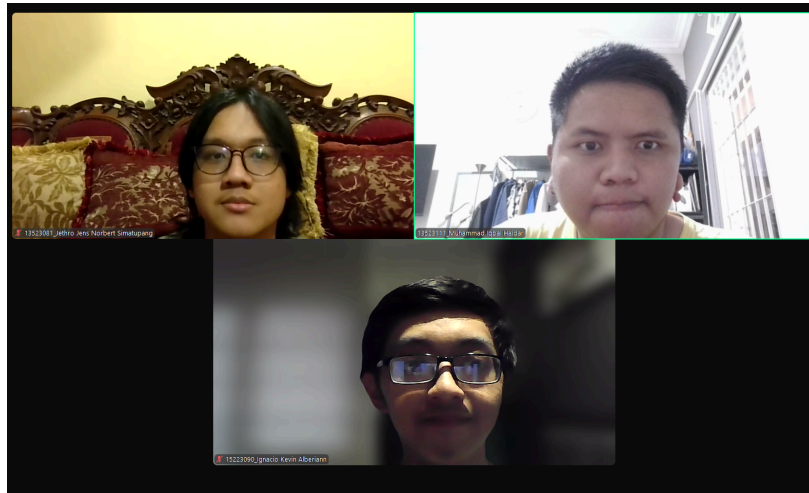


**LAPORAN TUGAS BESAR 2**  
**IF2211 STRATEGI ALGORITMA**  
**SEMESTER II TAHUN 2024/2025**

PEMANFAATAN ALGORITMA BFS DAN DFS DALAM PENCARIAN  
RECIPE PADA PERMAINAN LITTLE ALCHEMY 2



**DISUSUN OLEH:**

Jethro Jens Norbert Simatupang	13523081
Muhammad Iqbal Haidar	13523111
Ignacio Kevin Alberiann	15223090

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2025**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I DESKRIPSI TUGAS.....</b>	<b>3</b>
<b>BAB II LANDASAN TEORI.....</b>	<b>6</b>
2.1. Dasar Teori.....	6
2.2. Penjelasan Singkat Aplikasi Web.....	6
Anatomi permintaan dinamis.....	8
Melakukan pekerjaan lain.....	10
Mengembalikan sesuatu selain HTML.....	11
<b>BAB III ANALISIS PEMECAHAN MASALAH.....</b>	<b>12</b>
3.1. Langkah-Langkah Pemecahan Masalah.....	12
3.2. Proses Pemetaan Masalah menjadi Elemen-Element Algoritma DFS dan BFS.....	12
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	13
3.4. Contoh Ilustrasi Kasus.....	13
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>15</b>
4.1. Implementasi Program Wajib.....	15
4.2. Spesifikasi Teknis Program.....	15
4.3. Penjelasan program.....	27
4.4. Pengetesan program.....	27
4.5. Analisis Hasil Pengujian.....	28
<b>BAB V KESIMPULAN, SARAN, DAN REFLEKSI.....</b>	<b>29</b>
5.1. Kesimpulan.....	29
5.2. Saran.....	29
5.3. Refleksi.....	29
<b>LAMPIRAN.....</b>	<b>30</b>
1. Tautan Repository Github.....	30
2. Tabel Ketercapaian.....	30
<b>DAFTAR PUSTAKA.....</b>	<b>31</b>

[illegible]

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 kedua Strategi Algoritma ini, mahasiswa diminta untuk menyelesaikan permainan Little Alchemy 2 ini dengan menggunakan strategi Depth First Search dan Breadth First Search.

1. Elemen Dasar  
Dalam permainan Little Alchemy 2, terdapat 4 elemen dasar yang tersedia yaitu water, fire, earth, dan air. Empat elemen dasar tersebut nanti dapat digabungkan atau di-*combine* menjadi elemen turunan yang semuanya berjumlah 720 elemen.



**Gambar 2.** Elemen Dasar pada Permainan Little Alchemy 2  
(Sumber: [https://little-alchemy.fandom.com/wiki/Elements\\_\(Little\\_Alchemy\\_2\)](https://little-alchemy.fandom.com/wiki/Elements_(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.

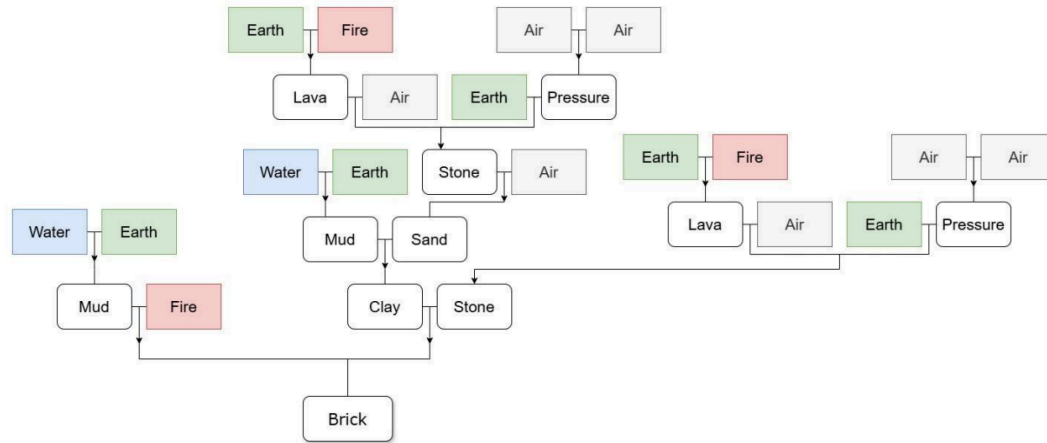
## 3. Mekanisme *Combine*

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 apabila elemen tersebut masih dapat digunakan untuk membuat elemen baru.

Spesifikasi wajib untuk Tugas Besar 2 ini adalah:

1. Membuat aplikasi pencarian resep elemen dalam permainan Little Alchemy 2 dengan menggunakan strategi algoritma **BFS dan DFS**. Aplikasi ini haruslah berbasis **web** dengan *frontend* dibangun menggunakan bahasa **Javascript** dengan *framework* [Next.js](#) atau [React.js](#), dan untuk *backend* menggunakan bahasa **Golang**.
2. Tugas dikerjakan berkelompok dengan anggota **minimal 2 orang** dan **maksimal 3 orang**.
3. Repository *frontend* dan *backend* dapat **digabung** atau **dipisah**.
4. Data elemen beserta resep diperoleh melalui langkah *data scraping* [website Fandom Little Alchemy 2](#).
5. Pada aplikasi, terdapat opsi untuk **memilih algoritma** BFS atau DFS. Terdapat juga *toggle button* untuk memilih dan menemukan **sebuah recipe** atau mencari **banyak resep (multiple recipe)** menuju suatu elemen tertentu. Apabila pengguna ingin mencari banyak resep, maka terdapat cara untuk memasukkan **parameter banyak resep maksimal** yang

ingin dicari. Aplikasi boleh mengeluarkan resep apapun asalkan berbeda dan memenuhi banyak resep yang diinginkan/telah ditentukan oleh pengguna. Mode pencarian *multiple recipe* wajib dioptimasi menggunakan **multithreading**. Aplikasi akan **memvisualisasikan** resep yang ditemukan sebagai sebuah *tree* yang menunjukkan kombinasi elemen yang diperlukan dari elemen dasar. Contoh visual grafis *tree* dapat dilihat sebagai berikut:



**Gambar 3.** Contoh Visualisasi Tree Resep Elemen  
(Sumber: Spesifikasi Tugas Besar 2 Stima 2024/2025)

Selain menunjukkan visualisasi grafis *tree*, aplikasi juga menampilkan **waktu pencarian** serta **banyak node** yang telah dikunjungi.

## BAB II LANDASAN TEORI

### 2.1. Dasar Teori

Algoritma traversal graf adalah proses mengunjungi simpul-simpul dalam suatu graf dengan cara yang sistematis. Algoritma ini dapat diterapkan pada pencarian solusi berbasis graf tanpa informasi (*uninformed/blind search*) maupun dengan informasi (*informed search*) yang berbasis heuristik. Contoh dari algoritma pencarian tanpa informasi adalah *Breadth First Search* (BFS) dan *Depth First Search* (DFS).

BFS adalah metode traversal yang menjelajahi graf secara melebar, yaitu dengan mengunjungi semua simpul pada suatu level sebelum berpindah ke level berikutnya. Langkah-langkah BFS mencakup mengunjungi simpul awal  $v$ , kemudian mengunjungi semua simpul yang bertetangga langsung dengan  $v$ , dilanjutkan dengan simpul-simpul yang bertetangga dengan simpul-simpul yang baru saja dikunjungi, dan seterusnya hingga seluruh simpul telah dikunjungi. Struktur data yang digunakan dalam BFS dapat berupa matriks ketetanggaan  $A = [a_{ij}]$  berukuran  $n \times n$ , antrian  $q$  untuk menyimpan simpul yang telah dikunjungi, dan tabel Boolean yang menandai simpul yang telah dikunjungi.

