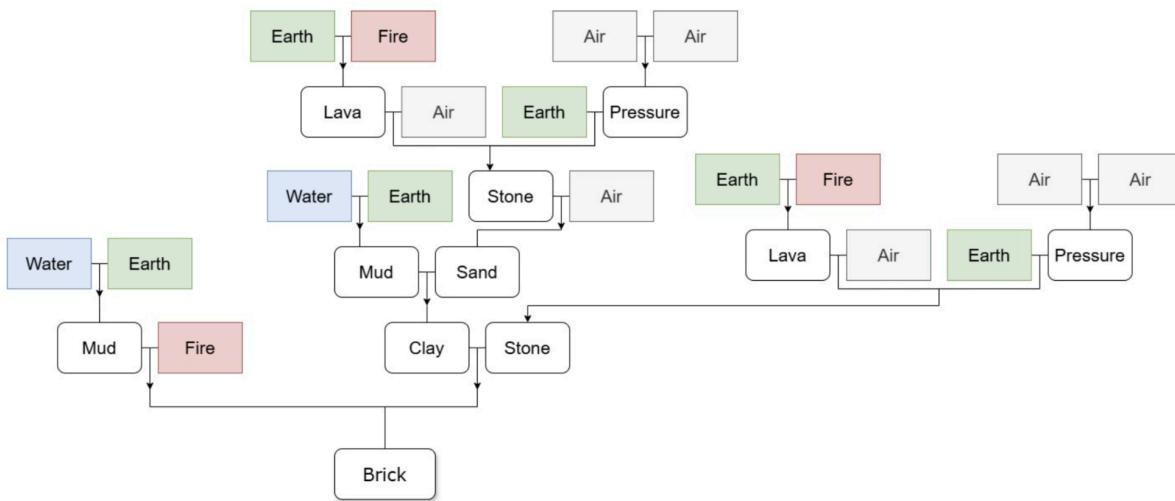
Terdapat 720 elemen turunan yang dibagi menjadi beberapa *tier* tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki *recipe* yang terdiri atas elemen lainnya atau elemen itu sendiri.

## 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.

Tugas ini mengharuskan pencarian resep elemen dalam permainan Little Alchemy 2 menggunakan strategi pencarian BFS dan DFS. Aplikasi dibangun berbasis *web* dengan *frontend* menggunakan JavaScript, menggunakan framework Next.js atau React.js, sementara *backend* menggunakan bahasa Golang. Tugas ini dilakukan secara berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, dan diperbolehkan lintas kelas maupun lintas kampus. Data elemen beserta resepnya dapat diperoleh melalui *scraping* dari website *Fandom Little Alchemy 2*. Aplikasi ini memungkinkan pengguna memilih algoritma BFS atau DFS (dengan opsi

*bidirectional* jika membuat bonus) dan juga menyediakan *toggle button* untuk memilih mode pencarian satu resep atau pencarian banyak resep menuju elemen tertentu. Pada mode pencarian banyak resep, pengguna dapat memasukkan parameter maksimal jumlah resep yang ingin dicari. Aplikasi harus mengoptimasi pencarian resep banyak dengan *multithreading*. Visualisasi hasil pencarian resep ditampilkan dalam bentuk *tree* yang menunjukkan kombinasi elemen yang diperlukan dari elemen dasar, dengan *leaf* dari *tree* selalu berupa elemen dasar. Aplikasi juga harus menampilkan waktu pencarian dan banyak *node* yang dikunjungi selama proses pencarian.



Gambar 2. Contoh Visualisasi *Recipe* Elemen

## BAB 2

# LANDASAN TEORI

### 2.1 Traversal Graf

Penjelajahan graf (atau *graph traversal*) adalah proses sistematis untuk mengunjungi semua simpul atau vertex dalam sebuah graf  $G = (V, E)$ , dengan  $V$  adalah himpunan simpul (vertex) dan  $E$  adalah himpunan sisi (edges) yang menghubungkan pasangan simpul. Tujuan utama dari penjelajahan graf adalah untuk “mengunjungi” setiap simpul tepat satu kali dan menjelajahi sisi-sisi yang terhubung dengannya untuk memahami struktur dan properti graf. Penjelajahan graf merupakan pondasi penting dalam ilmu komputer dan sering digunakan sebagai bagian dari algoritma yang lebih kompleks untuk menyelesaikan berbagai masalah. Beberapa aplikasi umum dari algoritma traversal graf:

1. Pencarian Jalur: menemukan apakah ada jalur antara dua simpul, atau menemukan jalur terpendek (seperti dalam GPS atau *routing* jaringan).
2. Penemuan Komponen Terhubung (*Connected Components*): mengidentifikasi kelompok-kelompok simpul dimana setiap simpul dalam satu kelompok dapat mencapai simpul lain dalam kelompok yang sama.
3. Analisis Jaringan: memahami koneksi dan ketergantungan dalam jaringan sosial, jaringan komputer, atau sistem biologis.
4. Deteksi Siklus (*Cycle Detection*): menemukan apakah ada jalur yang dimulai dan berakhir pada simpul yang sama.
5. *Topological Sorting*: Mengurutkan simpul dalam graf berarah asiklik (DAG) sedemikian rupa sehingga untuk setiap sisi berarah dari simpul  $u$  ke simpul  $v$ , simpul  $u$  muncul sebelum simpul  $v$  dalam urutan, dan sebagainya.

Ada dua metode utama dan paling fundamental dalam melakukan penjelajahan graf:

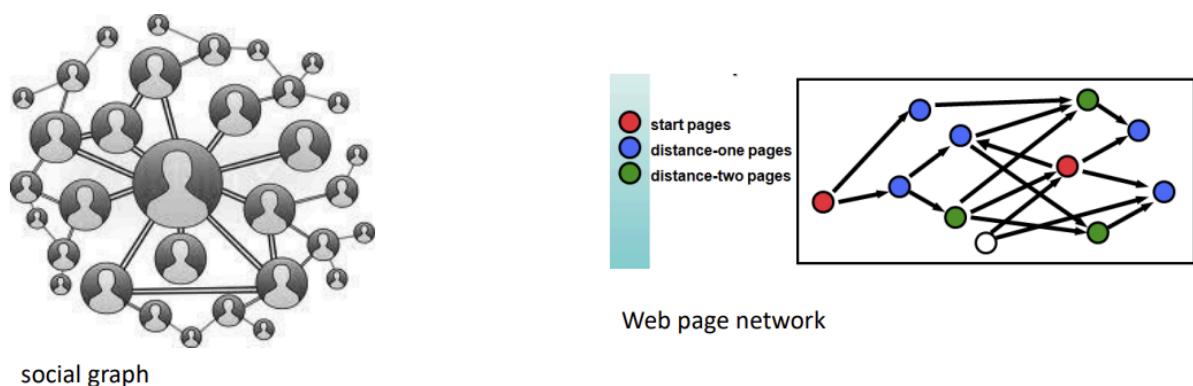
1. Pencarian melebar (*Breadth-First Search/BFS*)

BFS merupakan algoritma untuk menjelajahi graf dengan cara mengunjungi semua simpul pada tingkat kedalaman yang sama sebelum melanjutkan ke tingkat kedalaman berikutnya. BFS menggunakan *queue* dengan metode FIFO untuk melacak simpul yang dikunjungi berikutnya. Tujuan utama BFS pada umumnya adalah untuk mencari jalur terpendek dalam graf tidak berbobot.

2. Pencarian mendalam (*Depth-First Search/DFS*)

DFS merupakan algoritma untuk menjelajahi graf dengan bergerak sedalam mungkin sebelum kembali untuk menjelajahi jalur lainnya. Berbeda dengan BFS, DFS menggunakan *stack* dengan metode LIFO untuk melakukan rekursi dalam melacak perjalanan melalui graf. Tujuan utama DFS pada umumnya adalah menemukan komponen terhubung, mendeteksi siklus, dan melakukan *topological sorting*.

Ada dua pendekatan dalam proses pencarian solusi representasi graf, yakni graf statis dan graf dinamis. Graf statis merupakan graf yang sudah terbentuk sebelum proses pencarian dilakukan, sedangkan graf dinamis merupakan graf yang terbentuk saat proses pencarian dilakukan. Pada tugas ini, digunakan pendekatan graf statis berupa algoritma BFS dan DFS.



Gambar 3. Contoh Penggunaan Algoritma BFS/DFS

## 2.2 Algoritma BFS (*Breadth-First Search*)

*Breadth-First Search* (BFS) adalah sebuah algoritma yang digunakan untuk menjelajahi atau mencari elemen dalam sebuah struktur data graf atau pohon. BFS merupakan teknik eksplorasi menggunakan pendekatan melebar untuk menjelajahi semua simpul pada tingkat tertentu sebelum dilanjutkan ke tingkat berikutnya. Berikut adalah langkah-langkah untuk melakukan algoritma BFS:

### 1. Inisialisasi

- Pilih sebuah simpul awal.
- Masukkan simpul awal tersebut ke dalam antrian.
- Tandai simpul awal sebagai “sudah dikunjungi” untuk menghindari pemrosesan berulang.

### 2. Proses pencarian

- Keluarkan simpul dari depan antrian.
- Proses simpul tersebut (contohnya dengan memeriksa apakah simpul tersebut adalah simpul tujuan).

### 3. Pemeriksaan Simpul Tetangga:

Untuk setiap simpul tetangga dari simpul yang baru saja dikeluarkan, jika simpul tetangga belum pernah dikunjungi:

- Tandai simpul tersebut sebagai “sudah dikunjungi”.
- Masukkan simpul tersebut ke dalam antrian.

Pengulangan:

- Ulangi langkah 2 dan 3 sampai antrian menjadi kosong (artinya semua simpul yang terjangkau telah dikunjungi) atau sampai simpul tujuan (solusi) ditemukan.

## 2.3 Algoritma DFS (*Depth-First Search*)

*Depth-First Search* (BFS) adalah sebuah algoritma yang digunakan untuk menjelajahi atau mencari elemen dalam sebuah struktur data graf atau pohon. DFS merupakan teknik eksplorasi menggunakan pendekatan mendalam untuk menjelajahi sejauh mungkin sepanjang setiap cabang sebelum melakukan *backtracking*. Berikut adalah langkah-langkah untuk melakukan algoritma DFS:

### 1. Inisialisasi

- Pilih sebuah simpul awal.
- Tandai simpul tersebut sebagai “sudah dikunjungi”.
- Proses simpul tersebut (misalnya, periksa apakah simpul tersebut adalah solusi).

### 2. Eksplorasi Tetangga (Rekursif atau Iteratif dengan *Stack*)

Untuk setiap simpul tetangga dari simpul yang saat ini sedang diproses, jika simpul tetangga belum pernah dikunjungi, panggil (secara rekursif) DFS mulai dari simpul tetangga.

### 3. *Backtracking*

Ketika sebuah simpul telah dikunjungi dan semua tetangganya juga telah dijelajahi (atau dari simpul tersebut tidak ada lagi tetangga yang belum dikunjungi yang bisa dijangkau), maka proses DFS akan *backtrack* ke simpul dari mana simpul pertama kali dikunjungi. Algoritma kemudian akan mencoba menjelajahi cabang lain dari simpul sebelumnya tersebut.

### 4. Penghentian

Pencarian berakhir jika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul-simpul yang telah dikunjungi (semua komponen yang terhubung dengan simpul awal telah dijelajahi) atau ketika solusi telah ditemukan dan diputuskan untuk berhenti.

## **2.4 Frontend**

Sisi *frontend* dari aplikasi pencarian *recipe* Little Alchemy 2 dibangun secara terpisah dari repositori *backend* menggunakan *framework* Next.js dalam bahasa JavaScript. Next.js adalah *open-source* React *framework* yang dikembangkan oleh Vercel untuk pembuatan aplikasi web secara *full-stack*. *Framework* ini menyederhanakan proses *deployment* karena hosting dapat dilakukan secara gratis dan mudah melalui situs Vercel. *Dependencies* yang dimiliki oleh program diatur menggunakan *package manager* NPM. Adapun *libraries* yang digunakan sebagai *dependencies* untuk mempermudah proses penulisan program adalah Tailwind CSS untuk *styling* yang diimplementasi langsung pada HTML, React untuk mengelola *states* pada aplikasi, Dotenv untuk mengelola *environment variables*, dan Vis.js untuk menampilkan *recipe* sebagai sebuah tampilan grafis *network* atau jaringan yang merepresentasikan struktur pohon dari resep yang ditemukan.

## **2.4 Backend**

*Backend* untuk aplikasi pencarian *recipe* Little Alchemy 2 ini dibangun menggunakan bahasa pemrograman Go (Golang). Pemilihan Go didasarkan pada performanya yang efisien, kemampuannya dalam menangani konkurensi, dan ekosistem *library* standar yang kuat untuk membangun aplikasi *web*. Arsitektur *backend* secara umum mengikuti pola *client-server*, di mana *backend* bertugas untuk menyediakan data elemen dan resep, memproses permintaan pencarian resep dari *frontend*, menjalankan algoritma pencarian (BFS/DFS) untuk menemukan resep, serta mengirimkan kembali hasil pencarian resep ke *frontend* dalam format yang ditentukan. Berikut adalah komponen-komponen utama *backend*:

1. Pengambilan dan Pemrosesan Data Awal ([scraper.go](#))

Sumber data untuk *scraping*, informasi elemen dan resep diambil dari halaman Fandom Wiki Little Alchemy 2 ([https://little-alchemy.fandom.com/wiki/Elements\\_\(Little\\_Alchemy\\_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2))). *Library* *scraping* yang digunakan dalam proses *scraping* dilakukan dengan bantuan library Go [github.com/PuerkitoBio/goquery](https://github.com/PuerkitoBio/goquery). Hasil scraping mentah disimpan sementara. Kemudian, data ini diproses untuk membangun representasi graf dari semua resep. Graf ini disimpan

dalam memori menggunakan `map[string]*model.RecipeTreeNodeTier`, dengan kunci peta adalah nama elemen, dan nilainya adalah pointer ke struct `model.RecipeTreeNodeTier`. Selama proses scraping, urutan pertama kali sebuah nama elemen hasil ditemukan di halaman wiki juga dicatat. Urutan ini digunakan untuk memberikan ID numerik yang konsisten pada data yang akan disajikan. *Scraper* juga menerapkan beberapa *filter* untuk mengabaikan entri yang bukan merupakan elemen game standar (misalnya, link ke pack “Myths and Monsters” atau kategori).

## 2. Transformasi Data dan Perhitungan Tier ([main.go](#))

Setelah data graf `RecipeTreeNodeTier` terbentuk, dilakukan dua proses penting saat aplikasi *backend* dimulai, yakni perhitungan tier menggunakan algoritma *Breadth-First Search* (BFS). Elemen dasar (Air, Earth, Fire, Water) diberi tier 0. Elemen yang dibuat dari dua elemen tier 0 akan menjadi tier 1, dan seterusnya. Hasilnya disimpan dalam `map[string]int (globalElementTiers)`. Kemudian, dilakukan transformasi ke format slice `[]model.Element`. Struktur ini dirancang untuk digunakan oleh implementasi algoritma pencarian resep.

## 3. Server HTTP dan Routing ([main.go](#))

*Backend* menggunakan library [github.com/gorilla/mux](https://github.com/gorilla/mux) sebagai router HTTP. Mux mempermudah pendefinisian rute API, termasuk menangani parameter pada path URL. Terdapat beberapa *endpoint* yang disediakan untuk mempermudah proses *debugging*, seperti `/graph-data` (GET) yang menyajikan seluruh data elemen yang sudah diproses dalam format `[]model.Element` sebagai JSON. Outputnya diurutkan berdasarkan ID (yang merefleksikan urutan penemuan *scraper*) dan sudah difilter untuk hanya menyertakan elemen dasar atau elemen yang memiliki resep. `/api/recipes/{elementName}` (GET) adalah *endpoint* API utama untuk pencarian resep spesifik yang menerima `elementName` sebagai path parameter dan menerima query parameters: *algorithm* (bfs/dfs), *mode* (shortest/multiple), dan *count* kemudian mengembalikan hasil pencarian dalam format JSON yang berisi pohon resep, waktu pencarian, dan jumlah *node* yang dikunjungi.

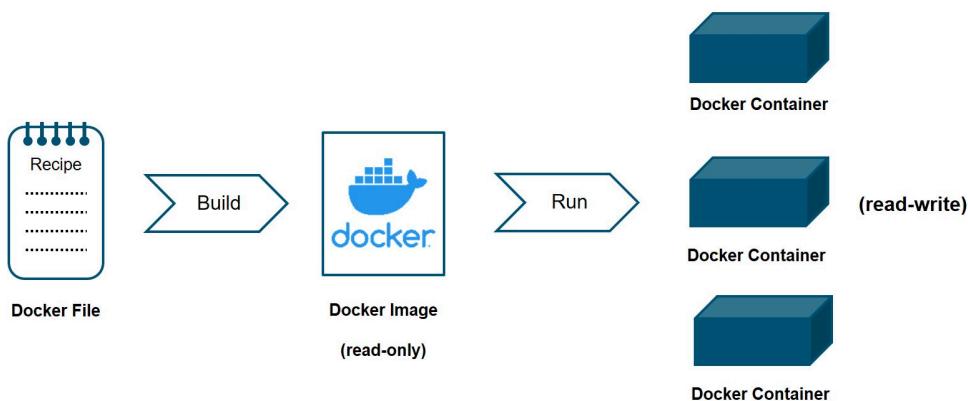
## 4. Handler API untuk Pencarian Resep ([handlers.go](#))

Bertanggung jawab untuk menerima permintaan HTTP. Fungsi ini mengekstrak `elementName` dari path URL dan parameter `algorithm`, `mode`, `count` dari `query string` dan melakukan validasi serta normalisasi input.

