

Laporan Tugas Besar 2
IF2211 Strategi Algoritma
Pemanfaatan Algoritma BFS dan DFS dalam Pencarian Recipe
pada Permainan Little Alchemy 2



Dipersiapkan oleh:

Rafif Farris	13523095
Muhammad Edo Raduputu A	13523096
Athian Nugraha Muarajuang	13523106

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025

DAFTAR ISI

DAFTAR ISI.....	1
BAB 2	
LANDASAN TEORI.....	4
2.1. Traversal Graf.....	4
2.2. Algoritma Breadth First Search.....	4
2.3. Algoritma Depth First Search.....	5
2.4. Penerapan Bonus.....	5
2.4.1. Algoritma Bidirectional Search.....	5
2.5. Website.....	6
2.5.1. Frontend.....	6
2.5.2. Backend.....	6
BAB 3	
ANALISIS PEMECAHAN MASALAH.....	8
3.1. Dekomposisi Masalah.....	8
3.2. Pemetaan Masalah Menjadi Elemen-Elemen Algoritma.....	8
3.2.1. Pemetaan Masalah Menjadi Bentuk BFS.....	8
3.2.2 Pemetaan Masalah Menjadi Bentuk DFS.....	9
3.3. Fitur fungsional dan Arsitektur Aplikasi Web.....	10
3.3.1. Fitur Fungsional.....	10
3.3.2. Arsitektur Aplikasi Web.....	10
3.4. Ilustrasi Kasus.....	11
3.4.1. Ilustrasi Kasus dengan Algoritma BFS dalam Kasus “Find Recipe”.....	11
3.4.2. Ilustrasi Kasus dengan Algoritma DFS dalam Kasus “Find Recipe”.....	12
BAB 4	
IMPLEMENTASI DAN PENGUJIAN.....	13
4.1. Spesifikasi Teknis Program.....	13
4.2. Tata Cara Penggunaan Program.....	21
4.2.1. Cara Menjalankan Secara Lokal.....	21
3. Kemudian buka local host http://localhost:3000/ pada browser.....	21
4.2.2. Interface Program.....	21
4.3. Hasil Pengujian.....	21
4.3.1. Test Case 1.....	21
4.3.2. Test Case 2.....	23
4.3.3. Test Case 3.....	24
BAB 5	
KESIMPULAN DAN SARAN.....	27
5.1. Kesimpulan.....	27
5.2. Saran.....	27
LAMPIRAN.....	28
REFERENSI.....	29

BAB 1

DESKRIPSI TUGAS



Gambar 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 2. Elemen dasar pada Little Alchemy 2

2. Elemen turunan

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

3. *Combine Mechanism*

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

BAB 2

LANDASAN TEORI

2.1. Traversal Graf

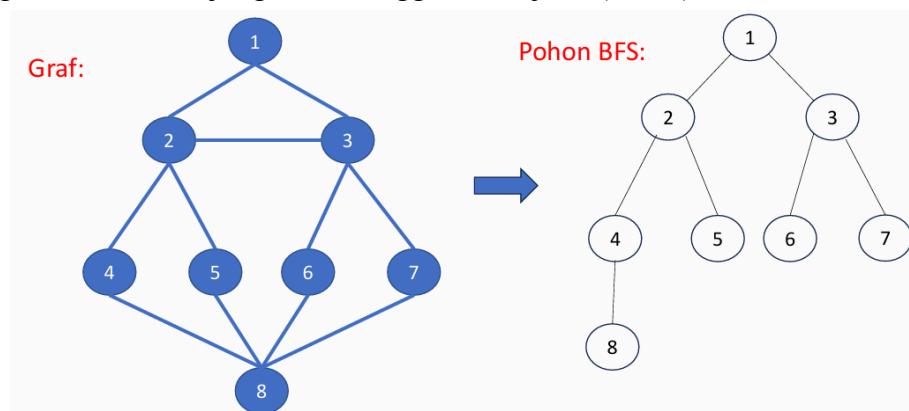
Graf merupakan struktur data yang terdiri dari kumpulan simpul (vertices) dan sisi (edges) yang menghubungkan pasangan simpul. Secara matematis, graf G didefinisikan sebagai $G = (V, E)$, di mana V adalah himpunan simpul dan E adalah himpunan sisi. Penjelajahan graf adalah proses sistematis untuk mengunjungi (memeriksa dan/atau memperbarui status) setiap titik dalam sebuah graf. Tujuan dari penjelajahan graf dapat bervariasi, mulai dari menemukan titik tertentu yang memenuhi kriteria tertentu, menemukan jalur antara dua titik, memeriksa konektivitas graf, hingga mengumpulkan informasi tentang struktur graf secara keseluruhan. Terdapat berbagai algoritma untuk melakukan penjelajahan graf, di antaranya adalah BFS dan DFS.

2.2. Algoritma Breadth First Search

Breadth-First Search (BFS) adalah sebuah algoritma fundamental yang digunakan untuk melakukan pencarian atau traversal pada struktur data graf atau pohon. Prinsip utama BFS adalah memulai dari sebuah titik akar (atau titik awal yang ditentukan dalam graf) dan menjelajahi semua titik tetangga pada kedalaman saat ini secara menyeluruh sebelum melanjutkan ke titik-titik pada tingkat kedalaman berikutnya.

Untuk mengelola urutan titik yang akan dikunjungi, BFS memanfaatkan struktur data *queue* yang beroperasi berdasarkan prinsip *First-In, First-Out* (FIFO). Proses kerja BFS dapat diringkas sebagai berikut:

1. Titik awal dimasukkan ke dalam antrian dan ditandai sebagai telah dikunjungi (atau sedang diproses).
2. Selama antrian tidak kosong, titik di bagian depan antrian dikeluarkan untuk diproses.
3. Semua titik tetangga dari titik yang baru diproses tersebut yang belum dikunjungi akan ditandai sebagai telah dikunjungi dan dimasukkan ke bagian belakang antrian.
4. Proses berlanjut hingga antrian menjadi kosong, yang berarti semua titik yang dapat dijangkau telah dikunjungi, atau hingga titik tujuan (solusi) ditemukan.



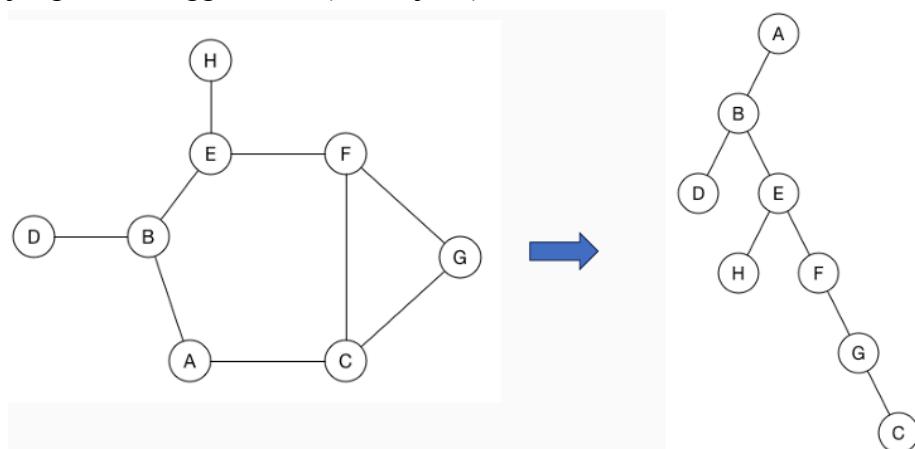
Gambar 3. Representasi traversal graf secara BFS
(sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir>)

2.3. Algoritma Depth First Search

Depth-First Search (DFS) adalah algoritma fundamental lainnya untuk melakukan pencarian atau traversal pada struktur data graf atau pohon. Berbeda dengan BFS yang menjelajah secara melebar, DFS bekerja dengan cara memulai dari titik akar (atau titik awal yang dipilih) dan menjelajahi sejauh mungkin di sepanjang satu cabang tertentu sebelum akhirnya melakukan backtracking (mundur) untuk menjelajahi cabang-cabang alternatif.

