

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



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

Asisten Pembimbing : Farhan Nafis Rayhan (13522037)

Disusun oleh:

Kelompok 42 – BrokenHeart

Maheswara Bayu Kaindra (13523015)

Jessica Allen (13523059)

Shanice Feodora Tjahjono (13523097)

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

**2025**

## **BAB 1 : Deskripsi Tugas**

**Batas pengumpulan :**

- Senin, 12 Mei 2025 pukul 22.11 WIB

## Arsip pengumpulan :

- Source program yang dapat dijalankan disertai README
  - Laporan (soft copy)

## 1.1 Deskripsi Tugas



*Gambar 1. Little Alchemy 2*

(sumber: <https://www.thegamer.com>, dimodifikasi dengan remove background)

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.

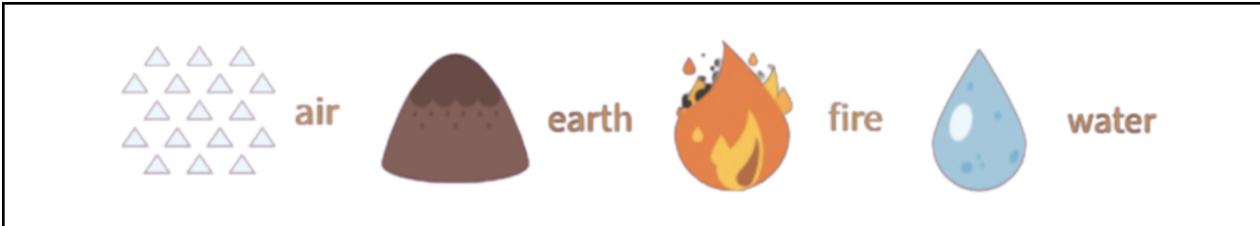
Mekanisme dari permainan ini adalah pemain dapat menggabungkan kedua elemen dengan melakukan drag and drop, jika kombinasi kedua elemen valid, akan memunculkan elemen baru, jika kombinasi tidak valid maka tidak akan terjadi apa-apa. Permainan ini tersedia di web browser, Android atau iOS

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.

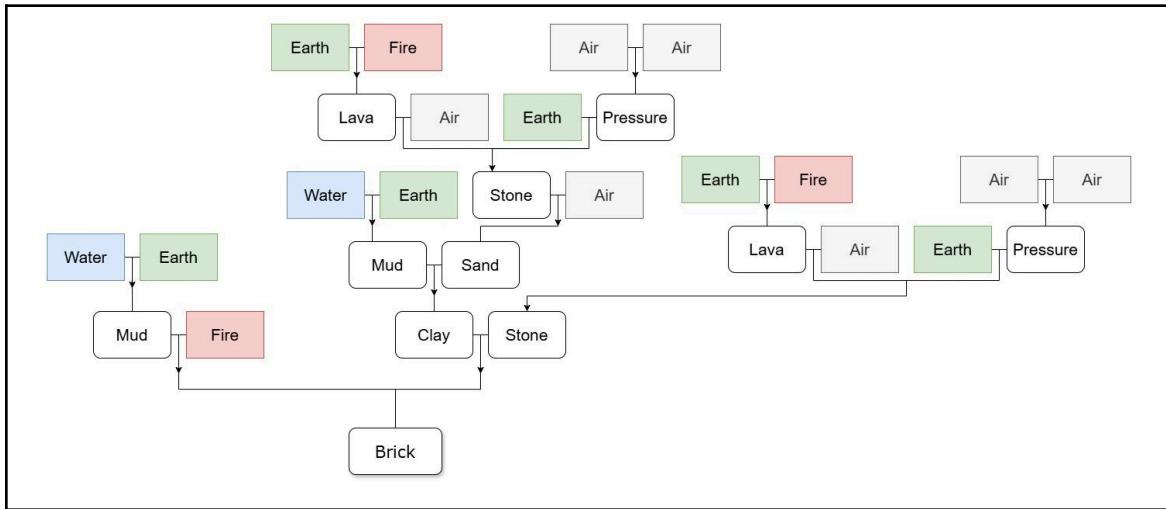
### 3. Combine Mechanism

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.

## 1.2 Spesifikasi Wajib

- Buatlah aplikasi pencarian recipe elemen dalam permainan Little Alchemy 2 dengan menggunakan strategi BFS dan DFS.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
- Aplikasi berbasis web, untuk frontend dibangun menggunakan bahasa Javascript dengan framework Next.js atau React.js, dan untuk backend menggunakan bahasa Golang.
- Untuk repository frontend dan backend diperbolehkan digabung maupun dipisah.
- Untuk data elemen beserta resep dapat diperoleh dari scraping [website Fandom Little Alchemy 2](#).
- Terdapat opsi pada aplikasi untuk memilih algoritma BFS atau DFS (juga bidirectional jika membuat bonus)
- Terdapat toggle button untuk memilih untuk menemukan sebuah recipe terpendek (output dengan rute terpendek) atau mencari banyak recipe (multiple recipe) menuju suatu elemen tertentu. Apabila pengguna ingin mencari banyak recipe maka terdapat cara bagi pengguna untuk memasukkan parameter banyak recipe maksimal yang ingin dicari. Aplikasi boleh mengeluarkan recipe apapun asalkan berbeda dan memenuhi banyak yang diinginkan pengguna (apabila mungkin).
- Mode pencarian multiple recipe wajib dioptimasi menggunakan multithreading.

- Aplikasi akan memvisualisasikan recipe yang ditemukan sebagai sebuah tree yang menunjukkan kombinasi elemen yang diperlukan dari elemen dasar. Agar lebih jelas perhatikan contoh berikut



Gambar 3. Contoh visualisasi recipe elemen

Gambar diatas menunjukkan contoh visualisasi recipe dari elemen Brick. Setiap elemen bersebelahan menunjukkan elemen yang perlu dikombinasikan. Amati bahwa leaf dari tree selalu berupa elemen dasar. Apabila dihitung, gambar diatas menunjukkan 5 buah recipe untuk Brick (karena Brick dapat dibentuk dengan kombinasi Mud + Fire atau Clay + Stone, begitu pula Stone yang dapat dibentuk oleh kombinasi Lava+Air atau Earth+Pressure). Visualisasi pada aplikasi tidak perlu persis seperti contoh diatas, tetapi pastikan bahwa recipe ditampilkan dengan jelas.

- Aplikasi juga menampilkan waktu pencarian serta banyak node yang dikunjungi.

### 1.3 Spesifikasi Bonus

- **(maks 5)** Membuat video tentang aplikasi BFS dan DFS pada permainan Little Alchemy 2 di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll. Semakin menarik video, maka semakin banyak poin yang diberikan.
- **(maks 3)** Aplikasi dijalankan menggunakan Docker baik untuk frontend maupun backend.
- **(maks 3)** Aplikasi di-deploy ke aplikasi deployment (aplikasi deployment bebas) agar bisa diakses secara daring
- **(maks 3)** Menambahkan algoritma bukan hanya DFS dan BFS, tetapi juga [strategi bidirectional](#)
- **(maks 6)** Aplikasi memiliki fitur Live Update visualisasi recipe selama proses pencarian. Tree visualisasi akan dibangun bertahap secara real time sesuai dengan progress pencarian. Tambahkan delay pada Live Update karena pencarian dapat berjalan dengan sangat cepat.
- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.

- Terdapat demo dengan asisten untuk mendemonstrasikan aplikasi yang telah dibuat. Pengumuman mengenai demo akan diberitahukan lebih lanjut oleh asisten.
- Setiap kelompok harap mengisi nama kelompok dan anggotanya pada link berikut, paling lambat Kamis, 24 April pukul 22.11 WIB.

**+** Pendataan Kelompok Tubes 2 Stima 2024/2025

- Diwajibkan untuk memilih asisten meskipun tidak melakukan asistensi, karena asisten yang dipilih akan menjadi asisten saat asistensi (opsional) dan demo tugas besar. Pemilihan asisten dapat dilakukan pada link berikut, paling lambat Kamis, 24 April 2025 pukul 22.11 WIB.

**+** Pendataan Kelompok Tubes 2 Stima 2024/2025

- Program disimpan dalam repository yang bernama Tubes2\_NamaKelompok (bila digabung) dan Tubes2\_FE/BE\_NamaKelompok (bila dipisah) dengan nama kelompok sesuai dengan yang di sheets diatas. Berikut merupakan struktur dari isi repository tersebut:
  - Folder src berisi program yang dapat dijalankan
  - Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf
  - README untuk tata cara penggunaan yang minimal berisi:
    - Penjelasan singkat algoritma DFS dan BFS yang diimplementasikan
    - Requirement program dan instalasi tertentu bila ada
    - Command atau langkah-langkah dalam meng-compile atau build program
    - Author (identitas pembuat)
- Sangat disarankan untuk menggunakan semantic commit. Buatlah release dengan format v1.x dengan x adalah nomor revisi dimulai dari revisi 0. Contoh v1.0 untuk release pertama, v1.1 untuk revisi selanjutnya.
- Pastikan untuk membuat repository bersifat Public paling lambat H+1 deadline (1 hari setelah deadline). Sebelum deadline repository harus bersifat Private.
- Laporan dikumpulkan hari Senin, 12 Mei 2025 pada alamat Google Form berikut paling lambat pukul 22.11 WIB:

<https://bit.ly/tubes2stima25>

PERINGATAN: Keterlambatan akan mengurangi nilai sebanyak 1 poin untuk setiap menit keterlambatan.

- Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut: <https://bit.ly/QnA-Stima-25>.

## 1.4 Isi laporan

- **Cover:** Cover laporan ada foto anggota kelompok (foto bertiga). Foto ini menggantikan logo “gajah” ganesha.
- **Bab 1:** Deskripsi tugas (dapat menyalin spesifikasi tugas ini).
- **Bab 2:** Landasan Teori.
  - Dasar teori (Penjelajahan Graf serta algoritma Breadth First Search dan Depth First Search secara umum serta bonus).
  - Penjelasan singkat mengenai aplikasi web yang dibangun.
- **Bab 3:** Analisis Pemecahan Masalah.
  - Langkah-langkah pemecahan masalah.

- Proses pemetaan masalah menjadi elemen-elemen algoritma DFS dan BFS.
- Fitur fungsional dan arsitektur aplikasi web yang dibangun.
- Contoh ilustrasi kasus.
- **Bab 4:** Implementasi dan pengujian.
  - Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).
  - Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).
  - Hasil pengujian minimal 3 buah elemen dalam bentuk screenshot antarmuka. Variasikan setiap kasus dengan berbagai fitur yang diimplementasikan (Algoritma serta banyak recipe yang dicari).
  - Analisis hasil pengujian.
- **Bab 5:** Kesimpulan, Saran, dan Refleksi tentang Tugas Besar 2.
- **Lampiran:** Tautan repository GitHub (dan video jika membuat)
- **Daftar Pustaka**

**Keterangan laporan:**

1. Laporan ditulis dalam bahasa Indonesia yang baik dan benar.
2. Laporan mengikuti format pada section “Isi laporan” dengan baik dan benar.
3. Identitas per halaman harus jelas (misalnya : halaman, kode kuliah).

## 1.5 Penilaian

### 1. Bagian 1: Laporan (40%)

- a. Langkah-langkah pemecahan masalah, proses pemetaan masalah, fitur fungsional dan arsitektur aplikasi web yang dibangun, dan contoh ilustrasi kasus. (20 %)
- b. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun), penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya), hasil pengujian (Screenshot antarmuka dan variasi pengujian kasus berdasarkan fitur aplikasi), dan analisis hasil pengujian. (15 %)
- c. Kelengkapan komponen-komponen pada laporan dan README. (5 %)

### 2. Bagian 2: Implementasi Program dan Demo (60%)

- a. Kecocokan output dari test case yang dimiliki Asisten. (25 %)
- b. Kebenaran algoritma Depth First Search dan Breadth First Search, sesuai dengan apa yang telah diajarkan di kelas. (15%)
- c. Pemahaman tugas besar dan algoritma yang telah dibuat oleh masing-masing anggota. (10%)
- d. Keberhasilan input dan output, sesuai dengan komponen-komponen yang ada pada spesifikasi. (5%)
- e. Modularitas/keterbacaan penulisan program. (5%)

### **3. Bagian 3: Komponen Bonus (Maksimal 20 Poin)**

- a. Program dapat menampilkan Live Update visualisasi secara real time. (bonus maksimal 6 poin)
- b. Aplikasi dapat melakukan pencarian dengan strategi Bidirectional. (bonus maksimal 3 poin)
- c. Aplikasi di-deploy dan dapat diakses melalui internet. (bonus maksimal 3 poin)
- d. Program dijalankan menggunakan Docker baik untuk frontend maupun backend. (bonus maksimal 3 poin)
- e. Bonus dalam membuat video kelompok. (bonus maksimal 5 poin)

#### **1.6 Perhatian**

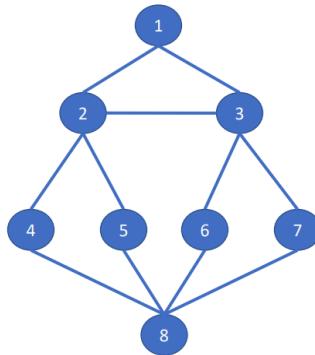
- Dilarang keras copy paste program dari internet, AI, repository lain, ataupun program milik teman. Program harus dibuat sendiri, kecurangan akan diberikan sanksi berat yaitu nilai tugas menjadi nol.
- Pastikan program dapat dikompilasi setidaknya pada windows dan linux.
- Apabila program tidak dapat dijalankan maka tidak akan dinilai oleh asisten.
- Keterlambatan pengumpuluan akan mengurangi 1 poin untuk setiap menit keterlambatan.

## BAB 2 : Landasan Teori

Dalam pengimplementasian pencarian pada *Little Alchemy 2*, elemen-elemen dapat direpresentasikan dengan simpul-simpul dalam sebuah graf. Untuk menelusuri setiap simpul dari graf elemen tersebut, dapat digunakan dua algoritma sistematis, yaitu pencarian melebar (*Breadth First Search*) dan pencarian mendalam (*Depth First Search*).

### 2.1 Algoritma BFS (Breadth First Search)

Breadth First Search merupakan suatu algoritma pencarian berbasis graf yang memungkinkan iterasi dari satu *node* ke *node* lain tanpa adanya informasi tambahan. Berikut merupakan algoritma penelusuran simpul graf menggunakan BFS.



Gambar 4. Graf Contoh

Sumber : Rinaldi Munir

1. Traversal dimulai dari simpul pertama (simpul 1), kunjungi simpul pertama.
2. Kunjungi semua simpul tetangga simpul yang sedang ditinjau (jika sekarang Anda berada pada simpul 1, kunjungi simpul 2, lalu 3).
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang dikunjungi sebelumnya (begitu seterusnya sampai semua simpul berhasil dikunjungi).

Pada gambar di atas, urutan kunjungan simpul dengan menggunakan BFS adalah 1-2-3-4-5-6-7-8.

Untuk membentuk algoritma ini menjadi program, dibutuhkan matriks ketetanggaan yang berukuran  $n \times n$ , dengan  $n$  merupakan banyaknya simpul; sebuah *queue* untuk menyimpan daftar simpul yang akan dikunjungi; tabel *boolean* yang menyimpan informasi sudah atau belumnya kunjungan ke suatu simpul (*true* jika sudah, *false* jika belum).

```
procedure BFS(input v:integer)
Deklarasi
  w : integer
  q : queue
  procedure BuatAntrian(input/output q : antrian)
  { membuat antrian kosong, kepala diisi dengan nilai 0 }
```

```

procedure MasukAntrian(input/output q : antrian, input v : integer)
{ memasukkan v ke dalam paling belakang q }
procedure HapusAntrian(input/output q : antrian, input v : integer)
{ menghapus nilai paling depan q dan disimpan ke dalam v }
function AntrianKosong(input q : antrian) → boolean
{ bernilai true jika antrian q kosong, false jika tidak }

Algoritma
BuatAntrian(q)
write(v) { cetak simpul awal yang dikunjungi }
dikunjungi(v) ← true
MasukAntrian(q,v)

{ kunjungi semua simpul graf hingga antrian kosong }
while not AntrianKosong(q) do
    HapusAntrian(q,v)
    for (simpul w : w tetangga v) do
        if not dikunjungi(w) then
            write(w){ cetak simpul yang dikunjungi }
            MasukkanAntrian(q,w)
            dikunjungi(w) ← true
        endif
    endfor
endwhile

```

## 2.2 Algoritma DFS (Depth First Search)

Berbeda dengan BFS, algoritma DFS merupakan algoritma pencarian simpul pohon yang mendalam, di mana simpul yang lebih dalam akan terus dikunjungi sebelum tidak ada lagi cabang mendalam yang dapat dikunjungi. Oleh karena itu, pencarian dengan DFS cenderung lebih hemat ruang, namun lebih memakan waktu.

Berikut merupakan algoritma DFS. Gunakan gambar di atas (Gambar 4) sebagai ilustrasi.

1. Kunjungi simpul pertama, misal simpul v (di sini simpul 1).
2. Kunjungi simpul tetangga v, misal w.
3. Ulangi pemanggilan DFS secara rekursif dengan simpul w sebagai simpul pertama.
4. Lakukan pencarian runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya apabila **semua simpul tetangga sudah dikunjungi**.
5. Pencarian dianggap selesai jika tidak ada lagi simpul yang bisa dikunjungi (semua sudah dikunjungi).

