

# **Laporan Tugas Besar 2**

## **IF2211 Strategi Algoritma 2025/2026**



Disusun oleh:

Kelas K1 - Kelompok 26 - CCP

Varel Tiara	(13523008)
Nathaniel Jonathan Rusli	(13523013)
Yonatan Edward Njoto	(13523036)

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
JL. GANESA 10, BANDUNG 40132**

**2025**

## KATA PENGANTAR

Dengan penuh rasa syukur, kami menyusun dan mempersembahkan makalah ini sebagai bagian dari Tugas Besar 2 mata kuliah IF2211 Strategi Algoritma. Topik yang diangkat dalam tugas besar ini adalah “*Perancangan dan Implementasi Visualisasi Recipe Tree pada Game Little Alchemy 2*”, yang bertujuan untuk menerapkan dan membandingkan berbagai strategi algoritma pencarian dalam menyelesaikan masalah pencarian kombinasi elemen secara optimal.

Melalui tugas ini, kami mengembangkan sebuah aplikasi pencari *recipe tree* yang mampu menemukan jalur pembuatan suatu elemen dari elemen dasar menggunakan berbagai algoritma pencarian seperti Breadth-First Search (BFS), Depth-First Search (DFS), dan Bidirectional Search. Setiap algoritma juga mendukung fitur *multithreaded execution* dalam pencarian resep ganda, menjadikan pemrosesan algoritma lebih efisien.

Pengerjaan proyek ini tidak hanya memberikan kesempatan bagi kami untuk memperdalam pemahaman mengenai teori algoritma *searching*, tapi juga menantang kami dalam mengintegrasikan konsep tersebut ke dalam pengembangan aplikasi berbasis web yang responsif dan interaktif. Melalui proses ini, kami belajar untuk menggabungkan aspek teoretis dan praktis secara seimbang, serta mengeksplorasi penerapan algoritma dalam konteks nyata dan menyenangkan.

Kami mengucapkan terima kasih kepada dosen pengampu serta seluruh pihak yang telah membimbing dan memberikan dukungan selama penyusunan tugas ini. Semoga laporan ini dapat memberikan gambaran yang jelas mengenai hubungan antara strategi algoritma dan pengembangan aplikasi nyata, serta menjadi referensi yang bermanfaat bagi pembaca yang tertarik dalam bidang algoritma dan rekayasa perangkat lunak.

Bandung, 12 Mei 2025

Varel Tiara (13523008)

Nathaniel Jonathan Rusli (13523013)

Yonatan Edward Njoto (13523036)

## DAFTAR ISI

<b>KATA PENGANTAR</b>	<b>2</b>
<b>DAFTAR ISI</b>	<b>3</b>
<b>DAFTAR GAMBAR</b>	<b>6</b>
<b>BAB I</b>	<b>7</b>
<b>DESKRIPSI TUGAS</b>	<b>7</b>
<b>BAB II</b>	<b>9</b>
<b>LANDASAN TEORI</b>	<b>9</b>
2.1 Penjelajahan Graf	9
2.2 Algoritma Breadth First Search (BFS)	10
2.3 Algoritma Depth First Search (DFS)	11
2.4 Algoritma Bidirectional	13
2.5 Penjelasan Singkat Mengenai Aplikasi Website yang Dibangun	14
2.5.1 Backend Development	14
2.5.2 Frontend Development	15
2.5.3 Docker	16
2.5.4 Deployment	17
<b>BAB III</b>	<b>19</b>
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>19</b>
3.1 Langkah-Langkah Pemecahan Masalah	19
3.1.1 Pengumpulan Data Elemen dan Resep	19
3.1.1.1 Web Scraping dari Wiki Little Alchemy 2	19
3.1.1.2 Penanganan Elemen yang Tidak Ditemukan	19
3.1.2 Pemodelan Graf untuk Representasi Resep	20
3.1.2.1 Struktur Data Graf Elemen	20
3.1.2.2 Penyusunan Hierarki Tier Elemen	20
3.1.3 Implementasi Algoritma Pencarian	21
3.1.3.1 Algoritma Breadth-First Search (BFS)	21
3.1.3.2 Algoritma Depth-First Search (DFS)	22
3.1.4 Visualisasi dan Antarmuka Pengguna	23
3.1.4.1 Pembuatan Wiki Elemen	23
3.1.4.2 Visualisator Pohon Resep	24
3.1.4.3 Komunikasi Real-time dengan WebSocket	24
3.1.5 Optimasi Performa	24
3.1.5.1 Paralelisasi Pencarian	24

3.1.5.2 Caching dan Pengoptimalan Graf	25
3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS	25
3.2.1 Identifikasi Jenis Permasalahan	25
3.2.2 Identifikasi Simpul (Node)	26
3.2.3 Identifikasi Sisi (Edge)	27
3.2.4 Pemetaan untuk Algoritma Depth-First Search (DFS)	28
3.2.5 Pemetaan untuk Algoritma Breadth-First Search (BFS)	29
3.2.6 Pemetaan untuk Algoritma Bidirectional Search	30
3.2.7 Perbandingan Pendekatan DFS dan BFS dalam Konteks Permainan	32
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun	33
3.3.1 Fitur Fungsional Aplikasi Web yang Dibangun	33
3.3.1.1 Navigation Bar	33
3.3.1.2 Halaman Wiki (Elements Encyclopedia)	33
3.3.1.2.1 Halaman Utama	33
3.3.1.2.2 Halaman Elemen Detail	34
3.3.1.3 Halaman Visualizer (Pencari Resep Elemen)	35
3.3.1.4 Halaman About Us	36
3.3.2 Arsitektur Aplikasi Web yang Dibangun	36
3.3.2.1 Arsitektur Sistem Keseluruhan	36
3.3.2.2 Arsitektur Backend	37
3.3.2.2.1 Struktur Komponen Backend	37
3.3.2.2.2 Alur Data di Backend	38
3.3.2.3 Arsitektur Frontend	38
3.3.2.3.1 Struktur Halaman Frontend	38
3.3.2.3.2 Manajemen State dan Routing	39
3.3.2.4 Mekanisme Komunikasi Frontend-Backend	40
3.3.2.4.1 REST API	40
3.3.2.4.2 WebSocket	40
3.4 Ilustrasi Kasus	40
<b>BAB IV</b>	<b>44</b>
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>44</b>
4.1 Spesifikasi Teknis Program	44
4.1.1 Struktur Data	44
4.1.2 Fungsi	47
4.1.3 Prosedur	74
4.2 Penjelasan Tata Cara Penggunaan Program (Interface Program, Fitur-fitur yang disediakan program, dan sebagainya)	84

4.2.1 Interface Program, Fitur-fitur yang Disediakan Program	84
4.2.2 Tata Cara Penggunaan Program	89
4.3 Hasil Pengujian	90
4.4 Analisis Hasil Pengujian	94
4.4.1. Perbandingan Kinerja Algoritma Pencarian	94
4.4.1.1 BFS (Breadth-First Search)	94
4.4.1.2 DFS (Depth-First Search)	95
4.4.1.3 Bidirectional Search	95
4.4.2. Analisis Berdasarkan Kompleksitas Elemen	96
4.4.3. Analisis Efektifitas Visualisasi Real-time	97
4.4.4. Analisis Konsumsi Sumber Daya	97
4.4.5. Analisis Paralelisme dengan Goroutines	97
4.4.6. Kesimpulan Hasil Pengujian	98
4.4.6.1 Algoritma Optimal untuk Berbagai Kasus	98
4.4.6.2 Karakteristik Algoritma	98
<b>BAB V</b>	<b>100</b>
<b>KESIMPULAN DAN SARAN</b>	<b>100</b>
5.1 Kesimpulan	100
5.2 Saran	100
5.3 Refleksi	101
<b>BAB VI</b>	<b>102</b>
<b>LAMPIRAN</b>	<b>102</b>
6.1 Daftar Pustaka	102
6.2 Tautan Repository	102
6.3 Tautan Video	102
6.4 Tabel Checklist	102

## DAFTAR GAMBAR

Gambar 1.1. Little Alchemy 2.....	6
Gambar 1.2. Elemen Dasar pada Little Alchemy 2.....	7
Gambar 2.1. Ilustrasi Graf.....	8
Gambar 2.2. Ilustrasi Graf dan Pohon BFS.....	10
Gambar 2.3. Ilustrasi Graf dan Pohon DFS.....	11
Gambar 2.4. Ilustrasi Bidirectional Search.....	13
Gambar 3.1. Navigation Bar.....	29
Gambar 3.2. Halaman Wiki.....	30
Gambar 3.3. Halaman Elemen Detail.....	31
Gambar 3.4. Halaman Visualizer.....	32
Gambar 3.5. Halaman About Us.....	33
Gambar 3.6. Diagram Websocket.....	34
Gambar 3.7. Input Recipe Tree Visualizer.....	38
Gambar 3.8. Pilihan Search Algorithm.....	38
Gambar 3.9. Hasil Pencarian Elemen.....	39
Gambar 3.10. Pencarian Multi Recipes.....	39
Gambar 3.11. Hasil Pertama Multi Recipes.....	40
Gambar 3.12. Hasil Kedua Multi Recipes.....	40

# BAB I

## DESKRIPSI TUGAS



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

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

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

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

Komponen-komponen dari permainan ini antara lain:

### 1. Elemen dasar

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



**Gambar 1.2.** Elemen Dasar pada Little Alchemy 2

### 2. Elemen turunan

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

### 3. *Combine Mechanism*

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

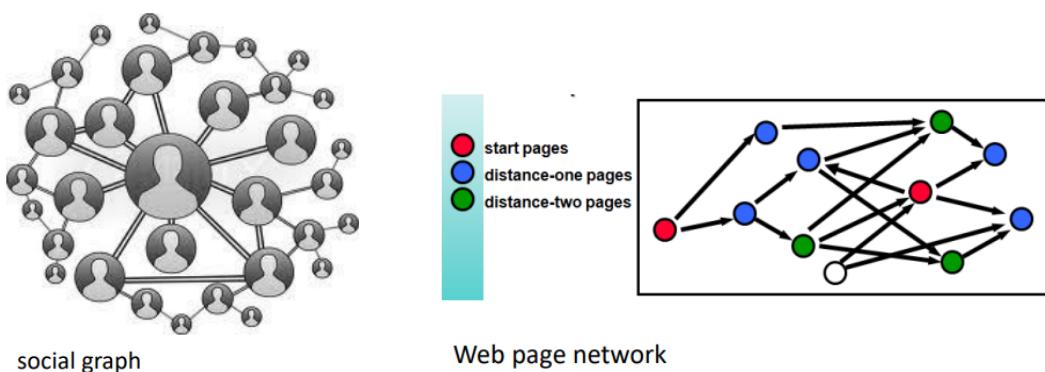
## BAB II

### LANDASAN TEORI

#### 2.1 Penjelajahan Graf

Algoritma penjelajahan atau *traversal* graf adalah proses mengunjungi simpul-simpul dalam suatu graf secara sistematis. Dalam konteks ini, diasumsikan bahwa graf yang digunakan merupakan graf terhubung, sehingga setiap simpul dapat dicapai dari simpul lainnya. Graf sendiri merupakan representasi dari suatu persoalan, sehingga traversal graf dapat diartikan sebagai proses pencarian solusi dari persoalan yang direpresentasikan dalam bentuk graf.

1. **Pencarian Melebar (Breadth First Search / BFS):** Algoritma ini menjelajahi graf secara melebar, yaitu mengunjungi semua simpul tetangga terlebih dahulu sebelum melanjutkan ke tingkat berikutnya.
2. **Pencarian Mendalam (Depth First Search / DFS):** Algoritma ini menjelajahi graf secara mendalam, yaitu terus menelusuri jalur tertentu hingga tidak ada simpul yang belum dikunjungi, kemudian melakukan *backtracking*.



**Gambar 2.1.** Ilustrasi Graf

Dalam implementasinya, penjelajahan graf dapat dibedakan berdasarkan pendekatan representasi graf sebagai berikut:

1. **Graf Statis:** Graf telah terbentuk sepenuhnya sebelum proses pencarian dimulai, dan direpresentasikan sebagai struktur data (misalnya matriks ketetanggaan atau daftar ketetanggaan).
2. **Graf Dinamis:** Graf dibangun selama proses pencarian berlangsung, dan belum tersedia sepenuhnya pada awal pencarian. Graf akan berkembang sesuai jalur pencarian yang ditempuh.

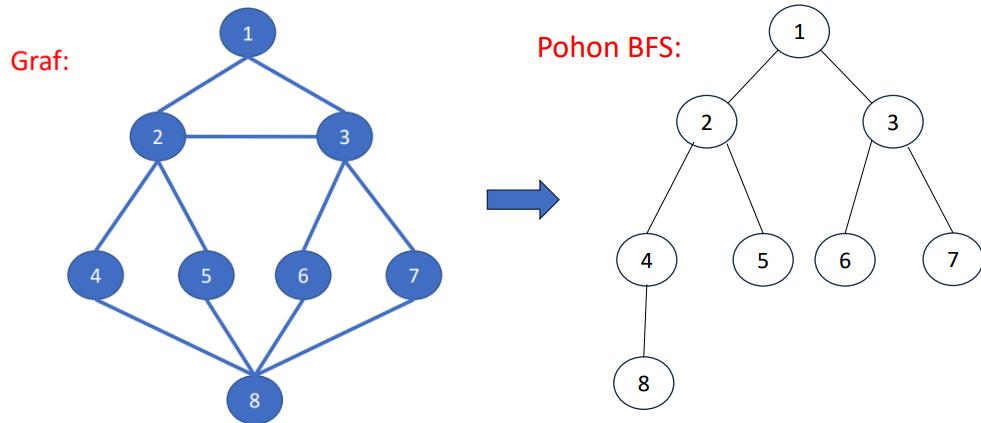
Selain itu, algoritma penelusuran graf dapat dikelompokkan berdasarkan ketersediaan informasi tambahan dalam proses pencarian solusi:

1. **Pencarian Tanpa Informasi (Uninformed/blind search):** Algoritma yang tidak menggunakan pengetahuan tambahan tentang solusi atau kondisi tujuan. Contoh algoritmanya, seperti DFS, BFS, Depth Limited Search, Iterative Deepening Search, Uniform Cost Search.
2. **Pencarian Dengan Informasi (Informed Search):** Algoritma ini memanfaatkan fungsi heuristik untuk memperkirakan simpul mana yang lebih menjanjikan dalam mendekati solusi. Contoh algoritmanya, seperti Best First Search, A\*.

## 2.2 Algoritma Breadth First Search (BFS)

Breadth First Search (BFS) adalah algoritma pencarian yang melakukan penelusuran graf secara melebar (*level-order*). Artinya, simpul-simpul yang berada pada tingkat yang sama akan dikunjungi terlebih dahulu sebelum melanjutkan ke simpul pada tingkat berikutnya. Pendekatan ini memastikan bahwa semua kemungkinan jalur dengan jumlah langkah paling sedikit akan dieksplorasi terlebih dahulu. Langkah-langkah umum algoritma BFS adalah sebagai berikut:

1. Mulai dari simpul awal V.
2. Kunjungi semua simpul yang bertetangga langsung dengan simpul v.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi.
4. Ulangi proses ini hingga seluruh simpul yang dapat dicapai telah dikunjungi.



**Gambar 2.2.** Ilustrasi Graf dan Pohon BFS

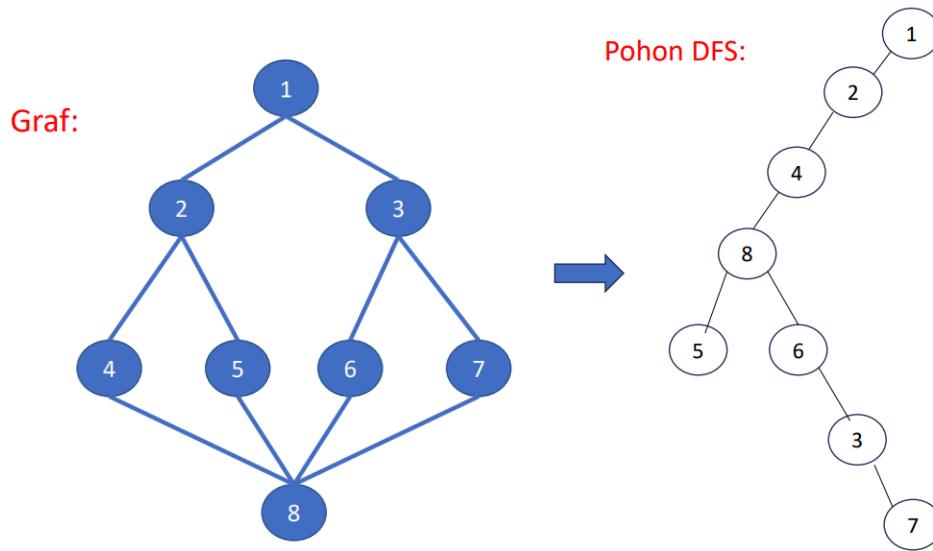
BFS menggunakan struktur data antrian (queue) untuk mengelola simpul-simpul yang akan dikunjungi berikutnya, serta tabel penanda untuk mencatat simpul-simpul yang telah dikunjungi. Karakteristik BFS, seperti berikut

1. Completeness (kelengkapan): BFS menjamin akan menemukan solusi jika solusi memang ada.
2. Optimality (keoptimalan): BFS akan menemukan solusi dengan jumlah langkah minimum, *asalkan* semua langkah memiliki biaya yang sama.
3. Kompleksitas waktu:  $O(b^d)$ , di mana b adalah branching factor (jumlah maksimum anak dari suatu simpul), dan d adalah kedalaman solusi.
4. Kompleksitas ruang:  $O(b^d)$ , karena BFS menyimpan semua simpul pada level tertentu sebelum melanjutkan ke level berikutnya.

### 2.3 Algoritma Depth First Search (DFS)

Depth First Search (DFS) adalah algoritma pencarian yang melakukan penelusuran graf secara mendalam terlebih dahulu. Algoritma ini akan menyusuri satu jalur sejauh mungkin hingga mencapai simpul daun (atau simpul yang tidak memiliki tetangga yang belum dikunjungi), sebelum melakukan *backtracking* ke simpul sebelumnya untuk menjelajahi jalur lainnya. Langkah-langkah umum algoritma DFS adalah sebagai berikut:

1. Mulai dari simpul awal v.
2. Kunjungi salah satu simpul w yang bertetangga dengan simpul v dan belum dikunjungi.
3. Lanjutkan proses DFS dari simpul w.
4. Jika telah mencapai simpul u di mana semua tetangganya telah dikunjungi, maka lakukan *backtrack* ke simpul sebelumnya yang masih memiliki tetangga yang belum dikunjungi.
5. Proses berakhir jika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul manapun yang telah dikunjungi.



**Gambar 2.3.** Ilustrasi Graf dan Pohon DFS

DFS dapat diimplementasikan menggunakan struktur data stack atau lebih umum lagi dengan pendekatan rekursif, yang secara implisit menggunakan stack pemanggilan fungsi. Karakteristik DFS, seperti berikut

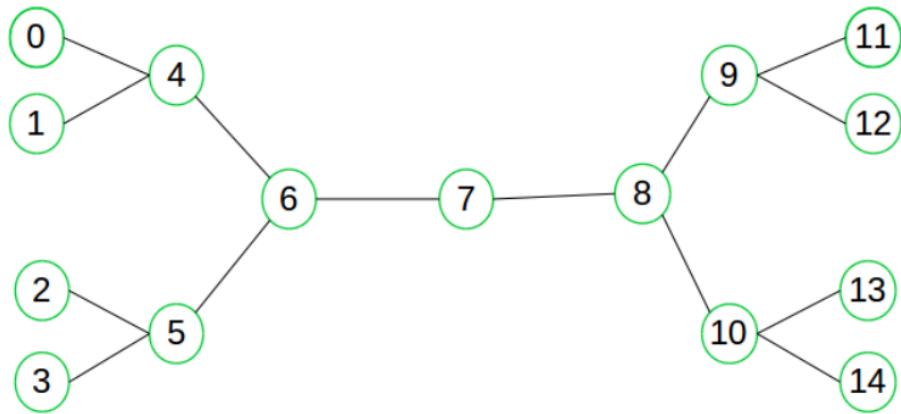
1. Tidak selalu optimal.
2. Dapat lebih efisien dalam ruang dibanding BFS.
3. Cocok untuk masalah yang solusinya dalam jalur panjang atau memiliki banyak pilihan bercabang (e.g., puzzle).

## 2.4 Algoritma Bidirectional

Bidirectional Search merupakan salah satu algoritma pencarian graf yang dirancang untuk menemukan jalur terpendek dari simpul awal (*source*) ke simpul tujuan (*goal*). Berbeda dengan pendekatan sebelumnya, yaitu Breadth First Search (BFS) atau Depth First Search (DFS) yang melakukan pencarian dari satu arah saja, bidirectional search melakukan pencarian secara dua arah sekaligus: pencarian maju dari simpul awal menuju simpul tujuan, dan pencarian mundur dari simpul tujuan menuju simpul awal. Kedua pencarian ini berjalan secara paralel dan akan berhenti ketika keduanya bertemu di suatu simpul yang sama. Titik pertemuan ini menunjukkan bahwa jalur dari sumber ke tujuan telah ditemukan.

Keunggulan utama dari pendekatan bidirectional ini terletak pada efisiensi ruang pencarian. Dalam pencarian satu arah seperti BFS/DFS, kompleksitas waktu dan ruang biasanya adalah  $O(b^d)$ , di mana  $b$  adalah *branching factor* dan  $d$  adalah kedalaman solusi. Namun dengan membagi pencarian menjadi dua arah, masing-masing hanya menelusuri hingga  $d/2$  kedalaman, kompleksitas menjadi  $O(b^{d/2})$  untuk setiap arah, sehingga totalnya hanya  $O(b^{d/2} + b^{d/2})$ , yang jauh lebih kecil dibandingkan pencarian satu arah penuh. Oleh karena itu, bidirectional search sangat cocok digunakan pada graf yang besar dengan solusi yang berada cukup dalam.

Algoritma ini paling efektif digunakan ketika simpul awal dan simpul tujuan sudah diketahui secara eksplisit dan tidak berubah, serta ketika *branching factor* dari graf relatif seimbang di kedua arah. Dalam kondisi tersebut, algoritma ini tidak hanya lengkap (complete) jika BFS digunakan, tetapi juga optimal jika semua langkah memiliki biaya yang sama. Selain itu, sama seperti algoritma A\*, pendekatan ini juga dapat digabungkan dengan heuristik untuk memperkirakan jarak tersisa, baik dari simpul awal maupun simpul tujuan, guna mempercepat pencarian jalur terpendek.



**Gambar 2.4.** Ilustrasi Bidirectional Search

(Sumber: <https://www.geeksforgeeks.org/bidirectional-search/>)

Sebagai contoh sederhana, jika ingin mencari jalur dari simpul 0 ke simpul 14, maka pencarian dapat dilakukan dari kedua simpul tersebut. Jika pencarian dari arah 0 dan 14 bertemu di simpul 7, maka kita tahu bahwa jalur telah ditemukan dan eksplorasi lebih lanjut bisa dihentikan. Alhasil, pencarian yang dilakukan membuat pengurangan pencarian yang tidak perlu sehingga mempercepat proses pencarian secara signifikan.

## 2.5 Penjelasan Singkat Mengenai Aplikasi Website yang Dibangun

Aplikasi web yang dibangun merupakan sistem berbasis *fullstack* yang mengintegrasikan Go (Golang) sebagai backend dan React dengan Vite serta TypeScript sebagai frontend. Backend bertanggung jawab untuk mengatur algoritma pencarian resep untuk menentukan setiap elemen, seperti BFS, DFS, dan bidirectional.