Implementasi DFS umumnya menggunakan struktur data *stack* dengan prinsip *Last-In, First-Out* (LIFO) secara eksplisit, atau secara implisit melalui mekanisme rekursi. Proses kerja DFS dapat diringkas sebagai berikut:

1. Titik awal dipilih dan ditandai sebagai telah dikunjungi.
2. Dari titik saat ini, pilih satu titik tetangga yang belum dikunjungi. Jadikan titik tetangga tersebut sebagai titik saat ini, tandai sebagai telah dikunjungi, dan ulangi proses ini (jika menggunakan rekursi, panggil fungsi DFS untuk titik tetangga tersebut; jika menggunakan tumpukan, masukkan titik saat ini ke tumpukan dan proses titik tetangga).
3. Jika dari titik saat ini tidak ada lagi titik tetangga yang belum dikunjungi (mencapai "jalan buntu" atau dead-end), algoritma melakukan backtrack. Jika menggunakan rekursi, fungsi akan kembali ke pemanggilnya (titik sebelumnya). Jika menggunakan tumpukan eksplisit, titik dari puncak tumpukan akan diambil (di-pop) untuk menjadi titik saat ini, dan dari sana algoritma mencoba menjelajahi cabang lain yang belum dikunjungi.
4. Proses ini berlanjut hingga semua titik yang dapat dijangkau dari titik awal telah dikunjungi atau hingga solusi (titik tujuan) ditemukan.



Gambar 4. Representasi traversal graf secara DFS
(sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir>)

2.4. Penerapan Bonus

2.4.1. Algoritma Bidirectional Search

Bidirectional Search (Pencarian Dua Arah) adalah sebuah strategi pencarian yang bekerja dengan menjalankan dua pencarian secara simultan: satu pencarian dimulai dari initial state (keadaan awal) menuju goal state (keadaan tujuan), dan pencarian lainnya dimulai

dari goal state menuju initial state. Proses pencarian ini akan berhenti ketika kedua jalur pencarian tersebut bertemu pada satu simpul di tengah.

2.5. Website

2.5.1. Frontend

Pengembangan Front End website menggunakan Next.js versi 15.3.1. Framework ini memungkinkan kita untuk membangun aplikasi web yang cepat dan responsif dengan integrasi yang baik dan cukup mudah. Dalam project ini, Next.js versi 15.3.1 dikombinasikan dengan React 19.0.0 sebagai library utama untuk membangun antarmuka. React memungkinkan pengembangan UI dengan pendekatan berbasis komponen yang modular dan reusable. Komponen-komponen ini akan memudahkan kita pada pengelolaan tampilan aplikasi.

Untuk styling, project menggunakan Tailwind CSS versi 4. Pendekatan ini memungkinkan kita untuk membuat design yang konsisten dan responsif dengan cepat melalui penggunaan class-class utility yang telah didefinisikan. Tailwind membantu dalam membangun tampilan yang menarik dengan lebih mudah dan ringkas.

Visualisasi jalur pencarian recipe diimplementasikan menggunakan kombinasi D3.js versi 7.9.0 dan React D3 Tree versi 3.6.6. D3.js merupakan library visualisasi data yang menurut kita cukup gampang digunakan, sementara React D3 Tree menyediakan komponen khusus untuk menampilkan struktur pohon yang interaktif dan customizable. Kedua library ini sangat berguna untuk merepresentasikan hasil pencarian jalur recipe dari algoritma BFS dan DFS dalam bentuk grafis yang mudah dipahami pengguna.

2.5.2. Backend

Pengembangan Back End website menggunakan bahasa Go (Golang) versi 1.23.0. Bahasa ini menawarkan konkurensi yang efisien melalui goroutines dan kemudahan dalam pengelolaan memori. Untuk membangun REST API, kami mengandalkan pustaka standar net/http dari Go. Pustaka ini digunakan untuk membuat endpoint API yang bertugas melayani permintaan pencarian jalur recipe dari frontend. Penanganan routing diatur menggunakan *http.ServeMux* dan logika untuk setiap endpoint diimplementasikan dalam fungsi-fungsi *handler* HTTP

Backend juga memanfaatkan library GoQuery versi 1.10.3 yang menyediakan fungsi-fungsi untuk melakukan parsing dan manipulasi dokumen HTML. Library ini memungkinkan web scraping data elemen-elemen Little Alchemy 2. Untuk mengoptimalkan pengiriman data, digunakan library Brotli versi 1.1.1 yang menyediakan algoritma kompresi efisien untuk komunikasi antara server dan client.

Implementasi algoritma pencarian menggunakan struktur data sendiri untuk merepresentasikan elemen-elemen dalam game Little Alchemy 2. Setiap node merepresentasikan elemen game, sementara edge merepresentasikan kombinasi yang dapat menghasilkan elemen baru. Algoritma BFS (Breadth-First Search) dan DFS (Depth-First Search) menawarkan alternatif pencarian yang dapat menemukan solusi, baik 1 solusi maupun banyak solusi yang sudah dioptimasi dengan multithread.

API backend mengembalikan hasil pencarian dalam format struktur data tersendiri yang berisi rangkaian langkah kombinasi dari elemen awal hingga elemen target, yang kemudian divisualisasikan oleh frontend menggunakan D3.js dan React D3 Tree.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1. Dekomposisi Masalah

Permasalahan utama dari permainan Little Alchemy II ini adalah mencari satu atau beberapa path/rute resep untuk suatu elemen target dari empat elemen dasar yaitu “Air”, “Water”, “Fire”, dan “Earth”.

Pada website, pengguna dapat memasukkan nama suatu elemen yang akan menjadi target elemen. Target elemen inilah yang akan dicari resep lengkapnya. Pengguna juga bisa memilih algoritma yang akan digunakan untuk menyelesaikan pencarian resep tersebut. Terdapat dua algoritma yang dapat dipilih, yaitu algoritma Breadth First Search (BFS) dan Depth First Search (DFS). Selain itu, pengguna juga bisa memilih untuk mencari satu resep saja atau juga memilih lebih dari satu resep.

Untuk bisa melakukan pencarian resep lengkap dari suatu elemen, harus dilakukan *scrapping* data terlebih dahulu untuk mendapatkan data lengkap semua resep dari setiap elemen. Data hasil *scrapping* inilah yang akan digunakan sebagai acuan pada algoritma BFS dan DFS.

Setelah pengguna memilih nama elemen, algoritma pencarian, dan jumlah resep yang ingin dicari, akan ditampilkan hasil dari pencarian tersebut pada frontend dalam bentuk tree.

3.2. Pemetaan Masalah Menjadi Elemen-Elemen Algoritma

Untuk memecahkan masalah pencarian recipe dalam game Little Alchemy 2, langkah awal yang dilakukan adalah merepresentasikan seluruh elemen dan resep dalam bentuk struktur graf. Dengan demikian, proses pencarian elemen target dapat dimodelkan sebagai pencarian jalur dalam graf dari elemen dasar menuju elemen target. Berikut adalah pemetaan komponen utama dalam game ke dalam elemen algoritma graf:

- Simpul (Node)
Setiap elemen unik dalam permainan direpresentasikan sebagai simpul dalam graf
- Sisi (Edge)
Sebuah resep yang terdiri dari dua elemen induk (P_1+P_2) yang menghasilkan elemen anak (C) dapat direpresentasikan sebagai sisi yang mengarah dari C ke P_1 dan P_2 . Dalam kode direpresentasikan melalui struktur data ChildToParents.