2.5 Docker

Dalam pengembangan aplikasi modern, Docker memainkan peran krusial baik untuk sisi *backend* maupun *frontend*. Untuk *backend*, Docker memungkinkan pengemasan aplikasi beserta seluruh dependensinya ke dalam sebuah *container*. Ini memastikan *backend* berjalan secara konsisten di berbagai lingkungan, mulai dari laptop pengembang hingga server produksi. Proses ini menyederhanakan *deployment*, skalabilitas, dan manajemen layanan *backend*, karena setiap layanan dapat diisolasi dan dikelola sebagai *container* independen.

Di sisi *frontend*, Docker juga memberikan banyak manfaat. Meskipun aplikasi *frontend* modern seringkali berupa aset statis (HTML, CSS, JavaScript), setelah proses *build*, Docker dapat digunakan untuk mengemas aset ini bersama dengan web server ringan untuk menyajikannya. Selain itu, Docker dapat menciptakan lingkungan *build* yang konsisten untuk aplikasi *frontend*, memastikan bahwa proses kompilasi dari framework seperti React, Angular, atau Vue selalu menghasilkan *output* yang sama, terlepas dari mesin yang menjalankannya. Dengan demikian, Docker memfasilitasi alur kerja yang lebih mulus antara pengembangan, pengujian, dan *deployment* untuk kedua sisi aplikasi.



Gambar 4. Ilustrasi Alur Kerja Docker

## BAB 3

# ANALISIS PEMECAHAN MASALAH

### 3.1 Langkah-langkah Pemecahan Masalah

Penyelesaian Tugas Besar 2 IF2211 Strategi Algoritma “Pemanfaatan Algoritma BFS dan DFS dalam Pencarian Recipe pada Permainan Little Alchemy 2” melibatkan serangkaian langkah pemecahan masalah yang terstruktur. Tujuan utamanya adalah membangun aplikasi *web* yang mampu menemukan dan memvisualisasikan cara membuat elemen-elemen dalam permainan Little Alchemy 2 dari empat elemen dasar (*Air, Earth, Fire, Water*). Berikut adalah uraian ide awal dan strategi yang ditempuh:

#### 3.1.1 Akuisisi dan Representasi Data Elemen dan Resep

Tantangan pertama adalah bagaimana mendapatkan data lengkap mengenai 720 elemen turunan beserta resep pembuatannya. Karena tidak ada API publik yang disediakan oleh permainan Little Alchemy 2, strategi yang paling memungkinkan adalah melakukan *web scraping*. Sumber data untuk *web scraping* didapatkan dari halaman Fandom Wiki Little Alchemy 2 (spesifiknya, [https://little-alchemy.fandom.com/wiki/Elements\\_\(Little\\_Alchemy\\_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2))). Halaman ini menyajikan daftar elemen dan cara pembuatannya dalam format yang relatif terstruktur (meskipun memerlukan inspeksi HTML mendalam). Untuk *backend* yang dibangun menggunakan Go, library goquery dipilih karena kemampuannya mem-*parsing* HTML dan melakukan seleksi elemen. *Scraper* dirancang untuk mengiterasi struktur HTML halaman Wiki (khususnya tabel atau daftar yang berisi elemen dan resepnya). Informasi yang diekstrak adalah nama elemen hasil dan pasangan nama elemen bahan. Dilakukan juga normalisasi nama dan filter awal untuk mengabaikan entri yang bukan elemen game inti (seperti *pack* ekspansi atau *header* tabel).

Setelah data resep mentah terkumpul, data tersebut perlu diubah menjadi struktur graf yang bisa dijelajahi oleh algoritma. Diputuskan untuk menggunakan `map[string]*model.RecipeTreeNodeTier` sebagai representasi graf utama di

memori. Kunci *map* adalah nama elemen, dan nilainya adalah pointer ke `model.RecipeTreeNode`. Struktur `model.RecipeTreeNodeTier` ini didefinisikan sebagai berikut:

```
type RecipeTreeNodeTier struct {
    NamaElemen string
    DibuatDari [][][2]*RecipeTreeNodeTier
}
```

### 3.2 Proses Pemecahan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS

Permasalahan utama terletak pada metode pencarian resep untuk sebuah elemen. Informasi yang dimiliki adalah elemen-elemen pembentuk dari sebuah elemen dengan *tier* yang lebih tinggi. Karena sebuah elemen hanya dapat dibentuk dari elemen yang memiliki *tier* lebih rendah darinya, struktur dari data hubungan antar-elemennya dapat digambarkan dalam bentuk *tree*. Sebuah *node* pada *tree* akan merepresentasikan sebuah elemen. Pencarian resep dari sebuah elemen dapat dilakukan dengan menggunakan metode *depth-first-search* (DFS) atau *breadth-first-search* (BFS). Metode DFS melakukan pencarian dengan menelusuri sebuah cabang hingga akhir secara rekursif, sebelum akhirnya berpindah ke cabang lainnya (mendalam), sedangkan metode BFS akan mencari semua cabang dari awal (melebar).

Tidak hanya itu, untuk opsi *multiple* resep, metode DFS dan BFS perlu dimodifikasi agar berhenti saat telah menemukan [input] resep, atau telah menelusuri seluruh *tree*. Maka dari itu, dilakukanlah penghitungan banyak resep yang telah ditemukan untuk setiap kali iterasi. Kunci dari menghitung banyak resep ini adalah bahwa banyak resep suatu elemen adalah jumlah dari perkalian dari banyak resep elemen-elemen pembangunnya. Pada DFS, penghitungan ini akan dilakukan setiap kali ia *backtrack* ke node parent-nya, sedangkan pada BFS, penghitungan ini akan dilakukan setiap ia telah menemukan elemen dasar.

Berikut adalah *pseudocode* untuk DFS:

```

Function dfs (idElemen, resepDicari, pengali) -> nodeTree :
    If nama elemen adalah basic, return
        totalResep = 0
    Iterasi sesuai banyak penyusun elemen:
        nodeKiri = dfs(idElemenKiri, resepDicari - totalResep, pengali)
        nodeKanan = dfs(idElemenKanan, resepDicari - totalResep, pengali * 
banyakResepKiri)
        totalResep = banyakResepKanan
    return

```

Untuk BFS, diperlukan fungsi helper untuk menghitung banyak resep yang telah ditemukan hingga saat itu. Fungsi ini diimplementasikan dengan cara meng-*update tree* dari dasar hingga paling atas. Berikut adalah pseudocode untuk helperBFS:

```

Function helperBFS (nodeSekarang) :
    Do
        banyakResepParent += banyakResepKiri*banyakResepKanan -
banyakResepPasanganSebelumUpdate
    While parent != nil
    return

```

Berikut adalah pseudocode untuk BFS:

```

Function bfs (idElemen, resepDicari) -> nodeTree :
    If nama elemen adalah basic, return
        totalResep = 0
    Masukkan setiap kemungkinan elemen pembangun ke queue
    Selama queue tidak kosong atau totalResep > resepDicari
        Majukan queue satu
        If elemen dasar, helperBFS(nodeSekarang) kemudian return
        Setiap anak dari elemen masukkan ke queue

```

### 3.3 Fitur Fungsional dan Arsitektur Aplikasi *Web* yang Dibangun

Aplikasi *web* ini dibangun dengan arsitektur *client-server* sebagai berikut: *Backend* (Go) bertugas sebagai “otak” yang menyimpan data graf, menjalankan algoritma, dan menyediakan API. Saat *startup*, *backend* melakukan *scraping*, membangun graf, menghitung tier, dan mentransformasi data menjadi `[]model.Element` untuk digunakan algoritma. Dipasang beberapa API *Endpoint*: GET /graph-data: menyajikan seluruh data `[]model.Element` yang sudah diproses (terurut sesuai penemuan *scraper*, dengan tier, dan difilter) sebagai JSON. GET /api/recipes/{elementName}: *Endpoint* utama untuk pencarian resep. Menerima parameter *algorithm*, *mode*, dan *count* untuk memanggil fungsi BFS atau DFS yang sesuai, kemudian mengembalikan hasil pencarian (termasuk pohon resep dalam format `model.RecipeTreeNodeTier`, waktu, dan jumlah *node*) sebagai JSON. Untuk Router, digunakan [github.com/gorilla/mux](https://github.com/gorilla/mux) untuk menangani *routing* permintaan HTTP ke fungsi *handler* yang tepat. Swagger/OpenAPI digunakan untuk mendokumentasikan *Endpoint* API secara interaktif. Karena sisi *backend* dan *frontend* berada pada alamat dan port yang berbeda, untuk menangani error CORS yang mencegah akses *resources* lintas alamat dan port, *router* yang digunakan pada pada *backend* diatur untuk menerima *request* dari alamat dan port yang berbeda dengan menetapkan `Access-Control-Allow-Origin`.

Program yang mengatur sisi *frontend* memiliki beberapa komponen utama sebagai berikut:

1. *Layout* (`layout.js`)

*Layout* adalah komponen dengan tingkat teratas yang membungkus semua komponen lain yang menjadi anak-anaknya. Komponen ini berperan untuk memberikan perilaku atau unsur tampilan yang sama untuk setiap komponen, baik itu *page* atau bukan. Pada program ini, komponen *layout* digunakan untuk memanggil komponen `<BackgroundMusic />` yang memutar musik sepanjang aplikasi dijalankan. Pada Layout juga dipasang *container* div yang berperan dalam menampilkan gambar *background* yang sama untuk setiap menu aplikasi. Selain itu, pendefinisian font eksternal yang digunakan juga dilakukan pada *file* ini.

## 2. Halaman Utama (page.js)

Aplikasi ini hanya memiliki sebuah *page* yang juga berperan sebagai halaman utama. Komponen ini mengatur animasi transisi antara setiap menu yang dapat dikunjungi pengguna: menu *home*, *start*, dan *about*. Halaman yang digunakan hanya satu karena aplikasi cukup sederhana dan cara ini memungkinkan pencegahan berhentinya musik bermain yang disebabkan perpindahan *page* melalui *routing*. Pada dasarnya, komponen ini mengatur perpindahan menu dengan mengatur *opacity* setiap komponen menu dan melakukan *rendering* secara kondisional.

## 3. Komponen Menu Home (home.js)

Komponen ini ditampilkan ketika pertama kali aplikasi dibuka. Home memiliki tombol *start* yang mengarahkan pengguna ke menu *start* untuk menjalankan fitur pencarian resep serta tombol *about* yang mengarahkan pengguna ke menu identitas kelompok pengembang.

## 4. Komponen Menu Start (start.js)

Komponen ini menjalankan fitur utama aplikasi yakni pencarian resep. Menu ini terdiri dari beberapa komponen tombol dan input teks atau angka untuk menerima masukan pengguna seperti pilihan algoritma (BFS/DFS, *bidirectional* tidak diimplementasikan), nama elemen yang ingin dicari, pilihan mode pencarian banyak resep atau tunggal, dan masukan jumlah resep yang ingin dicari. Input teks dan angka diimplementasikan pada komponen <RetroTextInput /> dan <RetroButton /> secara berturut-turut. Jika resep ditemukan, resep ditampilkan sebagai struktur pohon menggunakan komponen Network dari library Vis.js. Menu juga akan menampilkan lama waktu eksekusi algoritma dan jumlah simpul yang ditelusuri. Pencarian dimulai dengan menekan tombol dengan ikon kaca pembesar di sebelah kanan. Ketika tombol ditekan, aplikasi *frontend* melakukan permintaan ke *endpoint* API yang dimiliki *backend*. Alamat API disimpan sebagai *environment variable* yang terdapat pada file .env secara lokal atau pada pengaturan *environment variables* di Vercel. Jika respon diterima, data di-parse menjadi

JSON lalu diolah menjadi data JSON untuk menyimpan simpul (*nodes*) dan sisi (*edges*) sesuai format yang dapat diproses oleh Vis.js untuk divisualisasikan.

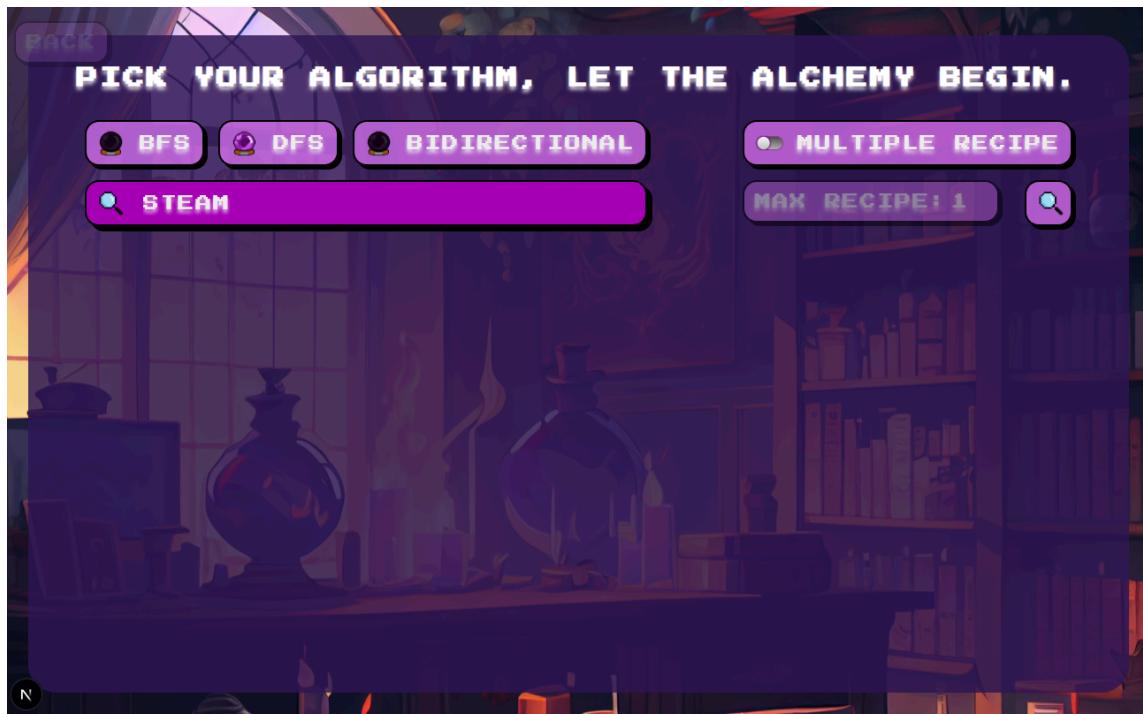
##### 5. Komponen Menu About (*about.js*)

Komponen ini menampilkan identitas anggota kelompok pengembang berupa asset publik berformat SVG.

#### 3.4 Contoh Ilustrasi Kasus

Pencarian resep “Steam”

##### 1. Penerimaan *Request* pada *Frontend*



Gambar 5. Penerimaan *Request User* dari *Frontend*

##### 2. Pengiriman *Request* dari *Frontend* ke *Backend*

```
try {  
  const algorithm = bfs ? 'bfs' : dfs ? 'dfs' : 'bidirectional';  
  const response = await  
fetch(`process.env.NEXT_PUBLIC_API_URL}/solve-recipe?element=${elements}&al`
```

```

gorithm=${algorithm}&count=${maxRecipe}`);
    const data = await response.json();
    setGraphData(convertDataToNetwork(data));
    setRecipeFound(data.recipes.length > 0);
    setSearched(true);
    setTime(data.searchTimeMs);
    setNode(data.nodesVisited);
}
catch (error) {
    setSearched(true);
}

```

### 3. Penerimaan *Request Backend* Menggunakan *Handler API*

*Backend* kemudian menerima *request* dan mengidentifikasi elemen target berupa “Steam” dengan parameter pencarian menggunakan DFS. Kemudian, dipanggil fungsi Dfs, handler melakukan Calling DFS for element ID 14 (Steam), needFound: 1.