Dari graf pada gambar 4, urutan kunjungan simpul dengan menggunakan DFS adalah 1-2-4-8-5-6-3-7.

```

procedure DFS(input v: integer)
Deklarasi
    w : integer
Algoritma
    write(v)

```

```
dikunjungi(v) ← true
for (w ← 1 to n) do
    if A[v, w] = 1 then simpul v dan w bertetangga
        if not dikunjungi(w) then
            DFS(w)
        endif
    endif
endfor
```

## 2.3 Arsitektur Client-Server dengan Menggunakan Next.js

### 2.3.1 Arsitektur Client-Server

Arsitektur client-server merupakan model dasar dalam pengembangan aplikasi perangkat lunak, terutama aplikasi berbasis jaringan atau internet. Pada arsitektur ini, biasanya peran sistem terpisah antara peran klien dan juga server. Pembagian ini memiliki tujuan untuk membuat sistem yang lebih terstruktur, modular, dan mudah untuk dikembangkan maupun dipelihara.

Client pada arsitektur ini berperan sebagai antarmuka yang digunakan oleh pengguna. Client biasanya berupa aplikasi web, aplikasi mobile atau bahkan perangkat lunak pada desktop yang berjalan di sisi pengguna. Tugas utama dari client ini adalah untuk menampilkan informasi kepada pengguna serta mengirimkan permintaan kepada server, yang didasarkan pada interaksi dari pengguna pada antarmuka. Misalnya, ketika seorang pengguna mengisi sebuah formulir pada antarmuka, kemudian menekan tombol “Masuk”, client kemudian akan mengirimkan data yang dimasukkan user tersebut ke server untuk diverifikasi.

Pada sisi lain, server berperan untuk memproses permintaan yang *di-passing* dari client tersebut dengan menjalankan logika pemrosesan serta mengakses atau manipulasi data. Sistem pemrosesan ini mencakup banyak hal, seperti validasi data, eksekusi logika bisnis, pengambilan data dari database, hingga pengolahan data melalui perhitungan atau algoritma tertentu. Selain itu, server juga mengatur akses terhadap sumber daya yang bersifat terbatas atau sensitif, seperti data pengguna atau informasi yang disimpan pada basis data. Kemudian, setelah pemrosesan selesai, pihak server akan mengembalikan hasilnya kepada client dalam bentuk data (respon). Data ini biasanya dibungkus dalam format JSON, XML, atau HTML, tergantung pada jenis aplikasinya.

### 2.3.2 Next.js

[Next.js](#) adalah sebuah framework open-source yang berbasis [React.js](#) yang dirancang untuk menyederhanakan proses pengembangan aplikasi web modern yang lebih cepat, dapat responsif, dan mudah untuk dioptimasi. [Next.js](#) ini dikembangkan oleh perusahaan yang bernama Vercel, yang pada saat ini telah menjadi salah satu framework yang paling populer di kalangan pengembang web karena kemampuannya menggabungkan berbagai pendekatan rendering, routing otomatis, serta fitur-fitur lainnya yang siap pakai dan menghemat waktu serta usaha dalam proses pengembangan.

Keunggulan utama dari framework [Next.js](#) adalah fleksibilitasnya dalam metode rendering, yang memungkinkan pengembang yang menggunakannya untuk memilih strategi yang paling sesuai untuk kebutuhan spesifik aplikasi mereka. Framework ini juga mendukung tiga jenis pendekatan rendering utama secara bawaan, yaitu *server-side rendering*, *static site generation*, dan *client-side rendering*.

Server-side rendering adalah pendekatan di mana halaman web di render terlebih dahulu di server setiap kali ada permintaan dari client. Hal ini berarti setiap kali pengguna mengakses sebuah halaman, server akan membuat halaman HTML berdasarkan data terkini yang mengirimkannya ke browser. Pendekatan ini sangat sesuai untuk aplikasi yang memerlukan konten yang dinamis serta ingin diindeks dengan baik oleh mesin pencari. Hal ini karena konten dari aplikasi sudah tersedia ketika halaman dimuat, maka performa dan pengalaman pengguna juga menjadi lebih baik. Dalam pengindeksan oleh Google atau platform mesin pencari lainnya juga menjadi lebih lancar.

Kemudian, pada *static site generation*, halaman web di-*render* terlebih dahulu saat proses proses build. Sistem ini sangat cocok untuk halaman yang datanya jarang berubah. Contohnya, halaman dokumentasi, blog, ataupun landing page. Halaman dalam pendekatan ini telah disiapkan terlebih dahulu dalam bentuk file HTML statis, sehingga performa akses menjadi sangat cepat, dan beban server pun berkurang drastis. Pendekatan SSG ini merupakan pilihan yang ideal untuk situs yang mengutamakan kecepatan dan efisiensi.

Terakhir, client-side rendering (CSR), yaitu pendekatan ketika rendering halaman terjadi sepenuhnya pada sisi client menggunakan bahasa JavaScript setelah halaman dimuat pada browser. Pendekatan ini umum digunakan pada aplikasi SPA (Single Page Application), yaitu ketika interaksi antarpengguna berlangsung tanpa perlu memuat ulang halaman dari server. CSR cocok untuk konten yang sangat dinamis dan bersifat personalisasi, misalnya dashboard pengguna atau aplikasi real-time yang banyak bergantung pada interaksi.

Kelebihan [Next.js](#) yang dapat menggabungkan berbagai pendekatan ini dalam satu proyek menjadikannya sangat fleksibel dan adaptif. Seorang pengembang dapat memilih untuk merender satu halaman dengan SSG untuk performa maksimal, sementara halaman lain menggunakan SSR untuk menampilkan data terbaru, atau CSR untuk interaktivitas yang tinggi, semuanya dalam satu aplikasi.

Selain kelebihan itu, [Next.js](#) juga menawarkan banyak fitur lain yang siap pakai dalam proses pengembangan. Hal ini tentunya sangat membantu bagi developer yang ingin mengembangkan dengan lebih cepat. Salah satu fitur tersebut adalah routing otomatis berbasis file, dimana setiap file dalam direktori app secara otomatis menjadi route URL tanpa perlu konfigurasi tambahan. Selain itu, terdapat juga code splitting otomatis, dimana hanya kode yang dibutuhkan untuk setiap halaman yang akan dimuat untuk meningkatkan efisiensi dan juga kecepatan. [Next.js](#) juga mendukung penggunaan TypeScript secara bawaan, yang memudahkan pengembangan aplikasi yang lebih terstruktur dan bebas dari error. Framework ini juga mendukung optimasi gambar, kompresi dan memuat gambar dengan cerdas untuk mempercepat waktu muat halaman. [Next.js](#)

juga memungkinkan penulisan fungsi-fungsi backend sederhana langsung dalam proyek tanpa memerlukan server terpisah.

Dalam proyek yang memiliki backend terpisah seperti menggunakan Go pada server, [Next.js](#) juga mendukung komunikasi antar sistem melalui API eksternal menggunakan fetch, axios, atau pustaka HTTP lainnya. Dengan pendekatan ini, [Next.js](#) berfungsi murni sebagai client yang mengonsumsi data dari server melalui protokol HTTP, tanpa bergantung pada logika backend internal, yang menjadikan arsitektur aplikasi lebih modular dan terdistribusi, yang membuat frontend dan backend dapat dikembangkan, diuji, dan di-deploy secara independen.

## BAB 3 : Analisis Pemecahan Masalah

### 3.1 Desain Pemetaan Permasalahan untuk Implementasi BFS dan DFS

Misalkan pada *website* setiap elemen disimpan dalam tabel dengan format:

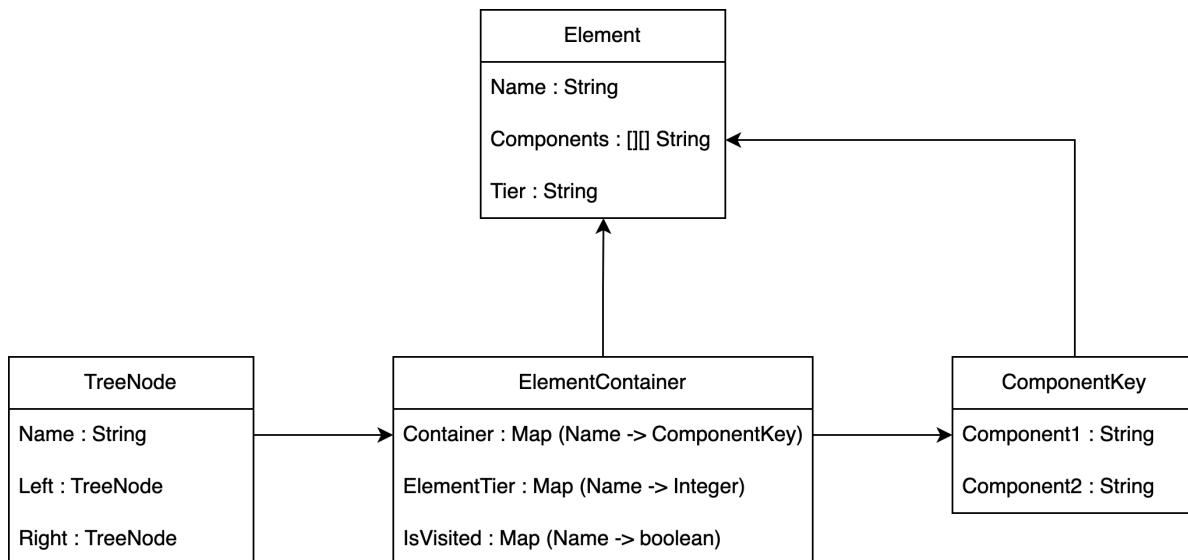
<nama-elemen>	<resep-pembentuk-1> : tuple of strings {Component1, Component2} <resep-pembentuk-2> <resep-pembentuk-3> ... <resep-pembentuk-n>
---------------	---

Untuk mengimplementasikan algoritma BFS dan DFS pada penelusuran resep setiap elemen tersebut, digunakan representasi struktur data sebagai berikut.

```
Element {
    Name : string
    Components : [][]string
    Tier : int
}
```

Elemen tersebut nantinya akan direpresentasikan menjadi sebuah pohon *binary* (dengan bantuan beberapa struktur data lain) dengan struktur:

```
BinaryTree {
    Head : string //diisi nama elemen
    Left : BinaryTree
    Right : BinaryTree
}
```



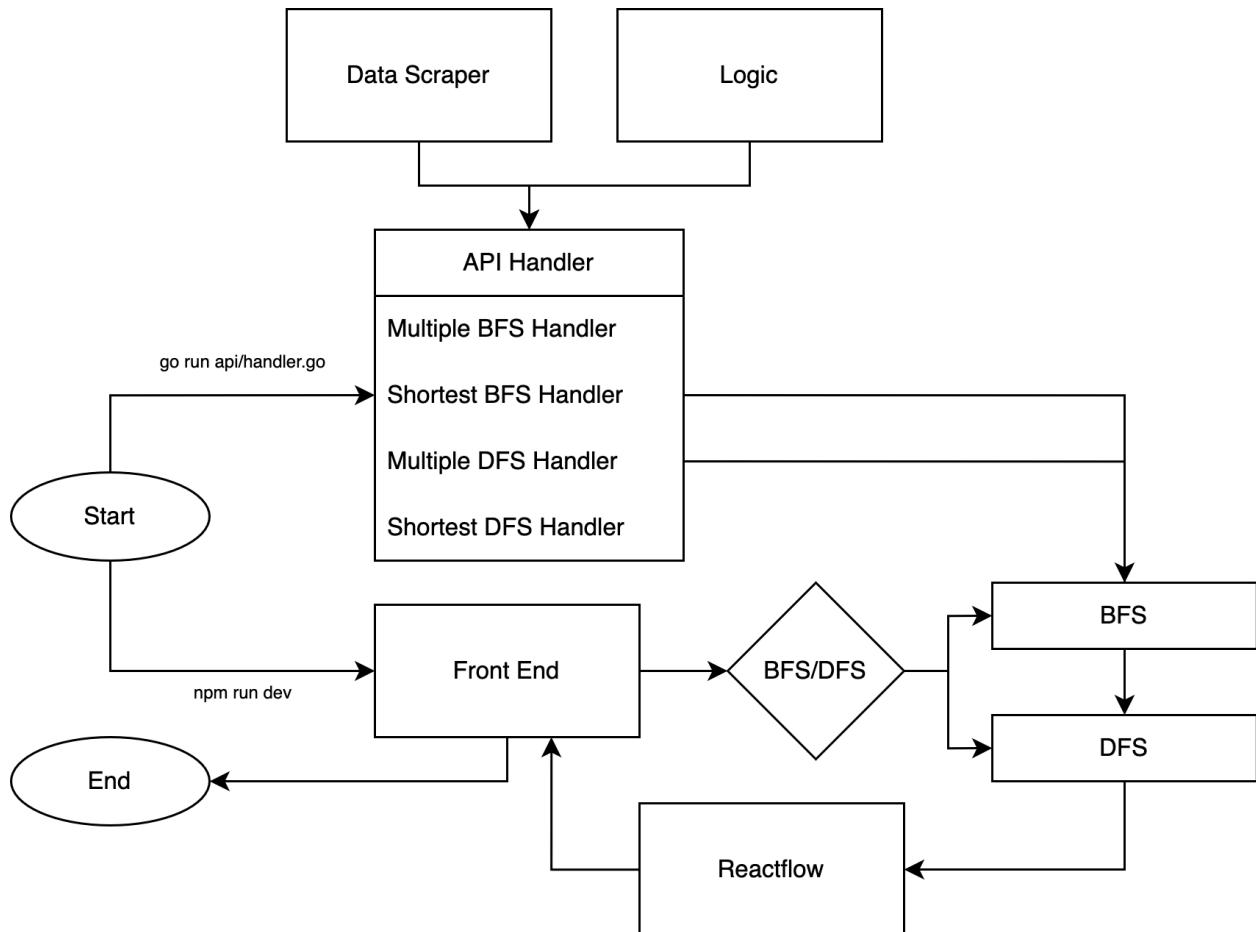
Gambar 5 : Diagram Kelas Sederhana untuk Menyampaikan Struktur Data yang digunakan.

Dibuat dengan [draw.io](#)

### 3.2 Desain Sekuensial Penyelesaian Masalah

Secara umum, Penyelesaian permasalahan (tahapan pengkerjaan) dapat didekomposisi menjadi beberapa tahapan utama yaitu sebagai berikut.

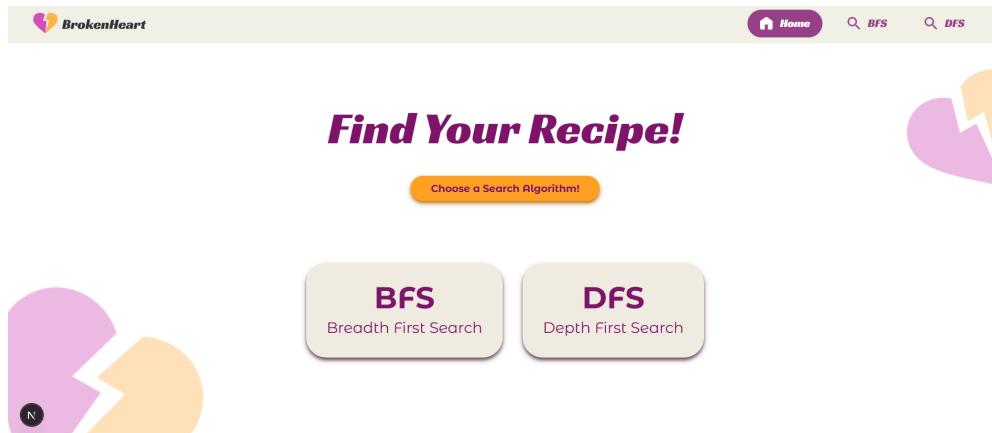
1. *Data scraping*, pengambilan data yang mencakup nama, resep, dan tier untuk setiap elemen data. Data elemen-elemen tersebut disimpan dalam suatu file .json dengan format nama elemen, komponen pembentuk, dan tier.
2. *Data reading*, data yang sudah discraping dibaca kembali dan disimpan di dalam struktur data agar dapat di-passing ke *frontend* (data sekarang berbentuk *ElementContainer*).
3. Setiap data pada *ElementContainer* diproses sesuai algoritma yang diinginkan (BFS atau DFS) dan dikembalikan dalam bentuk *TreeNode*.
4. *TreeNode* di-passing ke *frontend* dengan API (port 8080) untuk divisualisasikan dengan modul eksternal (misalnya *react flow*) untuk menghasilkan ilustrasi tree.