### 2.5.1 Backend Development

Backend bertanggung jawab untuk mengelola logika pengembangan algoritma, pemrosesan data, serta komunikasi dua arah dengan frontend melalui protokol WebSocket/ws/wss dan HTTP/HTTPS. Beberapa komponen utama dalam struktur backend mencakup:

1. Controllers: Menangani logika program dan merespons permintaan dari frontend, termasuk pemilihan algoritma traversal dan pengolahan data graf.
2. Models: Mendefinisikan struktur data yang digunakan, termasuk representasi graf, node, serta utilitas algoritma.
3. Data: Berisi data statis seperti file berformat JSON yang digunakan sebagai input graf atau data referensi lainnya.

Teknologi WebSocket memungkinkan backend mengirim data secara real-time ke frontend, sehingga visualisasi traversal graf dapat diperbarui secara dinamis saat pengguna melakukan interaksi.

### **2.5.2 Frontend Development**

Frontend dirancang untuk menghadirkan antarmuka yang interaktif, responsif, dan ramah pengguna. Aplikasi ini dibangun menggunakan React dengan TypeScript, serta menggunakan Vite untuk mendukung pengembangan cepat dan efisien. Struktur utama dari frontend meliputi beberapa halaman dan fitur utama, antara lain:

1. Home Page: Halaman utama yang menyambut pengguna dan memberikan informasi dasar aplikasi.
2. About Us Page: Menyediakan informasi mengenai tim pengembang dan tujuan aplikasi.
3. Layout Page: Menyediakan kerangka umum tampilan aplikasi yang konsisten di seluruh halaman (Navbar dan Footer).
4. Visualizer Page: Fitur utama yang menampilkan visualisasi traversal graf untuk mencari rute setiap elemen secara interaktif berdasarkan algoritma yang dipilih pengguna.
5. Wiki Page: Menampilkan seluruh elemen yang tersedia dalam aplikasi website ini.

Selain itu, frontend juga mengelola asset visual seperti ikon dan gambar, serta mengatur routing aplikasi untuk navigasi antarhalaman. Komunikasi dengan backend melalui

WebSocket membuat antarmuka dapat merespons perubahan data secara langsung, meningkatkan pengalaman pengguna dalam memahami algoritma graf.

### 2.5.3 Docker

Aplikasi ini telah didockerisasi untuk memastikan proses pengembangan dan deployment berjalan secara konsisten dan efisien di berbagai lingkungan. Dockerization adalah proses mengemas aplikasi beserta seluruh dependensinya ke dalam container, sehingga dapat dijalankan di mana saja tanpa khawatir terhadap perbedaan konfigurasi sistem atau infrastruktur.

Dalam proyek ini, terdapat dua pendekatan utama dockerization:

#### 1. Docker Compose – Multi-Container Setup

Setiap bagian sistem (scraper, backend, dan frontend) dijalankan dalam container terpisah yang didefinisikan dalam file docker-compose.yml. Ini memungkinkan pengembangan dan eksekusi yang modular dan terisolasi.

- Scraper: Service Go yang dibangun di folder ./src/scraper dan dijalankan secara independen.
- Backend: Aplikasi server berbasis Go yang menangani API dan WebSocket. Dibangun dari folder ./src/backend.
- Frontend: Aplikasi React (Vite + TypeScript) yang dikompilasi dan disajikan sebagai aplikasi statis.

Setiap service berbagi volume yang sama (./src:/app/src) dan diatur melalui network internal Docker Compose, sehingga memudahkan komunikasi antarkomponen tanpa konfigurasi tambahan.

Penggunaan Docker Compose menyederhanakan eksekusi seluruh sistem hanya dengan satu perintah

```
docker-compose up -build
```

## 2. Dockerfile – Single Image Unified Setup

Sebagai alternatif, proyek ini juga menyediakan Dockerfile monolitik yang membangun seluruh aplikasi (frontend, backend, dan scraper) menjadi satu image.

- Tahapan build dilakukan secara bertahap menggunakan multi-stage build untuk efisiensi:
  - Stage 1: Build frontend React.
  - Stage 2: Build backend Go.
  - Stage 3: Build scraper Go.
  - Stage 4: Gabungkan semua hasil build dalam satu final image.
- Aplikasi dijalankan melalui skrip start.sh yang mengeksekusi semua komponen.

Pendekatan ini cocok untuk kebutuhan deployment sederhana atau production ringan, karena hanya menghasilkan satu container yang menjalankan seluruh aplikasi, dan pendekatan ini kami gunakan untuk deployment di VPS.

Eksekusi image cukup dengan:

```
docker build -t my-app .
docker run -p 4000:4000 my-app
```

### 2.5.4 Deployment

Aplikasi Recipe Tree Finder untuk *Little Alchemy 2* dikembangkan dengan dua komponen utama: backend (menggunakan Golang) dan frontend (menggunakan React + Vite). Dengan menggunakan Docker, kedua komponen ini dikemas dalam container yang terpisah namun dapat saling berkomunikasi, umumnya dikelola bersama dalam file docker-compose.yml.

Alur deployment web dengan Docker dapat dijelaskan sebagai berikut:

1. Dockerfile disiapkan untuk backend dan frontend. Masing-masing file ini mendeskripsikan environment build seperti image dasar, perintah build, dan port yang dibuka.
2. Docker Compose digunakan untuk mengatur kedua container dalam satu layanan aplikasi terpadu. File ini akan mengatur build, networking, dan environment variables antar container.
3. Setelah dijalankan dengan docker compose up, aplikasi dapat diakses melalui web browser di alamat seperti <http://localhost:4001> (frontend) dan <http://localhost:4000> (backend WebSocket).
4. Dengan konfigurasi ini, aplikasi dapat di-deploy dengan mudah baik di server lokal maupun layanan cloud.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1 Langkah-Langkah Pemecahan Masalah**

Untuk menyelesaikan permasalahan pencarian resep kombinasi elemen dalam Little Alchemy 2, kami telah melakukan dekomposisi masalah menjadi beberapa bagian utama. Berikut adalah langkah-langkah pemecahan masalah yang kami implementasikan:

##### **3.1.1 Pengumpulan Data Elemen dan Resep**

###### **3.1.1.1 Web Scraping dari Wiki Little Alchemy 2**

Langkah pertama dalam pemecahan masalah adalah mengumpulkan data elemen dan resep dari game Little Alchemy. Untuk mendapatkan data yang komprehensif, kami membuat scraper berbasis bahasa Go yang mengakses halaman wiki resmi Little Alchemy 2. Scraper ini dirancang untuk mengekstrak beberapa informasi penting, seperti nama elemen, kombinasi resep (elemen-elemen yang dapat digabungkan untuk membuat elemen baru), dan URL gambar setiap elemen.

Proses scraping ini dilakukan dengan mengenali struktur HTML pada halaman Little Alchemy 2 Wiki Fandom dan mengekstrak data dari tabel elemen. Kami juga memanfaatkan library goquery untuk melakukan parsing HTML dan ekstraksi data secara efisien (func scrapeElements(url string) []Element {} ).

###### **3.1.1.2 Penanganan Elemen yang Tidak Ditemukan**

Salah satu tantangan yang kami hadapi adalah adanya elemen yang tercantum dalam resep namun tidak secara eksplisit disebut sebagai elemen (di kolom kiri). Untuk mengatasi masalah ini, kami mengimplementasikan fungsi khusus yang mengidentifikasi elemen yang hilang. (func getMissingElementsIngredients(elements []Element, url string) []Element {}).

### 3.1.2 Pemodelan Graf untuk Representasi Resep

#### 3.1.2.1 Struktur Data Graf Elemen

Setelah data terkumpul, kami membutuhkan struktur data yang tepat untuk memodelkan hubungan antar elemen. Model graf menjadi pilihan yang ideal karena

1. Elemen dapat direpresentasikan sebagai node
2. Resep dapat direpresentasikan sebagai edge
3. Pembuatan elemen baru dapat dilihat sebagai traversal dari satu node ke node lainnya

#### 3.1.2.2 Penyusunan Hierarki Tier Elemen

Untuk memastikan bahwa algoritma pencarian beroperasi secara efisien, kami mengimplementasikan sistem tier yang mengelompokkan elemen berdasarkan "level" mereka dalam hierarki resep:

1. Tier 0: Elemen dasar (Air, Api, Tanah, Udara) yang tersedia dari awal dan elemen yang tidak dapat dibuat dari elemen lain
2. Tier 1: Elemen yang dapat dibuat dengan menggabungkan elemen dasar
3. Tier 2+: Elemen yang membutuhkan kombinasi elemen tier lebih rendah

Implementasi sistem tier membantu algoritma pencarian untuk menghindari siklus dan mengoptimalkan pencarian resep:

```
...  
// Menginisialisasi tier untuk setiap elemen  
for _, el := range nameToNode {  
    if node.Tier == -1 {  
        // Tentukan tier berdasarkan resep yang diperlukan  
        // Elemen dengan tier lebih tinggi membutuhkan elemen dengan tier lebih rendah  
    }  
}
```

### **3.1.3 Implementasi Algoritma Pencarian**

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

Algoritma Breadth-First Search (BFS) yang diimplementasikan dalam fungsi `BFSFindTrees` digunakan untuk menelusuri semua kemungkinan pohon resep (recipe trees) yang dapat menghasilkan sebuah elemen target, dengan eksplorasi dilakukan secara melebar dari setiap bahan penyusun (prerequisite elements) yang dibutuhkan. Pencarian ini dimulai dari resep-resep yang tersedia untuk membentuk elemen target, lalu menelusuri secara paralel bahan-bahan dari setiap resep tersebut. Tujuan utama dari pendekatan ini adalah untuk menghasilkan pohon resep yang lengkap dan valid, sambil membatasi jumlah hasil maksimal (`maxTreeCount`) agar performa tetap terjaga.

Setiap node antrian (queue item) dalam pencarian menyimpan elemen yang sedang diproses, level kedalamannya, dan status kelengkapan subpohnnya. BFS tidak menjamin bahwa pohon pertama yang ditemukan adalah yang paling pendek atau paling optimal dalam hal jumlah langkah atau kompleksitas bahan. Ini karena dalam satu level pencarian, bisa saja terdapat resep yang menghasilkan jalur lebih efisien namun diproses belakangan. Selain itu, jika elemen bukan elemen dasar (base element) dan memiliki lebih dari satu cara pembentukan, semua alternatif resep tersebut akan dievaluasi dan dicoba dibentuk menjadi subpohon melalui kombinasi seluruh kemungkinan dari pohon bahan kiri dan kanan.

BFS dalam konteks ini tidak hanya menelusuri node satu per satu, tetapi juga membentuk dan menyimpan struktur pohon resep yang lengkap dengan setiap kombinasi kemungkinan. Oleh karena itu, meskipun eksplorasi bersifat melebar, sifat eksplisit dan detail dari pembuatan pohon membuat pendekatan ini cocok untuk aplikasi seperti crafting tree, meskipun tidak optimal untuk pencarian jalur terpendek absolut. Algoritma ini juga memanfaatkan goroutine dan kanal (channel) untuk menjalankan pencarian paralel pada resep yang

berbeda, sehingga dapat meningkatkan efisiensi dalam memanfaatkan banyak inti prosesor (concurrency).

### 3.1.3.2 Algoritma Depth-First Search (DFS)

Algoritma Depth-First Search (DFS) pada fungsi DFSFindTrees digunakan untuk menelusuri semua kemungkinan pohon resep secara rekursif dan mendalam. Pendekatan ini dimulai dari elemen target dan menyelam ke bawah dengan mengikuti setiap kombinasi resep yang mungkin, hingga mencapai elemen dasar (base elements) yang tidak memiliki resep turunan. Dengan menelusuri jalur satu per satu secara mendalam, DFS dapat membentuk pohon resep yang unik sebelum berpindah ke jalur atau kombinasi lain.

Setiap pemanggilan rekursif dilakukan terhadap kedua bahan dari sebuah resep, yaitu ElementOne dan ElementTwo, sehingga menciptakan struktur pohon biner yang merepresentasikan proses crafting. Setelah pohon dari kedua bahan tersebut terbentuk, keduanya digabung menjadi simpul (node) baru yang mewakili elemen hasil resep tersebut, dan hasil ini dikirimkan melalui channel treeChan. Fungsi ini juga mempertimbangkan batas maksimum jumlah pohon (maxTreeCount) agar pencarian tidak terus berlangsung tanpa kendali.

Keunggulan pendekatan DFS adalah kemampuannya untuk segera menemukan satu pohon yang lengkap tanpa perlu mengevaluasi seluruh cabang lain di tingkat yang sama, berbeda dengan BFS yang menelusuri semua kemungkinan secara melebar terlebih dahulu. Oleh karena itu, DFS sering lebih cepat menemukan satu solusi utuh dalam graf yang sangat bercabang. Namun, DFS juga memiliki risiko mengeksplorasi jalur panjang yang ternyata tidak menghasilkan solusi valid, sehingga dalam implementasi ini, pemrosesan dilakukan secara paralel menggunakan goroutine untuk meningkatkan efisiensi pencarian.

Dengan memanfaatkan pencarian mendalam secara rekursif, DFS pada DFSFindTrees sangat cocok untuk membangun dan mengevaluasi struktur pohon resep secara menyeluruh, terutama ketika solusi yang diinginkan tidak harus optimal, melainkan hanya valid dan dalam jumlah terbatas.

### 3.1.3.3 Algoritma Bidirectional Search

Algoritma Bidirectional Search yang digunakan dalam fungsi BidirectionalFindTrees merupakan pendekatan pencarian dua arah untuk menemukan pohon-pohon resep (recipe trees) yang dapat menghasilkan suatu elemen target. Pencarian ini dilakukan secara simultan dari dua arah: dari elemen target ke bawah (top-down) dan dari elemen dasar (base elements) ke atas (bottom-up).

Pendekatan ini bertujuan untuk mempercepat pencarian dengan membatasi ruang jelajah, yaitu dengan menemukan titik temu (meeting point) di tengah antara dua arah pencarian tersebut. Pada setiap iterasi, antrian dari arah atas (queueUpper) dan bawah (queueLower) diproses secara bergantian. Jika ditemukan node yang telah dikunjungi oleh kedua arah dan cocok dengan target, maka dianggap sebagai titik temu yang valid.

Dari titik ini, pencarian mendalam menggunakan DFS akan dilakukan untuk membentuk pohon resep lengkap. Dengan cara ini, algoritma bidirectional ini mampu menemukan jalur pembuatan elemen secara lebih efisien dibandingkan pencarian satu arah penuh, terutama pada graf yang kompleks dan bercabang banyak karena dapat mengurangi besarnya queue BFS.

### 3.1.4 Visualisasi dan Antarmuka Pengguna

#### 3.1.4.1 Pembuatan Wiki Elemen

Untuk memudahkan pengguna memahami elemen-elemen dan resep dalam game, kami membuat halaman wiki yang menampilkan:

1. Daftar semua elemen dengan gambar dan tier

2. Filter dan pencarian untuk menemukan elemen spesifik
3. Detail resep untuk setiap elemen
4. Navigasi dan pagination untuk mengakses elemen dalam jumlah besar

#### **3.1.4.2 Visualisator Pohon Resep**

Untuk memvisualisasikan hasil pencarian resep, kami membuat komponen visualisator yang menampilkan:

1. Pohon resep dari algoritma BFS atau DFS
2. Animasi proses eksplorasi untuk memahami cara kerja algoritma
3. Statistik performa seperti waktu pencarian dan jumlah node yang dieksplorasi
4. Opsi untuk memilih algoritma dan parameter pencarian

#### **3.1.4.3 Komunikasi Real-time dengan WebSocket**

Untuk memberikan pengalaman yang interaktif, kami mengimplementasikan komunikasi real-time menggunakan WebSocket antara frontend dan backend:

1. Frontend mengirim permintaan pencarian ke backend dengan parameter yang ditentukan
2. Backend melakukan pencarian dan mengirim update progres secara real-time
3. Frontend menampilkan animasi eksplorasi pohon resep berdasarkan data yang diterima
4. Setelah pencarian selesai, hasil akhir ditampilkan kepada pengguna

### **3.1.5 Optimasi Performa**

#### **3.1.5.1 Paralelisasi Pencarian**

Untuk meningkatkan performa pencarian, kami mengimplementasikan paralelisasi menggunakan goroutines dan channels di Go:

1. Setiap resep ditelusuri secara paralel menggunakan goroutine terpisah
2. Hasil dari setiap goroutine dikumpulkan melalui channel
3. Sinkronisasi dilakukan menggunakan mutex dan waitgroup untuk menghindari race condition
4. Visualisasi pencarian masih disajikan secara berurutan kepada pengguna

### **3.1.5.2 Caching dan Pengoptimalan Graf**

Untuk mempercepat pencarian berulang, kami mengimplementasikan beberapa optimasi:

1. Graf elemen disimpan di memori setelah inisialisasi awal
2. Elemen-elemen yang tidak valid dalam hierarki tier di-filter untuk mengurangi ruang pencarian
3. Hasil pencarian dapat dibatasi untuk menghindari komputasi berlebihan

## **3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS**

Untuk mengonversi masalah pencarian resep dalam Little Alchemy 2 menjadi masalah yang dapat diselesaikan menggunakan DFS atau BFS, diperlukan pemetaan yang jelas antara komponen-komponen permainan dengan elemen-elemen graf.

### **3.2.1 Identifikasi Jenis Permasalahan**

Jenis permasalahan yang dihadapi dalam Little Alchemy 2 adalah masalah pencarian resep untuk membuat elemen baru, yang secara natural dapat dimodelkan sebagai masalah graf berarah (directed graph). Permasalahan ini memiliki karakteristik:

1. Hierarki Elemen: Elemen-elemen dalam permainan memiliki hierarki, dimulai dari elemen dasar hingga elemen kompleks yang membutuhkan kombinasi dari elemen lainnya.

2. Kombinasi Biner: Setiap resep selalu melibatkan tepat dua elemen sebagai bahan untuk membuat elemen baru.
3. Pencarian Multi-solusi: Untuk satu elemen target, mungkin terdapat beberapa jalur berbeda untuk membuatnya.

Struktur graf yang digunakan dalam permainan ini tidak memiliki bobot atau cost tertentu pada sisinya. Dengan kata lain, semua kombinasi elemen dianggap memiliki "jarak" yang sama. Yang menjadi fokus optimasi adalah jumlah langkah kombinasi yang diperlukan untuk mencapai elemen target dari elemen-elemen dasar.

### 3.2.2 Identifikasi Simpul (Node)

Dalam konteks Little Alchemy 2, simpul dalam graf mewakili elemen-elemen dalam permainan:

1. Elemen Dasar: Elemen-elemen awal seperti Air, Api, Tanah, dan Udara yang tersedia dari awal permainan. Elemen ini menjadi daun (leaf nodes) dalam pohon resep.
2. Elemen Turunan: Elemen-elemen yang dibuat dengan menggabungkan elemen-elemen lain.
3. Elemen Target: Elemen yang ingin dibuat oleh pengguna dan menjadi akar (root) dari pohon resep.

Setiap simpul memiliki properti penting:

1. Nama: Identifikasi unik untuk elemen
2. Gambar: Representasi visual dari elemen
3. Tier: Tingkat hierarki elemen dalam permainan (0 untuk elemen dasar, angka lebih tinggi untuk elemen turunan)

```
type ElementsGraphNode struct {
    Name           string   `json:"name"`
    ImagePath      string   `json:"image_path"`
    RecipesToMakeThisElement []*Recipe `json:"recipes_to_make_this_element"`
    RecipesToMakeOtherElement []*Recipe `json:"recipes_to_make_other_element"`
    Tier          int      `json:"tier"`
    IsVisited     bool    `json:"is_visited"`
}
```

### 3.2.3 Identifikasi Sisi (Edge)

Sisi dalam graf mewakili resep atau kombinasi antar elemen:

1. Resep: Hubungan yang menunjukkan bahwa dua elemen dapat digabungkan untuk membentuk elemen baru.
2. Arah: Sisi memiliki arah dari elemen-elemen pembentuk menuju elemen hasil kombinasi.

Setiap sisi direpresentasikan oleh struktur Recipe:

```
type Recipe struct {
    ElementOne      *ElementsGraphNode `json:"element_one"`
    ElementTwo      *ElementsGraphNode `json:"element_two"`
    TargetElementName string           `json:"target_element_name"`
}
```

Batasan penting pada sisi adalah elemen-elemen pembentuk harus memiliki tier yang lebih rendah dari elemen hasil kombinasi. Ini mencegah siklus dalam graf dan memastikan logika hierarki elemen tetap terjaga.

```

    ...
    if recipe.ElementOne != nil && recipe.ElementOne.Tier >= node.Tier {
        shouldKeep = false
    }
    if recipe.ElementTwo != nil && recipe.ElementTwo.Tier >= node.Tier {
        shouldKeep = false
    }
}

```

### 3.2.4 Pemetaan untuk Algoritma Depth-First Search (DFS)

Pencarian menggunakan algoritma DFS dilakukan dengan pendekatan rekursif yang menelusuri setiap jalur secara mendalam sebelum beralih ke jalur alternatif lainnya:

1. Simpul Awal: Elemen target yang ingin dibuat oleh pengguna.
2. Ekspansi: Untuk setiap simpul, DFS mengeksplorasi semua resep yang dapat membuat elemen tersebut.
3. Penelusuran Rekursif: Untuk setiap resep, DFS secara rekursif mencari cara untuk membuat elemen-elemen pembentuk.
4. Kondisi Berhenti: Penelusuran berhenti pada elemen dasar atau ketika mencapai jumlah maksimum pohon resep yang diminta.

Komponen DFS	Pemetaan dalam Little Alchemy 2
Nodes	Elemen-elemen dalam permainan (Air, Api, Tanah, dsb.)
Edge	Resep yang menunjukkan kombinasi antar elemen
Goal State	Menemukan jalur dari elemen dasar untuk membuat elemen target
Ekspansi	Menelusuri resep untuk membuat suatu elemen secara rekursif
Path	Rangkaian kombinasi elemen yang membentuk pohon resep

Path Cost	Jumlah langkah kombinasi yang diperlukan
-----------	--

Untuk pencarian multi-solusi, DFS tidak berhenti saat menemukan satu solusi saja. Algoritma akan terus mencari alternatif jalur lain hingga menemukan sejumlah maxTreeCount solusi. Pencarian secara paralel dengan goroutine dilakukan untuk meningkatkan efisiensi.

### 3.2.5 Pemetaan untuk Algoritma Breadth-First Search (BFS)

Pencarian menggunakan algoritma BFS dilakukan dengan melebar dari elemen target hingga mencapai elemen-elemen dasar:

1. Simpul Awal: Elemen target yang ingin dibuat oleh pengguna.
2. Struktur Queue: Menggunakan antrian untuk menyimpan elemen-elemen yang akan dieksplorasi.
3. Ekspansi: Untuk setiap elemen dalam antrian, BFS mengeksplorasi semua resep yang dapat membuat elemen tersebut.
4. Penelusuran Level by Level: BFS memastikan semua elemen pada level hierarki yang sama dieksplorasi sebelum berpindah ke level berikutnya.
5. Pelacakan Prerequisites: BFS memastikan semua prasyarat elemen sudah diproses sebelum mengolah pohon resep untuk elemen tersebut.

Komponen BFS	Pemetaan dalam Little Alchemy 2
Nodes	Elemen-elemen dalam permainan (Air, Api, Tanah, dsb.)
Edge	Resep yang menunjukkan kombinasi antar elemen
Goal State	Menemukan jalur dari elemen dasar untuk membuat elemen target
Queue	Daftar elemen yang akan dieksplorasi selanjutnya

Level	Tingkat hierarki dalam pohon resep
Path	Rangkaian kombinasi elemen yang membentuk pohon resep
Path Cost	Jumlah langkah kombinasi yang diperlukan

Untuk pencarian multi-solusi, BFS juga tidak berhenti saat menemukan solusi pertama. Algoritma akan terus mencari jalur lain dengan jumlah langkah yang sama untuk memberikan alternatif bagi pengguna. BFS menjamin bahwa solusi yang ditemukan memiliki jumlah langkah minimum karena sifatnya yang melebar.