3.2.1. Pemetaan Masalah Menjadi Bentuk BFS

- Antrian (Queue)
Digunakan untuk menyimpan “state” pencarian. Setiap state (*BFSMPStateBackward* dalam pathfinding/bfs/BFS.go) dapat berisi:
 - ElementsToDeconstruct : Daftar elemen yang masih perlu dipecah menjadi komponen-komponennya.

- PathTakenSoFar : Urutan langkah resep yang telah diambil dari target awal hingga ke elemen-elemen yang sedang dipertimbangkan untuk didekonstruksi.
- elementsInCurrentPathTree : Map untuk melacak elemen yang sudah ada dalam pohon jalur saat ini untuk deteksi siklus sederhana.
- Proses Ekspansi
Saat sebuah state diambil dari antrian, salah satu elemen yang bukan elemen dasar dari ElementsToDeconstruct dipilih. Untuk elemen ini, semua resep (pasangan elemen induk) yang menghasilkannya akan dieksplorasi. Setiap resep akan menghasilkan state baru yang ditambahkan ke antrian. State baru ini akan berisi elemen-elemen induk dari resep tersebut (yang perlu didekonstruksi lebih lanjut) dan path yang diperbarui.
- Kondisi Berhenti
Pencarian untuk satu *branch path* akan berhenti ketika semua ElementsToDeconstruct dalam state tersebut adalah elemen dasar. Rute lengkap kemudian direkonstruksi. Pencarian keseluruhan dapat berhenti setelah sejumlah maxPaths rute unik ditemukan atau semua kemungkinan resep telah dicoba tapi tidak ada yang berhasil.

3.2.2 Pemetaan Masalah Menjadi Bentuk DFS

- Stack (Implisit melalui Rekursi)
Panggilan fungsi rekursif (dfsRecursiveHelperString atau dfsRecursiveHelperForWorkerPath) secara implisit menggunakan call stack untuk melacak jalur pencarian.
- Struktur Data Pendukung
 - pathSteps : Map untuk menyimpan langkah resep yang valid yang ditemukan selama penelusuran untuk satu jalur.
 - currentlySolving : Map untuk mendeteksi siklus dalam path rekursif saat ini, mencegah penelusuran tak terbatas.
 - memo (Memoization) : Map untuk menyimpan hasil apakah sebuah elemen dapat dibuat atau tidak, untuk menghindari komputasi berulang pada sub-problem yang sama.
- Proses Ekspansi
Untuk sebuah target element, algoritma mencoba setiap resep satu per satu. Untuk setiap resep, ia akan secara rekursif mencoba membuat kedua elemen induknya.
- Kondisi Berhenti (Rekursi)
 - Elemen adalah elemen dasar (kasus dasar sukses).
 - Elemen tidak memiliki resep (kasus dasar gagal).
 - Siklus terdeteksi.
 - Hasil untuk elemen sudah ada di memo.
 - Semua resep untuk elemen tersebut telah dicoba dan tidak ada yang berhasil.

3.3. Fitur fungsional dan Arsitektur Aplikasi Web

3.3.1. Fitur Fungsional

ID	Fitur Fungsional	Penjelasan
F01	P/L menyediakan fasilitas bagi pengguna untuk menentukan nama elemen target yang akan dicari	Pengguna dapat menginput nama elemen target yang ingin ditemukan dalam pencarian recipe.
F02	P/L menyediakan fasilitas bagi pengguna untuk menentukan metode apa yang ingin digunakan untuk mencari recipe (BFS/DFS)	Pengguna dapat memilih algoritma pencarian yang digunakan, yaitu Breadth-First Search (BFS) atau Depth-First Search (DFS).
F03	P/L menyediakan fasilitas bagi pengguna untuk memilih untuk menemukan 1 recipe atau multiple recipe	Pengguna dapat memilih apakah ingin mencari satu recipe saja atau beberapa recipe sekaligus.
F04	P/L menyediakan fasilitas bagi pengguna untuk memilih seberapa banyak recipe yang ingin dicari pada kasus multiple recipe	Pengguna dapat menentukan jumlah maksimum recipe yang ingin ditemukan jika memilih opsi pencarian multiple recipe.
F05	P/L menyediakan fasilitas bagi pengguna untuk melakukan pencarian recipe dengan menekan 1 tombol “GO”	Pengguna dapat memulai proses pencarian recipe dengan menekan tombol “GO”.
F06	P/L menyediakan fasilitas bagi pengguna untuk melihat hasil recipe dalam bentuk tree (baik untuk single recipe maupun multiple recipe)	Pengguna dapat melihat hasil pencarian recipe yang divisualisasikan dalam bentuk struktur pohon (tree).
F07	P/L menyediakan fasilitas bagi pengguna untuk melihat hasil dari pencarian yaitu waktu, node visited	Pengguna dapat melihat informasi hasil pencarian berupa waktu pencarian dan jumlah simpul (node) yang dikunjungi selama proses pencarian.

3.3.2. Arsitektur Aplikasi Web

Aplikasi web yang kami bangun menggunakan arsitektur berbasis **Next.js**, dengan repository yang terpisah antara komponen frontend dan logika pencarian di backend. Berikut adalah elemen utama dari arsitektur aplikasi web ini:

1. Komponen Frontend
 - a. Komponen Halaman Utama

Komponen utama yang mengelola user interface pencarian recipe, termasuk input elemen target, pemilihan metode pencarian (BFS/DFS), jumlah recipe, dan tombol

“GO”. Komponen ini menggunakan state dan handler untuk mengatur logika UI dan interaksi pengguna.

b. Komponen Visualisasi Tree

Bertanggung jawab untuk menampilkan hasil pencarian dalam bentuk tree interaktif yang menampilkan gambar elemen menggunakan React D3 Tree. Komponen ini menerima data struktur tree dari backend dan memetakannya menjadi node dan edge.

c. Komponen Result

Menampilkan informasi tambahan dari hasil pencarian, seperti waktu eksekusi dan jumlah node yang dikunjungi. Komponen ini memberikan feedback kinerja dari algoritma pencarian kepada pengguna.

2. Penggunaan State dan Hook

Aplikasi menggunakan hook React seperti useState dan useEffect untuk mengelola state aplikasi dan efek samping. Ini digunakan dalam:

- Mengelola input pengguna (target elemen, metode pencarian, jumlah recipe).
- Menyimpan hasil pencarian dari backend.
- Menampilkan dan memperbarui tree visual secara dinamis saat data berubah.

3. Komunikasi Frontend - Backend (Fetch API)

Komponen frontend mengirim permintaan ke backend menggunakan fetch. Permintaan dikirim ke endpoint API milik backend berbasis Go, dan menerima response dalam format JSON yang kemudian diolah untuk visualisasi dan informasi tambahan.

4. Interaksi dan Pengendalian Aplikasi

Aplikasi memiliki berbagai handler dan fungsi kontrol seperti:

- Handler input dan tombol “GO”
- Handler pemilihan metode pencarian
- Handler pemrosesan dari backend untuk BFS dan DFS

Fungsi-fungsi ini membantu menjaga aliran logika aplikasi agar tetap konsisten dan mudah diikuti oleh pengguna.

5. Backend API dan Logika Pencarian (Go)

Backend dibangun menggunakan Golang v1.23.0, dengan pustaka net/http untuk routing. Setiap permintaan pencarian akan diproses sebagai berikut:

- Scraping data elemen dari Little Alchemy 2 menggunakan GoQuery.
- Menjalankan algoritma BFS atau DFS (dengan opsi single path/ multiple path dan multithreading).
- Mengembalikan hasil pencarian dalam bentuk struktur pohon dan statistik pencarian.