### 4. Algoritma DFS

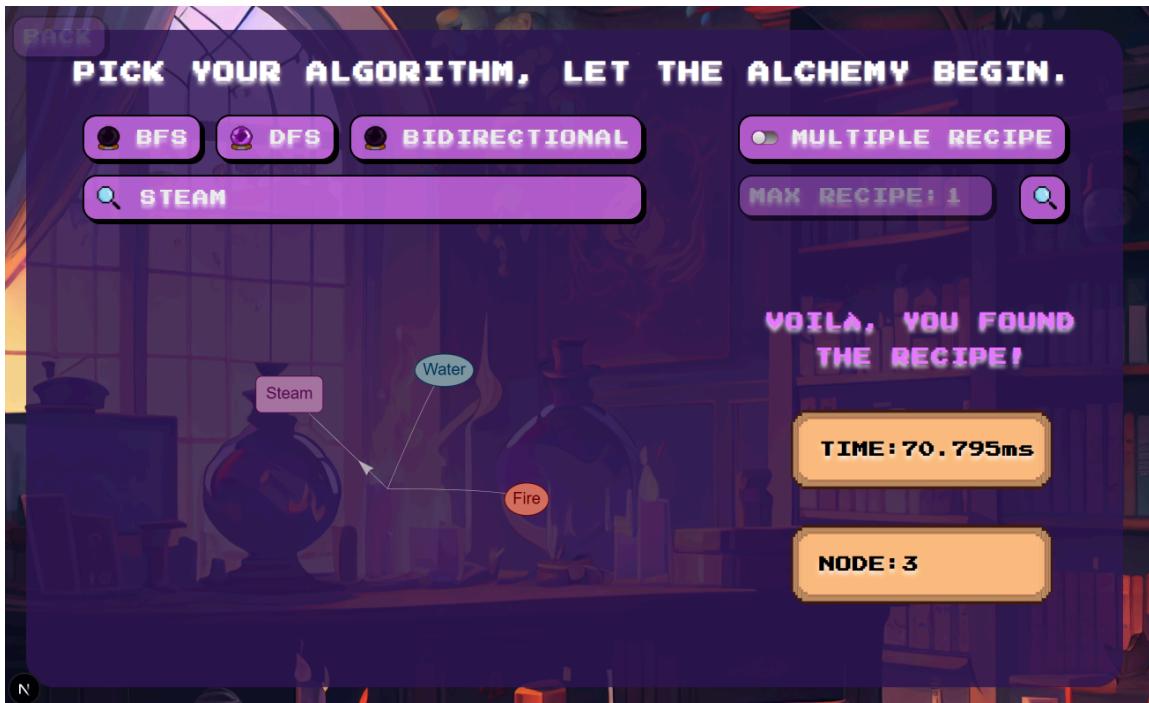
Program akan memanggil fungsi DFS, kemudian akan melakukan pencarian secara rekursif pada pasangan elemen-elemen pembangun steam. Kebetulan, saat pertama mencoba, pasangan elemen yang ditemukan adalah water dan fire sehingga langsung ditemukan satu resep DFS.

### 5. Penerimaan Hasil Algoritma Menggunakan *Handler API*

*Handler* kemudian menerima *tree* hasil pencarian resep, jumlah *node*, dan status *found=true* dari DFS. Kemudian, *handler* menghitung waktu eksekusi dan membentuk respons JSON yang berisi pohon *model.RecipeTreeNode* untuk “Steam”, serta informasi lain yang dibutuhkan oleh *Frontend*.

### 6. Visualisasi Pohon Resep

*Frontend* kemudian menerima respons JSON dan menampilkan visualisasi pohon resep “Steam” beserta informasi waktu/jumlah *node*.



Gambar 6. Visualisasi Pohon Resep oleh *Frontend*

## BAB 4

# IMPLEMENTASI DAN PENGUJIAN

### 4.1 Struktur Data

```
type RecipeTreeNode struct {
    NamaElemen string
    DibuatDari []RecipeTreeNodeChild
    BanyakResep int
    Parent      *RecipeTreeNodeChild
}
```

Struktur ini adalah blok utama untuk pohon solusi resep yang dihasilkan oleh algoritma DFS. Setiap *node* merepresentasikan sebuah elemen dalam resep, dan melalui *DibuatDari* ditunjukkan bagaimana elemen tersebut dapat diuraikan menjadi komponen-komponen penyusunnya. *BanyakResep* memberikan metrik tentang variasi pembuatan elemen tersebut.

```
type RecipeTreeNodeChild struct {
    Parent      *RecipeTreeNode
    LeftChild   *RecipeTreeNode
    RightChild  *RecipeTreeNode
    LeftChildID int
    RightChildID int
}
```

Struktur ini merepresentasikan satu spesifik kombinasi resep yang menghasilkan elemen Parent-nya. Dalam pohon solusi, ini adalah relasi yang menghubungkan elemen hasil (*parent*) dengan dua elemen bahan (*children*). *RecipeTreeNode.DibuatDari* adalah kumpulan dari *RecipeTreeNodeChild* ini, menunjukkan semua pasangan bahan yang bisa membuat elemen tersebut.

```
type RecipeTreeNodeTier struct {
    NamaElemen string
    DibuatDari [][][2]*RecipeTreeNodeTier
}
```

Ini adalah struktur data sementara yang digunakan selama fase pemrosesan data oleh *scraper*. Tujuannya adalah untuk merepresentasikan graf elemen dan resepnya dalam bentuk yang mudah untuk diiterasi dan dianalisis guna menentukan *tier* setiap elemen. Setelah *tier* dihitung dan ID ditetapkan, data ini diubah menjadi struktur Element.

```
type Element struct {
    Id      int
    Tier   int
    Name   string
    FromPair [][]int
    CanMake []int
}
```

Ini adalah struktur data yang merepresentasikan setiap elemen dalam permainan setelah diproses oleh *scraper*.

```
type RecipeSolution struct {
    ElementName string `json:"elementName"`
    SearchParams SearchParams `json:"searchParams"`
    Found       bool `json:"found"`
    SearchTimeMs float64 `json:"searchTimeMs"`
    NodesVisited int `json:"nodesVisited"`
    Recipes     []RecipePath `json:"recipes"`
}
```

Struktur ini dirancang untuk representasi data resep dalam format JSON yang akan dikirim sebagai respons API.

```
type SearchParams struct {
    Algorithm string `json:"algorithm"`
    Count     int    `json:"count"`
    Mode      string `json:"mode"`
}
```

Struktur ini adalah bentuk struktur penerimaan respons dari *Frontend* untuk kemudian diproses pada *Backend* oleh algoritma yang diinginkan oleh pengguna.

```
type RecipePath struct {
    NamaElemen   string      `json:"namaElemen"`
    IsBaseElement bool       `json:"isBaseElement"`
    DibuatDari   [][]RecipePath `json:"dibuatDari,omitempty"`
}
```

```
}
```

Struktur ini adalah bentuk *recipe path* yang dikirimkan oleh *Backend* yang dapat di-fetch dan ditampilkan oleh *Frontend*.

## 4.2 Fungsi dan Prosedur Program

BFSal.go:

Fungsi/Prosedur	Penjelasan
<pre>func Bfs(start int, needFound int, mult int) *RecipeTreeNode {     // log.Printf("[BFS_DEBUG] === Memulai BFS untuk start (digunakan sbg INDEKS): %d, needFound: %d, mult: %d. Panjang elements: %d ===", start, needFound, mult, len(elements))      // PENTING: Logika asli menggunakan 'start' langsung sebagai indeks.     // Jika 'start' adalah ID 1-based dari handler, ini akan salah.     // Log akan menunjukkan elemen apa yang diakses.      if start &lt; 0    start &gt;= len(elements) {         // log.Printf("[BFS_DEBUG] ERROR: 'start' (%d) sebagai INDEKS di luar jangkauan 'elements' (panjang %d). Mengembalikan Tree kosong.", start, len(elements))         return &amp;RecipeTreeNode{             NamaElemen: fmt.Sprintf("Error_Indeks_Root_%d_Invalid", start),             DibuatDari: make([]RecipeTreeNodeChild, 0),             BanyakResep: 0,         }     }     // log.Printf("[BFS_DEBUG] Root Tree akan dibuat dari elements[%d]: Nama=%s', ID=%d, Tier=%d", start, elements[start].Name, elements[start].ID, elements[start].Tier)      var Tree = RecipeTreeNode{         NamaElemen: elements[start].Name,</pre>	Kode algoritma BFS yang dilakukan secara rekursif.

```

DibuatDari: make([]RecipeTreeNodeChild, 0),
BanyakResep: 0,
// Parent untuk root Tree adalah nil, ini OK.
}
// log.Printf("[BFS_DEBUG] Tree root '%s' dibuat.
BanyakResep awal: %d", Tree.NamaElemen, Tree.BanyakResep)

queue := make([]RecipeTreeNodeChild, 0)

// --- Inisialisasi Queue Awal ---
if len(elements[start].FromPair) == 0 {
    // log.Printf("[BFS_DEBUG] [InitQueue] Elemen
root '%s' (dari INDEKS %d) tidak memiliki FromPair.",

Tree.NamaElemen, start)
}
for i := 0; i < len(elements[start].FromPair); i++ {
    pair := elements[start].FromPair[i]
    // ID dari FromPair adalah ID elemen, BUKAN
indeks. Ini perlu diperhatikan saat mengakses 'elements'
lagi.
    leftChildID_fromPair := pair[0]
    rightChildID_fromPair := pair[1]
    // log.Printf("[BFS_DEBUG] [InitQueue] Memeriksa
pair untuk root '%s': LeftChildID=%d, RightChildID=%d",
Tree.NamaElemen, leftChildID_fromPair,
rightChildID_fromPair)

    // IsValid menerima ID yang akan digunakannya
sebagai INDEKS jika mengikuti logika asli IsValid.
    if IsValid(start, leftChildID_fromPair,
rightChildID_fromPair) {
        // log.Printf("[BFS_DEBUG] [InitQueue] Pair
(LeftID: %d, RightID: %d) VALID.", leftChildID_fromPair,
rightChildID_fromPair)

        // Mengakses elemen anak menggunakan ID dari
pair SEBAGAI INDEKS (sesuai logika asli)
        // Ini berpotensi besar menjadi sumber
masalah jika ID != Indeks
        if leftChildID_fromPair < 0 ||
leftChildID_fromPair >= len(elements) ||
rightChildID_fromPair < 0 || rightChildID_fromPair >=
len(elements) {
            // log.Printf("[BFS_DEBUG] [InitQueue]

```

```

ERROR: ID anak dari FromPair (L:%d, R:%d) di luar
jangkauan jika digunakan sebagai INDEKS. Melewati pair.",
leftChildID_fromPair, rightChildID_fromPair)
    continue
}
// log.Printf("[BFS_DEBUG] [InitQueue]
Membuat LeftChild dari elements[%d]: Nama='%s',
leftChildID_fromPair,
elements[leftChildID_fromPair].Name)
// log.Printf("[BFS_DEBUG] [InitQueue]
Membuat RightChild dari elements[%d]: Nama='%s',
rightChildID_fromPair,
elements[rightChildID_fromPair].Name)

// 1. Buat node RecipeTreeNode (anak) -
Parent belum diset
tempLeftChildNode := &RecipeTreeNode{
    NamaElemen:
elements[leftChildID_fromPair].Name, // Menggunakan ID
sebagai INDEKS
    DibuatDari: make([]RecipeTreeNodeChild,
0),
    BanyakResep: 0,
}
tempRightChildNode := &RecipeTreeNode{
    NamaElemen:
elements[rightChildID_fromPair].Name, // Menggunakan ID
sebagai INDEKS
    DibuatDari: make([]RecipeTreeNodeChild,
0),
    BanyakResep: 0,
}

// 2. Buat RecipeTreeNodeChild
var childRelation = RecipeTreeNodeChild{
    Parent:      &Tree,
    LeftChild:   tempLeftChildNode,
    RightChild:  tempRightChildNode,
    LeftChildID: leftChildID_fromPair, //
Simpan ID asli
    RightChildID: rightChildID_fromPair, //
Simpan ID asli
}

```

```

        // 3. ATUR Parent dari node anak untuk
menunjuk ke childRelation
        // Ini adalah perbaikan yang disarankan
sebelumnya untuk mengatasi panic di BfsHelper
        tempLeftChildNode.Parent = &childRelation
        tempRightChildNode.Parent = &childRelation
        // log.Printf("[BFS_DEBUG] [InitQueue]
Pointer Parent untuk '%s' dan '%s' telah diatur ke
RecipeTreeNodeChild.", tempLeftChildNode>NamaElemen,
tempRightChildNode>NamaElemen)

queue = append(queue, childRelation)
// log.Printf("[BFS_DEBUG] [InitQueue]
Menambahkan ke queue: Parent='%s', Left='%s'(ID:%d),
Right='%s'(ID:%d). Queue size:
%d", childRelation.Parent>NamaElemen,
tempLeftChildNode>NamaElemen, leftChildID_fromPair,
tempRightChildNode>NamaElemen, rightChildID_fromPair,
len(queue))
} else {
    // log.Printf("[BFS_DEBUG] [InitQueue] Pair
(LeftID: %d, RightID: %d) TIDAK VALID.",
leftChildID_fromPair, rightChildID_fromPair)
}
// log.Printf("[BFS_DEBUG] Inisialisasi queue
selesai. Ukuran queue akhir: %d", len(queue))

// --- Loop Utama BFS ---
processedInQueue := 0
for len(queue) > 0 {
    processedInQueue++
    current := queue[0]
    queue = queue[1:]

    if current.Parent == nil {
        continue
    }
    current.Parent.DibuatDari =
append(current.Parent.DibuatDari, current)

    // Proses LeftChild dari 'current'
    if IsBasicElement(*current.LeftChild) {

```

```

        BfsHelperORI(current.LeftChild)
    } else {
        expandingParentNode := current.LeftChild // Ini adalah *RecipeTreeNode yang akan menjadi .Parent untuk anak-anaknya
        expandingID_asIndex := current.LeftChildID // INI ADALAH ID, AKAN DIGUNAKAN SEBAGAI INDEKS (POTENSI MASALAH)

        if expandingID_asIndex < 0 ||
expandingID_asIndex >= len(elements) {
            } else {

                for i := 0; i <
len(elements[expandingID_asIndex].FromPair); i++ {
                    pair :=
elements[expandingID_asIndex].FromPair[i]
                    grandLeftID_fromPair := pair[0]
                    grandRightID_fromPair := pair[1]

                    if IsValid(current.LeftChildID,
grandLeftID_fromPair, grandRightID_fromPair) { // IsValid pakai ID (yg akan jadi indeks)

                        if grandLeftID_fromPair < 0 ||
grandLeftID_fromPair >= len(elements) ||
grandRightID_fromPair < 0 || grandRightID_fromPair >=
len(elements) {
                            continue
                        }

                        tempGrandLeftNode :=
&RecipeTreeNode{NamaElemen:
elements[grandLeftID_fromPair].Name, DibuatDari:
make([]RecipeTreeNodeChild, 0)}
                        tempGrandRightNode :=
&RecipeTreeNode{NamaElemen:
elements[grandRightID_fromPair].Name, DibuatDari:
make([]RecipeTreeNodeChild, 0)}

                        var newChildRelation =
RecipeTreeNodeChild{
                            Parent:
expandingParentNode, // Parent-nya adalah

```

```

current.LeftChild
                    LeftChild:
tempGrandLeftNode,
                    RightChild:
tempGrandRightNode,
                    LeftChildID:
grandLeftID_fromPair,
                    RightChildID:
grandRightID_fromPair,
                }
                tempGrandLeftNode.Parent =
&newChildRelation
                tempGrandRightNode.Parent =
&newChildRelation

                queue = append(queue,
newChildRelation)
            }
        }
    }

    // Proses RightChild dari 'current' (logika
serupa dengan LeftChild)
    if IsBasicElement(*current.RightChild) {
        BfsHelperORI(current.RightChild)
    } else {
        expandingParentNode := current.RightChild
        expandingID_asIndex := current.RightChildID
// ID DIGUNAKAN SEBAGAI INDEKS

        if expandingID_asIndex < 0 ||
expandingID_asIndex >= len(elements) {
            } else {

                for i := 0; i <
len(elements[expandingID_asIndex].FromPair); i++ {
                    pair :=
elements[expandingID_asIndex].FromPair[i]
                    grandLeftID_fromPair := pair[0]
                    grandRightID_fromPair := pair[1]

                    if IsValid(current.RightChildID,
grandLeftID_fromPair, grandRightID_fromPair) { // IsValid

```

pakai ID (yg akan jadi indeks)

```
        if grandLeftID_fromPair < 0 ||
grandLeftID_fromPair >= len(elements) ||
grandRightID_fromPair < 0 || grandRightID_fromPair >=
len(elements) {
            continue
        }

        tempGrandLeftNode :=
&RecipeTreeNode{NamaElemen:
elements[grandLeftID_fromPair].Name, DibuatDari:
make([]RecipeTreeNodeChild, 0)}
        tempGrandRightNode :=
&RecipeTreeNode{NamaElemen:
elements[grandRightID_fromPair].Name, DibuatDari:
make([]RecipeTreeNodeChild, 0)}

        var newChildRelation =
RecipeTreeNodeChild{
            Parent:
expandingParentNode,
            LeftChild:
tempGrandLeftNode,
            RightChild:
tempGrandRightNode,
            LeftChildID:
grandLeftID_fromPair,
            RightChildID:
grandRightID_fromPair,
        }
        tempGrandLeftNode.Parent =
&newChildRelation
        tempGrandRightNode.Parent =
&newChildRelation

        queue = append(queue,
newChildRelation)
    }
}
}

if Tree.BanyakResep >= needFound {
```

```

        break
    }

    var res = &Tree
    return BFSCleaner(res)
}

func BfsHelperORI(a *RecipeTreeNode) {
    kiri := (a.Parent.LeftChild == a)
    newBanyakResep := 1
    for a.Parent != nil {
        if a.Parent.LeftChild != nil &&
a.Parent.RightChild != nil {
            if a.Parent.LeftChild.BanyakResep > 0
&& a.Parent.RightChild.BanyakResep > 0 {
                a.Parent.Parent.BanyakResep --
a.Parent.LeftChild.BanyakResep *
a.Parent.RightChild.BanyakResep
                a.BanyakResep = newBanyakResep
            } else if
a.Parent.LeftChild.BanyakResep == 0 &&
a.Parent.RightChild.BanyakResep == 0 {
                a.BanyakResep = newBanyakResep
                newBanyakResep = 0
            } else if (kiri &&
a.Parent.LeftChild.BanyakResep == 0) || (!kiri &&
a.Parent.RightChild.BanyakResep == 0) {
                a.BanyakResep = newBanyakResep
                if kiri {
                    newBanyakResep =
a.BanyakResep * a.Parent.RightChild.BanyakResep
                } else {
                    newBanyakResep =
a.BanyakResep * a.Parent.LeftChild.BanyakResep
                }
            } else if (kiri &&
a.Parent.RightChild.BanyakResep == 0) || (!kiri &&
a.Parent.LeftChild.BanyakResep == 0) {
                a.BanyakResep = newBanyakResep
                newBanyakResep = 0
            } else { // ERROR
!!!!!!!!!!!!!!!!!!!!!!
                newBanyakResep = 0
            }
        }
    }
}

```

Digunakan untuk melakukan backtrack ke *root* dari *tree* setiap kali menemukan ujung.

