LAPORAN TUGAS BESAR
IF2211 STRATEGI ALGORITMA
Tugas Besar 2 Strategi Algoritma

Kelompok 47 : FullStima Alchemist

Shannon Aurellius Anastasya Lie	13523019
Angelina Efrina Prahastaputri	13523060
Sebastian Hung Yansen	13523070

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

DAFTAR ISI **2**
BAB I DESKRIPSI TUGAS **3**
BAB II LANDASAN TEORI **6**
2.1. Penjelajahan Graf 6
2.2. Algoritma Breadth First Search 6
2.3. Algoritma Depth First Search 6
2.4. Penjelasan Web Pencarian Recipe Elemen dalam Permainan Little Alchemy 2 7
BAB III ANALISIS PEMECAHAN MASALAH **8**
3.1. Mapping Persoalan Pencarian Recipe Elemen dengan BFS dan DFS 8
3.2. Langkah-Langkah Pemecahan Masalah dengan BFS dan DFS 9
3.2.1. BFS 9
3.2.2. DFS 9
3.3. Arsitektur Website (FrontEnd) 10
3.4. Arsitektur Program (BackEnd) 13
3.5. Ilustrasi Kasus 13
BAB IV IMPLEMENTASI DAN PENGUJIAN **14**
4.1. Spesifikasi Teknis Program 14
4.1.1. Struktur Repository 14
4.1.2. Struktur Data 15
4.1.3. Fungsi dan Prosedur 16
4.2. Cara Menggunakan Program 22
4.3. Hasil Pengujian 23
4.4 Analisis Pengujian 31
BAB V KESIMPULAN DAN SARAN **34**
5.1. Kesimpulan 34
5.2. Saran 34
5.3. Komentar 34
5.4. Refleksi 34
LAMPIRAN **36**
DAFTAR PUSTAKA **37**

BAB I
DESKRIPSI TUGAS

Gambar 1. Little Alchemy 2
(sumber: https://www.thegamer.com)

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Recloak yang dirilis pada tahun 2017, permainan ini bertujuan untuk membuat 720 elemen dari 4 elemen dasar yang tersedia yaitu *air, earth, fire,* dan *water*. Permainan ini merupakan sekuel dari permainan sebelumnya yakni Little Alchemy 1 yang dirilis tahun 2010. Mekanisme dari permainan ini adalah pemain dapat menggabungkan kedua elemen dengan melakukan *drag and drop*, jika kombinasi kedua elemen valid, akan memunculkan elemen baru, jika kombinasi tidak valid maka tidak akan terjadi apa-apa. Permainan ini tersedia di *web browser,* Android atau iOS

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

Komponen-komponen dari permainan ini antara lain:

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

Gambar 2. Elemen dasar pada Little Alchemy 2

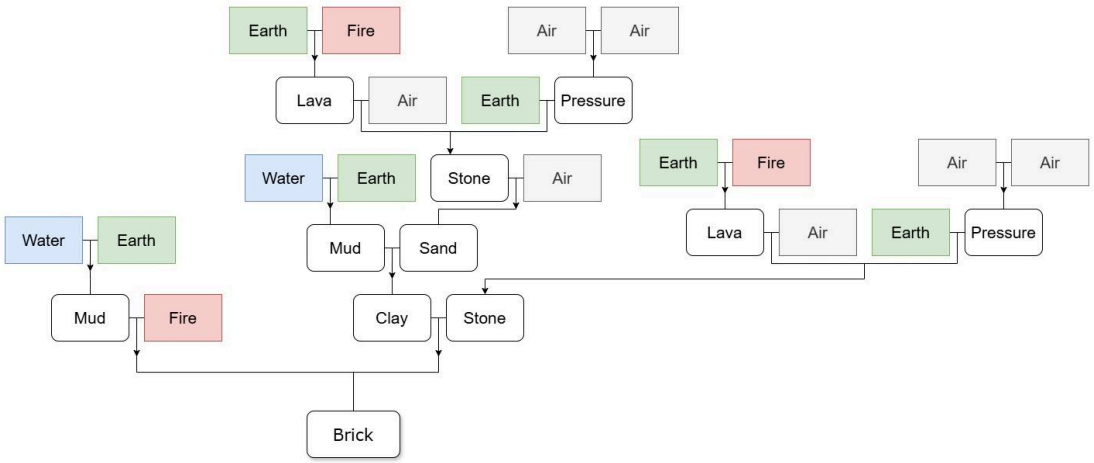
2. Elemen turunan
   Terdapat 720 elemen turunan yang dibagi menjadi beberapa *tier* tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki *recipe* yang terdiri atas elemen lainnya atau elemen itu sendiri.
3. *Combine Mechanism*
   Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

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*** (Silahkan yang mana saja) ~~**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.***
- Elemen yang digunakan pada suatu *recipe* harus berupa elemen dengan ***tier* lebih rendah** dari elemen yang ingin dibentuk.
- 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 di atas 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.

BAB II
LANDASAN TEORI

2.1. Penjelajahan Graf

Graf adalah struktur data yang digunakan untuk merepresentasikan objek-objek diskrit serta hubungan di antara objek-objek tersebut. Dalam konteks graf, objek-objek tersebut direpresentasikan sebagai simpul (*vertex*), dan hubungan antar objek direpresentasikan sebagai sisi (*edge*). Secara formal, sebuah graf G didefinisikan sebagai pasangan himpunan G = (V, E) dengan V adalah himpunan tidak kosong dari simpul-simpul (*vertices*), misalnya: V = {v_1, v_2, ..., $v_□$} dan E adalah himpunan sisi-sisi (*edges*) yang menghubungkan pasangan simpul, misalnya: E = {e_1, e_2, ..., $e_□$}. Graf sering digunakan dalam berbagai aplikasi, seperti peta jaringan jalan yang menghubungkan kota-kota (setiap kota sebagai simpul, dan jalan sebagai sisi), jaringan komputer, dan relasi antar individu dalam media sosial, dan lain sebagainya.

2.2. Algoritma *Breadth First Search*

Breadth-First Search (BFS) adalah salah satu algoritma pencarian dan penelusuran graf yang digunakan untuk mengunjungi semua simpul dalam suatu graf. BFS bekerja dengan menelusuri graf secara melebar dari suatu simpul awal. Artinya, BFS mengunjungi semua tetangga (simpul yang bertetangga langsung) dari simpul awal terlebih dahulu, kemudian berlanjut ke tetangga dari tetangga tersebut, dan seterusnya. BFS diimplementasikan menggunakan struktur data *queue* (antrian), yang mengikuti prinsip FIFO (*First In First Out*). Berdasarkan pendekatan ini, BFS sangat baik digunakan untuk mencari jarak terpendek dalam graf tak berberat, serta untuk menentukan level dari tiap simpul relatif terhadap simpul awal.

2.3. Algoritma *Depth First Search*

Depth-First Search (DFS) merupakan salah satu algoritma pencarian dan penelusuran graf yang digunakan untuk mengunjungi semua simpul dalam suatu graf. DFS bekerja dengan cara menelusuri graf secara mendalam. Dari simpul awal, DFS terus menyusuri simpul-simpul tetangga sejauh mungkin hingga tidak ada simpul yang belum dikunjungi, lalu kembali (*backtrack*) ke simpul sebelumnya untuk menjelajah jalur lain. DFS diimplementasikan menggunakan stack (baik

eksplisit maupun secara implisit melalui rekursi), yang mengikuti prinsip LIFO (*Last In First Out*). DFS cocok digunakan untuk menemukan komponen yang terhubung, melakukan deteksi siklus, dan menghasilkan urutan topologis dalam graf berarah.

2.4. Penjelasan Web Pencarian *Recipe* Elemen dalam Permainan Little Alchemy 2

Web yang kami buat merupakan sebuah web pencarian resep dari elemen yang terdapat pada permainan Little Alchemy 2. Web ini memungkinkan pengguna untuk mencari satu atau lebih resep untuk suatu elemen dalam permainan Little Alchemy 2 menggunakan algoritma penelusuran BFS dan DFS. Backend menggunakan Golang dan frontend menggunakan Next.js.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1. *Mapping* Persoalan Pencarian *Recipe* Elemen dengan BFS dan DFS

Pertama-tama, dilakukan *scraping* terhadap situs wiki Fandom Little Alchemy untuk mendapatkan elemen beserta resep. Elemen dan resep yang didapat disimpan dalam bentuk .json untuk dibaca oleh fungsi utama. Masing-masing elemen disimpan pada *tier* tertentu yang berjarak dari 0 hingga 15. Elemen dengan *tier* 0 merupakan kumpulan elemen dasar yang mencakup *Air, Earth, Water, Fire*. Perlu diketahui bahwa proses pemetaan dari Go tidak teratur dan memang begitu dari sananya.

File json yang berisi kumpulan elemen dan resep. Isi file tersebut dipetakan dalam sebuah struct yang mengandung nama elemen dan resep-resep untuk membuat elemen tersebut. Untuk program utama akan dikembalikan resep, elemen dasar, serta angka *tier* elemen. Dikhususkan pada tugas besar ini, elemen *Time* dan semua resep yang mengandung elemen *Time* tidak diolah dalam pencarian resep.

Pencarian resep suatu elemen merupakan persoalan utama yang harus diselesaikan menggunakan algoritma *Breadth First Search* dan *Depth First Search*. Resep yang dihasilkan harus menyusun elemen target dari elemen dasar dan elemen target hanya dapat disusun dari elemen-elemen dengan *tier* yang tidak lebih tinggi daripada *tier* elemen target. Contohnya, elemen target dengan *tier* 9 tidak dapat disusun oleh elemen dengan *tier* 10 dan *tier-tier* di atasnya.

Tidak hanya menemukan satu resep saja, algoritma BFS dan DFS dapat digunakan untuk menemukan berbagai resep jika memutuskan untuk melakukan pencarian terhadap resep lain. Hal ini diterapkan dengan memodifikasi algoritma BFS dan DFS untuk melakukan penelusuran lebih lanjut untuk mencari resep lain meskipun satu resep sudah ditemukan. Hal ini dioptimasi menggunakan *multithreading*. *Multithreading* memungkinkan program menjalankan berbagai *tasks* secara konkuren sehingga pencarian resep yang berbeda untuk suatu elemen dapat dilakukan dengan lebih efisien.

3.2. Langkah-Langkah Pemecahan Masalah dengan BFS dan DFS

3.2.1. BFS

Langkah-langkah pemecahan masalah dengan BFS adalah sebagai berikut:

1. Fungsi BFS akan menerima parameter elemen target yang akan dicari resepnya, data resep atau kombinasi penyusun elemen, elemen dasar, dan data *tier* tiap elemen. Fungsi akan melakukan inisialisasi *queue* dan kumpulan elemen yang sudah ada atau *discovered elements*. *Queue* mula-mula berisi elemen dasar *Air, Earth, Fire, Water*.
2. Selama *queue* tidak kosong, fungsi akan mengambil elemen paling depan pada *queue* dan mencoba mengkombinasikan elemen tersebut dengan *discovered elements*, termasuk dengan elemen itu sendiri.
3. Jika hasil kombinasi adalah resep yang valid dan menghasilkan elemen yang belum ada pada *discovered elements* dan *tier*-nya tidak lebih tinggi daripada elemen target, maka elemen tersebut ditambahkan ke *queue*.
4. Fungsi akan terus mencoba semua kombinasi dari elemen yang diambil sekarang dengan *discovered elements* hingga elemen target berhasil diperoleh (elemen target berada pada awal *queue*) atau *queue* kosong. Fungsi kemudian mengembalikan jalur penelusuran dan resep dari elemen (kombinasi elemen yang mengarah langsung ke elemen target, menghilangkan *unused combinations*)
5. Node yang dianggap valid atau telah dikunjungi ditunjukkan oleh kombinasi yang dicoba fungsi berdasarkan urutan *queue*, sehingga sangat wajar apabila kombinasi pada resep yang didapat lebih sedikit daripada total node yang dikunjungi saat jalur penelusuran resep.
6. Pencarian *multiple recipes* juga menggunakan alur yang sama, hanya saja keluarannya berupa jalur penelusuran, total node dari seluruh pencarian tiap resep, dan *list of recipes*. Pembeda satu resep dengan resep lainnya adalah minimal satu kombinasi berbeda.

