

**LAPORAN TUGAS BESAR 2**  
**IF2211 STRATEGI ALGORITMA**



**Kelompok 38 FullRustAlchemist**

Ranashahira Reztaputri 13523007

Syahrizal Bani Khairan 13523063

Nayla Zahira 13523079

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

## DAFTAR ISI

DAFTAR ISI	1
BAB I	3
DESKRIPSI TUGAS	3
BAB II	6
LANDASAN TEORI	6
2.1. Penjelajahan Graf	6
2.2. Algoritma Breadth-First Search (BFS)	6
2.3. Algoritma Depth-First Search (DFS)	8
2.4. Aplikasi Web yang Dibangun	9
2.4.1. Front End Aplikasi Web	9
2.4.2. Back End Aplikasi Web	10
BAB III	11
ANALISIS PEMECAHAN MASALAH	11
3.1. Langkah-langkah Pemecahan Masalah.	11
3.1.1. Mencari Tahu Hubungan Antar Elemen	11
3.1.2. Menelusuri Semua Elemen Pembentuk hingga Mencapai Elemen Dasar	11
3.1.3. Algoritma Penelusuran Graf Elemen dengan Multithreading	11
3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma DFS dan BFS.	12
3.2.1. Algoritma BFS	12
3.2.2. Algoritma DFS	13
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web	15
3.4. Contoh Ilustrasi Kasus	15
3.4.1 DFS	15
3.4.2 BFS	17
BAB IV	19
IMPLEMENTASI DAN PENGUJIAN	19
4.1. Struktur Data Program	19
4.1.1. Struktur Graf Resep	19
4.1.2. Struktur DFS	19
4.1.3. Struktur BFS	20
4.2. Fungsi dan Prosedur	20
4.2.1 Scraper	20
4.2.2 BFS	21
4.2.3 DFS	22
4.3. Tata Cara Penggunaan Program	23
4.3.1. Cara Menjalankan Program	23

4.3.2. Interface Program	24
4.4. Hasil pengujian program	30
4.4.1. Test Case 1	30
4.4.1.1. Single Recipe - BFS	30
4.4.1.2. Single Recipe - DFS	31
4.4.1.3. Multiple Recipe - BFS	33
4.4.1.4. Multiple Recipe - DFS	36
4.4.2. Test Case 2	39
4.4.2.1. Single Recipe - BFS	40
4.4.2.2. Single Recipe - DFS	41
4.4.2.3. Multiple Recipe - BFS	42
4.4.2.4. Multiple Recipe - DFS	44
4.3.3. Test Case 3	46
4.3.3.1. Single Recipe - BFS	47
4.3.3.2. Single Recipe - DFS	47
4.3.3.3. Multiple Recipe - BFS	47
4.3.3.4. Multiple Recipe - DFS	47
4.3.4. Test Case 4	48
4.3.4.1. Single Recipe - BFS	48
4.3.4.2. Single Recipe - DFS	49
4.3.4.3. Multiple Recipe - BFS	49
4.3.4.4. Multiple Recipe - DFS	50
4.5. Analisis hasil pengujian	51
BAB V	53
KESIMPULAN, SARAN, DAN REFLEKSI	53
5.1. Kesimpulan	53
5.2. Saran	53
5.3. Refleksi	54
LAMPIRAN	55
DAFTAR PUSTAKA	56

## **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.

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

### **Spesifikasi Bonus**

- (maks 5) Membuat video tentang aplikasi BFS dan DFS pada permainan Little Alchemy 2 di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll. Semakin menarik video, maka semakin banyak poin yang diberikan.
- (maks 3) Aplikasi dijalankan menggunakan Docker baik untuk frontend maupun backend.

- (maks 3) Aplikasi di-deploy ke aplikasi deployment (aplikasi deployment bebas) agar bisa diakses secara daring
- (maks 3) Menambahkan algoritma bukan hanya DFS dan BFS, tetapi juga strategi bidirectional
- (maks 6) Aplikasi memiliki fitur Live Update visualisasi recipe selama proses pencarian. Tree visualisasi akan dibangun bertahap secara real time sesuai dengan progress pencarian. Tambahkan delay pada Live Update karena pencarian dapat berjalan dengan sangat cepat.

## BAB II

### LANDASAN TEORI

#### **2.1. Penjelajahan Graf**

Graf merupakan struktur diskrit yang terdiri dari himpunan simpul (*vertex*) dan sisi (*edge*). Secara formal, sebuah graf  $G$  dapat didefinisikan sebagai pasangan himpunan  $(V, E)$ , di mana  $V$  adalah himpunan simpul dan  $E$  adalah himpunan sisi yang menghubungkan sepasang simpul. Algoritma traversal graf atau penjelajahan graf adalah algoritma untuk mengunjungi simpul-simpul di dalam graf dengan cara yang sistematik. Dalam proses pencarian solusi, graf dapat bersifat statis (graf yang sudah terbentuk sebelum proses pencarian dilakukan) atau dinamis (graf yang terbentuk saat proses pencarian dilakukan). Pada graf statis, graf direpresentasikan sebagai struktur data, sedangkan pada graf dinamis, graf dibangun selama pencarian solusi.

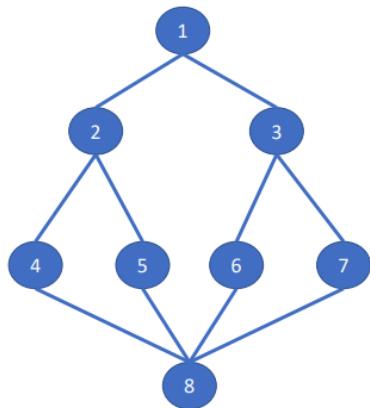
Dalam penjelajahan graf, terdapat dua algoritma utama yang banyak diimplementasikan, yakni pencarian melebar (*Breadth First Search/BFS*) dan pencarian mendalam (*Depth First Search/DFS*). Kedua algoritma ini dikategorikan sebagai algoritma pencarian tanpa informasi (*uninformed/blind search*), karena beroperasi tanpa menggunakan informasi heuristik tambahan untuk mengarahkan proses pencarian. Hal ini berbeda dengan algoritma pencarian berbasis informasi (*informed search*) seperti *Best First Search* dan *A\** yang memanfaatkan fungsi heuristik untuk meningkatkan efisiensi pencarian.

Algoritma traversal graf memiliki banyak aplikasi praktis, di antaranya pemetaan kutipan (*citation map*), web spider untuk mesin pencari, dan penelusuran direktori (folder) pada sistem komputer.

#### **2.2. Algoritma Breadth-First Search (BFS)**

Breadth-First Search (BFS) adalah algoritma penjelajahan graf yang mengunjungi simpul secara melebar. Traversal dimulai dari sebuah simpul awal, kemudian dilanjutkan dengan mengunjungi semua simpul yang bertetangga dengan simpul awal tersebut terlebih dahulu. Setelah semua simpul tetangga tersebut dikunjungi, algoritma melanjutkan dengan mengunjungi simpul-simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sebelumnya dikunjungi, demikian seterusnya hingga semua simpul terjelajahi.

Implementasi BFS memerlukan struktur data antrian (*queue*) yang beroperasi dengan prinsip FIFO (First In First Out), matriks ketetanggaan atau senarai ketetanggaan untuk merepresentasikan graf, serta tabel Boolean "dikunjungi" yang menyimpan status setiap simpul.



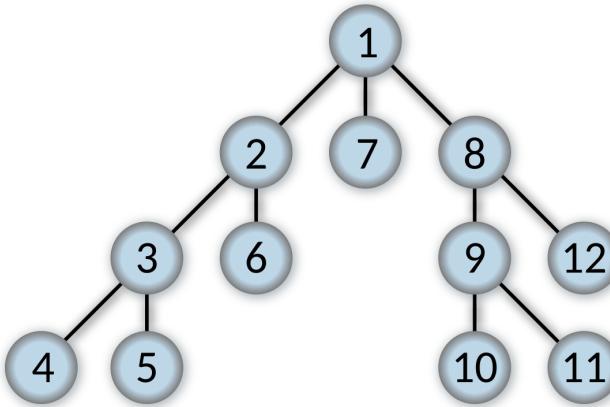
Iterasi	v	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Proses algoritma BFS dimulai dengan memasukkan simpul awal ke dalam antrian dan menandainya "dikunjungi". Selama antrian tidak kosong, algoritma akan mengeluarkan simpul pertama dari depan antrian, memprosesnya, dan memasukkan seluruh tetangganya yang belum dikunjungi ke dalam antrian. Sebagai ilustrasi, pada graf di atas urutan kunjungan BFS dari simpul 1 adalah 1, 2, 3, 4, 5, 6, 7, 8. Kompleksitas waktu BFS adalah  $O(V^2)$  untuk representasi matriks ketetanggaan dan  $O(V + E)$  untuk representasi senarai ketetanggaan, di mana V adalah jumlah simpul dan E adalah jumlah sisi.

Algoritma BFS sangat bermanfaat untuk menemukan lintasan terpendek dalam graf tidak berbobot karena menjelajahi simpul-simpul berdasarkan jaraknya dari simpul awal. Hal ini menjamin bahwa ketika simpul tujuan ditemukan, lintasan yang terbentuk akan menjadi lintasan terpendek dari simpul awal ke simpul tujuan. BFS juga digunakan dalam berbagai aplikasi seperti pendekripsi komponen terhubung dalam graf tidak berarah, algoritma Prim untuk Minimum Spanning Tree, dan penyelesaian teka-teki yang memerlukan jalur terpendek.

### 2.3. Algoritma Depth-First Search (DFS)



Depth-First Search (DFS) atau pencarian mendalam adalah algoritma penjelajahan graf yang mengunjungi simpul secara mendalam sebelum melebar. Traversal dimulai dari sebuah simpul awal, kemudian dilanjutkan dengan mengunjungi salah satu simpul tetangga, lalu mengunjungi salah satu tetangga dari simpul yang baru dikunjungi tersebut, dan seterusnya hingga mencapai simpul yang tidak memiliki tetangga yang belum dikunjungi. Ketika mencapai simpul yang semua tetangganya telah dikunjungi, algoritma akan melakukan runut-balik (*backtrack*) ke simpul terakhir yang memiliki tetangga yang belum dikunjungi, dan melanjutkan proses dari sana. Algoritma DFS dapat diimplementasikan secara rekursif atau iteratif menggunakan struktur data *stack* yang beroperasi dengan prinsip LIFO (Last In First Out).

Perihal efisiensi, algoritma DFS memiliki kompleksitas waktu yang sama dengan BFS yaitu  $O(V^2)$  dalam representasi matriks ketetanggaan dan  $O(V + E)$  dalam representasi senarai ketetanggaan. Namun, dari segi penggunaan memori, DFS cenderung lebih efisien dibandingkan dengan BFS karena hanya perlu menyimpan simpul-simpul di jalur saat ini, bukan semua simpul pada level yang sama.

DFS ini memiliki banyak aplikasi praktis, termasuk pendekripsi siklus dalam graf, penentuan komponen terhubung kuat (*strongly connected components*) dalam graf berarah, penyelesaian teka-teki labirin, dan topological sorting pada graf berarah asiklik (DAG). Khusus untuk graf berarah, jika beberapa simpul tidak dapat dicapai dari simpul awal yang dipilih, algoritma DFS dapat dijalankan kembali dengan simpul yang belum dikunjungi sebagai simpul awal baru, hingga semua simpul dalam graf telah dikunjungi.

