

## **LAPORAN TUGAS BESAR 2**

### **IF2211 STRATEGI ALGORITMA**

Pemanfaatan Algoritma BFS, DFS, dan Bidirectional dalam Pencarian  
Recipe pada Permainan Little Alchemy 2



Disusun oleh :

Alvin Christopher Santausa	13523033
Kenneth Poenadi	13523040
Ivan Wirawan	13523046

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESHA 10, BANDUNG 40132**  
**2025**

## **Daftar Isi**

Daftar Isi.....	1
Deskripsi Tugas.....	2
Landasan Teori.....	3
A. Dasar Teori.....	3
a. Penjelajahan Graf.....	3
b. Algoritma Breadth First Search.....	3
c. Algoritma Depth First Search.....	3
d. Algoritma Bidirectional Search.....	4
B. Penjelasan Singkat Aplikasi Web.....	4
Analisis Pemecahan Masalah.....	6
A. Langkah Pemecahan masalah.....	6
B. Proses Pemetaan Masalah.....	6
C. Fitur Fungsional dan Arsitektur Aplikasi Web.....	7
D. Contoh Ilustrasi Kasus.....	8
Implementasi dan Pengujian.....	19
A. Spesifikasi Teknis Program.....	19
B. Tata Cara Penggunaan Program.....	30
C. Pengujian dan Hasil.....	32
D. Analisis Hasil Pengujian.....	41
Penutup.....	42
A. Kesimpulan.....	42
B. Saran.....	42
C. Refleksi.....	42
Lampiran.....	43

## BAB I

### Deskripsi Tugas

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.

2. Elemen turunan

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

3. Combine Mechanism

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

## BAB II

### Landasan Teori

#### A. Dasar Teori

##### a. Penjelajahan Graf

Dalam penyelesaian masalah melalui algoritma komputasi, sebuah masalah seringkali direpresentasikan sebagai sebuah graf. Definisi dari graf sendiri adalah struktur data yang terdiri dari titik dan sisi yang menghubungkan titik-titik tersebut. Sebuah graf merepresentasikan objek diskrit dan hubungan diantaranya.

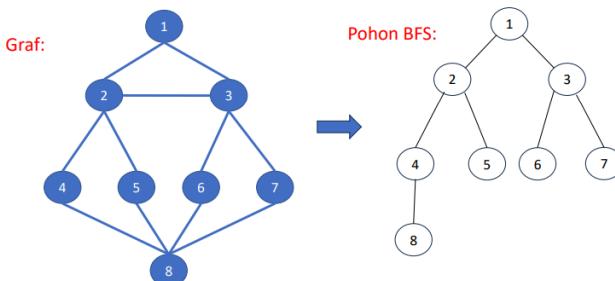
Contoh masalah yang dapat direpresentasikan dalam graf adalah peta dengan berbagai jalur. Pencarian jalur terdekat dapat dilakukan dengan analisis graf. Selain itu, graf yang berbentuk pohon merepresentasikan sebuah ruang status dalam eksperimen diskrit. Contohnya adalah ruang status dari permainan 8-Puzzle, catur, dan sudoku.

Dalam menentukan solusi yang tepat, seluruh simpul harus dikunjungi dengan cara yang sistematik. Algoritma traversal graf yang umum digunakan adalah pencarian melebar (*breadth first search/BFS*) dan pencarian mendalam (*depth first search/DFS*). Kedua algoritma ini dapat diterapkan pada pencarian solusi tanpa informasi di dalam sebuah graf statis.

##### b. Algoritma Breadth First Search

Algoritma pencarian melebar yang dimulai dari simpul v akan dilakukan melalui prosedur berikut :

1. Kunjungi simpul v (simpul pusat).
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.



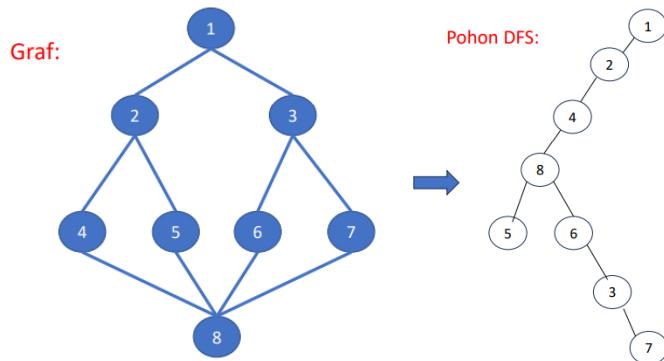
Algoritma ini berfokus pada pencarian simpul yang bertetangga dari simpul utama terlebih dahulu. Dalam prosesnya, hasil pencarian akan berbentuk melebar karena merepresentasikan percabangan simpul utama.

##### c. Algoritma Depth First Search

Algoritma pencarian mendalam yang dimulai dari simpul v akan dilakukan melalui prosedur berikut :

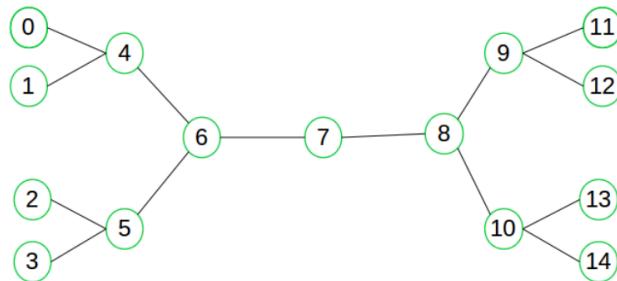
1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Algoritma ini berfokus untuk menemukan simpul akhir (simpul daun) dari tetangga pertama dari simpul utama.



#### d. Algoritma Bidirectional Search

Berbeda dengan BFS dan DFS yang pencarinya dilakukan searah dimulai dari simpul akar, algoritma *bidirectional search* akan mencari jalur dengan pencarian maju dari akar ke tujuan dan pencarian mundur dari tujuan ke akar. Tujuan utamanya adalah membagi pencarian graf menjadi 2 upa-graf yang lebih kecil. Melalui pembagian ini, jumlah eksplorasi yang diperlukan akan berkurang. Ketika pencarian dari kedua graf tersebut bertemu, maka proses pencarian akan selesai.



## B. Penjelasan Singkat Aplikasi Web

Aplikasi web yang dibangun berbasis kepada React + TypeScript + Vite. Bahasa pemrograman yang digunakan dalam aplikasi ini adalah JavaScript dalam bentuk TypeScript. Library yang digunakan adalah React untuk membangun antarmuka grafis. Sedangkan Vite merupakan *build tool* untuk pengembangan aplikasi web berbasis JavaScript.

Struktur dari direktori file *front end* terdiri dari beberapa file yaitu App.css, App.tsx, index.css, dan main.tsx. Untuk file App merupakan komponen utama dari aplikasi. Seluruh konten dan penataan serta hubungan dengan bagian *back end* diatur dalam file tersebut. Desain antarmuka menggunakan file CSS pada App.css. Namun, CSS yang digunakan dalam aplikasi ini Tailwind CSS. Selain itu ada file index.css yang berisi file CSS global, dan main.tsx untuk entry point dan bootstrapper aplikasi React

## BAB III

### Analisis Pemecahan Masalah

#### A. Langkah Pemecahan masalah

Dalam pencarian *recipe* untuk setiap elemen dari permainan Little Alchemy 2, langkah pemecahan dibagi menjadi beberapa bagian. Tahapan pertama adalah melakukan *scraping* kepada data-data yang tersedia untuk *recipe* setiap elemennya. *Scraping* ini dilakukan menggunakan bahasa pemrograman Go, sama dengan bahasa pemrograman algoritma *back end*. Hasil dari *scraping* pada Website [https://little-alchemy.fandom.com/wiki/Elements\\_\(Little\\_Alchemy\\_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)) berisikan nama elemen, *recipe* dan setiap bahannya (bisa lebih dari satu), tingkatan, dan *file* gambarnya.

Data hasil *scraping* ini akan dimuat terlebih dahulu ke dalam program. Struktur data yang dibentuk sedemikian rupa agar algoritma pencarian dapat berjalan. Sebuah elemen akan menjadi sebuah Node dan bahan-bahan pembuatannya menjadi Sisi dan membentuk sebuah pohon. Setelah itu, algoritma pencarian *recipe* akan dijalankan. Jenis algoritma yang dijalankan bergantung kepada masukan dari pengguna. Namun, pilihan yang tersedia mencakup *Breadth First Search* (BFS), *Depth First Search* (DFS), dan *Bidirectional*.

Ketiga jenis algoritma ini digunakan dalam mencari jalur *recipe* dari sebuah elemen hingga dapat dibentuk dari elemen-elemen dasar. Pengubahan bentuk struktur data menjadi sebuah pohon akan memberikan kemampuan untuk penerapan algoritma BFS, DFS, dan *Bidirectional* pada permasalahan ini. Secara umum, setiap elemen akan memiliki elemen pembentuknya. Kemudian, setiap elemen pembentuk pasti memiliki elemen pembentuknya lagi hingga sampai ke elemen dasar. Maka dari itu, program ini akan melakukan pencarian secara rekursif hingga ke elemen dasar dalam menyelesaikan masalah pencarian *recipe* sebuah elemen.

#### B. Proses Pemetaan Masalah

Dalam masalah ini, pemetaan pokok masalah untuk algoritma penyelesaiannya dilakukan dengan mengubah struktur datanya menjadi sebuah pohon. Setiap elemen adalah node yang memiliki 2 buah sisi, yaitu 2 elemen pembentuknya. Tujuan akhir yang diinginkan dalam adalah bagaimana sebuah elemen yang tidak dasar dapat dicari pembentuknya hingga mencapai elemen dasar. Perbedaan yang terdapat pada ketiga jenis algoritma yang dapat digunakan adalah pendekatan pencarinya.

Algoritma *Breadth First Search* (BFS) adalah algoritma yang berfokus pada pencarian lebar pada sebuah pohon. Maksudnya adalah setiap elemen awal akan dicari seluruh elemen pembentuk pertama sebelum pencarian elemen pembentuk kedua dari pembentuk pertama. Jadi, algoritma ini akan menghasilkan pohon status yang melebar bukan mendalam. Fokusnya adalah menemukan *recipe* terpendek dari pencarian melebar.

Dalam program ini sendiri, pencarian akan dimulai dari elemen target yang ingin dibuat. Agar dapat berbentuk BFS, maka program ini menggunakan antrian resep sebagai basisnya. Setiap resep pembentuk elemen akan ditambahkan pada antrian yang telah

diatur agar berfokus pada BFS. Isi dari antrian akan diperiksa apakah bahannya merupakan elemen dasar. Jika seluruh bahannya merupakan elemen dasar, maka pencarian akan selesai. Namun, jika tidak maka resep-resep untuk membuat bahan tersebut akan ditambahkan lagi ke antrian dan pencarian dilanjutkan secara rekursif.

Berbeda dengan BFS, algoritma *Depth First Search* (DFS) berfokus pada pencarian mendalam. Jadi, elemen pembentuk dari sebuah elemen awal akan dicari bahannya hingga mendapatkan elemen dasar. Secara dasar, algoritma ini tetap berawal dari elemen yang ingin dicari. Pencarian akan dilakukan secara rekursif selagi menambahkan jalur *recipe* apabila telah mendapatkan elemen dasar sebagai pembentuk. Jadi, algoritma ini berfokus untuk mengeksplorasi sebuah jalur hingga mencapai elemen dasar sebelum berpindah ke elemen selanjutnya.

Algoritma *Bidirectional* merupakan satu-satunya algoritma pemecahan masalah yang digunakan dengan pendekatan yang berbeda. BFS dan DFS mengusung konsep *top-down* yang berarti dimulai dari elemen yang dicari hingga elemen dasar. Namun, *Bidirectional* melakukan pencarian dari atas dan bawah secara persamaan. Seluruh hasil akan disimpan sementara apabila belum ditemukan titik temu. Ketika adanya sebuah pertemuan, elemen yang sama yang dikunjungi dari dua arah, maka jalur tersebut akan disambungkan dan dikembalikan sebagai solusi.