3.4. Ilustrasi Kasus

3.4.1. Ilustrasi Kasus dengan Algoritma BFS dalam Kasus “*Find Recipe*”

Algoritma BFS diimplementasikan secara iteratif dengan memanfaatkan struktur data *queue* untuk masing-masing node yang dikunjunginya Berikut contoh implementasinya

untuk mencari elemen "Clay". BFS memulai dengan elemen target, "Clay", dan secara bertahap memperluas pencarian ke elemen-elemen yang dapat membuatnya (induknya), lalu induk dari induknya, dan seterusnya, level demi level. Tujuannya adalah menemukan jalur dari "Clay" kembali ke elemen-elemen dasar. Misalkan "Clay" memiliki resep Mud + Stone dan Mineral + Stone. BFS akan mengeksplorasi semua kemungkinan ini pada level pertama. State awal dalam antrian adalah {ElementsToDeconstruct: ["Clay"], PathTakenSoFar: []}. BFS kemudian mengeluarkan state ini, dan untuk setiap resep "Clay" (misalnya Mud + Stone), ia akan membuat state baru seperti {ElementsToDeconstruct: ["Mud", "Stone"], PathTakenSoFar: [Clay = Mud + Stone]} dan memasukkannya ke antrian.

Selanjutnya, BFS akan memproses state-state berikutnya dari antrian. Jika ia memproses state yang mengandung "Mud" dan "Stone", ia akan mencoba menguraikan salah satunya, misalnya "Mud" menjadi Water + Earth. State baru yang mungkin terbentuk adalah {ElementsToDeconstruct: ["Water", "Earth", "Stone"], PathTakenSoFar: [Clay = Mud + Stone, Mud = Water + Earth]}. Proses ini berlanjut, menjelajahi semua kombinasi induk secara level per level, hingga semua elemen dalam ElementsToDeconstruct adalah elemen dasar. Karena BFS menjelajah level demi level, algoritma ini cenderung menemukan jalur terpendek.

3.4.2. Ilustrasi Kasus dengan Algoritma DFS dalam Kasus "*Find Recipe*"

Algoritma DFS diimplementasikan secara rekursif untuk masing-masing node yang dikunjunginya, berikut contoh implementasinya untuk mencari elemen "Clay". Pencarian untuk "Clay" dimulai dengan memilih satu-satunya resep yang tersedia, yaitu Clay = Mud + Stone. kemudian algoritma tersebut akan fokus untuk menyelesaikan "Mud" terlebih dahulu. Untuk "Mud", DFS menemukan resep Mud = Water + Earth. Karena "Water" dan "Earth" adalah elemen dasar, cabang pencarian untuk "Mud" ini berhasil diselesaikan dan langkahnya dicatat. Setelah "Mud" terpecahkan, DFS beralih ke elemen "Stone".

Untuk "Stone", DFS memilih resep Sand = Earth + Pressure. "Earth" adalah elemen dasar, namun "Pressure" bukan. Karena Earth sudah merupakan *base element*, DFS akan melanjutkan pencarian untuk "Pressure", misalnya dengan resep Stone = Air + Air. "Air" adalah elemen dasar. Karena kedua elemen pembentuk "Pressure" adalah "Air" serta merupakan *base element*, maka "Pressure" dapat dibuat. Karena Pressure terbentuk, maka "Stone" juga berhasil diselesaikan. Setelah semua prasyarat untuk "Clay" terpenuhi, DFS menyimpulkan bahwa "Clay" dapat dibuat dan merekonstruksi seluruh jalur resep yang telah ditemukan.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

1. Struktur Data Utama

A. Struktur Data yang Digunakan untuk Response API

- BFSRequest dan DFSRequest

Digunakan oleh frontend untuk mengirim permintaan pencarian recipe menggunakan algoritma BFS.

- BFSResponse

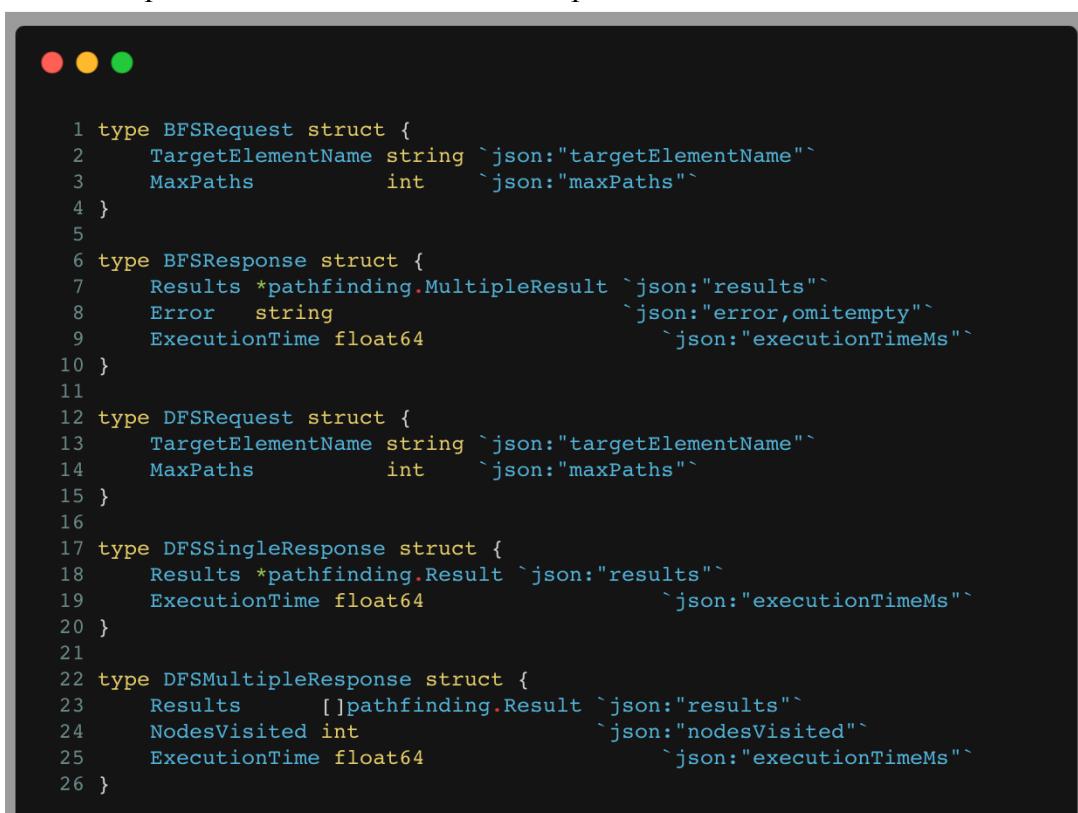
Digunakan backend untuk mengembalikan hasil pencarian BFS.