## **2.4. Aplikasi Web yang Dibangun**

Dalam perancangan dan pembuatan website Little Alchemy Recipe Finder, secara umum, terdapat dua bagian utama yang saling melengkapi, yaitu frontend dan backend. Pada sisi frontend, kami mengadopsi Next.js, yaitu sebuah framework React terkemuka yang memungkinkan pengembangan antarmuka pengguna yang interaktif dan responsif. Untuk mendukung pengelolaan data dan algoritma pemrosesan resep, backend kami ditangani oleh Gin, yaitu sebuah framework web yang dibangun di atas bahasa pemrograman Golang. Keunggulan Gin dalam menangani konkurensi dan throughput tinggi menjadikannya pilihan ideal untuk aplikasi yang membutuhkan respons cepat dan skalabilitas.

### **2.4.1. Front End Aplikasi Web**

Frontend merupakan bagian dari aplikasi yang berfungsi untuk menampilkan informasi kepada pengguna sekaligus menyediakan antarmuka interaktif. Di sisi ini, seluruh interaksi seperti klik tombol, penyajian elemen visual, hingga tampilan grafis dikelola untuk memastikan pengalaman pengguna (user experience) berjalan mulus dan intuitif.

Dalam pengembangan aplikasi web ini, digunakan React dan Next.js karena masing-masing memiliki keunggulan tersendiri. React mempermudah pembuatan antarmuka dinamis berbasis komponen, sehingga setiap bagian tampilan dapat diatur secara lebih modular dan efisien. Penggunaan Next.js memudahkan integrasi dengan backend sehingga dapat mempercepat proses pengembangan. Struktur folder yang digunakan untuk frontend mengikuti struktur dasar yang dihasilkan secara otomatis oleh Next.js.

Penggunaan TSX (TypeScript XML), yang menggabungkan TypeScript dan HTML, memungkinkan penulisan kode frontend menjadi lebih intuitif, aman, dan mudah dipahami. Dengan kombinasi React dan Next.js, frontend aplikasi ini dapat dibangun secara lebih modern, terstruktur, dan responsif, mendukung terciptanya pengalaman pengguna yang optimal.

Selain itu, untuk mendukung tampilan yang konsisten dan mudah disesuaikan, digunakan Tailwind CSS. Dengan pendekatan utility-first classes, Tailwind memungkinkan gaya ditulis langsung dalam elemen HTML, sehingga meminimalkan ketergantungan pada file CSS yang panjang dan kompleks. Pendekatan ini mempercepat proses styling dan meningkatkan efisiensi pengembangan antarmuka.

Penggunaan React dalam proyek ini juga menghadirkan manfaat berupa pembuatan komponen yang dapat digunakan kembali (reusable components), yang mempercepat workflow dan mengurangi duplikasi kode. Ditambah lagi, dukungan terhadap state management dinamis memungkinkan aplikasi memberikan interaktivitas tinggi tanpa perlu melakukan reload halaman.

#### **2.4.2. Back End Aplikasi Web**

Backend merupakan bagian dari aplikasi yang bertugas menangani logika bisnis, pengolahan data, serta integrasi dengan layanan eksternal. Backend menyediakan data yang dibutuhkan oleh frontend melalui API, serta memastikan keamanan dan efisiensi dalam proses komunikasi antara klien dan server.

Untuk aplikasi ini, kami menggunakan Golang dengan framework Gin sebagai teknologi backend. Golang dipilih karena performanya yang tinggi, pengelolaan memori yang efisien, dan dukungan konkurensi yang baik melalui goroutines. Bahasa pemrograman ini sangat cocok untuk aplikasi yang membutuhkan kecepatan dan skalabilitas tinggi.

Gin framework menyederhanakan pengembangan API dengan Golang melalui routing yang cepat dan middleware yang fleksibel. Framework ini memungkinkan penanganan permintaan HTTP dengan efisien, validasi data masukan, dan respons yang terstruktur. Gin juga terkenal dengan performanya yang luar biasa, mampu menangani ribuan permintaan per detik.

Arsitektur backend kami menerapkan prinsip REST API, memisahkan dengan jelas antara model, controller, dan router. Model menangani struktur data dan interaksi dengan database, controller berisi logika bisnis, dan router mengarahkan permintaan ke handler yang sesuai. Backend Gin berinteraksi dengan frontend Next.js melalui pertukaran data dalam format JSON, memproses permintaan dari pengguna, mengakses database, dan mengembalikan data yang diperlukan untuk ditampilkan di antarmuka pengguna.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

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

Permasalahan utama yang akan diselesaikan pada tugas besar ini adalah mencari resep yang berisi kombinasi elemen pembentuk dari elemen target yang disertai dengan kombinasi resep dari elemen pembentuk itu pula hingga mencapai hingga semua elemen pembentuk berupa elemen dasar (base element). Dalam memecahkan permasalahan ini, terdapat langkah-langkah penyelesaian masalah yang diterapkan, di antaranya adalah:

##### **3.1.1. Mencari Tahu Hubungan Antar Elemen**

Untuk mengetahui resep yang berisi kombinasi elemen untuk menghasilkan elemen lain, perlu diketahui hubungan antar elemen dalam graf, yang mana yang menjadi pembentuk dan yang mana yang membentuk. Informasi berupa parent, child, id, nama, dan tier dari seluruh elemen didapatkan melalui proses scraping dari website fandom Little Alchemy 2. Setelah melakukan proses scraping saat pertama kali menjalankan program, penyelesaian masalah dapat diteruskan ke langkah selanjutnya.

##### **3.1.2. Menelusuri Semua Elemen Pembentuk hingga Mencapai Elemen Dasar**

Dalam mencari kombinasi resep yang hanya terdiri dari elemen dasar, terdapat dua algoritma yang digunakan dalam tugas besar ini, yaitu algoritma Depth First Search (DFS) dan Breadth First Search (BFS). Kedua algoritma tersebut menggunakan pendekatan yang berbeda, di mana DFS melakukan traversal kedalaman dan backtracking dan BFS melakukan traversal per level.

##### **3.1.3. Algoritma Penelusuran Graf Elemen dengan Multithreading**

Algoritma penelusuran graf dilakukan dari simpul awal berupa target yang resepnya ingin dicari dan kemudian akan ditelusuri hingga resep teresolusi menjadi empat elemen dasar. Sebuah simpul graf memiliki sejumlah resep yang berupa pasangan elemen. Resep yang ditelusuri valid jika kedua elemen dalam suatu recipe ditemukan memiliki jalur resep yang berujung pada elemen dasar. Untuk mempercepat penelusuran, akan digunakan metode penelusuran yang paralel dengan multithreading.

### **3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma DFS dan BFS.**

Permasalahan Little Alchemy 2 yang melibatkan banyak elemen dan resep ini dapat diselesaikan dengan algoritma DFS dan BFS. Penyelesaian masalah dengan memanfaatkan algoritma DFS dan BFS mengharuskan dilakukannya pemetaan masalah menjadi elemen-elemen dari algoritma DFS dan BFS. Pemetaan masalah dapat dijabarkan sebagai berikut:

#### **3.2.1. Algoritma BFS**

Pada algoritma BFS, semua elemen yang dapat membentuk suatu elemen akan ditelusuri terlebih dahulu. Untuk setiap resep dari suatu elemen, akan dibangkitkan dua elemen pencarian. Dengan menggunakan struktur data queue, hasil pembangkitan elemen akan ditambahkan pada queue. Kemudian dari queue akan diambil elemen terdepan untuk diproses hingga queue tidak memiliki elemen yang dapat diproses. Ketika queue kosong, semua jalur valid menuju target telah ditemukan.

Aspek yang dapat diparalelasi adalah pemrosesan elemen queue. Dalam algoritma BFS yang diimplementasikan, pemrosesan elemen dilakukan per level. Queue pada setiap level memiliki elemen yang berjarak sama dari target. Untuk setiap level, akan dibangkitkan routine berjumlah tetap. Routine yang dibangkitkan akan memproses elemen yang ada dalam queue, dengan routine pertama yang bebas akan mengambil elemen dalam queue hingga queue habis untuk level. Semua routine akan mengirimkan hasil ke dalam queue baru yang akan dijadikan queue proses dalam level selanjutnya dalam iterasi.

Implementasi multithreading dengan goroutine menggunakan channel bertipe QueueElement untuk distribusi elemen di queue ke routine dan untuk agregasi hasil pemrosesan routine. Selain itu, setiap route yang dibangkitkan akan ditunggu penyelesaiannya dengan menggunakan WaitGroup untuk menandakan dapat dimulainya iterasi BFS dengan level lebih mendalam. Semua routine akan keluar dari eksekusi ketika channel distribusi elemen queue telah ditutup karena queue sudah kosong. Ketika semua routine sudah keluar dari eksekusi, dapat dilakukan merging solusi dalam membangun jalur valid. Setelah merging dilakukan, BFS dapat dilanjutkan ke iterasi berikutnya.

Jumlah routine yang dibangkitkan setiap iterasi BFS adalah tetap. Ada dua routine yang melakukan distribusi elemen queue dan pengumpulan hasil pemrosesan. Routine utama program adalah yang melakukan distribusi task berupa elemen queue dengan media Channel. Pengiriman elemen di channel akan memblok routine sehingga ada yang menerima elemen. Sebelum mengirimkan elemen queue, routine utama akan membangkitkan routine yang akan standy untuk menerima elemen hasil pemrosesan routine lain melalui channel.

Selain kedua routine tersebut, routine yang dibangkitkan adalah routine pemroses. Routine pemroses akan mengambil task yang disalurkan melalui channel dan mengirimkan hasilnya melalui channel berbeda. Setiap routine pemroses memiliki state yang berisi informasi resep yang diproses oleh routine tersebut dan nantinya akan dimerge. Setiap routine pemrosesan akan ditunggu keselesaiannya sebagai WaitGroup. Channel bersifat first come, first serve; routine pertama yang siap untuk menerima elemen akan mengambilnya dari channel.

### 3.2.2. Algoritma DFS

Pada algoritma DFS, satu resep dari elemen akan ditelusuri hingga mencapai elemen dasar. Penelusuran suatu simpul elemen akan berlanjut ke resep selanjutnya jika semua jalur valid telah ditemukan untuk resep sebelumnya. Karena resep terdiri dari dua elemen, setiap resep harus menemukan semua path valid ke kedua elemen dalam resep tertentu. Ini adalah salah satu aspek yang dapat diparalelisasi.

Sebelum memasuki algoritma multithreading, akan dikemukakan terlebih dahulu algoritma DFS untuk graf elemen. Algoritma DFS diimplementasikan menggunakan fungsi rekursif. Fungsi ini akan menelusuri resep sebuah simpul dan mengembalikan senarai resep yang setiap resep di dalamnya memiliki informasi jalur valid dari simpul elemen ke elemen dasar. Fungsi ini akan memanggil dirinya sendiri sebanyak dua kali untuk kedua elemen dalam satu resep. Kemudian, setelah kedua fungsi selesai dieksekusi, hasil dari pemanggilan rekursi akan digabungkan dan ditambahkan resep terkait kedua elemen yang baru saja ditelusuri.

Aspek yang dapat mempercepat penelusuran dengan multithreading adalah pemanggilan rekursi fungsi pada kedua elemen suatu resep. Kedua elemen dapat ditelusuri secara bersamaan karena masing elemen tidak mempengaruhi hasil

penelusuran elemen lainnya (meskipun kedua elemen dalam resep dapat menggunakan elemen yang sama, resep menuju elemen tersebut dapat berbeda). Ini mempercepat penelusuran secara eksponensial karena elemen kedua dalam resep tidak perlu menunggu seluruh jalur ke elemen pertama selesai dicari.

Implementasi multithreading dengan goroutine menggunakan Channel sebagai sarana komunikasi antar routine dalam pengiriman sinyal dan Mutex untuk status pencarian yang aksesnya dibagi untuk semua routine. Setiap routine melakukan pencarian resep suatu elemen dan akan membangkitkan dan mengontrol dua routine dalam suatu waktu. Dengan ini, sifat pembangkitan routine dilakukan secara rekursif.