### C. Fitur Fungsional dan Arsitektur Aplikasi Web

Aplikasi web yang telah dikembangkan mencakup berbagai fitur dasar dalam menyelesaikan masalah yang diangkat. Halaman pada aplikasi ini hanya ada satu yang mencakup beberapa bagian. Bagian pertama adalah blok untuk memberi masukan melalui *keyboard* untuk elemen target yang ingin dicari. Pengguna dapat mengetik nama elemen yang ingin dicari *recipe*-nya. Setelah itu, blok kedua adalah pilihan hasil representasi *recipe* yang diinginkan pengguna. Pilihan yang tersedia adalah sebuah *recipe* bebas atau banyak *recipe* sekaligus. Untuk pilihan banyak *recipe* sekaligus, pengguna dapat memasukkan parameter banyak *recipe* maksimal yang ingin dicari. Kemudian, untuk bagian algoritma, pengguna dapat memilih algoritma pencarian yang ingin digunakan baik itu BFS, DFS dan *Bidirectional*. Pengguna dapat menjalankan algoritma dengan mengklik tombol *Find Recipes* yang tersedia pada halaman Web. Setelah pencarian selesai, program akan menampilkan waktu yang dibutuhkan, jumlah simpul yang ditemui, dan banyaknya pohon yang ditemukan selama pencarian. Tidak hanya itu, program juga akan menampilkan visualisasi dari *recipe* lengkap dengan gambar setiap elemen agar pengguna dapat melihat pembentukan elemen secara jelas dan detail.

Aplikasi Web ini sudah terintegrasi menggunakan *Application Programming Interface* (API) kepada bagian *back end*. Fungsi algoritma akan dilakukan pada bagian *back end*. Sedangkan untuk *front end* akan menampilkan hasil kepada pengguna. Tahap pertamanya adalah hasil *scraping* akan dimuat ke dalam program. Setelah itu, pengguna akan memberikan masukan kepada pengguna dan menjalankan algoritma. Pencarian akan dilakukan dan hasilnya dikembalikan kepada pengguna melalui tampilan visual *front end*.

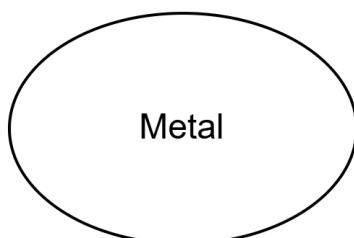
#### D. Contoh Ilustrasi Kasus

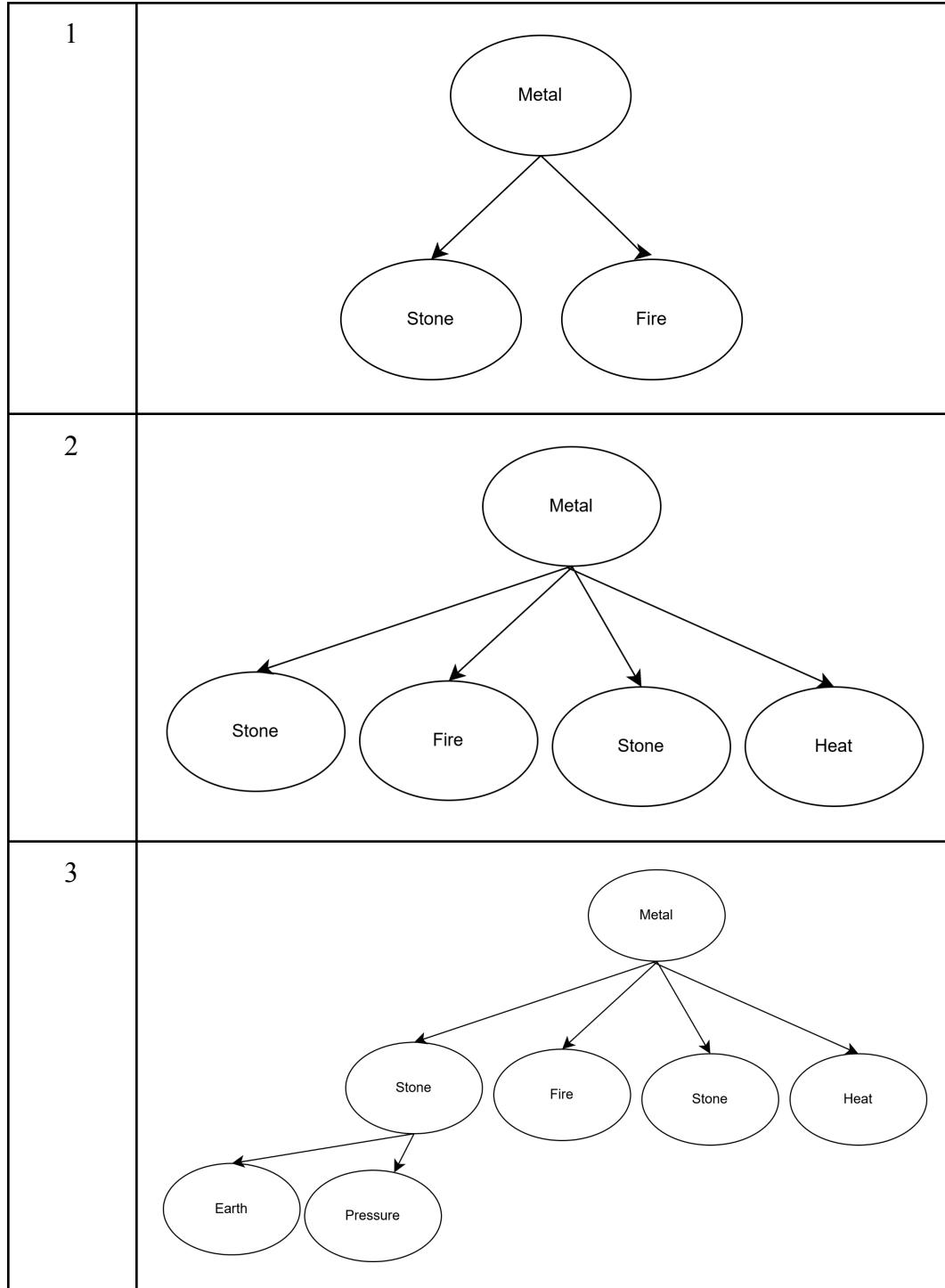
Sebagai ilustrasi kasus pencarian, target elemen yang digunakan adalah *Metal*. Elemen ini sendiri merupakan elemen pada tingkat ke 3 dengan 5 *recipes* yang berbeda. Namun, spesifikasi program ini hanya akan melakukan pencarian untuk elemen pembentuk yang berada pada tingkat yang lebih rendah, maka ada beberapa *recipe* yang secara otomatis tidak akan dicari. Namun, untuk penjelasan pencarinya akan sesuai dengan algoritmanya yaitu :

- Algoritma *Breadth First Search* (BFS)

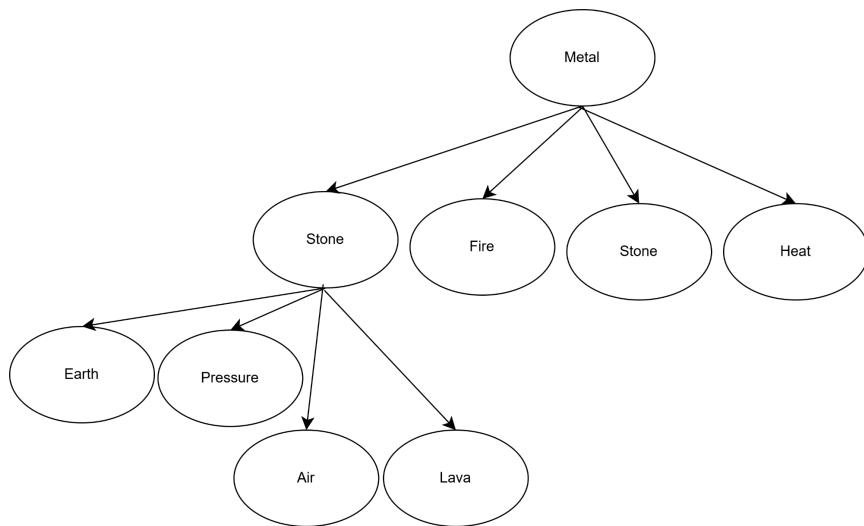
Algoritma BFS ini akan berfokus pada pencarian melebar terlebih dahulu. Jadi, setiap *recipe* dari elemen awal akan dicari. Untuk *recipe* elemen *Metal*, terdapat 2 *recipe* yang ditemukan program yaitu *Stone + Fire* dan *Stone + Heat*. Untuk algoritma BFS, maka ruang status yang akan terbentuk setelah iterasi satu tingkatan adalah elemen *Metal* memiliki 2 *recipe* tersebut. Setelah itu, pencarian akan dilanjutkan untuk elemen bawahnya dimana *Stone* akan dapat dibentuk dari *Earth + Pressure*, *Air + Lava*. Elemen *Fire* merupakan elemen dasar sehingga pencarian dihentikan dan *Heat* merupakan hasil dari *Air + Energy*.. Pencarian lanjutannya adalah *Earth* merupakan elemen dasar, *Pressure* memiliki bahan *Air + Air* yang keduanya merupakan elemen dasar, *Air* merupakan elemen dasar terakhir *Lava* dibentuk dari *Earth + Fire*. Karena sudah didapatkan elemen dasar, maka pencarian selesai. Jadi, terdapat 2 *recipe* yang diperoleh dengan tingkat kedalaman yang sama yaitu *Metal* yang dibangun dari *Stone + Heat*. *Stone* yang terbentuk dari *Air + Lava* dan *Lava* yang dibentuk oleh *Earth + Fire*. Kemudian *Heat* yang dibentuk dari *Air* dan *Energy* dan *Energy* hasil dari *Fire + Fire*. *Recipe* kedua yaitu *Stone + Fire* yang dimana *Stone* merupakan hasil dari *Earth + Pressure* dan *Pressure* terbentuk dari *Air + Air*.