3.2.2. DFS

Langkah-langkah pemecahan masalah dengan DFS adalah sebagai berikut:

1. Fungsi DFS akan menerima parameter elemen target yang dicari, data resep, elemen dasar, serta data *tier* masing-masing elemen.

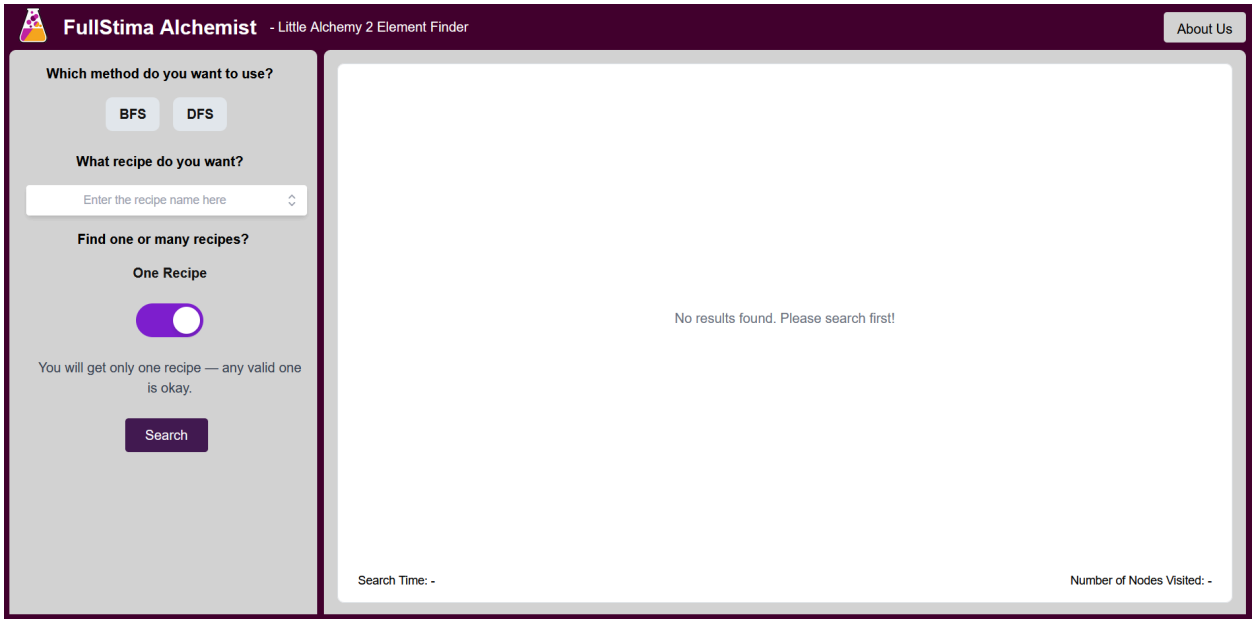
2. Fungsi DFS pencarian satu resep melakukan pencarian secara rekursif dan menyimpan state elemen serta jalurnya. Pada awal program, fungsi menyimpan elemen dasar yang terdiri atas *air, fire, earth, water* sebagai state elemen awal.
3. Program akan mencoba menggabungkan elemen-elemen dalam state satu sama lain secara berurutan. Jika hasil kombinasi valid dan menghasilkan elemen yang baru serta tier elemen tidak lebih besar, elemen beserta jalurnya akan disimpan sebagai state baru dan fungsi DFS akan dipanggil kembali dengan state yang baru secara rekursif hingga elemen target ditemukan.
4. Fungsi DFS pencarian lebih dari satu resep melakukan pencarian menggunakan *stack*. Program menginisialisasi stack yang berisi elemen dasar.
5. *Head* dari stack di-pop dan akan diperiksa apakah sesuai dengan target atau tidak dan apakah path sama persis dengan pencarian sebelumnya atau tidak. Apabila tidak, fungsi akan mencoba menggabungkan elemen-elemen dalam state. Jika hasil kombinasi valid dan menghasilkan elemen yang baru serta tier elemen tidak lebih besar, elemen beserta jalurnya akan disimpan sebagai state baru.
6. State baru akan di-push ke stack dan program akan mengulangi fungsi hingga menemukan elemen target yang ingin dicari.
7. Jika fungsi berhasil menemukan elemen target beserta resepnya, *path* pencarian elemen akan disimpan. Apabila jumlah resep yang ingin dicari sudah cukup, fungsi akan keluar dengan sendirinya dan mengembalikan *path-path* elemen

3.3. Arsitektur *Website* (*FrontEnd*)

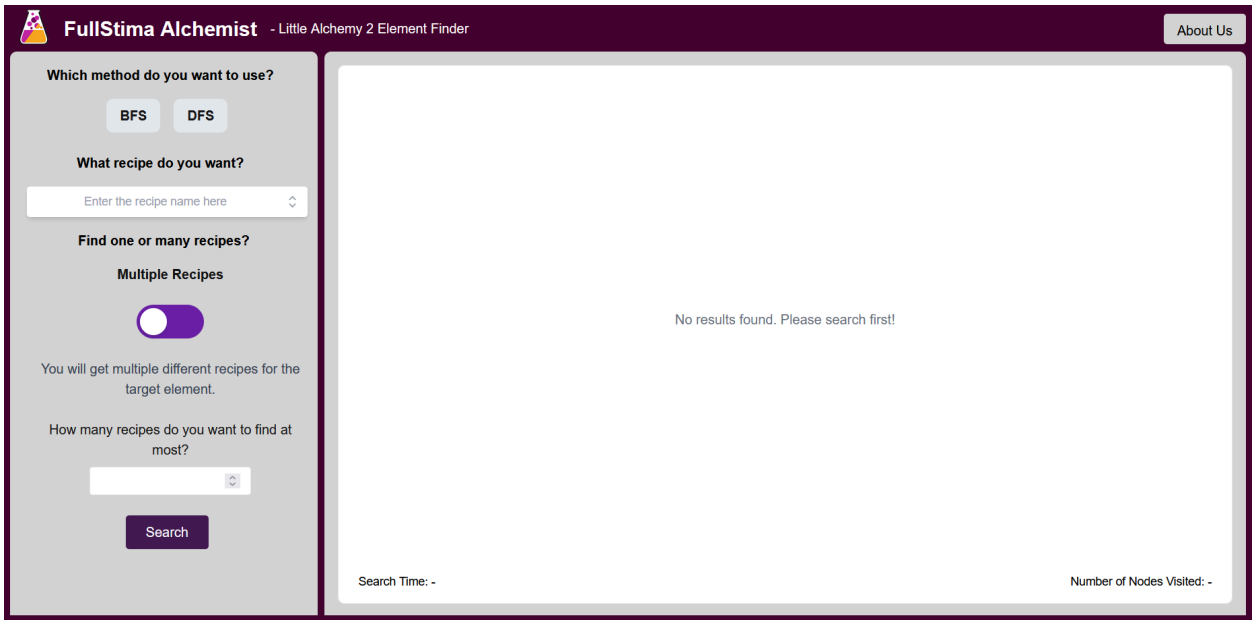
Website FullStima Alchemist dikembangkan menggunakan *framework* Next.js untuk sisi *frontend*, dipadukan dengan Tailwind CSS untuk *styling* yang efisien dan responsif. Proyek ini memiliki struktur modular yang terdiri dari tiga file JavaScript utama. File pertama, page.js yang berada di dalam direktori src/app, berfungsi sebagai halaman utama (*homepage*) dari *website*. Di dalamnya dimuat struktur dasar antarmuka serta pemanggilan komponen visual utama. Komponen kedua adalah TreeChart.js, sebuah modul yang mengelola dan merender visualisasi berbentuk pohon (*tree visualization*) berbasis data resep menggunakan pustaka D3.js. Komponen ini memegang peranan penting dalam menyajikan hubungan antar elemen secara interaktif dan intuitif. File ketiga adalah page.js di dalam direktori src/app/aboutpage, yang bertugas menampilkan halaman informasi "About Us" (About Page) dari *website*, berisi penjelasan atau

latar belakang mengenai FullStima Alchemist. Seluruh antarmuka disusun dengan gaya desain modern berkat dukungan dari Tailwind CSS, sehingga menghasilkan tampilan yang bersih, terstruktur, dan mudah diakses pengguna. Program hanya dapat dijalankan melalui browser dengan spesifikasi minimal Mozilla 5.0.

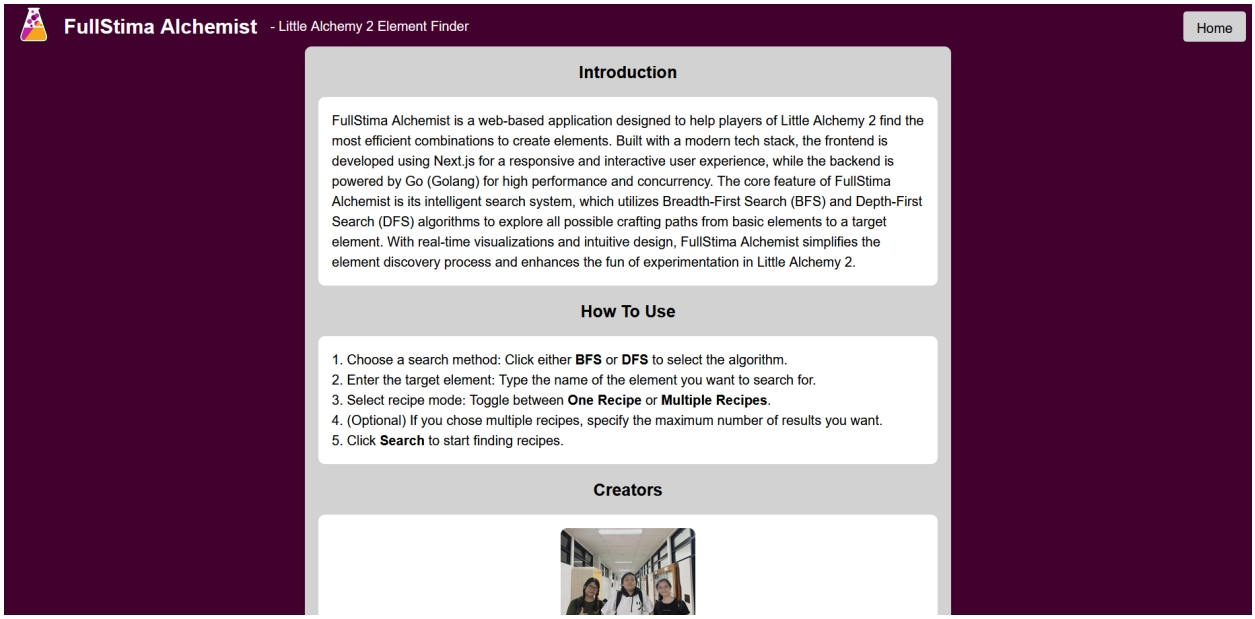
Berikut merupakan beberapa *screenshot* dari antarmuka pengguna dalam keadaan belum melakukan pencarian dan laman *About Us*.