<pre>         a = a.Parent.Parent     } } a.BanyakResep += newBanyakResep  } </pre>	
<pre> func IsBasicElement(a RecipeTreeNode) bool {     // Normalisasi string nama elemen dasar agar     konsisten (misal semua TitleCase)     // Logika asli Anda menggunakan TitleCase, jadi kita     asumsikan NamaElemen juga TitleCase.     isBase := a.NamaElemen == "Water"    a.NamaElemen ==     "Fire"    a.NamaElemen == "Earth"    a.NamaElemen ==     "Air"     // log.Printf("[IS_BASIC_DEBUG] IsBasicElement untuk     '%s': %t", a.NamaElemen, isBase)     return isBase }  func IsBasicElement_typeElement(a scraper.Element) bool {     // Sama seperti di atas, perhatikan kapitalisasi     isBase := a.Name == "Water"    a.Name == "Fire"        a.Name == "Earth"    a.Name == "Air"     // log.Printf("[IS_BASIC_TYPE_DEBUG]     IsBasicElement_typeElement untuk '%s' (ID %d): %t",     a.Name, a.ID, isBase)     return isBase } </pre>	<p>Digunakan untuk mengecek apakah merupakan elemen dasar.</p>
<pre> func IsValid(idParent int, idLeftChild int, idRightChild int) bool {     // log.Printf("[IS_VALID_DEBUG] Memeriksa IsValid:     idParent=%d, idLeftChild=%d, idRightChild=%d. Panjang     elements: %d", idParent, idLeftChild, idRightChild,     len(elements))      // PENTING: Kode ini mengasumsikan ID adalah INDEKS.     // Jika ID adalah 1-based, ini akan salah.     // Log akan membantu melihat elemen apa yang diakses.      // Pengecekan batas untuk semua ID yang digunakan     sebagai indeks     if idParent &lt; 0    idParent &gt;= len(elements) {         // log.Printf("[IS_VALID_DEBUG] ERROR: idParent </pre>	<p>Saringan ganda untuk mencegah adanya elemen tidak valid.</p>

```

(%d) di luar jangkauan 'elements' (panjang %d).
Mengembalikan false.", idParent, len(elements))
    return false
}
elParent := elements[idParent]

if idLeftChild < 0 || idLeftChild >= len(elements) {
    // log.Printf("[IS_VALID_DEBUG] ERROR:
idLeftChild (%d) di luar jangkauan 'elements' (panjang
%d). Mengembalikan false.", idLeftChild, len(elements))
    return false
}
elLeft := elements[idLeftChild]

if idRightChild < 0 || idRightChild >= len(elements)
{
    // log.Printf("[IS_VALID_DEBUG] ERROR:
idRightChild (%d) di luar jangkauan 'elements' (panjang
%d). Mengembalikan false.", idRightChild, len(elements))
    return false
}
elRight := elements[idRightChild]

// log.Printf("[IS_VALID_DEBUG] Data untuk IsValid:
Parent (diakses pada indeks %d): '%s' (ID %d, Tier %d),
Left (diakses pada indeks %d): '%s' (ID %d, Tier %d),
Right (diakses pada indeks %d): '%s' (ID %d, Tier
%d)", idParent, elParent.Name, elParent.ID,
elParent.Tier, idLeftChild, elLeft.Name, elLeft.ID,
elLeft.Tier, idRightChild, elRight.Name, elRight.ID,
elRight.Tier)

result := elParent.Tier > elLeft.Tier &&
elParent.Tier > elRight.Tier
// log.Printf("[IS_VALID_DEBUG] Hasil perbandingan
Tier: (%d > %d && %d > %d) -> %t", elParent.Tier,
elLeft.Tier, elParent.Tier, elRight.Tier, result)
return result
}

```

```

func BFSCleaner(tree *RecipeTreeNode) *RecipeTreeNode {
    // log.Println("[BFSCleaner] Memulai pembersihan tree
untuk: %s", tree.NamaElemen)
    // log.Println("[BFSCleaner] BanyakResep: %d",
tree.BanyakResep)

```

Digunakan untuk  
membersihkan *tree*  
dari elemen yang tidak

```

if tree == nil || tree.BanyakResep == 0 {
    return nil
}

newTree := &RecipeTreeNode{
    NamaElemen: tree.NamaElemen,
    DibuatDari: make([]RecipeTreeNodeChild, 0),
    BanyakResep: tree.BanyakResep,
}

for _, child := range tree.DibuatDari {
    // log.Println("[BFSCleaner] Memproses child dari
tree: %s", child.Parent.NamaElemen)
    leftCleaned := BFSCleaner(child.LeftChild)
    rightCleaned := BFSCleaner(child.RightChild)

    if leftCleaned != nil && rightCleaned != nil {
        newChild := RecipeTreeNodeChild{
            Parent:      newTree,
            LeftChild:   leftCleaned,
            RightChild:  rightCleaned,
            LeftChildID: child.LeftChildID,
            RightChildID: child.RightChildID,
        }
        newTree.DibuatDari =
append(newTree.DibuatDari, newChild)
    }
}

if len(newTree.DibuatDari) == 0 &&
len(tree.DibuatDari) > 0 {
    return nil
}

return newTree
}

func ParallelBfs(targetID, needCount int) *RecipeTreeNode
{
    // log.Print("[PARALLEL_BFS_DEBUG] === Memulai
Parallel BFS untuk targetID: %d, needCount: %d. Panjang
elements: %d ===", targetID, needCount, len(elements))
    queue := make([]RecipeTreeNodeChild, 0)

    if targetID < 0 || targetID >= len(elements) {
        return nil
    }
}

```

mencapai akhir.

Digunakan untuk BFS Paralel (*error*).

```

}

root := RecipeTreeNode{
    Parent:      nil,
    NamaElemen: elements[targetID].Name,
    DibuatDari:  []RecipeTreeNodeChild{},
    BanyakResep: 0,
}

var queueMu sync.Mutex
var treeMu sync.Mutex // Protects all tree
modifications
var resepCount int32 = 0
var wg sync.WaitGroup

// Initial queue population with validity checks
for _, pair := range elements[targetID].FromPair {
    if !IsValid(targetID, pair[0], pair[1]) {
        continue
    }

    left := &RecipeTreeNode{
        NamaElemen: elements[pair[0]].Name,
        DibuatDari:  make([]RecipeTreeNodeChild, 0),
        BanyakResep: 0,
    }
    right := &RecipeTreeNode{
        NamaElemen: elements[pair[1]].Name,
        DibuatDari:  make([]RecipeTreeNodeChild, 0),
        BanyakResep: 0,
    }

    child := RecipeTreeNodeChild{
        Parent:      &root,
        LeftChild:   left,
        RightChild:  right,
        LeftChildID: pair[0],
        RightChildID: pair[1],
    }
    left.Parent = &child
    right.Parent = &child

    queue = append(queue, child)
}

```

```

        for len(queue) > 0 && atomic.LoadInt32(&resepCount) <
int32(needCount) {
    // log.Println("[PARALLEL_BFS_DEBUG] Queue
size:", len(queue))
    queueMu.Lock()
    if len(queue) == 0 {
        queueMu.Unlock()
        break
    }
    item := queue[0]
    queue = queue[1:]

    queueMu.Unlock()
    wg.Add(1)
    go func(node RecipeTreeNodeChild) {
        defer wg.Done()
        // log.Println("[PARALLEL_BFS_DEBUG]
Memproses node:", node.LeftChild.NamaElemen, "dan",
node.RightChild.NamaElemen)

        // Add relation to parent with lock
        treeMu.Lock()
        node.Parent.DibuatDari =
append(node.Parent.DibuatDari, node)
        treeMu.Unlock()

        processChild := func(childID int, childNode
*RecipeTreeNode) {
            if childID < 0 || childID >=
len(elements) {
                return
            }

            if
IsBasicElement_typeElement(*elements[childID]) {
                // log.Println("[PARALLEL_BFS_DEBUG]
Memproses elemen dasar:", childNode.NamaElemen)
                // tempChildNode := childNode
                treeMu.Lock()
                BfsHelperORI(childNode)
                atomic.StoreInt32(&resepCount,
int32(root.BanyakResep))
                treeMu.Unlock()
            }
        }
    }
}

```

```

    } else {
        // log.Println("[PARALLEL_BFS_DEBUG]
Memproses anak dari node:", childNode.NamaElemen)
        for _, pair := range
elements[childID].FromPair {
            if !IsValid(childID, pair[0],
pair[1]) {
                //
log.Println("[PARALLEL_BFS_DEBUG] Pair tidak valid:",
pair, "dari ID:", childID)
                continue
            }

            left := &RecipeTreeNode{
                NamaElemen:
elements[pair[0]].Name,
                DibuatDari:
make([]RecipeTreeNodeChild, 0),
                BanyakResep: 0,
            }
            right := &RecipeTreeNode{
                NamaElemen:
elements[pair[1]].Name,
                DibuatDari:
make([]RecipeTreeNodeChild, 0),
                BanyakResep: 0,
            }

            newChild := RecipeTreeNodeChild{
                Parent:      childNode,
                LeftChild:   left,
                RightChild:  right,
                LeftChildID: pair[0],
                RightChildID: pair[1],
            }
            left.Parent = &newChild
            right.Parent = &newChild

            // Add to queue safely
            queueMu.Lock()
            queue = append(queue, newChild)
            queueMu.Unlock()
            //
log.Println("[PARALLEL_BFS_DEBUG] Menambahkan ke queue:",
```

```

newChild.LeftChild.NamaElemen,
newChild.RightChild.NamaElemen, "Parent:",
newChild.Parent.NamaElemen)
        }
    }
}
// log.Println("[PARALLEL_BFS_DEBUG-1] Queue
size:", len(queue))
processChild(node.LeftChildID,
node.LeftChild)
// log.Println("[PARALLEL_BFS_DEBUG0] Queue
size:", len(queue))
processChild(node.RightChildID,
node.RightChild)
// log.Println("[PARALLEL_BFS_DEBUG1] Queue
size:", len(queue))
}(item)
wg.Wait()
// log.Println("[PARALLEL_BFS_DEBUG2] Queue
size:", len(queue))
}
// log.Println("[PARALLEL_BFS_DEBUG3] Queue size:",
len(queue))

// root.BanyakResep =
int(atomic.LoadInt32(&resepCount))
// root.BanyakResep = 3
return BFSCleaner(&root)
}

```

DFSal.go:

Fungsi/Prosedur	Penjelasan
<pre> func Dfs(start int, needFound int, mult int) *RecipeTreeNode {     // log.Printf("[DFS_DEBUG] Memulai DFS untuk start (ID/Indeks?): %d, needFound: %d, mult: %d. Panjang elements: %d", start, needFound, mult, len(elements))      // Pengecekan batas KRUSIAL sebelum mengakses elements[start]     if start &lt; 0    start &gt;= len(elements) {         // log.Printf("[DFS_DEBUG] ERROR: 'start' (%d) di </pre>	Kode algoritma DFS yang dilakukan secara rekursif.

```

luar jangkauan slice 'elements' (panjang: %d).
Mengembalikan nil.", start, len(elements))
    return nil // Jika 'start' adalah indeks, ini
    penting. Jika 'start' adalah ID 1-based, maka akses
    seharusnya elements[start-1]
}
// Jika 'start' adalah ID 1-based dan Anda belum
mengubah logika, log ini akan menunjukkan elemen yang
salah
// log.Printf("[DFS_DEBUG] Mengakses elements[%d]:"
Nama='%s', ID=%d, Tier=%d, Jumlah FromPair=%d",start,
elements[start].Name, elements[start].ID,
elements[start].Tier, len(elements[start].FromPair))

if IsBasicElement_typeElement(*elements[start]) {
    // log.Printf("[DFS_DEBUG] Elemen '%s' (ID: %d)
adalah elemen dasar. Mengembalikan node dasar.",
elements[start].Name, elements[start].ID)
    return &RecipeTreeNode{
        NamaElemen: elements[start].Name,
        DibuatDari: nil,
        BanyakResep: 1,
    }
}

// log.Printf("[DFS_DEBUG] Elemen '%s' (ID: %d) BUKAN
elemen dasar. Membuat Tree.", elements[start].Name,
elements[start].ID)
var Tree = RecipeTreeNode{
    NamaElemen: elements[start].Name,
    DibuatDari: make([]RecipeTreeNodeChild, 0),
    BanyakResep: 0,
}
// log.Printf("[DFS_DEBUG] Tree untuk '%s' dibuat.
BanyakResep awal: %d", Tree.NamaElemen, Tree.BanyakResep)

if len(elements[start].FromPair) == 0 {
    // log.Printf("[DFS_DEBUG] Elemen '%s' (ID: %d)
tidak memiliki FromPair. Akan mengembalikan Tree dengan
BanyakResep: %d.", Tree.NamaElemen, elements[start].ID,
Tree.BanyakResep)
}

for i := 0; i < len(elements[start].FromPair); i++ {

```

```

// parentElementName := elements[start].Name //
Untuk logging
    pair := elements[start].FromPair[i]
    leftChildID := pair[0]
    rightChildID := pair[1]
    // log.Printf("[DFS_DEBUG] [%s] Iterasi FromPair
ke-%d: LeftChildID=%d, RightChildID=%d",
parentElementName, i, leftChildID, rightChildID)

    if !IsValid(start, leftChildID, rightChildID) {
        // log.Printf("[DFS_DEBUG] [%s] Pair (LeftID:
%d, RightID: %d) TIDAK VALID menurut IsValid. Melanjutkan
ke pair berikutnya.", parentElementName, leftChildID,
rightChildID)
        continue
    }
    // log.Printf("[DFS_DEBUG] [%s] Pair (LeftID: %d,
RightID: %d) VALID. Memanggil DFS rekursif.",
parentElementName, leftChildID, rightChildID)

    // Logika needFound-Tree.BanyakResep bisa jadi
kompleks, mari kita log nilainya
    needFoundForLeft := needFound - Tree.BanyakResep
    // log.Printf("[DFS_DEBUG] [%s] Memanggil DFS
untuk LeftChildID: %d, needFoundForLeft: %d",
parentElementName, leftChildID, needFoundForLeft)
    var leftNode = Dfs(leftChildID, needFoundForLeft,
mult) // `mult` mungkin perlu disesuaikan untuk anak
kanan

    if leftNode == nil {
        // log.Printf("[DFS_DEBUG] [%s] Panggilan DFS
untuk LeftChildID: %d mengembalikan nil. Mungkin tidak
bisa membentuk resep ini.", parentElementName,
leftChildID)
        // Jika salah satu anak nil, mungkin
kombinasi ini tidak valid atau tidak bisa diselesaikan
        // Tergantung logika bisnis, Anda bisa
`continue` atau menangani kasus ini
        // Untuk saat ini, biarkan logika asli
berjalan, tapi waspadai dampaknya
    } else {
        // log.Printf("[DFS_DEBUG] [%s] Panggilan DFS
untuk LeftChildID: %d selesai. leftNode.BanyakResep: %d",

```

```

parentElementName, leftChildID, leftNode.BanyakResep)
}

//leftnode tidak ditemukan
if leftNode.BanyakResep == 0 {
    // log.Printf("[DFS_DEBUG] [%s]
leftNode.BanyakResep adalah 0. Menghentikan loop
FromPair.", parentElementName)
    continue
}

// mult untuk anak kanan menggunakan
leftNode.BanyakResep
needFoundForRight := needFound - Tree.BanyakResep
// Ini mungkin perlu dihitung ulang jika Tree.BanyakResep
berubah oleh leftNode
multForRight := 1
// Default, jika leftNode nil atau BanyakResep 0
if leftNode != nil {
    multForRight = leftNode.BanyakResep * mult
}
// log.Printf("[DFS_DEBUG] [%s] Memanggil DFS
untuk RightChildID: %d, needFoundForRight: %d,
multForRight: %d", parentElementName, rightChildID,
needFoundForRight, multForRight)
var rightNode = Dfs(rightChildID,
needFoundForRight, multForRight)

if rightNode == nil {
    // log.Printf("[DFS_DEBUG] [%s] Panggilan DFS
untuk RightChildID: %d mengembalikan nil. Mungkin tidak
bisa membentuk resep ini.", parentElementName,
rightChildID)
    // Sama seperti leftNode, tangani jika perlu
} else {
    // log.Printf("[DFS_DEBUG] [%s] Panggilan DFS
untuk RightChildID: %d selesai. rightNode.BanyakResep:
%d", parentElementName, rightChildID,
rightNode.BanyakResep)
}

// rightNode tidak ditemukan
if rightNode.BanyakResep == 0 {
    // log.Printf("[DFS_DEBUG] [%s]