Ruang status dari algoritma ini dapat dilihat pada tabel di bawah ini :

Urutan	Pohon Ruang Status
0	



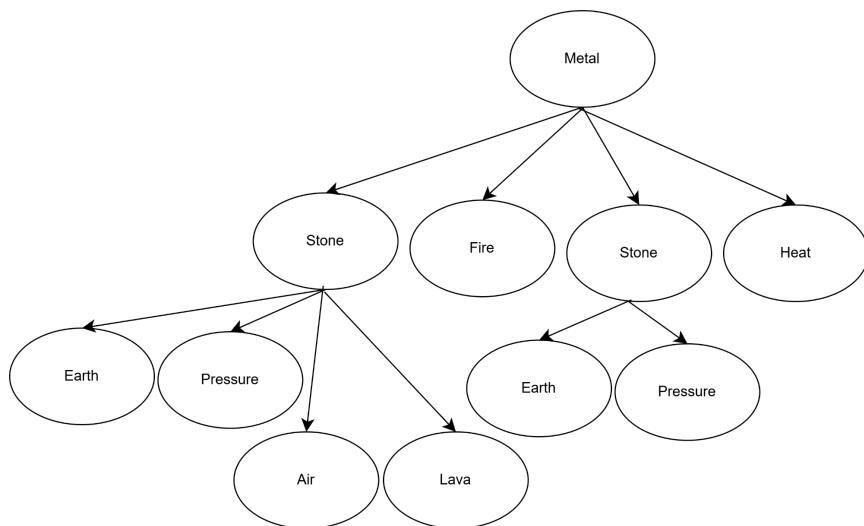
4



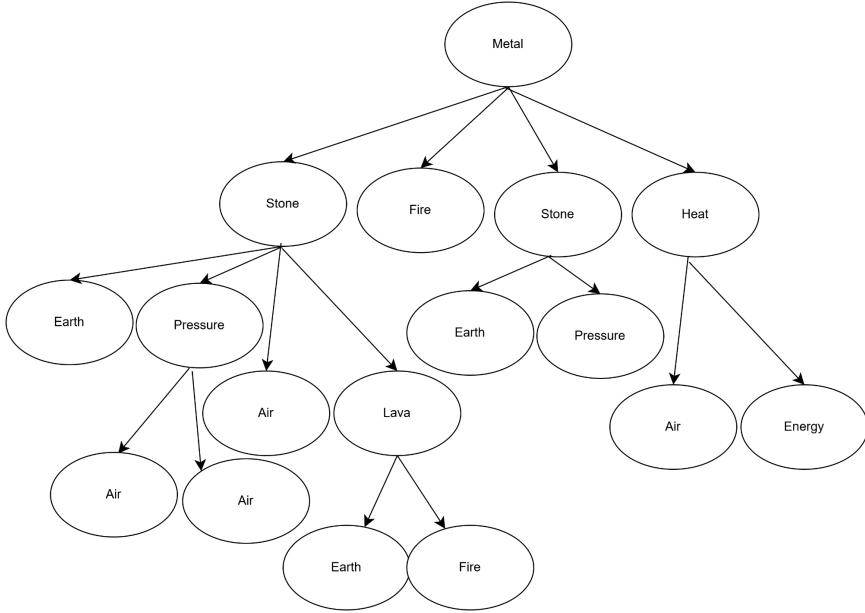
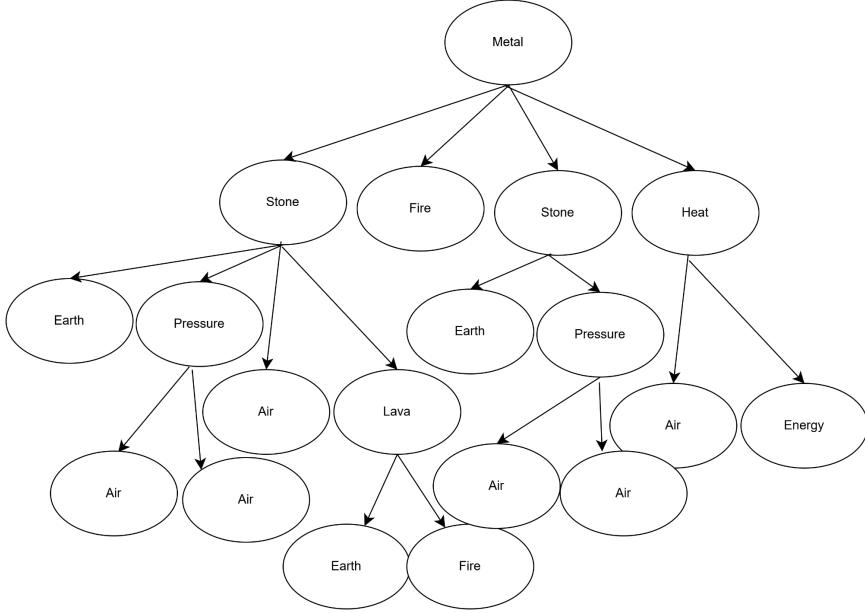
5

Fire merupakan elemen dasar

6



7	<pre>graph TD; Metal --&gt; Stone1[Stone]; Metal --&gt; Fire[Fire]; Metal --&gt; Stone2[Stone]; Metal --&gt; Heat[Heat]; Stone1 --&gt; Earth1[Earth]; Stone1 --&gt; Pressure1[Pressure]; Fire --&gt; Earth2[Earth]; Fire --&gt; Pressure2[Pressure]; Heat --&gt; Air1[Air]; Heat --&gt; Energy[Energy]; Earth1 --&gt; Air2[Air]; Pressure1 --&gt; Air2; Pressure2 --&gt; Air3[Air]; Air2 --&gt; Energy</pre>
8	Earth merupakan elemen dasar
9	<pre>graph TD; Metal --&gt; Stone1[Stone]; Metal --&gt; Fire[Fire]; Metal --&gt; Stone2[Stone]; Metal --&gt; Heat[Heat]; Stone1 --&gt; Earth1[Earth]; Stone1 --&gt; Pressure1[Pressure]; Fire --&gt; Earth2[Earth]; Fire --&gt; Pressure2[Pressure]; Heat --&gt; Air1[Air]; Heat --&gt; Energy[Energy]; Earth1 --&gt; Air2[Air]; Pressure1 --&gt; Air2; Pressure2 --&gt; Air3[Air]; Air2 --&gt; Energy</pre>
10	Air merupakan elemen dasar

11	
12	Earth merupakan elemen dasar
13	
14	Air merupakan elemen dasar

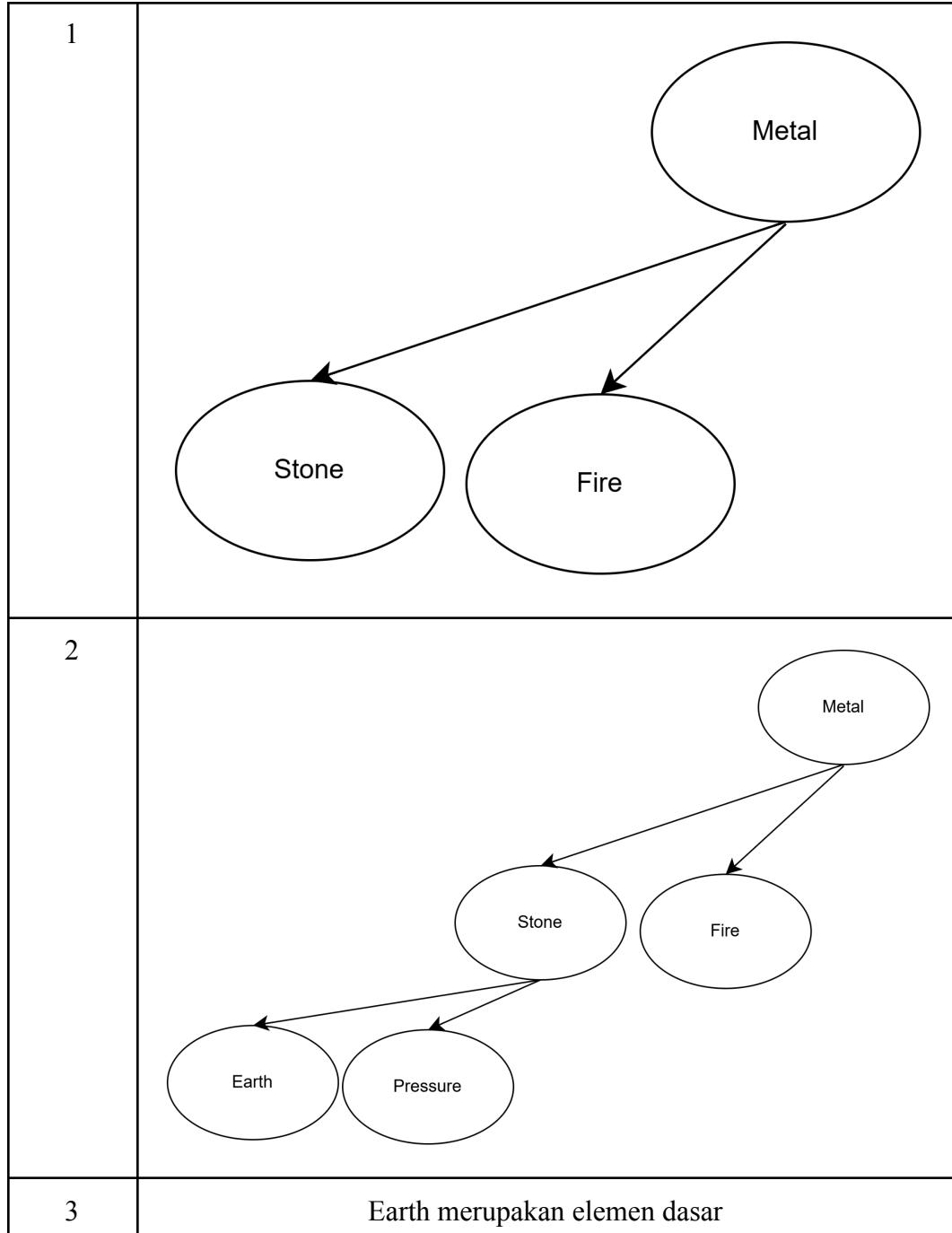
15	<pre> graph TD     Metal((Metal)) --&gt; Stone1((Stone))     Metal --&gt; Fire1((Fire))     Metal --&gt; Stone2((Stone))     Metal --&gt; Heat1((Heat))     Stone1 --&gt; Earth1((Earth))     Stone1 --&gt; Pressure1((Pressure))     Fire1 --&gt; Earth2((Earth))     Fire1 --&gt; Pressure2((Pressure))     Heat1 --&gt; Air1((Air))     Heat1 --&gt; Energy1((Energy))     Earth1 --&gt; Air2((Air))     Earth1 --&gt; Air3((Air))     Pressure1 --&gt; Air4((Air))     Pressure1 --&gt; Air5((Air))     Earth2 --&gt; Air6((Air))     Pressure2 --&gt; Air7((Air))     Air2 --&gt; Earth3((Earth))     Air2 --&gt; Fire1((Fire))     Air3 --&gt; Earth4((Earth))     Air4 --&gt; Fire2((Fire))     Air5 --&gt; Fire3((Fire))     Air6 --&gt; Fire4((Fire))     Air7 --&gt; Fire5((Fire))   </pre>
16	Seluruh elemen sisanya merupakan elemen dasar

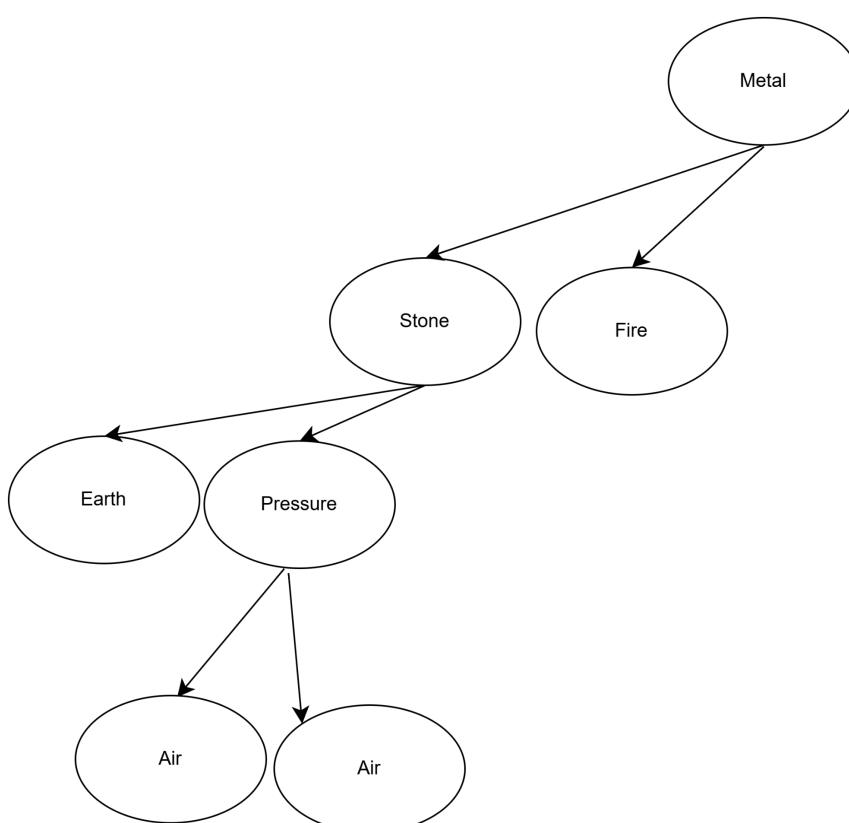
- Algoritma *Depth First Search* (DFS)

Algoritma DFS akan berfokus untuk melakukan pencarian secara mendalam. Jadi, pencarian akan dimulai dari *recipe* pertama Metal yaitu Stone + Fire. Setelah itu, akan dilakukan pencarian pada elemen Stone terlebih dahulu yaitu Earth + Pressure. Karena Earth merupakan elemen dasar, pencarian dilanjutkan pada Pressure yang terbentuk dari Air + Air. Kedua elemen merupakan elemen dasar hingga algoritma akan melakukan runut balik hingga ke elemen Fire yang sebelumnya belum dicek statusnya. Pada akhirnya, Fire merupakan elemen dasar, sehingga ditemukan sebuah *recipe* yang berisi Metal dibentuk dari Stone + Fire. Kemudian Stone dibentuk Earth + Pressure dan Pressure dibentuk oleh Air + Air.