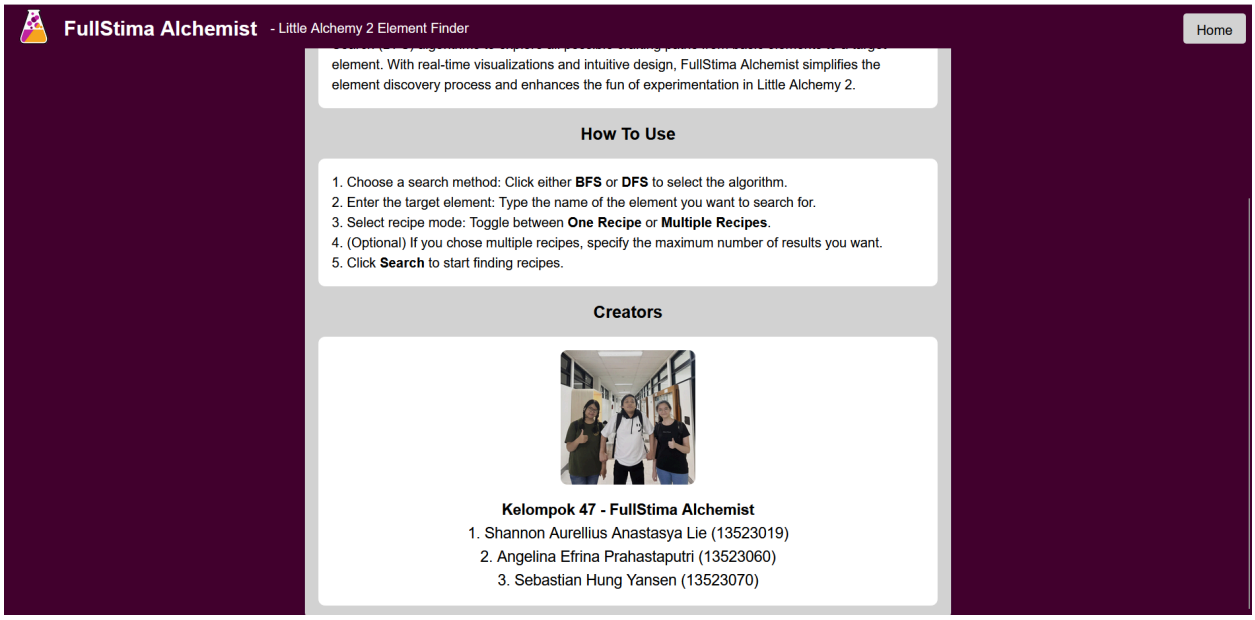
Gambar 4 Laman Pencarian awal dalam Kondisi *One Recipe* (*default*)



Gambar 5 Laman Pencarian awal dalam Kondisi *Multiple Recipes*



Gambar 6 Laman *About Us* (1)



Gambar 7 Laman *About Us* (2)

3.4. Arsitektur Program (*BackEnd*)

Bagian backend dari program ini dikembangkan menggunakan Golang (Go), sebuah bahasa pemrograman yang dikenal karena efisiensi, kecepatan eksekusi, dan kemampuan concurrency-nya. File utama seperti main.go berfungsi sebagai titik masuk aplikasi, sedangkan algorithm.go dan scrapper.go menangani logika bisnis, termasuk proses alur pembuatan resep dan pengambilan data. File recipe.json menyimpan data resep elemen dalam format terstruktur yang dapat dibaca oleh server. Dependensi proyek dikelola melalui go.mod dan go.sum, yang memastikan konsistensi versi modul selama proses build. Selain itu, folder public/svgs berisi seluruh aset gambar elemen dalam format .svg, termasuk ikon default, yang dapat dilayani secara langsung oleh server ke frontend. Meskipun terdapat file server.js, backend utama tetap dijalankan dengan Go, menjadikan arsitektur ini cepat, ringan, dan cocok untuk aplikasi berbasis data seperti FullStima Alchemist. Program hanya dapat dijalankan melalui browser dengan spesifikasi minimal Mozilla 5.0.

3.5. Ilustrasi Kasus

Berikut merupakan contoh kasus dan ilustrasi penggunaan *website* FullStima Alchemist. Kasus yang terjadi adalah seorang pengguna ingin mengetahui cara membuat elemen “Stone” dari elemen-elemen dasar yang tersedia. Berikut ilustrasi alur penggunaan program:

1. Pengguna membuka halaman utama frontend (dibuat dengan Next.js dan Tailwind).
2. Pengguna mencari atau memilih elemen “Stone”.
3. Sistem *frontend* mengirim permintaan ke backend (dibuat dengan Golang) untuk mendapatkan tree resep pembuatan “Stone”.
4. Backend memproses permintaan dengan membaca recipe.json dan menggunakan algoritma dari algorithm.go untuk membentuk pohon resep.
5. Respon dikirim kembali ke *frontend* dalam bentuk data JSON.
6. Komponen TreeChart.js menerima data ini dan merender visualisasi pohon elemen yang menunjukkan bahwa “Stone” berasal dari kombinasi misalnya “Earth” + “Pressure”. Pengguna bisa melihat jalur pembuatan elemen tersebut dalam bentuk graf.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

4.1.1. Struktur *Repository*

Berikut merupakan struktur *repository* program yang terdiri dari tiga folder utama yaitu backend, doc, dan frontend. Folder backend berisi seluruh komponen logika server yang dibangun dengan bahasa pemrograman Go (Golang), termasuk file algoritma, pemrosesan data resep, serta direktori public/svgs yang menyimpan aset gambar elemen. Folder doc digunakan untuk menyimpan dokumentasi laporan. Sementara itu, folder frontend berisi source code antarmuka pengguna yang dikembangkan menggunakan Next.js dan Tailwind CSS, termasuk konfigurasi dan file halaman utama serta halaman informasi. Struktur ini memisahkan tanggung jawab dengan jelas antara sisi server, dokumentasi, dan antarmuka pengguna.

```
|  README.md
|
+---backend
|  |  algorithm.go
|  |  go.mod
|  |  go.sum
|  |  main.go
|  |  recipe.json
|  |  scrapper.go
|  |  server.js
|  |
|  \---public
|      \---svgs //berisi .svg seluruh element dan default image
|
+---doc
|      FullStima Alchemist.pdf
|
\---frontend
   |  .eslintrc.json
   |  .gitignore
   |  jsconfig.json
   |  next.config.mjs
   |  package-lock.json
```

```
|   package.json
|   postcss.config.js
|   postcss.config.mjs
|   tailwind.config.js
|
+---.next
|   |
|   +---build
|   |   \---chunks
|   |
|   +---cache
|   |
|   +---server
|   |   |
|   |   +---app
|   |   |   |   page.js
|   |   |   |   page.js.map
|   |   |   |   page_client-reference-manifest.js
|   |   |   |
|   |   |   +---aboutpage
|   |   |   |   |   page.js
|   |   |   |   |   page.js.map
|   |   |   |   |   page_client-reference-manifest.js
|   |   |   |   |
|   |   |   |   \---page
```

4.1.2. Struktur Data

Resep yang didapatkan melalui scraping website disimpan dalam bentuk json untuk memudahkan pengaksesan.

Pada backend, Node disimpan dalam bentuk struct yang menyimpan atribut nama elemen, resep yang bersangkutan, serta elemen anak yang yang membuat elemen tersebut. Resep-resep disimpan dalam map of string, tier dari masing-masing elemen disimpan dalam map.

Pada masing-masing fungsi, BFS single recipe menggunakan queue, DFS single recipe menggunakan struct, BFS multi recipe menggunakan struct dan queue, dan DFS multi recipe menggunakan state dan struct. BFS multi recipe dan DFS multi recipe keduanya menggunakan multithreading dengan goroutines.

4.1.3. Fungsi dan Prosedur

Berikut merupakan penjelasan mengenai fungsi dan prosedur pada file-file backend.

1. **File algorithm.go**
    - **Atribut**

Atribut	Deskripsi

type Node struct {
Element string `json:"element"`
Recipe string `json:"recipe"`
Children []*Node `json:"children"`
}
	Struct Node digunakan untuk membentuk struktur pohon dari proses pembuatan elemen, di mana setiap node bisa memiliki dua anak (karena setiap elemen terbentuk dari dua elemen sebelumnya).

type ElementFromFandom struct {
Element string `json:"element"`
Recipes [][2]string `json:"recipes"`
}
	Struct ini mencerminkan format JSON dari file elements.json. Element merupakan nama elemen yang akan dibentuk. Recipes merupakan array dari kombinasi pasangan bahan pembentuk elemen ini.

type queue []string	Struct queue untuk BFS

- **Konstruktor**

Konstruktor	Deskripsi

&Node{
Element: result,
Recipe: "...",
Children: []*Node{left, right},
}
	Kode digunakan saat membentuk node baru dalam struktur pohon selama proses DFS.

- **Metode**

Metode	Deskripsi
func createKey(a, b string) [2]string	Mengembalikan key unik berdasarkan dua elemen (diurutkan alfabetis) sebagai pasangan reaksi.
func keys(m map[string]bool) []string	Fungsi keys menerima sebuah map bertipe map[string]bool dan mengembalikan array (slice) berisi semua kunci (key) dari map tersebut. Fungsi ini sering digunakan

	untuk mengekstrak himpunan elemen dari sebuah set yang diwakili dengan map boolean. Misalnya, jika map tersebut menyimpan elemen-elemen yang sudah ditemukan dalam sebuah state, maka fungsi ini akan menghasilkan daftar elemen yang bisa diolah lebih lanjut.
func copySet(original map[string]bool) map[string]bool	Fungsi copySet membuat salinan (deep copy) dari sebuah set yang direpresentasikan oleh map[string]bool. Ini penting agar ketika sebuah state atau kondisi disalin, perubahan pada salinan tidak memengaruhi data asli. Fungsi ini digunakan untuk menjaga keutuhan data saat melakukan eksplorasi lintasan dalam pencarian seperti BFS atau DFS.
func stateToString(set map[string]bool) string	Fungsi stateToString mengubah sebuah set map[string]bool menjadi string representatif. Biasanya, kunci-kunci disortir lalu digabung menjadi satu string. Tujuan utamanya adalah menghasilkan bentuk yang bisa digunakan sebagai "fingerprint" unik dari suatu keadaan (state) — misalnya saat menyimpan atau mengecek visited state dalam algoritma pencarian.
func pathToTree(path []string) *Node	Fungsi pathToTree mengonversi sebuah lintasan (path) — berupa daftar string kombinasi resep — menjadi struktur pohon *Node. Fungsi ini bekerja secara rekursif, memecah string seperti "Water + Fire" menjadi dua elemen anak dan membentuk struktur hierarki yang menggambarkan bagaimana sebuah elemen kompleks dibentuk dari elemen-elemen dasar.
func buildRecipeMap(elements []ElementFromFandom) map[[2]string]string	Fungsi ini membangun sebuah peta (map) dari pasangan bahan [2]string ke elemen hasil string. Fungsi ini digunakan untuk lookup cepat ketika ingin tahu apakah dua bahan tertentu dapat dikombinasikan untuk membentuk elemen baru. Misalnya,