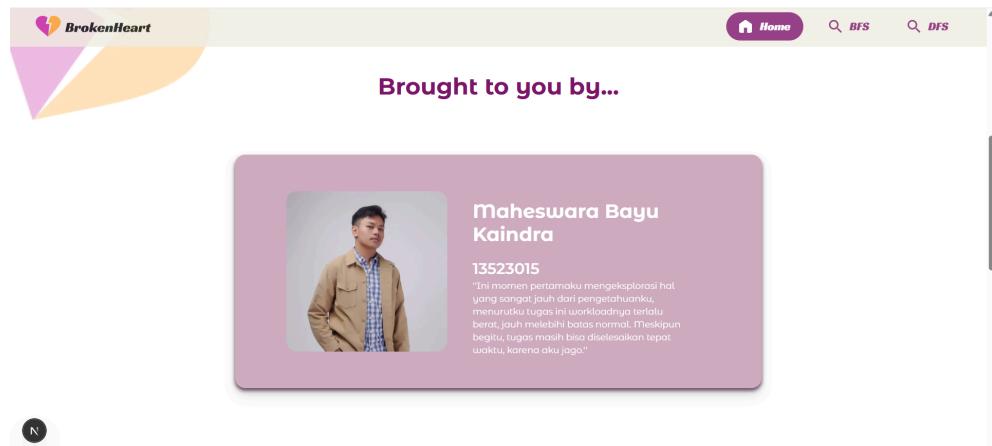
Gambar 6 : Diagram Sekuensial Penyelesaian Masalah

Dibuat dengan [draw.io](#)

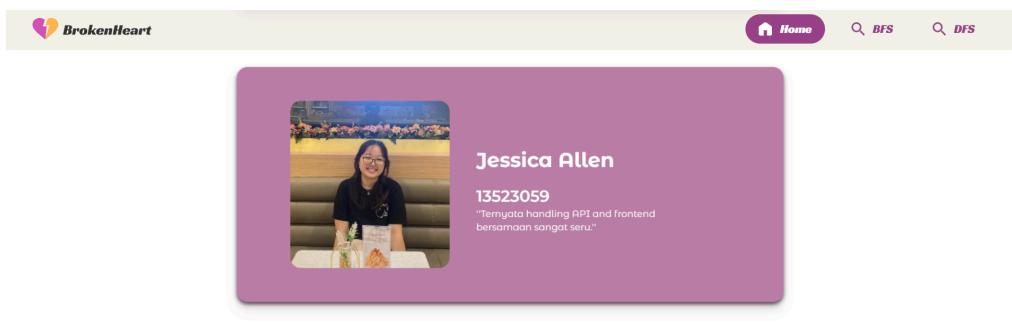
### 3.3 Struktur Aplikasi Web



Gambar 3.3.1.1. Home (1)



Gambar 3.3.1.2. Home (2)

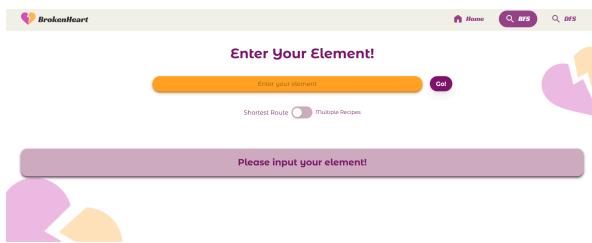


Gambar 3.3.1.3. Home (3)

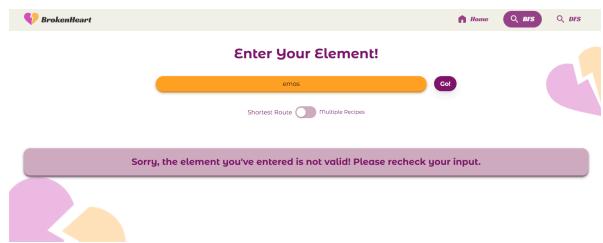


Gambar 3.3.1.4. Home (4)

Homepage aplikasi Little Alchemy 2 Recipe Finder ini berfungsi sebagai titik awal bagi pengguna untuk memulai pencarian resep elemen dalam permainan. Halaman ini menyediakan dua pilihan algoritma pencarian yang dapat digunakan, yaitu BFS (*Breadth First Search*) dan DFS (*Depth First Search*), yang keduanya ditampilkan dalam bentuk kartu pilihan di tengah halaman. Pengguna dapat memilih salah satu algoritma ini untuk melanjutkan ke halaman pencarian utama. Di bagian header, terdapat logo tim pengembang (BrokenHeart), tombol navigasi "Home" serta akses cepat ke algoritma "BFS" dan "DFS". Selain itu, terdapat juga informasi mengenai pengembang di bawah kartu pilihan algoritma. Informasi yang terdapat di dalamnya meliputi nama, NIM, serta pesan kesan pengembang dalam pembuatan proyek ini.



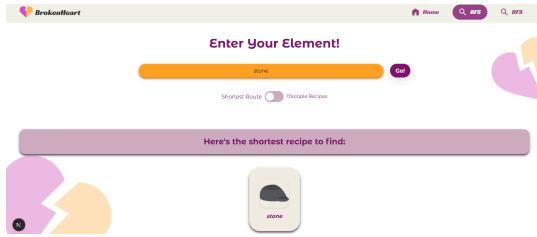
Gambar 3.3.2. Tampilan Awal BFS Shortest Route



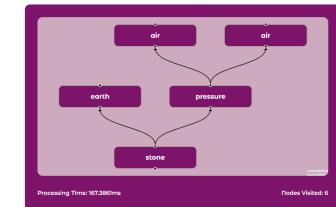
Gambar 3.3.3. Elemen Tidak Valid pada BFS Shortest Route

Gambar 3.3.2 merupakan tampilan awal halaman BFS Shortest Route. Terdapat instruksi awal untuk memberi tahu pengguna bahwa mereka perlu memasukkan nama elemen yang ingin dicari di kotak pencarian sebelum dapat melanjutkan proses pencarian. Terdapat toggle switch yang memungkinkan pengguna beralih antara mode "Shortest Route" (jalur terpendek) dan "Multiple Recipes" (beberapa resep).

Gambar 3.3.3 menunjukkan pesan kesalahan pada halaman BFS Shortest Route dengan teks "Sorry, the element you've entered is not valid! Please recheck your input." Pesan ini muncul ketika pengguna memasukkan nama elemen yang tidak terdapat dalam database permainan Little Alchemy 2, memberi tahu pengguna untuk memeriksa kembali masukan mereka dan mencoba dengan nama elemen yang valid.



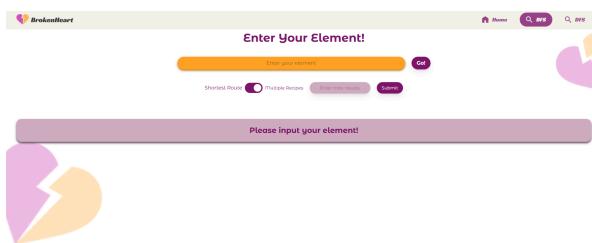
Gambar 3.3.4. Hasil Pencarian BFS Shortest Route



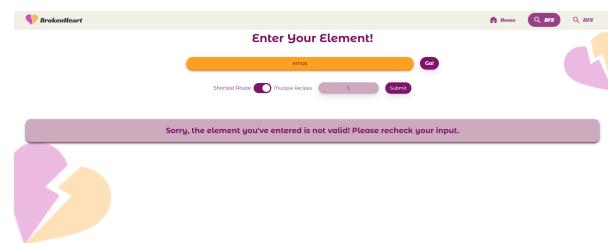
Gambar 3.3.5. Hasil Pencarian BFS Shortest Route

Ketika elemen ditemukan, ditampilkan hasil awal pencarian BFS Shortest Route dengan pesan "Here's the shortest recipe to find:". Di bawah pesan tersebut, ditampilkan ikon elemen yang merupakan elemen target pencarian. Tampilan ini menunjukkan bahwa sistem telah berhasil menemukan resep untuk membuat elemen stone menggunakan algoritma BFS.

Gambar 3.3.5 memperlihatkan visualisasi pohon resep lengkap hasil pencarian BFS Shortest Route untuk elemen yang dimasukkan user. Pohon menunjukkan jalur pembuatan elemen dari elemen-elemen dasar. Pada halaman ini, terdapat juga informasi tambahan seperti waktu yang dibutuhkan untuk pencarian, serta jumlah node yang dikunjungi dalam proses pencarian.



Gambar 3.3.6. Tampilan Awal BFS Multiple Recipe



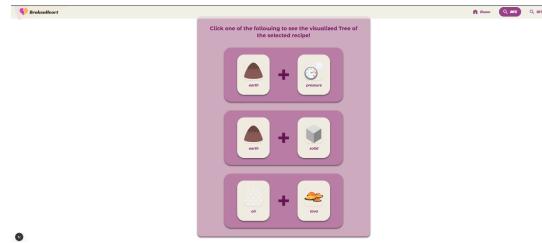
Gambar 3.3.7. Elemen Tidak Valid pada BFS Multiple Recipe

Gambar 3.3.6 menunjukkan tampilan awal halaman BFS Multiple Recipe dengan toggle switch yang diaktifkan pada posisi "Multiple Recipes". Pada halaman ini terdapat kotak input, tombol "Go!", serta kolom tambahan untuk memasukkan jumlah maksimal hasil yang diinginkan dan tombol "Submit" untuk mengonfirmasi. Panel notifikasi menampilkan pesan "Please input your element!" yang menginstruksikan pengguna untuk memasukkan nama elemen target sebelum memulai pencarian beberapa resep.

Gambar 3.3.7 memperlihatkan pesan kesalahan pada halaman BFS Multiple Recipe dengan teks "Sorry, the element you've entered is not valid! Please recheck your input." Pesan ini muncul ketika pengguna memasukkan teks yang bukan merupakan nama elemen valid dalam database permainan Little Alchemy 2.



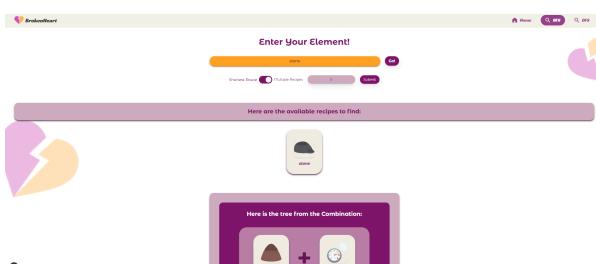
Gambar 3.3.8. Daftar Kombinasi BFS Multiple Recipe



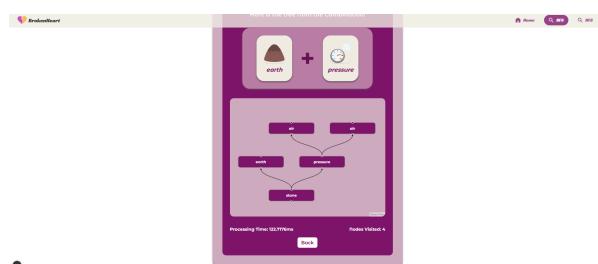
Gambar 3.3.9. Daftar Kombinasi BFS Multiple Route

Gambar 3.3.8 menampilkan hasil pencarian awal BFS Multiple Recipe untuk elemen "stone". Terdapat pesan "Here are the available recipes to find:" yang menandakan bahwa sistem telah menemukan beberapa resep untuk membuat elemen target. Di bawah pesan tersebut, ditampilkan ikon elemen yang menjadi target pencarian. Tampilan ini merupakan langkah pertama dari mode multiple recipe, di mana pengguna akan dapat melihat berbagai kombinasi resep yang tersedia.

Gambar 3.3.9 memperlihatkan lanjutan dari hasil pencarian BFS Multiple Recipe untuk elemen. Terdapat daftar kombinasi lengkap hasil pencarian BFS Multiple Route. Pada halaman ini, ditampilkan daftar kombinasi resep berbeda yang jumlah kombinasinya sudah dibatasi angka maksimum masukan pengguna, untuk membuat elemen pilihan pengguna. Setiap kombinasi ditampilkan di mana pengguna pilihan untuk mengklik salah satu kombinasi untuk melihat visualisasi pohon resep lengkap dari kombinasi tersebut. Tampilan ini memungkinkan pengguna menjelajahi berbagai cara alternatif untuk membuat elemen target.



Gambar 3.3.10. Hasil Pencarian BFS Multiple Recipe

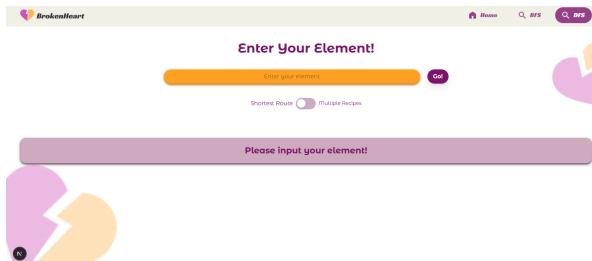


Gambar 3.3.11. Hasil Pencarian BFS Multiple Recipe

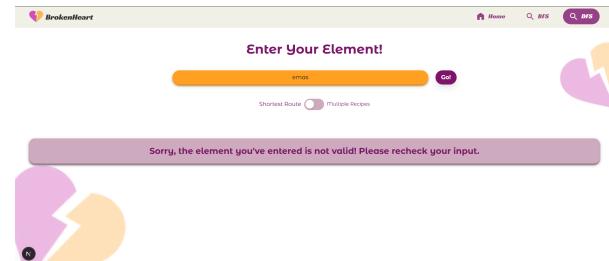
Gambar 3.3.10 menampilkan halaman hasil pencarian BFS Multiple Recipe yang menunjukkan kombinasi elemen yang telah dipilih oleh pengguna. Panel atas menampilkan pesan "Here are the available recipes to find:" dengan ikon elemen target yang dicari pengguna. Di bawahnya terdapat panel tambahan dengan label "Here is the tree from the Combinations:" yang menampilkan kombinasi elemen dasar yang dapat digunakan untuk membuat elemen target tersebut. Tampilan ini memperlihatkan bagian atas halaman visualisasi pohon resep dalam mode multiple recipe.

Gambar 3.3.11 memperlihatkan visualisasi pohon resep lengkap dari kombinasi elemen yang dipilih pengguna dalam mode BFS Multiple Recipe. Panel visualisasi menampilkan struktur hierarkis yang menunjukkan bagaimana elemen target dapat dibuat dari kombinasi elemen-elemen dasar dengan jalur yang jelas. Bagian bawah panel menyajikan informasi performa seperti "Processing Time" yang

menunjukkan waktu pemrosesan pencarian dan "Nodes Visited" yang mengindikasikan jumlah node yang dikunjungi selama proses pencarian. Tombol "Back" di bagian bawah memungkinkan pengguna kembali ke daftar kombinasi untuk memilih jalur alternatif lainnya.



Gambar 3.3.12. Tampilan Awal DFS Shortest Route



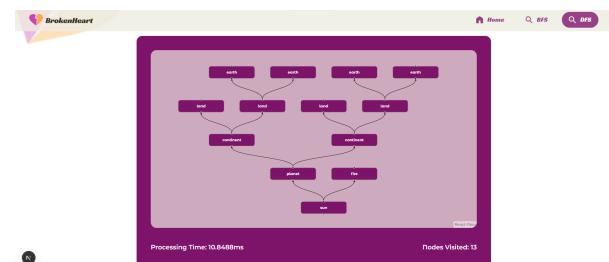
Gambar 3.3.13. Elemen Tidak Valid pada DFS Shortest Route

Gambar 3.3.12 menampilkan tampilan awal halaman DFS Shortest Route yang menyajikan antarmuka pencarian dengan algoritma Depth First Search. Halaman ini memiliki kotak pencarian dengan placeholder "Enter your element", tombol "Go!" untuk memulai pencarian, dan toggle switch yang diatur pada posisi "Shortest Route". Panel notifikasi menampilkan pesan "Please input your element!" yang memberi instruksi kepada pengguna untuk memasukkan nama elemen yang ingin dicari sebelum dapat memulai proses pencarian dengan algoritma DFS.

Gambar 3.3.13 memperlihatkan tampilan pesan kesalahan pada halaman DFS Shortest Route saat pengguna memasukkan elemen yang tidak valid. Panel notifikasi menampilkan pesan "Sorry, the element you've entered is not valid! Please recheck your input." yang mengindikasikan bahwa sistem tidak dapat menemukan elemen yang dimasukkan dalam database permainan. Tampilan ini memberikan umpan balik yang jelas kepada pengguna untuk memeriksa kembali masukan mereka dan mencoba dengan nama elemen yang valid dalam permainan Little Alchemy 2.



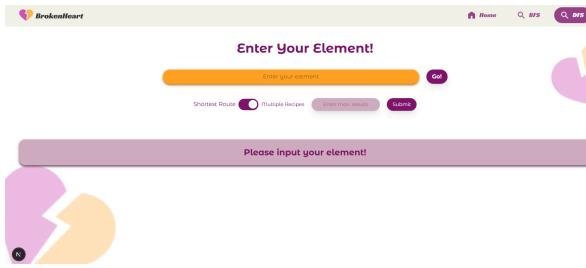
Gambar 3.3.14. Hasil Pencarian DFS Shortest Route



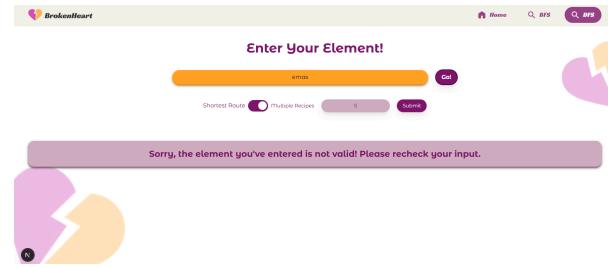
Gambar 3.3.15. Hasil Pencarian DFS Shortest Route

Gambar 3.3.14 menampilkan hasil awal pencarian DFS Shortest Route dengan panel notifikasi yang menunjukkan pesan "Here's the shortest recipe to find:". Di bawah pesan tersebut, ditampilkan ikon elemen target yang telah berhasil ditemukan oleh sistem. Tampilan ini mengindikasikan bahwa algoritma DFS telah berhasil menemukan jalur untuk membuat elemen yang dicari oleh pengguna, dan menampilkan representasi visual dari elemen target tersebut sebagai konfirmasi keberhasilan pencarian.

Gambar 3.3.15 memperlihatkan visualisasi pohon resep lengkap hasil pencarian DFS Shortest Route. Panel visualisasi menampilkan struktur hierarkis yang lebih kompleks dibandingkan dengan visualisasi BFS, menunjukkan jalur pencarian mendalam yang dilakukan algoritma DFS untuk menemukan cara membuat elemen target dari elemen-elemen dasar. Bagian bawah panel menyajikan informasi performa seperti "Processing Time" yang menunjukkan durasi pemrosesan pencarian dan "Nodes Visited: 13" yang mengindikasikan jumlah node yang dikunjungi selama proses pencarian DFS. Visualisasi ini memungkinkan pengguna memahami jalur pembuatan elemen yang ditemukan melalui algoritma Depth First Search.