Ruang status dari algoritma ini dapat dilihat pada tabel di bawah ini :

Urutan	Pohon Ruang Status
0	<p>Metal</p>



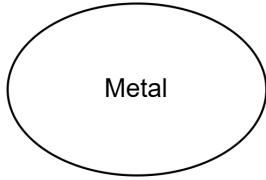
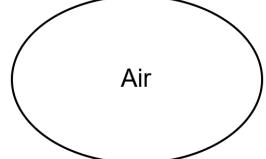
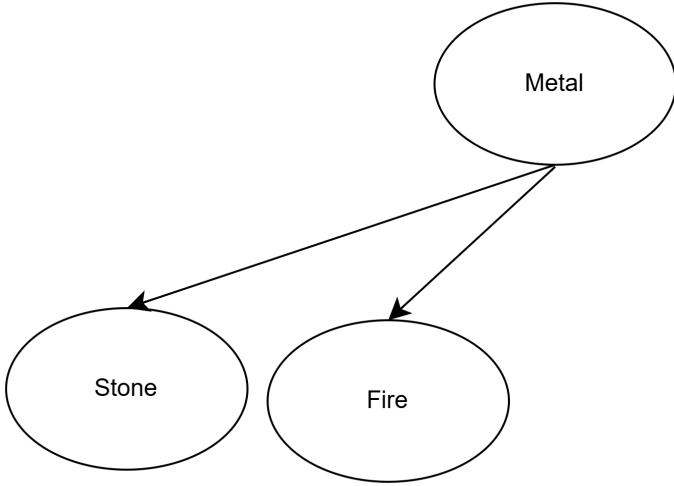
4	
5	Air merupakan elemen dasar
6	Air merupakan elemen dasar
7	Fire merupakan elemen dasar

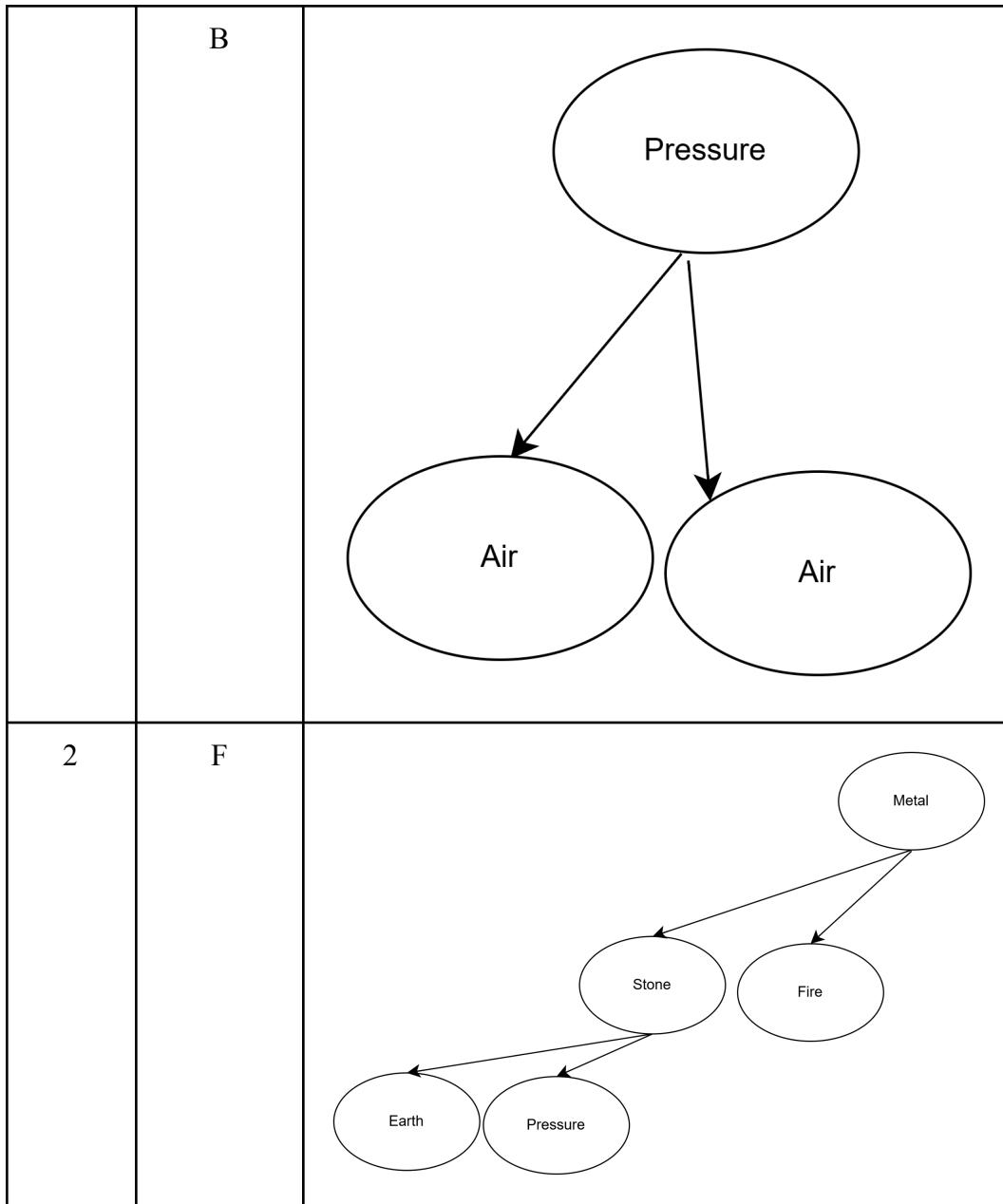
- Algoritma *Bidirectional*

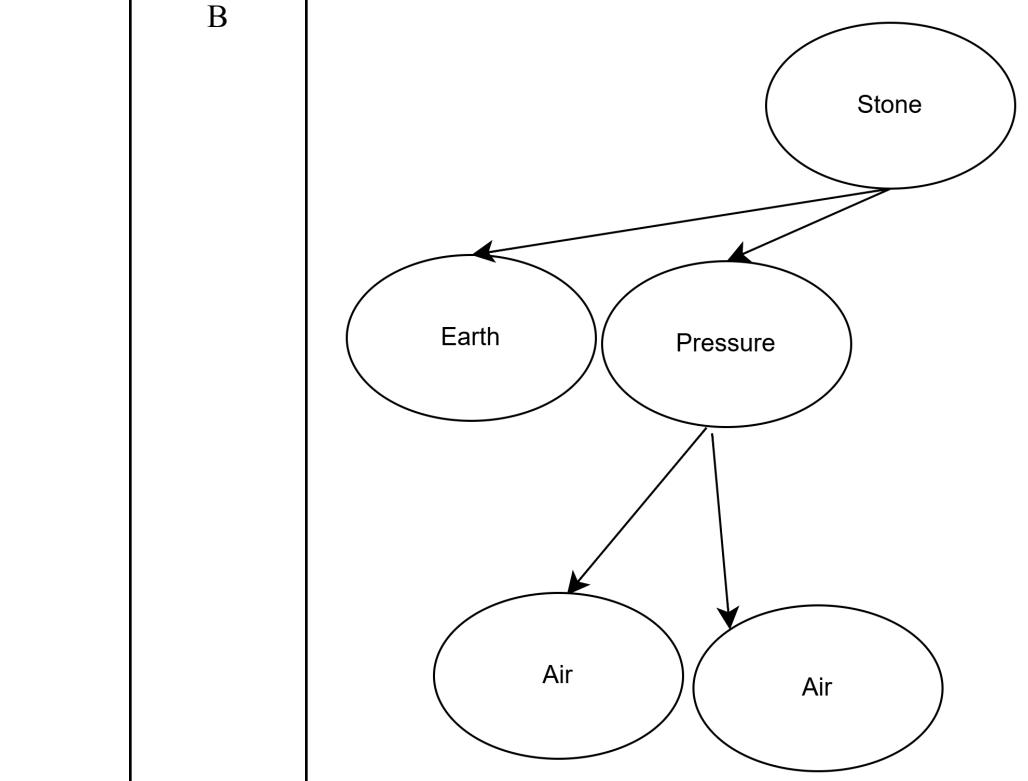
Algoritma *Bidirectional* terdiri dari 2 buah pencarian yaitu pencarian menaik dan pencarian menurun. Pencarian menaik atau *Forward* merupakan pencarian yang dilakukan dari elemen target menuju elemen dasar. Sedangkan pencarian menurun atau *Backward* dilakukan dari elemen dasar menuju elemen target. Dalam kasus *Metal*, tahap pertama adalah elemen pembentuk *Metal* adalah *Stone* dan *Fire*, untuk pencarian *Backward* terdapat *Air* di *expand* menuju *Pressure*. Tahap kedua adalah pencarian elemen dari pembentuk *Stone* yang terbentuk dari *Earth* + *Pressure* dan pencarian *Backward* membentuk *Stone* dari campuran *Earth* + *Pressure*. Maka, elemen yang diperiksa sama, sehingga pencarian akan dihentikan dan hasil jalur dikembalikan.

Ruang status dari algoritma ini dapat dilihat pada tabel di bawah ini :

Urutan	Forward /	Pohon Ruang Status
--------	-----------	--------------------

	<b>Backward</b>	
0	F	
	B	
1	F	



	B	
3	F B	Karena elemen yang sedang dicek sama, maka pencarian akan berhenti.

## BAB IV

### Implementasi dan Pengujian

#### A. Spesifikasi Teknis Program

##### 1. Struktur Data

Di dalam program ini terdapat beberapa jenis struktur data yang digunakan yaitu :

Struktur Data	Deskripsi
<pre>type Recipe struct {     Ingredients []string     `json:"ingredients"` }</pre>	Struktur data ini digunakan untuk menyimpan bahan pembentuk dari sebuah elemen. Penyimpanannya adalah dalam bentuk array of string yang menyimpan nama elemen pembentuk.
<pre>type ElementRecipe struct {     Ingredients []string     `json:"ingredients"` }</pre>	Struktur data ini mirip dengan Recipe namun akan digunakan untuk pembentukan elemen dengan beberapa resep yang berbeda.
<pre>type Element struct {     Name      string     `json:"name"`     ImagePath string     `json:"image,omitempty"`     LocalImage string     `json:"localImage,omitempty"`     "`      Recipes     []ElementRecipe     `json:"recipes,omitempty"`     Tier      int     `json:"tier"` }</pre>	Struktur data ini digunakan dalam merepresentasikan sebuah elemen pada program ini. Di dalamnya, disimpan nama elemen, letak gambar, nilai tingkat, dan daftar resep pembentuknya.
<pre>type SearchConfig struct {     TargetElement string     `json:"targetElement"`     Algorithm     string     `json:"algorithm"`     MaxResults    int     `json:"maxResults"`     SinglePath    bool }</pre>	Struktur data ini digunakan dalam algoritma pencarian kombinasi elemen. Struktur ini menyimpan nama dari target elemen, algoritma yang digunakan, jumlah resep yang diinginkan, dan pencarian <i>single</i> atau <i>multiple</i> resep.