	["Water", "Fire"] akan dipetakan ke "Steam" jika memang ada dalam data elements.
func buildTierMap(elements []ElementFromFandom) map[string]int	Fungsi buildTierMap menghitung tingkat (tier) dari setiap elemen berdasarkan urutan pembentukannya. Elemen dasar akan memiliki tier 0, dan elemen yang dibentuk dari kombinasi dua elemen lainnya akan memiliki tier yang lebih tinggi. Hasilnya adalah sebuah map dari elemen ke bilangan bulat yang menunjukkan tingkat kompleksitas atau kedalaman elemen dalam hierarki pembentukan.
func serializeTree(node *Node) string	Fungsi serializeTree mengubah struktur pohon dari node *Node ke dalam bentuk string JSON. Fungsi ini memudahkan penyimpanan atau pengiriman struktur pohon ke frontend (misalnya ke D3.js). Dalam implementasinya, fungsi ini mungkin menggunakan serialisasi rekursif untuk mengubah semua node dan anak-anaknya menjadi representasi teks terstruktur.
func (q *queue) isEmpty() bool	Metode isEmpty adalah bagian dari struktur data queue (antrian), dan digunakan untuk memeriksa apakah antrian kosong. Ini penting dalam algoritma pencarian seperti BFS, untuk menentukan kapan pencarian harus berhenti.
func (q *queue) enqueue(data string)	Metode enqueue menambahkan elemen bertipe string ke akhir antrian. Dalam konteks algoritma BFS, ini digunakan untuk menambahkan state atau langkah selanjutnya ke dalam antrian agar dieksplorasi di iterasi berikutnya.
func (q *queue) dequeue() (string, bool)	Metode dequeue mengambil elemen pertama dari antrian dan menghapusnya dari antrian tersebut. Fungsi ini mengembalikan elemen tersebut serta boolean yang menunjukkan apakah

	operasi berhasil (misalnya, false jika antrian kosong). Ini merupakan inti dari siklus BFS.
bfs(target string, recipesMap map[[2]string]string, reverseRecipes map[string][][2]string) ([]string, int, error)	Melakukan pencarian Breadth-First Search untuk mencari satu path menuju elemen target. Fungsi ini mengembalikan urutan langkah, jumlah node yang diperiksa, dan error jika tidak ditemukan. Menggunakan queue dan visited set untuk hindari siklus.
dfs(target string, recipesMap map[[2]string]string, reverseRecipes map[string][][2]string) ([]string, int, error)	Melakukan pencarian Depth-First Search untuk mencari satu path menuju elemen target. Fungsi ini mengembalikan urutan langkah, jumlah node yang diperiksa, dan error jika tidak ditemukan. DFS menggunakan stack (rekursif). Lebih cocok jika ingin mendapatkan jalur mendalam pertama.
func bfsMultiplePaths(target string, recipes map[[2]string]string, baseElements map[string]bool, amtOfMultiple int, elementToTier map[string]int,) (map[string][][]string, int, error) {	Melakukan pencarian Breadth-First Search untuk mencari sejumlah path menuju elemen target. Fungsi ini mengembalikan urutan langkah, jumlah node yang diperiksa, dan error jika tidak ditemukan. Menggunakan queue dan visited set untuk hindari siklus.
func equalStrings(a, b []string) bool	Fungsi equalStrings membandingkan dua buah slice bertipe []string untuk menentukan apakah mereka memiliki isi yang sama secara berurutan. Ini digunakan, misalnya, untuk membandingkan dua path atau dua representasi state dalam bentuk slice string. Perbandingan bersifat elemen demi elemen dan urut, bukan hanya isinya saja.
func dfsMultiplePaths(target string, recipes map[[2]string]string, baseElements map[string]bool, amtOfMultiple int, elementToTier map[string]int,) ([][]string, int, error) {	Melakukan pencarian Depth-First Search untuk mencari sejumlah path menuju elemen target. Fungsi ini mengembalikan urutan langkah, jumlah node yang diperiksa, dan error jika tidak ditemukan. DFS menggunakan stack (rekursif). Lebih cocok jika ingin mendapatkan jalur mendalam pertama.

func printFoundPaths(pathMap map[string][][]string)	Mencetak seluruh path yang ditemukan ke terminal/log, tidak mengembalikan nilai apapun.

2. **File main.go**

   - **Atribut**

Atribut	Deskripsi

type TreeResponse struct {
Trees []*Node `json:"trees"`
NodesVisited int `json:"nodesVisited"`
SearchTime float64 `json:"searchTime"`
}
	Struct TreeResponse digunakan untuk membentuk struktur pohon dari semua pohon resep, jumlah node yang dikunjungi, dan waktu pencarian.

var recipeData []ElementFromFandom	Berisi data semua resep dari hasil scraping

type SearchRequest struct {
Method string `json:"method"`
Target string `json:"target"` // nama elemen
Multiple bool `json:"multiple"`
MaxRecipes int `json:"maxRecipes"`
}
	Struct SearchRequest digunakan untuk menyimpan data untuk backend yang menyangkut metode yang digunakan, target elemen yang dituju, apakah pencarian multiple atau tidak, dan jumlah resep yang dicari.

   - **Konstruktor**

   Tidak terdapat konstruktor.

   - **Metode**

Metode	Deskripsi
func main()	Fungsi utama program yang memanggil scraper dan memanggil fungsi penyimpanan ke json
func recipeHandler(w http.ResponseWriter, r *http.Request)	Fungsi yang memproses *request* dari frontend, decode json, membuat mapping dari resep dan tier, serta memanggil fungsi bfs/dfs sesuai parameter dan menyimpannya pada struct TreeResponse.
func saveToJsonFile(data []ElementFromFandom, filename string)	Fungsi untuk menyimpan data scraping ke bentuk file json

func jsonHandler(w http.ResponseWriter, r *http.Request)	Fungsi yang membantu membuat file json dapat diakses

3. **File scrapper.go**
    - **Atribut**

Atribut	Deskripsi

type ElementFromFandom struct {
Name string `json:"element"`
Tier int `json:"tier"`
LocalSVGPath string `json:"local_svg_path"`
OriginalSVGURL string `json:"original_svg_url"`
Recipes [][]string `json:"recipes"`
}
	Struct ElementFromFandom digunakan untuk menyimpan nama elemen, tier, path gambar lokal, path gambar web, dan resep dari elemen

- **Konstruktor**

Tidak terdapat konstruktor.

- **Metode**

Metode	Deskripsi
func ScrapeAll() ([]ElementFromFandom, error)	Fungsi scraping utama elemen-elemen dari website yang dituju (dalam kasus ini, list elemen fandom wikia) dan menyimpannya pada ElementFromFandom struct
func downloadSVG(url, dest string)	Fungsi men-*download* gambar-gambar elemen dari website yang dituju (dalam kasus ini, list elemen fandom wikia) dan menyimpannya dalam bentuk svg

4.2. Cara Menggunakan Program

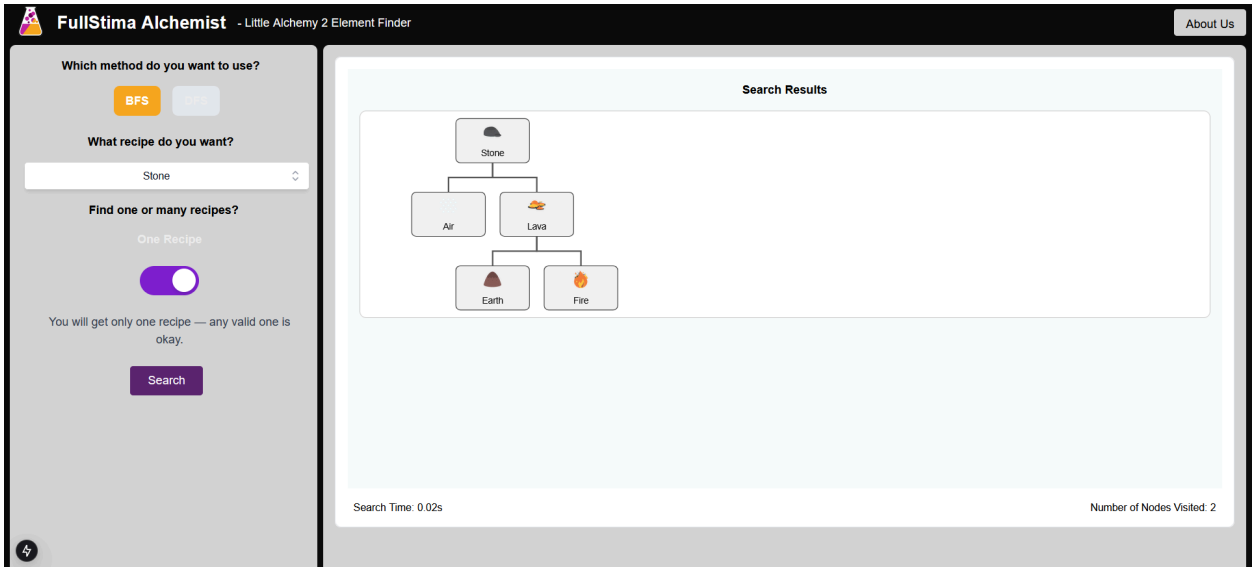
Program berbasis web dan pengguna memiliki pilihan untuk melakukan pencarian secara DFS atau BFS. Selain itu, pengguna memiliki pilihan untuk mencari 1 resep saja atau lebih dari 1 resep. Pengguna bisa memasukkan resep elemen yang ingin dicari. Setelah pengguna menekan tombol search, program akan memuat dan menampilkan *nodetree* sesuai dengan parameter yang telah diberikan.