Sebuah routine akan mengirimkan sinyal balik ke routine pembangkitnya setelah routine menemukan jalur valid. Setelah mengirimkan sinyal balik, routine akan menunggu sinyal dari pembangkit untuk mulai meneruskan pencarian ke resep selanjutnya dan kemudian mengirimkan sinyal balik. Routine akan mengirimkan sinyal balik ketika routine menelusuri elemen dasar, atau kedua routine yang dibangkitkan telah mengirim sinyal balik (menandakan kedua komponen dari resep telah ditemukan satu jalur validnya). Saat routine mendapat sinyal untuk meneruskan pencarian, routine akan mengirim sinyal penerusan ke routine pertama hingga routine selesai, kemudian mengirim sinyal penerusan ke routine kedua bersama dengan pembangkitan ulang routine untuk elemen pertama, hingga kedua routine telah selesai. Routine mengirim sinyal balik negatif sebagai penanda penyelesaian. Ketika kedua routine bangkitan telah selesai, semua jalur ke elemen routine ini telah dienumerasi.

Semua routine juga berbagi state dengan proteksi Mutex untuk mencegah race condition. State yang aksesnya terbagi semua routine adalah counter jumlah simpul yang ditelusuri dan senarai resep yang telah ditelusuri. Routine utama yang membangkitkan routine untuk penelusuran elemen target akan mengambil senarai resep dan mengosongkannya ketika menerima sinyal balik untuk meneruskan pencarian jika dibutuhkan beberapa path berbeda.

### 3.3. Fitur Fungsional dan Arsitektur Aplikasi Web

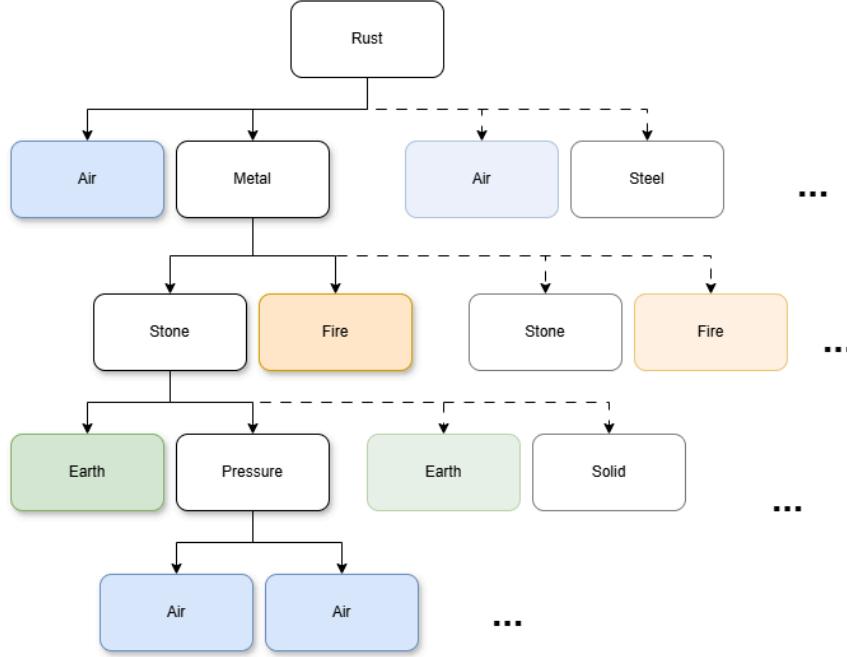
Aplikasi Little Alchemy Recipe dikembangkan untuk membantu pengguna menemukan kombinasi elemen dalam permainan Little Alchemy 2, dengan arsitektur web modern yang membagi sistem menjadi frontend dan backend yang terintegrasi. Frontend

dibangun menggunakan Next.js dan React untuk menciptakan antarmuka yang responsif dan mudah digunakan, dengan Tailwind CSS sebagai solusi styling. Aplikasi menyediakan dua mode pencarian utama: "Single Recipe" untuk menemukan satu jalur resep terpendek menuju elemen target dan "Multiple Recipe" yang menemukan hingga 5 jalur resep berbeda sesuai preferensi pengguna. Pengguna dapat dengan mudah beralih antara algoritma BFS dan DFS melalui tombol pilihan yang disediakan di antarmuka, memberikan fleksibilitas dalam strategi pencarian.

Backend aplikasi diimplementasikan menggunakan Golang dengan framework Gin yang menyediakan performa tinggi dalam menangani permintaan API. Data elemen dan kombinasinya diperoleh melalui scraping dari website Fandom Little Alchemy 2. Pencarian multiple recipe dioptimasi menggunakan teknik multithreading untuk meningkatkan kecepatan pemrosesan, memastikan respons yang cepat meski menangani hingga 5 jalur kombinasi berbeda. Sistem ini juga memastikan bahwa elemen yang digunakan dalam resep selalu berasal dari tier yang lebih rendah dibandingkan elemen target. Backend juga menghitung dan menampilkan metrik performa seperti waktu eksekusi dan jumlah node yang dikunjungi, sementara hasil pencarian divisualisasikan dalam bentuk diagram pohon yang menunjukkan jalur pembentukan dari elemen dasar hingga elemen target.

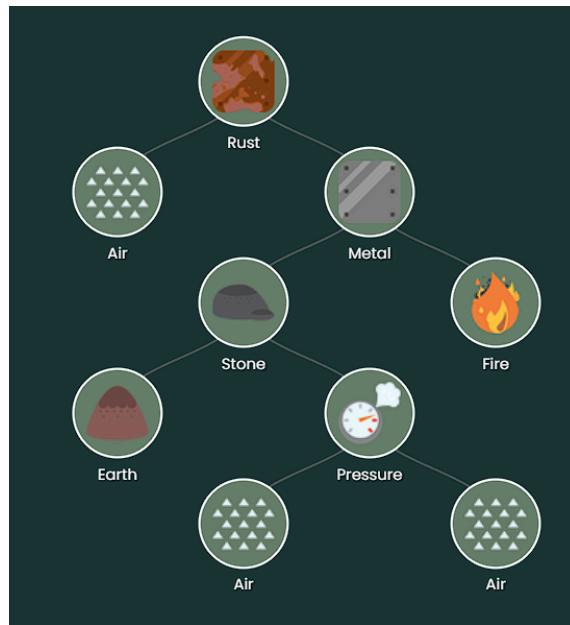
### **3.4. Contoh Ilustrasi Kasus**

#### **3.4.1 DFS**

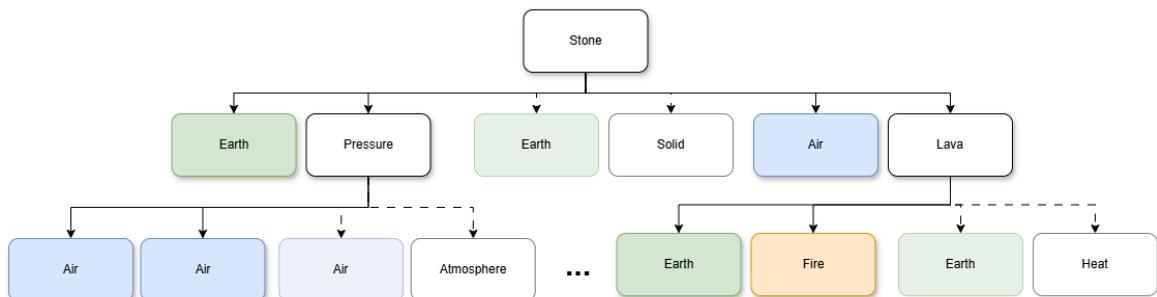


Pada contoh ini akan ditinjau kasus pencarian resep untuk target Rust dengan DFS. Untuk tiap resep akan dibangkitkan kedua elemen yang membentuk resep tersebut. Elemen yang berupa elemen dasar tidak akan membangkitkan elemen baru. Pencarian single recipe akan berakhir ketika semua telah diekspansi menjadi elemen dasar. Dapat diperhatikan resep Air+Steel tidak pernah dibangkitkan karena resep Air+Metal memiliki jalur yang valid. Begitupun resep dari Metal tidak membangkitkan Stone+Fire.

Iterasi	Elemen yang akan dibangkitkan	Pembangkitan
1	Rust	$\text{Rust} \rightarrow \text{Air, Metal}$
2	Air, Metal	Air adalah elemen dasar $\text{Metal} \rightarrow \text{Stone, Fire}$
3	Stone, Fire	$\text{Stone} \rightarrow \text{Earth, Pressure}$ Fire adalah elemen dasar
4	Earth, Pressure	Earth adalah elemen dasar $\text{Pressure} \rightarrow \text{Air, Air}$
5	Air, Air	Air adalah elemen dasar Air adalah elemen dasar



### 3.4.2 BFS



Pada contoh ini akan ditinjau kasus pencarian resep target Stone. Kedua elemen dari setiap resep target akan dibangkitkan terlebih dahulu. Lalu untuk setiap elemen yang dibangkitkan tadi akan membangkitkan dua elemen untuk setiap resepnya. Perhatikan ada resep yang tidak valid karena terdiri dari elemen yang berasal dari tier yang tidak kurang dari elemen yang dibuat. Pada iterasi ketiga, tidak ada simpul elemen yang dibangkitkan lagi, maka BFS dinyatakan selesai.

Iterasi	Queue BFS	Pembangkitan resep
1	Stone	Stone: Earth+Pressure, Air+Lava (Earth+Solid tidak valid)
2	Earth, Pressure, Air, Lava	Earth: Elemen dasar Pressure: Air+Air

		(Air+Atmosphere dan resep lainnya tidak valid) Air: Elemen dasar Lava: Earth+Fire (Earth+Heat dan resep lainnya tidak valid)
3	Air, Air, Earth, Fire	Air: Elemen dasar Air: Elemen dasar Earth: Elemen dasar Fire: Elemen dasar

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1. Struktur Data Program

##### 4.1.1. Struktur Graf Resep

<pre>type ElementNode struct {     ID      int     Name    string     Tier    int     Children []*ElementNode     Recipes [][]*ElementNode }</pre>	Mewakili node dalam graf resep.
<pre>type RecipeGraph struct {     Elements    []*ElementNode     BaseElements []*ElementNode }</pre>	<p>Representasi keseluruhan graf resep.</p> <p>Elements adalah array berisi seluruh elemen dan BaseElements adalah array berisi</p>

##### 4.1.2. Struktur DFS

<pre>type ResultTree struct {     mu    sync.Mutex     path []*Recipe }</pre>	<p>Struktur yang menyimpan jalur hasil pencarian.</p> <ul style="list-style-type: none"> <li>- mu: mutex untuk sinkronisasi akses</li> <li>- path: daftar resep hasil</li> </ul>
<pre>type Recipe struct {     element    *search.ElementNode     composition []*Recipe }</pre>	<p>Struktur untuk menyimpan resep elemen.</p> <p>element: elemen yang dihasilkan</p> <p>composition: elemen-elemen yang dikombinasikan untuk membuat elemen tersebut</p>

#### 4.1.3. Struktur BFS

<pre>type AncestryChain struct {     Element string     Parents *AncestryChain }</pre>	Menyimpan rantai elemen sebelumnya menuju elemen yang saat ini
<pre>type QueueItem struct {     Node          *search.ElementNode     AncestryChain *AncestryChain     Depth         int }</pre>	Struktur untuk menyimpan status pencarian BFS

#### 4.2. Fungsi dan Prosedur

##### 4.2.1 Scraper

Nama Fungsi atau Prosedur	Deskripsi Singkat
ScrapeRecipes()	Fungsi ScrapeRecipes() digunakan untuk mengambil data resep dari permainan Little Alchemy 2 dari wiki fandom. Fungsi ini mengambil data elemen permainan beserta resep-resep pembuatannya, tier setiap elemen, dan juga dapat mengunduh ikon tiap elemen.
GetScrapedRecipesJSON()	Fungsi GetScrapedRecipesJSON() digunakan untuk membaca dan memuat data resep yang telah disimpan dalam format JSON. Fungsi ini membuka file JSON dari lokasi yang ditentukan, menerjemahkan kontennya ke dalam struktur RecipeEntry dan mengembalikan data tersebut.
exportJSON()	Fungsi exportJSON bertugas untuk menyimpan data resep ke dalam file JSON. Fungsi ini menerima parameter berupa struktur RecipeEntry yang berisi data resep, membuat file baru di lokasi yang telah ditentukan, dan meng-encode

	data tersebut ke dalam format JSON.
getHTMLDocument()	Fungsi getHTMLDocument berfungsi untuk mengambil dokumen HTML dari url yang diberikan.
downloadImage()	Fungsi downloadImage digunakan untuk mengunduh gambar dari url tertentu dan menyimpannya ke dalam file lokal.