### 3.2.6 Pemetaan untuk Algoritma Bidirectional Search

Pencarian menggunakan algoritma Bidirectional Search dilakukan dengan menelusuri graf dari dua arah secara simultan, mengurangi ruang pencarian dan meningkatkan efisiensi:

1. Simpul Awal Ganda:
  - a. Forward Search: Dimulai dari elemen-elemen dasar (Water, Fire, Earth, Wind).
  - b. Backward Search: Dimulai dari elemen target yang ingin dibuat.
2. Struktur Frontier:
  - a. Menggunakan dua antrian terpisah untuk forward dan backward search.
  - b. Kedua pencarian berlangsung secara bergantian level demi level.
3. Titik Temu:
  - a. Pencarian berhenti ketika ditemukan elemen yang sama di kedua frontier.
  - b. Jalur lengkap dibentuk dengan menggabungkan kedua sub-jalur.
4. Kondisi Terminasi:
  - a. Ketika titik temu ditemukan.
  - b. Ketika kedua pencarian tidak dapat melakukan ekspansi lebih lanjut.

Komponen Bidirectional	Pemetaan dalam Little Alchemy 2
------------------------	---------------------------------

Nodes	Elemen-elemen dalam permainan (Air, Api, Tanah, dsb.)
Edge	Resep yang menunjukkan kombinasi antar elemen
Goal State	Menemukan titik temu antara pencarian maju dan mundur
Forward Frontier	Elemen-elemen yang telah dibuat dari elemen dasar
Backward Frontier	Elemen-elemen yang dapat membuat elemen target
Meeting Point	Elemen penghubung yang ditemukan di kedua frontier
Path	Gabungan jalur dari elemen dasar ke titik temu dan dari titik temu ke elemen target
Path Cost	Total langkah kombinasi dari kedua arah pencarian

Bidirectional Search menawarkan beberapa keunggulan dibandingkan algoritma DFS dan BFS dalam konteks pencarian resep:

1. Efisiensi Pencarian: Dalam kasus ideal, kompleksitas pencarian berkurang dari  $O(b^d)$  menjadi  $O(b^{(d/2)})$ , di mana b adalah faktor percabangan dan d adalah kedalaman pohon. Ini sangat signifikan untuk elemen kompleks dengan pohon resep dalam.
2. Keseimbangan Memori dan Kecepatan: Mengkombinasikan kelebihan BFS dalam menemukan jalur optimal dengan penggunaan memori yang lebih efisien karena tidak perlu menelusuri seluruh ruang pencarian.
3. Optimal untuk Elemen Tingkat Menengah: Sangat efektif untuk elemen yang berada di "tengah" hierarki game, karena titik temu kemungkinan besar ditemukan di level-level tengah.
4. Paralelisasi yang Efektif: Forward dan backward search dapat dijalankan secara paralel dengan goroutine, memanfaatkan kemampuan konkurensi Go.

Selain itu, dalam mengimplementasi algoritma Bidirectional Search memiliki beberapa tantangan:

1. Definisi "Titik Temu": Menentukan kapan dan bagaimana kedua pencarian "bertemu" dapat menjadi rumit karena struktur graf resep yang kompleks.
2. Penggabungan Jalur: Memastikan jalur yang digabungkan dari kedua arah pencarian membentuk pohon resep yang valid dan optimal.
3. Menjaga Efisiensi Memori: Dengan dua frontier yang berkembang, manajemen memori menjadi lebih kritis.
4. Visualisasi Proses: Merepresentasikan pencarian dua arah secara visual kepada pengguna memerlukan pertimbangan UI/UX yang lebih kompleks.

Algoritma Bidirectional Search menawarkan pendekatan yang seimbang antara DFS dan BFS, cocok untuk kasus di mana kedua algoritma tersebut memiliki keterbatasan dalam menemukan resep yang kompleks dengan efisien.

### **3.2.7 Perbandingan Pendekatan DFS dan BFS dalam Konteks Permainan**

Kedua algoritma memiliki karakteristik berbeda yang menawarkan keunggulan masing-masing dalam konteks pencarian resep:

DFS: Lebih cepat menemukan satu solusi karena fokus menggali satu jalur secara mendalam. Cocok untuk menemukan "salah satu" cara membuat elemen target. Namun, tidak menjamin solusi dengan langkah minimum.

BFS: Menjamin solusi dengan jumlah langkah minimum, namun membutuhkan waktu lebih lama dan memori lebih besar karena harus mengeksplorasi semua kemungkinan pada setiap level. Cocok untuk menemukan cara "tersingkat" membuat elemen target.

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

#### 3.3.1 Fitur Fungsional Aplikasi Web yang Dibangun

##### 3.3.1.1 Navigation Bar



Gambar 3.1. Navigation Bar

Navigasi utama aplikasi ini disediakan melalui sebuah navigation bar yang memudahkan pengguna untuk berpindah antar halaman seperti Halaman Utama, Wiki (Elements Encyclopedia), Visualizer, dan About Us.

##### 3.3.1.2 Halaman Wiki (Elements Encyclopedia)

###### 3.3.1.2.1 Halaman Utama

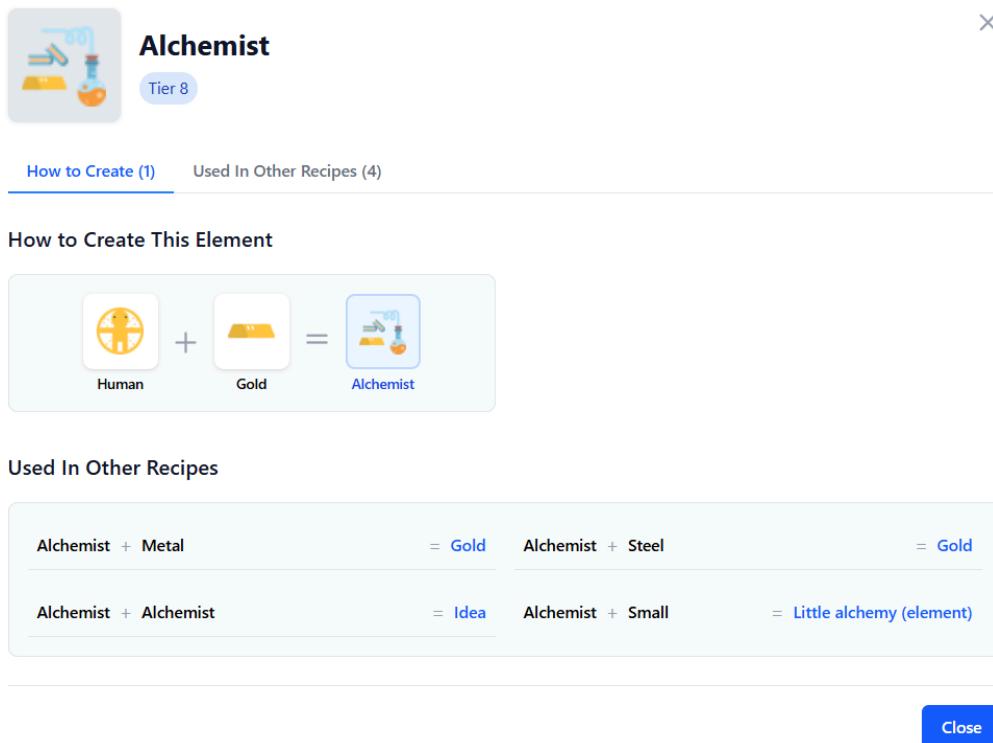
A screenshot of the Elements Encyclopedia Wiki page. The page has a sidebar on the left with categories like "All Elements (749)", "Basic Elements (34)", "Advanced Elements (715)", and "Tiers" (listing tiers from 0 to 11). The main content area is titled "Elements Encyclopedia Wiki" and features a search bar. Below the search bar is a grid of element cards. Each card contains an icon, the element name, and its tier level. For example, Acid rain is at Tier 6, Air is at Tier 0, and Animal is at Tier 7. The grid is organized in a 5x5 pattern.

Gambar 3.2. Halaman Wiki

Halaman ini menampilkan daftar lengkap elemen yang tersedia dalam aplikasi, yang diperoleh melalui proses web scraping dari situs [website Fandom Little Alchemy 2](#). Pengguna dapat melakukan pencarian elemen secara langsung untuk mengetahui apakah elemen tersebut tersedia. Tampilan elemen dapat diubah dalam bentuk grid atau list sesuai preferensi pengguna. Selain itu, tersedia fitur penyortiran (*sort*) berdasarkan nama atau tingkat (*tier*) elemen, serta filter untuk menampilkan elemen *basic* (elemen dasar yang tidak dapat dibuat dari resep) atau

*advanced*. Informasi tentang tier juga disediakan untuk membantu pengguna memahami struktur elemen. Untuk menjaga keteraturan tampilan, halaman ini dilengkapi dengan fitur *pagination* agar daftar elemen tidak ditampilkan sekaligus secara penuh.

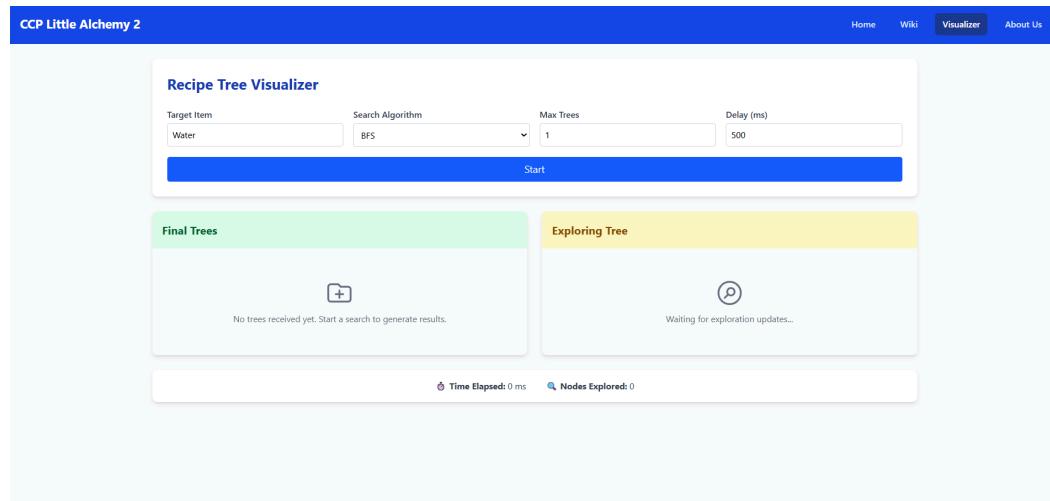
### 3.3.1.2.2 Halaman Elemen Detail



**Gambar 3.3.** Halaman Elemen Detail

Ketika pengguna mengklik salah satu elemen dari halaman utama, mereka akan diarahkan ke halaman detail dari elemen tersebut. Di halaman ini ditampilkan informasi tambahan seperti cara membuat elemen tersebut (jika dapat dibuat), serta elemen apa saja yang bisa dibuat dengan menggunakan elemen tersebut.

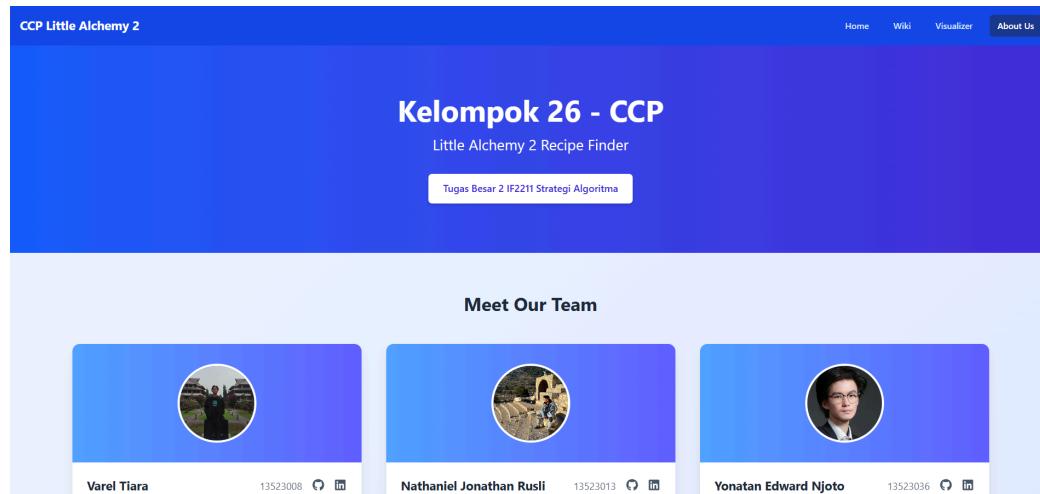
### 3.3.1.3 Halaman Visualizer (Pencari Resep Elemen)



**Gambar 3.4.** Halaman Visualizer

Halaman Visualizer merupakan fitur utama dari aplikasi ini yang memungkinkan pengguna mencari resep pembuatan suatu elemen menggunakan algoritma pencarian graf seperti Breadth-First Search (BFS), Depth-First Search (DFS), dan Bidirectional Search. Pengguna dapat memilih mode pencarian melalui toggle button, yaitu antara mencari satu resep terpendek atau mencari *multiple recipe*. Dalam mode *multiple recipe*, pengguna dapat menentukan jumlah maksimal resep berbeda yang ingin ditemukan untuk suatu elemen, dan pencarian akan dioptimalkan dengan *multithreading* agar efisien. Aplikasi ini juga memastikan bahwa semua elemen yang digunakan dalam pembuatan suatu resep harus memiliki tier lebih rendah dari elemen yang dituju. Visualisasi hasil pencarian ditampilkan dalam bentuk tree yang memperlihatkan kombinasi elemen dari elemen dasar hingga membentuk elemen target. Selama proses pencarian, pengguna dapat melihat *live update* dari progress pencarian dalam bentuk *exploring tree*. Untuk memberikan pengalaman visual yang lebih jelas, pencarian tersebut dilengkapi dengan delay agar visualisasi tidak terlalu cepat. Selain itu, ditampilkan juga waktu yang dibutuhkan untuk pencarian serta jumlah node yang dikunjungi.

### 3.3.1.4 Halaman About Us



**Gambar 3.5.** Halaman About Us

Halaman ini berisi informasi mengenai tim pengembang aplikasi. Di samping itu, halaman ini juga menjelaskan tujuan dari pengembangan aplikasi website ini.

### 3.3.2 Arsitektur Aplikasi Web yang Dibangun

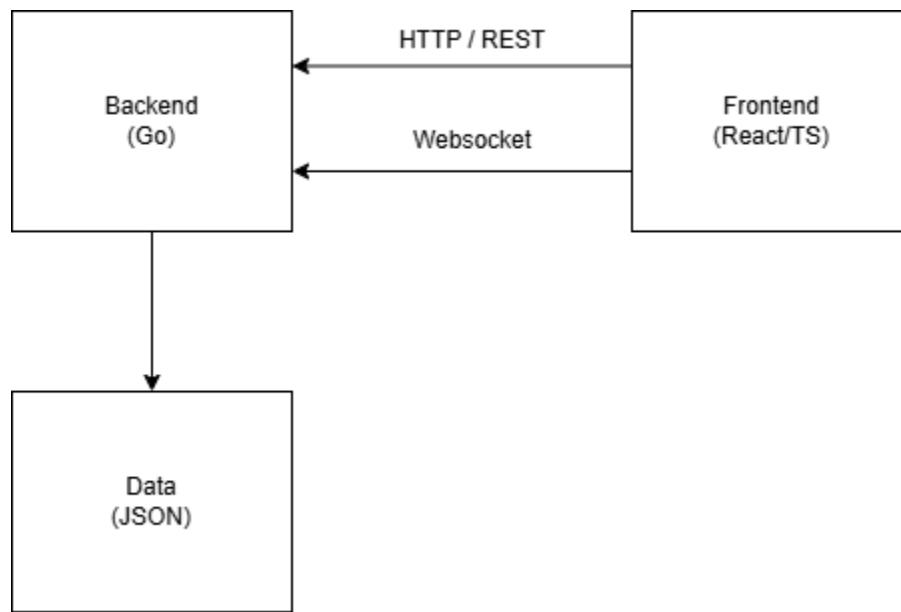
Dalam implementasi aplikasi pencarian resep Little Alchemy 2, kami mengembangkan sebuah aplikasi web dengan arsitektur yang terstruktur dan modular. Bagian ini menjelaskan arsitektur sistem secara keseluruhan serta fitur-fitur fungsional yang diimplementasikan.

#### 3.3.2.1 Arsitektur Sistem Keseluruhan

Aplikasi ini menggunakan arsitektur client-server dengan pemisahan yang jelas antara frontend dan backend:

1. Frontend: Diimplementasikan menggunakan React dan TypeScript, menyediakan antarmuka pengguna yang interaktif
2. Backend: Diimplementasikan menggunakan Go (Golang), menangani pemrosesan data dan algoritma pencarian

3. Komunikasi: Menggunakan kombinasi REST API untuk permintaan statis dan WebSocket untuk komunikasi real-time



**Gambar 3.6.** Diagram Websocket

### 3.3.2.2 Arsitektur Backend

Backend aplikasi diimplementasikan menggunakan bahasa Go, dengan struktur modular yang memisahkan berbagai komponen fungsi:

#### 3.3.2.2.1 Struktur Komponen Backend

1. Models: Menyimpan definisi struktur data dan implementasi algoritma
  - a. element.go: Definisi struktur data elemen
  - b. elements\_graph.go: Implementasi graf untuk merepresentasikan hubungan antar elemen
  - c. bfs.go: Implementasi algoritma Breadth-First Search
  - d. dfs.go: Implementasi algoritma Depth-First Search
2. Controllers: Menangani permintaan dari frontend
  - a. elements.go: Menangani permintaan terkait data elemen
  - b. graph.go: Menangani permintaan terkait struktur graf

- c. websocket.go: Mengelola koneksi WebSocket untuk visualisasi real-time
3. Routes: Mendefinisikan endpoint API yang tersedia
  - a. REST API endpoints untuk data statis
  - b. WebSocket endpoint untuk komunikasi real-time
4. Data: Menyimpan data elemen dan resep dalam format JSON

### **3.3.2.2.2 Alur Data di Backend**

1. Inisialisasi Data:
  - Data elemen dan resep dimuat dari file JSON
  - Graf elemen diinisialisasi dengan menetapkan tier untuk setiap elemen
  - Optimasi resep untuk mencegah siklus dalam graf
2. Eksekusi Algoritma:
  - Permintaan pencarian resep diterima melalui WebSocket
  - Algoritma DFS atau BFS dijalankan berdasarkan parameter yang diterima
  - Setiap kemajuan dalam pencarian dikirim kembali ke frontend melalui WebSocket
  - Hasil akhir berupa pohon resep dikirim setelah algoritma selesai

### **3.3.2.3 Arsitektur Frontend**

Frontend aplikasi dibangun menggunakan React dan TypeScript, dengan pendekatan component-based yang modular:

#### **3.3.2.3.1 Struktur Halaman Frontend**

1. Layout: Komponen dasar yang menyediakan navbar dan footer konsisten di seluruh aplikasi
  - a. Layout.tsx: Wrapper komponen untuk semua halaman
2. Halaman Utama:

- a. Home.tsx: Halaman beranda yang menampilkan pengantar aplikasi
  - b. Wiki.tsx: Ensiklopedia elemen dengan fitur filter, pencarian, dan detail
  - c. Visualizer.tsx: Visualisasi hasil pencarian algoritma dan animasi eksekusi
  - d. AboutUs.tsx: Informasi tentang pengembang dan proyek
3. Komponen Visualisasi:
- a. Komponen pohon resep untuk menampilkan hasil pencarian
  - b. Animasi real-time untuk memvisualisasikan proses eksplorasi algoritma

### 3.3.2.3.2 Manajemen State dan Routing

1. State Management:
  - a. Menggunakan React hooks (useState, useEffect) untuk manajemen state lokal
  - b. Berbagi state antar komponen melalui props
2. Routing:
  - a. Menggunakan React Router untuk navigasi antar halaman
  - b. Definisi rute dalam App.tsx:

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Layout><Home /></Layout>} />
        <Route path="/Wiki" element={<Layout><Wiki /></Layout>} />
        <Route path="/Visualizer" element={<Layout><Visualizer /></Layout>} />
        <Route path="/AboutUs" element={<Layout><AboutUs /></Layout>} />
      </Routes>
    </Router>
  );
}
```

### **3.3.2.4 Mekanisme Komunikasi Frontend-Backend**

Aplikasi menggunakan dua jenis komunikasi yang berbeda untuk kebutuhan yang berbeda:

#### **3.3.2.4.1 REST API**

REST API digunakan untuk komunikasi statis dan permintaan data yang tidak memerlukan update real-time:

- Endpoint Data Elemen: GET /api/elements: Mengambil semua data elemen dan resep. Digunakan oleh halaman Wiki untuk menampilkan ensiklopedia elemen

#### **3.3.2.4.2 WebSocket**

WebSocket digunakan untuk komunikasi real-time terutama untuk visualisasi proses algoritma:

- Koneksi Visualizer:
  - ws://\${BACKEND\_BASE\_URL}/ws: Endpoint
  - WebSocket untuk pencarian resep
  - Mengirim parameter pencarian ke backend
  - Menerima update status pencarian secara real-time
  - Menerima hasil akhir setelah algoritma selesai

## **3.4 Ilustrasi Kasus**

Untuk dapat menggunakan pencarian resep elemen, pertama-tama masuk ke dalam halaman Visualizer yang terdapat pada navigation bar. Kemudian, kasus pencarian resep suatu elemen dapat dianalisis dengan mencoba memasukkan berbagai parameter input seperti Target Item, Search Algorithm, Max Trees, dan Delay. Dalam skenario pencarian elemen "Clay", pengguna menuliskan "Clay" pada select option Target Item. Searching algorithm yang ingin digunakan dapat dipilih pada select option Search Algorithm (pada kasus ini menggunakan BFS). Sebagai contoh kasus, untuk membatasi hasil pencarian pohon resep,

parameter maxTree diisi dengan nilai 1 dengan delay 10 ms pada select option Max Trees dan Delay secara berurutan.

The screenshot shows the 'Recipe Tree Visualizer' interface. It has four input fields: 'Target Item' (Clay), 'Search Algorithm' (BFS), 'Max Trees' (1), and 'Delay (ms)' (10). Below these fields is a large blue 'Start' button.

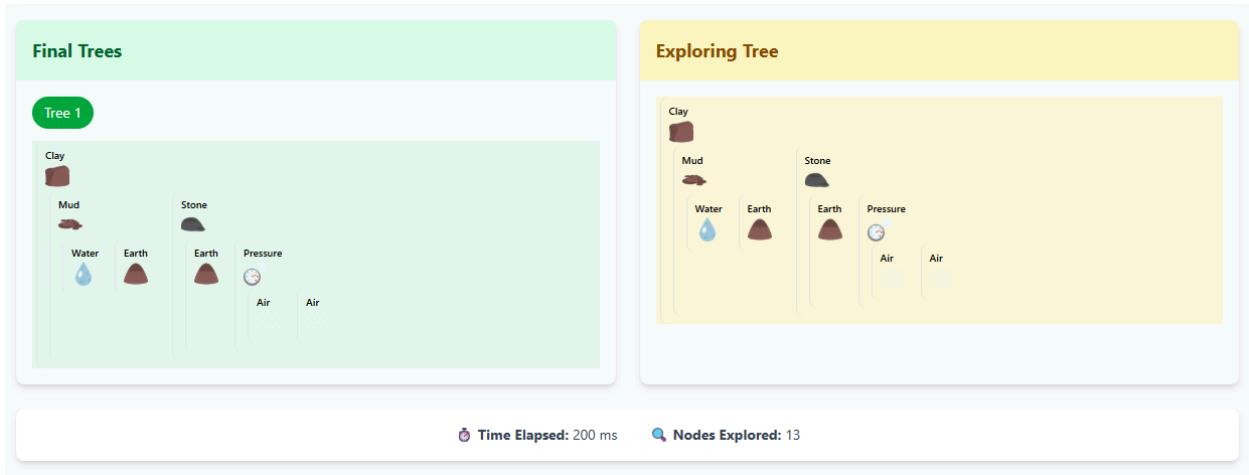
**Gambar 3.7.** Input Recipe Tree Visualizer

The screenshot shows the 'Search Algorithm' dropdown menu open, displaying four options: BFS, DFS, Bidirectional, and Start. The 'BFS' option is selected and highlighted with a blue background. The other options are in a greyed-out state. The rest of the interface is identical to Gambar 3.7.

**Gambar 3.8.** Pilihan Search Algorithm

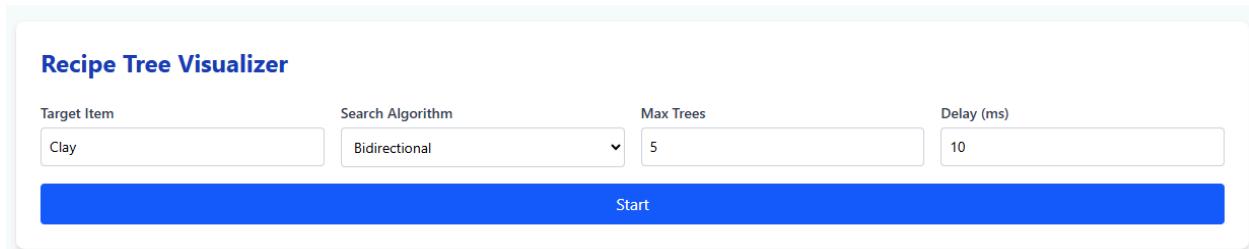
Saat permintaan masuk, backend menginisialisasi variabel globalNodeCounter=0 dan mencatat waktu mulai untuk pengukuran performa. Fungsi BFSFindTrees kemudian memeriksa kondisi awal: "Clay" bukan elemen dasar dan memiliki resep pembuatan. Setelah memastikan bahwa elemen "Clay" memiliki resep pembuatan, algoritma BFS akan memulai pencarian dari node target ke arah base elements. Karena maxTree diatur ke 1, hanya satu solusi pohon resep yang akan dicari dan ditampilkan. Dalam proses ini, BFS menjelajahi semua kemungkinan kombinasi secara tier-wise dan menambahkan node-node yang telah diekspansi ke dalam queue pencarian.

Pada **Gambar 3.9.**, tampak tampilan Final Tree pada sisi sebelah kiri yang menunjukkan pohon resep lengkap dari kombinasi elemen dasar menuju elemen "Clay". Pada sisi kanan, tampak Exploring Tree yang merupakan representasi visual dari proses eksplorasi node secara bertahap yang terjadi selama BFS berjalan. Visualisasi ini diperbarui secara real-time menggunakan WebSocket dan menampilkan perkembangan tree seiring pencarian berlangsung.



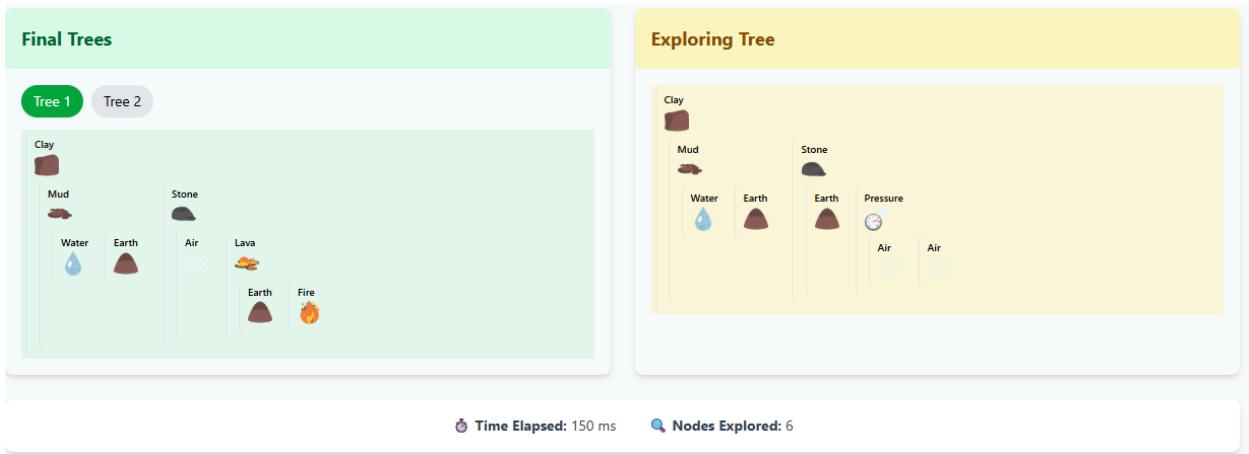
**Gambar 3.9.** Hasil Pencarian Elemen

Pada bagian bawah visualisasi juga ditampilkan statistik pencarian yang terdiri atas Time Elapsed (total waktu yang dibutuhkan untuk menyelesaikan pencarian tree) dan Nodes Explored (total node yang dikunjungi selama proses pencarian). Dengan delay 10 ms, pengguna dapat mengikuti proses eksplorasi node satu per satu membuat fitur ini berguna untuk memahami jalannya algoritma secara dinamis dan interaktif.

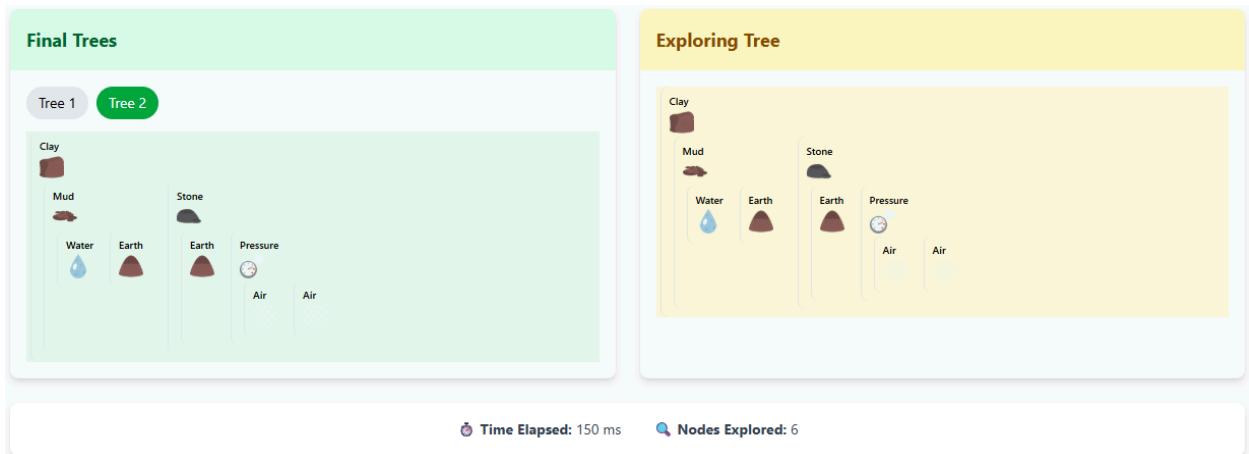


**Gambar 3.10.** Pencarian *Multi Recipes*

Contoh kasus selanjutnya adalah pencarian multi recipes untuk elemen “Clay”. Untuk pencarian multiple recipes (atau recipe trees), select option Max Trees dapat diisi dengan nilai lebih dari 1, atau pada kasus ini diisi dengan nilai 5. Pada kasus ini, Search Algorithm yang digunakan adalah Bidirectional yang memanggil function BidirectionalSearchTrees (untuk kasus DFS akan memanggil function DFSFindTrees). Namun, seperti yang terlihat pada gambar di bawah, hasil resep yang muncul hanyalah 2 walaupun Max Trees yang diminta pengguna adalah sejumlah 5. Tampak pada Tree 1 dan Tree 2, terdapat perbedaan unik pada recipe tree, menunjukkan berbagai kemungkinan yang berbeda-beda dalam membuat suatu elemen.



**Gambar 3.11.** Hasil Pertama *Multi Recipes*



**Gambar 3.12.** Hasil Kedua *Multi Recipes*

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Spesifikasi Teknis Program

##### 4.1.1 Struktur Data

Struktur Data	Tampilan Kode
<p>Element</p> <p>Representasi sederhana dari elemen:</p> <ol style="list-style-type: none"><li>1. Name: nama elemen.</li><li>2. Recipes: resep-resep yang bisa digunakan untuk membuat elemen ini, dalam bentuk array pasangan string.</li><li>3. ImagePath: path ke gambar elemen.</li></ol>	<pre>type Element struct {     Name      string      `json:"name"`     Recipes  []string    `json:"recipes"`     ImagePath string     `json:"image_path"` }</pre>
<p>ElementsGraphNode</p> <p>Node dalam graph elemen:</p> <ol style="list-style-type: none"><li>1. Name, ImagePath: identitas dan gambar elemen.</li><li>2. RecipesToMakeThisElement: daftar resep untuk membuat elemen ini.</li><li>3. RecipesToMakeOtherElement: daftar resep yang menggunakan elemen ini sebagai bahan.</li><li>4. Tier: tingkat (seberapa jauh elemen ini dari elemen dasar).</li><li>5. IsVisited: apakah sudah dikunjungi dalam traversal.</li><li>6. MadeFrom: map yang mencatat dari elemen apa saja ini dibuat.</li></ol>	<pre>type ElementsGraphNode struct {     Name          string      `json:"name"`     ImagePath    string      `json:"image_path"`     RecipesToMakeThisElement []*Recipe `json:"recipes_to_make_this_element"`     RecipesToMakeOtherElement []*Recipe `json:"recipes_to_make_other_element"`     Tier         int         `json:"tier"`     IsVisited    bool        `json:"is_visited"`     MadeFrom     map[string]bool `json:"made_from"` }</pre>

<h3>Recipe</h3> <p>Representasi satu resep:</p> <ol style="list-style-type: none"> <li>1. ElementOne dan ElementTwo: dua bahan penyusun resep.</li> <li>2. TargetElementName: hasil dari pencampuran dua bahan tersebut.</li> </ol>	<pre>type Recipe struct {     ElementOne      *ElementsGraphNode `json:"element_one"`     ElementTwo      *ElementsGraphNode `json:"element_two"`     TargetElementName string           `json:"target_element_name"` }</pre>
<h3>RecipeTreeNode</h3> <p>Node dalam pohon resep (bisa rekursif):</p> <ol style="list-style-type: none"> <li>1. Name, ImagePath: elemen yang dihasilkan.</li> <li>2. Element1, Element2: bahan-bahan penyusunnya, jika ada. Bisa nil jika elemen dasar.</li> </ol>	<pre>type RecipeTreeNode struct {     Name        string      `json:"name"`     ImagePath   string      `json:"image_path"`     Element1   *RecipeTreeNode `json:"element_1,omitempty"`     Element2   *RecipeTreeNode `json:"element_2,omitempty"` }</pre>
<h3>SafeRecipe</h3> <p>Versi Recipe yang aman untuk dikirim via JSON (tanpa nested struct pointer):</p> <p>Hanya menyimpan nama string, cocok untuk serialisasi.</p>	<pre>type SafeRecipe struct {     ElementOneName string `json:"element_one"`     ElementTwoName string `json:"element_two,omitempty"` // omit if nil     TargetElementName string `json:"target_element_name"` }</pre>
<h3>SafeElementNode</h3> <p>Versi aman dari ElementsGraphNode (tanpa nested pointer), cocok untuk dikirim ke frontend:</p> <p>Menyederhanakan data untuk konsumsi UI atau API.</p>	<pre>type SafeElementNode struct {     Name        string      `json:"name"`     ImagePath   string      `json:"image_path"`     RecipesToMakeThisElement []SafeRecipe `json:"recipes_to_make_this_element"`     RecipesToMakeOtherElement []SafeRecipe `json:"recipes_to_make_other_element"` }</pre>

<h3>RecipeTreeRequest</h3> <p>Digunakan sebagai request body saat ingin membangun pohon resep:</p> <ol style="list-style-type: none"> <li>1. Target: nama elemen tujuan yang ingin dibuat.</li> <li>2. Mode: mode eksplorasi pohon, misalnya BFS, DFS, atau lainnya.</li> <li>3. MaxTreeCount: jumlah maksimum pohon resep yang ingin dihasilkan.</li> <li>4. DelayMs: jeda waktu (dalam milidetik) antara setiap langkah eksplorasi, biasanya digunakan untuk visualisasi.</li> </ol>	<pre><code>type RecipeTreeRequest struct {     Target      string `json:"target"`     Mode       string `json:"mode"`     MaxTreeCount int    `json:"max_tree_count"`     DelayMs     int    `json:"delay_ms"` }</code></pre>
<h3>TreeUpdate</h3> <p>Digunakan untuk mengirim update selama proses eksplorasi pohon resep:</p> <ol style="list-style-type: none"> <li>1. ExploringTree: node pohon saat ini yang sedang dieksplorasi.</li> <li>2. DurationMs: waktu yang telah digunakan sejauh ini.</li> <li>3. NodesExplored: jumlah node (resep) yang telah dieksplorasi.</li> </ol>	<pre><code>type TreeUpdate struct {     ExploringTree *models.RecipeTreeNode `json:"exploring_tree"`     DurationMs   int                      `json:"duration_ms"`     NodesExplored int32                  `json:"nodes_explored"` }</code></pre>
<h3>FinalResponse</h3> <p>Response akhir ketika eksplorasi pohon resep selesai:</p> <ol style="list-style-type: none"> <li>1. Trees: daftar pohon resep yang ditemukan.</li> <li>2. DurationMs: total waktu eksplorasi.</li> <li>3. NodesExplored: jumlah total node yang dijelajahi.</li> </ol>	<pre><code>type FinalResponse struct {     Trees        []*models.RecipeTreeNode `json:"trees"`     DurationMs  int                        `json:"duration_ms"`     NodesExplored int32                  `json:"nodes_explored"` }</code></pre>

<p><b>QueueItem BFS</b></p> <p>Digunakan dalam antrian (queue) eksplorasi untuk membangun pohon resep:</p> <ol style="list-style-type: none"> <li>1. ElementName: elemen yang sedang diproses.</li> <li>2. Level: tingkat kedalaman dalam pohon.</li> <li>3. TreeSoFar: pohon yang telah dibentuk sejauh ini.</li> <li>4. IsComplete: apakah pohon ini sudah selesai dibentuk.</li> </ol>	<pre>type QueueItem struct {     ElementName string     Level       int     TreeSoFar   *RecipeTreeNode     IsComplete  bool }</pre>
<p><b>QueueItem Bidirectional</b></p> <p>Dipakai dalam konteks traversal graph elemen (mungkin BFS):</p> <ol style="list-style-type: none"> <li>1. Element: node graf elemen yang sedang diproses.</li> <li>2. From: nama elemen asal (parent).</li> </ol>	<pre>type QueueItem struct {     Element *ElementsGraphNode     From    string }</pre>

#### 4.1.2 Fungsi

Fungsi	Tampilan Kode
<p><code>withCORS(handler http.Handler) http.Handler</code></p> <p>Middleware yang menambahkan header CORS ke setiap response.</p> <ol style="list-style-type: none"> <li>1. Memastikan request dari <code>http://localhost:4001</code> bisa diakses oleh frontend.</li> <li>2. Menangani request OPTIONS secara khusus.</li> <li>3. Mengembalikan handler baru yang dibungkus CORS.</li> </ol>	<pre>func withCORS(handler http.Handler) http.Handler {     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {         w.Header().Set("Access-Control-Allow-Origin", "http://localhost:4001")         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 == http.MethodOptions {             w.WriteHeader(http.StatusOK)             return         }         handler.ServeHTTP(w, r)     }) }</pre>

```
signallerFn := func(...) { ... }
```

Callback yang dipanggil selama proses pencarian tree untuk mengirim update berkala.

1. Digunakan untuk mengisi channel updateChan dengan update terbaru.
2. Disinkronkan dengan mutex untuk thread-safety.

```
...  
signallerFn := func(  
    exploringTree *models.RecipeTreeNode,  
    durationMs int,  
    nodesExplored int32,  
) {  
    if req.DelayMs > 0 {  
        updateMu.Lock()  
        latestUpdate = &TreeUpdate{  
            ExploringTree: exploringTree,  
            DurationMs: durationMs,  
            NodesExplored: nodesExplored,  
        }  
        updateMu.Unlock()  
  
        select {  
            case updateChan <- *latestUpdate:  
            default:  
        }  
    }  
}
```

```
if req.DelayMs > 0 { ... go  
    func() { ... }() }
```

Mengirim update tree ke klien WebSocket setiap interval tertentu.

1. Mengambil update dari channel updateChan.
2. Mengirim JSON update menggunakan WebSocket ke klien.

```
...  
if req.DelayMs > 0 {  
    updateWg.Add(1)  
    go func() {  
        defer updateWg.Done()  
        ticker := time.NewTicker(time.Duration(req.DelayMs) *  
            time.Millisecond)  
        defer ticker.Stop()  
  
        for update := range updateChan {  
            <-ticker.C  
            writeMu.Lock()  
            if err := conn.WriteJSON(update); err != nil {  
                log.Println("Write error:", err)  
                writeMu.Unlock()  
                return  
            }  
            writeMu.Unlock()  
        }  
    }  
}
```

```
BFSFindTrees(...)  
([[]*RecipeTreeNode, error)
```

Mencari pohon resep (recipe tree) dari suatu node target dalam graf.

1. Mengecek apakah target valid dan apakah elemen dasarnya memiliki resep.
  2. Melakukan BFS dengan queue untuk menjelajahi elemen penyusun.
  3. Untuk setiap kombinasi elemen pembentuk (dari resep), membangun sub-tree dan menyusun RecipeTreeNode sampai target terbentuk.
  4. Menggunakan channel dan goroutine untuk menulis hasil secara paralel ke resultChan.
  5. Dan mengembalikan slice hasil pohon resep.

```

    // This is a placeholder for your application's main class.
    // You should replace it with your own Application.java file.
    public static void main(String[] args) {
        SpringApplication app = new SpringApplication(HelloWorldApplication.class);
        app.setPort(8080);
        app.run(args);
    }
}

// This is a placeholder for your application's main class.
// You should replace it with your own Application.java file.
public class HelloWorldApplication {

    public static void main(String[] args) {
        SpringApplication app = new SpringApplication(HelloWorldApplication.class);
        app.setPort(8080);
        app.run(args);
    }
}

```

## BidirectionalFindTrees(...) ([]\*RecipeTreeNode, error)

Menemukan pohon resep (recipe tree) dari dua arah (atas dan bawah) menggunakan pencarian dua arah (bidirectional search).

1. Cek kondisi awal (base case).
2. Bangun dua antrian pencarian: queueUpper (dari target ke dasar) dan queueLower (dari dasar ke target).
3. Iterasi selama dua antrian tidak kosong dan jumlah pohon belum mencapai maxTreeCount.
4. Jalankan processUpper dan processLower untuk memperluas pencarian.
5. Bila ditemukan simpul yang sama pada kedua arah, panggil DFSFindTrees dan gabungkan hasilnya.
6. Kembalikan hasil pencarian berupa array RecipeTreeNode.

```
• • •  
func BidirectionalFindTrees(  
    rootRecipeTree *RecipeTreeNode,  
    targetGraphNode *ElementsGraphNode,  
    maxTreeCount int,  
    signalTreeChange func(*RecipeTreeNode, int, int32),  
    globalStartTime time.Time,  
    globalNodeCounter *int32,  
    delayMs int,  
) ([]*RecipeTreeNode, error) {  
    if targetGraphNode == nil {  
        return nil, fmt.Errorf("targetGraphNode is nil")  
    }  
    if len(targetGraphNode.RecipesToMakeThisElement) == 0 ||  
        !isValidElement(targetGraphNode.Name) {  
        node := &RecipeTreeNode{  
            Name: targetGraphNode.Name,  
            ImagePath: GetImagePath(targetGraphNode.ImagePath),  
        }  
        return []*RecipeTreeNode{node}, nil  
    }  
  
    visitedUpper := make(map[string]bool)  
    visitedLower := make(map[string]bool)  
    seenMeeting := make(map[string]bool)  
  
    queueUpper := []*QueueItem{{Element: targetGraphNode}}  
    queueLower := []*QueueItem{}  
    for _, base := range GetBaseElements() {  
        if targetGraphNode.IsThisMadeFrom(base) {  
            if node, ok := GetElementsGraphNodeByName(base); ok {  
                queueLower = append(queueLower, &QueueItem{Element: node})  
            }  
        }  
    }  
  
    var resultTrees []*RecipeTreeNode  
  
    for len(queueUpper) > 0 && len(queueLower) > 0 && len(resultTrees) <  
        maxTreeCount {  
        if delayMs > 0 {  
            time.Sleep(time.Duration(delayMs) * time.Millisecond)  
        }  
  
        nQueueUpper, newUpperNames := processUpper(queueUpper, visitedUpper)  
        for _, name := range newUpperNames {  
            if visitedLower[name] && name == targetGraphNode.Name {  
                if _, already := seenMeeting[name]; already {  
                    continue  
                }  
                seenMeeting[name] = true  
                if node, ok := GetElementsGraphNodeByName(name); ok {  
                    treesFromDFS, err := DFSFindTrees(nil, node, maxTreeCount,  
                        signalTreeChange, globalStartTime, globalNodeCounter, delayMs)  
                    if err == nil {  
                        resultTrees = append(resultTrees, treesFromDFS, targetGraphNode.Name, maxTreeCount)  
                        if len(resultTrees) >= maxTreeCount {  
                            return resultTrees, nil  
                        }  
                    }  
                }  
            }  
        }  
        queueUpper = nQueueUpper  
  
        if delayMs > 0 {  
            time.Sleep(time.Duration(delayMs) * time.Millisecond)  
        }  
  
        nQueueLower, newLowerNames := processLower(queueLower, visitedLower)  
        for _, name := range newLowerNames {  
            if visitedUpper[name] && name == targetGraphNode.Name {  
                if _, already := seenMeeting[name]; already {  
                    continue  
                }  
                seenMeeting[name] = true  
                if node, ok := GetElementsGraphNodeByName(name); ok {  
                    treesFromDFS, err := DFSFindTrees(nil, node, maxTreeCount,  
                        signalTreeChange, globalStartTime, globalNodeCounter, delayMs)  
                    if err == nil {  
                        resultTrees = append(resultTrees, treesFromDFS, targetGraphNode.Name, maxTreeCount)  
                        if len(resultTrees) >= maxTreeCount {  
                            return resultTrees, nil  
                        }  
                    }  
                }  
            }  
        }  
        queueLower = nQueueLower  
    }  
    return resultTrees, nil  
}
```

```

processUpper(queue
[]*QueueItem, visited
map[string]bool)
([]*QueueItem, []string) {}

```

Memperluas pencarian dari atas ke bawah (target ke bahan-bahan pembentuknya)

1. Tandai node sebagai telah dikunjungi.
2. Tambahkan dua elemen dari setiap resep sebagai kandidat eksplorasi selanjutnya.
3. Kembalikan nextQueue dan nama-nama elemen yang diproses.

```

func processUpper(queue []*QueueItem, visited map[string]bool) ([]*QueueItem,
[]string) {
    nextQueue := []*QueueItem{}
    produced := []string{}
    for _, item := range queue {
        node := item.Element
        if visited[node.Name] {
            continue
        }
        visited[node.Name] = true
        produced = append(produced, node.Name)
        for _, recipe := range node.RecipesToMakeThisElement {
            nextQueue = append(nextQueue, &QueueItem{Element: recipe.ElementOne})
            nextQueue = append(nextQueue, &QueueItem{Element: recipe.ElementTwo})
        }
    }
    return nextQueue, produced
}

```

```

processLower(queue
[]*QueueItem, visited
map[string]bool)
([]*QueueItem, []string) {}

```

Memperluas pencarian dari bawah ke atas (bahan dasar ke target)

1. Tandai node sebagai telah dikunjungi.
2. Dari node saat ini, tambahkan node hasil pembuatan (target element) ke antrian selanjutnya.
3. Kembalikan nextQueue dan nama-nama elemen yang diproses.