4.3. Hasil Pengujian

1. Stone

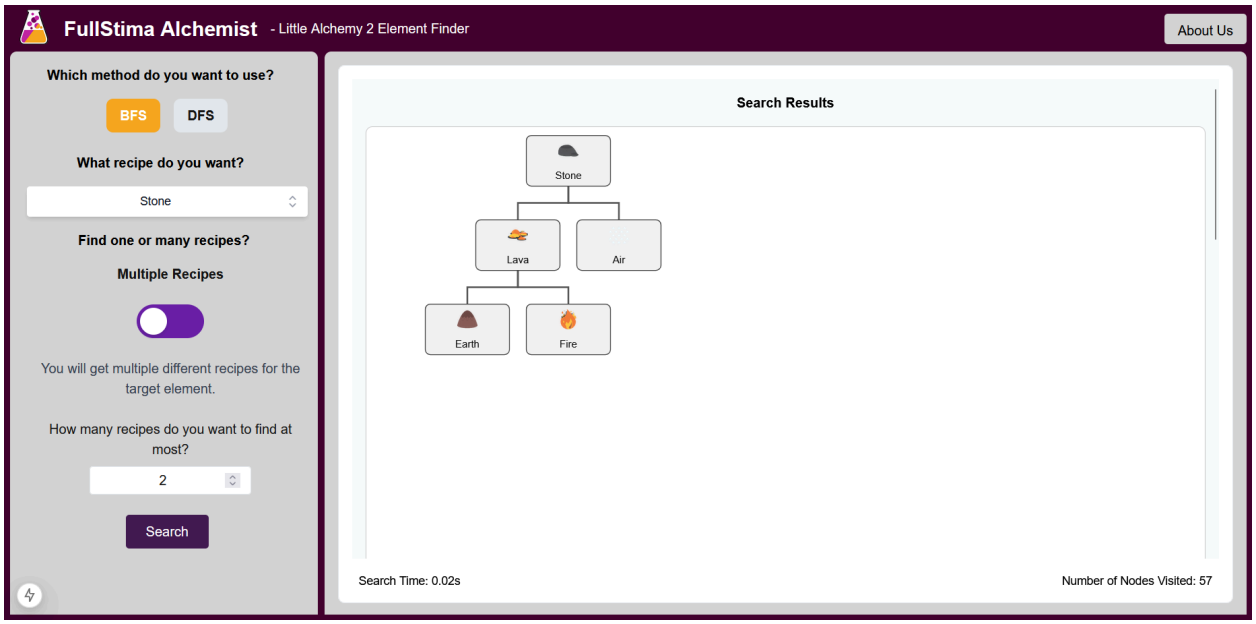
   Berikut merupakan hasil pengujian pada *website* untuk membentuk elemen Stone.

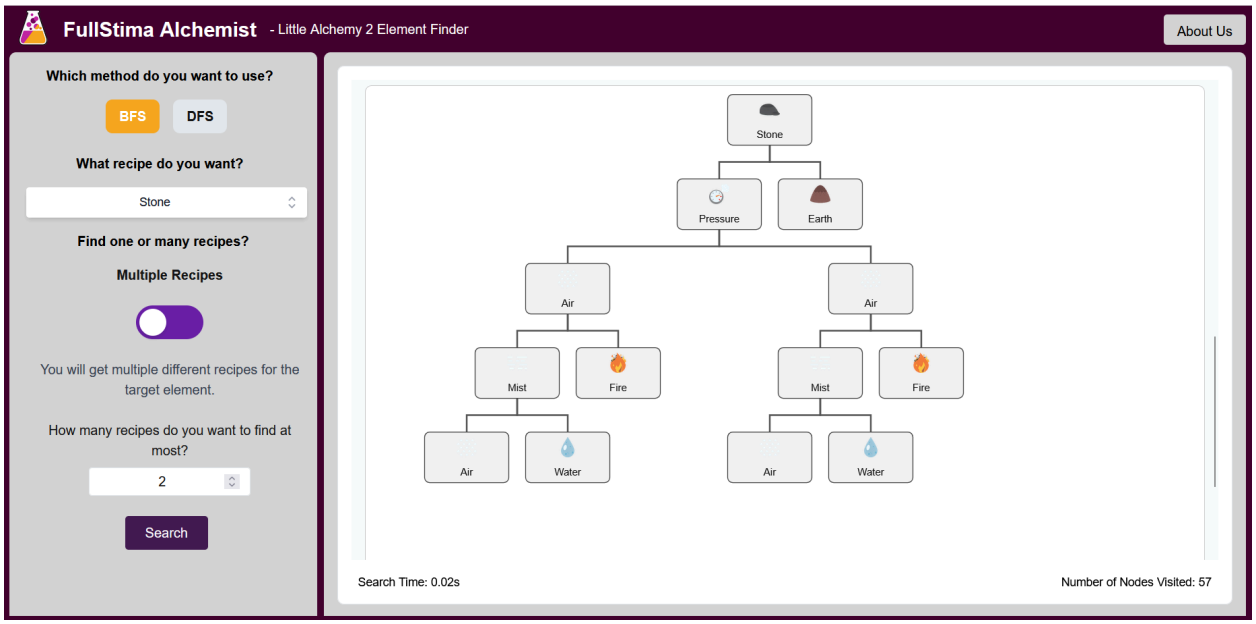
   a. Pengujian menggunakan Metode BFS Single



Gambar 8 Hasil pengujian menggunakan Metode BFS Single

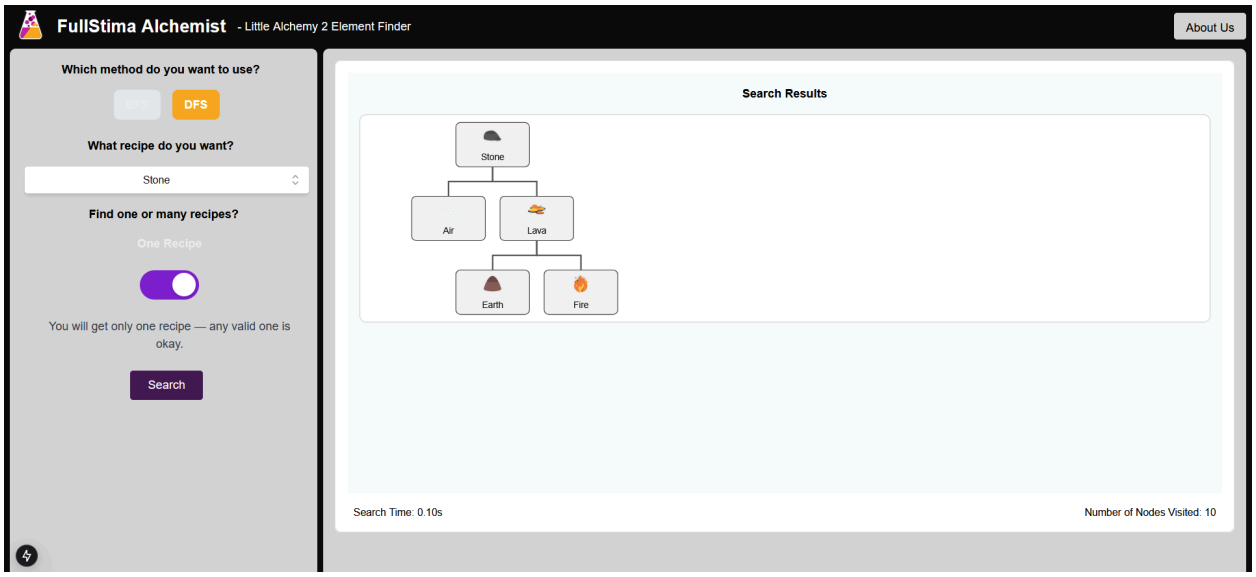
   b. Pengujian menggunakan Metode BFS Multiple





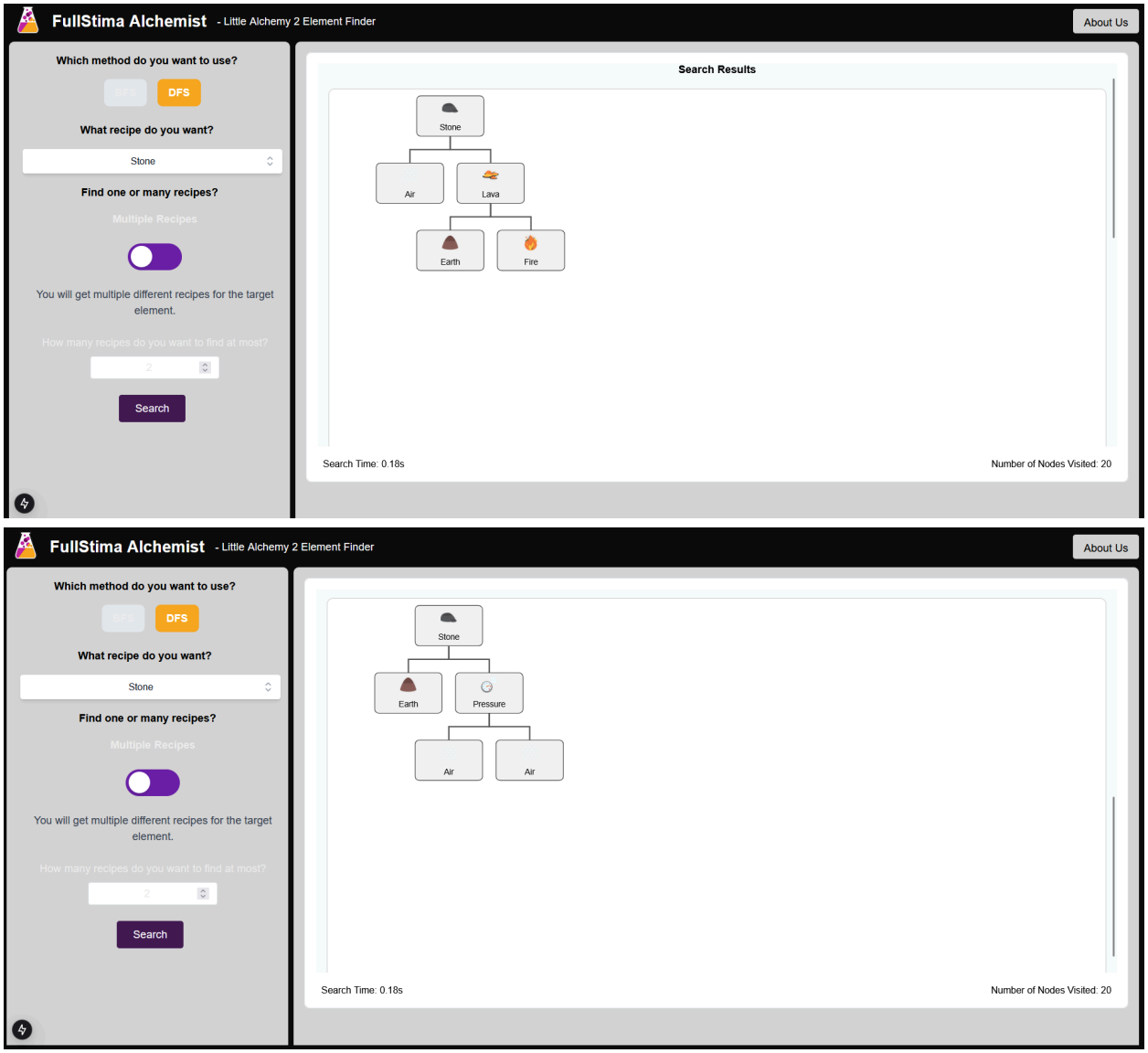
Gambar 9 Hasil pengujian menggunakan Metode BFS Multiple