```

```

rightNode.BanyakResep adalah 0. Menghentikan loop
FromPair.", parentElementName)
    continue
}

// Hanya tambahkan jika kedua node valid, untuk
menghindari panic jika salah satunya nil
if leftNode != nil && rightNode != nil {
    Tree.DibuatDari = append(Tree.DibuatDari,
RecipeTreeNodeChild{Parent: &Tree, LeftChild: leftNode,
RightChild: rightNode, LeftChildID: leftChildID,
RightChildID: rightChildID})
    newRecipesFromPair := rightNode.BanyakResep
    Tree.BanyakResep += newRecipesFromPair
    // log.Printf("[DFS_DEBUG] [%s] Pair (LeftID:
%d, RightID: %d) ditambahkan ke DibuatDari. Resep baru
dari pair: %d. Total Tree.BanyakResep:
%d", parentElementName, leftChildID, rightChildID,
newRecipesFromPair, Tree.BanyakResep)
} else {
    // log.Printf("[DFS_DEBUG] [%s] Tidak
menambahkan pair (LeftID: %d, RightID: %d) karena satu
atau kedua node anak adalah nil.", parentElementName,
leftChildID, rightChildID)
}

if Tree.BanyakResep >= needFound {
    // log.Printf("[DFS_DEBUG] [%s]
Tree.BanyakResep (%d) >= needFound (%d). Menghentikan
loop FromPair.", parentElementName, Tree.BanyakResep,
needFound)
    break
}
var result = &Tree
// log.Printf("[DFS_DEBUG] Selesai DFS untuk '%s' (ID
dari start: %d). Mengembalikan Tree.BanyakResep: %d.
Jumlah DibuatDari: %d", result.NamaElemen, start,
result.BanyakResep, len(result.DibuatDari))
return result
}

```

```

func ParallelDFS(targetID, needFound, maxGoroutine int)
*RecipeTreeNode {
    var wg sync.WaitGroup

```

Memanggil dfs secara parallel. Fungsi ini

```

found := int32(0)
sema := make(chan struct{}, maxGoroutine)
var mu sync.Mutex // Mutex to protect shared access
to results

var results = RecipeTreeNode{
    NamaElemen: elements[targetID].Name,
    DibuatDari: make([]RecipeTreeNodeChild, 0),
    BanyakResep: 0,
    Parent:      nil,
}
for _, pair := range elements[targetID].FromPair {
    wg.Add(1)

    // ambil slot di semaphore
    sema <- struct{}{}

    go func(pair [2]int) {
        defer wg.Done()
        defer func() { <-sema }()
        // kembalikan slot
setelah selesai
        if IsValid(targetID, pair[0], pair[1]) {
            // log.Printf("[PARALLEL_DFS_DEBUG] Pair
(LeftID: %d, RightID: %d) VALID. Melanjutkan goroutine.", pair[0], pair[1])
            left := Dfs(pair[0], needFound, 1)
            right := Dfs(pair[1], needFound, 1)

            if left != nil && right != nil &&
left.BanyakResep > 0 && right.BanyakResep > 0 {
                childNode := RecipeTreeNodeChild{
                    Parent:      &results,
                    LeftChild:   left,
                    RightChild:  right,
                    LeftChildID: pair[0],
                    RightChildID: pair[1],
                }
                mu.Lock() // Lock before modifying
shared data
                if atomic.LoadInt32(&found) <
int32(needFound) {
                    results.DibuatDari =
append(results.DibuatDari, childNode)
                    results.BanyakResep +=

```

digunakan apabila pengguna memilih untuk melakukan pemilihan multi resep, agar dilakukan *multithreading*.

```

right.BanyakResep
    atomic.AddInt32(&found,
int32(right.BanyakResep))
}
mu.Unlock() // Unlock after mod

```

start.js:

Fungsi/Prosedur	Penjelasan
<pre> const fetchGraphData = async () =&gt; {   if (elements.length &gt; 0 &amp;&amp; (bfs    dfs    bidirectional) &amp;&amp; maxRecipe &gt; 0) {     setSearched(false);     setRecipeFound(false);     setGraphData([]);     setTime(0);     setNode(0);     try {       const algorithm = bfs ? 'bfs' : dfs ? 'dfs' : 'bidirectional';       const response = await fetch(`#\${process.env.NEXT_PUBLIC_API_URL}/solve-recipe?element=\${elements}&amp;algorithm=\${algorithm}&amp;count=\${maxRecipe}`);       const data = await response.json();       setGraphData(convertDataToNetwork(data));       setRecipeFound(data.recipes.length &gt; 0);       setSearched(true);       setTime(data.searchTimeMs);       setNode(data.nodesVisited);     }     catch (error) {       setSearched(true);     }   } } </pre>	<p>Fungsi ini dibuat untuk melakukan permintaan pencarian resep ke <i>endpoint API</i> pada sisi <i>backend</i>. Setelah respon diterima, data <i>recipe</i> di-parse menjadi JSON dan diolah menjadi data JSON yang merepresentasikan <i>nodes</i> dan <i>edges</i> agar dapat divisualisasikan.</p>

utils.js:

Fungsi/Prosedur	Penjelasan
-----------------	------------

<pre> export function convertDataToNetwork(data) {   console.log("Converting data to network:", data);   const recipes = data.recipes;   var networkData = {nodes: [], edges: []};   if (recipes &amp;&amp; recipes.length &gt; 0) {     const root = recipes[0];     resetNodeCounter();     recipesToNetworkHelper(root, networkData);   } else {     networkData.nodes.push({       id: 0,       label: data.elementName,       group: "target"     });   }   return networkData; } </pre>	<p>Data JSON hasil parsing dari respon API yang masih mengandung metadata dan daftar resep dikonversi menjadi data <i>nodes</i> dan <i>list</i> agar dapat divisualisasikan.</p>
<pre> function recipesToNetworkHelper(parent, networkData) {   const id = generateNodeId();   const group = parent.isBaseElement ?     parent.namaElemen.toLowerCase() : 'derivedElement';   const node = {     id: id,     label: parent.namaElemen,     group: id === 0 ? "target" : group,   }   networkData.nodes.push(node);   if (parent.dibuatDari) {     parent.dibuatDari.forEach(combinations =&gt; {       const addId = generateNodeId();       const addNode = {         id: addId,         label: "",         group: "add"       };       networkData.nodes.push(addNode);       combinations.forEach(element =&gt; {         const childId = recipesToNetworkHelper(element, networkData);         networkData.edges.push({           from: childId,           to: addId,           arrows: ""         });       });     });   } } </pre>	<p>Fungsi ini melakukan konversi data resep mentah menjadi data <i>nodes</i> dan <i>edges</i> secara rekursif.</p>

```

    });
    networkData.edges.push({
        from: addId,
        to: id,
        arrows: "middle",
    });
}
return id;
}

```

scraper.go:

Fungsi/Prosedur	Penjelasan
<pre>func normalizeElementName(name string) string {     return strings.Title(strings.ToLower(strings.TrimSpace(name))) }</pre>	Membersihkan dan menstandardisasi format nama elemen.
<pre>func isStandardBaseElement(name string) bool {     normalizedName := strings.Title(strings.ToLower(name))     for _, baseName := range standardBaseElements {         if normalizedName == baseName {             return true         }     }     return false }</pre>	Fungsi ini dibuat untuk mengetahui apakah suatu elemen merupakan elemen dasar atau bukan.
<pre>func max(a, b int) int {     if a &gt; b {         return a     }     return b }</pre>	Fungsi ini dibuat untuk mencari nilai maksimum dari dua bilangan bulat.
<pre>func isRecipeValidInternal(ing1ID int, ing2ID int, resultID int, elementMapByID map[int]*Element) bool {     ing1Elem, ok1 := elementMapByID[ing1ID]     ing2Elem, ok2 := elementMapByID[ing2ID]     resultElem, okResult := elementMapByID[resultID]</pre>	Fungsi ini memeriksa apakah semua ID (bahan 1, bahan 2, hasil) ada dalam

<pre> if !ok1    !ok2    !okResult {     return false }  if forbiddenIngredientNames[ing1Elem.Name] {     return false } if forbiddenIngredientNames[ing2Elem.Name] {     return false }  // check if tier is valid (not -1, which indicates an // unprocessed or invalid tier) if ing1Elem.Tier == -1    ing2Elem.Tier == -1    resultElem.Tier == -1 {     return false }  return true } </pre>	<p>elementMapByID.</p> <p>Jika salah satu tidak ada, resep tidak valid.</p> <p>Memeriksa apakah <i>Tier</i> dari semua elemen yang terlibat (bahan 1, bahan 2, hasil) bukan -1. <i>Tier</i> -1 mengindikasikan elemen tersebut belum diproses atau <i>tier</i>-nya tidak valid. Jika semua pemeriksaan lolos, kembalikan <i>true</i>.</p>
---	---

<pre> func fetchDocument(url string) (*goquery.Document, error) {     res, err := http.Get(url)     if err != nil {         return nil, fmt.Errorf("failed GET request to %s: %w", url, err)     }     defer res.Body.Close()      if res.StatusCode != 200 {         return nil, fmt.Errorf("status code not OK: %d %s", res.StatusCode, res.Status)     }      doc, err := goquery.NewDocumentFromReader(res.Body)     if err != nil {         return nil, fmt.Errorf("failed to parse HTML: %w", err)     }     return doc, nil } </pre>	<p>Fungsi ini melakukan HTTP GET <i>request</i> ke url dan menangani <i>error</i> jika request gagal. Fungsi ini juga memeriksa status code respons. Jika bukan 200 (OK), kembalikan <i>error</i>. Kemudian, fungsi ini membuat objek goquery.</p> <p>Goquery adalah <i>library</i> untuk memanipulasi dan <i>query</i> dokumen HTML di Go.</p>
---	---

```

func extractElementTiers(doc *goquery.Document)
map[string]int {
    elementTiers := make(map[string]int)
    for _, baseName := range standardBaseElements {
        elementTiers[normalizeElementName(baseName)] = 0
    }
}

doc.Find("div.mw-parser-output > h2,
div.mw-parser-output > h3").Each(func(_ int, headingSel
*goquery.Selection) {
    headlineSpan :=
headingSel.Find("span.mw-headline")
    if headlineSpan.Length() == 0 {
        return
    }
    headlineID, idExists := headlineSpan.Attr("id")
    if !idExists {
        return
    }

    currentTierNum := -1
    if strings.HasPrefix(headlineID, "Tier_") &&
strings.HasSuffix(headlineID, "_elements") {
        tierStr :=
strings.TrimSuffix(strings.TrimPrefix(headlineID,
"Tier_"), "_elements")
        if val, errConv := strconv.Atoi(tierStr);
errConv == nil {
            currentTierNum = val
        }
    } else if headlineID == "Starting_elements" {
        currentTierNum = 0
    }

    if currentTierNum != -1 {
        node := headingSel.Next()
        foundTableForThisTier := false
        for node.Length() > 0 &&
!foundTableForThisTier {
            if node.Is("h2, h3") { // stop if hit the
next tier heading

```

Fungsi ini bertugas untuk memindai dokumen HTML yang telah diunduh guna mengidentifikasi dan mengekstrak informasi *tier* untuk setiap elemen. Fungsi ini mencari judul-judul bagian yang menandakan tingkatan elemen (misalnya, “Tier 1 elements”, “Starting elements”), kemudian membaca nama-nama elemen yang tercantum di bawah judul tersebut dalam tabel-tabel terkait.

```

        break
    }
    var tableToProcess *goquery.Selection
    if node.Is("table.list-table") {
        tableToProcess = node
    } else if node.Is("div") {
        tableToProcess =
node.Find("table.list-table").First()
    }

    if tableToProcess != nil &&
tableToProcess.Length() > 0 {
        foundTableForThisTier = true
        tableToProcess.Find("tbody
tr").Each(func(i int, rowSelection *goquery.Selection) {
            tds := rowSelection.Find("td")
            if tds.Length() < 1 {
                return
            }

            // attempt to get element name
from the first link
            resultElementNode :=
tds.Eq(0).Find("a[href^='/wiki/']").First()
            resultElementName :=
strings.TrimSpace(resultElementNode.Text())

            // fallback if no link or link
text is empty
            if resultElementName == "" {
                clonedTd := tds.Eq(0).Clone()
                clonedTd.Find("span, script,
style, sup, br, img, a.image, div.gallerytext").Remove()
                resultElementName =
strings.TrimSpace(clonedTd.Text())
                if
strings.Contains(resultElementName, "\n") {
                    resultElementName =
strings.TrimSpace(strings.Split(resultElementName,
"\n")[0])
                }
            }
            resultElementName =
normalizeElementName(resultElementName)

```

```
        if resultElementName != "" {
            isFilteredOut := false

        tds.Eq(0).Find("a[title*='Myths and Monsters Pack'],
img[alt*='Myths and Monsters Pack'],
span[class*='pack-icon']]").EachWithBreak(func(_ int, s
*goquery.Selection) bool {
                if
!isStandardBaseElement(resultElementName) {
                    isFilteredOut = true
                    return false
                }
                return true
            })
            if isFilteredOut {
                return
            }
            if tds.Length() > 1 {
                if
tds.Eq(1).Find("a[href='/wiki/Elements_(Myths_and_Monster
s)']").Length() > 0 {
                    isFilteredOut = true
                }
            }
            if isFilteredOut {
                return
            }

            // assign tier if not already
assigned or if previous assignment was placeholder (-1)
            if _, exists :=
elementTiers[resultElementName]; !exists ||
elementTiers[resultElementName] == -1 {

elementTiers[resultElementName] = currentTierNum
            }
        }
    }
}
})
```

<pre>         return elementTiers     } </pre>	
<pre> func extractRawRecipesAndElements(doc *goquery.Document) ([]RawRecipeEntry, map[string]bool, []string) {     var rawRecipes []RawRecipeEntry     validResultElementsFromPage := make(map[string]bool)     var tempOrderedResultNames []string     isNameInTempOrderedList := make(map[string]bool)      doc.Find("div.mw-parser-output table.list-table tbody tr").Each(func(i int, rowSelection *goquery.Selection) {         tds := rowSelection.Find("td")         if tds.Length() &lt; 1 {             return         }          // extract result element name (similar logic to tier extraction)         resultElementNode :=         tds.Eq(0).Find("a[href^='/wiki/]").First()         resultElementName :=         strings.TrimSpace(resultElementNode.Text())         if resultElementName == "" {             clonedTd := tds.Eq(0).Clone()             clonedTd.Find("span, script, style, sup, br, img, a.image, div.gallerytext").Remove()             resultElementName =             strings.TrimSpace(clonedTd.Text())             if strings.Contains(resultElementName, "\n") {                 resultElementName =                 strings.TrimSpace(strings.Split(resultElementName,                 "\n")[0])             }         }         if resultElementName == "" {             return         }         resultElementName =         normalizeElementName(resultElementName)          // filter out elements from specific packs         if tds.Length() &gt; 1 {             if </pre>	<p>Fungsi ini menjelajahi tabel-tabel pada halaman <i>web</i> untuk menemukan baris-baris yang mendefinisikan elemen hasil beserta bahan-bahan pembuatnya, melakukan normalisasi semua nama elemen, memastikan setiap resep memiliki minimal dua bahan, dan mengurutkan bahan, kemudian mengembalikan daftar resep mentah (RawRecipeEntry).</p>

```

        tds.Eq(1).Find("a[href='/wiki/Elements_(Myths_and_Monster
s)']").Length() > 0 {
            return
        }
    }
    var isVisuallyMarkedAsPack = false
    tds.Eq(0).Find("a[title*='Myths and Monsters
Pack'], img[alt*='Myths and Monsters Pack'],
span[class*='pack-icon']").Each(func(_ int, s
*goquery.Selection) {
        if !isStandardBaseElement(resultElementName)
{
            isVisuallyMarkedAsPack = true
        }
})
if isVisuallyMarkedAsPack {
    return
}

validResultElementsFromPage[resultElementName] =
true
if !isNameInTempOrderedList[resultElementName] {
    tempOrderedResultNames =
append(tempOrderedResultNames, resultElementName)
    isNameInTempOrderedList[resultElementName] =
true
}

if tds.Length() > 1 {
    tds.Eq(1).Find("ul > li").Each(func(j int,
recipeLi *goquery.Selection) {
        ingredientLinks :=
recipeLi.Find("a[href^='/wiki/']")
        if ingredientLinks.Length() >= 2 { // a
recipe needs at least two ingredients
            ing1 :=
normalizeElementName(ingredientLinks.Eq(0).Text())
            ing2 :=
normalizeElementName(ingredientLinks.Eq(1).Text())
            if ing1 != "" && ing2 != "" {
                sortedIng := []string{ing1, ing2}
                sort.Strings(sortedIng)
                rawRecipes = append(rawRecipes,
RawRecipeEntry{resultElementName, sortedIng[0],

```

```

sortedIng[1]})

        }

    })

}

// filter rawRecipes to ensure their result elements
// are considered valid
var finalRawRecipes []RawRecipeEntry
for _, rr := range rawRecipes {
    if validResultElementsFromPage[rr.ResultElement]
{
        finalRawRecipes = append(finalRawRecipes, rr)
    }
}
return finalRawRecipes, validResultElementsFromPage,
tempOrderedResultNames
}

```

```

func buildRecipeTreeNodes(rawRecipes []RawRecipeEntry,
allElementNamesOnPage []string) map[string]*model.RecipeTreeNodeTier {
    allNodes := make(map[string]*model.RecipeTreeNodeTier)

    // ensure all unique element names from page (and
    // base elements) have a node
    elementNameSet := make(map[string]bool)
    for _, name := range allElementNamesOnPage {
        if _, exists := allNodes[name]; !exists {
            allNodes[name] =
&model.RecipeTreeNodeTier{NamaElemen: name}
        }
        elementNameSet[name] = true
    }
    for _, baseName := range standardBaseElements {
        normBaseName := normalizeElementName(baseName)
        if _, exists := allNodes[normBaseName]; !exists {
            allNodes[normBaseName] =
&model.RecipeTreeNodeTier{NamaElemen: normBaseName}
        }
        elementNameSet[normBaseName] = true
    }
}