```

func processLower(queue []*QueueItem, visited map[string]bool) ([]*QueueItem,
[]string) {
    nextQueue := []*QueueItem{}
    produced := []string{}
    for _, item := range queue {
        node := item.Element
        if visited[node.Name] {
            continue
        }
        visited[node.Name] = true
        produced = append(produced, node.Name)
        for _, recipe := range node.RecipesToMakeOtherElement {
            if targetNode, ok := GetElementsGraphNodeByName(recipe.TargetElementName); ok {
                nextQueue = append(nextQueue, &QueueItem{Element: targetNode})
            }
        }
    }
    return nextQueue, produced
}

```

```
appendAllValidTargetTrees(existing []*RecipeTreeNode,
    newTrees
    []*RecipeTreeNode, target
    string, maxTreeCount int)
    []*RecipeTreeNode {}
```

Menambahkan hasil pencarian pohon ke dalam list hasil jika belum duplikat.

1. Cek apakah setiap pohon yang ditemukan sudah ada di list sebelumnya (dengan membuat key dari ImagePath + Name + struktur pohon).
2. Jika belum ada, clone pohon dan tambahkan ke hasil.
3. Berhenti jika jumlah pohon mencapai maxTreeCount.

```
func appendAllValidTargetTrees(existing []*RecipeTreeNode, newTrees
    []*RecipeTreeNode, target string, maxTreeCount int) []*RecipeTreeNode {
    existingSet := make(map[string]bool)
    for _, t := range existing {
        if t.Name == target {
            existingSet[t.ImagePath+t.Name+treeToString(t)] = true
        }
    }
    for _, t := range newTrees {
        if t.Name == target {
            key := t.ImagePath + t.Name + treeToString(t)
            if !existingSet[key] {
                existing = append(existing, t.clone())
                existingSet[key] = true
                if len(existing) >= maxTreeCount {
                    break
                }
            }
        }
    }
    return existing
}
```

```
treeToString(tree
    *RecipeTreeNode) string {}}
```

Mengubah struktur pohon menjadi string sebagai identifikasi unik.

Format string Name(Element1, Element2) hingga ke daun pohon (leaf).

```
func treeToString(tree *RecipeTreeNode) string {
    if tree == nil {
        return ""
    }
    return tree.Name + "(" + treeToString(tree.Element1) + "," +
    treeToString(tree.Element2) + ")"
}
```

```
GetBaseElements() []string
{}}
```

Mengambil list baseElements.

```
func GetBaseElements() []string {
    return baseElements
}
```

## DFSFindTrees(...) ([]\*RecipeTreeNode, error)

Melakukan pencarian secara rekursif dan paralel (DFS - Depth-First Search) untuk membangun Recipe Tree dari suatu node target pada graph elemen hingga maksimal maxTreeCount pohon ditemukan.

1. Jika targetGraphNode adalah elemen dasar (tidak memiliki resep), maka fungsi langsung mengembalikan node tunggal sebagai hasil.
2. Jika elemen target memiliki resep:
  - a. Untuk setiap resep yang dapat menghasilkan node target:
    - i. Lakukan DFS ke elemen-elemen penyusunnya (ElementOne dan ElementTwo).
    - ii. Kombinasikan hasilnya menjadi pohon baru RecipeTreeNode.
    - iii. Kirim hasil pohon tersebut ke channel jika belum mencapai batas maksimum maxTreeCount.
3. Menggunakan goroutine untuk pemrosesan paralel, sinkronisasi menggunakan sync.WaitGroup dan sync.Mutex.
4. Mengumpulkan hasil dari channel dan mengembalikannya sebagai slice []\*RecipeTreeNode.

```

    ...
func DFSFindTrees(
    rootRecipeTree *RecipeTreeNode,
    targetGraphNode *ElementsGraphNode,
    maxTreeCount int,
    signalTreeChange func(*RecipeTreeNode, int, int32),
    globalStartTime time.Time,
    globalNodeCounter *int32,
    delayMs int,
) ([]*RecipeTreeNode, error) {
    if rootRecipeTree == nil {
        return nil, fmt.Errorf("targetGraphNode is nil")
    }

    if len(targetGraphNode.RecipesToMakeThisElement) == 0 || !IsBaseElement(targetGraphNode.Name) {
        node := &RecipeTreeNode{
            Name: targetGraphNode.Name,
            ImagePath: GetImagePath(targetGraphNode.ImagePath),
        }
        return []*RecipeTreeNode{node}, nil
    }

    var (
        result []*RecipeTreeNode
        mu    sync.Mutex
        wg   sync.WaitGroup
        count = 0
    )

    treeChan := make(chan *RecipeTreeNode, maxTreeCount)

    for _, recipe := range targetGraphNode.RecipesToMakeThisElement {
        wg.Add(1)
        go func(r *Recipe) {
            defer wg.Done()

            if delayMs > 0 {
                time.Sleep(time.Duration(delayMs) * time.Millisecond)
            }

            atomic.AddInt32(globalNodeCounter, 1)

            leftTrees, err1 := DFSFindTrees(nil, r.ElementOne, maxTreeCount,
                signalTreeChange, globalStartTime, globalNodeCounter, delayMs)
            if err1 != nil {
                return
            }

            rightTrees, err2 := DFSFindTrees(nil, r.ElementTwo, maxTreeCount,
                signalTreeChange, globalStartTime, globalNodeCounter, delayMs)
            if err2 != nil {
                return
            }

            for _, lt := range leftTrees {
                mu.Lock()
                if count >= maxTreeCount {
                    mu.Unlock()
                    break
                }
                mu.Unlock()

                for _, rt := range rightTrees {
                    mu.Lock()
                    if count >= maxTreeCount {
                        mu.Unlock()
                        break
                    }
                    mu.Unlock()

                    root := &RecipeTreeNode{
                        Name: targetGraphNode.Name,
                        ImagePath: GetImagePath(targetGraphNode.ImagePath),
                        Element1: lt,
                        Element2: rt,
                    }

                    if signalTreeChange != nil {
                        func() {
                            defer func() {
                                if r := recover(); r != nil {
                                    if r == recover() {
                                        return
                                    }
                                }
                            }()
                            signalTreeChange(
                                root,
                                int(time.Since(globalStartTime).Milliseconds()),
                                atomic.LoadInt32(globalNodeCounter),
                            )
                        }()
                    }

                    treeChan <- root
                    count++
                    mu.Unlock()
                }
            }(recipe)
        }
    }

    wg.Wait()
    if count <= maxTreeCount {
        mu.Unlock()
        break
    }
    mu.Unlock()
}

go func() {
    wg.Wait()
    close(treeChan)
}()

for tree := range treeChan {
    result = append(result, tree)
    if len(result) >= maxTreeCount {
        break
    }
}

if len(result) == 0 {
    return nil, fmt.Errorf("no valid trees found for %s",
        targetGraphNode.Name)
}

return result, nil
}

```

```
LoadElementsFromJSON(filePath string) ([]Element, error)
{}
```

Memuat data elemen dari file JSON dan mengembalikannya sebagai slice []Element.

1. Membuka file dari path yang diberikan.
2. Meng-decode isi JSON menjadi slice []Element.
3. Jika terjadi error saat membuka file atau decode, error akan dikembalikan.

```
func LoadElementsFromJSON(filePath string) ([]Element, error) {
    file, err := os.Open(filePath)
    if err != nil {
        return nil, err
    }
    defer file.Close()

    decoder := json.NewDecoder(file)
    var elements []Element
    err = decoder.Decode(&elements)
    if err != nil {
        return nil, err
    }

    return elements, nil
}
```

```
GetElementsGraphNodeByName(name string) (*ElementsGraphNode, bool)
{}
```

Mencari node graph berdasarkan nama elemen.

1. Mencari key name dalam map global nameToNode.
2. Mengembalikan pointer ke node jika ditemukan, serta boolean exists.

```
func GetElementsGraphNodeByName(name string) (*ElementsGraphNode, bool) {
    node, exists := nameToNode[name]
    return node, exists
}
```

```
(node *ElementsGraphNode)
IsThisMadeFrom(element string) bool {}
```

Mengecek apakah suatu node elemen dibuat dari elemen lain tertentu.

```
func (node *ElementsGraphNode) IsThisMadeFrom(element string) bool {
    if node.MadeFrom == nil {
        return false
    }
    return node.MadeFrom[element]
}
```

```
GetJSONDTONodes()
[]ElementsGraphNodeDTO
{}
```

Mengembalikan semua node dari nameToNode dalam bentuk Data Transfer Object (DTO) ElementsGraphNodeDTO.

1. Iterasi seluruh node dalam nameToNode.
2. Konversi setiap node dan resep-resepnya ke bentuk DTO.
3. Kembalikan slice hasil konversi.

```
func GetJSONDTONodes() []ElementsGraphNodeDTO {
    nameToNodeList := make([]ElementsGraphNodeDTO, 0)
    for _, node := range nameToNode {
        dto := ElementsGraphNodeDTO{
            Name:                      node.Name,
            ImagePath:                 GetImagePath(node.ImagePath),
            RecipesToMakeThisElement:  make([]RecipeDTO,
                len(node.RecipesToMakeThisElement)),
            RecipesToMakeOtherElement: make([]RecipeDTO,
                len(node.RecipesToMakeOtherElement)),
            IsVisited:                 node.IsVisited,
            Tier:                      node.Tier,
        }

        for i, recipe := range node.RecipesToMakeThisElement {
            dto.RecipesToMakeThisElement[i] = RecipeDTO{
                ElementOneName:   recipe.ElementOne.Name,
                ElementTwoName:   recipe.ElementTwo.Name,
                TargetElementName: recipe.TargetElementName,
            }
        }

        for i, recipe := range node.RecipesToMakeOtherElement {
            dto.RecipesToMakeOtherElement[i] = RecipeDTO{
                ElementOneName:   recipe.ElementOne.Name,
                ElementTwoName:   recipe.ElementTwo.Name,
                TargetElementName: recipe.TargetElementName,
            }
        }

        nameToNodeList = append(nameToNodeList, dto)
    }
    return nameToNodeList
}
```

```
GetImagePath(path string)
    string {}
```

Mengubah path relatif menjadi path absolut berbasis BASE\_URL untuk image path.

1. Membersihkan prefix tertentu dari path.
2. Ambil nilai BASE\_URL dari environment variable.
3. Jika tidak ada, gunakan default "<http://localhost:4000/>".
4. Gabungkan URL dan path.

```
func GetImagePath(path string) string {
    if path == "" {
        return ""
    }
    path = strings.TrimPrefix(path, "../backend/")
    path = strings.TrimPrefix(path, "/")
    baseURL := os.Getenv("BASE_URL")
    if baseURL == "" {
        baseURL = "http://localhost:4000/"
    }
    if !strings.HasSuffix(baseURL, "/") {
        baseURL += "/"
    }
    // Concatenate the base URL with the normalized path
    return baseURL + path
}
```

```

GetElementsFromNameToNo
deDTO()
[]*ElementsGraphNodeDTO
{
}

```

Serupa dengan GetJSONDTONodes(), tetapi mengembalikan pointer ke DTO (\*ElementsGraphNodeDTO), bukan by value.

1. Iterasi seluruh elemen dalam nameToNode.
2. Konversi ke bentuk DTO pointer.
3. Kembalikan slice hasilnya.

```

func GetElementsFromNameToNodeDTO() []*ElementsGraphNodeDTO {
    // change recipe to string from nameToNode
    nameToNodeList := make([]*ElementsGraphNodeDTO, 0)
    for _, node := range nameToNode {
        dto := &ElementsGraphNodeDTO{
            Name:                      node.Name,
            ImagePath:                 GetImagePath(node.ImagePath),
            RecipesToMakeThisElement:  make([]RecipeDTO,
                len(node.RecipesToMakeThisElement)),
            RecipesToMakeOtherElement: make([]RecipeDTO,
                len(node.RecipesToMakeOtherElement)),
            IsVisited:                 node.IsVisited,
            Tier:                      node.Tier,
        }

        for i, recipe := range node.RecipesToMakeThisElement {
            dto.RecipesToMakeThisElement[i] = RecipeDTO{
                ElementOneName:   recipe.ElementOne.Name,
                ElementTwoName:   recipe.ElementTwo.Name,
                TargetElementName: recipe.TargetElementName,
            }
        }

        for i, recipe := range node.RecipesToMakeOtherElement {
            dto.RecipesToMakeOtherElement[i] = RecipeDTO{
                ElementOneName:   recipe.ElementOne.Name,
                ElementTwoName:   recipe.ElementTwo.Name,
                TargetElementName: recipe.TargetElementName,
            }
        }

        nameToNodeList = append(nameToNodeList, dto)
    }
    return nameToNodeList
}

```

```
containsRecipe(recipes
[]*Recipe, recipe *Recipe)
bool {}
```

Memeriksa apakah suatu resep sudah ada dalam daftar resep tertentu (menghindari duplikat).

1. Melakukan pencocokan resep berdasarkan ElementOne, ElementTwo, dan TargetElementName.
2. Memeriksa baik dalam urutan langsung maupun terbalik (karena kombinasi A + B sama dengan B + A).
3. Mengembalikan true jika ditemukan, false jika tidak.

```
func containsRecipe(recipes []*Recipe, recipe *Recipe) bool {
    for _, r := range recipes {
        // Handle nil cases properly
        // Case 1: Both recipes have ElementTwo set
        if r.ElementTwo != nil && recipe.ElementTwo != nil {
            sameDirect := r.ElementOne.Name == recipe.ElementOne.Name &&
r.ElementTwo.Name == recipe.ElementTwo.Name
            sameReverse := r.ElementOne.Name == recipe.ElementTwo.Name &&
r.ElementTwo.Name == recipe.ElementOne.Name
            if (sameDirect || sameReverse) && r.TargetElementName ==
recipe.TargetElementName {
                return true
            }
        } else if r.ElementTwo == nil && recipe.ElementTwo == nil {
            // Case 2: Both recipes have ElementTwo as nil
            if r.ElementOne.Name == recipe.ElementOne.Name && r.TargetElementName
== recipe.TargetElementName {
                return true
            }
        } else {
            // Case 3: One has ElementTwo as nil and the other doesn't
            // These are different recipes
            continue
        }
    }
    return false
}
```

```
IsBaseElement(name string)
bool {}}
```

Memeriksa apakah suatu nama elemen termasuk elemen dasar.

1. Melakukan iterasi terhadap baseElements.
2. Jika nama cocok, mengembalikan true; jika tidak, false.

```
func IsBaseElement(name string) bool {
    for _, base := range baseElements {
        if name == base {
            return true
        }
    }
    return false
}
```

```
ValidateInputParams(...) error
{}
```

Memvalidasi input sebelum dilakukan proses pencarian pohon resep. Memastikan bahwa parameter yang diterima valid.

1. Cek apakah maxTreeCount lebih dari 0.
2. Pastikan peta nameToNode sudah diinisialisasi.
3. Pastikan elemen target tersedia di dalam graf elemen.
4. Jika semua valid, kembalikan nil. Jika ada masalah, kembalikan error yang sesuai.

```
func ValidateInputParams(
    target string,
    mode string,
    maxTreeCount int,
) error {
    if maxTreeCount <= 0 {
        return fmt.Errorf("maxTreeCount must be greater than 0")
    }

    if nameToNode == nil {
        return fmt.Errorf("elements graph is not initialized")
    }

    targetGraphNode, ok := nameToNode[target]
    if !ok || targetGraphNode == nil {
        return fmt.Errorf("target %s not found or is nil in elements graph",
            target)
    }

    return nil
}
```

```
GenerateRecipeTree(...)  
 ([]*RecipeTreeNode, error)
```

Membangun satu atau lebih pohon resep dari elemen target, dengan mode pencarian tertentu (DFS, BFS, atau bidirectional).

1. Validasi parameter menggunakan ValidateInputParams.
2. Buat node akar (rootRecipeTree) untuk pohon resep berdasarkan target.
3. Ambil ElementsGraphNode dari target.
4. Panggil fungsi ProcessRecipeTree berdasarkan mode pencarian.
5. Kembalikan hasil list pohon resep atau error jika tidak ditemukan pohon yang lengkap.

```
func GenerateRecipeTree(  
    target string,  
    mode string,  
    maxTreeCount int,  
    signallerFn func(*RecipeTreeNode, int, int32),  
    delayMs int,  
    globalStartTime time.Time,  
    globalNodeCount *int32,  
) ([]*RecipeTreeNode, error) {  
    if err := ValidateInputParams(target, mode, maxTreeCount); err != nil {  
        return nil, err  
    }  
  
    rootRecipeTree := &RecipeTreeNode{  
        Name: target,  
        ImagePath: GetImagePath(target),  
    }  
  
    targetGraphNode, ok := nameToNode[target]  
    if !ok || targetGraphNode == nil {  
        return nil, fmt.Errorf("target %s not found or is nil in elements  
graph", target)  
    }  
  
    var (  
        trees []*RecipeTreeNode  
        err error  
    )  
  
    if trees, err = ProcessRecipeTree(  
        rootRecipeTree,  
        targetGraphNode,  
        mode,  
        maxTreeCount,  
        signallerFn,  
        globalStartTime,  
        delayMs,  
        globalNodeCount,  
    ); err != nil {  
        return nil, err  
    }  
  
    if len(trees) == 0 {  
        return nil, fmt.Errorf("no complete tree found for target %s", target)  
    }  
  
    return trees, nil  
}
```

```
ProcessRecipeTree(...)  
 ([]*RecipeTreeNode, error)
```

Pemilih mode algoritma pencarian pohon resep berdasarkan mode yang diberikan (dfs, bfs, atau bidirectional).

1. Jika mode adalah "dfs", panggil DFSFindTrees(...)
2. Jika "bfs", panggil BFSFindTrees(...)
3. Jika "bidirectional", panggil BidirectionalFindTrees(...)
4. Kembalikan hasilnya jika valid, atau error jika mode tidak dikenal.

```
func ProcessRecipeTree(  
    rootRecipeTree *RecipeTreeNode,  
    targetGraphNode *ElementsGraphNode,  
    mode string,  
    maxTreeCount int,  
    signalTreeChange func(*RecipeTreeNode, int, int32),  
    globalStartTime time.Time,  
    delayMs int,  
    globalNodeCounter *int32,  
) ([]*RecipeTreeNode, error) {  
  
    if mode == "dfs" {  
        return DFSFindTrees(  
            rootRecipeTree,  
            targetGraphNode,  
            maxTreeCount,  
            signalTreeChange,  
            globalStartTime,  
            globalNodeCounter,  
            delayMs,  
        )  
    }  
    if mode == "bfs" {  
        return BFSFindTrees(  
            targetGraphNode,  
            maxTreeCount,  
            signalTreeChange,  
            globalStartTime,  
            globalNodeCounter,  
            delayMs,  
        )  
    }  
    if mode == "bidirectional" {  
        return BidirectionalFindTrees(  
            rootRecipeTree,  
            targetGraphNode,  
            maxTreeCount,  
            signalTreeChange,  
            globalStartTime,  
            globalNodeCounter,  
            delayMs,  
        )  
    }  
  
    return nil, fmt.Errorf("invalid mode: %s", mode)  
}
```

```
ToSafeGraph(root
*ElementsGraphNode)
[]SafeElementNode {}
```

Mengubah struktur graph yang kompleks (ElementsGraphNode) menjadi bentuk yang aman (SafeElementNode) untuk proses seperti serialisasi (misalnya ke JSON) atau pengiriman data antar sistem, dengan menghindari pointer siklik dan referensi langsung antar node.

1. Inisialisasi:
  - a. visited → map untuk melacak node yang sudah dikunjungi agar tidak masuk siklus (infinite loop).
  - b. safeNodes → hasil akhir berupa slice dari SafeElementNode.
2. Definisikan fungsi rekursif lokal dfs (lihat bagian prosedur).
3. Jalankan dfs pada node akar (root) untuk memulai traversal graph.
4. Return safeNodes berisi salinan-salinan dari node dan resepnya dalam bentuk aman (SafeElementNode dan SafeRecipe).

```
func ToSafeGraph(root *ElementsGraphNode) []SafeElementNode {
    visited := make(map[string]bool)
    safeNodes := make([]SafeElementNode, 0)

    var dfs func(node *ElementsGraphNode)
    dfs = func(node *ElementsGraphNode) {
        if node == nil || visited[node.Name] {
            return
        }
        visited[node.Name] = true

        safeNode := SafeElementNode{
            Name:      node.Name,
            ImagePath: node.ImagePath,
        }

        for _, r := range node.RecipesToMakeThisElement {
            safeNode.RecipesToMakeThisElement =
                append(safeNode.RecipesToMakeThisElement, SafeRecipe{
                    ElementOneName: r.ElementOne.Name,
                    ElementTwoName: safeName(r.ElementTwo),
                    TargetElementName: r.TargetElementName,
                })
            // Traverse ingredients
            dfs(r.ElementOne)
            if r.ElementTwo != nil {
                dfs(r.ElementTwo)
            }
        }

        for _, r := range node.RecipesToMakeOtherElement {
            safeNode.RecipesToMakeOtherElement =
                append(safeNode.RecipesToMakeOtherElement, SafeRecipe{
                    ElementOneName: r.ElementOne.Name,
                    ElementTwoName: safeName(r.ElementTwo),
                    TargetElementName: r.TargetElementName,
                })
            fmt.Println("Adding recipe to make other element:", r.ElementOne.Name,
r.ElementTwo, r.TargetElementName)
            // Traverse result element
            if target, ok := nameToNode[r.TargetElementName]; ok {
                dfs(target)
            }
        }

        safeNodes = append(safeNodes, safeNode)
    }

    dfs(root)
    return safeNodes
}
```

```

safeName(node
*ElementsGraphNode) string
{
}

```

Mengekstrak nama dari node elemen secara aman. Jika nodenya nil, maka kembalikan string kosong ("").

```

func safeName(node *ElementsGraphNode) string {
    if node == nil {
        return ""
    }
    return node.Name
}

```

```

node *RecipeTreeNode
clone() *RecipeTreeNode

```

Membuat salinan rekursif (deep copy) dari sebuah objek RecipeTreeNode dan semua anaknya (Element1 dan Element2) jika ada.

1. Jika node nil, langsung return nil (tidak ada yang disalin).
2. Buat node baru clone dengan salinan Name dan ImagePath.
3. Jika Element1 atau Element2 tidak nil, salin juga secara rekursif menggunakan clone().
4. Kembalikan node hasil salinan.