#### 4.2.2 BFS

Nama Fungsi atau Prosedur	Deskripsi Singkat
getAncSignature()	Fungsi getAncSignature digunakan untuk membuat <i>identifier</i> unik dari <i>ancestry chain</i> elemen. Fungsi ini melakukan rekursi untuk menghasilkan representasi string dari jalur leluhur suatu elemen, dengan format "elemen<-leluhur<-leluhur_sebelumnya" dan seterusnya.
isElemCombUsed()	Fungsi isElemCombUsed digunakan untuk memeriksa apakah kombinasi elemen tertentu sudah pernah digunakan dalam jalur pencarian.
markElemCombUsed()	Fungsi markElemCombUsed bertugas untuk menandai kombinasi elemen sebagai telah digunakan.
isNoRecipe()	Fungsi isNoRecipe digunakan untuk memeriksa apakah suatu elemen tidak memiliki resep pembuatan, yang berarti elemen tersebut merupakan elemen dasar atau elemen spesial yang baru terbuka ketika mencapai kondisi tertentu.
isBaseElement()	Fungsi isBaseElement berfungsi untuk memeriksa apakah suatu elemen merupakan elemen dasar permainan (Air, Earth, Fire, atau Water).
ReverseBFS()	Fungsi ReverseBFS melakukan pencarian <i>breadth-first</i> terbalik untuk menemukan resep beserta elemen-elemen pembentuk

	dari elemen target. Fungsi ini menggunakan paralelisasi dengan beberapa goroutine untuk mempercepat pencarian.
ResetCaches()	Fungsi ResetCaches digunakan untuk mengatur ulang cache kombinasi elemen yang telah digunakan dengan cara membersihkan map usedElemComb
ProcessQueue()	Fungsi ProcessQueue adalah fungsi worker yang dijalankan sebagai goroutine untuk memproses item dalam antrian BFS. Fungsi ini menerima channel tugas yang berisi item untuk diproses, channel untuk mengirim item ke frontier berikutnya, pointer ke struktur hasil, dan wait group untuk sinkronisasi.

#### 4.2.3 DFS

Nama Fungsi atau Prosedur	Deskripsi Singkat
DFS()	Fungsi DFS melakukan pencarian <i>depth-first</i> untuk menemukan jalur pembuatan elemen target dalam permainan Little Alchemy 2. Fungsi ini memiliki dua mode operasi berbeda berdasarkan parameter maxPaths. Jika maxPaths adalah 1, fungsi akan memanggil findSinglePath untuk menemukan satu jalur pembuatan. Namun jika lebih dari 1, fungsi akan memanggil findMultiplePaths untuk menemukan beberapa jalur alternatif untuk membuat elemen target.
mergeTree()	Fungsi mergeTree digunakan untuk menggabungkan dua pohon hasil menjadi satu.
findSinglePath()	Fungsi findSinglePath melakukan pencarian rekursif untuk menemukan satu jalur pembuatan dari elemen-elemen dasar ke elemen target.

findMultiplePaths()	Fungsi findMultiplePaths digunakan untuk menemukan beberapa jalur pembuatan berbeda untuk elemen target.
findPath()	Fungsi findPath adalah implementasi konkuren dari pencarian jalur yang digunakan oleh findMultiplePaths. Fungsi ini menggunakan channel untuk komunikasi antar goroutine dan memungkinkan pencarian paralel dari berbagai resep dan jalur.
ParseCraftingPathToJson()	Fungsi ParseCraftingPathToJson mengubah struktur data ResultTree yang berisi jalur pembuatan menjadi format JSON.

### 4.3. Tata Cara Penggunaan Program

#### 4.3.1. Cara Menjalankan Program

##### a. Tanpa Docker

###### 1. Clone repository:

```
git clone https://github.com/naylzhra/Tubes2_FullRustAlchemist.git
```

###### 2. Lakukan instalasi dependency

```
cd src/frontend  
npm install
```

###### 3. Pada terminal yang sama, build frontend

```
npm run dev
```

###### 4. Run backend pada terminal berbeda.

```
cd src/backend  
go run .
```

###### 5. Kemudian, buka localhost:3000 pada browser: http://localhost:3000/

###### 6. Pilih mode pencarian resep yang diinginkan (single recipe/ multiple recipe)

###### 7. Masukkan input sesuai kebutuhan pencarian kemudian klik tombol search

##### b. Menggunakan Docker

###### 1. Clone repository

```
git clone https://github.com/naylzhra/Tubes2_FullRustAlchemist.git
```

###### 2. Buka dan jalankan aplikasi docker desktop

###### 3. Build dan jalankan docker

```
docker compose up --build
```

###### 4. Kemudian, buka localhost:3000 pada browser

```
http://localhost:3000/
```

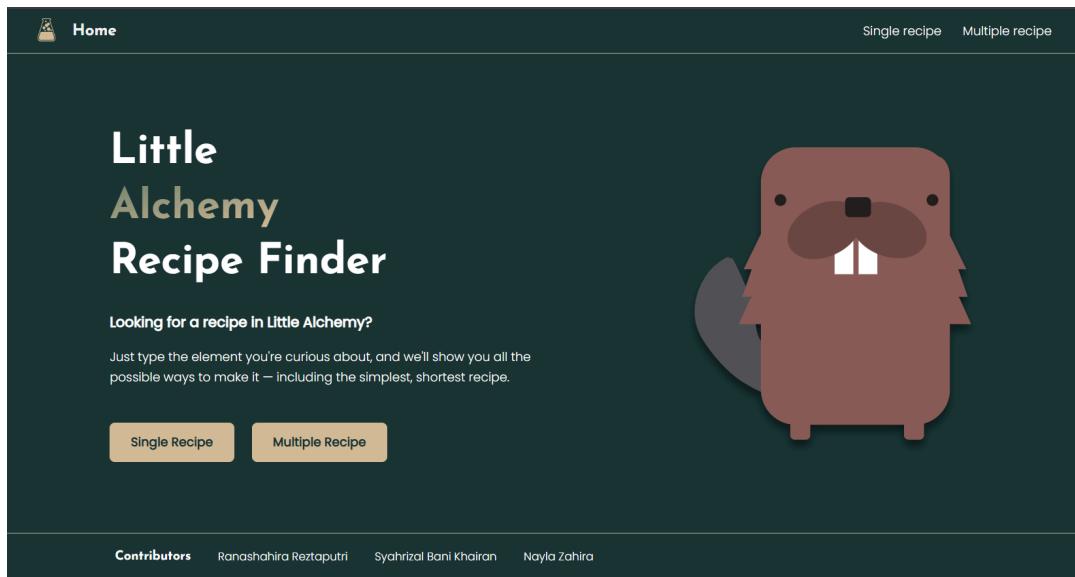
###### 5. Pilih mode pencarian resep yang diinginkan (single recipe/ multiple recipe)

6. Masukkan input sesuai kebutuhan pencarian kemudian klik tombol search
- c. Dengan hasil deploy  
<https://little-alchemy2-recipe-finder.vercel.app/>

#### 4.3.2. Interface Program

- a. Homepage

Laman ini merupakan laman utama yang menyediakan 2 pilihan mode pencarian kepada pengguna, yaitu *single recipe* dan *multiple recipe*.



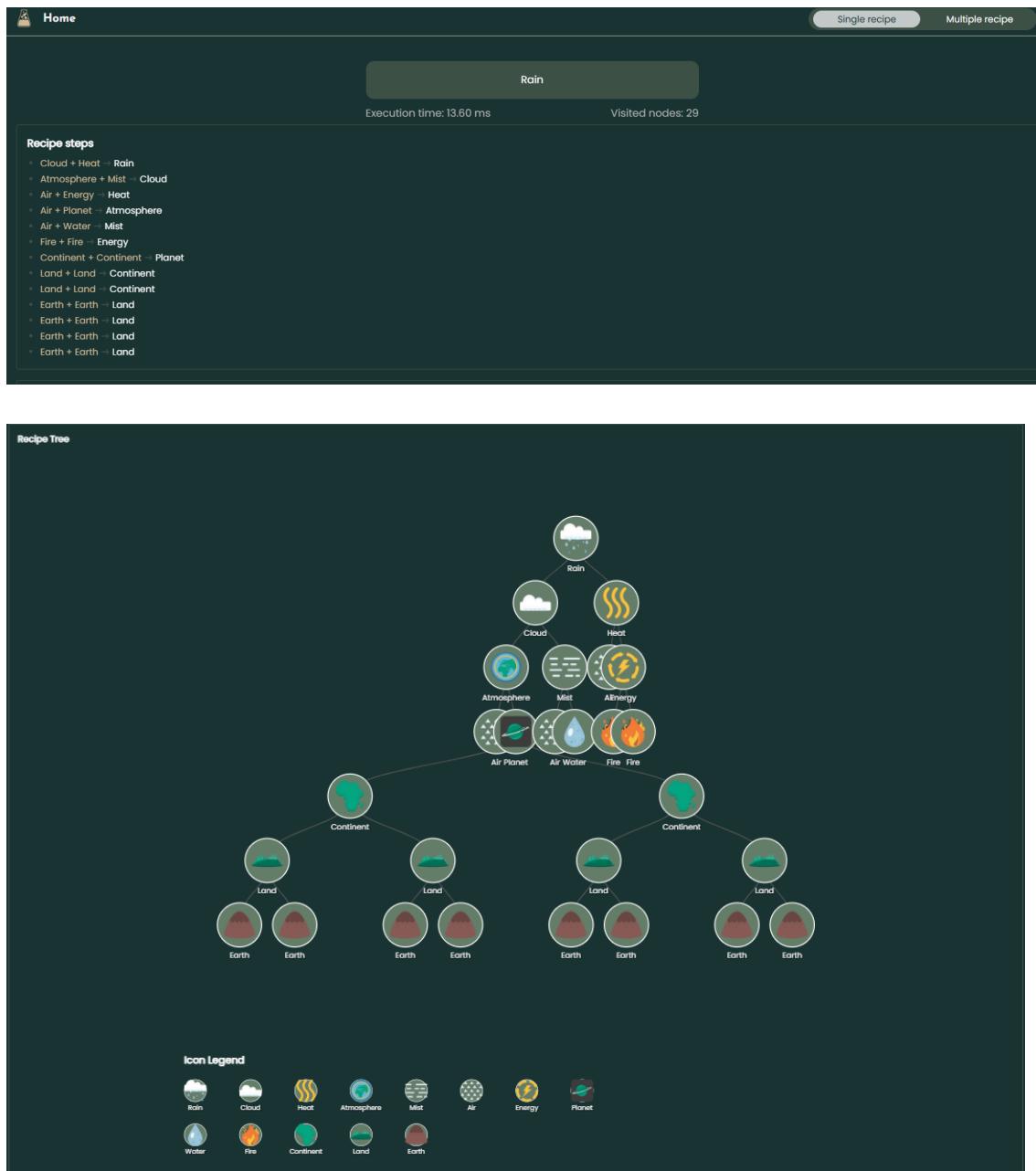
- b. Single Recipe Query Page

Laman ini berfungsi sebagai antarmuka pencarian resep *single* dari elemen Little Alchemy 2. Pengguna dapat menentukan elemen yang ingin dicari dan mengatur jumlah resep yang akan ditampilkan. Terdapat *dropdown* untuk memilih algoritma pencarian (BFS/DFS), kolom pencarian untuk memasukkan nama elemen target, dan tombol "Search" yang akan aktif setelah semua parameter pencarian terisi dengan lengkap.