Sementara itu, DFS merupakan metode traversal yang menjelajahi graf secara mendalam, dengan menelusuri lintasan sejauh mungkin sebelum kembali (*backtrack*) ke simpul sebelumnya jika tidak ada simpul bertetangga yang belum dikunjungi. Proses DFS dimulai dari mengunjungi simpul  $v$ , lalu dilanjutkan dengan mengunjungi simpul  $w$  yang bertetangga dengan simpul  $v$ , dan mengulangi proses DFS mulai dari simpul  $w$ . Ketika mencapai simpul  $u$  sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul  $w$  yang belum dikunjungi. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi. DFS umumnya menggunakan struktur data *stack*.

### 2.2. Penjelasan Singkat Aplikasi Web

Aplikasi *web* adalah sebuah sistem perangkat lunak yang dijalankan melalui peramban atau *browser*. Aplikasi *web* menggunakan arsitektur *client-server* dengan situs web dinamis. Pada arsitektur ini, terdapat dua komponen, yaitu:

#### 1. Client

Merupakan *browser* pengguna yang merupakan *frontend* di mana tugasnya antara lain:

- Mengirim permintaan (*request*) ke server untuk meminta suatu data
- Menampilkan *interface*/antarmuka (UI/UX) dan interaksi dengan pengguna melalui bahasa pemrograman HTML, CSS, dan JavaScript.
- Menangani input dari pengguna

#### 2. Server

Program yang menerima permintaan dari *client* dan mengirimkan respons. Server ada

pada *backend*. Tugasnya antara lain:

- Memproses permintaan dari klien
- Mengakses dan mengelola *database*
- Mengirimkan respon ke klien

Interaksi antara klien dengan server menggunakan HyperText Transfer Protocol (HTTP). Saat pengguna mengklik tautan pada halaman *web*, mengirimkan formulir, atau menjalankan pencarian, browser akan mengirimkan beberapa permintaan HTTP ke server. Permintaan ini meliputi:

- URL yang mengidentifikasi server dan sumber daya target (misal: berkas HTML, data CSS, gambar, atau file JavaScript)
- Metode yang menentukan tindakan yang diperlukan (misal: mendapatkan file, menyimpan file, memperbaharui data). Metode kerja tersebut adalah sebagai berikut:
- GET: Mendapatkan sumber daya tertentu
- POST: Membuat sumber daya baru
- HEAD: Mendapatkan informasi metadata tentang sumber daya tertentu tanpa mendapatkan isi seperti GET yang diinginkan.
- PUT: Memperbaharui sumber daya yang ada atau membuat baru jika belum ada
- DELETE: Menghapus sumber daya yang ditentukan
- TRACE, OPTIONS, CONNECT, PATCH: Metode kerja yang kurang umum
- Informasi tambahan dapat dikodekan dengan permintaan.

Server web akan menunggu pesan permintaan dari klien, lalu memprosesnya setelah mendapatkan pesan yang sesuai dan mengirimkannya kembali ke web dengan pesan respons HTTP. Respons tersebut juga berisi kode status yang menunjukkan apakah permintaan berhasil atau tidak ('200 OK' apabila berhasil, '404 Not Found' apabila sumber daya tidak ditemukan, '403 Forbidden' apabila pengguna tidak diizinkan, dan sebagainya).

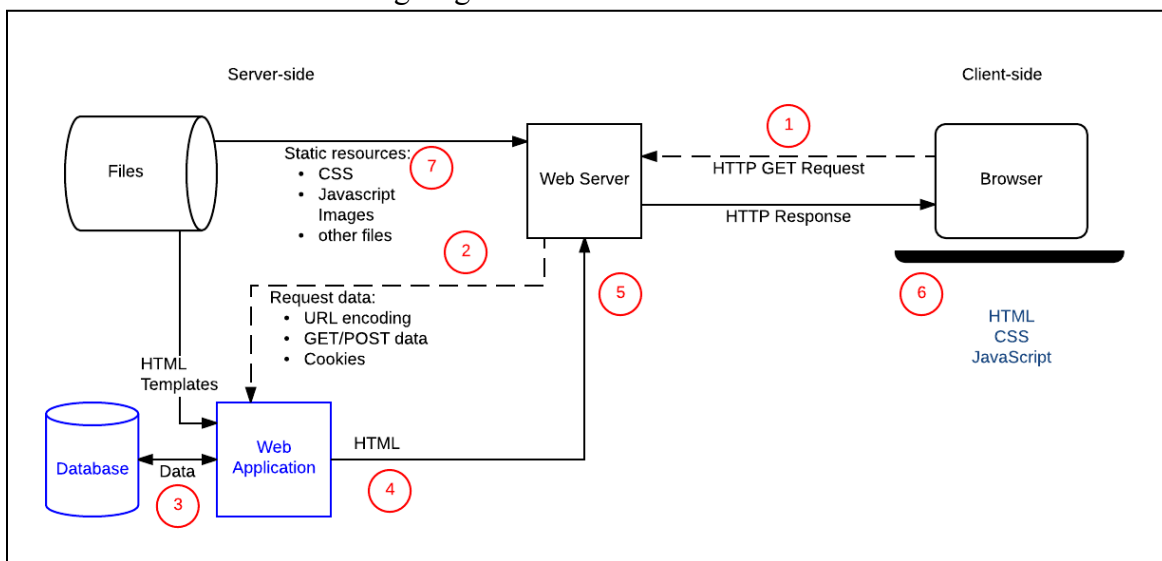
Terdapat dua jenis situs/*site*, yaitu statis dan dinamis. Dalam tugas ini, jenis situs yang digunakan adalah situs dinamis: situs yang dapat menghasilkan dan mengembalikan konten berdasarkan URL dan data permintaan tertentu. Pada contoh situs produk, server akan menyimpan "data" produk dalam basis data, bukan berkas HTML. Saat menerima permintaan HTTP 'GET' untuk suatu produk, server menentukan ID Produk, mengambil data dari basis data, lalu membuat halaman HTML untuk respons dengan memasukkan data tersebut ke dalam *template* HTML. Penggunaan basis data ini memberikan keefisienan dalam penyimpanan informasi produk dengan cara yang mudah diperluas, dimodifikasi, dan dicari. *Template* HTML memudahkan dalam mengubah struktur HTML sehingga program hanya perlu melakukannya di satu lokasi, bukan pada banyak halaman statis.

Aplikasi Web adalah bagian situs yang membuatnya dinamis, sebagai sebuah cara untuk menyebut kode sisi server yang memproses permintaan HTTP dan mengembalikan respons HTTP. Selain itu, terdapat juga Basis Data yang berisikan informasi-informasi sumber daya, dan *Template* HTML. Berikut adalah langkah-langkah urutan operasinya:

1. Pengguna mengirimkan formulir permintaan untuk melihat data
2. Browser membuat GET permintaan HTTP ke Web Server menggunakan URL

dasar untuk sumber daya dan parameter URL. Permintaan GET digunakan karena hanya ingin mengambil data.

3. Server Web mendeteksi adanya permintaan tersebut dan meneruskannya ke Aplikasi Web untuk diproses
4. Aplikasi Web mengidentifikasi maksud dari permintaan data tersebut dan mencari data tersebut pada *database* untuk memenuhi kebutuhan.
5. Aplikasi Web secara dinamis membuat halaman HTML yang dihasilkan ke *browser* dengan meletakkan data dari *database* (Basis Data) ke dalam *placeholder* di dalam *template* HTML.
6. Aplikasi Web mengembalikan HTML yang dihasilkan ke *browser* melalui *Server Web* beserta kode status.
7. *Browser* kemudian memproses HTML yang dikembalikan dan mengirimkan permintaan terpisah untuk mendapatkan file CSS atau JavaScript lain yang dirujuk (Langkah 7 pada Gambar)
8. Server Web memuat berkas status dari sistem berkas dan mengembalikannya ke browser secara langsung.



**Gambar 4.** Diagram Urutan Operasi

(Sumber: [https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Extensions/Server-side/First\\_steps/Client-Server\\_overview](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview))

Dari urutan operasi tersebut, tugas Aplikasi Web adalah menerima permintaan HTTP dan mengembalikan respons HTTP. Interaksi dengan *database* untuk mendapatkan atau memperbarui informasi adalah tugas yang umum. Selain itu, kode tersebut dapat melakukan hal lain pada saat yang sama, atau dibuat tidak berinteraksi dengan basis data sama sekali. Contoh dari tugas tambahan Aplikasi Web adalah mengirimkan *email* kepada pengguna untuk mengkonfirmasi pendaftaran mereka di situs tersebut. Selain itu, Kode situs Web Server tidak harus mengembalikan file HTML sebagai respons. Kode tersebut secara dinamis dalam membuat dan mengembalikan jenis file lain (txt, PDF, CSV, dan lainnya) atau data (JSON, XML, dan lainnya). Hal ini relevan untuk situs web yang bekerja dengan mengambil konten dari server menggunakan JavaScript dan memperbarui halaman secara dinamis, daripada selalu memuat halaman baru saat konten baru akan ditampilkan..



## **BAB III ANALISIS PEMECAHAN MASALAH**

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

Untuk menyelesaikan permasalahan pencarian resep pada permainan *Little Alchemy 2*, langkah pertama yang perlu dilakukan adalah menganalisis masalah dan tujuan pembuatan *web*. *Web* ini bertujuan untuk menemukan resep atau kombinasi elemen yang dibutuhkan untuk menciptakan suatu elemen target, dengan menggunakan dua strategi pencarian, yaitu *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS). Pengguna dapat memilih untuk mencari satu resep terpendek atau beberapa resep berbeda menuju elemen target. Oleh karena itu, perlu dilakukan data pengumpulan (scraping) terhadap semua elemen dan resep yang tersedia di situs Fandom *Little Alchemy 2*. Data hasil scraping harus diproses dan disimpan dalam struktur yang efisien, misalnya dalam bentuk array resep yang berisi data resep dengan output, input (pasangan elemen), dan tingkatnya.