- DFSSingleResponse

Response untuk DFS versi single recipe:

- DFSMultipleResponse

Response untuk DFS versi multi-recipe:



```
1 type BFSRequest struct {
2     TargetElementName string `json:"targetElementName"`
3     MaxPaths         int    `json:"maxPaths"`
4 }
5
6 type BFSResponse struct {
7     Results *pathfinding.MultipleResult `json:"results"`
8     Error   string                      `json:"error,omitempty"`
9     ExecutionTime float64                `json:"executionTimeMs"`
10 }
11
12 type DFSRequest struct {
13     TargetElementName string `json:"targetElementName"`
14     MaxPaths         int    `json:"maxPaths"`
15 }
16
17 type DFSSingleResponse struct {
18     Results *pathfinding.Result `json:"results"`
19     ExecutionTime float64        `json:"executionTimeMs"`
20 }
21
22 type DFSMultipleResponse struct {
23     Results     []pathfinding.Result `json:"results"`
24     NodesVisited int                  `json:"nodesVisited"`
25     ExecutionTime float64            `json:"executionTimeMs"`
26 }
```

B. Struktur Data untuk Web Scrapping

- Element

Digunakan untuk menyimpan hasil scraping dari website *Little Alchemy 2*

```
1 type Element struct {
2     Name      string      `json:"name"`
3     Recipes   [][]string `json:"recipes"`
4     Tier      int         `json:"tier"`
5     ImageURL string     `json:"imageUrl,omitempty"`
6 }
```

C. Struktur Data Hasil Pencarian

- PathStep
Mewakili satu langkah dalam jalur pembuatan elemen
- Result
Mewakili hasil pencarian 1 jalur menuju 1 elemen
- MultipleResult
Mewakili hasil pencarian beberapa jalur

```
1 type PathStep struct {
2     ChildName  string
3     Parent1Name string
4     Parent2Name string
5 }
6
7 type Result struct {
8     Path        []PathStep
9     NodesVisited int
10 }
11
12 type MultipleResult struct {
13     Results    []Result
14 }
```

D. Struktur Data Tree

Berbentuk objek atau array objek yang diformat sesuai dengan permintaan dari library react-d3-tree untuk menampilkan visualisasi tree. Setiap node memiliki properti seperti name, attributes, dan children.

2. Fungsi dan Prosedur Utama

A. DFSPATHFINDINGHANDLER

Fungsi ini bertugas menangani permintaan pencarian *recipe* menggunakan algoritma *Depth First Search* (DFS) untuk satu rute solusi.

```

1  func DFSPathfindingHandler(w http.ResponseWriter, r *http.Request) {
2      w.Header().Set("Access-Control-Allow-Origin", "*")
3      w.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS")
4      w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
5
6      if r.Method == "OPTIONS" {
7          w.WriteHeader(http.StatusOK)
8          return
9      }
10     // Method harus POST
11     if r.Method != http.MethodPost {
12         http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
13         return
14     }
15
16     var req DFSRequest
17     if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
18         http.Error(w, "Invalid request body", http.StatusBadRequest)
19         return
20     }
21
22     graph, err := loadrecipes.LoadBiGraph("elements_filtered.json")
23     if err != nil {
24         respondWithError(w, "Failed to load graph", http.StatusInternalServerError)
25         return
26     }
27
28     start := time.Now()
29     result, err := dfs.DFSFindPathString(graph, req.TargetElementName)
30
31     if err != nil {
32         respondWithError(w, "Failed to find paths: "+err.Error(), http.StatusInternalServerError)
33         return
34     }
35
36     executionTime := time.Since(start).Seconds() * 1000
37
38     w.Header().Set("Content-Type", "application/json")
39     json.NewEncoder(w).Encode(DFSSingleResponse{
40         Results:    result,
41         ExecutionTime: float64(executionTime),
42     })
43 }

```

B. DFSMultiplePathFindingHandler

Fungsi ini serupa dengan DFSPathfindingHandler, namun dirancang untuk mencari beberapa jalur solusi (*multiple recipes*) menggunakan algoritma DFS.

```

1 func DFSMultiplePathfindingHandler(w http.ResponseWriter, r *http.Request) {
2     w.Header().Set("Access-Control-Allow-Origin", "*")
3     w.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS")
4     w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
5
6     if r.Method == "OPTIONS" {
7         w.WriteHeader(http.StatusOK)
8         return
9     }
10
11    if r.Method != http.MethodPost {
12        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
13        return
14    }
15
16    var req DFSRequest
17    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
18        http.Error(w, "Invalid request body", http.StatusBadRequest)
19        return
20    }
21
22    graph, err := loadrecipes.LoadBiGraph("elements_filtered.json")
23    if err != nil {
24        respondWithError(w, "Failed to load graph", http.StatusInternalServerError)
25        return
26    }
27
28    start := time.Now()
29    result, nodesVisited, err := dfs.DFSFindMultiplePaths(graph, req.TargetElementName, req.MaxPaths)
30    executionTime := time.Since(start).Seconds() * 1000
31    if err != nil {
32        respondWithError(w, "Failed to find paths: "+err.Error(), http.StatusInternalServerError)
33        return
34    }
35
36    w.Header().Set("Content-Type", "application/json")
37    json.NewEncoder(w).Encode(DFSMultipleResponse{
38        Results:      result.Results,
39        NodesVisited: nodesVisited,
40        ExecutionTime: float64(executionTime),
41    })
42 }

```

C. BFSPathFindingHandler

Fungsi ini menangani permintaan pencarian *recipe* menggunakan algoritma *Breadth First Search* (BFS). Fungsi ini juga menerima jumlah maksimal rute yang diinginkan sehingga bisa digunakan untuk *single recipe* dan juga *multiple recipes*.

```

1  func BFSPathfindingHandler(w http.ResponseWriter, r *http.Request) {
2      w.Header().Set("Access-Control-Allow-Origin", "*")
3      w.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS")
4      w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
5
6      if r.Method == "OPTIONS" {
7          w.WriteHeader(http.StatusOK)
8          return
9      }
10
11     if r.Method != http.MethodPost {
12         http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
13         return
14     }
15
16     var req BFSRequest
17     if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
18         http.Error(w, "Invalid request body", http.StatusBadRequest)
19         return
20     }
21
22     graph, err := loadrecipes.LoadBiGraph("elements_filtered.json")
23     if err != nil {
24         respondWithError(w, "Failed to load graph", http.StatusInternalServerError)
25         return
26     }
27
28     start := time.Now()
29     result, err := bfs.BFSFindMultiplePaths(graph, req.TargetElementName, req.MaxPaths)
30     executionTime := time.Since(start).Seconds() * 1000
31
32     if err != nil {
33         respondWithError(w, "Failed to find paths: "+err.Error(), http.StatusInternalServerError)
34         return
35     }
36
37     w.Header().Set("Content-Type", "application/json")
38     json.NewEncoder(w).Encode(BFSResponse{
39         Results:    result,
40         Error:      "",
41         ExecutionTime: float64(executionTime),
42     })
43 }