### c. Single Recipe Result Page

Laman ini berfungsi sebagai antarmuka hasil pencarian resep *single* dari elemen yang sebelumnya dimasukkan sebagai input. Bagian atas menunjukkan informasi tentang pencarian yang telah dilakukan, termasuk waktu eksekusi algoritma dan jumlah node yang dikunjungi selama pencarian. Bagian hasilnya menampilkan langkah-langkah kombinasi elemen pembentuk dan memvisualisasikan resep dalam bentuk struktur pohon. Di bagian bawah tampilan pohon terdapat legenda ikon untuk membantu pengguna mengidentifikasi setiap elemen.



#### d. Multiple Recipe Query Page

Laman ini berfungsi sebagai antarmuka pencarian resep berjumlah banyak dari elemen Little Alchemy 2. Pengguna dapat menentukan elemen yang ingin dicari dan mengatur jumlah resep yang akan ditampilkan. Terdapat *dropdown* untuk memilih algoritma pencarian (BFS/DFS), kolom input untuk menentukan jumlah maksimum resep yang akan ditampilkan, kolom pencarian untuk memasukkan nama elemen target, dan tombol "Search" yang akan aktif setelah semua parameter pencarian terisi dengan lengkap.



e. Multiple Recipe ResultPage

localhost:3000/multiple-recipe/result?element=Supernova&algo=bfs&max=2

Single recipe    Multiple recipe

Found 2 recipes for Supernova

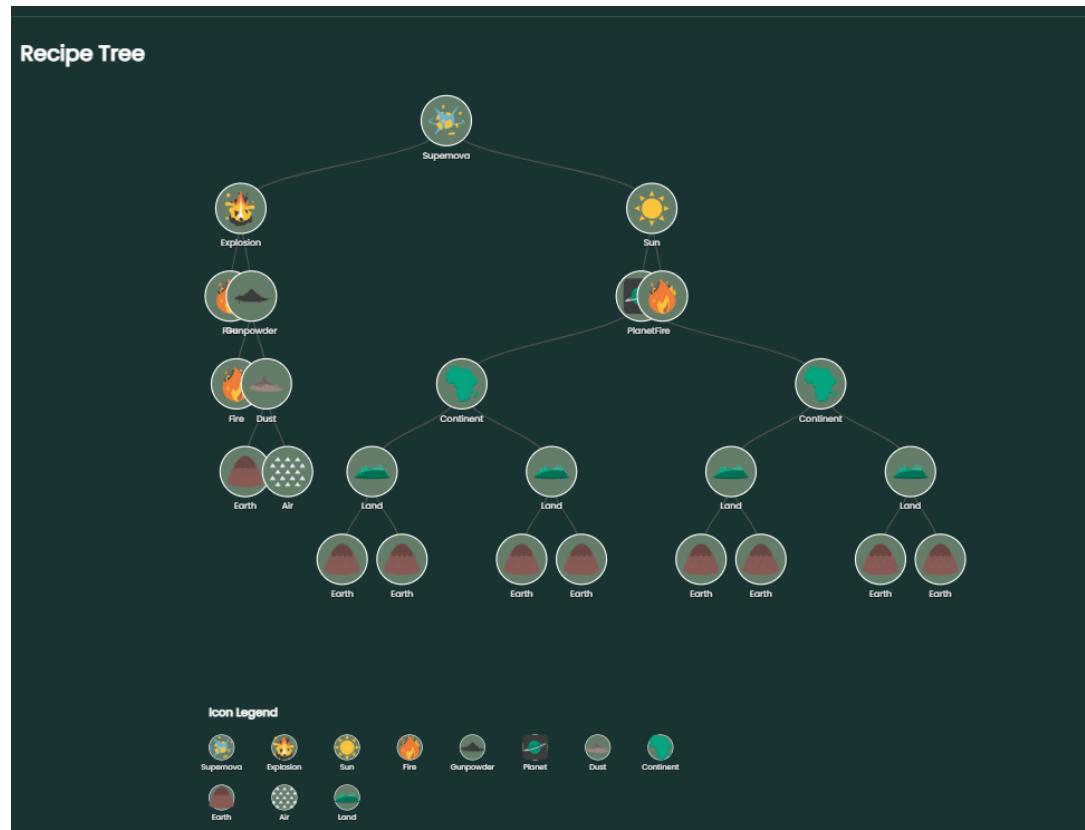
**Supernova**

Execution time: 11 ms    Visited nodes: 31

Path #1

**Recipe steps**

- Explosion + Sun → Supernova
- Fire + Gunpowder → Explosion
- Planet + Fire → Sun
- Fire + Dust → Gunpowder
- Continent + Continent → Planet
- Earth + Air → Dust
- Land + Land → Continent
- Land + Land → Continent
- Earth + Earth → Land

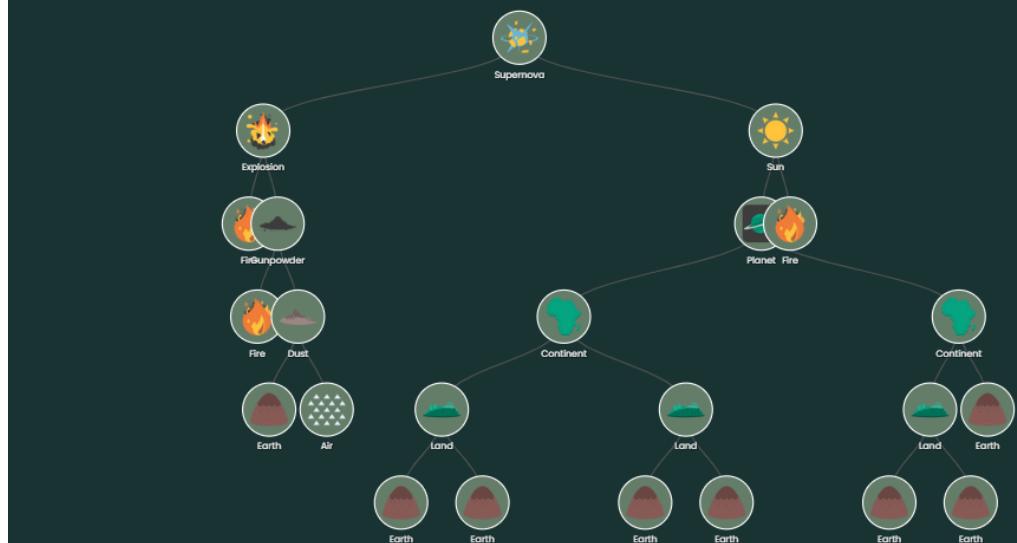


## Path #2

### Recipe steps

- Explosion + Sun → Supernova
- Fire + Gunpowder → Explosion
- Planet + Fire → Sun
- Fire + Dust → Gunpowder
- Continent + Continent → Planet
- Earth + Air → Dust
- Land + Land → Continent
- Land + Earth → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Earth + Earth → Land

### Recipe Tree



Icon Legend



## 4.4. Hasil pengujian program

### 4.4.1. Test Case 1

#### 4.4.1.1. Single Recipe - BFS

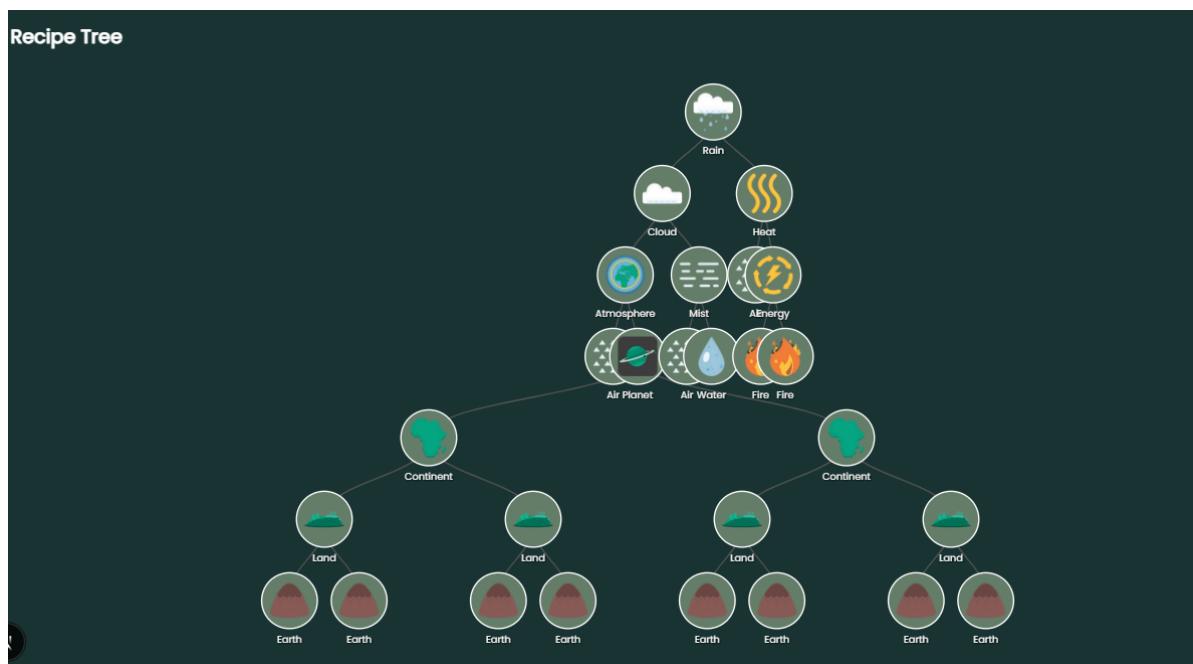
**Home**      Single recipe      Multiple recipe