Selanjutnya, backend aplikasi dibangun menggunakan bahasa Golang. Di bagian ini, dirancang dua algoritma pencarian utama, BFS dan DFS. Masing-masing dapat digunakan untuk menemukan banyak resep maupun resep terpendek. Pada sisi frontend, yang dikembangkan menggunakan React.js, pengguna diberikan antarmuka untuk memilih algoritma pencarian, mode pencarian (single atau multiple recipe), memasukkan elemen target, serta menentukan jumlah maksimum resep yang ingin ditemukan (jika multiple recipe). Hasil pencarian ditampilkan dalam bentuk visualisasi pohon kombinasi menggunakan library vis-network, yang merepresentasikan hubungan antar elemen dalam proses pembentukan elemen target. Visualisasi ini dilengkapi dengan statistik berupa waktu pencarian dan jumlah node yang dikunjungi, untuk memberi pemahaman yang lebih dalam kepada pengguna terkait efisiensi algoritma. Setelah backend dan frontend selesai dikembangkan, dilakukan proses integrasi melalui REST API. Data hasil pencarian dari backend dikirim dalam format JSON.

Terakhir, aplikasi diuji untuk memastikan bahwa semua mode pencarian berjalan dengan benar dan efisien. Pengujian dilakukan dengan mencoba berbagai target elemen serta mengevaluasi apakah hasil resep sesuai dengan kombinasi sebenarnya. Performa aplikasi juga diuji agar dapat menangani pencarian dalam jumlah banyak dengan cepat dan tanpa mengalami kegagalan.

### **3.2. Proses Pemetaan Masalah menjadi Elemen-Element Algoritma DFS dan BFS**

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

Algoritma pencarian resep sebuah element dengan metode DFS menggunakan representasi graf dinamis dalam prosesnya. Hal ini berarti pada setiap simpul yang terbentuk akan dievaluasi apakah sudah memenuhi goal state yakni berupa elemen dasar atau belum. Apabila semua cabang sudah dicoba namun tidak ada yang memenuhi batasan, maka simpul tersebut dianggap mati dan akan dilakukan backtracking ke cabang lainnya pada simpul sebelumnya. Selama proses pencarian, akan terbentuk pohon ruang status yang menandakan bagaimana proses

traversal dilakukan. Adapun pemetaan masalah menjadi elemen-elemen algoritma DFS adalah sebagai berikut;

1. Pohon ruang status: representasi proses traversal yang dilakukan selama pencarian
2. Simpul: representasi resep yang memenuhi batasan yakni semua bahan memiliki tier dibawah elemen target
  - Akar: elemen target yang ingin dicari resepnya
  - Daun: elemen dasar yang otomatis tersedia di awal permainan
3. Cabang: representasi resep alternatif yang dapat digunakan untuk membuat elemen target
4. Ruang solusi: himpunan resep yang dapat digunakan sebagai langkah untuk membuat elemen target
5. Ruang status: himpunan simpul/resep yang pernah dikunjungi dan memenuhi batasan selama proses pencarian

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

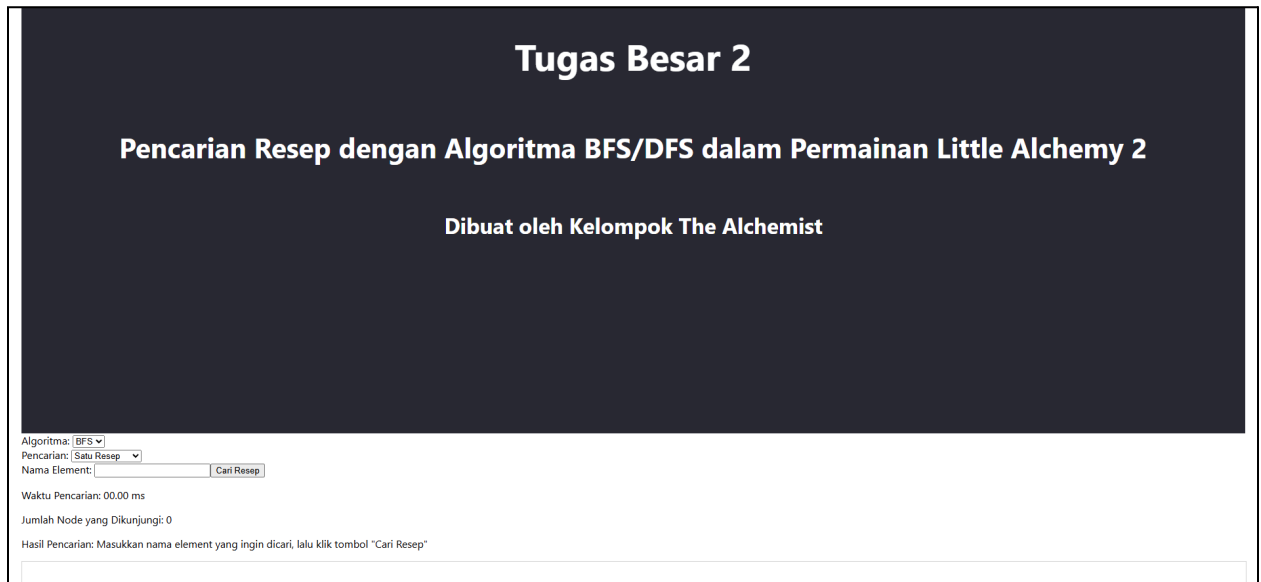
Algoritma pencarian resep sebuah elemen pada permainan *Little Alchemy 2* dengan metode BFS menggunakan representasi graf dinamis dalam prosesnya. BFS melakukan pencarian secara melebar dengan menelusuri semua simpul pada level yang sama terlebih dahulu sebelum lanjut ke level yang lebih dalam. Artinya, semua kemungkinan resep akan dicoba dari level bawah terlebih dahulu. Hal ini memastikan bahwa solusi pertama yang ditemukan adalah yang memiliki tier terendah. BFS juga cocok untuk menemukan semua jalur pembuatan elemen karena setiap kombinasi bahan akan dievaluasi dan dikombinasikan secara eksplisit.

Pemetaan masalah menjadi elemen-elemen algoritma BFS adalah sebagai berikut:

1. Graf: Graf dibentuk dari elemen-elemen yang berhubungan melalui resep-resep yang ada di recipeMap. Tier Map (tierMap) digunakan untuk mengatur tingkat (tier) dari setiap elemen, yang menentukan apakah dua elemen dapat digabungkan atau tidak (berdasarkan tingkatannya).
2. Simpul: Simpul merepresentasikan resep yang valid berdasarkan tier dan hasil kombinasi bahan-bahan penyusun, termasuk simpul hasil intermediate dari bahan ke elemen target.
  - Akar: Elemen target
  - Daun: Elemen dasar
3. Queue: Struktur data utama yang menyimpan elemen yang akan dijelajahi, yaitu QueueItem (berisi elemen dan jalur pembuatan).
4. Cabang: Alternatif kombinasi dua elemen yang membentuk suatu elemen yang akan dikunjungi secara melebar sesuai antrian.
5. Ruang solusi: Himpunan jalur resep lengkap yang dapat digunakan untuk membentuk elemen target.
6. Ruang status: Himpunan simpul (elemen atau kombinasi bahan) yang pernah dikunjungi selama pencarian (memoization).

7. Proses traversal: Proses traversal dilakukan dengan mengeluarkan elemen dari antrian dan mengeksplorasi semua kemungkinan resep yang bisa digunakan untuk membentuk elemen baru. Untuk setiap elemen yang sedang diproses, algoritma mencari resep-resep yang menggabungkan dua bahan, yang kemudian diulang dalam traversal BFS.

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



**Gambar 5.** ScreenShot Aplikasi *Web*  
(Sumber: Dokumentasi Pribadi)

#### 3.3.1. Fitur Fungsional

Aplikasi Web ini menyediakan beberapa fitur-fitur utama supaya pengguna dapat mencari resep elemen dalam permainan video Little Alchemy 2. Fitur-fitur tersebut adalah:

1. Opsi Mode Pencarian Mode Resep Tunggal dan Pencarian Semua Resep  
Pengguna dapat memilih mode pencarian apa yang ingin dilakukan untuk mendapatkan resep elemen: satu resep atau semua resep yang tersedia.
2. Masukan Jumlah Resep yang Ingin Dicari  
Pengguna bisa memasukkan jumlah resep yang ingin dicari apabila memilih Opsi Mode Pencarian Semua Resep.
3. Opsi Algoritma  
Pengguna dapat memilih jenis algoritma apa yang ingin digunakan, yakni BFS (*Breadth-First-Search*) atau DFS (*Depth-First-Search*).
4. Masukan Nama Elemen yang Ingin Dicari  
Pengguna dapat memasukkan nama elemen yang ingin dicari pada kolom pencarian. Apabila tidak diisi, aplikasi akan memunculkan pengingat untuk mengisi elemen dengan benar.
5. Tombol “Cari Resep”  
Setelah pengguna sudah melakukan pemilihan mode pencarian, algoritma, dan

memasukkan nama elemen yang dicari, pengguna dapat menekan tombol “Cari Resep” untuk mendapatkan data.