<pre>`json:"singlePath"` }</pre>	
<pre>type SearchResult struct {     Paths      [][]Node `json:"paths"`     TimeElapsed int64 `json:"timeElapsed" ` //ms     NodesVisited int `json:"nodesVisited"` }</pre>	Struktur data ini digunakan dalam menyimpan hasil pencarian yaitu jalur kombinasi elemen pembentuk, waktu yang dibutuhkan, serta jumlah simpul yang dilalui.
<pre>type Node struct {     Element      string `json:"element"`     ImagePath    string `json:"image,omitempty"`     Ingredients []string `json:"ingredients,omitempty"`     Position     int `json:"position,omitempty"` }</pre>	Struktur data ini digunakan dalam menyimpan bentuk elemen menjadi simpul dalam pohon. Data yang disimpan adalah nama elemen, letak gambar, elemen-elemen pembentuk, dan posisi.
<pre>type Handler struct {     elements map[string]model.Element }</pre>	Struktur data ini digunakan dalam komunikasi pada API yang menyimpan data elemen beserta informasinya.
<pre>type AnimationStep struct {     StepIndex    int `json:"stepIndex"`     TotalSteps   int `json:"totalSteps"`     Node json.RawMessage `json:"node,omitempty"`     Link json.RawMessage `json:"link,omitempty"` }</pre>	Struktur data ini digunakan dalam membentuk animasi pembentukan dalam aplikasi. Data yang disimpan mencakup index tahapan, jumlah tahapan, simpul, sisi, status simpul, status penyelesaian, dan tipe.

<pre> IsBaseNode bool `json:"isBaseNode"` IsCompleted bool `json:"isCompleted"` Type string `json:"type"` } </pre>	
<pre> type queueItem struct {     recipe *graph.Recipe     path   []*model.Node } </pre>	Struktur data ini digunakan dalam algoritma pencarian BFS untuk menyimpan antrian pada pencarian.
<pre> type PathSegment struct {     Path   []model.Node     LastElem string } </pre>	Struktur data ini digunakan dalam algoritma pencarian Bidirectional untuk menyimpan informasi segmen jalur pencarian.

## 2. Fungsi dan Prosedur

Dalam mengolah data tersebut, maka dibutuhkan beberapa fungsi dan prosedur yaitu :

- Pada *Application Program Interface* (API)

Fungsi / Prosedur	Deskripsi
HandleGetElements	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk menangani permintaan elemen dari bagian <i>front end</i> melalui HTTP GET dalam bentuk JSON.
NewHandler	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk menginisiasi Handler.
HandleBFSTree	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini menangani permintaan pencarian dengan algoritma BFS dengan menghasilkan output berupa pohon representasi BFS.
HandleDFSTree	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini

	menangani permintaan pencarian dengan algoritma DFS dengan menghasilkan output berupa pohon representasi DFS
HandleBidirectionalSearch	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini menangani permintaan pencarian dengan algoritma Bidirectional dengan menghasilkan output berupa pohon representasi Bidirectional.
countNodesInTree	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk menghitung jumlah simpul dalam pohon dengan pencarian rekursif.
ensureIngredientsExpanded	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk memastikan seluruh bahan pembentuk telah dicari hingga ke elemen dasar dan akan ditandai.
convertPathToTree	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk melakukan konversi jalur hasil pencarian menjadi sebuah pohon yang akan dipresentasikan.
isTreeFullyMakeable	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi untuk mengecek apakah tree dapat dibuat.
getTopLevelRecipeSignature	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini digunakan untuk membantu adanya duplikasi resep pada pencarian.
generateDetailedTreeSignature	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini berfungsi dalam perbandingan pohon yang sama.
ensureIngredientsRandomlyExpanded	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini digunakan dalam memastikan

	pembentukan bahan pada sebuah elemen terjadi secara acak sehingga menambah variansi.
extractSubPath	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini digunakan untuk melakukan ekstraksi kepada elemen target pada pembentukan tree
min	Fungsi sederhana ini membantu untuk mengembalikan nilai yang lebih kecil dari 2 buah bilangan.
isTreeFullyTraceable	Terletak pada bagian <i>Application Program Interface</i> (API), fungsi ini digunakan untuk memastikan semua elemen dapat dibentuk.

- Pada Algoritma Pencarian (BFS, DFS, dan Bidirectional)

Fungsi / Prosedur	Deskripsi
BFS	Terletak pada bagian Algoritma Pencarian, fungsi ini adalah implementasi algoritma pencarian <i>Breadth-First Search</i> dalam pencarian resep sebuah elemen dari elemen dasar.
MultiThreadedBFS	Terletak pada bagian Algoritma Pencarian, fungsi ini melakukan pencarian BFS secara paralel untuk setiap respon elemen.
IsFullyComposablePath	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk pemeriksaan apakah elemen dalam jalur bisa ditelusuri hingga elemen dasar.
IsElementTraceable	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk pemeriksaan apakah elemen bisa ditelusuri hingga elemen dasar
GeneratePathSignature	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk memberi penanda unik untuk

	menghindari adanya duplikasi jalur pencarian
generateRecipesSignature	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk memberi penanda unik pada resep
appendPath	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk menambah simpul
deduplicatePath	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk menanggapi duplikasi.
scorePathTraceability	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk menilai jalur terpendek dari kumpulan yang didapatkan
max	Fungsi sederhana yang mengembalikan nilai terbesar dari 2 bilangan.
BidirectionalBFS	Terletak pada bagian Algoritma Pencarian, fungsi ini adalah implementasi algoritma Bidirectional berbasis BFS.
MultiThreadedBidirectionalBFS	Terletak pada bagian Algoritma Pencarian, fungsi ini melakukan pencarian dengan algoritma Bidirectional secara paralel.
HybridSearch	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk mencoba penggunaan MultiThreadedBidirectional sebelum berpindah ke BidirectionalBFS jika gagal.
ConcurrentElementSearch	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk menjalankan pencarian resep untuk beberapa elemen secara konkuren
expandForwardFrontier	Terletak pada bagian Algoritma Pencarian, fungsi ini berfungsi untuk memperluas pencarian maju dan

	memeriksa kondisi dengan pencarian mundur.
expandBackwardFrontier	Terletak pada bagian Algoritma Pencarian, fungsi ini berguna untuk memperluas pencarian mundur dan memeriksa kondisi dengan pencarian mundur.
expandForwardFrontierTargeted	Terletak pada bagian Algoritma Pencarian, fungsi ini adalah bentuk optimal dengan prioritas target kepada bahan untuk fungsi expandForwardFrontier.
expandBackwardFrontierTargetd	Terletak pada bagian Algoritma Pencarian, fungsi ini adalah bentuk optimal dengan prioritas target kepada bahan untuk fungsi expandBackwardFrontier.
mergePaths	Terletak pada bagian Algoritma Pencarian, fungsi ini berfungsi untuk menggabungkan hasil pencarian maju dan mundur
postProcessPath	Terletak pada bagian Algoritma Pencarian, fungsi ini berfungsi untuk memeriksa dan memperbaiki jalur pencarian hasil algoritma Bidirectional
validateIngridientsInPath	Terletak pada bagian Algoritma Pencarian, fungsi ini memvalidasi seluruh bahan pada jalur telah sesuai dengan ketentuan
customBidirectionalSearch	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk mencari jalur khusus saat pencarian normal tidak memberikan hasil
containsPath	Terletak pada bagian Algoritma Pencarian, fungsi ini mengecek keberadaan jalur dalam hasil
pathsEqual	Terletak pada bagian Algoritma Pencarian, fungsi ini membandingkan

	jalur pencarian
getRecipeKey	Terletak pada bagian Algoritma Pencarian, fungsi ini membuat sebuah kunci untuk resep
DFS	Terletak pada bagian Algoritma Pencarian, fungsi ini adalah implementasi pencarian dengan algoritma <i>Depth-First Search</i>
Explore	Terletak pada bagian Algoritma Pencarian, fungsi ini melakukan pencarian rekursif pada resep
MultiThreadedDFS	Terletak pada bagian Algoritma Pencarian, fungsi ini akan menjalankan beberapa pencarian DFS secara paralel
exploreWithStrategy	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan dalam pencarian DFS dengan batasan kedalaman serta prioritas jalur yang sederhana.
MultiThreadedElementTreeDFS	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan dalam menghasilkan pohon visualisasi DFS
GetElementTreeDFS	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk mengembalikan sebuah elemen pada pohon.
filterTraceableRecipes	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk menyeleksi resep yang valid
hasDuplicateIngridents	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk memeriksa duplikasi elemen pembentuk
IsBaseElement	Terletak pada bagian Algoritma Pencarian, fungsi ini digunakan untuk memeriksa apakah elemen merupakan elemen dasar.
countBaseElements	Terletak pada bagian Algoritma

	Pencarian, fungsi ini menghitung jumlah elemen dasar pada hasil pencarian.
GeneratePathSignature	Terletak pada bagian Algoritma Pencarian, fungsi ini menandai jalur pencarian secara unik.

- Fungsi Utilitas / Pembantu Tambahan

Fungsi / Prosedur	Deskripsi
NewElementGraph	Terletak pada bagian pembentukan graf, fungsi ini berfungsi untuk menginisiasi pembentukan graf elemen.
getPossibleCombinations	Terletak pada bagian pembentukan graf, fungsi ini berfungsi untuk mencari kombinasi dari dua buah elemen.
LoadElements	Terletak pada bagian pemuatan elemen, fungsi ini digunakan untuk memuat data hasil <i>scraping</i> ke dalam program.
CompareTreeIngridentsDeep	Terletak pada bagian perbandingan pohon, fungsi ini membandingkan dua buah pohon secara rekursif
DeepCopyTree	Terletak pada bagian perbandingan pohon, fungsi ini membuat salinan mendalam dari sebuah pohon
CompareTreeIngridents	Terletak pada bagian perbandingan pohon, fungsi ini membandingkan bahan pembentuk pada tingkat pertama
GeneratePathFingerprint	Terletak pada bagian perbandingan pohon, fungsi ini memberikan penanda unik untuk jalur pencarian
GenerateTreeSignature	Terletak pada bagian perbandingan pohon, fungsi ini memberikan penanda unik untuk pohon
VerifyTreeIngridentsComplete	Terletak pada bagian perbandingan pohon, fungsi ini memeriksa apakah bahan pembentuk sudah sesuai dengan resep.