```

Fungsi ini bertujuan untuk membangun struktur data awal berupa graf yang merepresentasikan hubungan antar elemen sehingga pada akhirnya membuat struktur node model.RecipeTreeNodeTier.

```

// populate the DibuatDari field for each node
for _, rr := range rawRecipes {
    resNode, rExists := allNodes[rr.ResultElement]
    ing1Node, i1Exists := allNodes[rr.Ingredient1]
    ing2Node, i2Exists := allNodes[rr.Ingredient2]

    if rExists && i1Exists && i2Exists {
        if isStandardBaseElement(resNode.NamaElemen)
{ // base elements are not made from anything
            continue
        }
        // check if this combination already exists
        to avoid duplicates
        comboExists := false
        for _, combo := range resNode.DibuatDari {
            if (combo[0].NamaElemen == rr.Ingredient1
&& combo[1].NamaElemen == rr.Ingredient2) ||
                (combo[0].NamaElemen ==
rr.Ingredient2 && combo[1].NamaElemen == rr.Ingredient1)
{
                comboExists = true
                break
            }
        }
        if !comboExists {
            resNode.DibuatDari =
append(resNode.DibuatDari,
[2]*model.RecipeTreeNodeTier{ing1Node, ing2Node})
        }
    }
}
return allNodes
}

```

```

func assignIDsAndCreateElements(allNodes
map[string]*model.RecipeTreeNodeTier, elementTiersFromWeb
map[string]int, allElementNamesFromExtraction []string)
([[]*Element, map[string]int, map[int]*Element]) {
    type elementInfoForIDSorter struct {
        Name string
        Tier int
    }

    var elementsForIDSorting []elementInfoForIDSorter

```

Fungsi ini bertujuan untuk memberikan ID unik terurut kepada setiap elemen dan membuat struktur Element final untuk digunakan oleh

```

uniqueElementNames := make(map[string]bool)
for _, name := range allElementNamesFromExtraction {
    uniqueElementNames[name] = true
}

for _, baseName := range standardBaseElements {

uniqueElementNames[normalizeElementName(baseName)] = true
}

for name := range uniqueElementNames {
    currentTier := -1 // default to unassigned tier
    if isStandardBaseElement(name) {
        currentTier = 0
    } else if tier, found :=
elementTiersFromWeb[name]; found {
        currentTier = tier
    }
    elementsForIDSorting =
append(elementsForIDSorting, elementInfoForIDSorter{Name:
name, Tier: currentTier})
}

sort.Slice(elementsForIDSorting, func(i, j int) bool
{
    elI := elementsForIDSorting[i]
    elJ := elementsForIDSorting[j]
    if elI.Tier != elJ.Tier {
        if elI.Tier == -1 {
            return false
        }
        if elJ.Tier == -1 {
            return true
        }
        return elI.Tier < elJ.Tier
    }
    return elI.Name < elJ.Name
})

nameToID := make(map[string]int)
elementMapByID := make(map[int]*Element)
var orderedElements []*Element
currentIDCounter := 0

```

algoritma DFS/BFS.

```

for _, sortedElInfo := range elementsForIDSORTING {
    el := &Element{
        ID:      currentIDCounter,
        Name:    sortedElInfo.Name,
        Tier:    sortedElInfo.Tier,
        FromPair: make([][]int, 0),
        CanMake:  make([]int, 0),
    }
    nameToID[el.Name] = el.ID
    elementMapByID[el.ID] = el
    orderedElements = append(orderedElements, el)
    currentIDCounter++
}
return orderedElements, nameToID, elementMapByID
}

```

```

func populateElementRelationships(orderedElements
[*Element, allNodes
map[string]*model.RecipeTreeNodeTier, nameToID
map[string]int, elementMapByID map[int]*Element) {
    canMakeTemp := make(map[int]map[int]bool)

    for _, currentElement := range orderedElements {
        resultNode, nodeExists :=
allNodes[currentElement.Name]
        if !nodeExists ||
isStandardBaseElement(currentElement.Name) ||
resultNode.DibuatDari == nil ||
len(resultNode.DibuatDari) == 0 {
            continue
        }

        tempValidFromPairs := [][]int{}
        processedIngredientPairKeys :=
make(map[string]bool)

        for _, recipePairNodes := range
resultNode.DibuatDari {
            ing1Node := recipePairNodes[0]
            ing2Node := recipePairNodes[1]
            if ing1Node == nil || ing2Node == nil {
                continue
            }

            ing1ID, ok1 := nameToID[ing1Node>NamaElemen]

```

Fungsi ini bertujuan untuk melengkapi informasi hubungan antar elemen pada setiap *struct Element* yang sudah dibuat, yaitu mengisi *field* FromPair dan CanMake. Fungsi ini melakukan iterasi pada setiap elemen, mengambil resepnya dari struktur allNodes, mengkonversi nama bahan menjadi ID menggunakan nameToID, memvalidasi resep, dan jika valid,

```

        ing2ID, ok2 := nameToID[ing2Node.NamaElemen]

        if ok1 && ok2 {
            currentPairIDs := []int{ing1ID, ing2ID}
            if currentPairIDs[0] > currentPairIDs[1]
{
                currentPairIDs[0], currentPairIDs[1]
= currentPairIDs[1], currentPairIDs[0]
            }
            pairKey := fmt.Sprintf("%d-%d",
currentPairIDs[0], currentPairIDs[1])

            if !processedIngredientPairKeys[pairKey]
{
                processedIngredientPairKeys[pairKey]
= true
                if isRecipeValidInternal(ing1ID,
ing2ID, currentElement.ID, elementMapByID) {
                    tempValidFromPairs =
append(tempValidFromPairs, currentPairIDs)

                    if _, exists :=
canMakeTemp[ing1ID]; !exists {
                        canMakeTemp[ing1ID] =
make(map[int]bool)
                    }
                }

                canMakeTemp[ing1ID][currentElement.ID] = true

                if ing1ID != ing2ID {
                    if _, exists :=
canMakeTemp[ing2ID]; !exists {
                        canMakeTemp[ing2ID] =
make(map[int]bool)
                    }
                }

                canMakeTemp[ing2ID][currentElement.ID] = true
            }
        }
    }
}
currentElement.FromPair = tempValidFromPairs
}

```

menambahkan pasangan ID bahan ke FromPair elemen hasil, serta membangun data sementara untuk melengkapi *field* CanMake pada elemen-elemen bahan.

```

for ingID, producesMap := range canMakeTemp {
    if ingElement, ok := elementMapByID[ingID]; ok {
        for prodID := range producesMap {
            ingElement.CanMake =
append(ingElement.CanMake, prodID)
        }
        sort.Ints(ingElement.CanMake)
    }
}

func FetchAndProcessData() ([]*Element, error) {
    log.Println("starting scraping process from:", targetURL)

    doc, err := fetchDocument(targetURL)
    if err != nil {
        return nil, err
    }

    elementTiersFromWeb := extractElementTiers(doc)

    rawRecipes, _, tempOrderedResultNames :=
extractRawRecipesAndElements(doc)

    allNodes := buildRecipeTreeNodes(rawRecipes,
tempOrderedResultNames)

    orderedElements, nameToID, elementMapByID :=
assignIDsAndCreateElements(allNodes, elementTiersFromWeb,
tempOrderedResultNames)

    populateElementRelationships(orderedElements,
allNodes, nameToID, elementMapByID)

    var finalFilteredElements []*Element
    for _, el := range orderedElements {
        isBase := isStandardBaseElement(el.Name)
        hasValidRecipes := len(el.FromPair) > 0

        if isBase {
            finalFilteredElements =
append(finalFilteredElements, el)
        }
    }
}

```

Fungsi ini bertujuan untuk mengunduh dan mem-*parsing* halaman HTML target, dilanjutkan dengan memanggil *extractElementTiers* untuk mendapatkan informasi *tier*, *extractRawRecipesAndElements* untuk resep mentah dan nama elemen, *buildRecipeTreeNodes* untuk struktur awal, *assignIDsAndCreateElements* untuk membuat objek Element dengan ID, dan

<pre> } else if hasValidRecipes &amp;&amp; el.Tier != -1 {     finalFilteredElements = append(finalFilteredElements, el) } log.Println("all data processing in scraper complete. data will be returned directly.") return finalFilteredElements, nil } </pre>	<p>populateElement Relationships untuk melengkapi hubungan resep.</p>
<pre> func GetProcessedElements() []*Element {     log.Println("GetProcessedElements called: will perform a new fetch and process data.")     elements, err := FetchAndProcessData()     if err != nil {         log.Printf("error during FetchAndProcessData called from GetProcessedElements: %v. returning empty slice.", err)         return []*Element{}     }     return elements } </pre>	<p>Fungsi ini berfungsi untuk mendapatkan data elemen yang sudah diproses. Tujuannya adalah untuk menyediakan data yang paling baru, dan akan mengembalikan <i>slice</i> elemen kosong jika terjadi kesalahan selama proses pengambilan dan pemrosesan data tersebut.</p>

main.go:

Fungsi/Prosedur	Penjelasan
<pre> func initializeElementMaps(elements []*scraper.Element) {     elementNameToID = make(map[string]int)     elementIDToName = make(map[int]string)     for _, el := range elements {         normalizedName := strings.ToLower(el.Name)         elementNameToID[normalizedName] = el.ID         elementIDToName[el.ID] = el.Name     } } </pre>	<p>Fungsi ini bertujuan untuk menginisialisasi dan mengisi dua <i>map global</i>, yaitu</p>

<pre>         }     } } </pre>	<p>elementNameToID dan elementIDToName, berdasarkan data elemen yang diterima sebagai argumen.</p>
<pre> func main() {     log.SetFlags(log.LstdFlags   log.Lshortfile)     var err error     processedGraphData, err = scraper.FetchAndProcessData()     if err != nil {         log.Fatalf("error in scraping: %v", err)     }     if processedGraphData == nil    len(processedGraphData) == 0 {         log.Fatalf("invalid or empty data.")     }      initializeElementMaps(processedGraphData)  algorithm.InitializeAlgorithmElements(scraper.GetProcesse dElements())      solveRecipe := api.NewSolveHandler(elementNameToID, elementIDToName)          // router mux for handling API requests          //      @Summary          Get All Processed Graph Data         //      @Description For testing purposes, this endpoint returns all processed graph data in JSON format         //      @Tags            Graph Data         //      @Produce          json         //      @Success          200  {array} scraper.Element "Array element data in JSON format"         //      @Failure         500  {string}  string         //      @Description     "Error if graph data is not ready or invalid"         //      @Router           /graph-data [get] </pre>	<p>Fungsi main pertama-tama menginisialisasi <i>logging</i>, kemudian mengambil dan memproses data graf menggunakan <i>scraper</i>. Setelah data berhasil diambil dan divalidasi, fungsi ini memanggil <i>initializeElementMaps</i>. Selanjutnya, fungsi ini menyiapkan router HTTP menggunakan <i>gorilla/mux</i>, menambahkan <i>middleware</i> untuk menangani CORS, dan mendefinisikan rute-rute API. Terakhir, fungsi ini menjalankan <i>server</i></p>

```

r := mux.NewRouter()
r.Use(func(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w
http.ResponseWriter, r *http.Request) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "GET,
POST, PUT, DELETE, OPTIONS")

    w.Header().Set("Access-Control-Allow-Headers",
"Content-Type, Authorization")

        if r.Method == "OPTIONS" {
            w.WriteHeader(http.StatusOK)
            return
        }

        next.ServeHTTP(w, r)
    })
})

r.PathPrefix("/swagger/").Handler(httpSwagger.WrapHandler
)
    r.HandleFunc("/graph-data",
serveGraphDataHandler).Methods(http.MethodGet)

    r.Handle("/solve-recipe",
solveRecipe).Methods(http.MethodGet)

    // menyajikan API Swagger
    port := "8080"
    log.Printf("API server ready in
http://localhost:%s\n", port)
    log.Printf("-> access /graph-data to see graph data
in JSON.")
    log.Printf("-> access /solve-recipe to use solver
for a recipe.")
    log.Printf("-> access /swagger/index.html for
interactive API documentation.")
    if err := http.ListenAndServe(": "+port, r); err !=
nil {
        log.Fatalf("error: %v", err)
    }
}

```

HTTP pada port 8080 yang menerima dan melayani permintaan ke *endpoint* yang telah ditentukan.

<pre> }  func serveGraphDataHandler(w http.ResponseWriter, r *http.Request) {     if processedGraphData == nil    len(processedGraphData) == 0 {         log.Println("/graph-data: processedGraphData is not ready or empty.")         http.Error(w, "graph data is not ready.", http.StatusInternalServerError)         return     }      log.Printf("/graph-data: %d processed.", len(processedGraphData))      w.Header().Set("Content-Type", "application/json")     if err := json.NewEncoder(w).Encode(processedGraphData); err != nil {         log.Printf("error encoding processed graph data ke JSON: %v", err)         http.Error(w, "error in formatting graph data.", http.StatusInternalServerError)     } } </pre>	<p>Fungsi ini adalah <i>handler</i> HTTP yang bertanggung jawab untuk melayani permintaan GET ke <i>endpoint</i> /graph-data. Ketika permintaan diterima, fungsi ini pertama-tama memeriksa apakah data graf global (<i>processedGraphData</i>) sudah tersedia dan tidak kosong. Jika data belum siap, ia akan mengirimkan respons kesalahan HTTP 500 (<i>Internal Server Error</i>). Jika data tersedia, fungsi ini akan mengatur header Content-Type respons menjadi application/json dan kemudian melakukan <i>encode</i> data graf tersebut ke dalam format JSON, yang kemudian dikirimkan</p>
---	--

	kembali ke klien sebagai isi respons.
--	---------------------------------------

handler.go:

Fungsi/Prosedur	Penjelasan
<pre>func transformRecipeTree(node *algorithm.RecipeTreeNode) RecipePath {     if node == nil {         return RecipePath{}     }     isBase := algorithm.IsBasicElement(*node)     recipePath := RecipePath{         NamaElemen:    node.NamaElemen,         IsBaseElement: isBase,     }     if !isBase &amp;&amp; len(node.DibuatDari) &gt; 0 {         recipePath.DibuatDari = make([][]RecipePath, 0, len(node.DibuatDari))         for _, childNodePair := range node.DibuatDari {             if childNodePair.LeftChild != nil &amp;&amp; childNodePair.RightChild != nil {                 pair := []RecipePath{                     transformRecipeTree(childNodePair.LeftChild),                     transformRecipeTree(childNodePair.RightChild),                 }                 recipePath.DibuatDari = append(recipePath.DibuatDari, pair)             }         }     }     return recipePath }</pre>	<p>Fungsi ini mengubah struktur data <i>tree</i> resep internal yang dihasilkan oleh <i>algorithm</i> menjadi struktur RecipePath yang sesuai untuk respons API dan dapat di-<i>encode</i> ke JSON. Proses transformasi ini bersifat rekursif, dimana ia akan memanggil dirinya sendiri untuk setiap <i>child</i> dalam pohon resep hingga semua <i>node</i> telah dikonversi.</p>
<pre>func countNodesInTree(node *algorithm.RecipeTreeNode) int {     if node == nil {         return 0     }     visited := make(map[string]bool)</pre>	<p>Fungsi ini adalah pembungkus yang memulai proses penghitungan jumlah</p>

<pre>         return countNodesRecursive(node, visited)     } </pre>	<p>total <i>node</i> dalam sebuah pohon resep yang diberikan dengan memanggil fungsi rekursif countNodesRecursive untuk melakukan penghitungan sebenarnya.</p>
<pre> func countNodesRecursive(node *algorithm.RecipeTreeNode, visited map[string]bool) int {     if node == nil {         return 0     }      count := 1     visited[node.NamaElemen] = true      for _, childPair := range node.DibuatDari {         count += countNodesRecursive(childPair.LeftChild, visited)         count += countNodesRecursive(childPair.RightChild, visited)     }     return count } </pre>	<p>Fungsi rekursif ini melakukan penghitungan aktual jumlah <i>node</i> dalam pohon resep. Untuk setiap <i>node</i> yang tidak nil, ia akan menambahkan 1 ke hitungan (untuk <i>node</i> itu sendiri) dan kemudian secara rekursif memanggil dirinya sendiri untuk setiap anak (pasangan bahan) dalam <i>node</i>.</p>
<pre> func NewSolveHandler(nameToID map[string]int, idToName map[int]string) *SolveHandler {     return &amp;SolveHandler{         ElementNameToID: nameToID,         ElementIDToName: idToName,     } } </pre>	<p>Fungsi konstruktor yang bertanggung jawab untuk membuat dan menginisialisasi</p>