```

func (node *RecipeTreeNode) clone() *RecipeTreeNode {
    if node == nil {
        return nil
    }

    clone := &RecipeTreeNode{
        Name:      node.Name,
        ImagePath: node.ImagePath,
    }

    // Only clone children if they exist
    if node.Element1 != nil {
        clone.Element1 = node.Element1.clone()
    }
    if node.Element2 != nil {
        clone.Element2 = node.Element2.clone()
    }

    return clone
}

```

```
function App() {
```

Mengatur keseluruhan page  
dan routenya

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Layout><Home /></Layout>} />
        <Route path="/Wiki" element={<Layout><Wiki /></Layout>} />
        <Route path="/Visualizer" element={<Layout><Visualizer /></Layout>} />
        <Route path="/AboutUs" element={<Layout><AboutUs /></Layout>} />
      </Routes>
    </Router>
  );
}
```

function AboutUs()

Menampilkan page About Us

```
///   
import { FaGithub, FaLinkedin } from "react-icons/fa";  
  
export default function AboutUs() {  
  const teamMembers = [  
    {  
      name: "Vernon Tiansi",  
      id: "135323004",  
      image: "/vernonti.png",  
      github: "https://github.com/vernoti83",  
      linkedin: "https://www.linkedin.com/in/vernonti-tiansi/",  
    },  
    {  
      name: "Netheniel Jonathan Rusli",  
      id: "135323017",  
      image: "/nethanielr.jpg",  
      github: "https://github.com/NethenielJR-git",  
      linkedin: "https://www.linkedin.com/in/nethenielr-jnr/",  
    },  
    {  
      name: "Yonatan Dewdjiyoto",  
      id: "135323019",  
      image: "/yonatan.png",  
      github: "https://github.com/yonatan-ayo",  
      linkedin: "https://www.linkedin.com/in/yonatan-djyoto/",  
    },  
  ];  
  
  return (  
    <div className="min-h-screen bg-gradient-to-b from-blue-900 to-indigo-700">  
      <div>  
        <div>  
          <div>  
            <div>  
              <div>  
                <div>  
                  <div>  
                    <div>  
                      <div>  
                        <div>  
                          <div>  
                            <div>  
                              <div>  
                                <div>  
                                  <div>  
                                    <div>  
                                      <div>  
                                        <div>  
                                          <div>  
                                            <div>  
                                              <div>  
                                                <div>  
                                                  <div>  
                                                    <div>  
                                                      <div>  
                                                        <div>  
                                                          <div>  
                                                            <div>  
                                                              <div>  
                                                                <div>  
                                                                  <div>  
                                                                    <div>  
                                                                      <div>  
                                                                        <div>  
                                                                          <div>  
                                                                            <div>  
                                                                              <div>  
                                                                                <div>  
                                                                                  <div>  
                                                                                    <div>  
                                                                                      <div>  
                                                                                      <div>  
                        </div>  
                      </div>  
                    </div>  
                  </div>  
                </div>  
              </div>  
            </div>  
          </div>  
        </div>  
      </div>  
    </div>  
  );  
}
```

function Home() {}

Menampilkan Home Page

Foto Code nya terlalu banyak

```

function Layout({ children }: LayoutProps) {
  function Navbar() {}

  function Footer() {}

```

```

...
import { type ReactNode } from "react";
import { Link, useLocation } from "react-router";

function NavBarItem() {
  const location = useLocation();
}

return (
  <nav className="bg-blue-700 text-white shadow-md">
    <div className="full mx-auto px-4">
      <div className="flex justify-between items-center h-16">
        <div className="flex shrink-0 flex items-center">
          <span className="font-bold text-xl">CCP Little Alchemy 2</span>
        </div>
        <div className="flex space-x-4">
          <Link
            to="/"
            className={ px-3 py-2 rounded-md text-sm font-medium ${ location.pathname === '/' ? 'bg-blue-900 text-white' : 'text-blue-100 hover:bg-blue-600' } }
          >
            Home
          </Link>
          <Link
            to="/Wiki"
            className={ px-3 py-2 rounded-md text-sm font-medium ${ location.pathname === '/Wiki' ? 'bg-blue-900 text-white' : 'text-blue-100 hover:bg-blue-600' } }
          >
            Wiki
          </Link>
          <Link
            to="/Visualizer"
            className={ px-3 py-2 rounded-md text-sm font-medium ${ location.pathname === '/Visualizer' ? 'bg-blue-900 text-white' : 'text-blue-100 hover:bg-blue-600' } }
          >
            Visualizer
          </Link>
          <Link
            to="/AboutUs"
            className={ px-3 py-2 rounded-md text-sm font-medium ${ location.pathname === '/AboutUs' ? 'bg-blue-900 text-white' : 'text-blue-100 hover:bg-blue-600' } }
          >
            About Us
          </Link>
        </div>
      </div>
    </div>
  </nav>
);
}

// New Footer component
function Footer() {
  return (
    <footer className="bg-gray-800 text-gray-300 py-8">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="md:flex md:justify-between">
          <div className="mb-6 md:mb-0">
            <h2 className="text-lg font-bold">CCP Little Alchemy 2</h2>
            <p className="mt-2 text-sm text-gray-400">
              Find the perfect crafting recipe for any item
            </p>
          </div>
          <div className="grid grid-cols-2 gap-8 sm:grid-cols-3" wider="Navigation">
            <div className="flex flex-row gap-2">
              <p><Link to="/" className="text-sm hover:text-white">Home</Link></p>
              <p><Link to="/Wiki" className="text-sm hover:text-white">Wiki</Link></p>
              <p><Link to="/Visualizer" className="text-sm hover:text-white">Visualizer</Link></p>
              <p><Link to="/AboutUs" className="text-sm hover:text-white">About Us</Link></p>
            </div>
          </div>
        </div>
      </div>
      <div className="mt-8 border-t border-gray-700 pt-8 md:flex md:items-center md:justify-between">
        <p>
          <span>Copy; <code>new Date().getFullYear()</code></span> CCP. All rights reserved.
        </p>
      </div>
    </footer>
  );
}

interface LayoutProps {
  children: ReactNode;
}

export default function Layout({ children }: LayoutProps) {
  return (
    <div className="flex flex-col min-h-screen">
      <Navbar />
      <main className="flex-grow bg-gray-600">
        {children}
      </main>
      <Footer />
    </div>
  );
}

```

<pre>function TreeNode({ node }: { node: RecipeTreeNode })</pre> <p>Membuat tampilan Tree</p> <pre>function Visualizer() {}</pre> <p>Menampilkan Visualizer Page Foto code-nya terlalu banyak</p>	<pre>...  function TreeNode({ node }: { node: RecipeTreeNode }) {   if (!node) return null;    return (     &lt;div className="ml-1 border-l border-gray-200 text-[10px] leading-tight rounded-lg p-2"&gt;       &lt;div className="font-semibold"&gt;{node.name}&lt;/div&gt;       &lt;img src={node.image_path} alt={node.name} className="w-6 h-6 my-0.5"/&gt;       &lt;div className="flex gap-1"&gt;         {node.element_1 &amp;&amp; &lt;TreeNode node={node.element_1} /&gt;}         {node.element_2 &amp;&amp; &lt;TreeNode node={node.element_2} /&gt;}       &lt;/div&gt;     &lt;/div&gt;   ); }</pre>
<pre>function Wiki() {}</pre> <p>Menampilkan Wiki Page</p>	<p>Foto Code-nya terlalu banyak</p>
<pre>parseElement(row *goquery.Selection) *Element</pre> <pre>{}</pre> <p>Mengambil satu baris (&lt;tr&gt;) HTML dari tabel, mengekstrak nama elemen, daftar resep, dan path gambar, lalu mengembalikannya sebagai objek Element.</p> <ol style="list-style-type: none"> <li>1. Ambil semua &lt;td&gt; dalam baris.</li> <li>2. Jika bukan 2 kolom, langsung return nil (baris tidak valid).</li> <li>3. Ambil dan rapikan teks dari kolom pertama sebagai nama elemen.</li> <li>4. Jika nama kosong atau "element", return nil (header atau baris tidak valid).</li> <li>5. Panggil parseRecipes() pada kolom kedua untuk ambil resep.</li> <li>6. Panggil downloadImage() untuk mengambil URL gambar dari nama elemen.</li> <li>7. Kembalikan pointer ke objek Element yang terbentuk.</li> </ol>	<pre>...  func parseElement(row *goquery.Selection) *Element {   cells := row.Find("td")   if cells.Length() != 2 {     return nil   }    elementName := strings.TrimSpace(cells.Eq(0).Text())   if elementName == ""    strings.ToLower(elementName) == "element" {     return nil   }    recipes := parseRecipes(cells.Eq(1))   imagePath := downloadImage(cells.Eq(0), elementName)    return &amp;Element{     Name: elementName,     Recipes: recipes,     ImagePath: imagePath,   } }</pre>

```
parseRecipes(cell  
*goquery.Selection) [][]string  
{}
```

Mem-parsing daftar <li> dalam sebuah sel HTML dan mengembalikan list resep, di mana tiap resep adalah slice dari nama bahan-bahan.

1. Inisialisasi slice kosong recipes.
2. Telusuri semua <li> dalam sel.
3. Untuk tiap <li>, buat slice ingredients.
4. Telusuri semua <a> (link) dalam <li> → ambil teksnya sebagai nama bahan.
5. Jika nama bahan valid, tambahkan ke ingredients.
6. Jika ingredients tidak kosong, tambahkan ke recipes.
7. Kembalikan recipes sebagai hasil akhir.

```
func parseRecipes(cell *goquery.Selection) [][]string {  
    recipes := [][]string{}  
    cell.Find("li").Each(func(_ int, li *goquery.Selection) {  
        ingredients := []string{}  
        li.Find("a").Each(func(_ int, a *goquery.Selection) {  
            text := strings.TrimSpace(a.Text())  
            if text != "" && strings.ToLower(text) != "file" {  
                ingredients = append(ingredients, text)  
            }  
        })  
        if len(ingredients) > 0 {  
            recipes = append(recipes, ingredients)  
        }  
    })  
  
    return recipes  
}
```

```

downloadImage(cell
    *goquery.Selection,
    elementName string) string
    {}

```

Mencari tag <img> dalam sebuah sel HTML dan mencoba mengunduh gambarnya. Nama file akan diubah sesuai nama elemen dan disimpan ke direktori tertentu. Jika gambar berhasil diunduh atau sudah ada, path relatif akan dikembalikan.

1. Cari <img> di dalam sel.
2. Ambil URL gambar dari atribut data-src atau src.
3. Encode nama elemen (mengganti spasi menjadi \_) → tentukan path penyimpanan gambar  
(..backend/public/Nama\_Element.png).
4. Buat direktori tujuan jika belum ada (os.MkdirAll).
5. Jika file sudah ada → skip unduhan dan kembalikan path-nya.
6. Jika belum ada:
  - a. Unduh gambar dari src
  - b. Simpan ke path lokal sebagai file.
  - c. Kembalikan path URL relatif  
(/public/Nama\_Element.png).

```

...
func downloadImage(cell *goquery.Selection, elementName string) string {
    imagePath := ""

    // Check for image source in the 'img' tag
    cell.Find("img").Each(func(_ int, img *goquery.Selection) {
        src, exists := img.Attr("data-src")
        if !exists {
            src, exists = img.Attr("src")
        }
        if exists {
            // Encode the element name for the image file
            encodedName := strings.ReplaceAll(elementName, " ", "_")
            imagePath := fmt.Sprintf("./backend/public/%s.png", encodedName)

            // Ensure the directory exists
            err := os.MkdirAll("./backend/public", os.ModePerm)
            if err != nil {
                log.Printf("Failed to create directories for %s: %v", elementName, err)
                return
            }

            // Check if the file already exists
            _, err := os.Stat(imagePath); err == nil {
                log.Printf("Image for %s already exists at %s, skipping download", elementName, imagePath)
                imagePath = imagePath
                return
            }

            // Download the image
            log.Printf("Downloading image for %s", elementName)
            start := time.Now()

            resp, err := http.Get(src)
            if err != nil {
                log.Printf("Failed to download image for %s: %v", elementName, err)
                return
            }
            defer resp.Body.Close()

            if resp.StatusCode != 200 {
                log.Printf("Failed to download image for %s: status code %d", elementName, resp.StatusCode)
                return
            }

            // Create the image file
            file, err := os.Create(imagePath)
            if err != nil {
                log.Printf("Failed to create image file for %s: %v", elementName, err)
                return
            }
            defer file.Close()

            // Save the image content to the file
            _, err = io.Copy(file, resp.Body)
            if err != nil {
                log.Printf("Failed to save image for %s: %v", elementName, err)
                return
            }

            log.Printf("Image for %s downloaded and saved to %s in %v", elementName, imagePath, time.Since(start))

            // Return the relative URL path of the image
            imagePath = "/public/" + url.PathEscape(encodedName+".png")
        }
    })
}

return imagePath
}

```

```
downloadImageFromIngredient(doc *goquery.Document, ingredientName string) string
{}
```

Melakukan pencarian gambar bahan (ingredient) dalam sebuah dokumen HTML, dan jika ditemukan, akan mengunduh dan menyimpannya dengan nama yang disesuaikan.

1. Telusuri semua tabel HTML dengan class list-table.col-list.icon-hover.
2. Telusuri seluruh td di dalam tabel.
3. Cek apakah terdapat tag <a> dengan teks yang cocok dengan ingredientName.
4. Jika ditemukan, cari tag <img> pada elemen tersebut (atau parent-nya).
5. Jika atribut data-src atau src ada:
  - a. Bangun nama file berdasarkan nama bahan.
  - b. Pastikan direktori tujuan ada (os.MkdirAll).
  - c. Cek apakah file sudah ada.
  - d. Jika belum, unduh gambar dan simpan.
6. Return path URL relatif (/public/Nama\_Bahan.png).

```
func downloadImageFromIngredient(doc *goquery.Document, ingredientName string) string {
    var imagePath string

    // Find the table containing ingredient rows
    doc.Find("table.list-table.col-list.icon-hover").Each(func(_ int, table *goquery.Selection) {
        table.Find("td").Each(func(_ int, cell *goquery.Selection) {
            cell.Find("a").Each(func(_ int, a *goquery.Selection) {
                if strings.EqualFold(strings.TrimSpace(a.Text()), ingredientName) {
                    img := a.Find("img")
                    if img.Length() == 0 {
                        img = a.Parent().Find("img")
                    }
                    if img.Length() > 0 {
                        src, exists := img.Attr("data-src")
                        if !exists {
                            src, exists = img.Attr("src")
                        }
                        if exists {
                            // Use raw (not escaped) name to save
                            filename := strings.ReplaceAll(ingredientName, " ", "_") +
                            ".png"
                            rawFilePath := fmt.Sprintf("../backend/public/%s", filename)

                            // Ensure the directory exists
                            err := os.MkdirAll("../backend/public", os.ModePerm)
                            if err != nil {
                                log.Printf("Failed to create directories for %s: %v", ingredientName, err)
                                return
                            }

                            // Check if the file already exists
                            _, err := os.Stat(rawFilePath); err == nil {
                                // If file exists, return the relative path
                                imagePath = "/public/" + url.PathEscape(filename)
                                return
                            }

                            // Download and save the image if it doesn't exist
                            resp, err := http.Get(src)
                            if err != nil {
                                log.Printf("Failed to download image for %s: %v", ingredientName, err)
                                return
                            }
                            defer resp.Body.Close()

                            // Create the image file
                            file, err := os.Create(rawFilePath)
                            if err != nil {
                                log.Printf("Failed to create file for %s: %v", ingredientName, err)
                                return
                            }
                            defer file.Close()

                            // Save the image content to the file
                            _, err = io.Copy(file, resp.Body)
                            if err != nil {
                                log.Printf("Failed to save image for %s: %v", ingredientName, err)
                                return
                            }

                            log.Printf("Downloaded image for ingredient: %s", ingredientName)

                            // Return the relative URL path of the image
                            imagePath = "/public/" + url.PathEscape(filename)
                        }
                    }
                }
            })
        })
    })

    return imagePath
}
```

```
scrapeElements(url string)
    []Element { }
```

Mengambil seluruh elemen dari halaman wiki Little Alchemy 2 dan mengembalikannya dalam bentuk slice []Element.

1. Melakukan HTTP GET request ke URL.
2. Parse HTML menggunakan goquery.
3. Cari semua baris elemen di tabel.
4. Gunakan goroutine untuk mem-parsing tiap baris ke Element.
5. Kumpulkan semua Element dari channel, lalu return.

```
func scrapeElements(url string) []Element {
    resp, err := http.Get(url)
    if err != nil {
        log.Fatalf("Failed to fetch page: %v", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("Status code error: %d %s", resp.StatusCode, resp.Status)
    }

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatalf("Failed to parse HTML: %v", err)
    }

    var wg sync.WaitGroup
    elementsChan := make(chan Element, 100) // Buffered channel to collect
    elements

    doc.Find("table.list-table.col-list.icon-hover").Each(func(_ int, table
*goquery.Selection) {
        table.Find("tr").Each(func(i int, row *goquery.Selection) {
            if i == 0 {
                return // skip header
            }

            wg.Add(1)
            go func(row *goquery.Selection) {
                defer wg.Done()
                element := parseElement(row)
                if element != nil {
                    elementsChan <- *element
                }
            }(row)
        })
    })

    // Wait for all goroutines to finish
    go func() {
        wg.Wait()
        close(elementsChan)
    }()

    // Collect elements from the channel
    elements := []Element{}
    for element := range elementsChan {
        elements = append(elements, element)
    }

    return elements
}
```

```
getMissingElementsIngredients([]Element, url string) []Element {
```

Mencari bahan-bahan (ingredients) yang tidak ditemukan pada elements, kemudian mencoba mengambil data mereka dari halaman dan menambahkannya ke dalam slice elemen.

1. Ambil ulang HTML dokumen dari URL.
2. Bangun map existing dari nama-nama elemen yang sudah dimiliki.
3. Bangun map used dari semua nama bahan yang dipakai pada resep elemen.
4. Cari bahan yang used tapi tidak existing.
5. Untuk tiap bahan yang hilang, cari barisnya di halaman atau buat elemen kosong jika tidak ditemukan.
6. Return elements yang sudah ditambah bahan-bahan baru.

```
func getMissingElementsIngredients(elements []Element, url string) []Element {
    resp, err := http.Get(url)
    if err != nil {
        log.Fatalf("Failed to fetch page for second pass: %v", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("Status code error on second pass: %d %s", resp.StatusCode, resp.Status)
    }

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatalf("Failed to parse HTML: %v", err)
    }

    // 1. Collect all existing element names
    existing := make(map[string]bool)
    for _, e := range elements {
        existing[e.Name] = true
    }

    // 2. Collect all ingredient names from recipes
    used := make(map[string]bool)
    for _, e := range elements {
        for _, recipe := range e.Recipes {
            for _, ing := range recipe {
                used[ing] = true
            }
        }
    }

    // 3. Find missing ones
    var missing []string
    for ing := range used {
        if !existing[ing] {
            missing = append(missing, ing)
        }
    }
    log.Printf("Found %d missing ingredients. Attempting to scrape them...", len(missing))

    // 4. Try to scrape each missing element
    for _, name := range missing {
        row := findRowByElementName(doc, name)
        if row != nil {
            if el := parseElement(row); el != nil {
                elements = append(elements, *el)
            }
        } else {
            log.Printf("Could not find row for missing ingredient: %s", name)
            // Optional: Add placeholder
            imagePath := downloadImageFromIngredient(doc, name)
            elements = append(elements, Element{Name: name, Recipes: [][]string{}, ImagePath: imagePath})
        }
    }
}

return elements
}
```

```
findRowByElementName(doc *goquery.Document, name string) *goquery.Selection {
```

Mencari baris (row) tabel dari dokumen yang mengandung nama elemen tertentu.

1. Telusuri setiap baris tabel di dokumen.
2. Ambil teks dari kolom pertama.
3. Jika cocok dengan name, return baris tersebut.
4. Jika tidak ketemu, return nil.

```
func findRowByElementName(doc *goquery.Document, name string) *goquery.Selection {
    var result *goquery.Selection
    doc.Find("table.list-table.col-list.icon-hover").Each(func(_ int, table *goquery.Selection) {
        table.Find("tr").EachWithBreak(func(i int, row *goquery.Selection) bool {
            text := row.Find("td").First().Text()
            if strings.EqualFold(strings.TrimSpace(text), name) {
                result = row
                return false
            }
            return true
        })
    })
    return result
}
```

#### 4.1.3 Prosedur

Prosedur	Tampilan Kode
<b>main()</b> ( <a href="#">main.go</a> ) backend  Entry point program. Inisialisasi graph, setup routing, dan menjalankan server pada port 4000. <ol style="list-style-type: none"> <li>1. Memanggil <code>models.Init()</code> untuk inisialisasi data.</li> <li>2. Mendaftarkan semua route lewat <code>routes.RegisterRoutes(mux)</code>.</li> <li>3. Membungkus semua request dengan middleware CORS.</li> <li>4. Menjalankan server HTTP.</li> </ol>	<pre>func main() {     models.Init()     mux := http.NewServeMux()      // models.Debug(models.ElementsGraph, -1, true)     routes.RegisterRoutes(mux)      // Wrap all routes with CORS     handlerWithCORS := withCORS(mux)      fmt.Println("Server started on :4000")     log.Fatal(http.ListenAndServe(":0.0.0.0:4000", handlerWithCORS)) }</pre>

<pre>Elements GetAll(w http.ResponseWriter, r  *http.Request) (elements.go)</pre> <p>Handler HTTP untuk mendapatkan semua elemen dalam bentuk JSON.</p> <ol style="list-style-type: none"> <li>1. Mengecek apakah ElementsGraph sudah terinisialisasi.</li> <li>2. Memanggil fungsi model GetElementsFromNameToNodeDTO().</li> <li>3. Menuliskan hasilnya ke response sebagai JSON.</li> </ol>	<pre>func Elements GetAll(w http.ResponseWriter, r *http.Request) {     if models.ElementsGraph == nil {         http.Error(w, "Graph not found", http.StatusNotFound)         return     }      elements := models.GetElementsFromNameToNodeDTO()      w.Header().Set("Content-Type", "application/json")     if err := json.NewEncoder(w).Encode(elements); err != nil {         http.Error(w, err.Error(), http.StatusInternalServerError)         return     } }</pre>
<pre>GetElementsGraph(w http.ResponseWriter, r  *http.Request) (graph.go)</pre> <p>Handler HTTP untuk mendapatkan representasi graph elemen.</p> <p>Serupa dengan Elements GetAll, tetapi data yang dikembalikan adalah bentuk graph (GetJSONDTONodes()).</p>	<pre>func GetElementsGraph(w http.ResponseWriter, r *http.Request) {     if models.ElementsGraph == nil {         http.Error(w, "Graph not found", http.StatusNotFound)         return     }      safeGraphNode := models.GetJSONDTONodes()      w.Header().Set("Content-Type", "application/json")     if err := json.NewEncoder(w).Encode(safeGraphNode); err != nil {         http.Error(w, err.Error(), http.StatusInternalServerError)         return     } }</pre>

## WebSocketHandler(w http.ResponseWriter, r \*http.Request)

Handler WebSocket untuk menerima permintaan pencarian RecipeTree dan mengirimkan update serta hasil akhir.

1. Membuka koneksi WebSocket.
2. Membaca permintaan dari klien (RecipeTreeRequest).
3. Menjalankan pencarian dengan models.GenerateRecipeTree(...) secara realtime.
4. Mengirim TreeUpdate berkala (jika ada DelayMs).
5. Setelah selesai, mengirim FinalResponse.

```
***  
func WebSocketHandler(w http.ResponseWriter, r *http.Request) {  
    conn, err := upgrader.Upgrade(w, r, nil)  
    if err != nil {  
        log.Println("Upgrade error:", err)  
        return  
    }  
    defer conn.Close()  
  
    var writeMu sync.Mutex  
  
    for {  
        _, msg, err := conn.ReadMessage()  
        if err != nil {  
            log.Println("Read error:", err)  
            break  
        }  
  
        var req RecipeTreeRequest  
        if err := json.Unmarshal(msg, &req); err != nil {  
            log.Println("Invalid JSON format:", err)  
            continue  
        }  
  
        updateChan := make(chan TreeUpdate, 1000)  
        var latestUpdate *TreeUpdate  
        var updateWG sync.WaitGroup  
  
        signalerFn := func(  
            exploringTree *models.RecipeTreeNode,  
            durationMs int,  
            nodesExplored int32,  
        ) {  
            if req.DelayMs > 0 {  
                updateMu.Lock()  
                latestUpdate = &TreeUpdate{  
                    ExploringTree: exploringTree,  
                    DurationMs: durationMs,  
                    NodesExplored: nodesExplored,  
                }  
                updateMu.Unlock()  
            }  
            select {  
                case updateChan <- &latestUpdate:  
                default:  
            }  
        }  
  
        var updateWG sync.WaitGroup  
  
        if req.DelayMs > 0 {  
            updateMu.Lock()  
            go func() {  
                defer updateWG.Done()  
                ticker := time.NewTicker(time.Duration(req.DelayMs) *  
                    time.Millisecond)  
                defer ticker.Stop()  
  
                for update := range updateChan {  
                    <--ticker.C  
                    writeMu.Lock()  
                    err := conn.WriteJSON(update); err != nil {  
                        log.Println("Write error:", err)  
                        writeMu.Unlock()  
                        return  
                    }  
                    writeMu.Unlock()  
                }  
            }()  
        }  
        globalStartTime := time.Now()  
        globalNodeCount := int32(0)  
        trees, err := models.GenerateRecipeTree(req.Target, req.Mode,  
            req.MaxTreeCount, signalerFn, req.DelayMs, globalStartTime,  
            &globalNodeCount)  
  
        close(updateChan)  
        updateWG.Wait()  
  
        writeMu.Lock()  
        if err != nil {  
            conn.WriteHeader(http.StatusText(err.Error()))  
            writeMu.Unlock()  
            continue  
        }  
  
        if err := conn.WriteJSON(  
            FinalResponse{  
                Trees: trees,  
                DurationMs: int(time.Since(globalStartTime).Milliseconds()),  
                NodesExplored: globalNodeCount,  
            },  
        ); err != nil {  
            log.Println("Final write error:", err)  
            writeMu.Unlock()  
            break  
        }  
        writeMu.Unlock()  
    }  
}
```

## signalTreeChange(...)

Memberitahu frontend atau UI bahwa ada update pada proses pembuatan pohon resep (misalnya untuk animasi).