c. Pengujian menggunakan Metode DFS Single



Gambar 10 Hasil pengujian menggunakan Metode DFS Single

d. Pengujian menggunakan Metode DFS Multiple

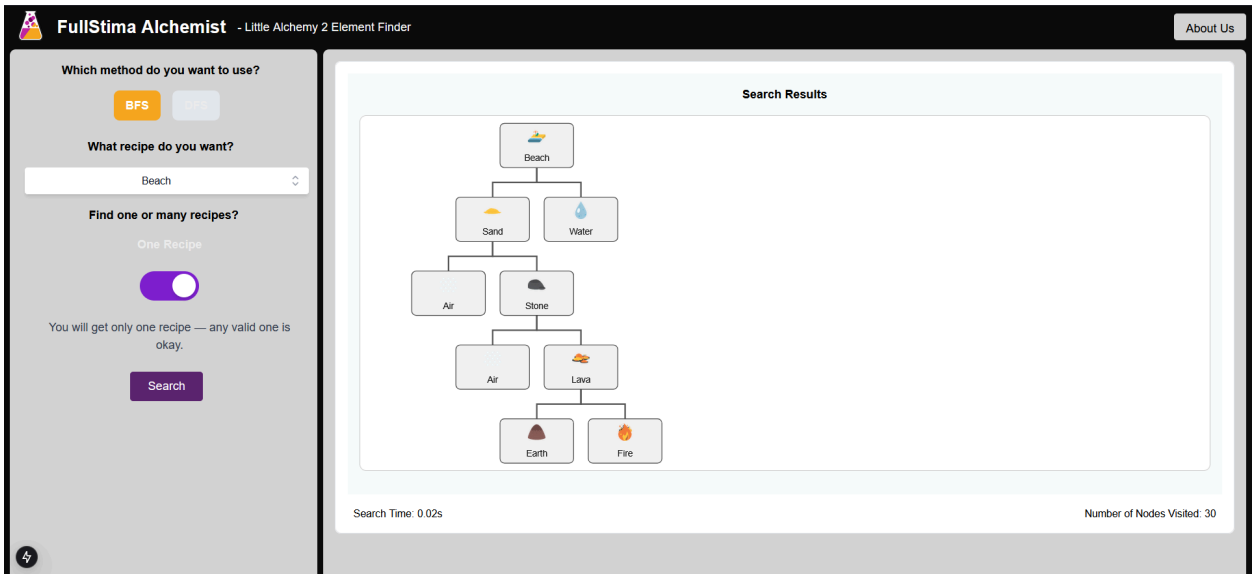


Gambar 11 Hasil pengujian menggunakan Metode DFS Multiple

2. Beach

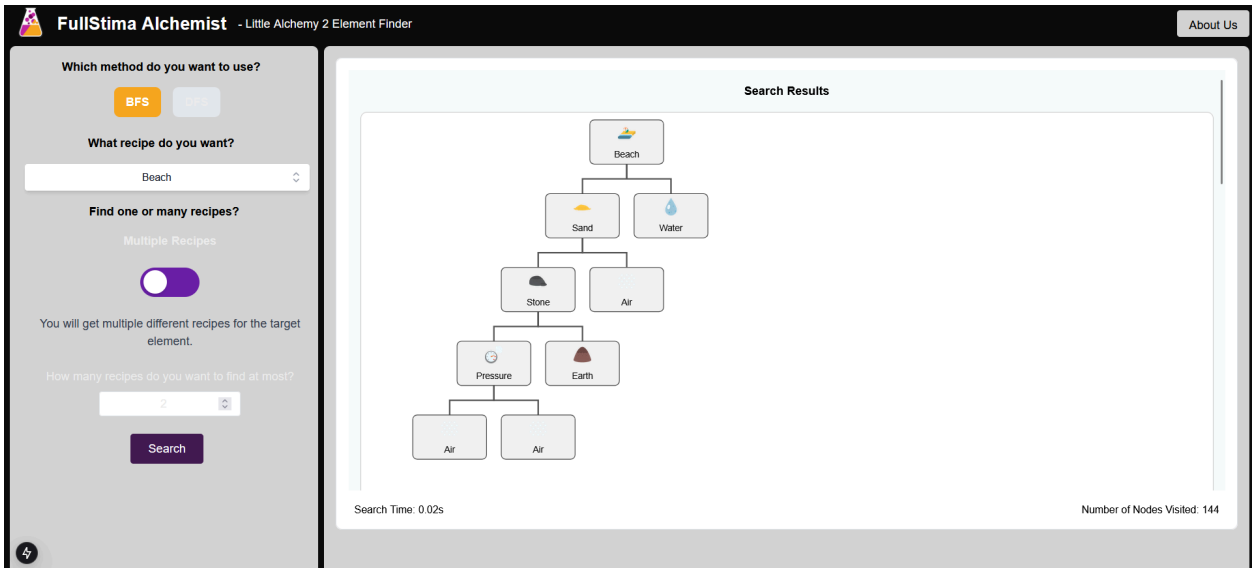
Berikut merupakan hasil pengujian pada *website* untuk membentuk elemen Beach.

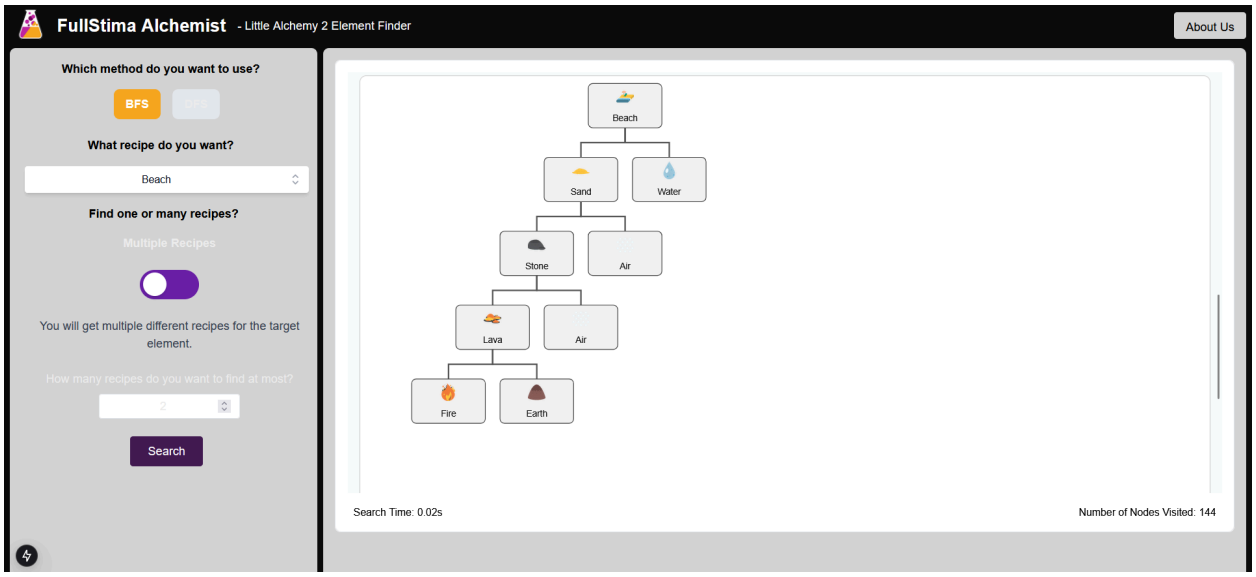
a. Pengujian menggunakan Metode BFS Single



Gambar 12 Hasil pengujian menggunakan Metode BFS Single

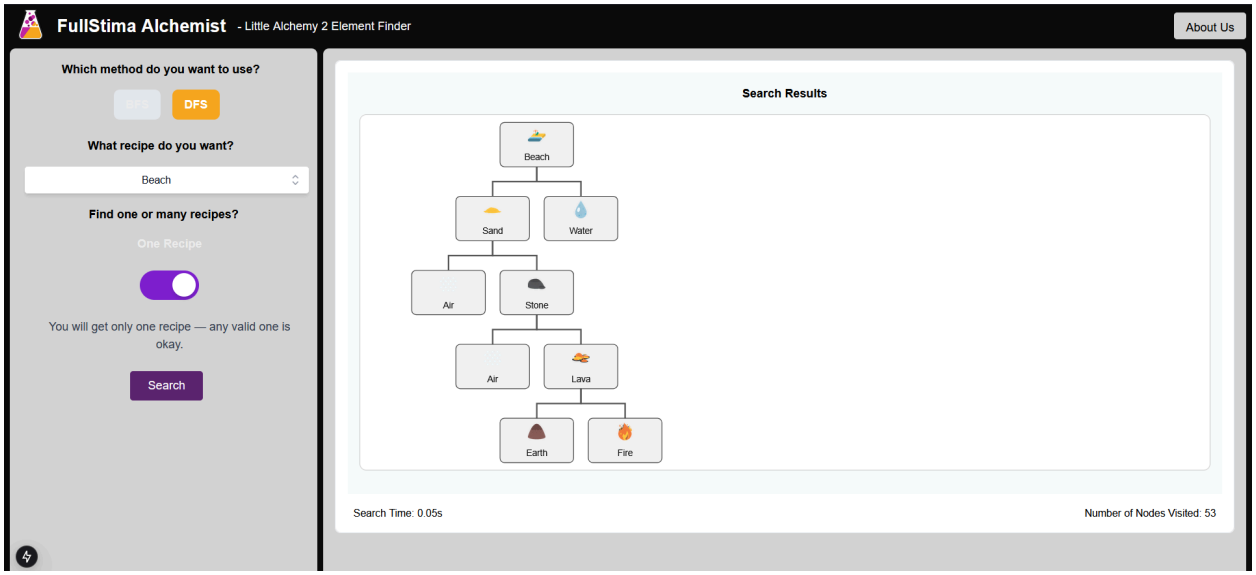
b. Pengujian menggunakan Metode BFS Multiple





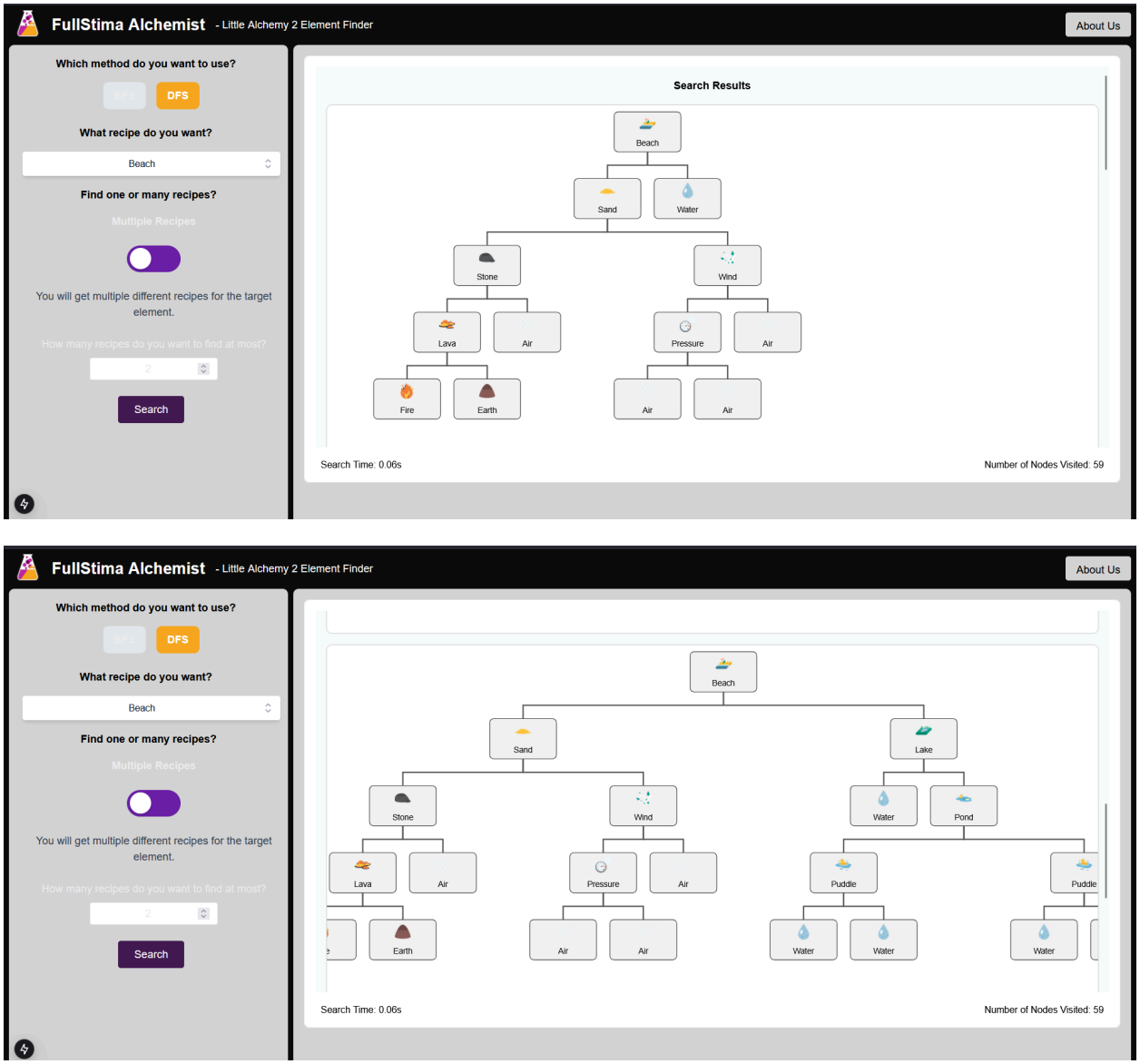
Gambar 13 Hasil pengujian menggunakan Metode BFS Multiple