```

D. LoadBigraph

Fungsi ini bertanggung jawab untuk memuat data elemen dan recipe dari sebuah JSON.

```

● ● ●
1 func LoadBiGraph(filepath string) (*BiGraphAlchemy, error) {
2     file, err := os.Open(filepath)
3     if err != nil {
4         return nil, err
5     }
6     defer file.Close()
7
8     data, err := ioutil.ReadAll(file)
9     if err != nil {
10        return nil, err
11    }
12
13     var elements []ElementInput
14     err = json.Unmarshal(data, &elements)
15     if err != nil {
16        return nil, err
17    }
18
19     graphData := &BiGraphAlchemy{
20         ChildToParents: make(map[string][]PairMats),
21         ParentPairToChild: make(map[PairMats][]string),
22         BaseElements: map[string]bool{
23             "Air": true,
24             "Fire": true,
25             "Earth": true,
26             "Water": true,
27         },
28         AllElements: make(map[string]bool),
29     }
30
31     for baseElem := range graphData.BaseElements {
32         graphData.AllElements[baseElem] = true
33     }
34
35     for _, element := range elements {
36         graphData.AllElements[element.Name] = true
37
38         if graphData.BaseElements[element.Name] {
39             continue
40         }
41
42         validRecipesForChild := []PairMats{}
43         for _, recipe := range element.Recipes {
44             if len(recipe) != 2 {
45                 log.Printf("[WARNING] Invalid recipe format for element %s: %v. Skipping recipe.", element.Name, recipe)
46                 continue
47             }
48
49             parent1, parent2 := recipe[0], recipe[1]
50             graphData.AllElements[parent1] = true
51             graphData.AllElements[parent2] = true
52
53             pair := ConstructPair(parent1, parent2)
54
55             if !ContainsString(graphData.ParentPairToChild[pair], element.Name) {
56                 graphData.ParentPairToChild[pair] = append(graphData.ParentPairToChild[pair], element.Name)
57             }
58
59             alreadyExists := false
60             for _, existingPair := range validRecipesForChild {
61                 if existingPair == pair {
62                     alreadyExists = true
63                     break
64                 }
65             }
66             if !alreadyExists {
67                 validRecipesForChild = append(validRecipesForChild, pair)
68             }
69         }
70
71         if len(validRecipesForChild) > 0 {
72             graphData.ChildToParents[element.Name] = validRecipesForChild
73         } else if !graphData.BaseElements[element.Name] {
74             log.Printf("[INFO] Element %s (non-base) has no valid recipes after loading.", element.Name)
75         }
76     }
77
78     for pair := range graphData.ParentPairToChild {
79         sort.Strings(graphData.ParentPairToChild[pair])
80     }
81
82     log.Printf("[INFO] Data successfully loaded from '%s'. Total Unique Elements: %d. Unique Parent Pairs: %d. Child-to-Parent Relations: %d.\n",
83     filepath, len(graphData.AllElements), len(graphData.ParentPairToChild), len(graphData.ChildToParents))
84
85     return graphData, nil
86 }

```

E. FetchBFSPaths

Fungsi di sisi frontend untuk mengirim permintaan pencarian BFS ke *backend*. Fungsi ini mengirim permintaan POST ke endpoint /api/pathfinding/bfs dengan menyertakan nama elemen target dan *maxPaths*.

```
1  export async function fetchBFSPaths(targetElementName, maxPaths) {
2    try {
3      const response = await fetch(` ${API_BASE_URL} /api/pathfinding/bfs`, {
4        method: 'POST',
5        headers: {
6          'Content-Type': 'application/json',
7        },
8        body: JSON.stringify({
9          targetElementName,
10         maxPaths,
11       }),
12     });
13
14    if (!response.ok) {
15      const errorData = await response.json();
16      throw new Error(errorData.error || 'Failed to fetch BFS paths');
17    }
18
19    const data = await response.json();
20    return data.results;
21  } catch (error) {
22    console.error('Error fetching BFS paths:', error);
23    throw error;
24  }
25 }
```

F. FetchDFSPaths

Fungsi di sisi frontend yang mengirimkan permintaan untuk pencarian DFS yang *single recipe*.

```
● ○ ●
1 export async function fetchDFSPaths(targetElementName, maxPaths) {
2   try {
3     const response = await fetch(`${API_BASE_URL}/api/pathfinding/dfs-single`, {
4       method: 'POST',
5       headers: {
6         'Content-Type': 'application/json',
7       },
8       body: JSON.stringify({
9         targetElementName,
10        maxPaths,
11      }),
12    );
13
14   if (!response.ok) {
15     const errorData = await response.json();
16     throw new Error(errorData.error || 'Failed to fetch DFS paths');
17   }
18
19   const data = await response.json();
20   return data.results;
21 } catch (error) {
22   console.error('Error fetching DFS paths:', error);
23   throw error;
24 }
25 }
```

G. FetchDFSMultiplePaths

Fungsi di sisi frontend yang mengirimkan permintaan untuk pencarian DFS yang *multiple recipe*.

```
● ○ ●
1 export async function fetchDFSMultiplePaths(targetElementName, maxPaths) {
2   try {
3     const response = await fetch(`${API_BASE_URL}/api/pathfinding/dfs-multiple`, {
4       method: 'POST',
5       headers: {
6         'Content-Type': 'application/json',
7       },
8       body: JSON.stringify({
9         targetElementName,
10        maxPaths,
11      }),
12    );
13
14   if (!response.ok) {
15     const errorData = await response.json();
16     throw new Error(errorData.error || 'Failed to fetch DFS multiple paths');
17   }
18
19   const data = await response.json();
20   return data.results;
21 } catch (error) {
22   console.error('Error fetching DFS multiple paths:', error);
23   throw error;
24 }
25 }
```

4.2. Tata Cara Penggunaan Program

4.2.1. Cara Menjalankan Secara Lokal

1. Build Front-End

Masuk ke direktori “Tubes2_FE_SayMyName”

Install node modules:

```
npm install
```

Kemudian Run frontend dengan command:

```
npm run dev
```

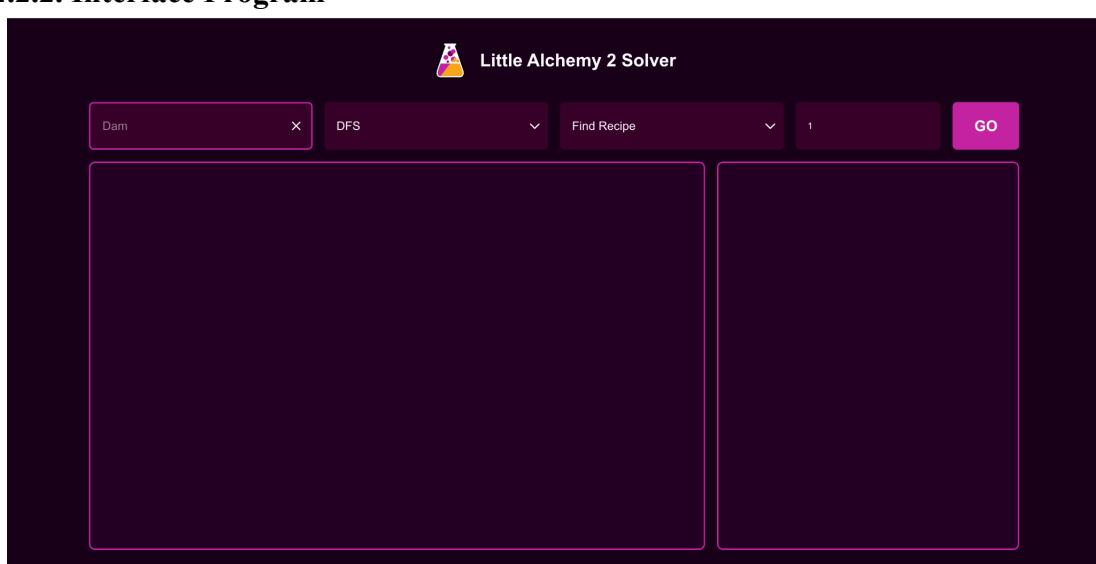
2. Dari direktori *Tubes2_FE_SayMyName*, pindah ke direktori *Tubes2_BE_SayMyName*:

Kemudian, run program server untuk menjalankan website:

```
go run main_server.go
```

3. Kemudian buka local host <http://localhost:3000/> pada browser
4. Pada website, pilih elemen yang akan dicari dengan memasukkan nama elemennya pada text box dan klik nama elemen
5. Pilih algoritma pencarian yang mau dipilih.
6. Pilih tipe recipe yang akan di-output-kan. Jika memilih “Multiple Recipe”, masukkan parameter berapa resep yang ingin terbuat.
7. Klik GO