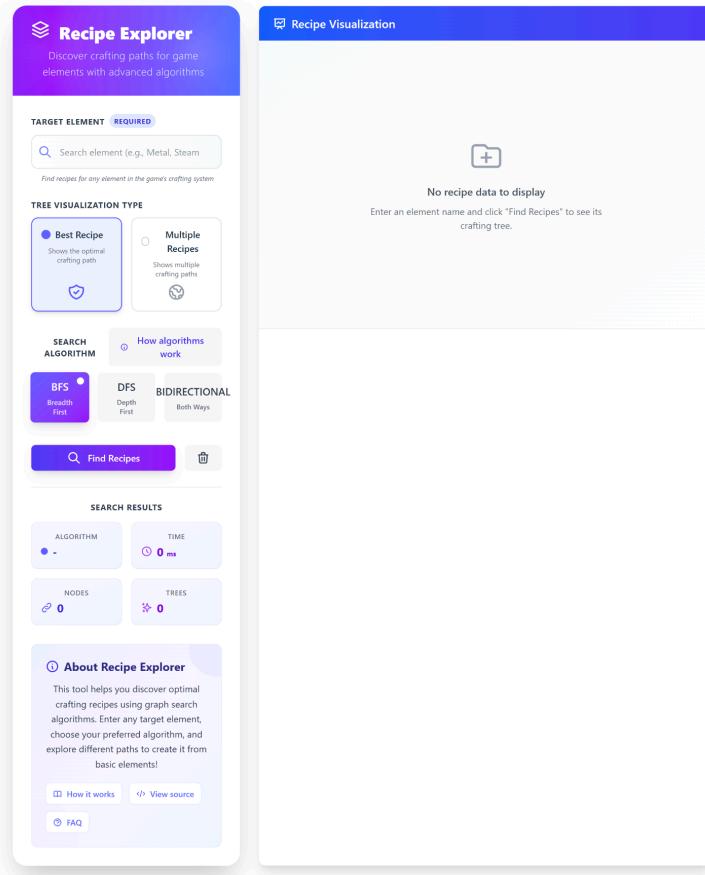
VerifyCompletePath	Terletak pada bagian perbandingan pohon, fungsi ini memvalidasi jalur pencarian sudah lengkap atau tidak
GenerateAllRecipeVariations	Terletak pada bagian pembentukan, fungsi ini menghasilkan semua variasi resep untuk sebuah elemen
GenerateRecipeVariationsWithSubIngredients	Terletak pada bagian pembentukan, fungsi ini membentuk variasi resep dari eksplorasi elemen pembentuk tingkat bawah
GenerateTreeCombinations	Terletak pada bagian pembentukan, fungsi ini digunakan untuk menghasilkan kombinasi pohon dari variasi bahan
GenerateDetailedTreeSignature	Terletak pada bagian pembentukan, fungsi ini berfungsi untuk memberikan penanda until untuk sebuah pohon
BuildIngredientTreeWithSpecificRecipe	Terletak pada bagian pembentukan, fungsi ini berfungsi untuk membangun pohon dengan resep spesifik
PathToTree	Terletak pada bagian manipulasi struktur data pohon, fungsi ini mengkonversi jalur pencarian menjadi sebuah pohon.
CreateSubtreeFromPath	Terletak pada bagian manipulasi struktur data pohon, fungsi ini membangun sebuah upa pohon dari upa jalur
ConvertPathToCompleteTree	Terletak pada bagian manipulasi struktur data pohon, fungsi ini digunakan untuk mengkonversi jalur ke pohon lengkap
ConvertPathToSubTree	Terletak pada bagian manipulasi struktur data pohon, fungsi ini digunakan untuk mengubah jalur menjadi sebuah upa pohon.
GenerateTreesForRecipe	Terletak pada bagian manipulasi struktur data pohon, fungsi ini digunakan untuk menghasilkan pohon untuk sebuah resep
BuildElementTreeBFS	Terletak pada bagian manipulasi struktur data pohon, fungsi ini digunakan untuk

	membentuk dengan algoritma BFS
BuildElementTreeDFS	Terletak pada bagian manipulasi struktur data pohon, fungsi ini digunakan untuk membentuk dengan algoritma DFS
ValidateRecipeTier	Terletak pada bagian validasi, fungsi ini digunakan untuk validasi resep agar pembentuk memiliki tingkatan lebih kecil.
IsBaseElementName	Terletak pada bagian validasi, fungsi ini digunakan untuk validasi elemen dasar
ValidateCircularDependencies	Terletak pada bagian validasi, fungsi ini digunakan untuk validasi dependensi sirkular dan menanganinya

## B. Tata Cara Penggunaan Program

Berikut adalah tampilan utama halaman dari program :

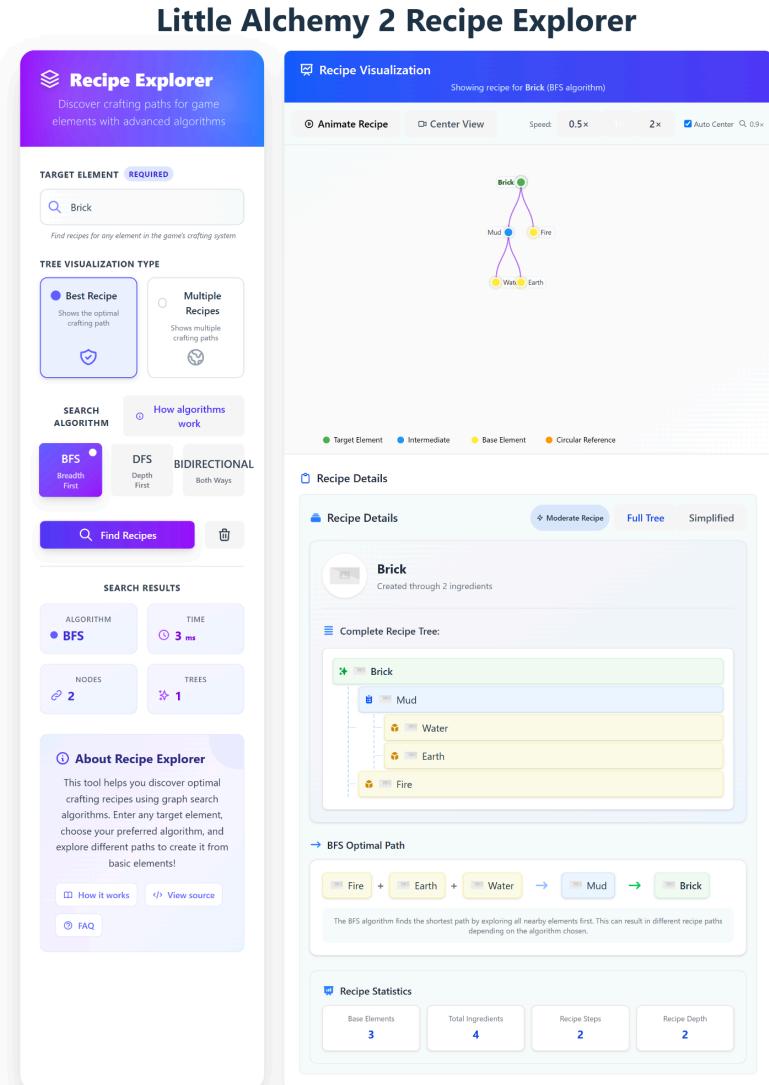
**Little Alchemy 2 Recipe Explorer**



Di bagian kiri terdapat bagian utama yang akan menerima masukan dari pengguna yaitu nama elemen, pilihan banyaknya elemen, serta jenis algoritma yang ingin digunakan. Setelah itu, di bagian bawahnya terdapat beberapa informasi tambahan mengenai hasil pencarian yang dilakukan seperti waktu pencarian, algoritma yang digunakan, banyak simpul yang ditemui dan jumlah pohon yang terbentuk. Bagian kanan halaman merupakan tempat bagi hasil dari pencarian akan ditampilkan dalam bentuk visualisasi.

Untuk menggunakannya, pengguna cukup mengetikkan elemen yang ingin dicari pada blok target element di bagian kiri halaman. Jika pengguna membutuhkan, aplikasi juga menyediakan daftar seluruh elemen pada permainan yang dapat diakses dengan *drop down* di blok yang sama. Setelah itu, pengguna bisa memilih apakah ingin mencari hanya satu buah resep dan banyak resep. Jika ingin banyak elemen, maka pengguna akan diminta memasukkan jumlah resep yang diinginkan untuk dicari. Selanjutnya, pengguna dapat memilih algoritma yang diinginkan dalam melakukan pencarian yaitu *Breadth-First Search* (BFS), *Depth-First Search* (DFS), dan *Bidirectional*. Pengguna dapat melihat penjelasan setiap algoritma pada bagian *How algorithms work* dengan mengkliknya. Setelah selesai memilih dan memasukkan seluruh data yang dibutuhkan, pengguna dapat menjalankan pencarian dengan mengklik tombol *Find Recipes*.

Contoh hasil pencarian dari aplikasi :



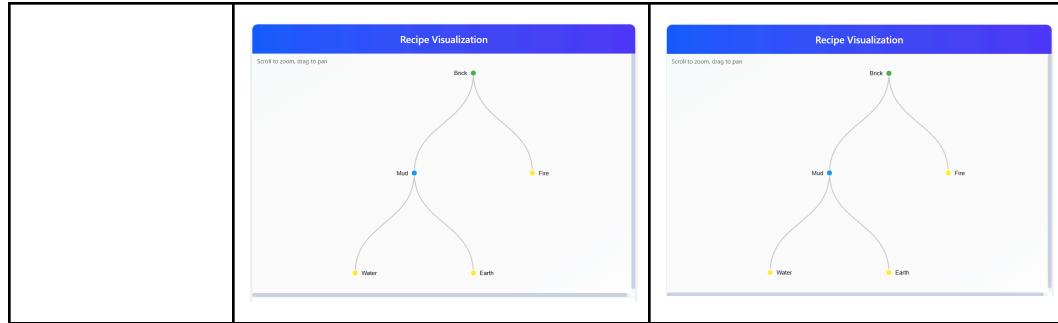
Setelah dijalankan, aplikasi akan secara otomatis menampilkan seluruh hasil pencarian beserta data-data pelengkapnya kepada pengguna. Di bagian kiri, dapat dilihat bahwa ada informasi mengenai algoritma yang dibutuhkan, waktu pencarian, jumlah simpul, dan jumlah pohon pencarian yang didapatkan. Di bagian kanan terdapat visualisasi resep dari elemen target (ditandai warna hijau) hingga ke elemen dasar (ditandai dengan warna kuning). Elemen selain itu merupakan elemen tengah yang ditandai dengan warna biru. Untuk detail pencarinya dapat dilihat di bagian bawah visualisasi yaitu resep komplit dari pohon elemen, urutan elemen pada pencarian jalur oleh algoritma, serta statistik resep seperti jumlah elemen dasar, jumlah bahan pembentuk, jumlah tahapan resep, dan kedalaman resep.