c. Pengujian menggunakan Metode DFS Single



Gambar 14 Hasil pengujian menggunakan Metode DFS Single

d. Pengujian menggunakan Metode DFS Multiple

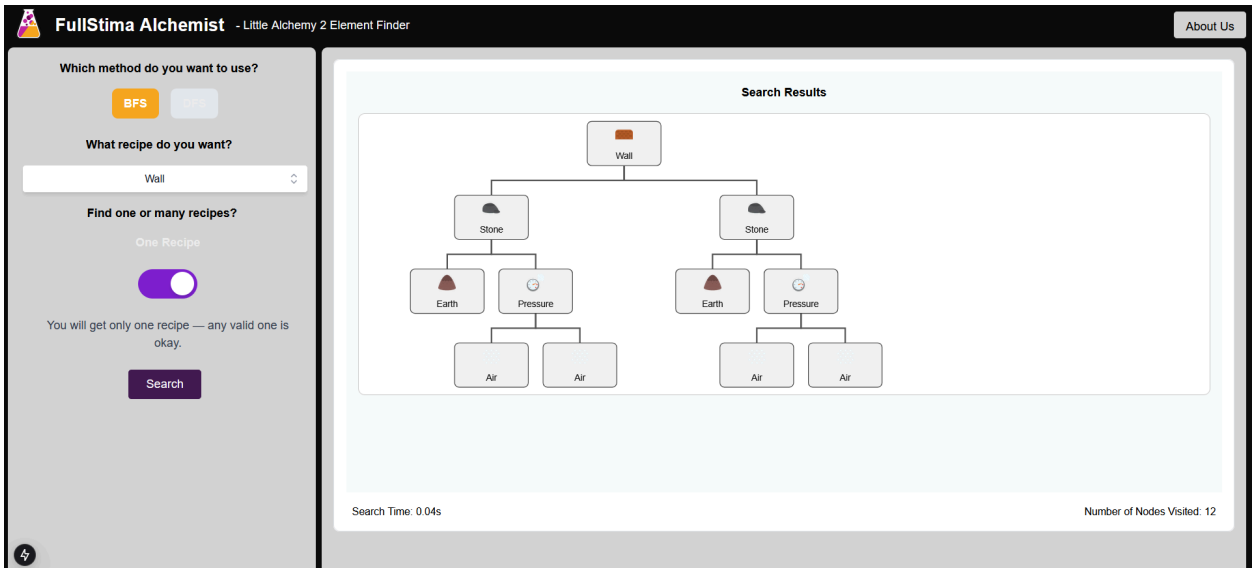


Gambar 15 Hasil pengujian menggunakan Metode DFS Multiple

3. Wall

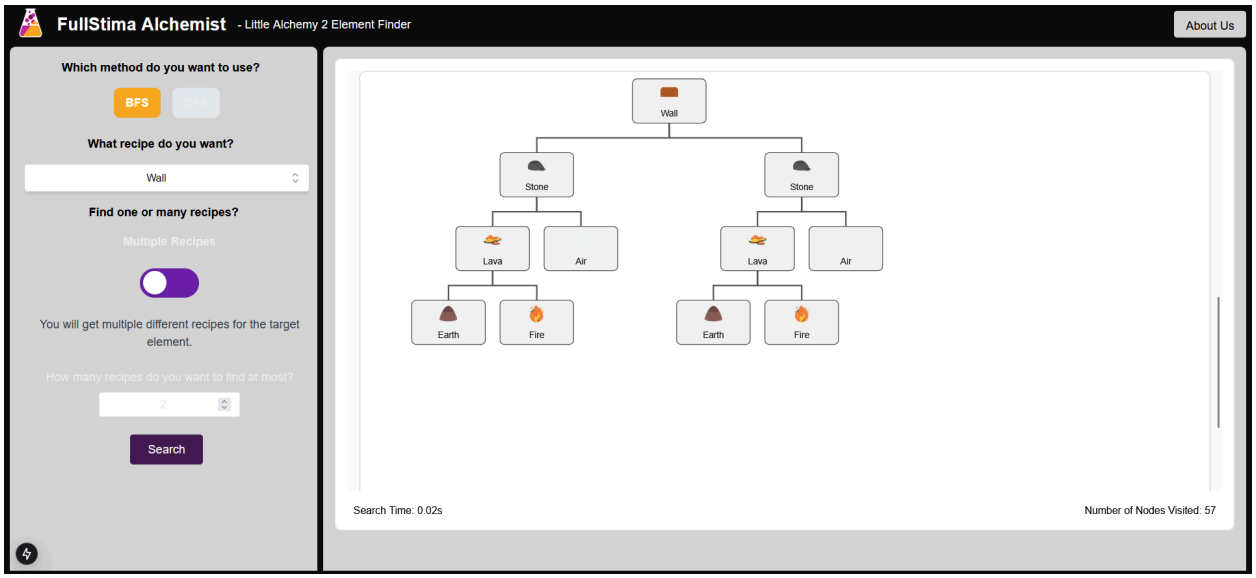
Berikut merupakan hasil pengujian pada *website* untuk membentuk elemen Wall.

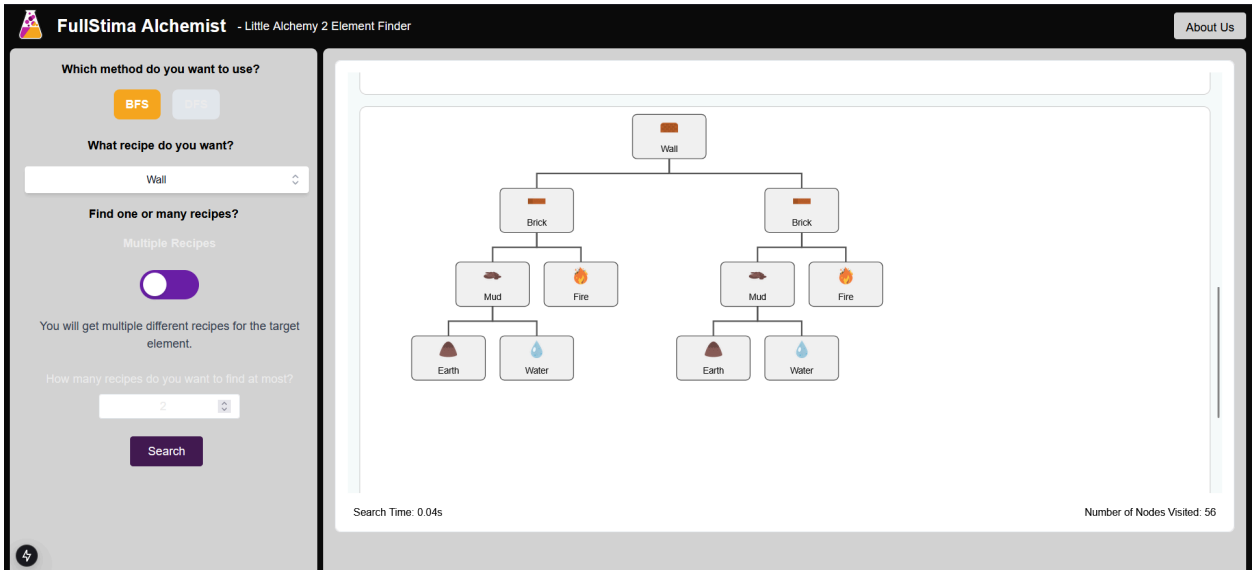
a. Pengujian menggunakan Metode BFS Single



Gambar 16 Hasil pengujian menggunakan Metode BFS Single

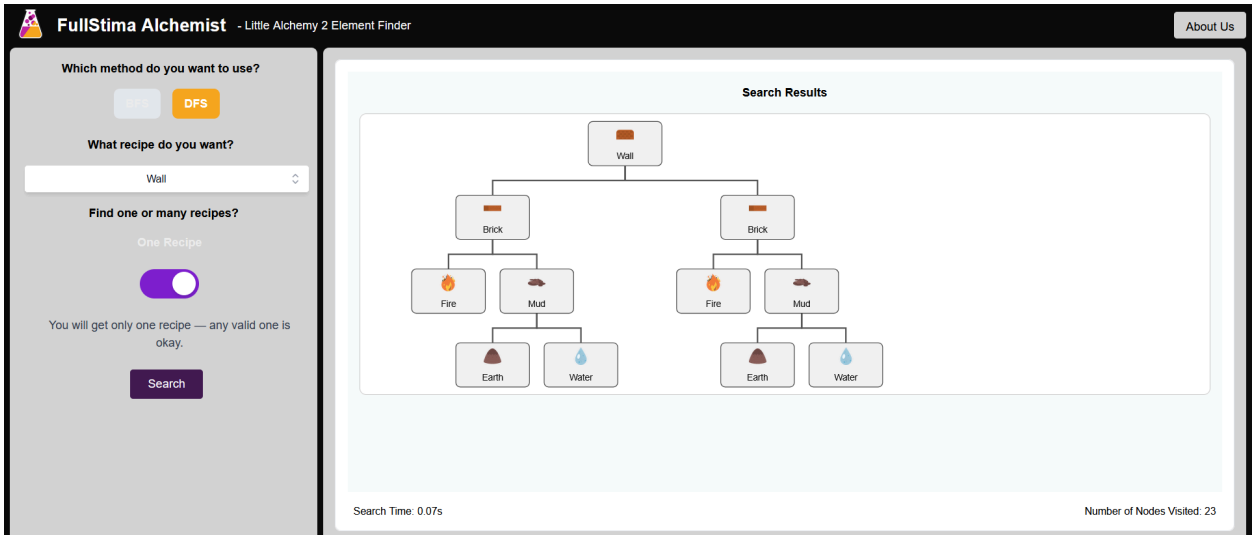
b. Pengujian menggunakan Metode BFS Multiple