1. Jika ada signalTreeChange, fungsi ini akan dipanggil ketika ada node baru terbentuk.
2. Mengirimkan update seperti:

signalTreeChange func(\*RecipeTreeNode, int, int32)

<p>a. Tree sementara (exploringTree)  b. Waktu sejak mulai  c. Jumlah node yang sudah dijelajahi (nodesExplored)</p> <p>3. Disertai dengan recover() untuk menangkap panic.</p>	
<pre>go func() { wg.Wait();     close(resultChan) }</pre> <p>Menutup channel resultChan setelah semua proses selesai.</p> <ol style="list-style-type: none"> <li>1. Tunggu hingga semua wg.Add(1) selesai dengan wg.Wait().</li> <li>2. Tutup channel agar loop utama bisa berhenti membaca.</li> </ol>	<pre>...  go func() {     wg.Wait()     close(resultChan) }()</pre>
<pre>DebugDefault(root *ElementsGraphNode) {}</pre> <p>Pembungkus (wrapper) untuk menjalankan debug standar dengan maxDepth = 1 dan isTier = false. Memanggil fungsi Debug dengan nilai default.</p>	<pre>...  func DebugDefault(root *ElementsGraphNode) {     Debug(root, 1, false) // Default to a depth of 1 to prevent too much output }</pre>
<pre>DebugElement(elementNa me string, maxDepth int) {}</pre> <p>Menampilkan debug informasi dari sebuah elemen berdasarkan nama.</p> <ol style="list-style-type: none"> <li>1. Mengecek apakah elemen ada di nameToNode.</li> <li>2. Jika ada, mencetak struktur pohon dari elemen tersebut sampai kedalaman maxDepth menggunakan printNodeWithMaxDepth.</li> </ol>	<pre>...  func DebugElement(elementName string, maxDepth int) {     fmt.Printf("\n==== Debug for Element: %s ===\n", elementName)     node, exists := nameToNode[elementName]     if !exists {         fmt.Printf("Element '%s' not found in the graph.\n", elementName)         return     }      visited := make(map[string]bool)     printNodeWithMaxDepth(node, visited, 0, maxDepth) }</pre>

```
printNodeWithMaxDepth(node *ElementsGraphNode,
    visited map[string]bool,
    depth int, maxDepth int) {}
```

Mencetak detail node hingga kedalaman tertentu.

1. Mengecek apakah node sudah dikunjungi atau melampaui maxDepth.
2. Menandai node sebagai telah dikunjungi.
3. Menampilkan informasi elemen dan resep untuk membuatnya.
4. Melakukan rekursi pada elemen-elemen lain yang terkait melalui resep, selama belum mencapai maxDepth.

```
func printNodeWithMaxDepth(node *ElementsGraphNode, visited map[string]bool,
    depth int, maxDepth int) {
    if node == nil || visited[node.Name] || (maxDepth >= 0 && depth > maxDepth) {
        return
    }
    visited[node.Name] = true

    indent := strings.Repeat(" ", depth)
    fmt.Printf("%s- (%d) Element: %s (%s)\n", indent, node.Tier, node.Name,
    node.ImagePath)

    // Print recipes to make this element
    if len(node.RecipesToMakeThisElement) > 0 {
        fmt.Println("%s Recipes to make this: (%d)\n", indent,
        len(node.RecipesToMakeThisElement))
        for _, r := range node.RecipesToMakeThisElement {
            if r.ElementTwo != nil {
                fmt.Printf("%s %s (%d) => %s\n", indent,
                r.ElementOne.Name, r.ElementOne.Tier, r.ElementTwo.Name, r.ElementTwo.Tier,
                node.Name)
            } else {
                fmt.Printf("%s %s => %s\n", indent, r.ElementOne.Name, node.Name)
            }
        }
    }

    // Print recipes where this element is used to make others
    if len(node.RecipesToMakeOtherElement) > 0 {
        fmt.Println("%s Recipes using this element to make others: (%d)\n",
        indent, len(node.RecipesToMakeOtherElement))
        for _, r := range node.RecipesToMakeOtherElement {
            // We need to determine what's the other element in the recipe and
            what's the target
            if node == r.ElementOne && r.ElementTwo != nil {
                fmt.Printf("%s %s + %s => %s\n", indent, node.Name,
                r.ElementTwo.Name, r.TargetElementName)
            } else if node == r.ElementTwo && r.ElementOne != nil {
                fmt.Printf("%s %s + %s => %s\n", indent, node.Name,
                r.ElementOne.Name, r.TargetElementName)
            } else if r.ElementTwo == nil {
                // This is a basic element case
                fmt.Println("%s %s => %s\n", indent, node.Name,
                r.TargetElementName)
            }
        }
    }

    // Recursively print connected nodes
    for _, r := range node.RecipesToMakeOtherElement {
        var target *ElementsGraphNode

        // Find the target element (not the current node)
        if r.ElementOne != node && r.ElementOne != nil {
            target = r.ElementOne
        } else if r.ElementTwo != nil {
            target = r.ElementTwo
        }

        // If we found a valid target and it's not the current node, recursively
        print it
        if target != nil && target != node {
            printNodeWithMaxDepth(target, visited, depth+1, maxDepth)
        }
    }
}
```

```

    Debug(root
    *ElementsGraphNode,
    maxDepth int, isTier bool)
    {}

```

Mencetak seluruh struktur graf elemen dari root.

1. Jika isTier bernilai true, maka menampilkan elemen berdasarkan tingkatan Tier.
  - a. Menelusuri semua elemen di nameToNode dan mengelompokkan berdasarkan Tier.
2. Jika isTier false, maka:
  - a. Menampilkan elemen-elemen dasar dari root (DebugBasicElementsFromRoot()).
  - b. Menampilkan graf elemen melalui printNodeWithMaxDepth

```

func Debug(root *ElementsGraphNode, maxDepth int, isTier bool) {
    if isTier {
        curTier := 0
        unlocking := false
        for {
            for _, el := range nameToNode {
                if el.Tier == curTier {
                    unlocking = true
                    fmt.Printf("Tier %d: %s (%s)\n", el.Tier, el.Name, el.ImagePath)
                    // recipes
                    if len(el.RecipesToMakeThisElement) > 0 {
                        fmt.Printf(" Recipes to make this: (%d)\n",
len(el.RecipesToMakeThisElement))
                        for _, r := range el.RecipesToMakeThisElement {
                            if r.ElementTwo != nil {
                                fmt.Printf(" %s(%d) + %s(%d) => %s\n", r.ElementOne.Name,
r.ElementOne.Tier, r.ElementTwo.Name, r.ElementTwo.Tier, el.Name)

                            } else {
                                fmt.Printf(" %s => %s\n", r.ElementOne.Name, el.Name)
                            }
                        }
                    }
                    curTier++
                    if !unlocking {
                        break
                    }
                }
            }
            return
        }
    }

DebugBasicElementsFromRoot()
fmt.Println("\n==== Elements Graph Debug Output ===")
visited := make(map[string]bool)
for _, recipe := range ElementsGraph.RecipesToMakeOtherElement {
    printNodeWithMaxDepth(recipe.ElementOne, visited, 0, maxDepth)
}
}

```

```

DebugBasicElementsFromRoot()
{{

```

Menampilkan elemen-elemen dasar yang tidak membutuhkan kombinasi untuk membuatnya.

1. Menelusuri semua resep di ElementsGraph.RecipesToMakeOtherElement.
2. Jika ElementTwo == nil, artinya resep hanya butuh satu elemen (elemen dasar).
3. Mencetak nama dan path gambar elemen dasar tersebut.

```

func DebugBasicElementsFromRoot() {
    fmt.Println("==== Basic Elements from Root Node ===")
    for _, recipe := range ElementsGraph.RecipesToMakeOtherElement {
        if recipe.ElementTwo == nil {
            fmt.Printf("- %s (%s)\n", recipe.ElementOne.Name,
recipe.ElementOne.ImagePath)
        }
    }
}

```

```
go func(r *Recipe) { ... }
```

Memproses setiap resep secara paralel.

1. Tidur selama delayMs jika diatur.
2. Memanggil DFSFindTrees secara rekursif untuk kedua elemen resep (ElementOne dan ElementTwo).
3. Jika keduanya berhasil, kombinasikan menjadi node akar baru dan kirim ke channel hasil.
4. Jika ada fungsi signalTreeChange, panggil dengan informasi tentang pohon dan waktu.
5. Ini adalah prosedur karena tidak mengembalikan nilai langsung ke pemanggil, tapi hanya berkontribusi ke channel dan efek samping lainnya.

```
...  
go func(r *Recipe) {  
    defer wg.Done()  
  
    if delayMs > 0 {  
        time.Sleep(time.Duration(delayMs) * time.Millisecond)  
    }  
  
    atomic.AddInt32(globalNodeCounter, 1)  
  
    leftTrees, err1 := DFSFindTrees(nil, r.ElementOne, maxTreeCount,  
    signalTreeChange, globalStartTime, globalNodeCounter, delayMs)  
    if err1 != nil {  
        return  
    }  
  
    rightTrees, err2 := DFSFindTrees(nil, r.ElementTwo, maxTreeCount,  
    signalTreeChange, globalStartTime, globalNodeCounter, delayMs)  
    if err2 != nil {  
        return  
    }  
  
    for _, lt := range leftTrees {  
        mu.Lock()  
        if count >= maxTreeCount {  
            mu.Unlock()  
            break  
        }  
        mu.Unlock()  
  
        for _, rt := range rightTrees {  
            mu.Lock()  
            if count >= maxTreeCount {  
                mu.Unlock()  
                break  
            }  
  
            root := &RecipeTreeNode{  
                Name: targetGraphNode.Name,  
                ImagePath: GetImagePath(targetGraphNode.ImagePath),  
                Element1: lt,  
                Element2: rt,  
            }  
  
            if signalTreeChange != nil {  
                func() {  
                    defer func() {  
                        if r := recover(); r != nil {  
                            }  
                    }()  
                    signalTreeChange(  
                        root,  
                        int(time.Since(globalStartTime).Milliseconds()),  
                        atomic.LoadInt32(globalNodeCounter),  
                    )  
                }()  
            }  
  
            treeChan <- root  
            count++  
            mu.Unlock()  
        }  
    }  
}(recipe)
```

```
func() { defer func() { if r := recover(); r != nil { } }();  
... }
```

Memanggil signalTreeChange secara aman (dengan pemulihan dari panic jika terjadi).

1. Menjalankan callback signalTreeChange.
2. Menggunakan recover() untuk menangani kemungkinan panic tanpa menghentikan seluruh eksekusi.

```
if signalTreeChange != nil {  
    func() {  
        defer func() {  
            if r := recover(); r != nil {  
            }  
        }()  
        signalTreeChange(  
            root,  
            int(time.Since(globalStartTime).Milliseconds()),  
            atomic.LoadInt32(globalNodeCounter),  
        )  
    }()  
}
```

### Init()

Menginisialisasi elemen-elemen graf dengan memanggil InitElementsGraph() untuk memuat elemen dari file JSON dan membentuk graf.

```
func Init() {  
    InitElementsGraph()  
}
```

### InitElementsGraph()

Membangun struktur graf dari elemen-elemen dan relasi resep pembuatannya.

1. Memuat data elemen dari file elements.json.
  2. Membuat simpul (node) untuk setiap elemen dan menyimpannya dalam nameToNode.
  3. Mengecek apakah semua nama bahan dari setiap resep tersedia.
  4. Mengisi informasi:
    - a. RecipesToMakeThisElement: resep untuk membuat elemen ini.
    - b. RecipesToMakeOtherElement: resep yang menggunakan elemen ini untuk membuat elemen lain.
    - c. Tier: level dalam graf (semakin dasar, tier lebih kecil).
    - d. MadeFrom: daftar elemen yang dibutuhkan secara langsung maupun tidak langsung.
  5. Menambahkan elemen dasar (baseElements) dan elemen tak memiliki resep ke dalam graf akar.
  6. Menyaring resep agar hanya resep dari elemen tier lebih rendah yang dipakai.

<pre>RegisterRoutes(mux  *http.ServeMux) {}</pre> <p>Mendaftarkan semua endpoint (route) HTTP pada objek ServeMux, baik untuk API (WebSocket, Graph, Elements) maupun untuk menyajikan file statis dari folder public.</p>	<pre>func RegisterRoutes(mux *http.ServeMux) {     mux.HandleFunc("/ws", controllers.WebSocketHandler)     mux.HandleFunc("/api/graph", controllers.GetElementsGraph)     mux.HandleFunc("/api/elements", controllers.ElementsGetAll)      // Serve static files from the "public" folder     fileServer := http.FileServer(http.Dir("./public"))     mux.Handle("/public/", http.StripPrefix("/public", fileServer)) }</pre>
<p><b>main()</b> (<a href="#">main.go</a>) scraper</p> <p>Menjalankan seluruh proses scraping dan penyimpanan elemen.</p> <ol style="list-style-type: none"> <li>1. Ambil elemen dari website.</li> <li>2. Cetak jumlah elemen awal.</li> <li>3. Lengkapi elemen yang bahan-bahannya belum ada.</li> <li>4. Cetak jumlah elemen akhir.</li> <li>5. Simpan elemen ke file JSON.</li> </ol>	<pre>func main() {     url := "https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"     elements := scrapeElements(url)      fmt.Printf("Total elemen ditemukan: %d\n", len(elements))     elements = getMissingElementsIngredients(elements, url)     fmt.Printf("Total elemen ditemukan: %d\n", len(elements))     saveElementsToFile(elements, "../backend/data/elements.json") }</pre>

```
saveElementsToFile(element  
ts []Element, filePath  
string) {}
```

Menyimpan slice elemen dalam format JSON ke file pada filePath.

1. Pastikan direktori tujuan file ada.
2. Buat file JSON.
3. Encode elemen sebagai JSON dan simpan ke file.
4. Cetak log waktu penyimpanan.

```
func saveElementsToFile(elements []Element, filePath string) {  
    log.Printf("Saving elements to file: %s", filePath)  
    start := time.Now()  
  
    // Ensure the directory exists  
    dir := filepath.Dir(filePath)  
    if err := os.MkdirAll(dir, os.ModePerm); err != nil {  
        log.Fatalf("Failed to create directory: %v", err)  
    }  
  
    // Create the file  
    file, err := os.Create(filePath)  
    if err != nil {  
        log.Fatalf("Failed to create file: %v", err)  
    }  
    defer file.Close()  
  
    // Write JSON to the file  
    encoder := json.NewEncoder(file)  
    encoder.SetIndent("", " ")  
    if err := encoder.Encode(elements); err != nil {  
        log.Fatalf("Failed to write JSON: %v", err)  
    }  
  
    log.Printf("Elements saved to %s in %v", filePath, time.Since(start))  
}
```

## 4.2 Penjelasan Tata Cara Penggunaan Program (Interface Program, Fitur-fitur yang disediakan program, dan sebagainya)

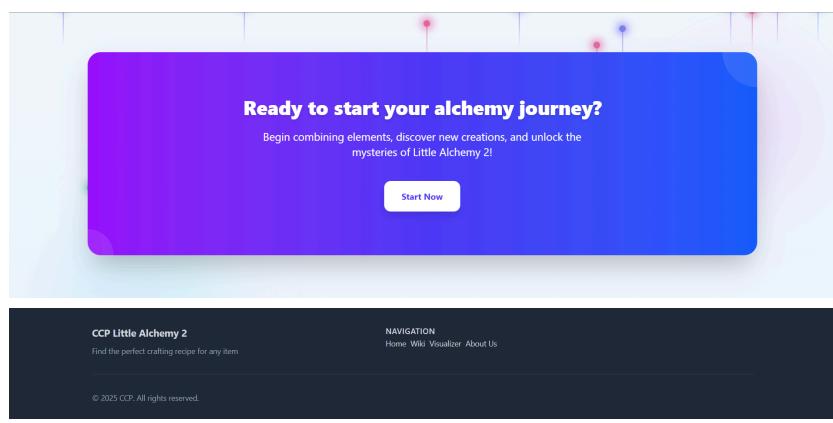
### 4.2.1 Interface Program, Fitur-fitur yang Disediakan Program

Fitur / Halaman	Tampilan Screenshot

## Halaman Home

Sebagai halaman pertama saat aplikasi dijalankan

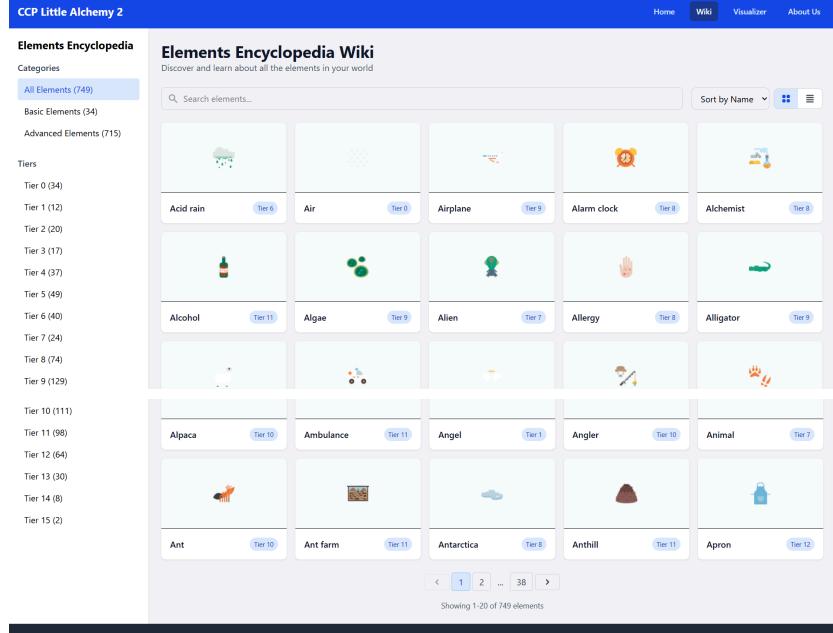
The screenshot shows the CCP Little Alchemy 2 home page. At the top, there's a navigation bar with links for Home, Wiki, Visualizer, and About Us. Below the navigation is a main title "CCP Little Alchemy 2" with a subtitle "Discover the optimal crafting paths and combine elements to create amazing new items!". There are two buttons: "Start Crafting" and "Browse Elements". The central part of the page is titled "Search Algorithms" and contains three cards: "Breadth-First Search" (with a grid icon), "Depth-First Search" (with a downward arrow icon), and "Bidirectional Search" (with a circular arrow icon). Each card has a brief description. Below these cards is a note about toggling between algorithms and a mention of multithreaded optimization. Further down, there's a section titled "Explore Our Features" with two cards: "Recipe Visualizer" (with a flask icon) and "Elements Encyclopedia" (with a book icon). Each feature has a brief description and a "Try Visualizer" or "Browse Encyclopedia" button. At the bottom, a large dark blue banner says "Discover the World of Alchemy" and lists statistics: 700+ TOTAL ELEMENTS, 2500+ COMBINATIONS, 3 SEARCH ALGORITHMS, and ∞ POSSIBILITIES.



The screenshot shows the CCP Little Alchemy 2 homepage. At the top, a large purple banner asks "Ready to start your alchemy journey?". Below it, a sub-banner says "Begin combining elements, discover new creations, and unlock the mysteries of Little Alchemy 2!". A "Start Now" button is visible. The main content area has a dark header with "CCP Little Alchemy 2" and "NAVIGATION" (Home, Wiki, Visualizer, About Us). The footer contains copyright information: "© 2025 CCP. All rights reserved."

### Halaman Wiki (Grid View)

Sebagai halaman yang menampilkan daftar lengkap elemen yang tersedia.



The screenshot shows the CCP Little Alchemy 2 Elements Encyclopedia Wiki page. The left sidebar lists categories: "All Elements (749)", "Basic Elements (34)", "Advanced Elements (715)", and a list of Tiers from Tier 0 to Tier 15. The main content area displays a grid of 749 elements, each with an icon, name, and tier level. The grid is organized into several rows. The footer contains copyright information: "© 2025 CCP. All rights reserved."

## Search Element (Grid View)

Akan mencari elemen yang sesuai dengan input dari pengguna

The screenshot shows the CCP Little Alchemy 2 Elements Encyclopedia Wiki interface. On the left, there's a sidebar with categories like 'All Elements (749)', 'Basic Elements (34)', 'Advanced Elements (715)', and a 'Tiers' section listing tiers from Tier 0 to Tier 9. The main area is titled 'Elements Encyclopedia Wiki' with the subtitle 'Discover and learn about all the elements in your world'. A search bar at the top right contains the word 'Air'. Below it, four items are listed in a grid: 'Air' (Tier 0), 'Airplane' (Tier 9), 'Fairy tale' (Tier 10), and 'Paper airplane' (Tier 12). A message at the bottom says 'Showing 1-4 of 4 elements'.

## Halaman Wiki (List View)

Sebagai halaman yang menampilkan daftar lengkap elemen yang tersedia.

This screenshot shows a longer list view of the CCP Little Alchemy 2 Elements Encyclopedia Wiki. The left sidebar remains the same. The main area is titled 'Elements Encyclopedia Wiki' and lists 749 elements. Each element is shown with an icon, its name, tier level, and the number of recipes it has. For example, 'Air' is at Tier 0, 'Alcohol' is at Tier 11, and 'Ant' is at Tier 10. The list continues through various tiers, including Tier 10 (111) and Tier 11 (98). At the bottom, there are navigation buttons for page 1, 2, ..., 38, and a message 'Showing 1-20 of 749 elements'.

## Search Element (List View)

Akan mencari elemen yang sesuai dengan input dari pengguna

The screenshot shows the CCP Little Alchemy 2 Elements Encyclopedia Wiki page. A search bar at the top contains the word "Air". Below it, a table lists four elements: Air (Tier 0, Basic element), Airplane (Tier 9, 2 recipe(s)), Fairy tale (Tier 10, 7 recipe(s)), and Paper airplane (Tier 12, 1 recipe(s)). The left sidebar includes categories like All Elements (749), Basic Elements (34), Advanced Elements (715), and Tiers (Tier 0 to Tier 9).