}	<p><i>instance</i> baru dari <code>SolveHandler</code>. Fungsi ini menerima peta <code>nameToID</code> dan <code>idToName</code> sebagai argumen dan menyimpannya dalam <code>field</code> yang sesuai pada <code>struct</code> <code>SolveHandler</code> yang baru dibuat.</p>
<pre>func (sh *SolveHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {     queryParams := r.URL.Query()     elementNameQuery := strings.TrimSpace(queryParams.Get("element"))     algo := strings.ToLower(strings.TrimSpace(queryParams.Get("algorithm")))     countStr := strings.TrimSpace(queryParams.Get("count"))     mode := strings.ToLower(strings.TrimSpace(queryParams.Get("mode"))) )      if elementNameQuery == "" {         http.Error(w, "Query parameter 'element' is required.", http.StatusBadRequest)         return     }      if algo == "" {         algo = "dfs"     }     if algo != "dfs" &amp;&amp; algo != "bfs" {         http.Error(w, "Invalid 'algorithm' parameter. Use 'dfs' or 'bfs'.", http.StatusBadRequest)         return     } }</pre>	<p>Metode ini adalah inti dari <i>endpoint</i> API <code>/solve-recipe</code> yang menangani seluruh logika permintaan HTTP untuk mencari resep. Fungsi ini yang mengirimkan seluruh informasi hasil pencarian resep sebagai respons JSON dalam bentuk <code>RecipeSolution</code>, termasuk penanganan kasus jika elemen tidak ditemukan atau tidak ada resep yang berhasil ditemukan.</p>

```

    }

    var recipeCount int = 1
    if countStr != "" {
        var err error
        recipeCount, err = strconv.Atoi(countStr)
        if err != nil || recipeCount < 1 {
            http.Error(w, "Invalid 'count' parameter.
Must be a positive integer.", http.StatusBadRequest)
            recipeCount = 1
        }
    }

    if mode == "" {
        mode = "shortest"
    }

    log.Printf("Handler /solve-recipe: element='%s',
algorithm=%s', count=%d, mode=%s'", elementNameQuery,
algo, recipeCount, mode)

    normalizedQueryName :=
strings.ToLower(elementNameQuery)
    startID, ok :=
sh.ElementNameToID[normalizedQueryName]

    elementNameForOutput := elementNameQuery
    if !ok {
        http.Error(w, fmt.Sprintf("Element '%s' not found
in our database.", elementNameQuery),
http.StatusNotFound)
        return
    }
    if officialName, nameOk :=
sh.ElementIDToName[startID]; nameOk {
        elementNameForOutput = officialName
    }

    var resultTree *algorithm.RecipeTreeNode
    var nodesVisitedByAlgo int

    startTime := time.Now()
    multParam := 1 // placeholder

```

```

    if algo == "dfs" {
        log.Printf("Calling DFS for element ID %d (%s),
needFound: %d", startID, elementNameForOutput,
recipeCount)
        if recipeCount == 1 {
            resultTree = algorithm.Dfs(startID,
recipeCount, multParam)
        } else {
            resultTree = algorithm.ParallelDFS(startID,
recipeCount, 2)
        }
    } else { // bfs
        log.Printf("Calling BFS for element ID %d (%s),
needFound: %d", startID, elementNameForOutput,
recipeCount)
        if recipeCount == 1 {
            resultTree = algorithm.Bfs(startID,
recipeCount, multParam)
        } else {
            resultTree = algorithm.ParallelBfs(startID,
recipeCount, 2)
        }
    }

    searchTimeMs :=
float64(time.Since(startTime).Microseconds()) / 1000.0

    if resultTree == nil || (resultTree.BanyakResep == 0
&& !algorithm.IsBasicElement(*resultTree)) {
        isActuallyBase := false
        if resultTree != nil {
            isActuallyBase =
algorithm.IsBasicElement(*resultTree)
        }
        if !isActuallyBase {
            log.Printf("Algorithm did not find a valid
recipe tree for '%s' (ID: %d).", elementNameForOutput,
startID)
            solution := RecipeSolution{
                ElementName: elementNameForOutput,
               SearchParams: SearchParams{Algorithm:
algo, Count: recipeCount, Mode: mode},
                Found:         false,
                SearchTimeMs: searchTimeMs,
            }
        }
    }
}

```

```

        NodesVisited: nodesVisitedByAlgo,
        Recipes:      []RecipePath{},
    }
    w.Header().Set("Content-Type",
"application/json")
    json.NewEncoder(w).Encode(solution)
    return
}
}

var recipePaths []RecipePath
nodesVisitedInResultTree := 0
if resultTree != nil {
    recipePaths = append(recipePaths,
transformRecipeTree(resultTree))
    nodesVisitedInResultTree =
countNodesInTree(resultTree)
}

finalNodesVisited := nodesVisitedByAlgo
if finalNodesVisited == 0 && nodesVisitedInResultTree
> 0 {
    finalNodesVisited = nodesVisitedInResultTree
}

solution := RecipeSolution{
    ElementName: elementNameForOutput,
   SearchParams: SearchParams{Algorithm: algo,
Count: recipeCount, Mode: mode},
    Found:      true,
    SearchTimeMs: searchTimeMs,
    NodesVisited: finalNodesVisited,
    Recipes:     recipePaths,
}

w.Header().Set("Content-Type", "application/json")
if err := json.NewEncoder(w).Encode(solution); err !=
nil {
    log.Printf("Error encoding solution to JSON for
element %s: %v", elementNameForOutput, err)
    http.Error(w, "Failed to format solution data.",
http.StatusInternalServerError)
}
}

```

### 4.3 Tata Cara Penggunaan Program

#### 1. Membuka aplikasi web

Aplikasi dapat diakses melalui tautan [tubes2-fe-ian.vercel.app](https://tubes2-fe-ian.vercel.app). Tautan ini mengarahkan pengguna ke sisi *frontend* aplikasi.

#### 2. Inisialisasi Backend

Program `main.go` dijalankan dengan menuliskan *command* `go run src/cmd/main.go`. *Command* ini akan melakukan *scraping* pada Wiki Little Alchemy 2, kemudian menjalankan rangkaian *Backend* untuk menerima permintaan dari *Frontend* dan mendapatkan hasil dari algoritma DFS/BFS yang diformat dalam JSON untuk di-*fetch* oleh *Frontend*.

#### 3. Permintaan Pencarian Resep Melalui *Frontend*

Pengguna memasukkan nama elemen yang ingin dicari resepnya pada komponen *input* teks lalu memilih algoritma yang ingin digunakan (BFS atau DFS). Kemudian, pengguna dapat memilih untuk memberikan jumlah resep yang ingin dicari atau tidak dengan menekan tombol *toggle* “MULTIPLE RECIPE”. Jika *toggle* dinyalakan, maka pengguna harus memasukkan jumlah resep yang ingin dicari pada input angka di bawahnya. Ketika kriteria sudah diatur, pengguna dapat menekan tombol dengan ikon kaca pembesar di sebelah kanan layar untuk melakukan pencarian resep.

#### 4. Tampilan Akhir pada *Frontend*

Setelah *frontend* menerima respon API dari *backend*, data respon diolah dan ditampilkan di layar sebagai sebuah pohon. Elemen yang diberikan pengguna berada pada akar pohon di paling atas dan elemen-elemen anaknya membentuk resep yang ingin dicari.

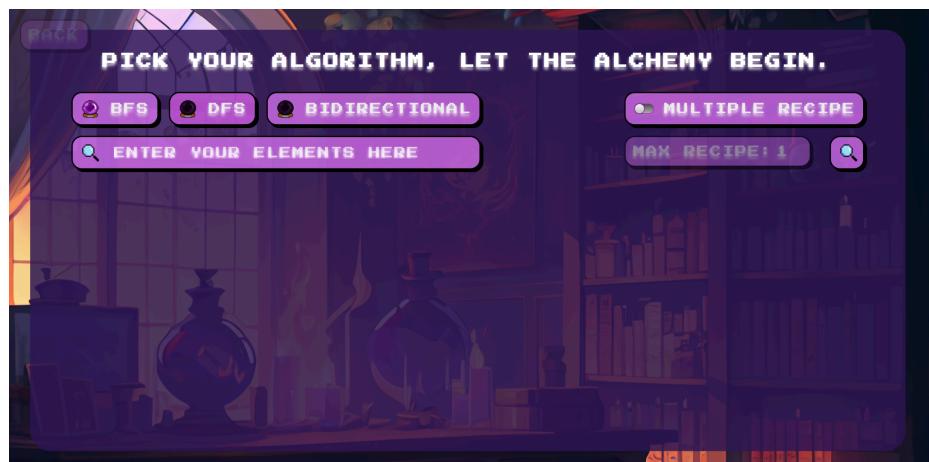
Fitur	Screenshot
-------	------------

Menu Home yang ditampilkan ketika pengguna pertama kali membuka aplikasi



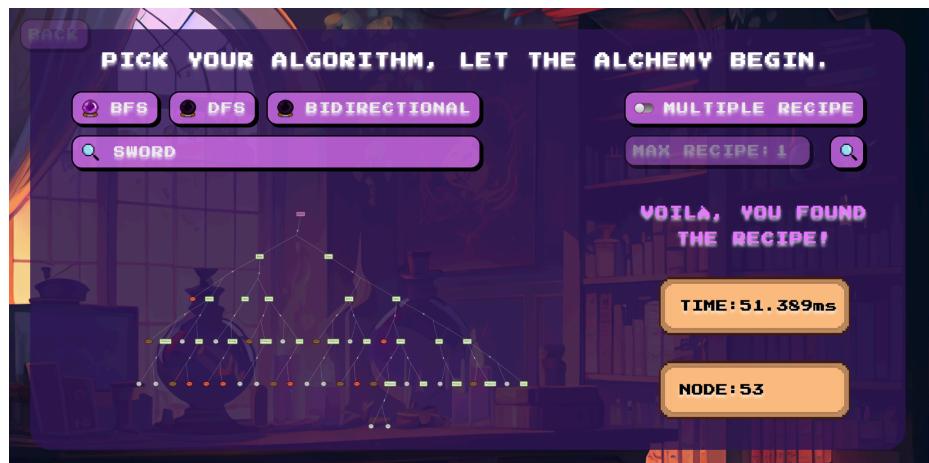
Gambar 7. Interface Frontend

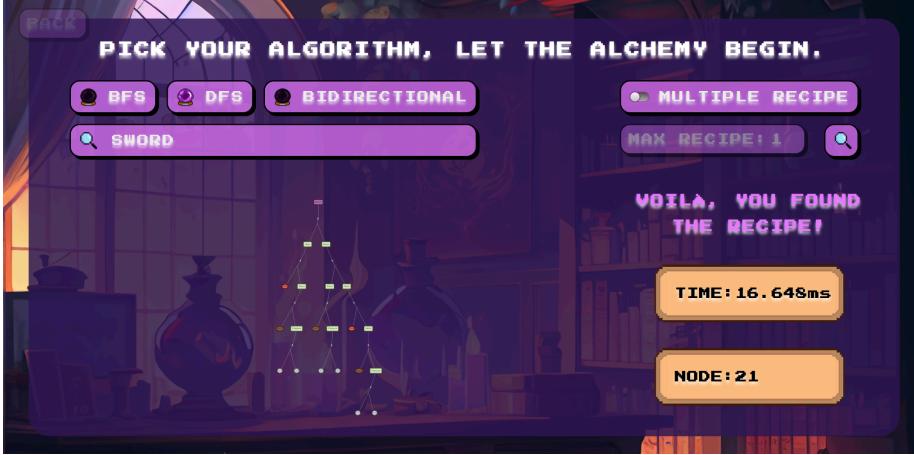
Tampilan awal menu Start



Gambar 8. Interface Frontend

Pencarian elemen dengan metode BFS



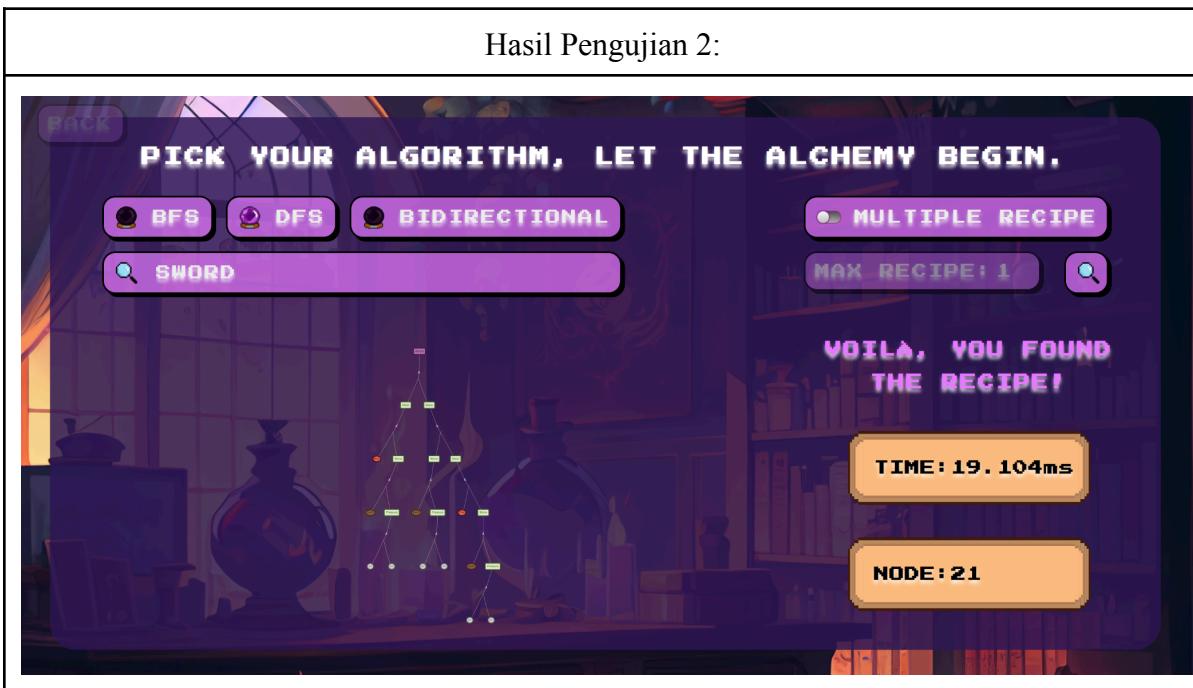
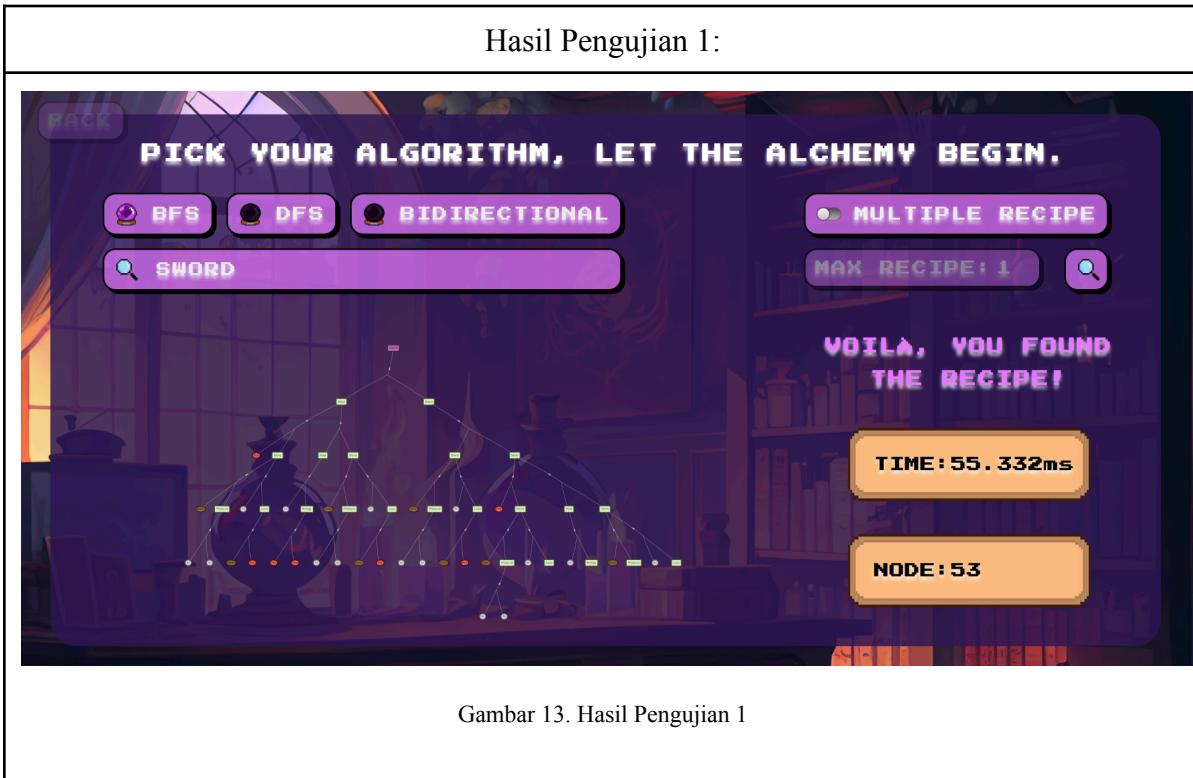
	<p>Gambar 9. Interface Frontend</p> 
Pencarian elemen dengan metode DFS	<p>Gambar 10. Interface Frontend</p> 
Pencarian elemen dengan memberikan jumlah resep	<p>Gambar 11. Interface Frontend</p> 