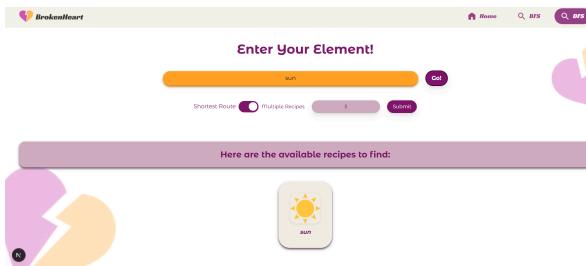
Gambar 3.3.16. Tampilan Awal DFS Multiple Recipe



Gambar 3.3.17. Elemen Tidak Valid pada DFS Multiple Recipe

Gambar 3.3.16 menampilkan tampilan awal halaman DFS Multiple Recipe dengan toggle switch yang diaktifkan pada posisi "Multiple Recipes". Halaman ini menyediakan kotak pencarian, tombol "Go!", serta kolom tambahan untuk memasukkan jumlah maksimal hasil dan tombol "Submit" untuk mengonfirmasi pencarian. Panel notifikasi menampilkan pesan "Please input your element!" yang memberi instruksi kepada pengguna untuk memasukkan nama elemen yang ingin dicari sebelum dapat melanjutkan proses pencarian beberapa resep dengan algoritma DFS.

Gambar 3.3.17 memperlihatkan tampilan pesan kesalahan pada halaman DFS Multiple Recipe ketika pengguna memasukkan elemen yang tidak valid. Panel notifikasi menampilkan pesan "Sorry, the element you've entered is not valid! Please recheck your input." yang mengindikasikan bahwa sistem tidak dapat menemukan elemen yang dimasukkan dalam database permainan. Tampilan ini masih berada dalam mode "Multiple Recipes" dengan kotak input jumlah maksimal hasil dan tombol "Submit" terlihat di bawah toggle switch. Pesan kesalahan ini memberikan umpan balik yang jelas kepada pengguna untuk mencoba kembali dengan nama elemen yang valid.



Gambar 3.3.18. Daftar Kombinasi DFS Multiple Recipe



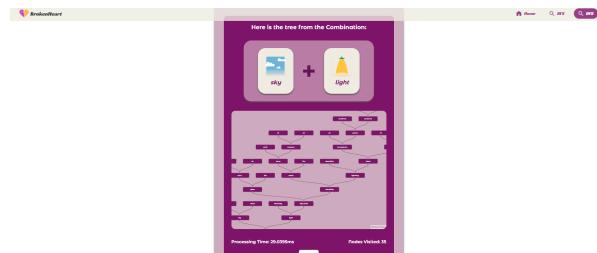
Gambar 3.3.19. Daftar Kombinasi DFS Multiple Route

Gambar 3.3.18 menampilkan hasil pencarian awal DFS Multiple Recipe dengan pesan "Here are the available recipes to find:". Di bawah pesan tersebut, ditampilkan ikon elemen target yang telah dipilih oleh pengguna untuk dicari kombinasi resepnya. Tampilan ini menandakan tahap awal dari mode multiple recipe menggunakan algoritma DFS, di mana sistem telah berhasil mengidentifikasi elemen target dan akan menyajikan berbagai kombinasi resep yang dapat digunakan untuk membuatnya.

Gambar 3.3.19 memperlihatkan daftar kombinasi lengkap hasil pencarian DFS Multiple Route. Panel ini menampilkan beberapa pasangan kombinasi elemen yang dapat digunakan untuk membuat elemen target. Setiap kombinasi ditampilkan dengan memberikan representasi visual yang jelas dari pilihan-pilihan resep yang tersedia. Tampilan ini memungkinkan pengguna untuk memilih salah satu kombinasi dengan mengkliknya untuk melihat visualisasi pohon resep lengkap dari kombinasi tersebut, sehingga pengguna dapat menjelajahi berbagai cara alternatif untuk membuat elemen target menggunakan algoritma DFS.



Gambar 3.3.20. Hasil Pencarian DFS Multiple Recipe



Gambar 3.3.21. Hasil Pencarian DFS Multiple Recipe

Gambar 3.3.20 menampilkan hasil pencarian DFS Multiple Recipe yang menunjukkan kombinasi elemen yang telah dipilih oleh pengguna. Panel atas menampilkan pesan "Here are the available recipes to find:" dengan ikon elemen target yang dicari. Di bagian bawah terdapat panel dengan label "Here is the tree from the Combination:" yang menampilkan pasangan elemen dasar yang dapat dikombinasikan untuk membuat elemen target.

Gambar 3.3.21 memperlihatkan visualisasi pohon resep lengkap dari kombinasi elemen yang dipilih pengguna dalam mode DFS Multiple Recipe. Panel visualisasi menampilkan struktur hierarkis yang kompleks, menunjukkan cara pembuatan elemen target melalui berbagai tingkatan kombinasi elemen, mulai dari elemen-elemen dasar hingga elemen target. Bagian bawah panel menampilkan informasi "Processing Time" yang menunjukkan waktu pemrosesan pencarian dan "Nodes Visited" yang mengindikasikan jumlah node yang dikunjungi selama proses pencarian. Tombol "Back" di bagian bawah memungkinkan pengguna kembali ke daftar kombinasi untuk memilih jalur alternatif lainnya.

### 3.4 Contoh Ilustrasi Kasus

Ketika pengguna telah memasuki page salah satu *search algorithm* dan memasukkan nama elemen, contohnya "stone", serta memilih opsi *shortest route/multiple routes*, frontend mengirim POST request ke backend (yang berjalan pada port 8080). Setelah itu, backend menerima request tersebut dan

memprosesnya menggunakan data resep hasil *scraping*, informasi tier dari setiap elemen, serta gambar (*image*) dari setiap elemen untuk visualisasi.

#### 3.4.1. BFS Shortest Route

Frontend mengirim request ke backend untuk mencari *shortest path* untuk elemen “stone”. Kemudian, backend memproses request ini dengan *load* data resep dari file JSON, *run* algoritma BFS dengan “stone” sebagai target, menemukan bahwa “stone” bisa dibuat dengan kombinasi “earth” + “pressure” atau “lava” + “air”. Lalu, dipilih jalur (*path*) dengan nilai tier gabungan terendah serta dihitung waktu yang diperlukan untuk proses *search*. Setelah itu, backend mengembalikan struktur pohon yang mempresentasikan jalur resep beserta waktu yang diperlukan dan jumlah *node*. Frontend memvisualisasikan pohon ini menggunakan React Flow dan menampilkannya kepada pengguna.

#### 3.4.2. BFS Multiple Recipes

Pengguna memasukkan “stone” ke dalam kotak pencarian, mengaktifkan mode “Multiple Recipes”, dan memasukkan jumlah maksimum hasil yang ingin mereka lihat. Frontend mengirim request ke backend untuk mencari berbagai cara membuat “stone” menggunakan algoritma BFS. Request tersebut diterima backend lalu ia mengidentifikasi semua kombinasi elemen yang mungkin untuk membuat “stone”. Setelah itu, ia membatasi jumlah kombinasi dengan angka maksimum yang dimasukkan pengguna. Untuk setiap kombinasi, struktur pohon dihasilkan menggunakan algoritma BFS dimulai dengan “stone”. Setiap pohon merepresentasikan jalur terpendek (*shortest path*) untuk kombinasi terkait.

Backend mengembalikan array kombinasi resep dengan jumlah yang sudah dibatasi, pohon *shortest path* untuk setiap kombinasi, serta informasi waktu eksekusi dan jumlah *node*. Frontend kemudian menampilkan semua kombinasi yang diberikan backend, dan pengguna dapat memilih salah satu kombinasi untuk melihat pohon resep lengkap dari kombinasi tersebut. Setelah itu, pengguna dapat melihat pohon resep yang merepresentasikan *shortest path* menuju elemen tersebut dengan kombinasi yang dipilih, serta waktu eksekusi dan jumlah *node*-nya.

#### 3.4.3. DFS Shortest Route

Frontend mengirim request ke backend untuk mencari shortest path untuk membuat elemen “stone” menggunakan algoritma DFS. Backend kemudian menjalankan algoritma DFS untuk menemukan satu jalur lengkap dari “stone” hingga elemen dasar. Algoritma ini menelusuri suatu jalur secara mendalam terlebih dahulu sebelum mencoba alternatif lain. Selama proses ini, backend juga mencatat waktu eksekusi pencarian dan jumlah *node* dari pohon. Setelah jalur ditemukan, backend mengembalikan struktur pohon dari jalur tersebut beserta informasi waktu pencarian dan jumlah *node*. Frontend kemudian memvisualisasikan pohon resep ini menggunakan React Flow dan menampilkannya ke pengguna, beserta dengan informasi waktu pencarian dan jumlah *node*.

#### **3.4.4. DFS Multiple Recipes**

Pengguna memasukkan “stone” ke dalam kotak pencarian, mengaktifkan mode “Multiple Recipes”, dan memasukkan jumlah maksimum hasil yang ingin mereka lihat. Frontend kemudian mengirim request ke backend untuk mencari berbagai cara membuat “stone” menggunakan algoritma DFS. Backend memproses permintaan ini dengan menjalankan algoritma yang dirancang untuk memproses setiap kombinasi resep berdasarkan indeks. Untuk setiap kombinasi, backend menggunakan goroutine untuk menjelajahi komponen kiri dan kanan secara paralel, serta menerapkan sistem mutex agar tidak terjadi konflik saat menandai elemen yang sudah dikunjungi. Algoritma ini akan mengembalikan jalur lengkap pertama yang ditemukan untuk tiap kombinasi.

Setelah proses pencarian selesai, backend menghasilkan beberapa struktur pohon untuk setiap kombinasi resep hingga batas maksimum yang ditentukan pengguna. Frontend kemudian menampilkan semua kombinasi resep (yang telah dibatasi) yang diterima dari backend dalam bentuk daftar yang bisa diklik. Ketika pengguna memilih salah satu kombinasi, frontend akan menampilkan pohon lengkap yang menunjukkan bagaimana elemen “stone” dapat dibuat dari jalur tersebut serta waktu eksekusi *search* dan jumlah *node* pohon.

## BAB 4 : Implementasi dan Pengujian

### 4.1 Implementasi

#### 4.1.1 Overview

Implementasi program ini menggunakan Go Language, dengan dibagi menjadi *front end* dan *back end*. *Back end* program ini dijabarkan sebagai berikut.

1. **Data scraping**, terdiri atas *data-scraping.go* (membaca jenis-jenis elemen dan kombinasi resep dari website Little Alchemy 2 dan menyimpannya dalam *recipes.json*) dan *tier-scraping.go* (menyimpan data setiap bahan dengan *tier*-nya).
2. **Logic**, terdiri atas *read-json.go*, *general-logic.go* (deklarasi struktur data), *bfs.go*, *dfs.go*, dan *helper-logic.go* (berisi fungsi-fungsi *helper*).

#### 4.1.2 Data Scraping

Secara umum, proses *data-scraping* terdiri atas dua program utama, yaitu *data-scraping* dan *tier-scraping* yang membaca setiap elemen dalam website beserta *tier* dari setiap elemen tersebut, berikut merupakan implementasinya.

1. *data-scraping* berisi algoritma pengambilan data untuk setiap elemen di website, elemen-elemen tersebut disimpan di *recipe.json* dalam format sebagai berikut.

```
{  
    "element": <nama-elemen>,  
    "components": [  
        [  
            <resep-1>  
        ],  
        [  
            <resep-2>  
        ]  
        Dan seterusnya.  
    ]  
}
```

Berikut merupakan algoritma yang membentuk *data-scraping*.

#### MythAndMonstersElements() → []string

Mengambil seluruh elemen *Myth and Monsters* yang pada akhirnya akan dihapus dari list, karena bukan termasuk elemen yang harus di-scrape.

```
func MythAndMonstersElements() ([]string, error) {  
    url := "https://little-alchemy.fandom.com/wiki/Category:Myths_and_Monsters"  
    client := &http.Client{Timeout: 30 * time.Second}
```

```

res, err := client.Get(url)
if err != nil {
    return nil, err
}
defer res.Body.Close()
if res.StatusCode != 200 {
    return nil, fmt.Errorf("bad status: %d", res.StatusCode)
}
doc, err := goquery.NewDocumentFromReader(res.Body)
if err != nil {
    return nil, err
}
var elements []string
doc.Find(".category-page__member-link").Each(func(i int, s *goquery.Selection) {
    name := strings.TrimSpace(s.Text())
    if name != "" {
        elements = append(elements, name)
    }
})
return elements, nil
}

```

#### BanMythAndMonsters(elements []string) → map[string]bool

Mengembalikan map untuk setiap elemen *myth and monsters* dengan nilai true, sehingga berguna untuk *skip* iterasi ketika *data scraping*.

```

func BanMythAndMonsters(elements []string) map[string]bool {
    banlist := make(map[string]bool)
    for _, element := range elements {
        banlist[element] = true
    }
    return banlist
}

```

#### AllElements() → []Element

Melakukan *scraping* untuk setiap elemen pada *website*, kecuali untuk *myth and monsters*, memanggil dua fungsi di atas untuk membentuk *map* banList. Apabila *keyword* berada dalam banList, *keyword* tersebut tidak akan di-*append* ke hasil yang akan dikembalikan.

```

func AllElements() ([]Element, error) {
    URL := "https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"
    base := "https://little-alchemy.fandom.com"

    client := &http.Client{Timeout: 30 * time.Second}
    res, err := client.Get(URL)
    if err != nil {
        return nil, err
    }
}

```

```

    defer res.Body.Close()

    if res.StatusCode != 200 {
        return nil, fmt.Errorf("bad status: %d", res.StatusCode)
    }

    doc, err := goquery.NewDocumentFromReader(res.Body)
    if err != nil {
        return nil, err
    }

    var elements []Element
    seen := make(map[string]bool)

    myths, _ := MythAndMonstersElements()
    banlist := BanMythAndMonsters(myths)

    doc.Find("a").Each(func(i int, s *goquery.Selection) {
        href, exists := s.Attr("href")
        title, titleExists := s.Attr("title")

        if exists && titleExists && !strings.Contains(href, ":") {
            if !seen[title] && !banlist[title] {
                seen[title] = true
                elements = append(elements, Element{Name: title, URL:
base + href})
            }
        }
    })
    return elements, nil
}

```

### **ElementPage(element Element, validElements map[string]bool → ([][]string)**

Mencari resep yang membentuk elemen. Pada kode ini, diberi Timeout 30 second (dapat diganti apabila diperlukan lebih lama). Resep untuk lebih dari satu elemen memerlukan pemanggilan fungsi ini berulang kali.

```

func ElementPage(element Element, validElements map[string]bool) ([][]string, error) {
    client := &http.Client{Timeout: 30 * time.Second}
    res, err := client.Get(element.URL)
    if err != nil {
        return nil, err
    }
    defer res.Body.Close()

    if res.StatusCode != 200 {
        return nil, fmt.Errorf("bad status: %d", res.StatusCode)
    }

    doc, err := goquery.NewDocumentFromReader(res.Body)

```

```

        if err != nil {
            return nil, err
        }

        var combinations [][]string
        found := false

        doc.Find("h3").EachWithBreak(func(i int, selection *goquery.Selection) bool {
            if strings.Contains(selection.Find(".mw-headline").Text(), "Little
Alchemy 2") {
                found = true
                ul := selection.NextFiltered("ul").First()
                if ul != nil {
                    // Web structure T_T.
                    ul.Find("li").Each(func(i int, li *goquery.Selection) {
                        var combo []string
                        li.Find("a").Each(func(i int, a
*goquery.Selection) {
                            name := strings.TrimSpace(a.Text())
                            if name != "" {
                                combo = append(combo, name)
                            }
                        })
                        if len(combo) >= 2 && validElements[combo[0]] &&
validElements[combo[1]] {
                            // Intinya nambahin di sini.
                            combinations = append(combinations,
combo[:2])
                        }
                    })
                }
                return false
            }
            return true
        })

        if !found {
            fmt.Printf("No 'Little Alchemy 2' section found for %s\n", element.Name)
            return nil, nil
        }

        return combinations, nil
    }
}

```

### GetAllRecipes(elements []Element) → []Recipe

Memanggil ElementPage untuk semua elemen dan memasukkan kombinasi ke struktur elemen.

```

func GetAllRecipes(elements []Element) ([]Recipe, error) {
    validElements := make(map[string]bool)
}

```

```

        for _, element := range elements {
            validElements[element.Name] = true
        }

        myths, _ := MythAndMonstersElements()
        banlist := BanMythAndMonsters(myths)
        for elem := range banlist {
            validElements[elem] = false
        }

        var allRecipes []Recipe
        for _, element := range elements {
            combinations, err := ElementPage(element, validElements)
            if err != nil {
                fmt.Printf("Error getting recipes for %s: %v\n", element.Name,
err)
                continue
            }
            if len(combinations) > 0 {
                allRecipes = append(allRecipes, Recipe{
                    Element: element.Name,
                    Components: combinations,
                })
            }
        }
        return allRecipes, nil
    }
}

```

2. *tier-scraping* berisi algoritma pengambilan data untuk setiap *tier* dari elemen. *Tier* elemen tersebut disimpan di *tiers* dengan format sebagai berikut.

```
{
    <nama-elemen>: <tier>,
    <nama-elemen>: <tier>
}
```

Berikut merupakan implementasi algoritma yang membentuk *tier-scraping*.