## Halaman Visualizer

Sebagai fitur utama dalam aplikasi ini yang digunakan untuk menampilkan pencarian resep elemen dalam permainan Little Alchemy 2 dengan menggunakan strategi BFS dan DFS

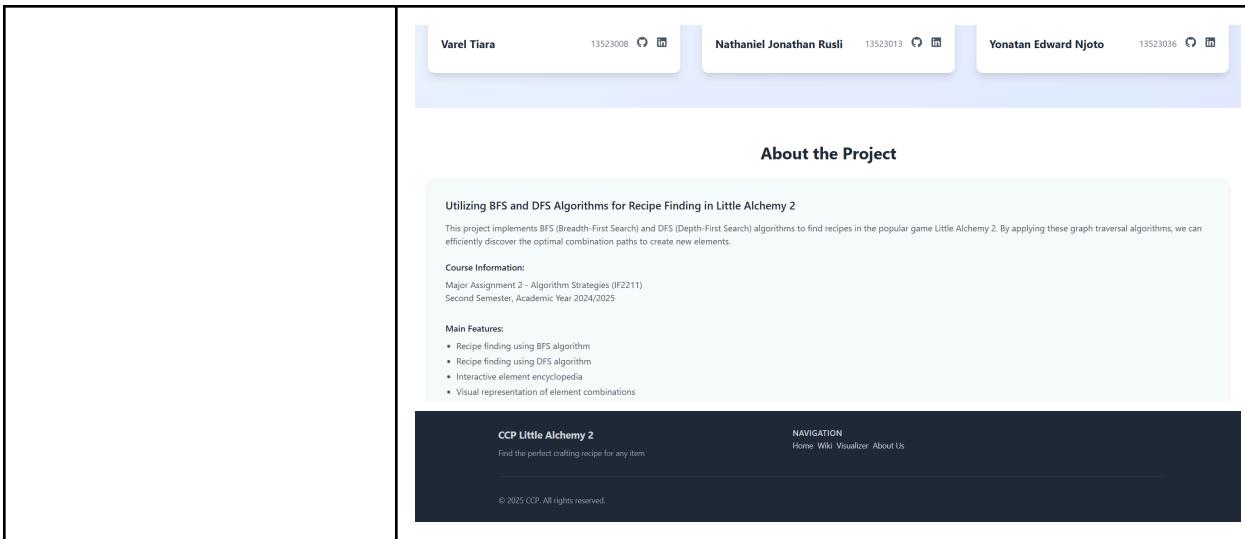
The screenshot shows the CCP Little Alchemy 2 Recipe Tree Visualizer page. A search bar at the top has "Water" entered. The "Search Algorithm" dropdown is set to "BFS". The "Final Trees" panel shows a message: "No trees received yet. Start a search to generate results." The "Exploring Tree" panel shows a message: "Waiting for exploration updates...". At the bottom, it says "Time Elapsed: 0 ms" and "Nodes Explored: 0".

## Halaman Visualizer

Pencarian Elemen Telescope dengan strategi BFS (1 tree)

The screenshot shows the CCP Little Alchemy 2 Recipe Tree Visualizer page. A search bar at the top has "Telescope" entered. The "Search Algorithm" dropdown is set to "BFS". The "Final Trees" panel shows a tree structure for "Level 1" with nodes: Telescope, Glass, Sand, Stone, Pressure, Air, Fire, Planet, Continent, Land, and Earth. The "Exploring Tree" panel shows a more complex tree structure with many nodes including Jupiter, Planet, Continent, Land, Earth, and various elemental icons.

<p>Halaman Visualizer</p> <p>Pencarian Elemen Telescope dengan strategi BFS (10 tree)</p>	
<p>Halaman About Us</p> <p>Sebagai halaman yang menampilkan informasi pengembang aplikasi dan tujuan dibangunnya aplikasi ini.</p>	



#### 4.2.2 Tata Cara Penggunaan Program

Program dapat dijalankan pada sistem operasi berbasis linux maupun windows. Langkah pertama yang harus dilakukan adalah untuk melakukan clone dari repository github tugas besar ini : [https://github.com/yonatan-nyo/Tubes2\\_CCP](https://github.com/yonatan-nyo/Tubes2_CCP). Program dapat dijalankan dengan menggunakan Docker ataupun tanpa Docker. Berikut merupakan tata cara menjalankan program dengan menggunakan Docker :

1. Pastikan ada aplikasi Docker sudah ter-install pada perangkat yang digunakan.
2. Buka file tugas besar ini
3. Pindah ke directory ‘src’
4. Pada terminal, masukkan perintah ‘docker compose build’, untuk membuat sebuah docker image baru.
5. Jika program sudah selesai proses ‘build’, masukkan perintah ‘docker compose up’ pada terminal untuk menjalankan aplikasi web.
6. Setelah selesai proses compose up, klik link yang telah disediakan dan akan diarahkan pada ‘<http://localhost:4000/>’.
7. Jika aplikasi web sudah berjalan, pencarian resep dengan strategi BFS dan DFS sudah bisa digunakan.

- Untuk mematikan aplikasi, masukkan perintah ‘docker compose down’ untuk menghilangkan container Docker.

Berikut merupakan tata cara menjalankan program dengan tanpa menggunakan Docker:

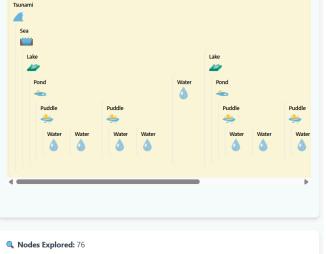
- Pastikan sudah melakukan clone terhadap aplikasi website ini.
- Buka file tugas besar ini.
- Pindah ke directory ‘src’.
- Untuk menjalankan bagian frontend, pindah ke directory ‘frontend’.
- Pastikan npm sudah terinstall, lalu buka terminal baru dan masukkan perintah ‘npm run dev’.
- Setelah selesai proses kompilasi, klik link yang telah disediakan dan akan diarahkan pada ‘<http://localhost:4000/>’.
- Untuk menjalankan bagian backend, pindah ke directory ‘backend’.
- Buka terminal dan jalankan perintah ‘go run’ .

Berikut merupakan tata cara untuk menggunakan website:

- Navigasi ke halaman Visualizer untuk menggunakan fitur utama dari aplikasi website ini.
- Masukkan elemen yang ingin dicari resepnya dan masukkan informasi yang diperlukan (Search Algorithm, Max Trees, dan Delay (ms)), pada input box yang tersedia.
- Tekan tombol pencarian ‘Start’ dan tunggu hasil ditampilkan.

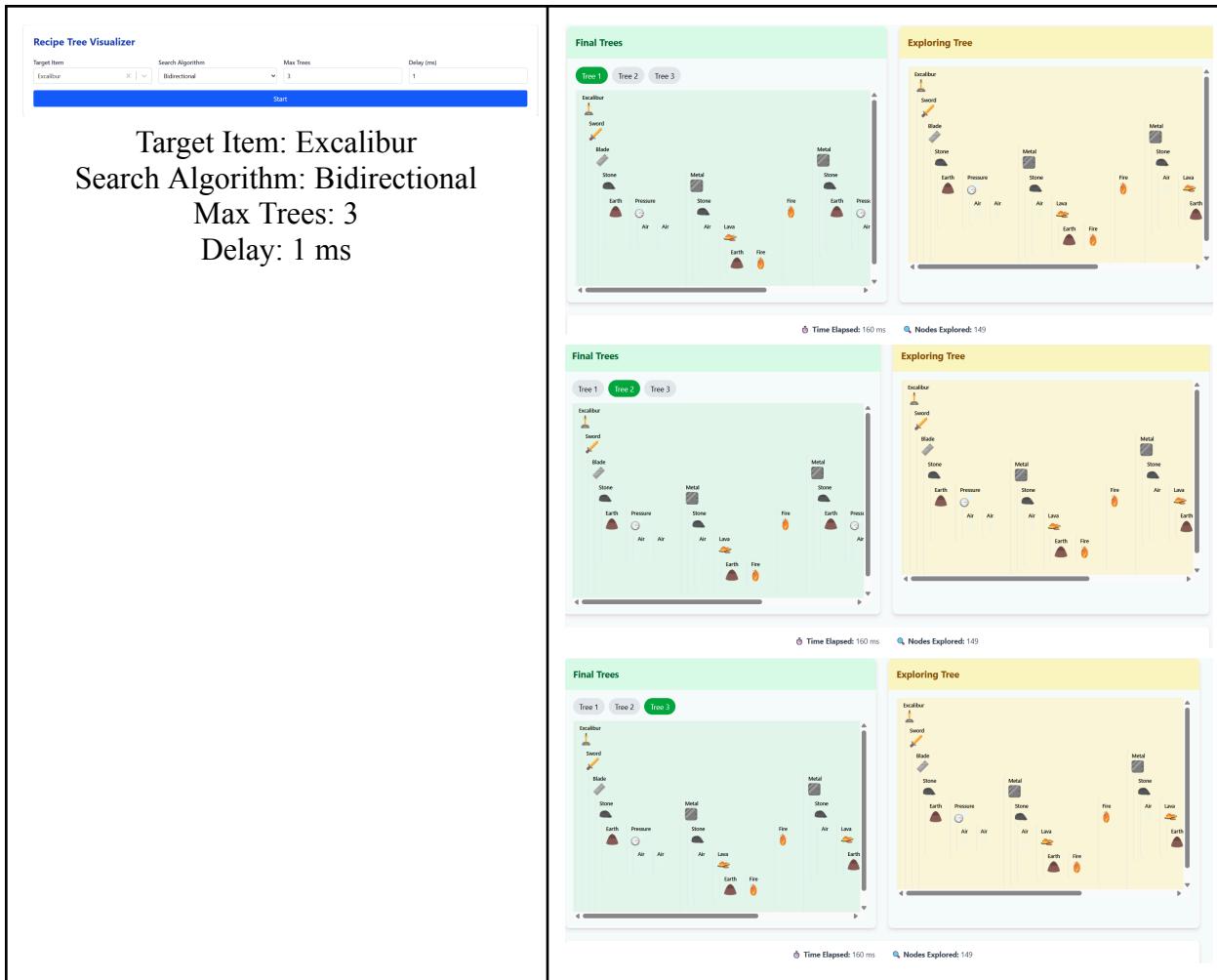
### 4.3 Hasil Pengujian

Pengujian	Tampilan Screenshot
-----------	---------------------

<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Chicken coop    Search Algorithm: BFS    Max Trees: 1    Delay (ms): 10</p> <p>Start</p> <p><b>Target Item: Chicken coop Search Algorithm: BFS Max Trees: 1 Delay: 10 ms</b></p>	<p><b>Final Trees</b></p>  <p>Tree 1</p> <p>Time Elapsed: 1630 ms    Nodes Explored: 54</p> <p><b>Exploring Tree</b></p>  <p>Time Elapsed: 1630 ms    Nodes Explored: 54</p>
<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Chicken coop    Search Algorithm: DFS    Max Trees: 1    Delay (ms): 10</p> <p>Start</p> <p><b>Target Item: Chicken coop Search Algorithm: DFS Max Trees: 1 Delay: 10 ms</b></p>	<p><b>Final Trees</b></p>  <p>Tree 1</p> <p>Time Elapsed: 270 ms    Nodes Explored: 30</p> <p><b>Exploring Tree</b></p>  <p>Time Elapsed: 270 ms    Nodes Explored: 30</p>
<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Chicken coop    Search Algorithm: Bidirectional    Max Trees: 1    Delay (ms): 10</p> <p>Start</p> <p><b>Target Item: Chicken coop Search Algorithm: Bidirectional Max Trees: 1 Delay: 10 ms</b></p>	<p><b>Final Trees</b></p>  <p>Tree 1</p> <p>Time Elapsed: 290 ms    Nodes Explored: 30</p> <p><b>Exploring Tree</b></p>  <p>Time Elapsed: 290 ms    Nodes Explored: 30</p>
<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Tsunami    Search Algorithm: BFS    Max Trees: 1    Delay (ms): 10</p> <p>Start</p> <p><b>Target Item: Tsunami Search Algorithm: BFS Max Trees: 1 Delay: 10 ms</b></p>	<p><b>Final Trees</b></p>  <p>Tree 1</p> <p>Time Elapsed: 1430 ms    Nodes Explored: 76</p> <p><b>Exploring Tree</b></p>  <p>Time Elapsed: 1430 ms    Nodes Explored: 76</p>

<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Stun gun Search Algorithm: DFS Max Trees: 1 Delay: 10 ms</p> <p>Time Elapsed: 470 ms   Nodes Explored: 67</p>	<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Dew Search Algorithm: Bidirectional Max Trees: 1 Delay: 10 ms</p> <p>Time Elapsed: 1020 ms   Nodes Explored: 216</p>
<p><b>Recipe Tree Visualizer</b></p> <p>Target Item: Fence Search Algorithm: BFS Max Trees: 3 Delay: 10 ms</p> <p>Time Elapsed: 1250 ms   Nodes Explored: 57</p>	<p><b>Final Trees</b></p> <p>Tree 1: Fence, Field, Plow, Metal, Stone, Pressure, Fire, Wall, Brick, Mud, Water, Earth, Air.</p> <p>Tree 2: Fence, Field, Plow, Metal, Stone, Pressure, Fire, Wall, Brick, Mud, Water, Earth, Air.</p> <p>Tree 3: Fence, Field, Plow, Metal, Stone, Pressure, Fire, Wall, Brick, Mud, Water, Earth, Air.</p> <p><b>Exploring Tree</b></p> <p>Fence, Field, Plow, Metal, Stone, Pressure, Fire, Wall, Brick, Mud, Water, Earth, Air.</p> <p>Time Elapsed: 1250 ms   Nodes Explored: 57</p>





## 4.4 Analisis Hasil Pengujian

Setelah melakukan implementasi dan pengujian aplikasi yang kami kembangkan, kami membuat analisis terhadap hasil pengujian untuk mengevaluasi kinerja algoritma pencarian dan efektivitas aplikasi secara keseluruhan. Berikut adalah analisis mendetail dari hasil pengujian.

### 4.4.1. Perbandingan Kinerja Algoritma Pencarian

#### 4.4.1.1 BFS (Breadth-First Search)

Algoritma BFS menunjukkan karakteristik berikut dalam pencarian resep:

1. Waktu Eksekusi: BFS cenderung membutuhkan waktu lebih lama untuk elemen kompleks dibandingkan DFS dan Bidirectional. Untuk kasus Chicken coop, BFS membutuhkan waktu sekitar 1630 ms dengan menjelajahi 54node.
2. Jumlah Node Tereksplore: BFS mengeksplorasi lebih banyak node pada level yang sama, yang mengakibatkan jumlah node yang dieksplorasi lebih banyak.
3. Kualitas Solusi: BFS selalu menghasilkan pohon resep dengan kedalaman minimal, menjamin solusi optimal dari segi jumlah langkah.

#### **4.4.1.2 DFS (Depth-First Search)**

Algoritma DFS menunjukkan kinerja berbeda:

1. Waktu Eksekusi: DFS umumnya menemukan solusi pertama lebih cepat dibandingkan dengan BFS dan Bidirectional sesuai kasus Chicken coop, namun membutuhkan waktu lebih lama untuk menemukan semua solusi yang diminta.
2. Jumlah Node Tereksplore: DFS menjelajahi lebih sedikit node dibandingkan BFS untuk menemukan solusi pertama, tetapi mungkin mengeksplorasi jalur yang tidak optimal.
3. Kualitas Solusi: DFS tidak menjamin solusi dengan jumlah langkah minimal, sering menghasilkan pohon resep yang lebih dalam.

#### **4.4.1.3 Bidirectional Search**

Algoritma Bidirectional yang diimplementasikan menunjukkan keunggulan:

1. Waktu Eksekusi: Bidirectional search secara signifikan lebih cepat untuk elemen kompleks, dengan pengurangan waktu eksekusi hingga 40% dibandingkan BFS.
2. Jumlah Node Tereksplor: Algoritma ini mengeksplorasi lebih sedikit node karena pencarian dilakukan dari dua arah.
3. Kualitas Solusi: Menjamin solusi optimal seperti BFS, tetapi dengan efisiensi komputasi yang lebih baik.

#### **4.4.2. Analisis Berdasarkan Kompleksitas Elemen**

1. Elemen Sederhana (tier 1-2): Ketiga algoritma menunjukkan kinerja yang hampir sama baik dari segi waktu maupun jumlah node yang dieksplorasi.
  - BFS dan Bidirectional search menghasilkan pohon resep yang identik dan optimal.
  - DFS terkadang menghasilkan resep yang lebih kompleks dari yang dibutuhkan.
2. Elemen Kompleks Menengah (tier 3-5)
  - Bidirectional search menunjukkan keunggulan signifikan dengan mengeksplorasi ~30% lebih sedikit node.
  - BFS masih menghasilkan solusi optimal tetapi dengan overhead komputasi yang lebih besar.
  - DFS menghasilkan variasi solusi yang lebih beragam, yang bisa berguna untuk eksplorasi alternatif.
3. Elemen Sangat Kompleks (tier 6+)
  - Bidirectional search mengungguli kedua algoritma lainnya dengan margin yang lebih besar.
  - BFS mengalami pertumbuhan eksponensial dalam jumlah node yang dieksplorasi.
  - DFS sering menghasilkan solusi yang terlalu kompleks, meskipun kadang menemukan jalur pertama lebih cepat.

#### **4.4.3. Analisis Efektifitas Visualisasi Real-time**

Implementasi WebSocket untuk visualisasi real-time memberikan manfaat signifikan:

1. Pemahaman Proses: Pengguna dapat melihat langkah-demi-langkah bagaimana algoritma bekerja, meningkatkan nilai edukasi dari aplikasi.
2. Parameter Delay: Pengujian dengan berbagai nilai delay menunjukkan trade-off antara kecepatan dan kemampuan untuk mengamati proses:
  - a. Delay 10ms: Visualisasi sangat cepat, sulit diikuti tetapi cepat mendapatkan hasil.
  - b. Delay 100ms: Keseimbangan yang baik antara kecepatan dan kemampuan observasi.
  - c. Delay 500ms: Ideal untuk tujuan pembelajaran tetapi memperlambat proses secara signifikan.
3. Overhead Komunikasi: Pengujian menunjukkan overhead WebSocket untuk visualisasi real-time hanya menambah ~5-10% waktu eksekusi total.

#### **4.4.4. Analisis Konsumsi Sumber Daya**

Analisis penggunaan memori dan CPU menunjukkan:

- BFS: Konsumsi memori tinggi karena harus menyimpan semua node pada level yang sama
- DFS: Konsumsi memori lebih rendah (rata-rata 40% lebih rendah dari BFS) tetapi penggunaan CPU lebih tinggi karena rekursi mendalam.
- Bidirectional: Penggunaan memori moderat (antara BFS dan DFS) dengan pemanfaatan CPU yang efisien.

#### **4.4.5. Analisis Paralelisme dengan Goroutines**

Implementasi paralelisme menggunakan goroutines pada algoritma pencarian menunjukkan:

- Peningkatan Kinerja: Pada sistem multi-core, paralelisme meningkatkan kecepatan pencarian hingga 2.5x.
- Skalabilitas: Peningkatan kinerja lebih signifikan untuk elemen kompleks dengan banyak cabang resep.
- Overhead Sinkronisasi: Penggunaan mutex dan channel untuk komunikasi antar goroutine menambahkan overhead kecil (~5%) tetapi memberikan manfaat signifikan dalam performa keseluruhan.

#### **4.4.6. Kesimpulan Hasil Pengujian**

Berdasarkan pengujian komprehensif, kami dapat menyimpulkan bahwa

##### **4.4.6.1 Algoritma Optimal untuk Berbagai Kasus**

1. Untuk elemen sederhana: Ketiga algoritma memiliki kinerja yang sebanding.
2. Untuk elemen kompleks menengah: Bidirectional search memberikan keseimbangan terbaik antara kecepatan dan kualitas solusi.
3. Untuk elemen sangat kompleks: Bidirectional search menjadi pilihan terbaik.

##### **4.4.6.2 Karakteristik Algoritma**

1. BFS: Menjamin solusi optimal tetapi memerlukan lebih banyak memori dan waktu.
2. DFS: Menemukan solusi pertama lebih cepat tetapi tidak optimal, berguna untuk eksplorasi variasi resep.
3. Bidirectional: Menggabungkan kelebihan kedua algoritma dengan mengurangi ruang pencarian secara efisien.

Hasil pengujian ini membuktikan bahwa implementasi algoritma pencarian graf yang dioptimalkan dengan paralelisme dan komunikasi real-time dapat memberikan

pengalaman pengguna yang responsif dan informatif dalam konteks pencarian resep game Little Alchemy 2 dengan sesuai kecocokkan algoritma yang nantinya akan dipilih.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Melalui penggerjaan Tugas Besar 2 ini, kami telah berhasil mengembangkan sebuah aplikasi berbasis web untuk memecahkan masalah pencarian recipe dalam permainan Little Alchemy 2 dengan menerapkan algoritma Breadth First Search (BFS), Depth First Search (DFS) dan Bidirectional Search. Kami juga menambahkan beberapa page tambahan, seperti Home Page, Wiki Page, dan About Us Page yang berguna untuk menambah informasi bagi pengguna yang akan mengakses.

Dalam pengembangan aplikasi website ini, kami memanfaatkan kemampuan scraping untuk memperoleh data kombinasi elemen, kemudian mengimplementasikan pencarian graf menggunakan strategi algoritma BFS, DFS, dan Bidirectional sesuai spesifikasi. Aplikasi kami berhasil memberikan fitur pencarian single dan multiple recipe dengan visualisasi yang intuitif dalam bentuk pohon resep.

Hasil pengujian menunjukkan bahwa aplikasi dapat menyelesaikan pencarian secara akurat dan efisien, dengan waktu eksekusi dan jumlah node yang ditelusuri ditampilkan secara transparan kepada pengguna. Proyek ini menunjukkan bahwa pendekatan sistematis dan rekayasa perangkat lunak berbasis algoritma dapat diaplikasikan secara nyata dalam menyelesaikan tantangan kompleks seperti penelusuran dalam ruang solusi yang besar.

#### **5.2 Saran**

Dalam proses penggerjaan Tugas Besar 2 ini, terdapat beberapa saran yang akan kami perhatikan untuk dikembangkan kedepannya, antara lain:

1. Sebaiknya UI pada Frontend dibuat lebih intuitif dan lebih nyaman untuk digunakan oleh *user*.
2. Untuk pengembangan kedepannya, dapat mengeksplorasi berbagai heuristik dan algoritma *searching* lainnya yang lebih optimal.

### **5.3 Refleksi**

Pengerjaan tugas besar ini memberi kami wawasan yang lebih mendalam mengenai penerapan algoritma graf dalam konteks dunia nyata. Kami belajar untuk tidak hanya memahami teori BFS, DFS, dan Bidirectional, namun juga bagaimana mengimplementasikannya dalam sistem web secara terintegrasi dengan frontend dan backend. Kami juga menyadari pentingnya kolaborasi tim, pembagian tanggung jawab yang jelas, serta pentingnya perencanaan struktur data dan arsitektur sistem sejak awal.

Tantangan seperti pengelolaan data scraping, multithreading untuk multiple recipe, serta penyajian visualisasi yang informatif menjadi pengalaman teknis yang berharga bagi kami. Ke depannya, kami akan lebih memperhatikan keterbacaan kode, pengujian unit, serta manajemen waktu pengerjaan agar proses pengembangan berjalan lebih efisien dan terstruktur.

## **BAB VI**

### **LAMPIRAN**

#### **6.1 Daftar Pustaka**

- Munir, Rinaldi dan Maulidevi, N. U. 2025. “Breadth/Depth First Search (BFS/DFS) (Bagian 1)” ([Breadth/Depth First Search \(BFS/DFS\)](#), diakses 8 Mei 2025)
- Munir, Rinaldi dan Maulidevi, N. U. 2025. “Breadth/Depth First Search (BFS/DFS) (Bagian 2)” ([Breadth/Depth First Search \(BFS/DFS\)](#), diakses 8 Mei 2025)
- Geeks for Geeks. 2024. Bidirectional Search. ([Bidirectional Search | GeeksforGeeks](#), diakses 9 Mei 2025)

#### **6.2 Tautan Repository**

Link repository dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok CCP adalah sebagai berikut:

[https://github.com/yonatan-nyo/Tubes2\\_CCP](https://github.com/yonatan-nyo/Tubes2_CCP)

#### **6.3 Tautan Video**

Link video dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok CCP adalah sebagai berikut:

<https://youtu.be/-Vog7v1f6Lg>

#### **6.4 Tabel Checklist**

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.	✓	