6. Statistik Pencarian

Setelah pengguna meminta data, pengguna akan mendapatkan statistik pencarian berupa waktu pencarian yang dilakukan (dalam milisekon) dan jumlah node yang dikunjungi selama proses pencarian menggunakan algoritma.

7. Visualisasi Graf

Pengguna juga mendapatkan visualisasi langkah-langkah resep dengan elemen-elemen yang ada ditampilkan dalam bentuk graf pohon menggunakan *library vis-network*. Dari aplikasi yang dibuat, visualisasi graf berhasil untuk menunjukkan resep elemen secara berurutan, meski untuk beberapa elemen terbagi menjadi sub-graf *tree*.

### 3.3.2. Arsitektur Aplikasi Web

Sesuai pada landasan teori, aplikasi ini dirancang dengan menggunakan arsitektur *client-server* atau klien-server yang terdiri dari dua komponen utama, yaitu:

1. Frontend (Sisi Klien)

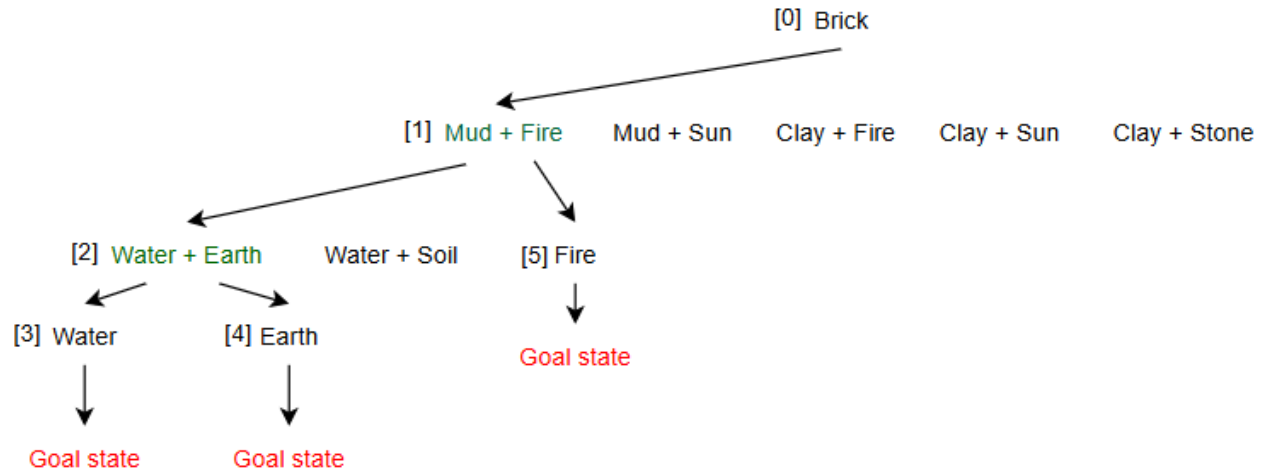
*Frontend* diprogram menggunakan [React.js](https://reactjs.org/) sebagai *user interface*. *Library vis-network* digunakan untuk visualisasi graf resep. Pada UI, terdapat *header* untuk judul proyek dan nama kelompok, *dropdown* untuk memilih algoritma (BFS/DFS), *dropdown* untuk mode pencarian (Satu Resep/Semua Resep), input angka untuk jumlah resep, input teks untuk nama elemen, tombol “Cari Resep”, dan area untuk menampilkan visualisasi grafis serta statistik pencarian.

2. Backend (Sisi Server)

*Backend* diprogram menggunakan Go (Golang) untuk memproses pencarian. *Backend* menyediakan dua endpoint API, yaitu */bfs*, dan */dfs*, yang menerima parameter *element*, *multiple*, dan *n*. Backend mengembalikan respon data dalam format JSON, termasuk langkah-langkah resep (*Steps*), waktu pencarian, dan jumlah node yang dikunjungi.

## 3.4. Contoh Ilustrasi Kasus

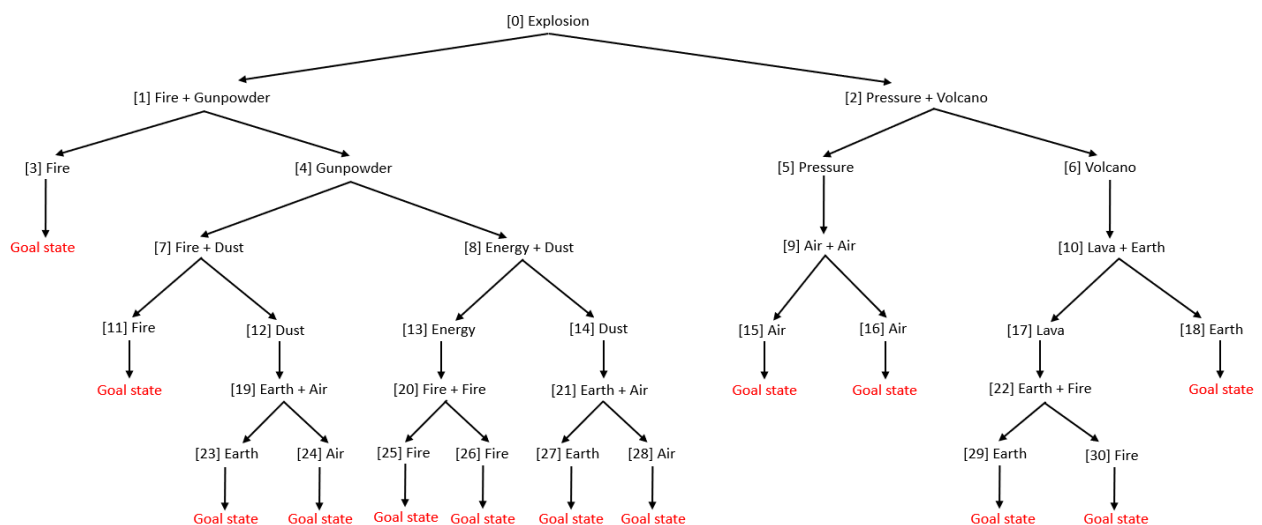
### Depth First Search (DFS)



**Gambar 6.** Ilustrasi Pohon Ruang Status DFS  
(Sumber: Dokumentasi Pribadi)

Misalkan ingin mencari resep dari brick, maka algoritma DFS akan melakukan traversal dengan urutan sesuai nomor. Brick sebagai elemen yang dicari dijadikan sebagai akar dan elemen dasar dijadikan sebagai daun. Juga dapat dibuat percabangan dengan resep yang berbeda-beda, apabila suatu cabang tidak memenuhi batasan. Sehingga ruang solusi yang akan dihasilkan akan berbentuk [[Water, Earth, Mud], [Mud, Fire, Brick]]

### Breadth First Search (BFS)



**Gambar 7.** Ilustrasi Pohon Ruang Status BFS  
(Sumber: Dokumentasi Pribadi)

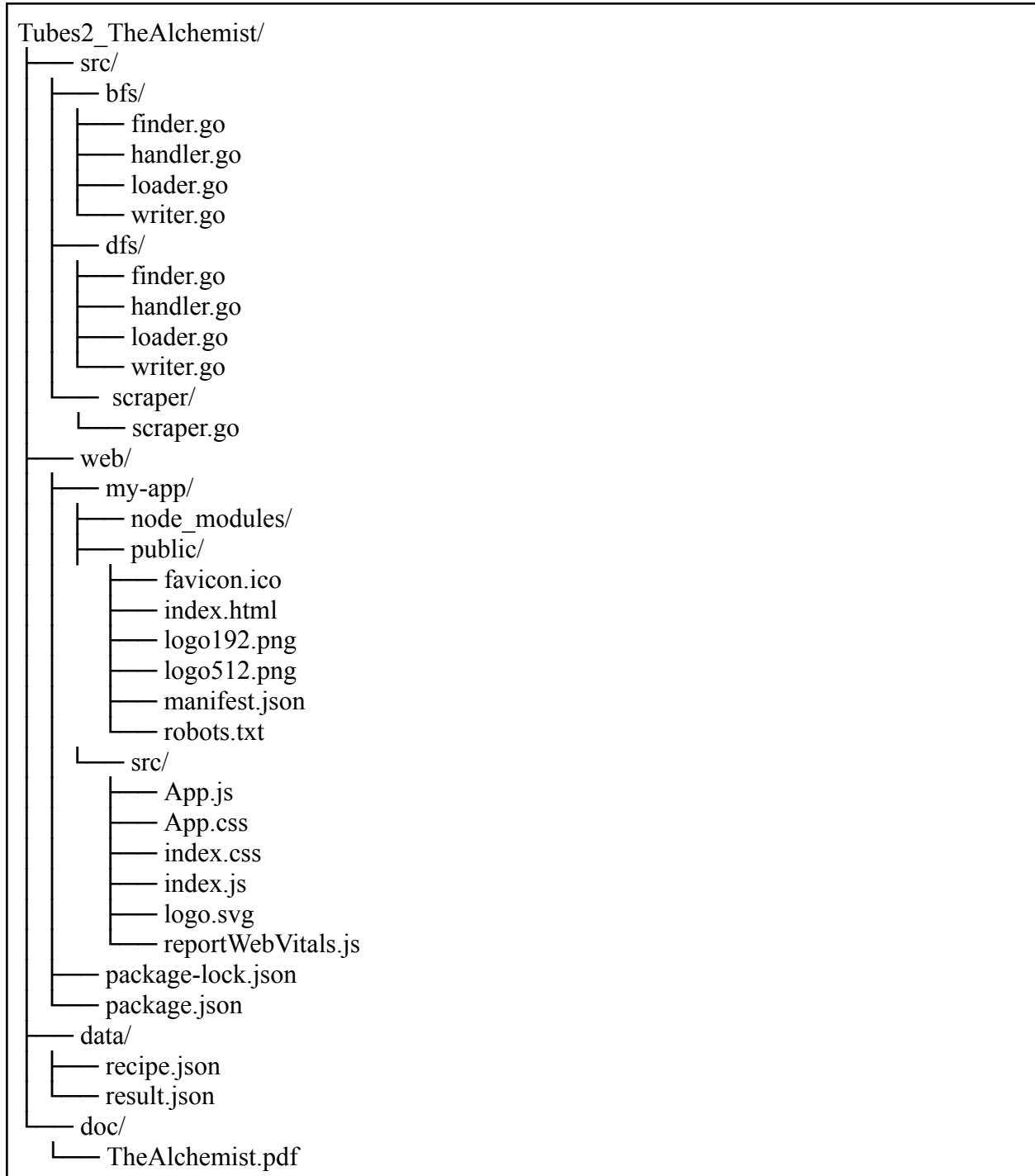
Pada proses pencarian elemen "Explosion" dengan algoritma Breadth-First Search (BFS), algoritma akan melakukan pencarian secara melebar dari elemen target ke elemen-elemen yang lebih dasar. Explosion sebagai elemen yang dicari dijadikan sebagai akar, dan elemen dasar