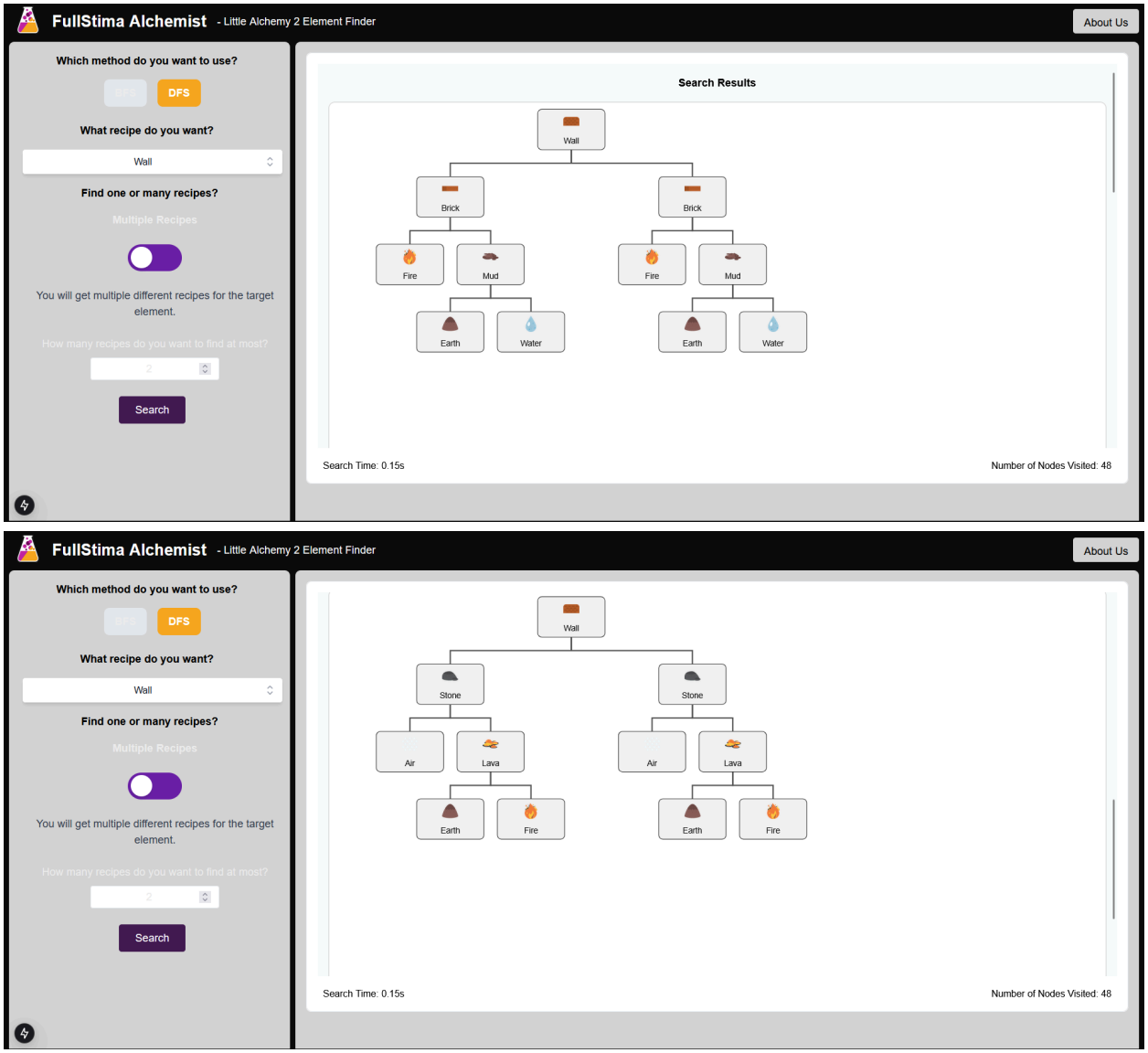
Gambar 17 Hasil pengujian menggunakan Metode BFS Multiple

c. Pengujian menggunakan Metode DFS Single



Gambar 18 Hasil pengujian menggunakan Metode DFS Single

d. Pengujian menggunakan Metode DFS Multiple



Gambar 19 Hasil pengujian menggunakan Metode DFS Multiple

4.4 Analisis Pengujian

1. Stone

   Berikut merupakan analisis hasil pengujian pada *website* untuk membentuk elemen Stone.

   a. Pengujian menggunakan Metode BFS Single

      Pengujian menggunakan Metode BFS Single berhasil untuk menemukan 1 resep yang membutuhkan 0.02 detik dan melewati *node* sejumlah 2.

   b. Pengujian menggunakan Metode BFS Multiple

Pengujian menggunakan Metode BFS Multiple berhasil untuk menemukan 2 resep yang membutuhkan 0.02 detik dan melewati *node* sejumlah 57.

c. Pengujian menggunakan Metode DFS Single

Pengujian menggunakan Metode DFS Single berhasil untuk menemukan 1 resep yang membutuhkan 0.1 detik dan melewati *node* sejumlah 10.

d. Pengujian menggunakan Metode DFS Multiple

Pengujian menggunakan Metode DFS Multiple berhasil namun dapat memakan waktu yang cukup lama berdasarkan ukuran penelusuran. Pengujian berhasil menemukan 2 resep yang membutuhkan 0.18 detik dan melewati *node* sejumlah 20.

2. Beach

Berikut merupakan analisis hasil pengujian pada *website* untuk membentuk elemen Beach.

a. Pengujian menggunakan Metode BFS Single

Pengujian menggunakan Metode BFS Single berhasil untuk menemukan 1 resep yang membutuhkan 0.02 detik dan melewati *node* sejumlah 30.

b. Pengujian menggunakan Metode BFS Multiple

Pengujian menggunakan Metode BFS Multiple berhasil untuk menemukan 2 resep yang membutuhkan 0.02 detik dan melewati *node* sejumlah 144.

c. Pengujian menggunakan Metode DFS Single

Pengujian menggunakan Metode DFS Single berhasil untuk menemukan 2 resep yang membutuhkan 0.05 detik dan melewati *node* sejumlah 53.

d. Pengujian menggunakan Metode DFS Multiple

Pengujian menggunakan Metode DFS Multiple berhasil namun dapat memakan waktu yang cukup lama berdasarkan ukuran penelusuran. Pengujian berhasil menemukan 2 resep yang membutuhkan 0.06 detik dan melewati *node* sejumlah 59.

3. Wall

Berikut merupakan analisis hasil pengujian pada *website* untuk membentuk elemen Wall.

a. Pengujian menggunakan Metode BFS Single

Pengujian menggunakan Metode BFS Single berhasil untuk menemukan 1 resep yang membutuhkan 0.04 detik dan melewati *node* sejumlah 12.

b. Pengujian menggunakan Metode BFS Multiple

   Pengujian menggunakan Metode BFS Multiple berhasil untuk menemukan 2 resep yang membutuhkan 0.02 detik dan melewati *node* sejumlah 57.

c. Pengujian menggunakan Metode DFS Single

   Pengujian menggunakan Metode DFS Single berhasil untuk menemukan 1 resep yang membutuhkan 0.04 detik dan melewati *node* sejumlah 56.

d. Pengujian menggunakan Metode DFS Multiple

   Pengujian menggunakan Metode DFS Multiple berhasil namun dapat memakan waktu yang cukup lama berdasarkan ukuran penelusuran. Pengujian berhasil menemukan 2 resep yang membutuhkan 0.15 detik dan melewati *node* sejumlah 48.

Berdasarkan hasil pengujian, dapat dilihat perbedaan dari metode BFS dan DFS. Metode BFS mengembalikan resep yang lebih besar seperti yang dilihat pada resep pembuatan Stone sedangkan DFS mengembalikan resep yang lebih dalam. Kelebihan BFS muncul ketika mencari resep yang "dangkal" karena BFS mengeksplorasi *node* berdasarkan level atau tetangganya. Dengan DFS, *node* yang dikunjunginya bisa saja lebih banyak dikarenakan DFS eksplorasi *node* secara mendalam dan bisa saja mencari node yang tidak berguna. Yang menjadi kekurangan BFS adalah ketika diberikan data yang sangat besar. BFS menggunakan memori yang cukup besar apabila diberikan graf yang sangat besar dan bisa saja membuat program *crash*. Tidak hanya itu, apabila solusi jauh di dalam graf, BFS akan memakan waktu yang cukup lama dan beda dengan DFS yang bisa saja mencarinya dalam waktu yang lumayan baik.

Selain itu, waktu pencarian untuk BFS dan DFS bergantung dengan seberapa kompleks resep elemen penyusunnya, semakin banyak elemen penyusunnya maka waktu pencarian yang dibutuhkan juga akan semakin lama, sebaliknya apabila resep disusun oleh lebih sedikit elemen maka waktu pencarian yang dibutuhkan juga akan semakin sedikit.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pengerjaan dan proses pengembangan web pencarian *recipe* ini, kami semakin memahami bagaimana cara kerja penelusuran elemen yang ingin didapatkan resepnya melalui kombinasi atau penggabungan dua elemen hingga elemen target terbentuk. Kami juga mempelajari lebih dalam dua algoritma utama yang populer digunakan dalam penelusuran yakni *Breadth First Search* dan *Depth First Search*. Selain itu, kami memperoleh wawasan lebih dalam mengenai bagaimana implementasi kedua algoritma dan optimalisasi keduanya menggunakan *multithreading* (meskipun belum pernah diajarkan di perkuliahan). Proses ini juga memberikan pemahaman lebih lanjut akan pengembangan web yang fungsional.

5.2. Saran

Program dan web yang kami buat masih jauh dari sempurna. Untuk kedepannya, dapat lebih memperhatikan struktur program, flow program yang lebih jelas, dan juga menyisihkan waktu lebih banyak untuk bug fixing secara total. Selain itu, diharapkan spesifikasi tugas besar yang lebih terstruktur agar menghindari revisi/perbaikan spesifikasi mendekati tanggal pengumpulan tugas besar.

5.3. Komentar

Kami mengucapkan terima kasih kepada asisten-asisten yang bertanggung jawab atas pelaksanaan tugas besar ini dan dosen-dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan wawasan dan pengetahuan kepada kami saat perkuliahan. Kami juga mengucapkan terima kasih kepada mahasiswa Teknik Informatika ITB lain yang telah memberikan semangat dan dukungan sehingga pada akhirnya tugas besar ini dapat selesai dengan baik.

5.4. Refleksi

Tugas besar ini telah memberikan kami pengalaman yang sangat mengesankan dari senang hingga tekanan. Kami menghadapi berbagai macam kendala seperti alokasi waktu yang kurang efektif akibat kesibukannya masing-masing, masih kurangnya pengalaman dalam membuat web, dan kinerja dari kami sendiri yang kurang baik. Untuk kedepannya, diharapkan tugas besar ini

bisa menjadi motivasi untuk kami agar kami lebih berusaha baik dalam pengerjaan tugas-tugas berikutnya.

LAMPIRAN

Link Repository GitHub:

https://github.com/angelinaefrina/Tubes2_FullStima-Alchemist

Link video :

https://youtu.be/zUrnzlA6O3E?si=qJ-MEvgyvLKtU76u

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

DAFTAR PUSTAKA

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

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

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf

[4] https://littlealchemy2.com/

[5] https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)