### C. Pengujian dan Hasil

#### 1. Elemen Brick

Algoritma	Tipe Pencarian
-----------	----------------

	Satu Resep	Banyak Resep (Maksimal 5)								
BFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 2 ms</td> </tr> <tr> <td>NODES 🔗 2</td> <td>TREES ✿ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 2 ms	NODES 🔗 2	TREES ✿ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 2 ms</td> </tr> <tr> <td>NODES 🔗 2</td> <td>TREES ✿ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 2 ms	NODES 🔗 2	TREES ✿ 1
ALGORITHM ● BFS	TIME ⌚ 2 ms									
NODES 🔗 2	TREES ✿ 1									
ALGORITHM ● BFS	TIME ⌚ 2 ms									
NODES 🔗 2	TREES ✿ 1									
DFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td> <td>TIME ⌚ 6 ms</td> </tr> <tr> <td>NODES 🔗 40</td> <td>TREES ✿ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● DFS	TIME ⌚ 6 ms	NODES 🔗 40	TREES ✿ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td> <td>TIME ⌚ 73 ms</td> </tr> <tr> <td>NODES 🔗 40</td> <td>TREES ✿ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● DFS	TIME ⌚ 73 ms	NODES 🔗 40	TREES ✿ 1
ALGORITHM ● DFS	TIME ⌚ 6 ms									
NODES 🔗 40	TREES ✿ 1									
ALGORITHM ● DFS	TIME ⌚ 73 ms									
NODES 🔗 40	TREES ✿ 1									
Bidirectional	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM BIDIRECTIONAL</td> <td>TIME ⌚ 2 ms</td> </tr> <tr> <td>NODES 🔗 622</td> <td>TREES ✿ 1</td> </tr> </table>	ALGORITHM BIDIRECTIONAL	TIME ⌚ 2 ms	NODES 🔗 622	TREES ✿ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM BIDIRECTIONAL</td> <td>TIME ⌚ 1 ms</td> </tr> <tr> <td>NODES 🔗 622</td> <td>TREES ✿ 1</td> </tr> </table>	ALGORITHM BIDIRECTIONAL	TIME ⌚ 1 ms	NODES 🔗 622	TREES ✿ 1
ALGORITHM BIDIRECTIONAL	TIME ⌚ 2 ms									
NODES 🔗 622	TREES ✿ 1									
ALGORITHM BIDIRECTIONAL	TIME ⌚ 1 ms									
NODES 🔗 622	TREES ✿ 1									



## 2. Elemen Clay

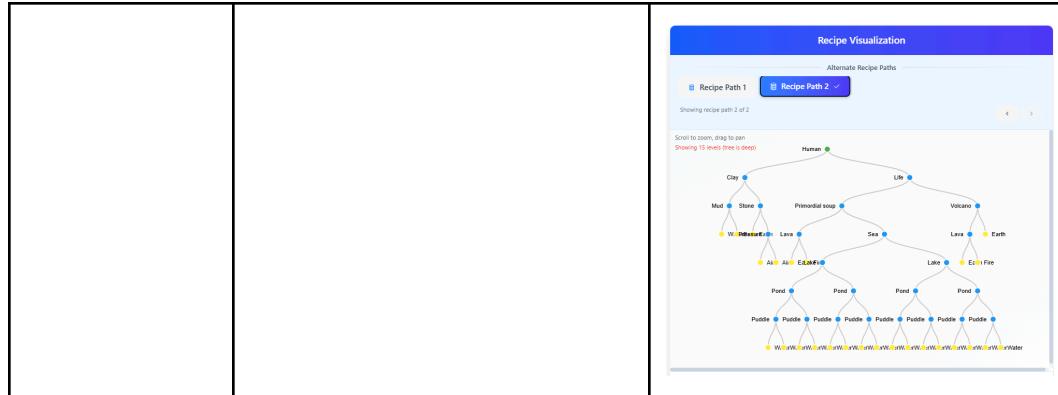
Algoritma	Tipe Pencarian																	
	Satu Resep	Banyak Resep (Maksimal 5)																
BFS	<p><b>SEARCH RESULTS</b></p> <table> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>● BFS</td> <td>⌚ 2 ms</td> </tr> </table> <table> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>🔗 6</td> <td>💡 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	TIME	● BFS	⌚ 2 ms	NODES	TREES	🔗 6	💡 1	<p><b>SEARCH RESULTS</b></p> <table> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>● BFS</td> <td>⌚ 5 ms</td> </tr> </table> <table> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>🔗 6</td> <td>💡 2</td> </tr> </table> <p><b>Recipe Visualization</b></p> <p>Alternate Recipe Paths</p> <p>Showing recipe path 1 of 2</p> <p><b>Recipe Visualization</b></p> <p>Alternate Recipe Paths</p> <p>Showing recipe path 2 of 2</p>	ALGORITHM	TIME	● BFS	⌚ 5 ms	NODES	TREES	🔗 6	💡 2
ALGORITHM	TIME																	
● BFS	⌚ 2 ms																	
NODES	TREES																	
🔗 6	💡 1																	
ALGORITHM	TIME																	
● BFS	⌚ 5 ms																	
NODES	TREES																	
🔗 6	💡 2																	

DFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>● DFS</td> <td>⌚ 5 ms</td> </tr> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>⌚ 100</td> <td>骺 1</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	TIME	● DFS	⌚ 5 ms	NODES	TREES	⌚ 100	骺 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>● DFS</td> <td>⌚ 110 ms</td> </tr> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>⌚ 100</td> <td>骺 2</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	TIME	● DFS	⌚ 110 ms	NODES	TREES	⌚ 100	骺 2
ALGORITHM	TIME																	
● DFS	⌚ 5 ms																	
NODES	TREES																	
⌚ 100	骺 1																	
ALGORITHM	TIME																	
● DFS	⌚ 110 ms																	
NODES	TREES																	
⌚ 100	骺 2																	
Bidirectional	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>BIDIRECTIONAL</td> <td>⌚ 7 ms</td> </tr> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>⌚ 653</td> <td>骺 1</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	TIME	BIDIRECTIONAL	⌚ 7 ms	NODES	TREES	⌚ 653	骺 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>TIME</td> </tr> <tr> <td>BIDIRECTIONAL</td> <td>⌚ 1 ms</td> </tr> <tr> <td>NODES</td> <td>TREES</td> </tr> <tr> <td>⌚ 650</td> <td>骺 1</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	TIME	BIDIRECTIONAL	⌚ 1 ms	NODES	TREES	⌚ 650	骺 1
ALGORITHM	TIME																	
BIDIRECTIONAL	⌚ 7 ms																	
NODES	TREES																	
⌚ 653	骺 1																	
ALGORITHM	TIME																	
BIDIRECTIONAL	⌚ 1 ms																	
NODES	TREES																	
⌚ 650	骺 1																	

### 3. Elemen Human

Algoritma	Tipe Pencarian									
	Satu Resep		Banyak Resep (Maksimal 5)							
BFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 4 ms</td> </tr> <tr> <td>NODES ∅ 360</td> <td>TREES ☒ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 4 ms	NODES ∅ 360	TREES ☒ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 31 ms</td> </tr> <tr> <td>NODES ∅ 0</td> <td>TREES ☒ 5</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 31 ms	NODES ∅ 0	TREES ☒ 5
ALGORITHM ● BFS	TIME ⌚ 4 ms									
NODES ∅ 360	TREES ☒ 1									
ALGORITHM ● BFS	TIME ⌚ 31 ms									
NODES ∅ 0	TREES ☒ 5									
DFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td> <td>TIME ⌚ 18 ms</td> </tr> <tr> <td>NODES ∅ 720</td> <td>TREES ☒ 1</td> </tr> </table>	ALGORITHM ● DFS	TIME ⌚ 18 ms	NODES ∅ 720	TREES ☒ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td> <td>TIME ⌚ 185 ms</td> </tr> <tr> <td>NODES ∅ 720</td> <td>TREES ☒ 5</td> </tr> </table>	ALGORITHM ● DFS	TIME ⌚ 185 ms	NODES ∅ 720	TREES ☒ 5
ALGORITHM ● DFS	TIME ⌚ 18 ms									
NODES ∅ 720	TREES ☒ 1									
ALGORITHM ● DFS	TIME ⌚ 185 ms									
NODES ∅ 720	TREES ☒ 5									

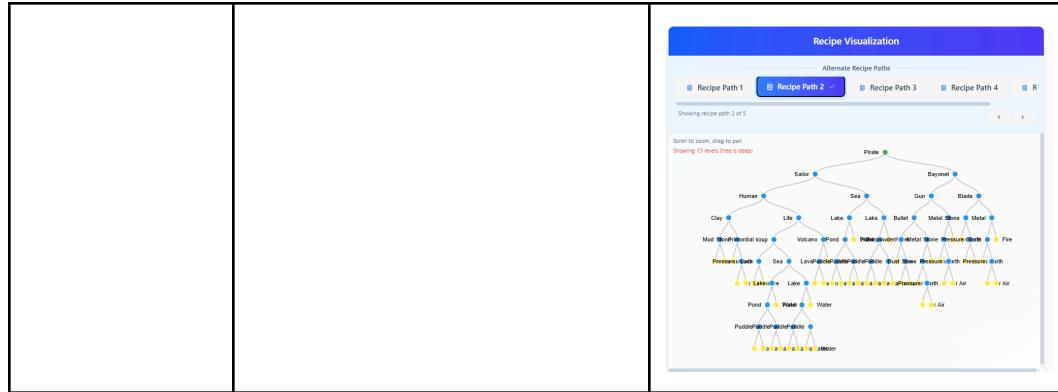
	<p><b>Recipe Visualization</b></p>	<p><b>Recipe Visualization</b></p>								
Bidirectional	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM <b>BIDIRECTIONAL</b></td> <td>TIME ⌚ 1 ms</td> </tr> <tr> <td>NODES 🔗 109</td> <td>TREES ✳️ 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM <b>BIDIRECTIONAL</b>	TIME ⌚ 1 ms	NODES 🔗 109	TREES ✳️ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM <b>BIDIRECTIONAL</b></td> <td>TIME ⌚ 1 ms</td> </tr> <tr> <td>NODES 🔗 109</td> <td>TREES ✳️ 2</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM <b>BIDIRECTIONAL</b>	TIME ⌚ 1 ms	NODES 🔗 109	TREES ✳️ 2
ALGORITHM <b>BIDIRECTIONAL</b>	TIME ⌚ 1 ms									
NODES 🔗 109	TREES ✳️ 1									
ALGORITHM <b>BIDIRECTIONAL</b>	TIME ⌚ 1 ms									
NODES 🔗 109	TREES ✳️ 2									



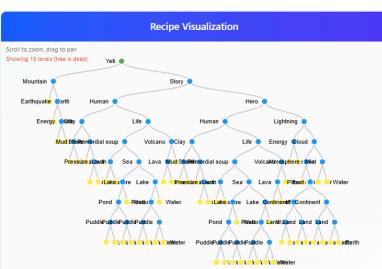
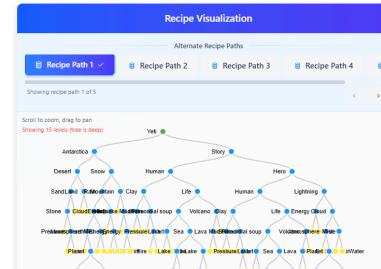
#### 4. Elemen Pirate

Algoritma	Tipe Pencarian																			
	Satu Resep	Banyak Resep (Maksimal 5)																		
BFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 19 ms</td> </tr> <tr> <td>NODES 🔗 4117</td> <td>TREES 📌 1</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 19 ms	NODES 🔗 4117	TREES 📌 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 45 ms</td> </tr> <tr> <td>NODES 🔗 0</td> <td>TREES 📌 5</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 45 ms	NODES 🔗 0	TREES 📌 5	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 45 ms</td> </tr> <tr> <td>NODES 🔗 0</td> <td>TREES 📌 5</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 45 ms	NODES 🔗 0	TREES 📌 5	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BFS</td> <td>TIME ⌚ 45 ms</td> </tr> <tr> <td>NODES 🔗 0</td> <td>TREES 📌 5</td> </tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BFS	TIME ⌚ 45 ms	NODES 🔗 0	TREES 📌 5
ALGORITHM ● BFS	TIME ⌚ 19 ms																			
NODES 🔗 4117	TREES 📌 1																			
ALGORITHM ● BFS	TIME ⌚ 45 ms																			
NODES 🔗 0	TREES 📌 5																			
ALGORITHM ● BFS	TIME ⌚ 45 ms																			
NODES 🔗 0	TREES 📌 5																			
ALGORITHM ● BFS	TIME ⌚ 45 ms																			
NODES 🔗 0	TREES 📌 5																			