#### **getDirectory() → string**

Mengambil directory tempat penyimpanan file .json (*hardcode* langsung ke folder data), mengembalikan *file path*.

```

func getDirectory() string {
    _, filename, _, ok := runtime.Caller(0)
    if !ok {
        log.Fatal("Invalid file path.")
    }
    return filepath.Join(filepath.Dir(filepath.Dir(filename)), "data")
}

```

```
}
```

### **saveTier(path string, tierMap map[string]int)**

Fungsi ini membuat *file path* baru jika belum ada dan menyimpan tierMap ke bentuk format *.json*. Fungsi ini akan digunakan oleh *scrapeTier*.

```
func saveTier(path string, tierMap map[string]int) error {
    if err := os.MkdirAll(filepath.Dir(path), 0755); err != nil {
        return err
    }
    f, err := os.Create(path)
    if err != nil {
        return err
    }
    defer f.Close()
    enc := json.NewEncoder(f)
    enc.SetIndent("", "    ")
    return enc.Encode(tierMap)
}
```

### **scrapeTier() → []ElementTier**

Mengambil data tier dari setiap elemen. Terdapat *page* berbeda di mana elemen memiliki informasi *tier* (berbeda dengan *page* untuk *data scraping* awal).

```
func scrapeTier() ([]ElementTier, error) {
    const url =
"https://little-alchemy.fandom.com/wiki/Elements_%28Little_Alchemy_2%29"

    req, _ := http.NewRequest("GET", url, nil)
    req.Header.Set("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64)")
    res, err := http.DefaultClient.Do(req)
    if err != nil {
        return nil, fmt.Errorf("HTTP GET: %w", err)
    }
    defer res.Body.Close()
    if res.StatusCode != http.StatusOK {
        return nil, fmt.Errorf("Unexpected %d", res.StatusCode)
    }

    doc, err := goquery.NewDocumentFromReader(res.Body)
    if err != nil {
        return nil, fmt.Errorf("Parse HTML: %w", err)
    }

    var items []ElementTier

    tempContainer := doc.Find("div.mw-parser-output > h3")

    tempContainer.Each(func(i int, h3 *goquery.Selection) {
```

```

span := h3.Find("span.mw-headline")
hdr := strings.TrimSpace(span.Text())

id, _ := span.Attr("id")
if !strings.HasPrefix(id, "Tier_") {
    return
}

parts := strings.Fields(hdr)
if len(parts) < 2 {
    return
}
tierNumber, err := strconv.Atoi(parts[1])
if err != nil {
    return
}

sib := h3.Next()
for sib.Length() > 0 && goquery.NodeName(sib) != "table" {
    sib = sib.Next()
}
if sib.Length() == 0 {
    return
}

// TableMAXXING.
rows := sib.Find("tr")
rows.Each(func(j int, row *goquery.Selection) {
    if row.Find("th").Length() > 0 {
        return
    }
    cell := row.Find("td").First()
    a := cell.Find("a[title]").First()
    name := strings.TrimSpace(a.Text())
    href, _ := a.Attr("href")

    if name == "" || !strings.HasPrefix(href, "/wiki/") {
        return
    }

    items = append(items, ElementTier{
        Name: name,
        URL: "https://little-alchemy.fandom.com" + href,
        Tier: tierNumber,
    })
})
})

if len(items) == 0 {
    return nil
}
return items, nil

```

```
}
```

### scrapeData()

Menggabungkan *scrapeTier* dan menyimpannya ke dalam .json (beserta validasi).

```
func scrapeData() {
    items, err := scrapeTier()
    if err != nil {
        log.Fatalf("Failed to scrape: %v", err)
    }

    tierMap := make(map[string]int)
    for _, item := range items {
        tierMap[item.Name] = item.Tier
    }

    outputPath := filepath.Join(getDirectory(), "tiers.json")
    if err := saveTier(outputPath, tierMap); err != nil {
        log.Fatalf("Failed to save: %v", err)
    }

    fmt.Printf("%d elements was saved to %s.\n", len(tierMap), outputPath)
}
```

#### 4.1.3 general-logic (Data structure)

*General-logic* di sini merupakan implementasi untuk setiap enkapsulasi struktur data yang digunakan dalam penyelesaian masalah. Berikut merupakan penjabaran setiap tipe data.

##### 1. Element

```
type Element struct {
    Name      string      `json:"element"`
    Components [][]string `json:"components"`
    Tier      int
}
```

##### 2. ComponentKey

```
type ComponentKey struct {
    Component1 string
    Component2 string
}
```

##### 3. ElementContainer

```

type ElementContainer struct {
    Container map[string][]ComponentKey
    ElementTier map[string]int
    IsVisited map[string]bool
}

```

#### 4. TreeNode

```

type TreeNode struct {
    Name      string
    Left     *TreeNode
    Right    *TreeNode
}

```

#### 5. Result

```

type Result struct {
    Node *TreeNode
    VisitedCount int
}

```

### 4.1.4 Logic

1. *read-json.go* mengandung algoritma yang membaca dan memproses file JSON yang mengandung resep elemen, tier elemen, serta gambar elemen. Setelah itu, data diolah dan disusun menjadi sebuah struktur data yang memungkinkan setiap elemen dikombinasikan berdasarkan pasangan komponen yang telah ditentukan, sehingga dapat digunakan untuk pencarian resep atau visualisasi pohon elemen dalam aplikasi. Berikut merupakan algoritma yang membentuk *read-json.go*.

#### ReadJSON(recipes string, tiersFile string, imagesFile string) → []Element

Memuat dan memproses tiga file JSON yang berisi resep elemen, tingkat tier, dan gambar, menormalkan semua data menjadi huruf kecil, serta memastikan elemen dasar yang dibutuhkan ada sebelum mengembalikan array elemen yang terstruktur.

```

func ReadJSON(recipes string, tiersFile string, imagesFile string) ([]Element, error) {
    file, err := os.Open(recipes)
    if err != nil {
        return nil, fmt.Errorf("Error opening recipes file: %w", err)
    }
    defer file.Close()
    byteValue, err := ioutil.ReadAll(file)
    if err != nil {
        return nil, fmt.Errorf("Error reading recipes file: %w", err)
    }
    var elementContainer ElementContainer
    err = json.Unmarshal(byteValue, &elementContainer)
    if err != nil {
        return nil, fmt.Errorf("Error unmarshaling recipes file: %w", err)
    }
    tiers, err := readTiers(tiersFile)
    if err != nil {
        return nil, fmt.Errorf("Error reading tiers file: %w", err)
    }
    images, err := readImages(imagesFile)
    if err != nil {
        return nil, fmt.Errorf("Error reading images file: %w", err)
    }
    elementContainer.Tier = tiers
    elementContainer.Image = images
    return elementContainer.Elements, nil
}

```

```

    }
    var elements []Element
    if err := json.Unmarshal(byteValue, &elements); err != nil {
        return nil, fmt.Errorf("Error parsing recipes JSON: %w", err)
    }
    for i := range elements {
        elements[i].Name = strings.ToLower(elements[i].Name)
        for j := range elements[i].Components {
            for k := range elements[i].Components[j] {
                elements[i].Components[j][k] =
strings.ToLower(elements[i].Components[j][k])
            }
        }
    }
    file2, err := os.Open(tiersFile)
    if err != nil {
        return nil, fmt.Errorf("Error opening tiers file: %w", err)
    }
    defer file2.Close()
    byteTier, err := ioutil.ReadAll(file2)
    if err != nil {
        return nil, fmt.Errorf("Error reading tiers file: %w", err)
    }
    var tiers map[string]int
    if err := json.Unmarshal(byteTier, &tiers); err != nil {
        return nil, fmt.Errorf("Error parsing tiers JSON: %w", err)
    }
    normalizedTiers := make(map[string]int)
    for k, v := range tiers {
        normalizedTiers[strings.ToLower(k)] = v
    }
    for i := range elements {
        if tier, ok := normalizedTiers[elements[i].Name]; ok {
            elements[i].Tier = tier
        } else {
            elements[i].Tier = 0
        }
    }
    type imageEntry struct {
        Name string `json:"name"`
        Img string `json:"img"`
    }
    file3, err := os.Open(imagesFile)
    if err != nil {
        return nil, fmt.Errorf("Error opening images file: %w", err)
    }
    defer file3.Close()
    byteImage, err := ioutil.ReadAll(file3)
    if err != nil {
        return nil, fmt.Errorf("Error reading images file: %w", err)
    }
    var images []imageEntry

```

```

        if err := json.Unmarshal(byteImage, &images); err != nil {
            return nil, fmt.Errorf("Error parsing images JSON: %w", err)
        }
        normalizedImages := make(map[string]string)
        for _, img := range images {
            normalizedImages[strings.ToLower(img.Name)] = img.Img
        }
        for i := range elements {
            if img, ok := normalizedImages[elements[i].Name]; ok {
                elements[i].Image = img
            }
        }
        required := []string{"fire", "time"} // HARDCODED BOZO HAHAHAHA
        existing := make(map[string]bool)
        for _, el := range elements {
            existing[el.Name] = true
        }
        for _, name := range required {
            if !existing[name] {
                elements = append(elements, Element{
                    Name: name,
                    Components: [][]string{},
                    Tier: 0,
                })
            }
        }
        return elements, nil
    }
}

```

### **BuildElementContainer(elements []Element) → ElementContainer**

Mengubah array elemen menjadi struktur data yang terorganisir dengan memetakan nama elemen ke pasangan komponennya, tingkat tier, dan gambar, sekaligus menginisialisasi sistem pelacakan untuk algoritma pencarian.

```

func BuildElementContainer(elements []Element) ElementContainer {
    container := make(map[string][]ComponentKey)
    isVisited := make(map[string]bool)
    elementTier := make(map[string]int)
    elementImage := make(map[string]string)
    for _, el := range elements {
        for _, pair := range el.Components {
            if len(pair) == 2 {
                key := ComponentKey{
                    Component1: pair[0],
                    Component2: pair[1],
                }
                container[el.Name] = append(container[el.Name], key)
            }
        }
        isVisited[el.Name] = false
    }
}

```

```

        elementTier[el.Name] = el.Tier
        elementImage[el.Name] = el.Image
    }
    return ElementContainer{
        Container: container,
        IsVisited: isVisited,
        ElementTier: elementTier,
        ElementImage: elementImage,
    }
}

```

2. *general-logic.go* mendefinisikan struktur data untuk sistem *game*, termasuk representasi elemen, komponen-komponennya, *container* untuk melacak kombinasi, serta struktur pohon untuk memvisualisasikan hubungan antar elemen. Berikut merupakan pendefinisian type pada *general-logic*.

<b>Element</b>
Mengandung nama elemen, komponen, tier elemen, dan gambar elemen.
<pre> type Element struct {     Name      string `json:"element"`     Components [][]string `json:"components"`     Tier      int     Image     string } </pre>
<b>ComponentKey</b>
Struktur pasangan yang menyimpan dua elemen komponen yang diperlukan untuk membuat elemen lainnya.
<pre> type ComponentKey struct {     Component1 string     Component2 string } </pre>
<b>ElementContainer</b>
Mengorganisir semua elemen, resepnya, tier, gambar, dan status kunjungan untuk algoritma pencarian.
<pre> type ElementContainer struct {     Container map[string][]ComponentKey     ElementTier map[string]int     IsVisited map[string]bool     ElementImage map[string]string } </pre>

### **TreeNode**

Node pohon biner yang digunakan untuk membangun jalur resep.

```
type TreeNode struct {
    Name string
    Image string
    Left *TreeNode
    Right *TreeNode
}
```

### **SearchState**

Melacak *current node* dan elemen target selama operasi pencarian.

```
type SearchState struct {
    Node *TreeNode
    Target string
}
```

3. *bfs.go* mengandung algoritma *breadth-first search* untuk menghasilkan pohon resep yang menunjukkan cara membuat elemen target dengan menelusuri semua kemungkinan kombinasi secara melebar level demi level, menggunakan antrian untuk menjamin jalur terpendek.

### **BreadthFirstSearch(target string, container \*ElementContainer, index int) → \*TreeNode**

Membangun pohon resep untuk sebuah elemen target menggunakan penelusuran breadth-first, dengan awalnya menggunakan indeks pasangan komponen yang ditentukan, lalu memilih pasangan-pasangan berikutnya berdasarkan nilai tier gabungan terendah, sambil menghindari referensi sirkuler dan menghentikan perluasan pada elemen dasar.

```
func BreadthFirstSearch(target string, container *ElementContainer, index int)
*TreeNode {
    target = strings.ToLower(target)
    queue := list.New()
    root := &TreeNode{Name: target, Image: container.ElementImage[target]}
    queue.PushBack(root)
    first := true
    for queue.Len() > 0 {
        element := queue.Front()
        queue.Remove(element)
        parentNode := element.Value.(*TreeNode)
        pairs := container.Container[parentNode.Name]
        if len(pairs) == 0 {
            continue
        }
        i := index
        if !first {
            shortestMap := make(map[int]int)
```

```

        i = 0
        for _, pair := range container.Container[parentNode.Name] {
            // fmt.Printf("[EXPAND] %s - %s + %s\n", parentNode.Name,
pair.Component1, pair.Component2)
            t1, ok1 := container.ElementTier[pair.Component1]
            t2, ok2 := container.ElementTier[pair.Component2]
            tTarget, okT := container.ElementTier[parentNode.Name]

            if !ok1 || !ok2 || !okT {
                i++
                continue
            }
            if t1 >= tTarget || t2 >= tTarget {
                i++
                continue
            }
            shortestMap[i] = container.ElementTier[pair.Component1] +
container.ElementTier[pair.Component2]
                i++
            }
            i = minKey(shortestMap)
        }
        pair := pairs[i]
        leftName := pair.Component1
        rightName := pair.Component2

        if (leftName == parentNode.Name || rightName == parentNode.Name) {
            continue
        }

        leftNode := &TreeNode{ Name: leftName, Image:
container.ElementImage[leftName] }
        rightNode := &TreeNode{ Name: rightName, Image:
container.ElementImage[rightName] }
        if !isBaseElement(leftName) {
            parentNode.Left = leftNode
            queue.PushBack(leftNode)
        } else {
            parentNode.Left = &TreeNode{Name: leftName, Image:
container.ElementImage[leftName]}
        }

        if !isBaseElement(rightName) {
            parentNode.Right = rightNode
            queue.PushBack(rightNode)
        } else {
            parentNode.Right = &TreeNode{Name: rightName, Image:
container.ElementImage[rightName]}
        }
        first = false
    }
    return root
}

```

}

4. `dfs.go` mengandung algoritma *depth-first search* untuk menghasilkan pohon resep yang menggali jalur pembuatan elemen target secara mendalam hingga menemukan elemen dasar sebelum beralih ke jalur alternatif, menggunakan rekursi dan penandaan untuk mencegah siklus.

**depthFirstSearch(target string, container \*ElementContainer) → \*TreeNode**

Membangun pohon resep secara rekursif untuk sebuah elemen target dengan memeriksa elemen dasar dan siklus, memfilter pasangan komponen berdasarkan tier, serta mengeksplorasi setiap jalur kombinasi yang valid hingga menemukan solusi lengkap atau semua kemungkinan setelah dieksplorasi.

```

        }
        if t1 > tTarget || t2 > tTarget {
            continue
        }
        left := depthFirstSearch(pair.Component1, container)
        right := depthFirstSearch(pair.Component2, container)
        container.IsVisited[target] = false
        if left != nil && right != nil {
            return &TreeNode{
                Name: target,
                Image: container.ElementImage[target],
                Left: left,
                Right: right,
            }
        }
    }
    return nil
}

```

5. *helper-logic.go* mengandung metode-metode *helper* yang mengidentifikasi elemen dasar, menstandarkan pasangan komponen, mencari nilai minimum dalam map, dan mengakses informasi resep dari struktur container.