seperti *Air*, *Fire*, *Earth*, dan sebagainya dijadikan sebagai daun (goal state). BFS akan menjelajahi semua kemungkinan kombinasi resep untuk membentuk Explosion, dimulai dari tingkat yang paling dekat dengan target dan memperluas pencarian ke elemen-elemen penyusunnya secara bertahap. Jika jumlah jalur maksimal telah tercapai, maka ruang solusi yang dihasilkan akan berbentuk. Masing-masing jalur menunjukkan langkah-langkah eksplisit dari elemen dasar hingga membentuk elemen target secara lengkap.

## BAB IV IMPLEMENTASI DAN PENGUJIAN

### 4.1. Implementasi Program Wajib

Berikut adalah struktur program:



## 4.2 Spesifikasi Teknis Program

### Struktur data

```
type Step [3]string
type RecipePath []Step
type QueueItem struct {
    element string
    path    [] [3]string
}
type Recipe struct {
    Output string    `json:"Output"`
    Inputs [] [2]string `json:"Inputs"`
    Tier   int        `json:"Tier"`
}

recipeMap : map[string] [] [2]string
tierMap : map[string] int
visitedMap : map[string] bool
memo : map[string] [] RecipePath
baseElements : map[string] bool
queue : [] QueueItem {element: elementName, path: [] [3]string {}}
parent : map[string] [2]string
solutionPath : map[string] [] [3]string
solutions := map[string] [] RecipePath
```

### Kode fungsi scraping

```
func Scrape() {
    url :=
    "https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"

    resp, err := http.Get(url)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatal(err)
    }
}
```



```

}

var results []Recipe
var tierCtr int = -1

doc.Find("table.list-table").Each(func(i int, table *goquery.Selection) {
    tierCtr++
    table.Find("tr").Each(func(j int, s *goquery.Selection) {
        tds := s.Find("td")
        if tds.Length() >= 2 {
            aTags := tds.Eq(0).Find("a")
            if aTags.Length() < 2 {
                return
            }
            elementHasil := strings.TrimSpace(aTags.Eq(1).Text())

            if elementHasil == "Time" {
                tierCtr--
            }

            var elementBahan [][]string
            tds.Eq(1).Find("li").Each(func(j int, li *goquery.Selection) {
                aFromLi := li.Find("a")
                if aFromLi.Length() >= 4 {
                    combo := []string{
                        strings.TrimSpace(aFromLi.Eq(1).Text()),
                        strings.TrimSpace(aFromLi.Eq(3).Text()),
                    }
                    elementBahan = append(elementBahan, combo)
                }
            })

            if elementHasil != "" && len(elementBahan) >= 0 {
                results = append(results, Recipe{
                    Output: elementHasil,
                    Inputs: elementBahan,
                    Tier: tierCtr,
                })
            }
        }
    })
})

```

```

    })

    // Simpan ke file JSON
    file, err := os.Create("./data/recipe.json")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    encoder := json.NewEncoder(file)
    encoder.SetIndent("", " ")
    encoder.SetEscapeHTML(false)

    err = encoder.Encode(results)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Scraping selesai! Data disimpan ke ./data/recipe.json")
}

```

### Kode Breadth First Search (BFS)

```

func bfsSingle(elementName string, recipeMap map[string][][2]string, tierMap
map[string]int, nodeCountElement *int, nodeCountRecipe *int) (bool,
[][3]string) {
    base := map[string]bool{"Air": true, "Water": true, "Fire": true, "Earth":
true, "Time": true}
    if base[elementName] {
        return true, [][3]string{}
    }

    type QueueItem struct {
        element string
        path    [][3]string
    }

    queue := []QueueItem{{element: elementName, path: [][3]string{}}}
    visited := make(map[string]bool)
    parent := make(map[string][2]string)
    solutionPath := make(map[string][][3]string)

```

```

for len(queue) > 0 {
    current := queue[0]
    queue = queue[1:]
    (*nodeCountElement)++

    if visited[current.element] {
        continue
    }
    visited[current.element] = true

    if base[current.element] {
        solutionPath[current.element] = [][3]string{}
        continue
    }

    elementTier := tierMap[current.element]
    foundSolution := false

    for _, recipe := range recipeMap[current.element] {
        nameA := recipe[0]
        nameB := recipe[1]
        tierA := tierMap[nameA]
        tierB := tierMap[nameB]

        if tierA >= elementTier || tierB >= elementTier {
            continue
        }

        (*nodeCountRecipe)++

        if _, existsA := solutionPath[nameA]; !existsA {
            queue = append(queue, QueueItem{element: nameA, path:
append(current.path, [3]string{nameA, nameB, current.element})})
        }
        if _, existsB := solutionPath[nameB]; !existsB {
            queue = append(queue, QueueItem{element: nameB, path:
append(current.path, [3]string{nameA, nameB, current.element})})
        }

        if solutionA, okA := solutionPath[nameA]; okA {

```

```

        if solutionB, okB := solutionPath[nameB]; okB {
            combined := append(append(solutionA, solutionB...),
[3]string{nameA, nameB, current.element})
            solutionPath[current.element] = combined
            foundSolution = true
            if current.element == elementName {
                return true, combined
            }
            break
        }
    }

    if !foundSolution {
        parent[current.element] = [2]string{}
    }
}

if path, ok := solutionPath[elementName]; ok {
    return true, path
}

return false, nil
}

func bfsMultiple(
    elementName string,
    recipeMap map[string][][2]string,
    tierMap map[string]int,
    memo map[string][]RecipePath,
    nodeCountElement *int,
    nodeCountRecipe *int,
    maxPaths int,
) []RecipePath {
    (*nodeCountElement)++

    if baseElements[elementName] {
        return []RecipePath{{}}
    }
}

```

```

memoLock.RLock()
if paths, found := memo[elementName]; found {
    memoLock.RUnlock()
    return paths
}
memoLock.RUnlock()

type QueueItem struct {
    element string
    paths    []RecipePath
}

queue := []QueueItem{{element: elementName, paths: []RecipePath{}}}
solutions := make(map[string][]RecipePath)
elementTier := tierMap[elementName]

for len(queue) > 0 && len(solutions[elementName]) < maxPaths {
    current := queue[0]
    queue = queue[1:]
    (*nodeCountElement)++

    if baseElements[current.element] {
        solutions[current.element] = []RecipePath{{}}
        continue
    }

    for _, recipe := range recipeMap[current.element] {
        nameA, nameB := recipe[0], recipe[1]
        tierA, tierB := tierMap[nameA], tierMap[nameB]

        if tierA >= elementTier || tierB >= elementTier {
            continue
        }

        (*nodeCountRecipe)++

        // Check if we have solutions for the components
        pathsA, hasA := solutions[nameA]
        if !hasA {
            pathsA = bfsMultiple(nameA, recipeMap, tierMap, memo,
nodeCountElement, nodeCountRecipe, maxPaths)

```

```

        solutions[nameA] = pathsA
    }

    pathsB, hasB := solutions[nameB]
    if !hasB {
        pathsB = bfsMultiple(nameB, recipeMap, tierMap, memo,
nodeCountElement, nodeCountRecipe, maxPaths)
        solutions[nameB] = pathsB
    }

    var newPaths []RecipePath
    for _, pathA := range pathsA {
        for _, pathB := range pathsB {
            combined := make(RecipePath, 0, len(pathA)+len(pathB)+1)
            combined = append(combined, pathA...)
            combined = append(combined, pathB...)
            combined = append(combined, Step{nameA, nameB,
current.element})
            newPaths = append(newPaths, combined)

            if current.element == elementName && len(newPaths) >=
maxPaths {
                break
            }
        }
        if current.element == elementName && len(newPaths) >= maxPaths
{
            break
        }
    }

    if len(newPaths) > 0 {
        if _, exists := solutions[current.element]; !exists {
            solutions[current.element] = newPaths
        } else {
            solutions[current.element] =
append(solutions[current.element], newPaths...)
        }

        if current.element == elementName &&
len(solutions[current.element]) >= maxPaths {