Menu About yang menampilkan identitas kelompok



Gambar 12. Interface *Frontend*

#### 4.4 Hasil Pengujian



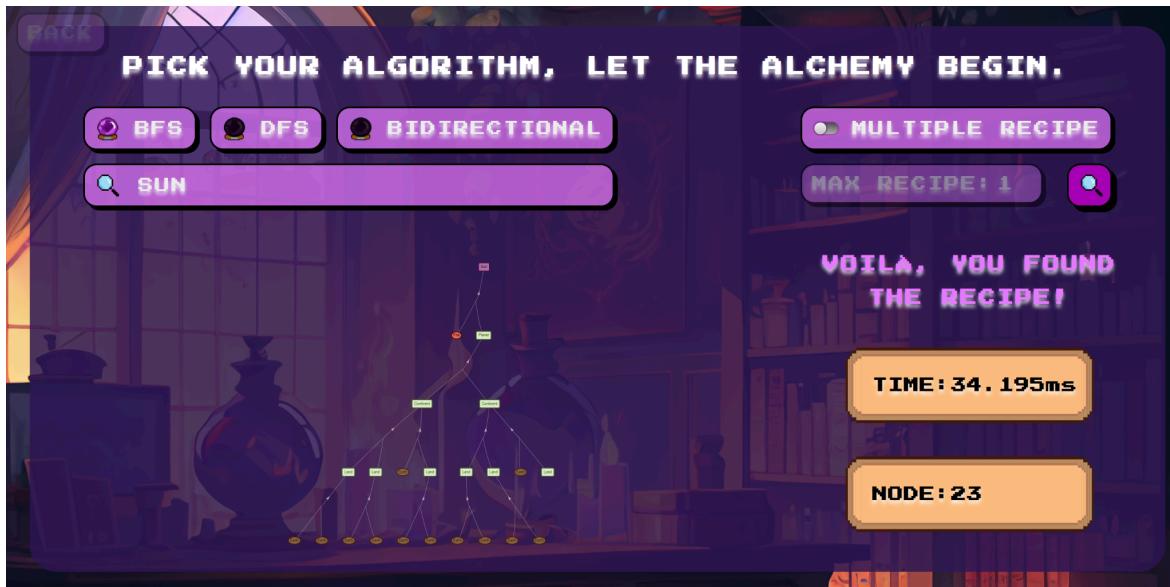
Gambar 14. Hasil Pengujian 2

Hasil Pengujian 3:



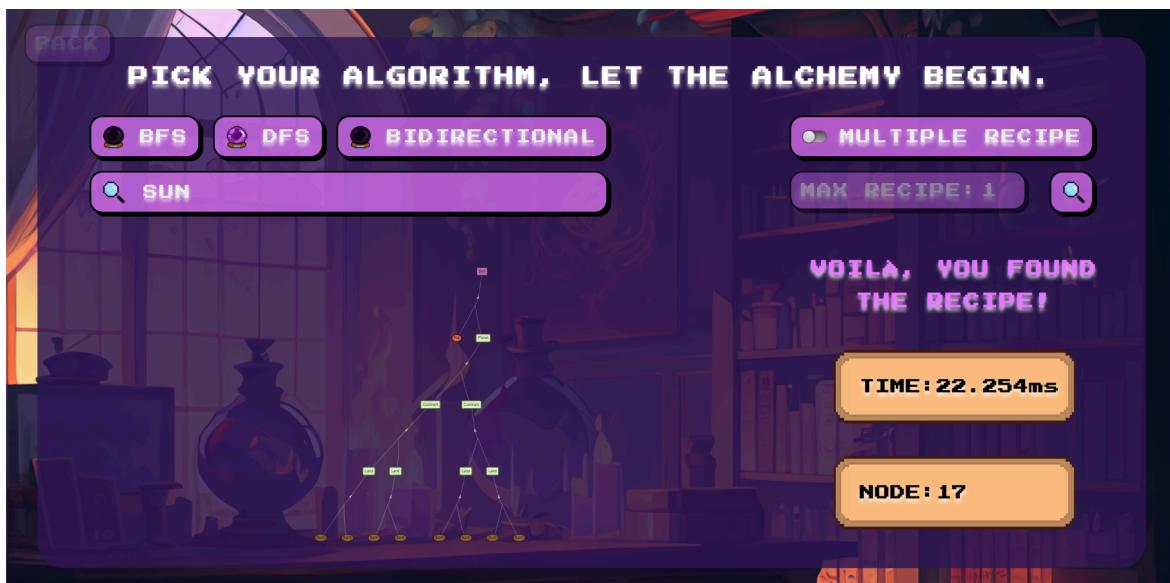
Gambar 15. Hasil Pengujian 3

Hasil Pengujian 4:



Gambar 16. Hasil Pengujian 4

Hasil Pengujian 5:



Gambar 17. Hasil Pengujian 5

Hasil Pengujian 6:



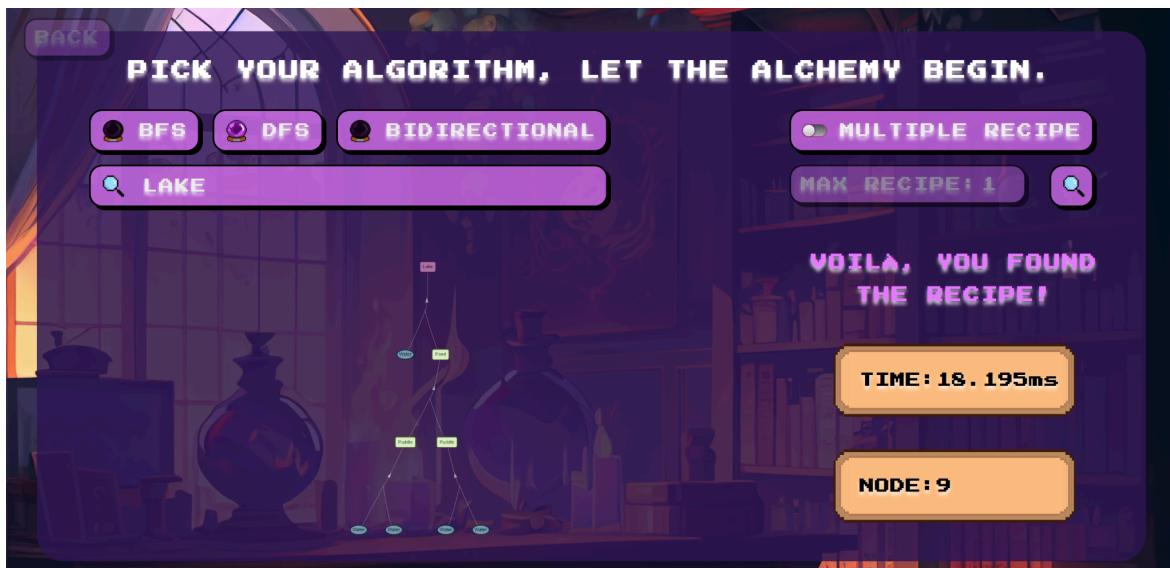
Gambar 18. Hasil Pengujian 6

Hasil Pengujian 7:



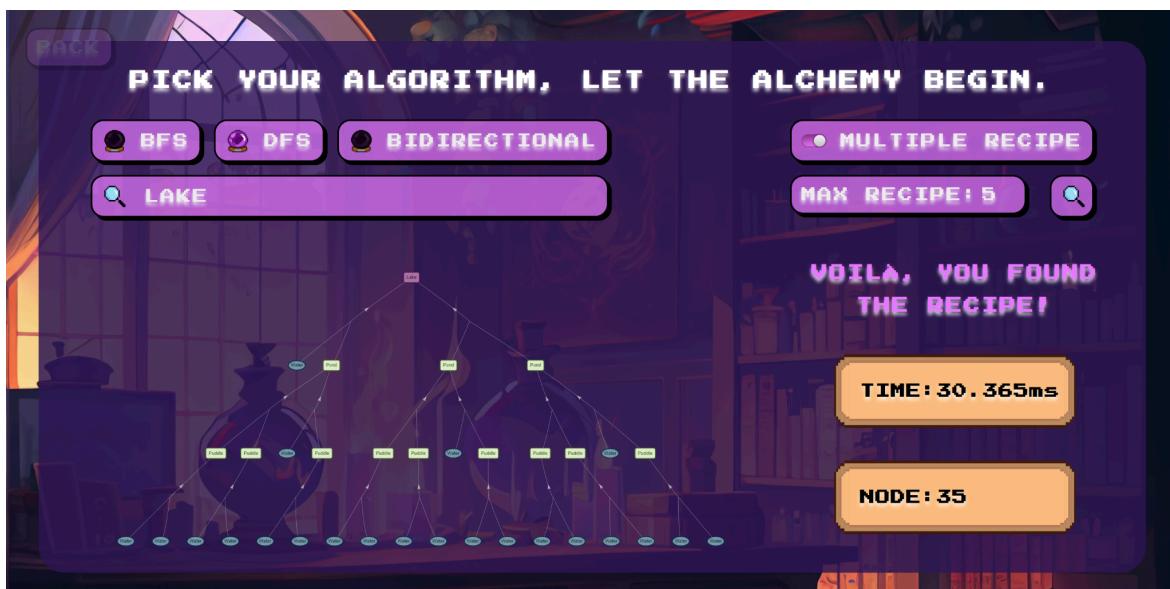
Gambar 19. Hasil Pengujian 7

Hasil Pengujian 8:



Gambar 20. Hasil Pengujian 8

Hasil Pengujian 9:



Gambar 21. Hasil Pengujian 9

## 4.5 Analisis Hasil Pengujian

Dari serangkaian pengujian dengan elemen target seperti “SWORD”, “SUN”, dan “LAKE”, dalam pencarian satu resep, algoritma DFS mengungguli BFS, tidak hanya dalam hal kecepatan eksekusi yang lebih rendah tetapi juga dalam menghasilkan solusi dengan jumlah *node* yang lebih sedikit, mengindikasikan jalur resep yang lebih ringkas atau penemuan solusi yang lebih cepat di kedalaman graf. Sebagai contoh, untuk elemen “LAKE”, DFS hanya membutuhkan sekitar 18 ms dengan 9 node, sementara BFS memerlukan sekitar 30 ms dengan 21 node. Keunggulan serupa juga terlihat pada elemen “SWORD” dengan DFS (19 ms, 21 node) jauh lebih efisien dibandingkan BFS (55 ms, 53 node), serta pada elemen “SUN” dengan DFS (22 ms, 17 node) yang lebih cepat dari BFS (34 ms, 23 node) untuk menemukan satu resep.

Ketika parameter “*Max Recipe*” ditingkatkan, pada algoritma BFS, peningkatan jumlah resep yang dicari menggunakan BFS tidak selalu mengakibatkan peningkatan waktu eksekusi secara drastis, meskipun jumlah *node* dalam keseluruhan solusi yang ditampilkan cenderung bertambah. Ini mengindikasikan bahwa setelah BFS melakukan eksplorasi hingga kedalaman tertentu untuk menemukan solusi pertama, penemuan beberapa solusi tambahan pada kedalaman yang serupa atau sedikit lebih jauh tidak menambah beban waktu pencarian secara signifikan, tetapi akan memperbesar kompleksitas total dari gabungan semua resep yang ditampilkan.

Secara keseluruhan, hal ini memperkuat pemahaman mengenai karakteristik DFS dan BFS. DFS dengan pendekatannya yang mendalam terlebih dahulu, terbukti sangat efektif untuk menemukan satu solusi dengan cepat, terutama jika solusi tersebut tidak berada pada kedalaman graf yang ekstrem dan banyak jalur alternatif yang mungkin. Di sisi lain, BFS yang melakukan pencarian lapis demi lapis, meskipun membutuhkan waktu lebih lama dan melibatkan lebih banyak *node* untuk satu solusi karena eksplorasinya yang menyeluruh menunjukkan kemampuan untuk mengakomodasi pencarian beberapa resep tanpa peningkatan waktu yang signifikan.

## BAB 5

# KESIMPULAN, SARAN, DAN REFLEKSI

### 5.1 Kesimpulan

Tugas Besar 2 mata kuliah Strategi Algoritma ini telah berhasil diselesaikan dengan pengembangan aplikasi *web* pencarian *recipe* untuk permainan Little Alchemy 2. Aplikasi ini mengimplementasikan dua algoritma traversal graf fundamental, yaitu *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS), untuk menemukan cara membuat elemen turunan dari 4 elemen dasar. Proses pengembangan dimulai dengan akuisisi data elemen dan resep melalui teknik *web scraping* dari *fandom* Wiki Little Alchemy 2 menggunakan library Go goquery. Data mentah yang diperoleh kemudian diproses dan direpresentasikan sebagai struktur graf internal di sisi *backend* yang dibangun dengan Go. Untuk memudahkan operasi algoritma dan penyajian data, informasi tambahan seperti ID elemen unik (berdasarkan urutan penemuan scraper), *tier* elemen, daftar resep pembentuk (FromPair berbasis ID), dan daftar elemen yang dapat dibuat (CanMake berbasis ID) juga dihitung dan disimpan dalam struktur `[]model.Element`. *Backend* menyediakan *endpoint* API utama (`GET /api/recipes/{elementName}`) yang memungkinkan *frontend* (dibangun terpisah menggunakan JavaScript dengan Next.js/React.js) untuk meminta pencarian resep dengan parameter algoritma (BFS/DFS), mode pencarian (*shortest/multiple*), dan jumlah resep yang diinginkan. Respons API dikirim dalam format JSON dan berisi pohon resep, waktu pencarian, serta jumlah *node* yang dikunjungi oleh algoritma. Selain itu, disediakan juga *endpoint* `/graph-data` untuk menampilkan keseluruhan data graf elemen yang telah diproses, serta dokumentasi API interaktif menggunakan Swagger.

Berdasarkan hasil pengujian, kedua algoritma mampu memberikan respons terhadap permintaan pencarian. Data graf yang dihasilkan oleh *scraper* berhasil mencakup mayoritas elemen yang ada di permainan, dan struktur data yang dipilih memungkinkan representasi hubungan resep dengan baik.

## 5.2 Saran

Untuk pengembangan lebih lanjut, beberapa saran yang dapat dipertimbangkan adalah:

1. Optimasi Algoritma

Optimasi *multithreading* pada BFS/DFS untuk pencarian *multiple recipe* perlu diimplementasikan secara cermat untuk mendapatkan manfaat performa yang signifikan tanpa menimbulkan *race condition*.

2. Validasi Data *Scraper* yang Lebih Mendalam

Meskipun *scraper* saat ini berhasil mengambil banyak data, validasi lebih lanjut terhadap keakuratan setiap resep dapat meningkatkan kualitas data graf. Perbandingan dengan daftar elemen resmi jika tersedia akan sangat membantu.

3. Strategi *Error Handling* yang Lebih Baik di API

Menyediakan pesan *error* yang lebih spesifik dan informatif jika terjadi masalah (elemen tidak ditemukan, parameter tidak valid, dan sebagainya).

4. Pengujian yang Lebih Luas

Perlu dilakukan pengujian dengan cakupan kasus yang lebih luas, termasuk elemen-elemen yang kompleks atau yang memiliki banyak jalur pembuatan untuk memastikan akurasi algoritma.

## 5.3 Refleksi

Pengerjaan Tugas Besar 2 ini memberikan pengalaman praktis dalam menerapkan konsep Strategi Algoritma pada kasus nyata. Proses mulai dari pengumpulan data melalui *web scraping*, pemodelan data menjadi struktur graf, perhitungan atribut graf, hingga perancangan API dan implementasi algoritma BFS dan DFS memberikan pemahaman yang lebih mendalam tentang pengembangan perangkat lunak dalam kehidupan nyata.

Salah satu tantangan utama yang dihadapi adalah konsistensi data dan struktur, terutama saat data berasal dari sumber eksternal (Wiki) yang memiliki format yang tidak selalu seragam. Selain itu, keputusan desain terkait struktur data internal dan bagaimana data ini akan digunakan oleh berbagai bagian aplikasi (algoritma, API *handler*) memerlukan pertimbangan yang matang untuk keseimbangan antara kemudahan implementasi dan efisiensi. Proyek ini juga menekankan pentingnya modularitas, di mana pemisahan antara *scraper*, logika inti, dan API *handler* akan sangat membantu dalam pengelolaan dan pengembangan jangka panjang.

## LAMPIRAN

### Tautan Repository Github

[https://github.com/karolyangqian/Tubes2\\_FE\\_ian](https://github.com/karolyangqian/Tubes2_FE_ian)

[https://github.com/graceevelyns/Tubes2\\_BE\\_ian](https://github.com/graceevelyns/Tubes2_BE_ian)

### Tautan Video

<https://youtu.be/-NADok-gNOo>

### Hasil Akhir Tugas Besar

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi dapat memperoleh data <i>recipe</i> melalui scraping.	✓	
3	Algoritma <i>Depth First Search</i> dan <i>Breadth First Search</i> dapat menemukan <i>recipe</i> elemen dengan benar.	✓	
4	Aplikasi dapat menampilkan visualisasi <i>recipe</i> 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 <i>Bidirectional</i> .		✓
9	Membuat bonus <i>Live Update</i> .		✓
10	Aplikasi di- <i>containerize</i> dengan Docker.	✓	
11	Aplikasi di- <i>deploy</i> dan dapat diakses melalui internet.	✓	

## LAMPIRAN

Docker. (n.d.). Docker Documentation. Diakses pada 12 Mei 2025, dari <https://docs.docker.com/>

Go Programming Language. (n.d.). Documentation. Diakses pada 08 Mei 2025, dari <https://go.dev/doc/>

Munir, R. (2025a). Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1. Departemen Informatika, Institut Teknologi Bandung. Diakses pada 08 Mei 2025, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf)

Munir, R. (2025b). Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1. Departemen Informatika, Institut Teknologi Bandung. Diakses pada 08 Mei 2025 dari [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)

Tailwind CSS. (n.d.). Documentation - Tailwind CSS v2. Diakses pada 08 Mei 2025, dari <https://v2.tailwindcss.com/docs>