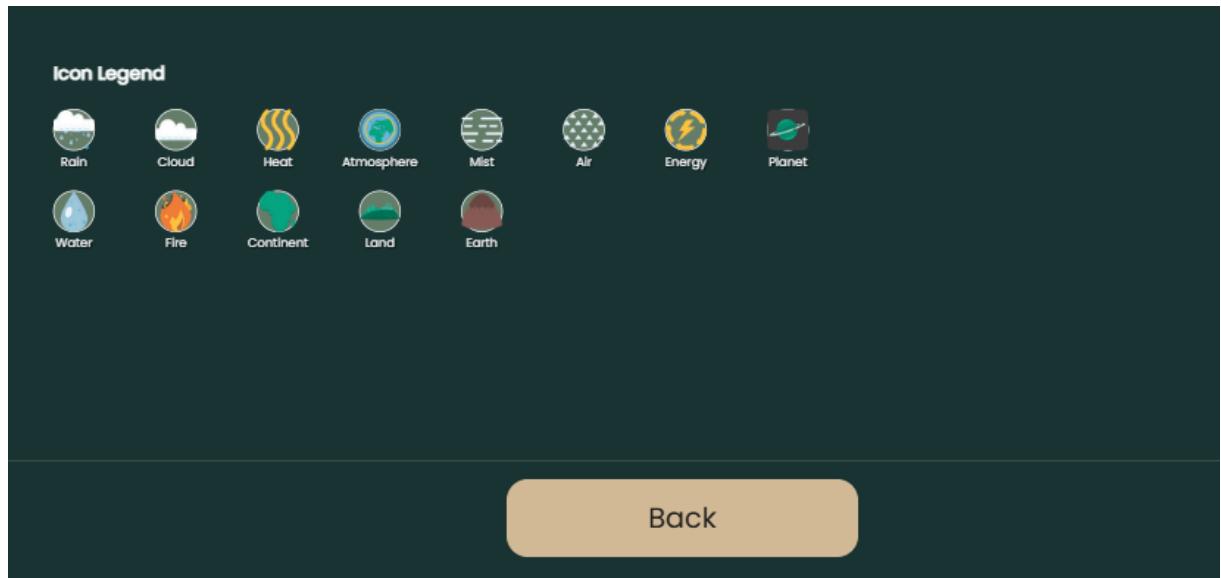
Rain

Execution time: 16.40 ms      Visited nodes: 29

**Recipe steps**

- Cloud + Heat → Rain
- Atmosphere + Mist → Cloud
- Air + Energy → Heat
- Air + Planet → Atmosphere
- Air + Water → Mist
- Fire + Fire → Energy
- Continent + Continent → Planet
- Land + Land → Continent
- Land + Land → Continent
- Earth + Earth → Land





#### 4.4.1.2. Single Recipe - DFS

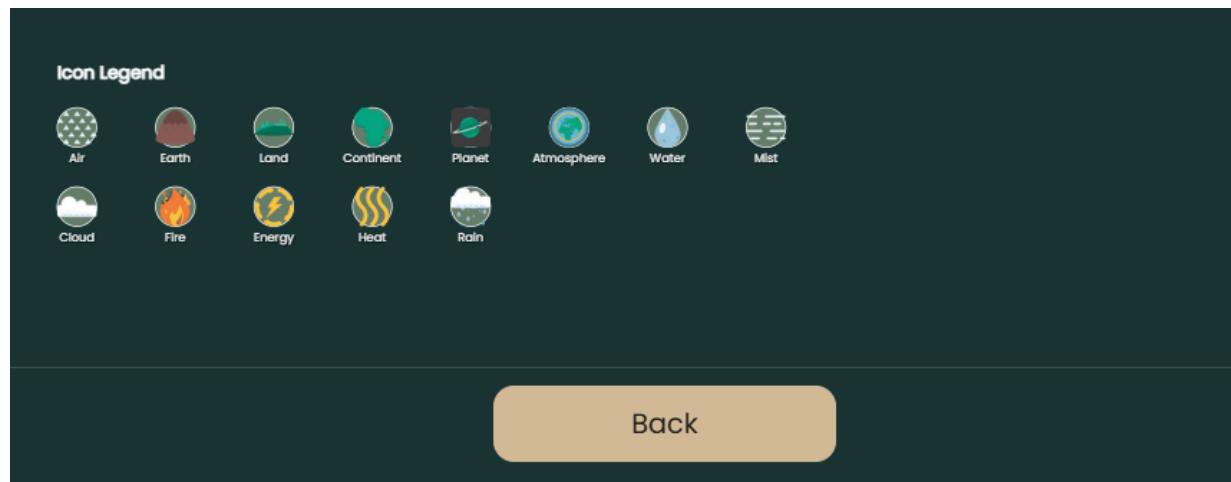
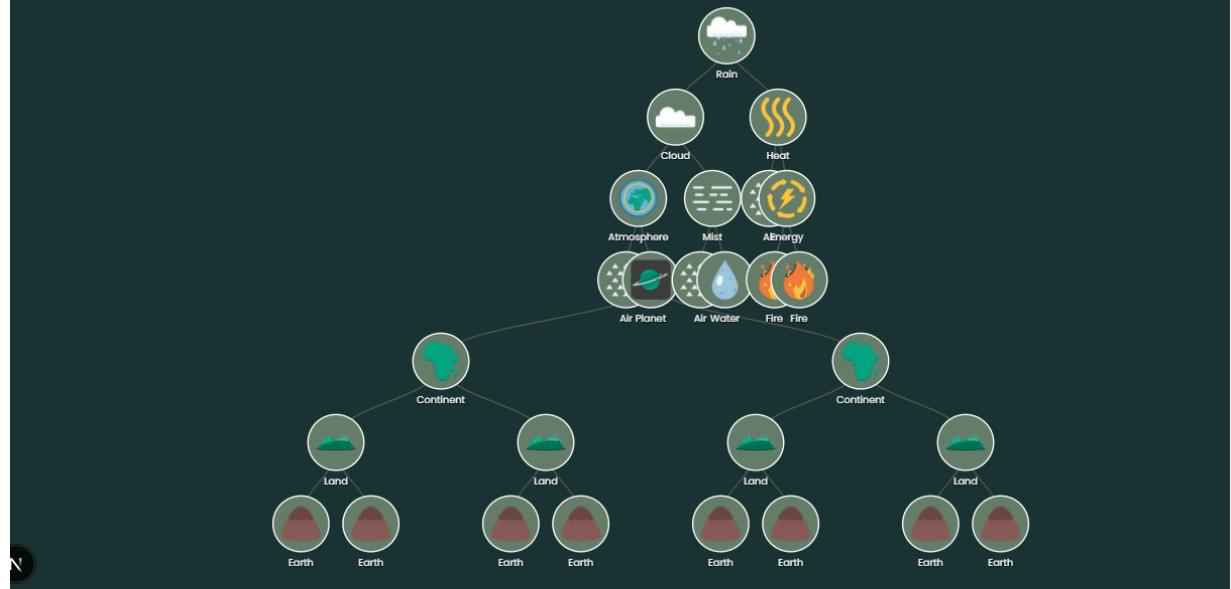
A screenshot of a web browser displaying the results of a single recipe search using the Depth-First Search (DFS) algorithm. The URL in the address bar is "localhost:3000/single-recipe/result?element=Rain&algo=DFS".

The page shows a green header bar with the word "Rain". Below it, the text "Time execution: 23.90 ms" and "Visited nodes: 27".

**Recipe steps**

- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Continent + Continent → Planet
- Air + Planet → Atmosphere
- Air + Water → Mist
- Atmosphere + Mist → Cloud
- Fire + Fire → Energy
- Air + Energy → Heat
- Cloud + Heat → Rain

### Recipe Tree

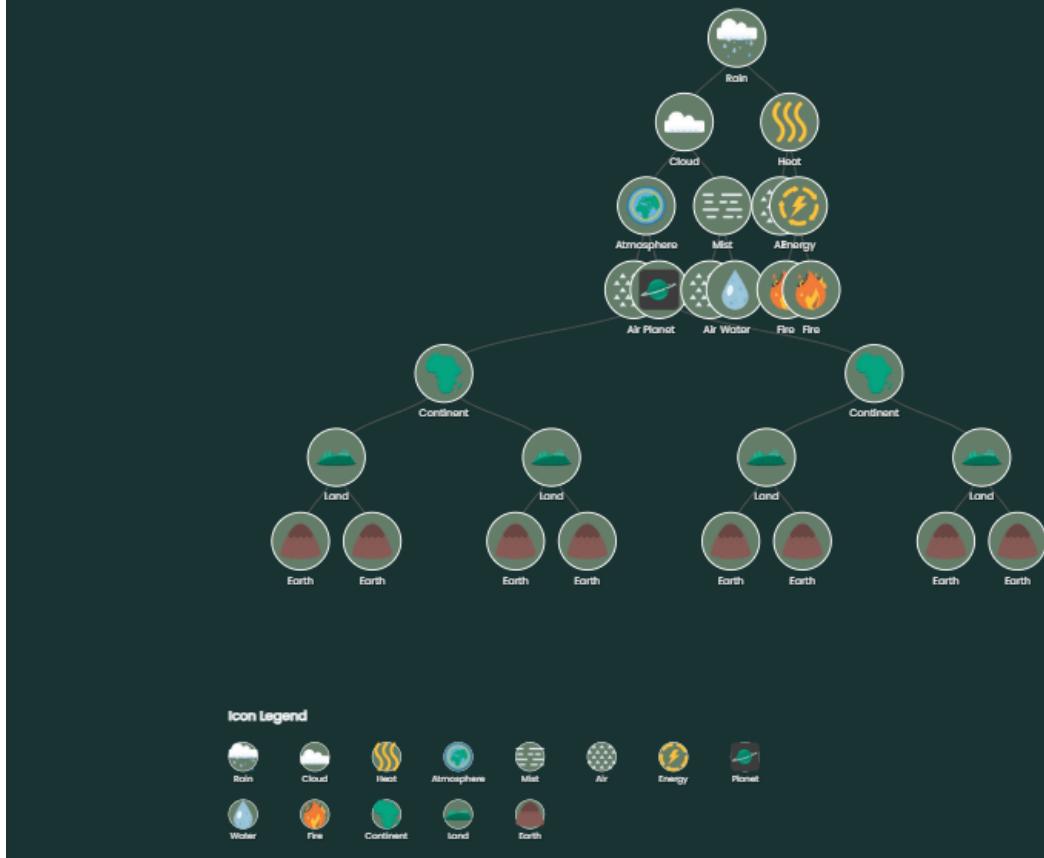


#### 4.4.1.3. Multiple Recipe - BFS

The screenshot shows a web browser window with the URL `localhost:3000/multiple-recipe/result?element=Rain&algo=bfs&max=3`. The page title is "Home". There are two tabs at the top: "Single recipe" and "Multiple recipe". The main content area displays the results for "Rain". It shows "Found 3 recipes for Rain" and a large button labeled "Rain". Below the button, it says "Execution time: 11 ms" and "Visited nodes: 28". A section titled "Path #1" lists the "Recipe steps" as follows:

- Cloud + Heat → Rain
- Atmosphere + Mist → Cloud
- Air + Energy → Heat
- Air + Planet → Atmosphere
- Air + Water → Mist
- Fire + Fire → Energy
- Continent + Continent → Planet
- Land + Land → Continent
- Land + Land → Continent
- Earth + Earth → Land