```

```

        solutions[current.element] =
solutions[current.element][:maxPaths]
        break
    }
}
}
}

memoLock.Lock()
memo[elementName] = solutions[elementName]
memoLock.Unlock()

return solutions[elementName]
}

```

### Kode Depth First Search (DFS)

```

func dfsSingle(elementName string, recipeMap map[string][][2]string, tierMap
map[string]int, nodeCountElement *int, nodeCountRecipe *int, visitedMap
map[string]bool) (bool, [][][3]string) {
    (*nodeCountElement)++
    base := map[string]bool{"Air": true, "Water": true, "Fire": true, "Earth":
true, "Time": true}
    if base[elementName] {
        return true, [][][3]string{}
    }

    if visitedMap[elementName] {
        return true, [][][3]string{}
    }

    elementTier := tierMap[elementName]

    for _, recipe := range recipeMap[elementName] {
        nameA := recipe[0]
        nameB := recipe[1]
        tierA := tierMap[nameA]
        tierB := tierMap[nameB]

        if tierA >= elementTier || tierB >= elementTier {
            continue
        }
    }
}

```

```

    }

    okA, stepsA := dfsSingle(nameA, recipeMap, tierMap, nodeCountElement,
nodeCountRecipe, visitedMap)
    if !okA {
        continue
    }

    okB, stepsB := dfsSingle(nameB, recipeMap, tierMap, nodeCountElement,
nodeCountRecipe, visitedMap)
    if !okB {
        continue
    }

    combined := append(append(stepsA, stepsB...), [3]string{nameA, nameB,
elementName})
    (*nodeCountRecipe)++
    visitedMap[elementName] = true
    return true, combined
}

return false, nil
}

func dfsMultiple(
    elementName string,
    recipeMap map[string][][2]string,
    tierMap map[string]int,
    memo map[string][]RecipePath,
    nodeCountElement *int,
    nodeCountRecipe *int,
    maxPaths int,
) []RecipePath {
    (*nodeCountElement)++

    if baseElements[elementName] {
        return []RecipePath{{}} // Base element: satu path kosong
    }

```



```

// Cek di memo dengan read lock
memoLock.RLock()
if paths, found := memo[elementName]; found {
    memoLock.RUnlock()
    return paths
}
memoLock.RUnlock()

elementTier := tierMap[elementName]
var allPaths []RecipePath

for _, recipe := range recipeMap[elementName] {
    nameA, nameB := recipe[0], recipe[1]
    tierA, tierB := tierMap[nameA], tierMap[nameB]

    if tierA >= elementTier || tierB >= elementTier {
        continue
    }

    // Paralelkan pencarian nameA dan nameB
    chA := make(chan []RecipePath, 1)
    chB := make(chan []RecipePath, 1)

    go func() {
        chA <- getPathsConcurrent(nameA, recipeMap, tierMap, memo,
nodeCountElement, nodeCountRecipe, maxPaths)
    }()

    go func() {
        chB <- getPathsConcurrent(nameB, recipeMap, tierMap, memo,
nodeCountElement, nodeCountRecipe, maxPaths)
    }()

    pathsA := <-chA
    pathsB := <-chB

    (*nodeCountRecipe)++

    for _, pathA := range pathsA {
        for _, pathB := range pathsB {
            combined := append([]Step{}, pathA...)

```

```

        combined = append(combined, pathB...)
        combined = append(combined, Step{nameA, nameB, elementName})
        allPaths = append(allPaths, combined)

        if len(allPaths) >= maxPaths {
            // Simpan ke memo dengan write lock
            memoLock.Lock()
            memo[elementName] = allPaths
            memoLock.Unlock()
            return allPaths
        }
    }
}

// Simpan hasil akhir ke memo
memoLock.Lock()
memo[elementName] = allPaths
memoLock.Unlock()
return allPaths
}

func getPathsConcurrent(
    name string,
    recipeMap map[string][][2]string,
    tierMap map[string]int,
    memo map[string][]RecipePath,
    nodeCountElement *int,
    nodeCountRecipe *int,
    maxPaths int,
) []RecipePath {
    memoLock.RLock()
    if paths, found := memo[name]; found {
        memoLock.RUnlock()
        return paths
    }
    memoLock.RUnlock()

    result := dfsMultiple(name, recipeMap, tierMap, memo, nodeCountElement,

```

```

nodeCountRecipe, maxPaths)

    memoLock.Lock()
    memo[name] = result
    memoLock.Unlock()

    return result
}

```

Kode main.go untuk menjalankan server

```

package main

import (
    "encoding/json"
    "littlealchemy/src/bfs"
    "littlealchemy/src/dfs"
    "log"
    "net/http"
)

type Message struct {
    Message string `json:"message"`
}

// untuk cek hubungan ke Golang dari React

func withCORS(w http.ResponseWriter) {
    w.Header().Set("Access-Control-Allow-Origin", "http://localhost:3000")
    w.Header().Set("Access-Control-Allow-Methods", "GET, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
}

func handler(w http.ResponseWriter, r *http.Request) {
    if r.Method == http.MethodOptions {
        withCORS(w)
        w.WriteHeader(http.StatusOK)
        return
    }
}

```

```

    withCORS(w)
    w.Header().Set("Content-Type", "application/json")
    message := Message{Message: "Hello World! From Golang!"}
    json.NewEncoder(w).Encode(message)
}

func dfsWithCORS(w http.ResponseWriter, r *http.Request) {
    if r.Method == http.MethodOptions {
        withCORS(w)
        w.WriteHeader(http.StatusOK)
        return
    }
    withCORS(w)
    dfs.DFSHandler(w, r)
}

func bfsWithCORS(w http.ResponseWriter, r *http.Request) {
    if r.Method == http.MethodOptions {
        withCORS(w)
        w.WriteHeader(http.StatusOK)
        return
    }
    withCORS(w)
    bfs.BFSHandler(w, r)
}

func main() {

    router := http.NewServeMux()

    // handler untuk cek hubungan ke Golang dari React
    router.HandleFunc("/ping", handler)

    // Menyediakan handler untuk route /dfs
    router.HandleFunc("/dfs", dfsWithCORS)

    // Menyediakan handler untuk route /bfs
    router.HandleFunc("/bfs", bfsWithCORS)

    // Jalankan server di port 8080
    log.Println("Server berjalan di http://localhost:8080")
}

```

```
log.Fatal(http.ListenAndServe(":8080", router))
}
```

## Kode Aplikasi Web dan *User Interface*

```
import { useState } from 'react';
import { Network } from 'vis-network/standalone';
import 'vis-network/styles/vis-network.css';
import './App.css';

// Program Aplikasi berbasis Web
function App() {
  const [algorithm, setAlgorithm] = useState('bfs');
  const [search, setSearch] = useState('satu');
  const [element, setElement] = useState(' ');
  const [searchTime, setSearchTime] = useState('00.00');
  const [nodesVisited, setNodesVisited] = useState('0');
  const [recipeResult, setRecipeResult] = useState('Masukkan nama element yang ingin dicari, lalu klik tombol "Cari Resep"');
  const [maxRecipes, setMaxRecipes] = useState(2);

  // Fetch data API
  const findRecipes = async () => {
    if (!element.trim()) {
      alert('Tolong masukkan nama element yang benar!');
      return;
    }

    // const startTime = performance.now();

    try {
      const query = new URLSearchParams({
        element : element.trim(),
        multiple: search === 'semua' ? true : false,
        n: search === 'semua' ? maxRecipes : 1
      }).toString();

      // Memilih algoritma
      const endpoint = algorithm === 'bfs' ? '/bfs' : '/dfs';
```

```

        const responseData = await
fetch(`http://localhost:8080${endpoint}?${query}`, {
    method: 'GET',
    headers: {
        'Content-Type': 'application/json',
    },
});

if (!responseData.ok) {
    const errorText = await responseData.text();
    throw new Error(`Gagal memuat data dari backend:
${responseData.status} ${responseData.statusText} - ${errorText}`);
}

const responseDataJson = await responseData.json();
console.log('Backend response: ' , responseDataJson);

// Penyesuaian Format Backend ke FrontEnd
let searchTime = responseDataJson.SearchTimeInMilliseconds || 0;
let nodesVisited = (responseDataJson.NodeCountElement || 0) +
(responseDataJson.NodeCountRecipe || 0);

let nodes = [];
let edges = [];
let idCounter = 1;

if (responseDataJson.isFound !== undefined) {

    // Satu Resep
    const steps = responseDataJson.Steps || [];

    if (!steps || steps.length === 0) {
        setRecipeResult(`Tidak ada resep yang ditemukan!`);
        return;
    }

    const rootId = idCounter++;