#### **isBaseElement(name string) → bool**

Menentukan apakah suatu elemen merupakan blok penyusun dasar dengan memeriksa apakah namanya cocok dengan salah satu dari lima elemen dasar yang telah ditentukan.

```

func isBaseElement(name string) bool {
    switch name {
    case "air", "water", "fire", "earth", "time":
        return true
    default:
        return false
    }
}

```

#### **normalizeKey(a, b string) → ComponentKey**

Membuat ComponentKey yang terstandarisasi dari dua nama elemen dengan mengurutkannya secara alfabet untuk memastikan pencarian resep yang konsisten, sehingga tidak ada resep bolak-balik yang tersimpan (tidak efisien).

```

func normalizeKey(a, b string) ComponentKey {
    if a < b {
        return ComponentKey{a, b}
    }
    return ComponentKey{b, a}
}

```

### **minKey(m map[int]int) → int**

Mengidentifikasi key dengan nilai terkecil dalam sebuah map, digunakan untuk menemukan kombinasi komponen dengan total nilai tier terendah.

```
func minKey(m map[int]int) int {
    minKey := -1
    minValue := int(^uint(0) >> 1)
    for k, v := range m {
        if v < minValue {
            minValue = v
            minKey = k
        }
    }
    return minKey
}
```

### **getLength(container \*ElementContainer, element string) → int**

Mengembalikan jumlah pasangan komponen berbeda yang dapat digunakan untuk membuat suatu elemen tertentu.

```
func getLength(container *ElementContainer, element string) int {
    return len(container.Container[element])
}
```

### **getRecipe(container \*ElementContainer, element string, many int) → []ComponentKey**

Mengambil sejumlah resep pasangan komponen untuk sebuah elemen dari container, digunakan untuk mencari multiple solutions.

```
func getRecipe(container *ElementContainer, element string, many int) []ComponentKey {
    returnValue := make([]ComponentKey, 0)
    for i := 0; i < many; i++ {
        returnValue = append(returnValue, container.Container[element][i])
    }
    return returnValue
}
```

## **4.1.5 API (handler.go)**

### **1. Type**

```
type SearchRequest struct {
    Target string `json:"target"`
    Index  int    `json:"index"`
}

type Recipe struct {
    Element      string     `json:"element"`
    Components  [[[string]]] `json:"components"`
}
```

```

}

type Element struct {
    Name string
    URL  string
}

```

## 2. Functions

### **enableCors(w http.ResponseWriter)**

Mengatur header CORS agar server dapat menerima permintaan dari domain mana pun menggunakan metode POST, GET, dan OPTIONS.

```

func enableCors(w http.ResponseWriter) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "POST, GET, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
}

```

### **loadElementContainerFromFiles() → (\*logic.ElementContainer, error)**

Membaca file JSON yang berisi data resep, tier, dan gambar, lalu membangun dan mengembalikan container elemen.

```

func loadElementContainerFromFiles() (*logic.ElementContainer, error) {
    recipesPath := filepath.Join(".", "data", "recipes.json")
    tiersPath := filepath.Join(".", "data", "tiers.json")
    imagesPath := filepath.Join(".", "data", "images.json")

    elements, err := logic.ReadJSON(recipesPath, tiersPath, imagesPath)
    if err != nil {
        return nil, err
    }
    container := logic.BuildElementContainer(elements)
    return &container, nil
}

```

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

Menangani permintaan POST untuk mencari rute tercepat ke target menggunakan algoritma BFS dan mengembalikan hasilnya dalam format JSON.

```

func shortestbfsHandler(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    enableCors(w)
    if r.Method == http.MethodOptions {
        return
    }
    if r.Method != http.MethodPost {
        http.Error(w, "Only POST method is allowed",
        http.StatusMethodNotAllowed)
        return
    }
}

```

```

    }

    var req SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, "Invalid JSON: "+err.Error(), http.StatusBadRequest)
        return
    }

    container, err := loadElementContainerFromFiles()
    if err != nil {
        http.Error(w, "Error loading data: "+err.Error(),
http.StatusInternalServerError)
        return
    }

    root := logic.ShortestBreadthFirstSearch(req.Target, container)
    duration := time.Since(start)
    fmt.Printf("Target: %s, Result: %+v, Duration: %s\n", req.Target, root,
duration)

    response := map[string]interface{}{
        "data":           root,
        "executionTime": duration.String(),
    }
}

w.Header().Set("Content-Type", "application/json")
json.NewEncoder(w).Encode(response)
}

```

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

Menangani permintaan POST untuk mencari rute tercepat ke target menggunakan algoritma DFS dan mengembalikan hasilnya dalam format JSON.

```

func shortestdfsHandler(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    enableCors(w)
    if r.Method == http.MethodOptions {
        return
    }
    if r.Method != http.MethodPost {
        http.Error(w, "Only POST method is allowed",
http.StatusMethodNotAllowed)
        return
    }

    var req SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, "Invalid JSON: "+err.Error(), http.StatusBadRequest)
        return
    }

    container, err := loadElementContainerFromFiles()
    if err != nil {
        http.Error(w, "Error loading data: "+err.Error(),
http.StatusInternalServerError)
        return
    }
}

```

```

        var visitedCount = new(int)
        *visitedCount = 0
        root := logic.ShortestDepthFirstSearch(req.Target, container, visitedCount)
        duration := time.Since(start)

        response := map[string]interface{}{
            "data":           root,
            "executionTime": duration.String(),
        }

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(response)
    }
}

```

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

Menangani permintaan POST untuk mencari semua jalur ke target menggunakan variasi algoritma BFS dan mengembalikan tree serta resepnya.

```

func multiplebfsHandler(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    enableCors(w)
    if r.Method == http.MethodOptions {
        return
    }
    if r.Method != http.MethodPost {
        http.Error(w, "Only POST method is allowed",
http.StatusMethodNotAllowed)
        return
    }

    var req SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, "Invalid JSON: "+err.Error(), http.StatusBadRequest)
        return
    }

    container, err := loadElementContainerFromFiles()
    if err != nil {
        http.Error(w, "Error loading data: "+err.Error(),
http.StatusInternalServerError)
        return
    }

    loopCount := logic.GetLength(container, req.Target)
    recipes := logic.GetRecipe(container, req.Target, loopCount)
    var trees []interface{}

    for i := 0; i < loopCount; i++ {
        tree := logic.BreadthFirstSearch(req.Target, container, i)
        trees = append(trees, tree)
    }
    duration := time.Since(start)

    response := map[string]interface{}{
        "trees":   trees,
        "recipes": recipes,
    }
}

```

```

        "executionTime": duration.String(),
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(response)
}

```

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

Menangani permintaan POST untuk mencari semua jalur ke target menggunakan variasi algoritma DFS dan mengembalikan tree serta resepnya.

```

func multipledfsHandler(w http.ResponseWriter, r *http.Request) {
    start := time.Now()
    enableCors(w)
    if r.Method == http.MethodOptions {
        return
    }
    if r.Method != http.MethodPost {
        http.Error(w, "Only POST method is allowed",
http.StatusMethodNotAllowed)
        return
    }

    var req SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, "Invalid JSON: "+err.Error(), http.StatusBadRequest)
        return
    }

    container, err := loadElementContainerFromFiles()
    if err != nil {
        http.Error(w, "Error loading data: "+err.Error(),
http.StatusInternalServerError)
        return
    }

    loopCount := logic.GetLength(container, req.Target)
    recipes := logic.GetRecipe(container, req.Target, loopCount)
    var trees []interface{}

    for i := 0; i < loopCount; i++ {
        var visitedCount = new(int)
        *visitedCount = 0
        tree := logic.FirstDepthFirstSearch(req.Target, container, i,
visitedCount)
        trees = append(trees, tree)
    }
    duration := time.Since(start)

    response := map[string]interface{}{
        "trees": trees,
        "recipes": recipes,
        "executionTime": duration.String(),
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(response)
}

```

## main()

Menjalankan proses pengambilan data, mengatur endpoint API, dan memulai server HTTP pada port 8080.

```
func main() {
    scrapeAllData()
    http.HandleFunc("/api/bfs", shortestBFSHandler)
    http.HandleFunc("/api/dfs", shortestDFSHandler)
    http.HandleFunc("/api/bfsmultiple", multipleBFSHandler)
    http.HandleFunc("/api/dfsmultiple", multipleDFSHandler)

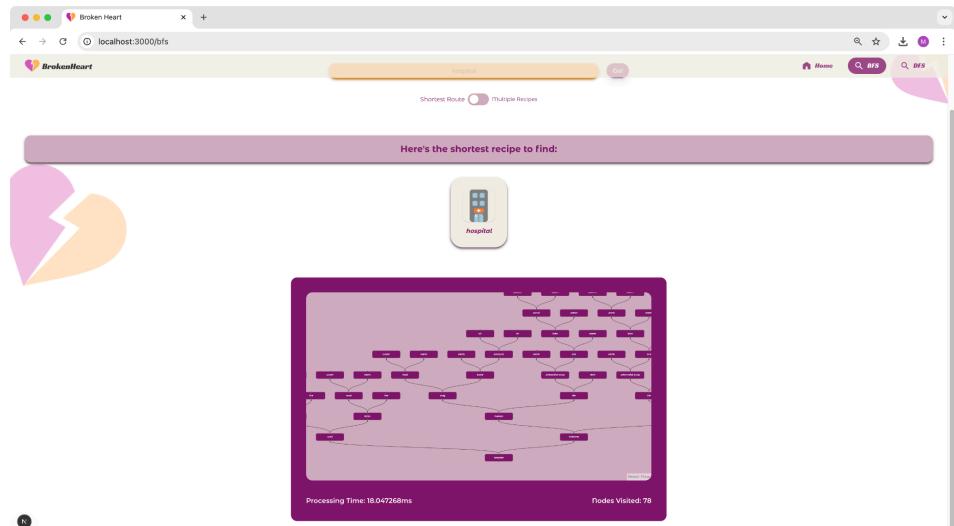
    fmt.Println("Server running on :8080")
    logErr := http.ListenAndServe(":8080", nil)
    if logErr != nil {
        fmt.Println("Server error:", logErr)
    }
}
```

## 4.2 Pengujian

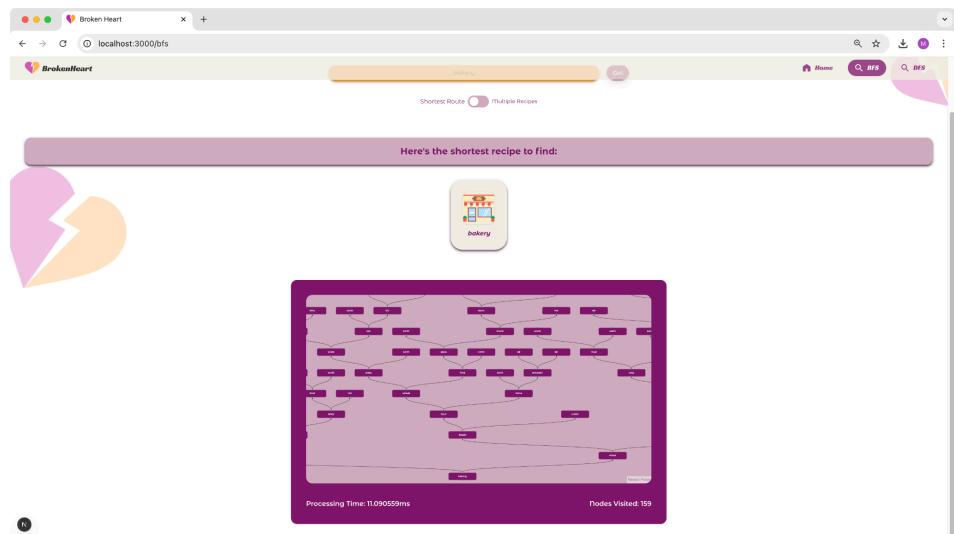
### 4.1.1 BFS - Shortest Path

Elemen Uji	Hasil
The One Ring	

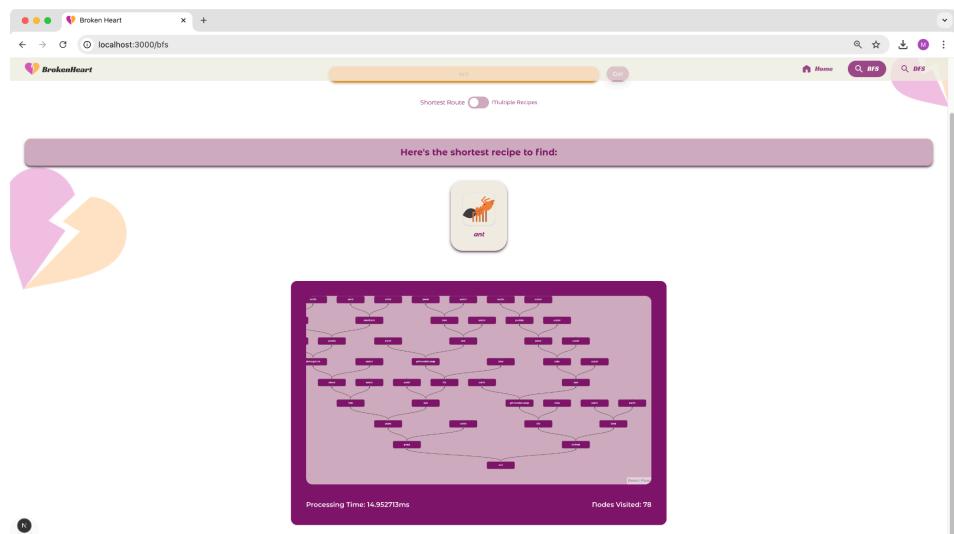
## Hospital



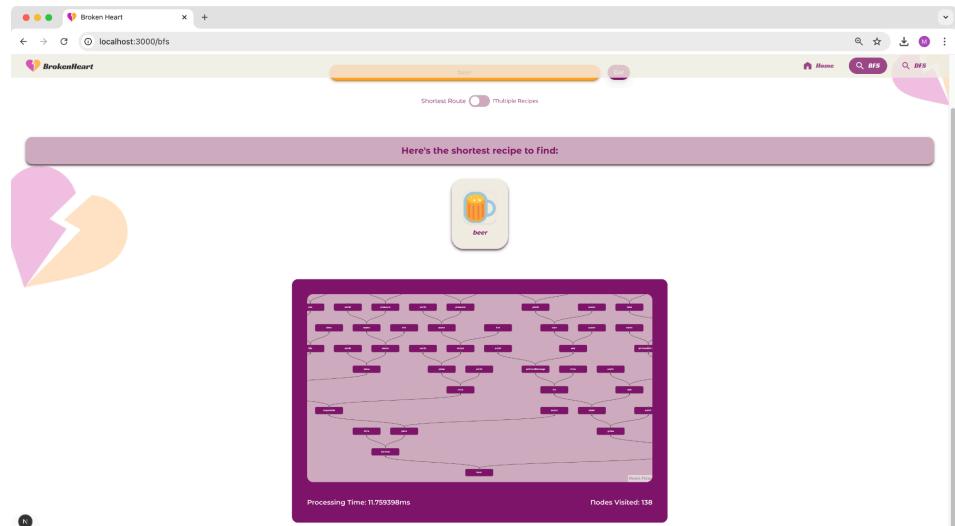
## Bakery



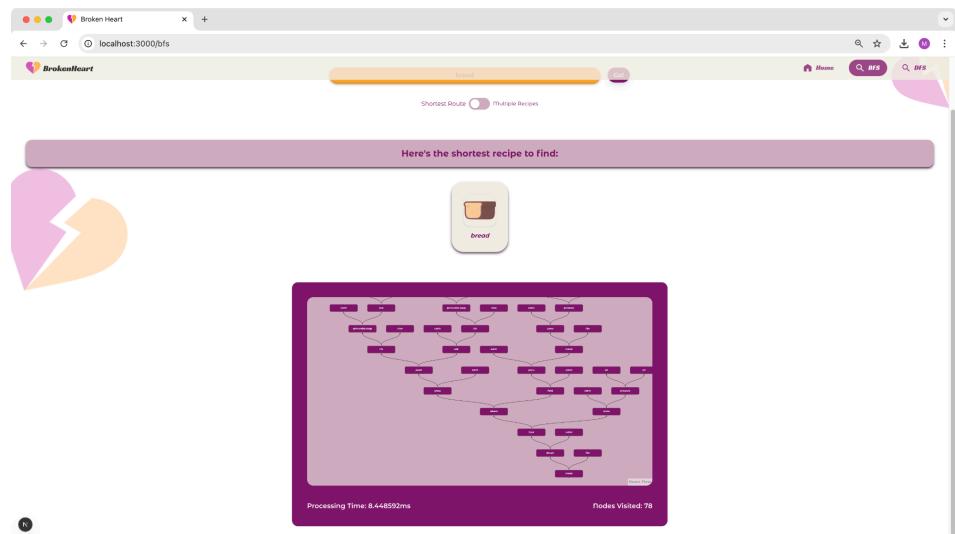
## Ant



Beer



Bread



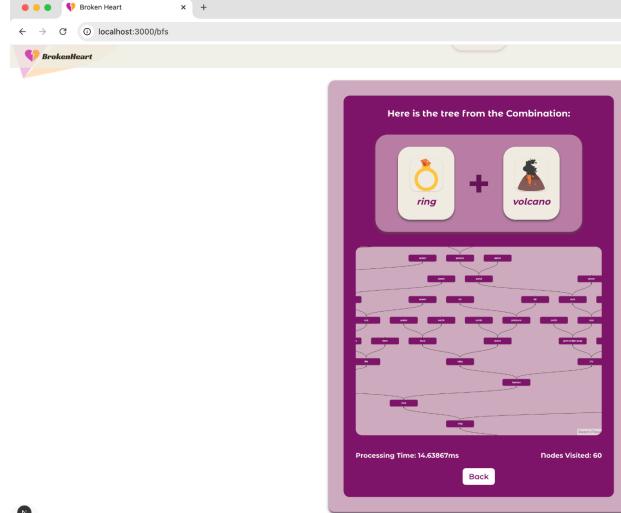
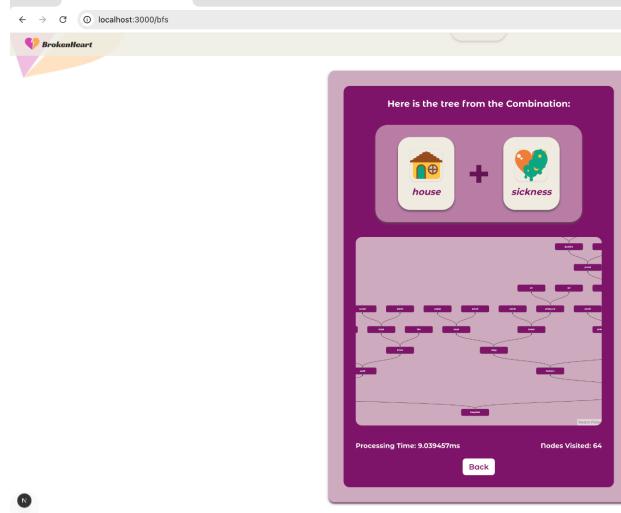
Spaghetti



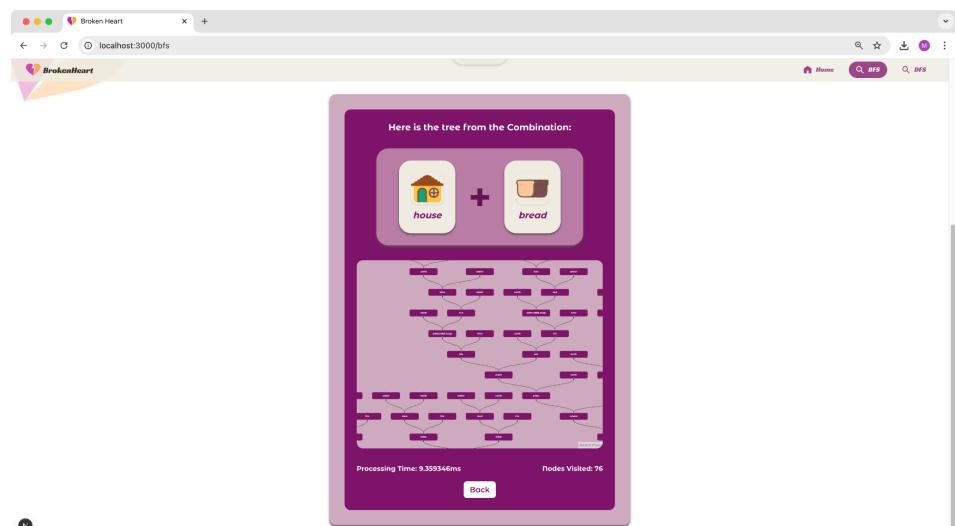
Pengujian di atas menunjukkan bahwa BFS *shortest path* mampu menunjukkan jalan terpendek dari elemen dasar ke elemen target. Selain itu, BFS mampu menjamin bahwa rute yang digunakan adalah rute terpendek, karena semua edges tidak memiliki cost yang berbeda, dan sudah dipilih rute dengan total tier terkecil, sehingga dapat dipastikan algoritma menggunakan rute tercepat untuk mencapai tier 0 element (elemen dasar).