## Recipe Tree

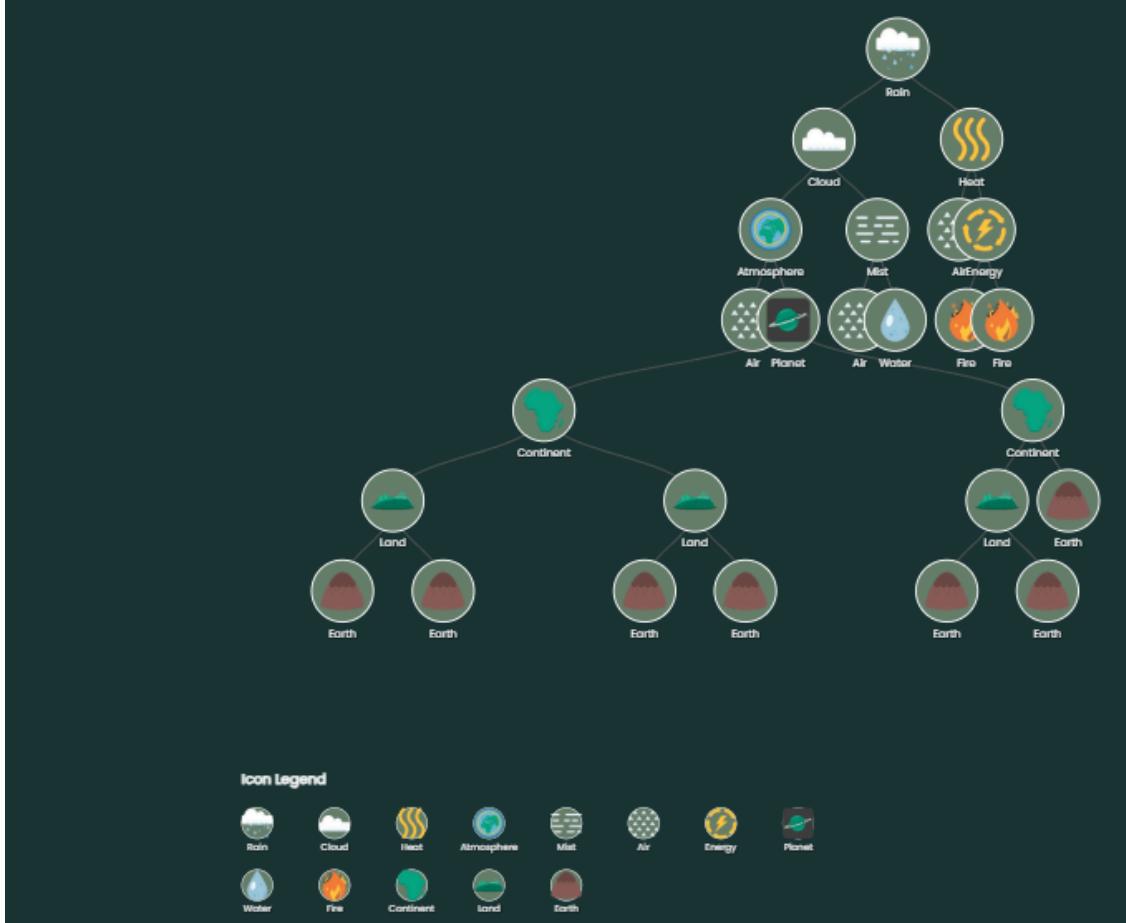


## Path #2

### Recipe steps

- Cloud + Heat → Rain
- Atmosphere + Mist → Cloud
- Air + Energy → Heat
- Air + Planet → Atmosphere
- Air + Water → Mist
- Fire + Fire → Energy
- Continent + Continent → Planet
- Land + Land → Continent
- Land + Earth → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Earth + Earth → Land

## Recipe Tree

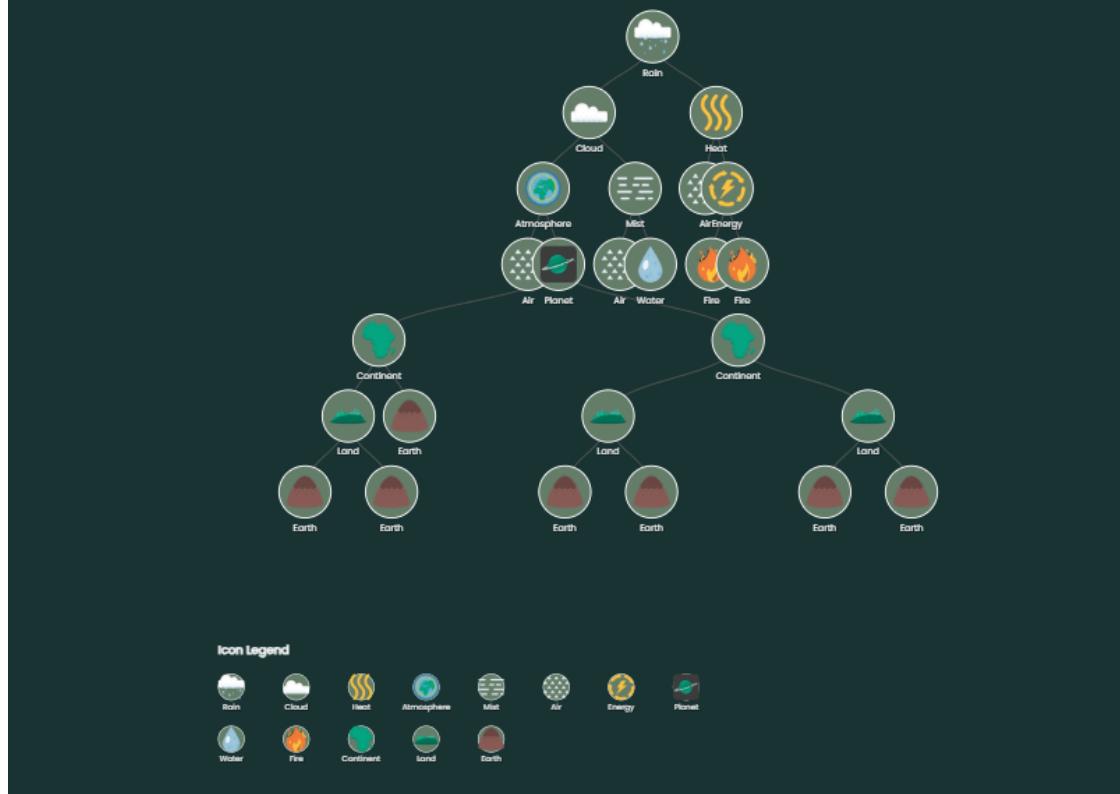


## Path #3

### Recipe steps

- Cloud + Heat → Rain
- Atmosphere + Mist → Cloud
- Air + Energy → Heat
- Air + Planet → Atmosphere
- Air + Water → Mist
- Fire + Fire → Energy
- Continent + Continent → Planet
- Land + Earth → Continent
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Earth + Earth → Land

## Recipe Tree



### 4.4.1.4. Multiple Recipe - DFS

localhost:3000/multiple-recipe/result?element=Rain&algo=dfs&max=3

Home Single recipe Multiple recipe

Found 3 recipes for Rain

Rain

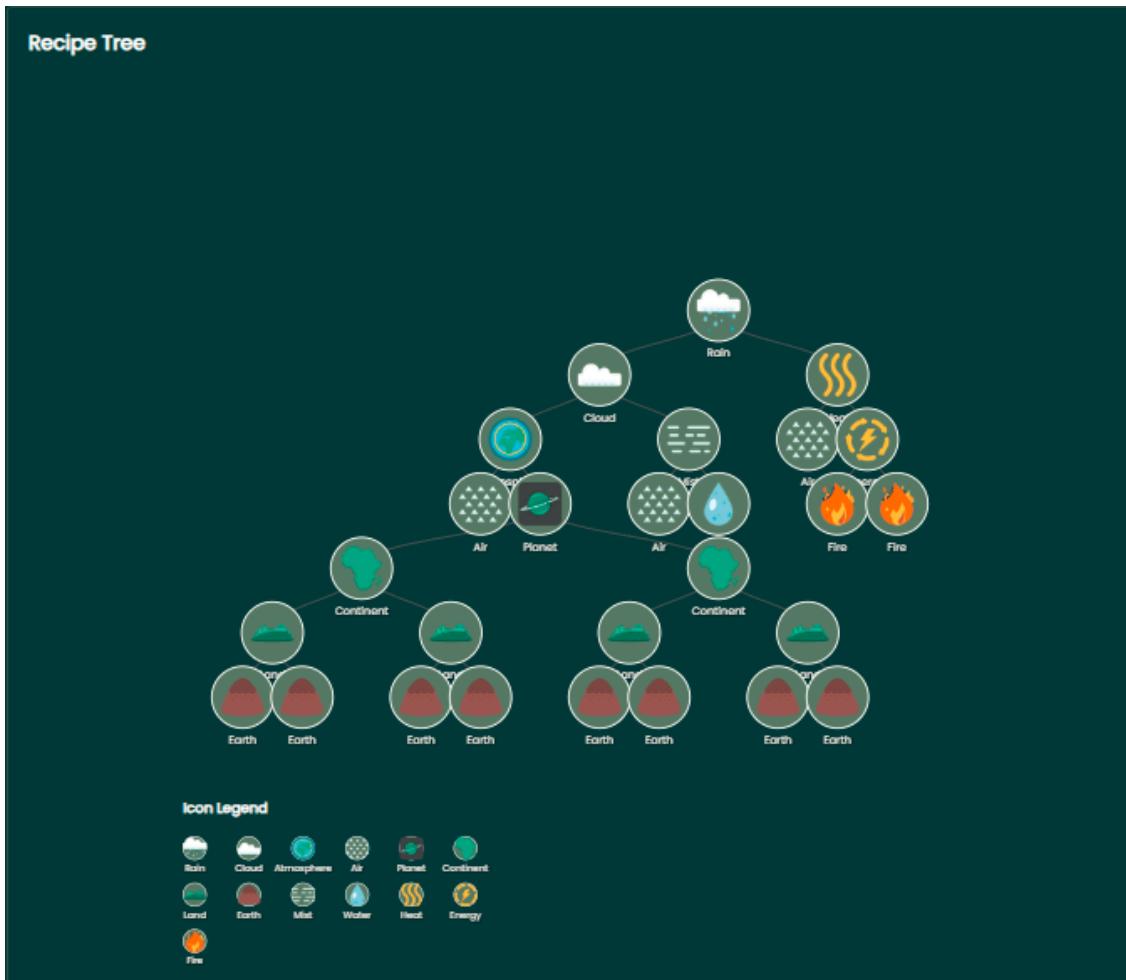
Execution time: 54 ms Visited nodes: 117

Path #1

**Recipe steps**

- Cloud + Heat → Rain
- Atmosphere + Mist → Cloud
- Air + Planet → Atmosphere
- Continent + Continent → Planet
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Air + Water → Mist
- Air + Energy → Heat
- Fire + Fire → Energy

### Recipe Tree

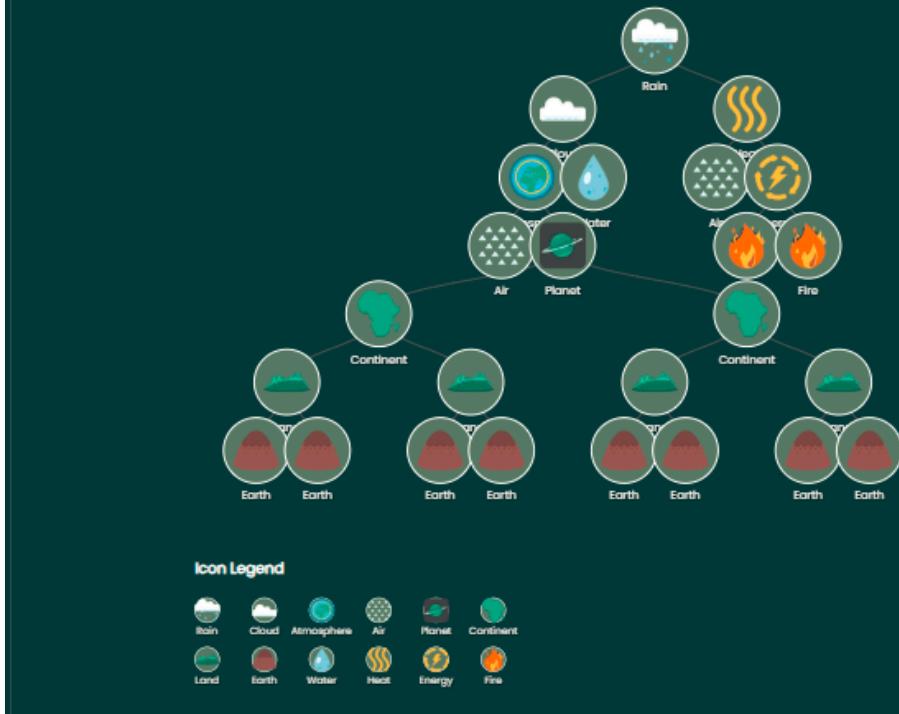


### Path #2

#### Recipe steps

- Cloud + Heat → Rain
- Atmosphere + Water → Cloud
- Air + Planet → Atmosphere
- Continent + Continent → Planet
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Air + Energy → Heat
- Fire + Fire → Energy