```

```

let prevResultId = rootId;
for (let i = 0; i < steps.length; i++) {

    const s = steps[i];
    const id1 = idCounter++;
    const id2 = idCounter++;
    const idResult = idCounter++;

    // Tambah Simpul/Nodes
    nodes.push({ id: id1, label: s[0] });
    nodes.push({ id: id2, label: s[1] });
    nodes.push({ id: idResult, label: s[2] });

    // Tambah Sisi/Edges
    edges.push({ from: id1, to: idResult });
    edges.push({ from: id2, to: idResult });

    // Menghubungkan Simpul
    if ( i === 0 ) {
        edges.push({ from: rootId , to: id1 });
        edges.push({ from: rootId , to: id2 });
    } else {
        const prevStep = steps[i-1];
        if (prevStep[2] === s[0]) {
            edges.push({ from: prevResultId, to: id1 });
        } else if (prevStep[2] === s[1]) {
            edges.push({ from: prevResultId, to: id2 });
        }
        else {
            edges.push({ from: rootId, to: id1 });
            edges.push({ from: rootId, to: id2 });
        }
    }
    prevResultId = idResult;
}

} else if (responseDataJson.isSatisfied !== undefined) {

    // Resep Banyak/Multiple
    const paths = responseDataJson.Steps || [];

```

```

let idCounter = 1;

if (!paths || paths.length === 0) {
  setRecipeResult(`Tidak ada resep yang ditemukan!`);
  return;
}

for (let i = 0; i < paths.length; i++) {
  const pathSteps = paths[i];
  let pathNodes = []
  let pathEdges = []

  for (let j = 0; j < pathSteps.length; j++) {
    const s = pathSteps[j];

    const id1 = idCounter++;
    const id2 = idCounter++;
    const idResult = idCounter++;

    // Tambah Simpul/Nodes
    pathNodes.push({ id: id1, label: s[0], group: i });
    pathNodes.push({ id: id2, label: s[1], group: i });
    pathNodes.push({ id: idResult, label: s[2], group: i });

    // Tambah Sisi/Edges
    pathEdges.push({ from: id1, to: idResult });
    pathEdges.push({ from: id2, to: idResult });

    // Menghubungkan Simpul
    if (j < pathSteps.length - 1) {
      const nextStep = pathSteps[j+1];

      if (nextStep[0] === s[2]) {
        const nextId1 = idCounter;
        pathEdges.push({ from: idResult, to: nextId1 });
      } else if (nextStep[1] === s[2]) {
        const nextId2 = idCounter + 1;
        pathEdges.push({ from: idResult, to: nextId2 })
      }
    }
  }
}

```



```

    }

    nodes = nodes.concat(pathNodes);
    edges = edges.concat(pathEdges);
  }
}

setSearchTime(searchTime.toFixed(2) || ' N/A');
setNodesVisited(nodesVisited || ' N/A');

let recipeCount = 0;
if (responseDataJson.isFound !== undefined) {
  recipeCount = 1;
} else if (responseDataJson.isSatisfied !== undefined) {
  recipeCount = (responseDataJson.Steps || []).length || 0;
}
setRecipeResult(search === 'satu' ? `Ditemukan satu resep untuk
${element}` : `Ditemukan ${recipeCount} resep untuk ${element}`);

const container = document.getElementById('tree');
const data = { nodes, edges };
const options = {
  layout: {
    hierarchical: {
      direction: 'UD',
      sortMethod: 'directed',

      nodeSpacing: 100,
      levelSeparation: 50,
      treeSpacing: 200
    },
  },
  edges: { arrows: 'to' },
  nodes: {
    shape: 'box',
    font: { size: 12 },
    color: {
      background: 'lightgray',
      border: 'black'
    }
  }
};

```

```

        },
        physics: {
            enabled: true
        },
        interaction: {
            hover: true
        }
    };
    new Network(container, data, options);
} catch(error) {
    console.error('Error:', error);
    setRecipeResult(`Gagal mengambil resep: ${error.message}`)
}
};

// Tampilan User Interface atau UI
return (
    <div style={{ padding: '20px' }}>
        <header className="App-header">
            <h1 className="App-title">Tugas Besar 2</h1>
            <h2 className="App-subtitle">Pencarian Resep dengan Algoritma
BFS/DFS dalam Permainan Little Alchemy 2</h2>
            <h3 className="App-subsubtitle">Dibuat oleh Kelompok The
Alchemist</h3>
        </header>

        <div>
            <label>Algoritma: </label>
            <select value={algorithm} onChange={ (e) =>
setAlgorithm(e.target.value)}>
                <option value="bfs">BFS</option>
                <option value="dfs">DFS</option>
            </select>
        </div>

        <div>
            <label>Pencarian: </label>
            <select value={search} onChange={ (e) =>

```

```

setSearch(e.target.value) }>
    <option value="resep">Satu Resep</option>
    <option value="semua">Semua Resep</option>
</select>
{search === 'semua' && (
    <input
        type="number"
        value={maxRecipes}
        onChange={ (e) => setMaxRecipes (Math.max(1,
parseInt(e.target.value))) }
        min="1"
        placeholder="Jumlah Resep Terbanyak"
    />
    ) }
</div>

<div>
    <label>Nama Element: </label>
    <input
        type="text"
        value={element}
        onChange={ (e) => setElement (e.target.value) }
        placeholder="Element"
    />
    <button onClick={findRecipes}>Cari Resep</button>
</div>

<div className="App-output">
    <div>
        <p>Waktu Pencarian: {searchTime} ms</p>
        <p>Jumlah Node yang Dikunjungi: {nodesVisited}</p>
        <p>Hasil Pencarian: {recipeResult}</p>
    </div>
    <div
        id="tree"
        style={{
            height: '500px',
            width: '100%',
            border: '1px solid lightgray',
        }}
    />

```

```
        </div>
    </div>
  )
}

export default App;
```

### 4.3 Penjelasan program

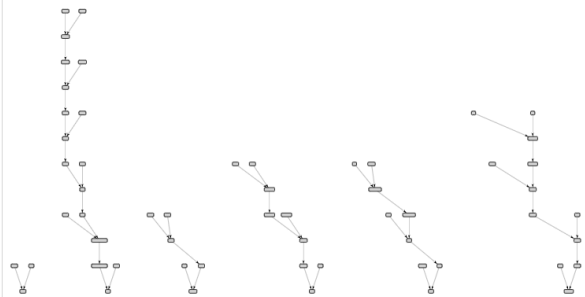
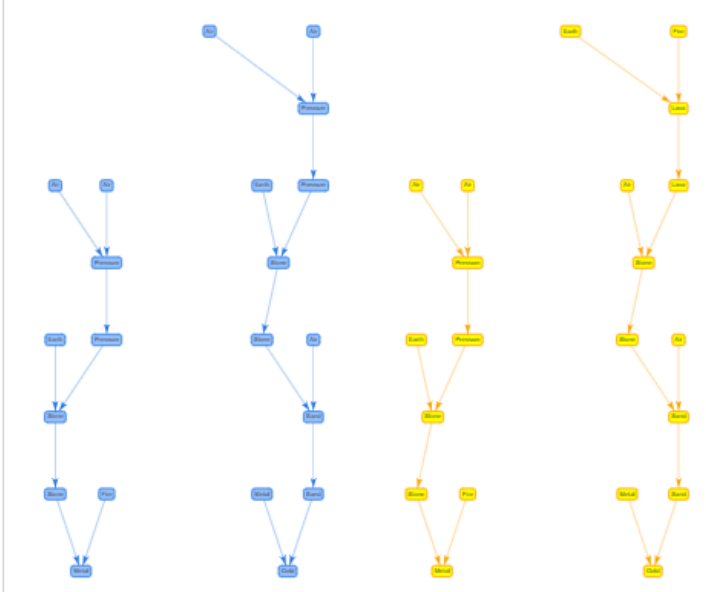

#### Alur penggunaan

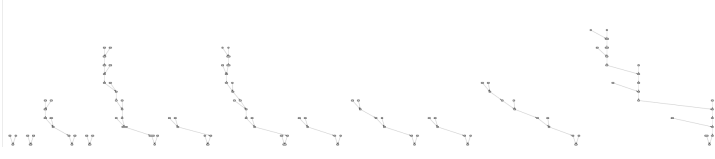
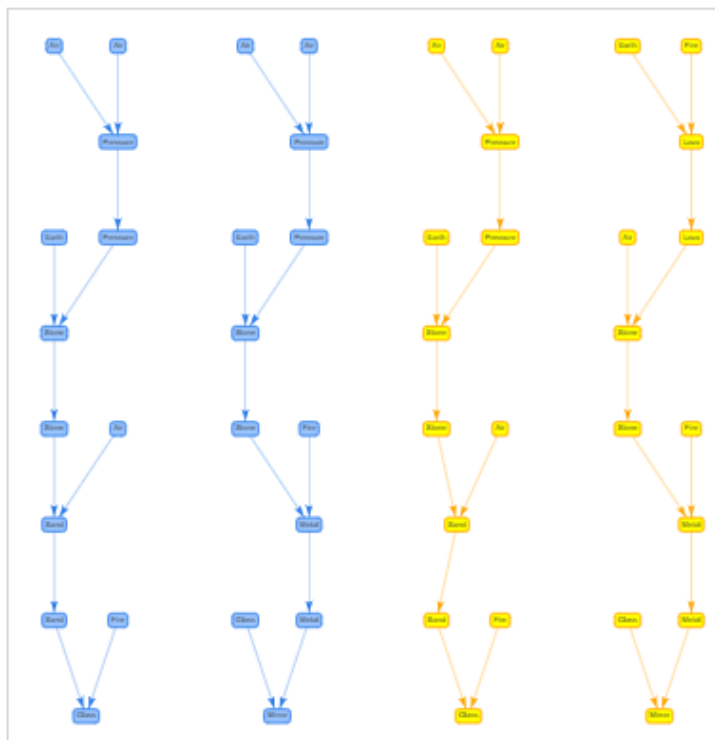

- Pengguna dapat memberikan input element yang akan dicari resepnya
- Pengguna dapat memilih algoritma yang akan digunakan
- Pengguna dapat memilih apabila ingin mencari beberapa alternatif resep dengan memberikan input n jumlah target resep

#### Fitur Program

- Program dapat melakukan scraping data secara otomatis dari web fandom little alchemy 2
- Program melakukan pencarian resep sesuai dengan batasan aturan dan pilihan algoritma pengguna
- Program memiliki fitur mencari single recipe ataupun multiple recipe dari sebuah elemen berdasarkan input pengguna
- Algoritma pencarian multiple recipe pada program dioptimasi menggunakan multithreading
- Program dapat menampilkan visualisasi hasil pencarian resep suatu elemen sebagai sebuah tree.