#### 4.1.2 BFS - Multiple Result

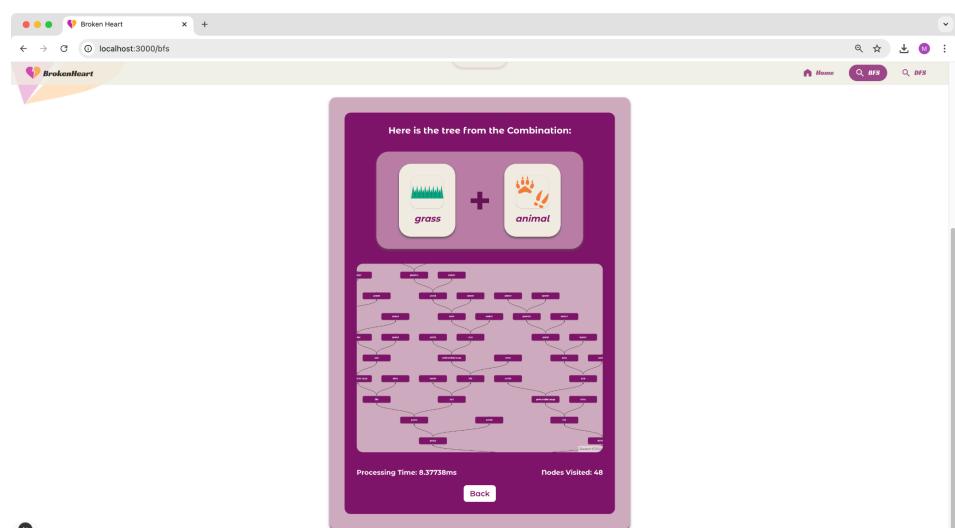
Untuk semua elemen, dipilih **pilihan pertama** dengan **max solution = 2**.

Elemen Uji	Hasil
The One Ring	
Hospital	

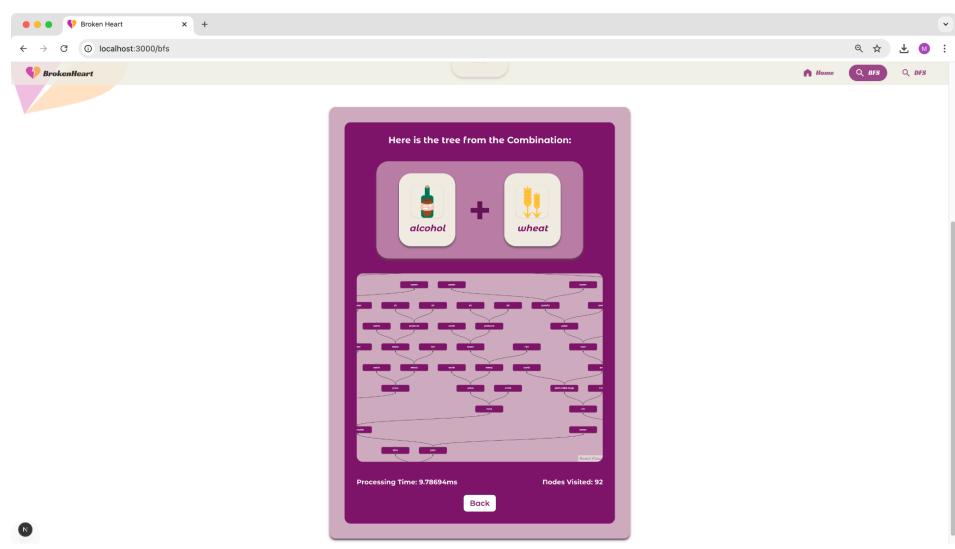
## Bakery

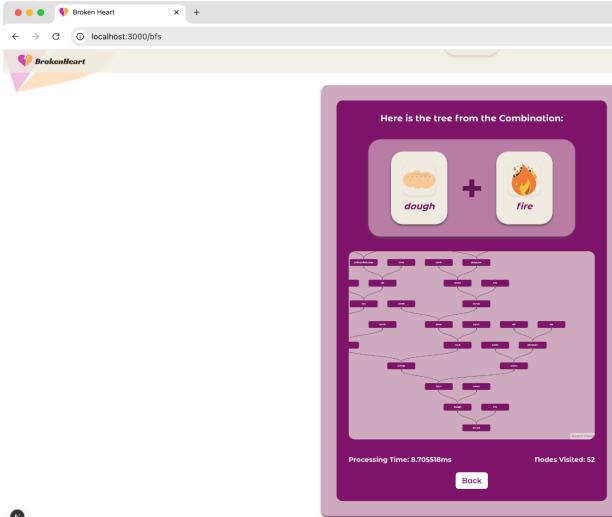
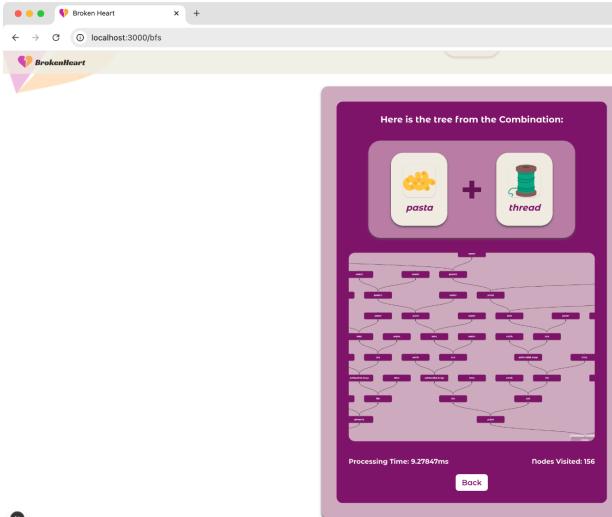


## Ant



## Beer



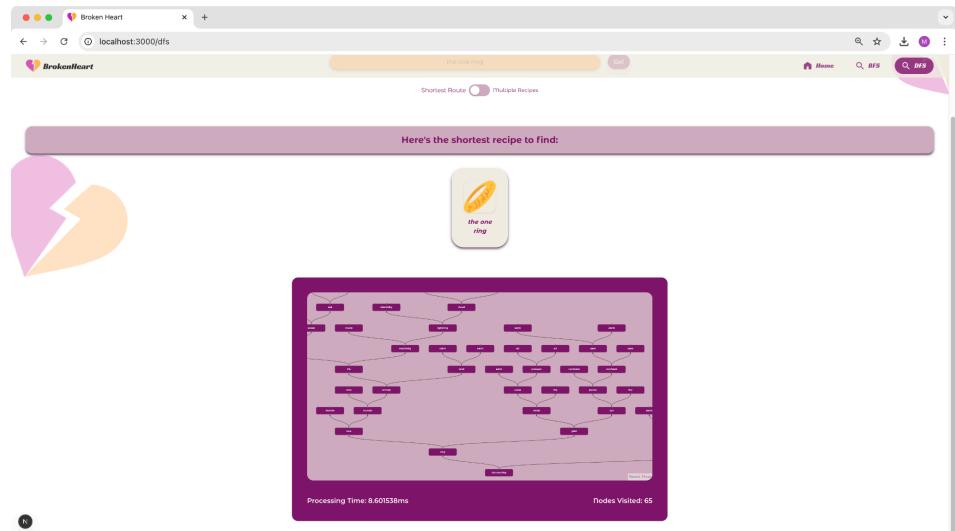
Bread	
Spaghetti	

Dalam mode BFS Multiple Result, program berhasil menampilkan kepada pengguna solusi sesuai dengan jumlah yang diminta, tetapi juga dibatasi dengan berapa yang mampu diberikan oleh program. User dapat memilih dengan bebas resep yang mana yang ingin dilihat treenya, kemudian program akan menampilkan tree yang sesuai. Program telah berjalan dengan baik.

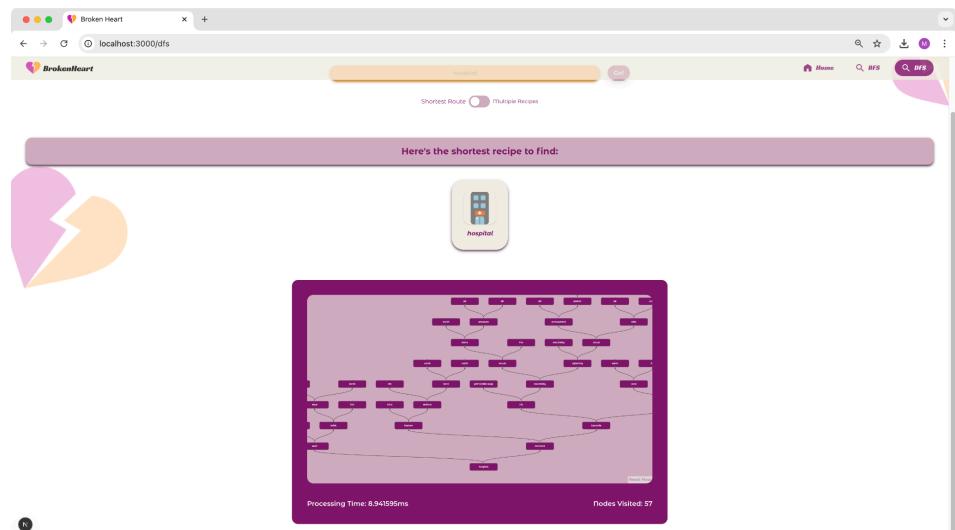
#### 4.1.3 DFS - Shortest Path

Elemen Uji	Hasil
------------	-------

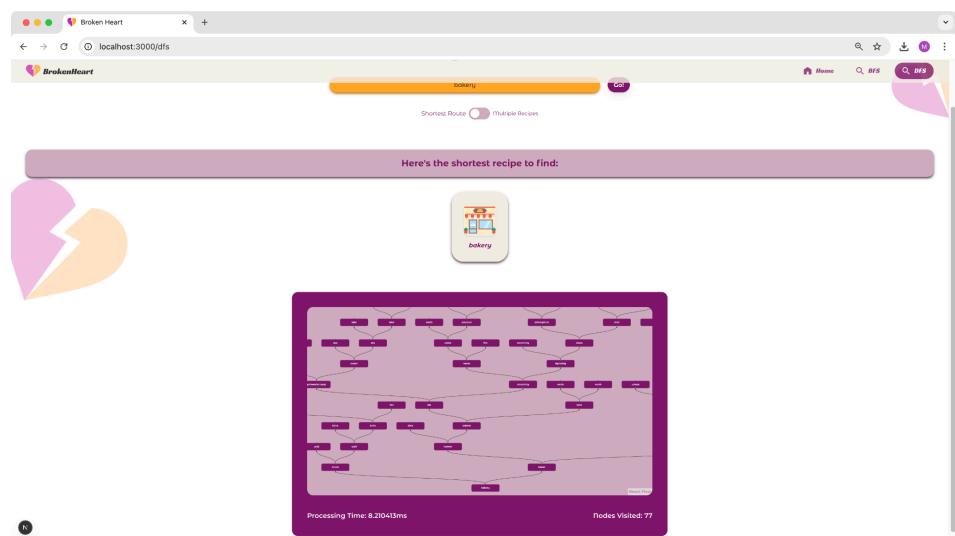
## The One Ring



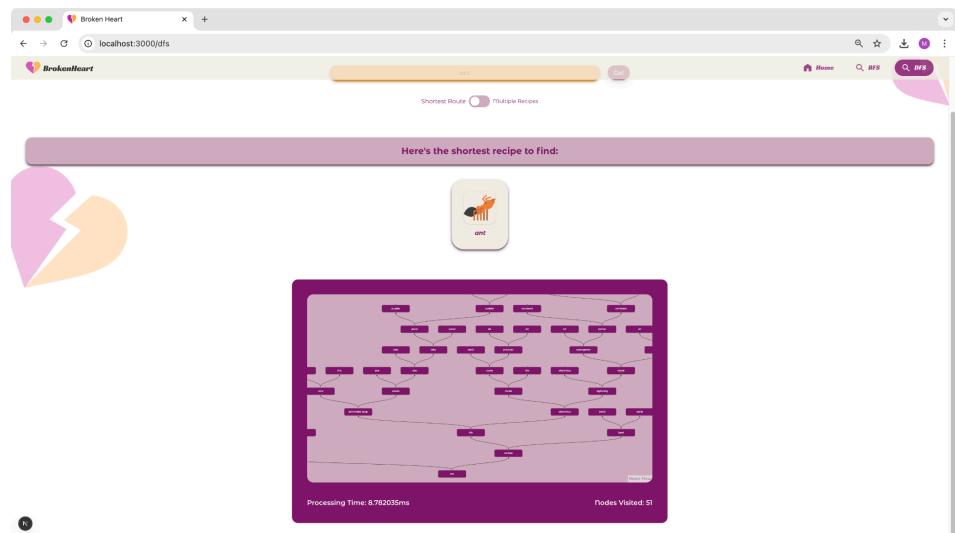
## Hospital



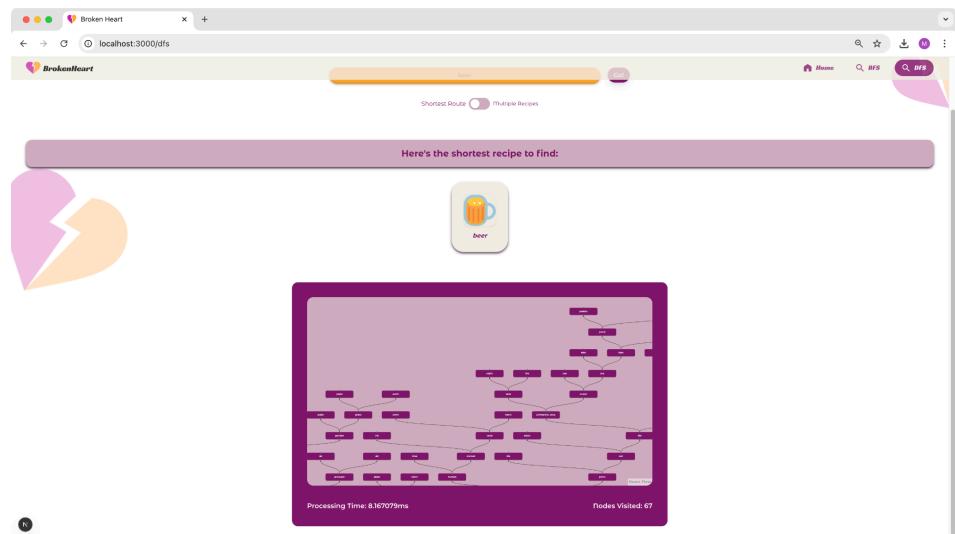
## Bakery



Ant



Beer



Bread



Spaghetti	<p>The screenshot shows a web browser window titled "Broken Heart" with the URL "localhost:3000/dfs". The search bar contains the text "spaghetti". A pink banner at the top says "Here's the shortest recipe to find:". Below it is a large icon of a broken heart. To the right is a small icon of a fork and knife labeled "spaghetti". The main area displays a complex search tree with many nodes and connections. At the bottom, it says "Processing Time: 8.759287ms" and "Nodes Visited: 109".</p>
-----------	---

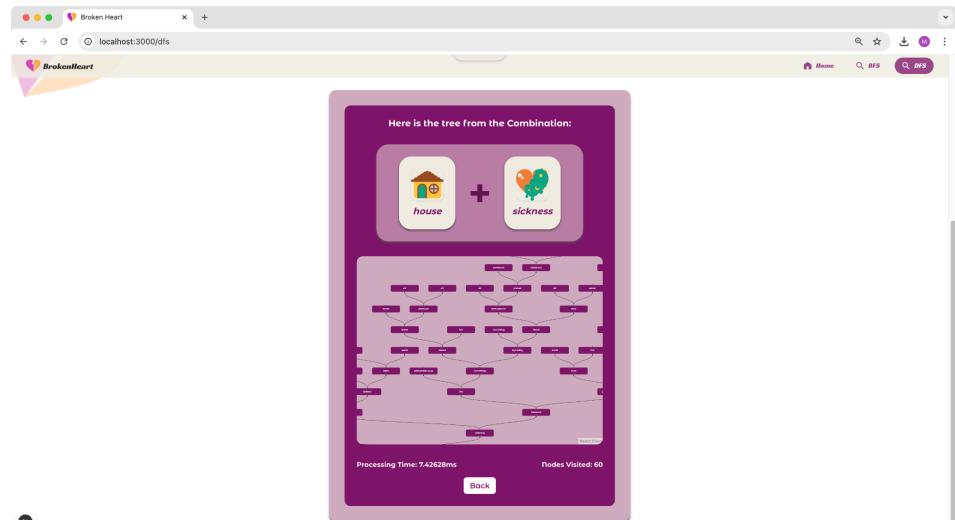
Pengujian di atas menunjukkan bahwa DFS *shortest path* mampu menunjukkan jalan terpendek dari elemen dasar ke elemen target, dengan algoritma DFS (Masih tidak sependek BFS). Oleh karena itu, DFS tidak dapat menjamin bahwa rute yang dilalui untuk mendapatkan elemen target dari elemen dasar bersifat *shortest*, tapi *shortest local* yang bisa didapatkan (algoritma sudah memilih minimum total tier, tapi masih tidak seefektif BFS).