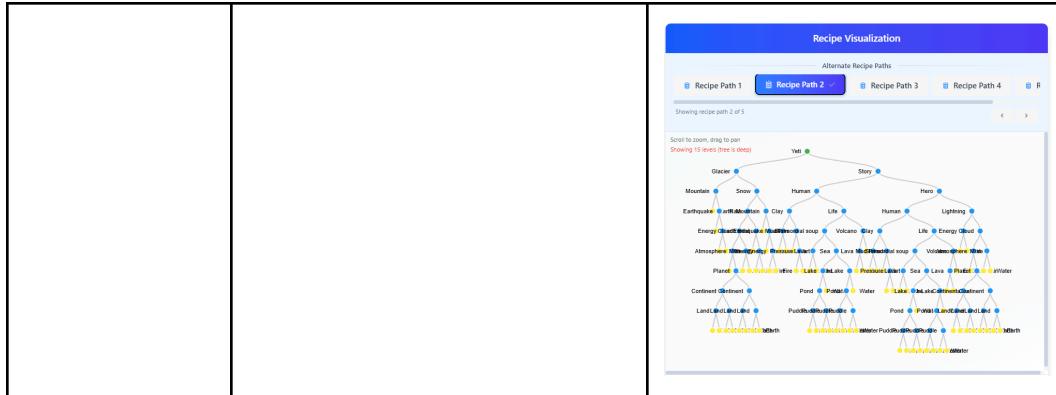
DFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td><td>TIME ⌚ 66 ms</td></tr> <tr> <td>NODES 🔗 9920</td><td>TREES ✳️ 1</td></tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● DFS	TIME ⌚ 66 ms	NODES 🔗 9920	TREES ✳️ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● DFS</td><td>TIME ⌚ 406 ms</td></tr> <tr> <td>NODES 🔗 9920</td><td>TREES ✳️ 5</td></tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● DFS	TIME ⌚ 406 ms	NODES 🔗 9920	TREES ✳️ 5
ALGORITHM ● DFS	TIME ⌚ 66 ms									
NODES 🔗 9920	TREES ✳️ 1									
ALGORITHM ● DFS	TIME ⌚ 406 ms									
NODES 🔗 9920	TREES ✳️ 5									
Bidirectional	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BIDIRECTIONAL</td><td>TIME ⌚ 1 ms</td></tr> <tr> <td>NODES 🔗 404</td><td>TREES ✳️ 1</td></tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BIDIRECTIONAL	TIME ⌚ 1 ms	NODES 🔗 404	TREES ✳️ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tr> <td>ALGORITHM ● BIDIRECTIONAL</td><td>TIME ⌚ 1 ms</td></tr> <tr> <td>NODES 🔗 352</td><td>TREES ✳️ 5</td></tr> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM ● BIDIRECTIONAL	TIME ⌚ 1 ms	NODES 🔗 352	TREES ✳️ 5
ALGORITHM ● BIDIRECTIONAL	TIME ⌚ 1 ms									
NODES 🔗 404	TREES ✳️ 1									
ALGORITHM ● BIDIRECTIONAL	TIME ⌚ 1 ms									
NODES 🔗 352	TREES ✳️ 5									



## 5. Elemen Yeti

Algoritma	Tipe Pencarian			
	Satu Resep	Banyak Resep (Maksimal 5)		
BFS	SEARCH RESULTS		SEARCH RESULTS	
	<b>ALGORITHM</b> <span style="color: blue;">● BFS</span>	<b>TIME</b> <span style="color: purple;">⌚ 41 ms</span>	<b>ALGORITHM</b> <span style="color: blue;">● BFS</span>	<b>TIME</b> <span style="color: purple;">⌚ 32 ms</span>
	<b>NODES</b> <span style="color: purple;">🔗 4744</span>	<b>TREES</b> <span style="color: purple;">✳️ 1</span>	<b>NODES</b> <span style="color: purple;">🔗 4</span>	<b>TREES</b> <span style="color: purple;">✳️ 5</span>
	<b>Recipe Visualization</b> 		<b>Recipe Visualization</b> 	
	<b>Recipe Visualization</b> 		<b>Recipe Visualization</b> 	

DFS	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>● DFS</td> </tr> <tr> <td>TIME</td> <td>⌚ 117 ms</td> </tr> <tr> <td>NODES</td> <td>🔗 10700</td> </tr> <tr> <td>TREES</td> <td>❖ 1</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	● DFS	TIME	⌚ 117 ms	NODES	🔗 10700	TREES	❖ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>● DFS</td> </tr> <tr> <td>TIME</td> <td>⌚ 627 ms</td> </tr> <tr> <td>NODES</td> <td>🔗 10700</td> </tr> <tr> <td>TREES</td> <td>❖ 5</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	● DFS	TIME	⌚ 627 ms	NODES	🔗 10700	TREES	❖ 5
ALGORITHM	● DFS																	
TIME	⌚ 117 ms																	
NODES	🔗 10700																	
TREES	❖ 1																	
ALGORITHM	● DFS																	
TIME	⌚ 627 ms																	
NODES	🔗 10700																	
TREES	❖ 5																	
Bidirectional	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>BIDIRECTIONAL</td> </tr> <tr> <td>TIME</td> <td>⌚ 2 ms</td> </tr> <tr> <td>NODES</td> <td>🔗 604</td> </tr> <tr> <td>TREES</td> <td>❖ 1</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	BIDIRECTIONAL	TIME	⌚ 2 ms	NODES	🔗 604	TREES	❖ 1	<p><b>SEARCH RESULTS</b></p> <table border="1"> <tbody> <tr> <td>ALGORITHM</td> <td>BIDIRECTIONAL</td> </tr> <tr> <td>TIME</td> <td>⌚ 2 ms</td> </tr> <tr> <td>NODES</td> <td>🔗 580</td> </tr> <tr> <td>TREES</td> <td>❖ 5</td> </tr> </tbody> </table> <p><b>Recipe Visualization</b></p>	ALGORITHM	BIDIRECTIONAL	TIME	⌚ 2 ms	NODES	🔗 580	TREES	❖ 5
ALGORITHM	BIDIRECTIONAL																	
TIME	⌚ 2 ms																	
NODES	🔗 604																	
TREES	❖ 1																	
ALGORITHM	BIDIRECTIONAL																	
TIME	⌚ 2 ms																	
NODES	🔗 580																	
TREES	❖ 5																	



#### D. Analisis Hasil Pengujian

Setelah melakukan pengujian, dapat ditarik beberapa kesimpulan. Dalam hal efektivitas dan efisiensi, algoritma pencarian *Bidirectional* selalu berada pada peringkat pertama. Jarak waktu pencarinya dapat dikategorikan cukup jauh. Contohnya adalah pada elemen Yeti, waktu pencarian dari BFS satu recipe adalah 41 ms, DFS 117 ms, dan Bidirectional 2 ms. Hal ini juga dapat dilihat dengan contoh lain seperti Pirate dengan BFS satu resep 19 ms, DFS 66 ms, dan Bidirectional 1 ms. Performa dari algoritma ini sangat baik dibandingkan dengan algoritma pencarian lainnya.

Dalam konteks jumlah simpul yang dikunjungi, algoritma DFS selalu memberikan yang terbanyak. Hal ini disebabkan karena algoritma ini harus menelusuri satu buah elemen pembentuk hingga ke elemen dasar sebelum melanjutkannya ke elemen lain. Sedangkan BFS yang melakukan pencarian melebar akan mampu memberikan hasil lebih cepat karena pencarian seluruh resep bersamaan. Contohnya adalah pada kasus Yeti, terdapat 4474 simpul yang dikunjungi pada algoritma BFS, 10.700 simpul pada DFS, dan 604 simpul pada Bidirectional.

Dalam pembahasan jumlah resep yang bisa dicari, sebagian besar hasil pencarian, terlepas dari algoritma pilihan, memberikan hasil yang sama. Akan tetapi, dalam beberapa kasus khusus, jumlah dari resep yang ditemukan oleh algoritma Bidirectional lebih sedikit daripada algoritma lainnya. Contohnya adalah pada pencarian resep elemen Human dengan batasan 5 resep, BFS dan DFS mengembalikan hasil 5 resep namun Bidirectional hanya mengembalikan 2. Hal ini disebabkan karena pencarian maju dan mundurnya tidak bertemu dalam beberapa kasus dalam batas yang telah ditentukan. Karena pencarinya terpisah dan ekspansi dari elemen target dan dasar sangat berbeda, maka dalam kasus khusus, hal ini dapat terjadi.

Namun, terlepas dari itu semua, seluruh algoritma pencarian resep sebuah elemen dapat bekerja dengan baik. Pembentukan resep menjadi sebuah struktur data pohon menjadikannya dapat dimanipulasi dengan baik. Dengan itu, algoritma pencarian pada pohon seperti *Breadth First Search*, *Depth First Search*, dan *Bidirectional* dapat digunakan untuk mencari. Setiap algoritma juga memberikan hasil yang tepat tanpa adanya kesalahan dalam prosesnya. Maka, seluruh algoritma dinyatakan berhasil.

## BAB V

### Penutup

#### A. Kesimpulan

Dalam masalah pencarian *recipe* dari sebuah elemen pada permainan *Little Alchemy 2*, pencarian tersebut dapat dilakukan dengan algoritma eksplorasi struktur data pohon. Pembentukan sebuah elemen dan bahan pembentuknya menjadi sebuah pohon akan membuat struktur pohon yang tersambung terus hingga elemen dasar. Adapun jenis algoritma yang diterapkan dapat berupa *Breadth First Search* (BFS), *Depth First Search* (DFS), dan *Bidirectional*.

Setiap algoritma memiliki konsepnya masing-masing yaitu BFS yang berfokus pada pencarian melebar setiap *recipe*, DFS berfokus pada pencarian mendalam pada sebuah *recipe*, dan *Bidirectional* yang melakukan pencarian dari 2 arah. Setiapnya memiliki keunggulan dan kelemahan masing-masing. Dengan perbedaan algoritmanya, hasil dari pohon jalurnya pun berbeda-beda.

Dalam memfasilitasi pengguna untuk mengakses program ini dengan baik, presentasi visual dilakukan dalam bentuk Web. Pengguna dapat memberi masukan melalui *keyboard* mengenai berbagai parameter yang akan diolah oleh program. Setelah itu, hasilnya akan dikembalikan lagi kepada pengguna dengan tampilan visual yang menarik.

Kami berharap bahwa aplikasi ini dapat dimanfaatkan bagi para pemain Little Alchemy 2 dalam bermain. Aplikasi ini hadir untuk mempermudah pemain meraih elemen unik dalam rangka meningkatkan pencapaian.

#### B. Saran

- Seharusnya algoritma dirancang dimulai dari dasar yang konkret sebelum lanjut kepada tahap implementasi.
- Seharusnya alokasi waktu dan tenaga lebih diprioritaskan pada tugas ini.
- Seharusnya spesifikasi dituruti dan sering membaca lembar *Question and Answer*

#### C. Refleksi

Melalui pengerjaan tugas besar ini, kami belajar mengenai banyak hal. Salah satu hal utama adalah pembelajaran mendalam mengenai pemrograman dengan bahasa Go. Eksplorasi ini membentuk pribadi yang adaptif terhadap teknologi yang digunakan. Selain itu, kami juga belajar mendalam mengenai konsep baru yaitu *web scrapping*. Mengatur pemilihan data sedemikian rupa dari struktur Web mengharuskan kami mempelajari struktur Website bukan milik kami. Sehingga, kami terus mempelajari pentingnya konvensi dan kerapian dalam menulis kode.

Tidak hanya itu, kami juga terus mendalamai pemahaman kami mengenai aplikasi algoritma eksplorasi struktur data pohon seperti BFS, DFS, dan Bidirectional. Terakhir, kami juga banyak mempelajari mengenai konsep *multithreading* yang dilakukan untuk mendorong kinerja dan efektivitas pengerjaan dalam pencarian skala besar. Alhasil, kami belajar banyak hal melalui pengerjaan tugas besar kali ini.

## Lampiran

### Referensi :

[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)

[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)

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

### Tautan Repository :

<https://github.com/Incheon21/Tubes-2-Stima>

### Tautan Video :

[https://www.youtube.com/shorts/jbC6\\_qiNNCg](https://www.youtube.com/shorts/jbC6_qiNNCg)

### Tautan Web :

[elementer.tecitb.com](http://elementer.tecitb.com) dan <http://152.42.246.157/>

### Tabel Spesifikasi Program :

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.	✓	
----	--	---	--