#### 4.4 Pengetesan program

Data	Hasil
Airplane, DFS, 1	<p>           Algoritma: <input type="text" value="DFS"/>             Pencarian: <input type="text" value="Satu Resep"/>             Nama Element: <input type="text" value="Airplane"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 58             Hasil Pencarian: Ditemukan satu resep untuk Airplane         </p> 
Gold, DFS, 2	<p>           Algoritma: <input type="text" value="DFS"/>             Pencarian: <input type="text" value="Semua Resep"/> 2             Nama Element: <input type="text" value="Gold"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 16             Hasil Pencarian: Ditemukan 2 resep untuk Gold         </p> 
Grilled cheese, DFS, 10	<p>           Algoritma: <input type="text" value="DFS"/>             Pencarian: <input type="text" value="Semua Resep"/> 10             Nama Element: <input type="text" value="Grilled cheese"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 216             Hasil Pencarian: Ditemukan 10 resep untuk Grilled cheese         </p> 

<p>Cow, BFS, 1</p>	<p>           Algoritma: <input type="text" value="BFS"/>             Pencarian: <input type="text" value="Semua Resep"/> <input type="text" value="2"/>             Nama Element: <input type="text" value="Mirror"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 78             Hasil Pencarian: Ditemukan 1 resep untuk Cow         </p> 
<p>Mirror, BFS, 2</p>	<p>           Algoritma: <input type="text" value="BFS"/>             Pencarian: <input type="text" value="Semua Resep"/> <input type="text" value="2"/>             Nama Element: <input type="text" value="Mirror"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 31             Hasil Pencarian: Ditemukan 2 resep untuk Mirror         </p> 
<p>Picnic, BFS, 10</p>	<p>           Algoritma: <input type="text" value="BFS"/>             Pencarian: <input type="text" value="Semua Resep"/> <input type="text" value="10"/>             Nama Element: <input type="text" value="Picnic"/> <input type="button" value="Cari Resep"/>             Waktu Pencarian: 0.00 ms             Jumlah Node yang Dikunjungi: 102             Hasil Pencarian: Ditemukan 10 resep untuk Picnic         </p> 

#### 4.5 Analisis Hasil Pengujian

Berdasarkan hasil pengetesan dapat ditarik kesimpulan bahwa algoritma DFS tidak menjamin akan selalu memberikan hasil berupa resep terpendek untuk mencapai elemen target. Hal tersebut sesuai dengan properti yang dimiliki DFS yakni *Optimality*, yaitu apabila DFS melakukan pencarian dengan langkah = biaya maka algoritma tidak menjamin hasilnya optimal. Namun, algoritma DFS memenuhi properti *Completeness* sebab sudah dilakukan penanganan *repeated states* dengan aturan sebuah elemen hanya bisa dibentuk dari elemen lain yang memiliki tier di bawahnya. Algoritma DFS juga memiliki kelebihan dibanding algoritma BFS, yakni kompleksitas ruang dalam polinomial, yaitu  $O(bm)$  dengan  $b$  adalah maksimum percabangan dari suatu simpul dan  $m$  adalah maksimum kedalaman dari ruang status. Sedangkan kompleksitas waktu dari algoritma DFS dalam eksponensial, yaitu  $O(b^m)$ .

Berbeda halnya dengan algoritma BFS yang selalu memberikan hasil berupa resep terpendek untuk mencapai elemen target. Hal tersebut sesuai dengan properti yang dimiliki BFS, yakni *Optimality* asalkan jumlah percabangan dari suatu simpul terbatas. Selain itu algoritma ini juga memenuhi properti *Completeness*. Sayangnya, Algoritma BFS juga memiliki kelemahan dibanding algoritma DFS, yakni kompleksitas ruang dalam eksponensial, yaitu  $O(b^d)$  dengan  $d$  adalah kedalaman dari solusi terbaik sehingga penggunaan memori bertumbuh sangat cepat untuk input data yang dalam misalnya elemen pada tier ke-15. Sedangkan kompleksitas waktu dari algoritma BFS dan DFS sama-sama dalam eksponensial, yaitu  $O(b^d)$ .

## **BAB V KESIMPULAN, SARAN, DAN REFLEKSI**

### **5.1. Kesimpulan**

Setelah melakukan pengujian dan analisis, dapat disimpulkan bahwa program pencarian resep pada permainan *Little Alchemy 2* dengan algoritma BFS (Breadth-First Search) dan DFS (Depth-First Search) telah berhasil diselesaikan dengan cukup baik. Implementasi algoritma pencarian tersebut mampu menunjukkan perbedaan jalur dan efisiensi pencarian terhadap kombinasi elemen dalam permainan. Penggunaan website sebagai antarmuka frontend turut memudahkan pengguna dalam mengakses dan mencoba fitur pencarian resep secara interaktif dan menarik.

### **5.2. Saran**

Untuk kedepannya bisa dipertimbangkan untuk melakukan improvisasi terhadap program dengan

1. Menggunakan algoritma pencarian lainnya seperti IDS, DLS, A\*
2. Menambahkan animasi proses pencarian resep sesuai dengan algoritma
3. Melakukan deployment agar program bisa diakses oleh khalayak luas

Lalu, saran terkhusus untuk kelompok adalah untuk memperbaiki manajemen waktu. Manajemen waktu yang baik sangat penting agar setiap tugas dapat diselesaikan dengan lebih terstruktur, menghindari penumpukan pekerjaan di akhir tenggat waktu, dan memastikan setiap anggota memiliki waktu yang cukup untuk melakukan revisi jika diperlukan.

### **5.3. Refleksi**

Pada Tugas Besar 2 ini, kami mendapatkan pengalaman berharga dalam memahami penerapan algoritma pencarian pada kasus nyata, serta bagaimana mengembangkan sistem berbasis web dari sisi backend maupun frontend. Proses kerja kelompok juga mengajarkan kami pentingnya komunikasi, pembagian tugas yang adil, serta manajemen waktu dalam menyelesaikan proyek teknis. Meskipun menghadapi berbagai tantangan, tugas ini memberikan pengalaman belajar yang berharga dan memperkuat pemahaman kami dalam berbagai aspek yang berkaitan dengan pengembangan proyek.



## LAMPIRAN

### 1. Tautan Repository Github

[https://github.com/iqbalhaidr/Tubes2\\_TheAlchemist](https://github.com/iqbalhaidr/Tubes2_TheAlchemist)

### 2. Tabel Ketercapaian

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 <i>multithreading</i>	√	
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-containerize dengan Docker.		√
11	Aplikasi di-deploy dan dapat diakses melalui internet.		√

## DAFTAR PUSTAKA

Amazon Web Service. (n.d.). *What is an API?*. Diambil dari

<https://aws.amazon.com/what-is/api/>

Institut Teknologi Bandung. (2025). *Spesifikasi Tugas Besar 2 Stima 2024/2025*. [Dokumen PDF]. Diambil dari

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tubes2-Stima-2025.pdf>

Little Alchemy 2. (n.d.). *Little Alchemy 2*. Diambil dari <https://littlealchemy2.com/>

Little Alchemy Wiki. (n.d.). *All elements in Little Alchemy 2*. Diambil dari

[https://little-alchemy.fandom.com/wiki/Elements\\_\(Little\\_Alchemy\\_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2))

MDN Web Docs. (n.d.). *Client-Server Architecture*. Diambil dari

[https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Extensions/Server-side/First\\_steps/Client-Server\\_overview](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview)

Microsoft. (n.d.). *Using React in Visual Studio Code*. Diambil dari

<https://code.visualstudio.com/docs/nodejs/reactjs-tutorial>

PuerkitoBio. (n.d.). *Goquery*. Diambil dari <https://github.com/PuerkitoBio/goquery>

React. (n.d.). *Learn React*. Diambil dari <https://react.dev/learn>

The Go Project. (n.d.). *Effective Go Concurrency*. Diambil dari

[https://go.dev/doc/effective\\_go#concurrency](https://go.dev/doc/effective_go#concurrency)

The Go Project. (n.d.). *Golang Documentation*. Diambil dari <https://go.dev/doc/>

Vercel. (n.d.). *Next Documentation*. Diambil dari <https://nextjs.org/docs>