4.2.2. Interface Program



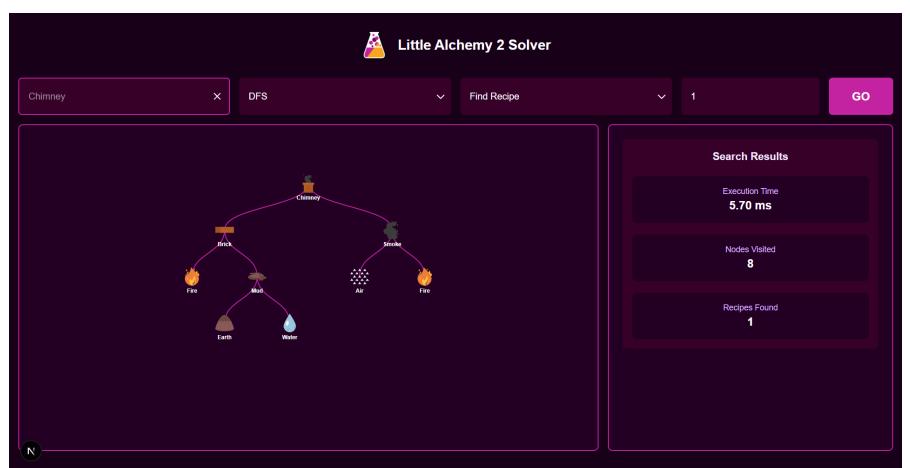
Website dirancang simple dengan single-page yang sudah dapat mencakup semua aspek yang diperlukan dalam pencarian recipe. Single-page dipilih untuk kemudahan dan keringkasan dalam penggunaannya.

4.3. Hasil Pengujian

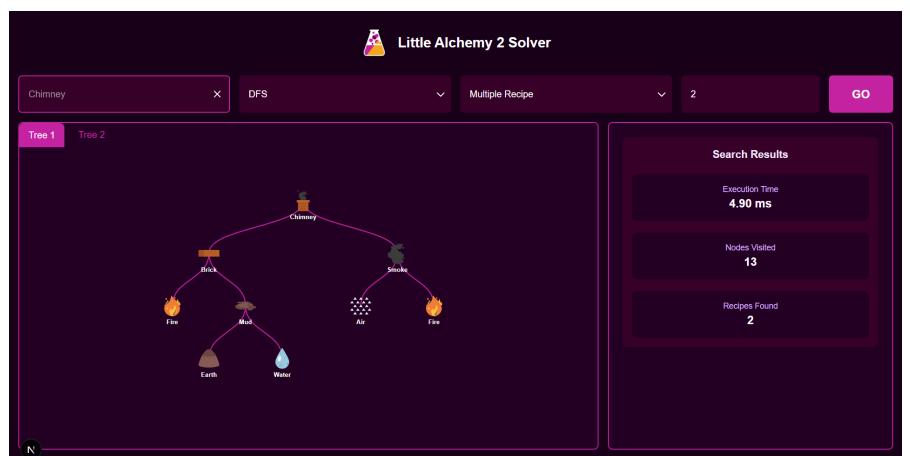
4.3.1. Test Case 1

Chimney	
Algoritma	Hasil

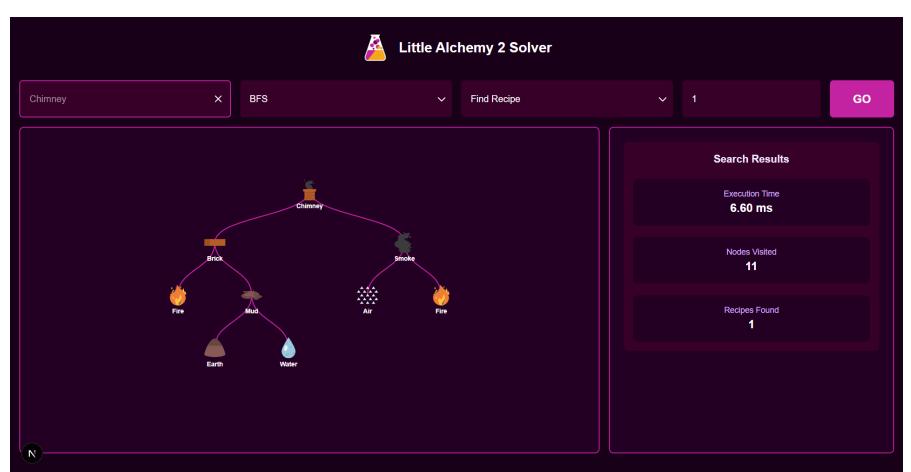
DFS (Single Recipe)



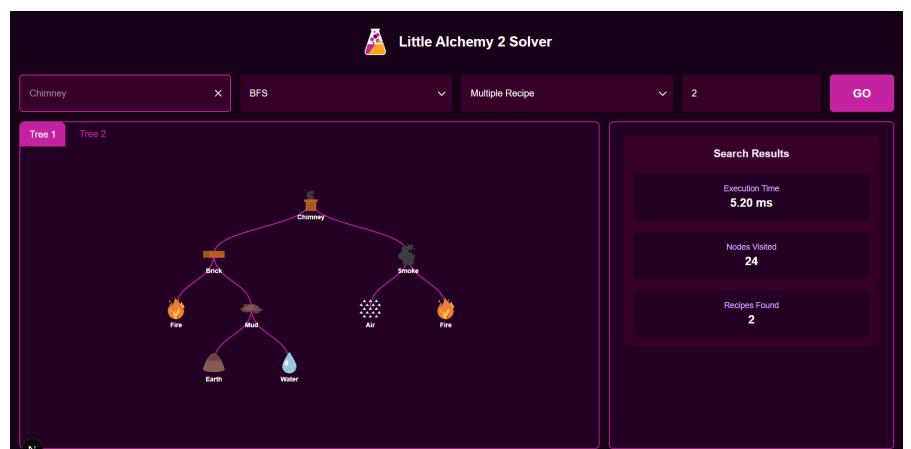
DFS (Multiple Recipe)



BFS (Single Recipe)



BFS (Multi Recipe)



4.3.2. Test Case 2

Chimney	
Algoritma	Hasil

DFS (Single Recipe)

Little Alchemy 2 Solver

Water X DFS Find Recipe 1 GO

No Path Found

Search Results

- Execution Time **6.50 ms**
- Nodes Visited **1**
- Recipes Found **1**

BFS (Single Recipe)