#### 4.1.4 DFS - Multiple Result

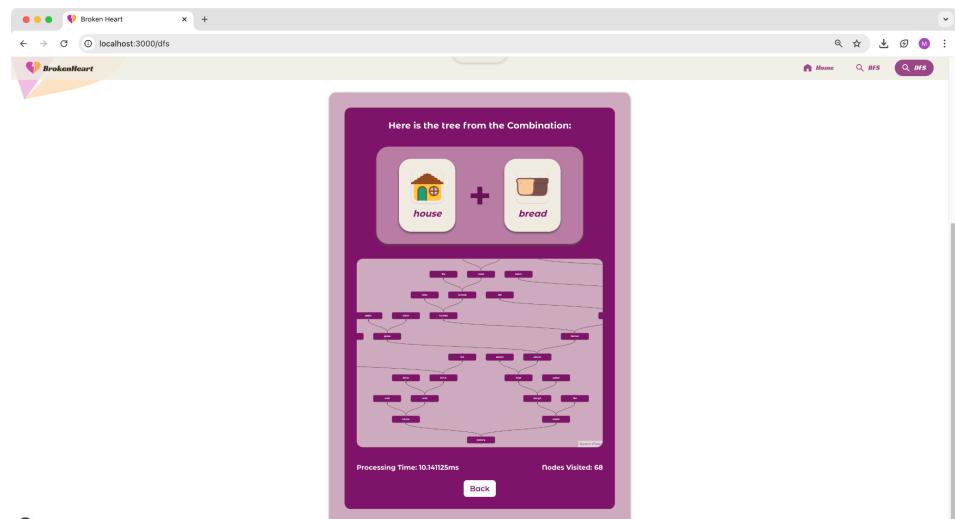
Untuk semua elemen, dipilih **pilihan pertama** dengan **max solution = 2**.

Elemen Uji	Hasil
The One Ring	<p>The screenshot shows a web browser window titled "Broken Heart" with the URL "localhost:3000/dfs". The search bar contains the text "ring" and "volcano". A pink banner at the top says "Here is the tree from the Combination:". Below it are icons for a ring labeled "ring" and a volcano labeled "volcano". To the right is a plus sign. The main area displays a complex search tree with many nodes and connections. At the bottom, it says "Processing Time: 8.828604ms" and "Nodes Visited: 59".</p>

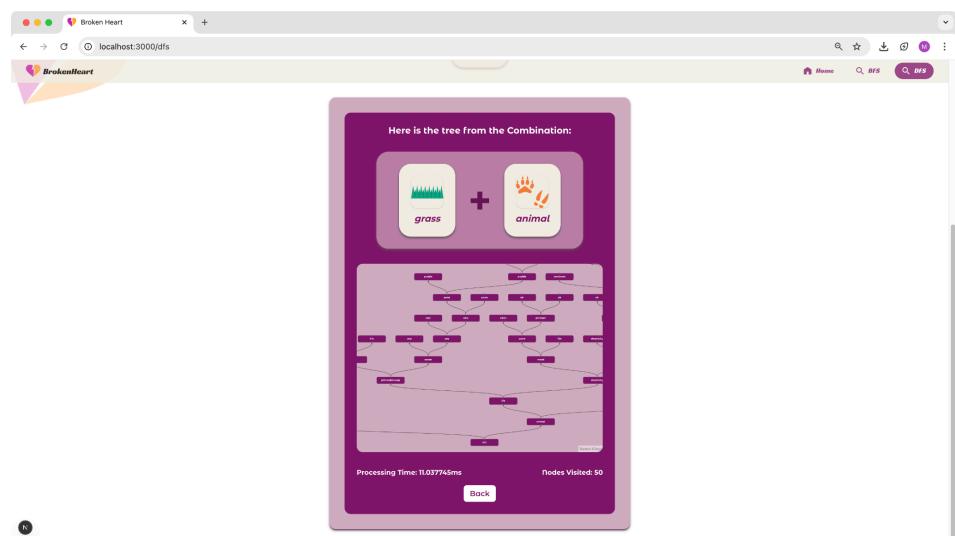
Hospital



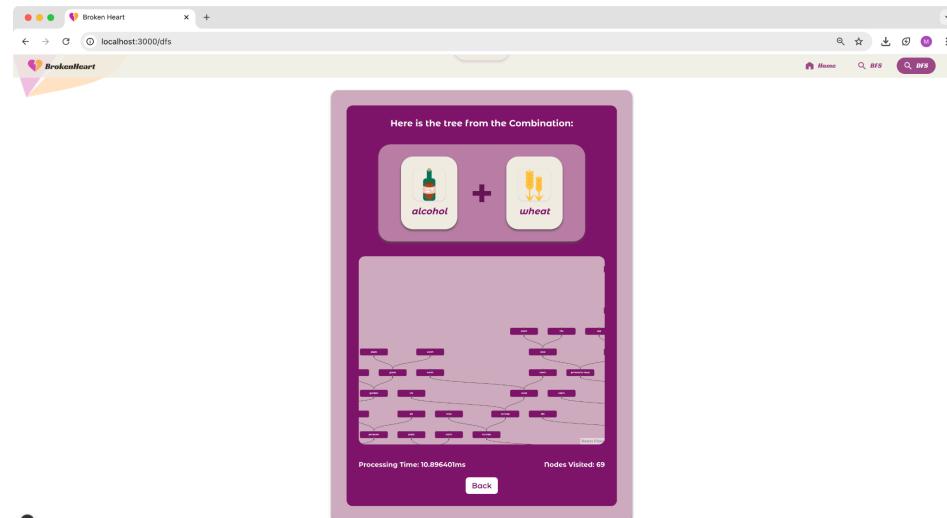
Bakery



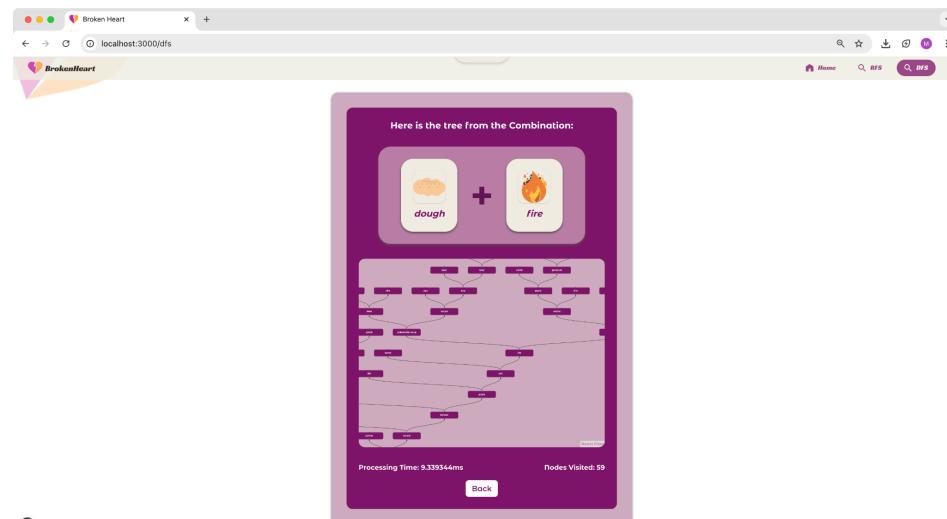
Ant



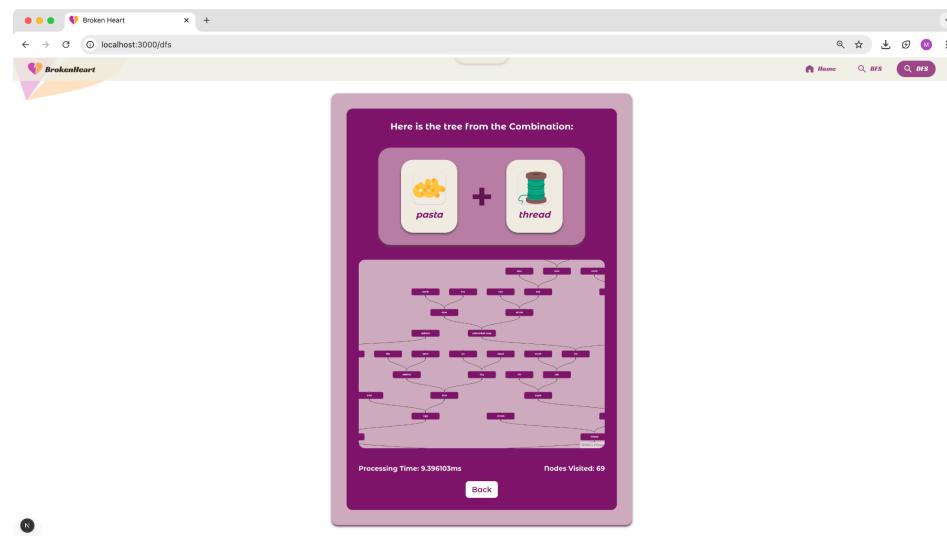
Beer



Bread

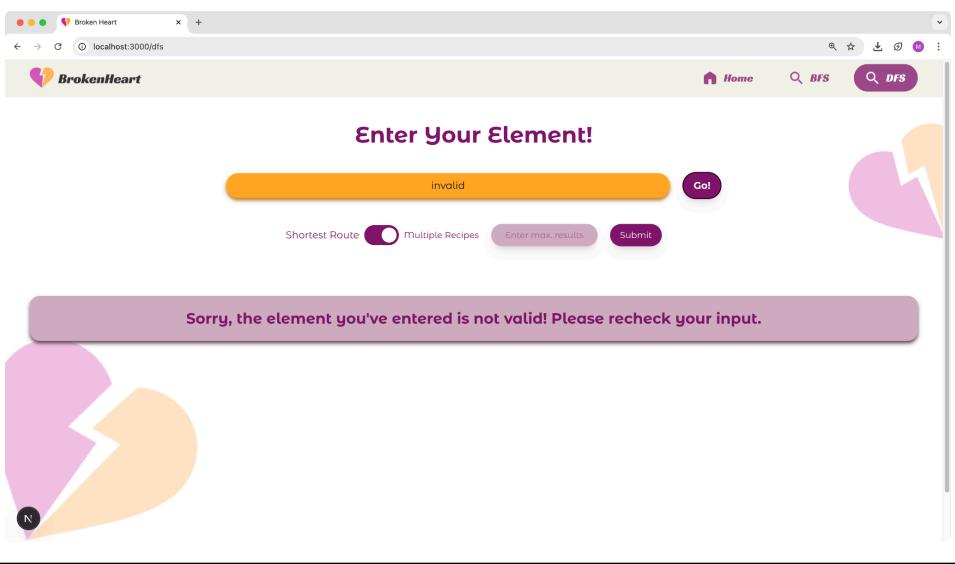


Spaghetti



Dalam mode DFS Multiple Result, program berhasil menampilkan kepada pengguna solusi sesuai dengan jumlah yang diminta, tetapi juga dibatasi dengan berapa yang mampu diberikan oleh program. User dapat memilih dengan bebas resep yang mana yang ingin dilihat treenya, kemudian program akan menampilkan tree yang sesuai. Program telah berjalan dengan baik. Akan tetapi, hasil dari program DFS tidak semangkus program BFS.

#### 4.1.5 Edge Case (Invalid Element)

Elemen Uji	Hasil
Invalid	 A screenshot of a web browser window titled "Broken Heart". The address bar shows "localhost:3000/dfs". The main content area has a pink header with the text "BrokenHeart". Below it is a form with a yellow input field containing the text "invalid". To the right of the input field is a purple "Go!" button. Underneath the input field are three buttons: "Shortest Route" (radio button), "Multiple Recipes" (checkbox), and "Enter max. results" (button). A purple "Submit" button is also present. A large, stylized broken heart graphic is on the right. A pink banner at the bottom of the page displays the message "Sorry, the element you've entered is not valid! Please recheck your input.".

Sesuai pengujian, program dapat menangani edge case, yaitu input elemen yang tidak valid dengan normal, tidak ada kerusakan dan force close. Oleh karena itu, dapat dikatakan bahwa edge case handling terlaksana dengan baik.

## BAB 5 : Kesimpulan dan Saran

### Kesimpulan

Aplikasi BrokenHeart, pencarian resep Little Alchemy 2 berhasil dibuat sesuai dengan spesifikasi, dengan asumsi tidak ada bug yang masih tersisa dan tidak disadari developer selama pengujian. Algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* berhasil diimplementasikan dengan baik dan diuji dalam dua mode pencarian, yaitu *shortest path* dan *multiple results*. Dalam mode *multiple results*, diasumsikan bahwa jumlah maksimum hasil pada mode multiple adalah sebanyak kombinasi yang memungkinkan untuk elemen target. Dari hasil pengujian, kedua algoritma (baik BFS maupun DFS) cenderung menghasilkan jalur yang relatif pendek (karena adanya aturan bahwa recipe harus memiliki tier yang lebih kecil dari target), walaupun tidak selalu merupakan jalur terpendek (kasusnya pada normal DFS). Pada aplikasi, terdapat mekanisme validasi yang cukup baik untuk menangani *edge case*. Secara keseluruhan, aplikasi ini telah memenuhi tujuan Tugas Besar 2 Strategi Algoritma.

### Saran

Masih bisa dilakukan peningkatan pada algoritma *data scraping*. Saat ini, *data scraping* dilakukan dengan mengunjungi satu per satu tautan halaman elemen, yang menyebabkan prosesnya cukup lama (lebih dari satu menit). Selain itu, pada implementasi BFS dan DFS, jalur yang dihasilkan masih dipotok untuk jalur memenuhi syarat yang pertama kali ditemukan dan jalur terpendek yang ditemukan, sehingga pengguna belum dapat memilih alternatif jalur secara manual (kecuali untuk iterasi pertama). Algoritma pencarian juga masih bisa diperlengkap dengan *algoritma bidirectional*, namun developer kehabisan waktu, sehingga manajemen waktu menjadi hal yang bisa ditingkatkan. Untuk penelitian lebih lanjut (meski tidak masuk ke spesifikasi), algoritma lain seperti *Branch and Bound* dapat dipertimbangkan, meskipun perlu *research* lebih lanjut kecocokannya dengan tugas besar ini. Apabila ada developer yang melakukan project yang sama, disarankan untuk meluangkan waktu lebih untuk eksperimen dan menentukan desain awal, karena desain struktur data yang kurang tempat dapat menjadi hambatan serius (Developer mengalami *stuck* hampir tiga hari karena kesalahan struktur data).

### Refleksi

Tantangan terbesar dalam penggerjaan tugas ini adalah merancang struktur data yang sesuai untuk BFS dan DFS, terutama karena setiap elemen punya lebih dari satu recipe. Dari sisi teknis, developer belajar banyak hal baru, terutama pada penggunaan bahasa Go Language, *data scraping*, implementasi algoritma untuk struktur data yang lebih rumit, dan pengintegrasian hasil *backend* dengan *frontend*. Meskipun waktu penggerjaan sangat terbatas, kerja sama tim berjalan cukup baik dan produktif. Salah satu momen paling memuaskan adalah ketika *backend* berhasil diintegrasikan dengan *frontend*. Secara keseluruhan, beban kerja yang diperlukan untuk menyelesaikan Tugas Besar ini terasa jauh lebih berat dari ekspektasi, terutama untuk beban mata kuliah 3 SKS. Meski begitu, developer merasa pencapaian ini sepadan dengan pembelajaran dan pengalaman teknis yang didapatkan.

## LAMPIRAN

### Tautan Github

Seluruh kode dan *resource* yang digunakan dalam *project* ini dapat diakses pada [tautan ini](#). Atau dengan mengunjungi [https://github.com/MaheswaraKaindra/Tubes2\\_BrokenHeart.git](https://github.com/MaheswaraKaindra/Tubes2_BrokenHeart.git). *Project* juga dapat diakses secara langsung melalui [tautan ini](#), atau dengan mengunjungi <https://tubes2-broken-heart.vercel.app/>.

### Tautan Video

Bonus video dapat diakses melalui [tautan ini](#). Atau dengan mengunjungi <https://youtu.be/PRx8s0Ti4Yk>.

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.	1/2	1/2

Keterangan: frontend terdeploy, backend tidak.

## **DAFTAR PUSTAKA**

Munir, R., & Maulidevi, N. U. (2025). *Breadth First Search dan Depth First Search (Bagian 1)*. Institut Teknologi Bandung.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf)

Munir, R., & Maulidevi, N. U. (2025). *Breadth First Search dan Depth First Search (Bagian 2)*. Institut Teknologi Bandung.

[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)