Little Alchemy 2 Solver

Water X BFS Find Recipe 1 GO

No Path Found

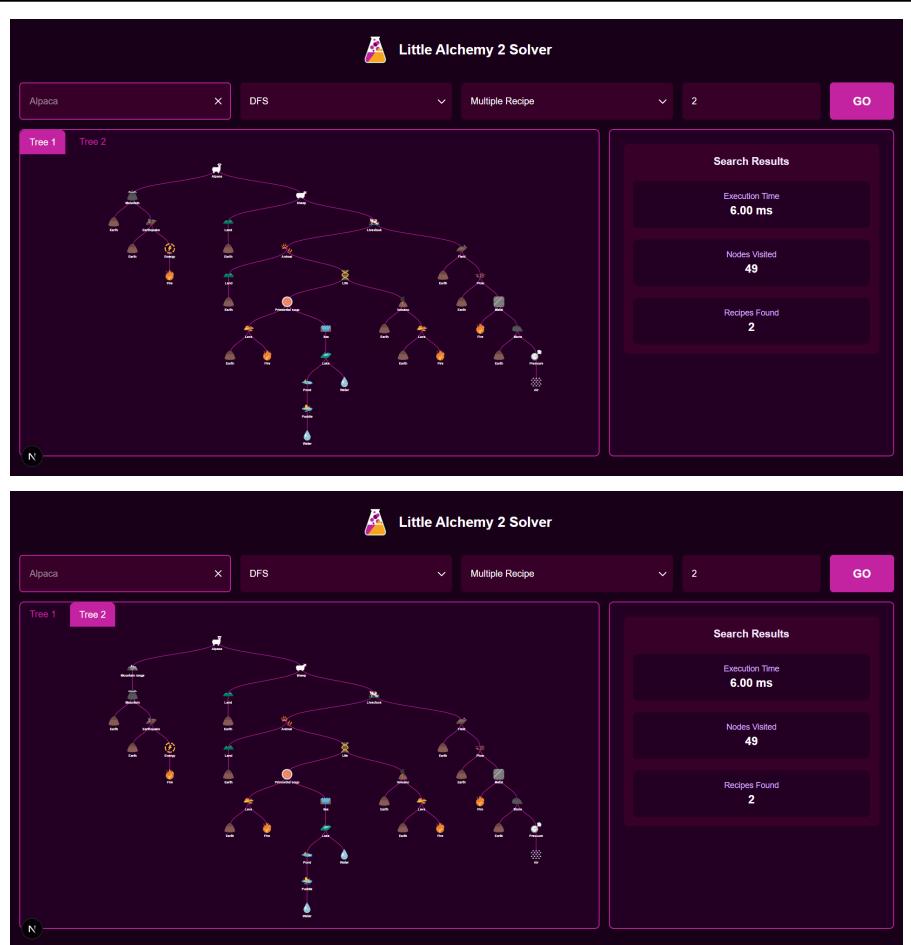
Search Results

- Execution Time **5.70 ms**
- Nodes Visited **1**
- Recipes Found **1**

4.3.3. Test Case 3

Alpaca	
Algoritma	Hasil
DFS (Single Recipe)	<p>Little Alchemy 2 Solver</p> <p>Alpaca X DFS Find Recipe 1 GO</p> <p>No Path Found</p> <p>Search Results</p> <ul style="list-style-type: none">Execution Time 5.40 msNodes Visited 25Recipes Found 1

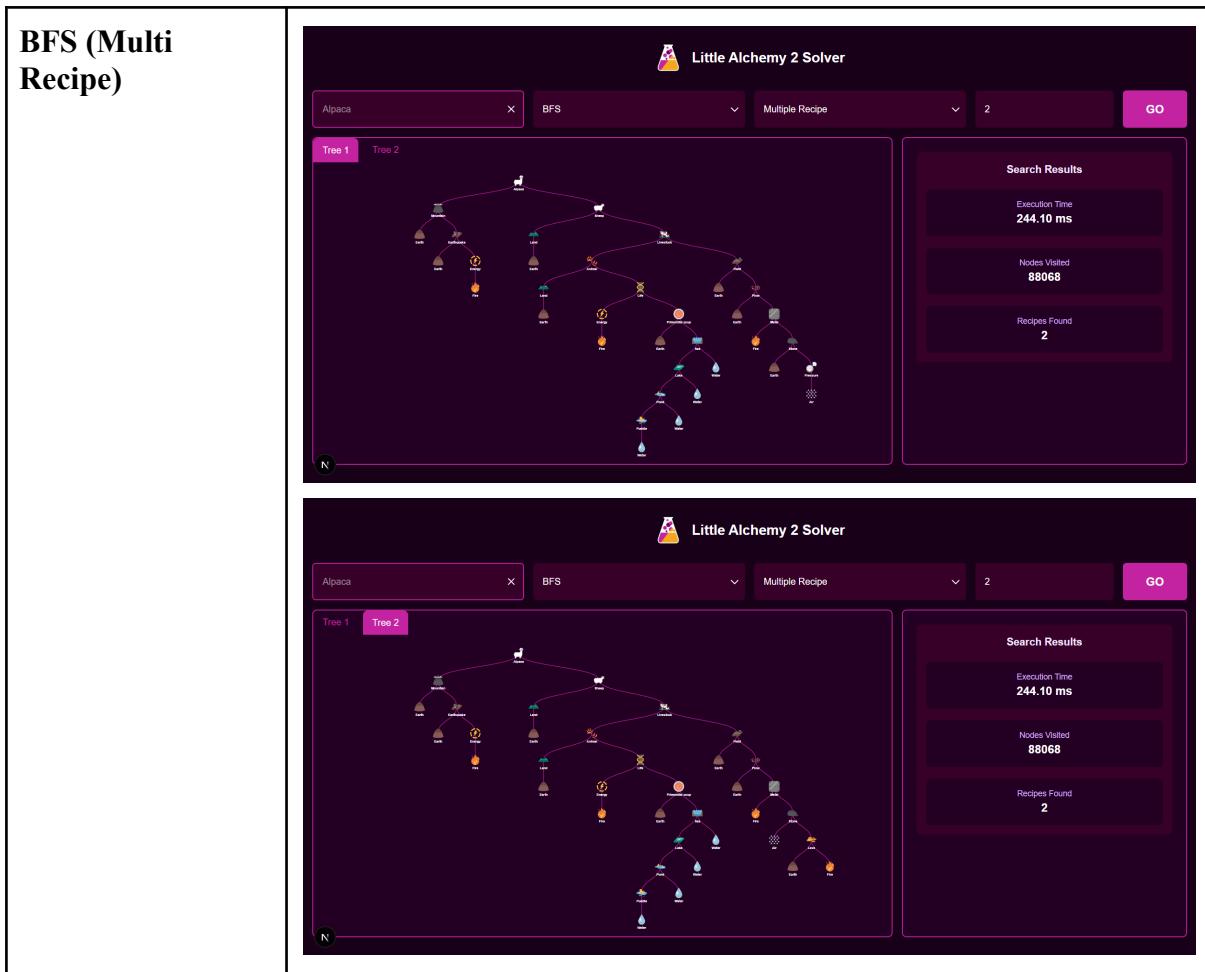
DFS (Multi Recipe)



BFS (Single Recipe)



BFS (Multi Recipe)



BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari tugas besar 2 IF2211 Strategi algoritma ini, kami telah berhasil membuat aplikasi web yang bisa digunakan untuk melakukan pencarian resep suatu elemen dari game Little Alchemy II menggunakan algoritma Breadth First Search (BFS) dan Depth First Search (DFS). Untuk menyelesaikan tugas ini, kami menggunakan bahasa pemrograman Golang pada bagian *backend* dan Next JS pada bagian *frontend*.

5.2. Saran

Adapun saran perbaikan untuk kelompok kami selama penggerjaan tugas besar 2 ini antara lain adalah penggerjaan yang seharusnya bisa dicicil lebih awal sehingga tugas bisa dikerjakan dengan baik dan kami juga punya waktu lebih untuk melakukan perancangan terstruktur untuk mempermudah eksekusi dan memungkinkan lebih banyak melakukan eksplorasi pada penggerjaan tugas terkhusus pada bagian bonus. Namun, terlepas dari itu semua, pada tugas besar ini kami belajar banyak hal baru awalnya memang menjadi tantangan seperti penggunaan bahasa Golang, penggunaan *goroutine* untuk concurrency, Framework Gin untuk backend dan NextJS untuk Frontend, *scrapping* data dari website, serta penggunaan docker dan deploy website. Hal-hal tersebut tentunya menjadi pengalaman berharga dan memberikan banyak pengetahuan baru bagi kami.

LAMPIRAN

Repository:

Front-End: https://github.com/rafifrs/Tubes2_FE_SayMyName.git

Back-End: https://github.com/Starath/Tubes2_BE_SayMyName.git

Link Video: <https://youtu.be/rMC5FlFq78?si=MRM43uclVPQKQ9>

Tabel Progress:

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.		✓

REFERENSI

- [1]. R. Munir, "Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1" Institut Teknologi Bandung, 2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf). [Diakses: Mei. 11, 2025].
- [2]. R. Munir, "Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 2" Institut Teknologi Bandung, 2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf). [Diakses: Mei. 11, 2025].