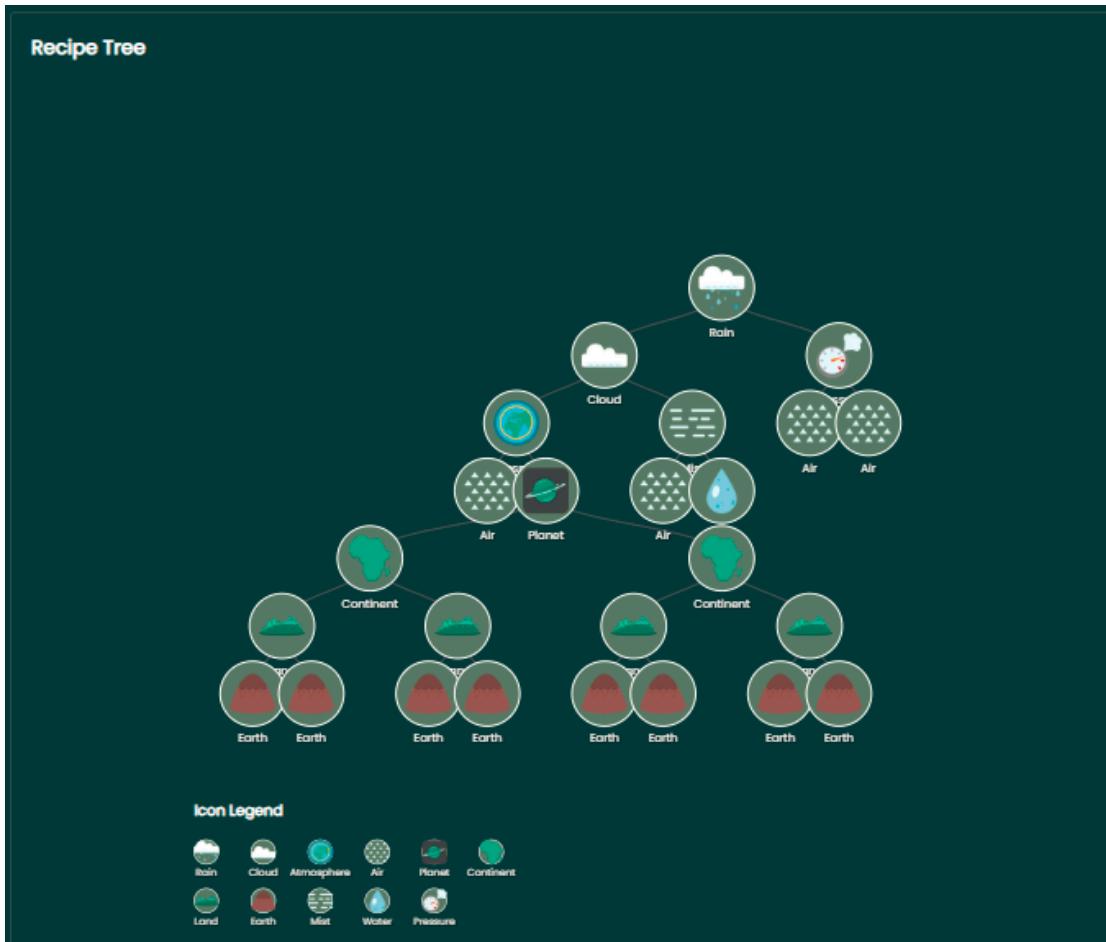
## Recipe Tree



## Path #3

### Recipe steps

- Cloud + Pressure → Rain
- Atmosphere + Mist → Cloud
- Air + Planet → Atmosphere
- Continent + Continent → Planet
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Air + Water → Mist
- Air + Air → Pressure



#### 4.4.2. Test Case 2

Pada test case ini, kami melakukan penelusuran resep dari elemen tier 5, yaitu “Supernova” dengan 4 variasi:

#### 4.4.2.1. Single Recipe - BFS

localhost:3000/single-recipe/result?element=Supernova&algo=BFS

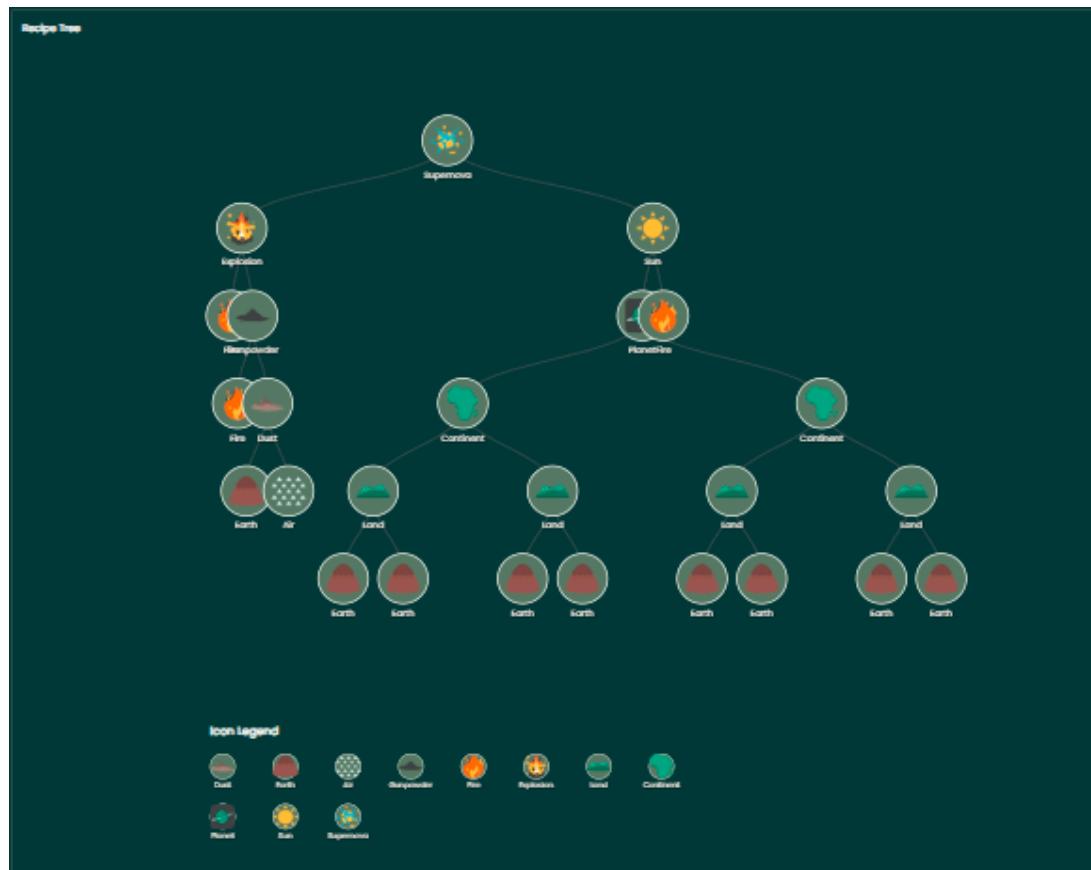
**Home** Single recipe Multiple recipe

**Supernova**

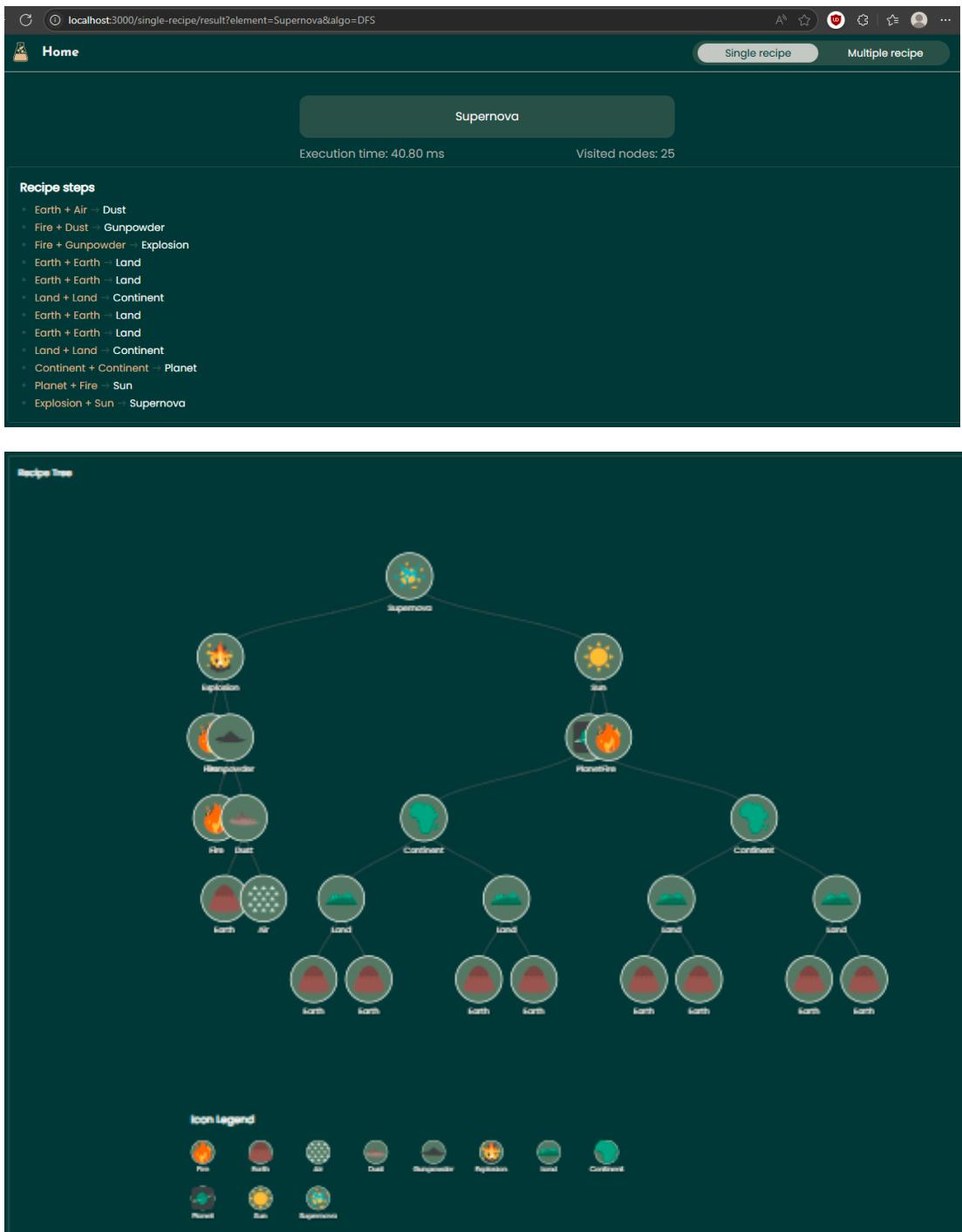
Execution time: 20.70 ms Visited nodes: 31

**Recipe steps**

- Explosion + Sun → Supernova
- Fire + Gunpowder → Explosion
- Planet + Fire → Sun
- Fire + Dust → Gunpowder
- Continent + Continent → Planet
- Earth + Air → Dust
- Land + Land → Continent
- Land + Land → Continent
- Earth + Earth → Land



#### 4.4.2.2. Single Recipe - DFS



#### 4.4.2.3. Multiple Recipe - BFS

localhost:3000/multiple-recipe/result?element=Supernova&algo=bfs&max=2

Single recipe    Multiple recipe

Found 2 recipes for Supernova

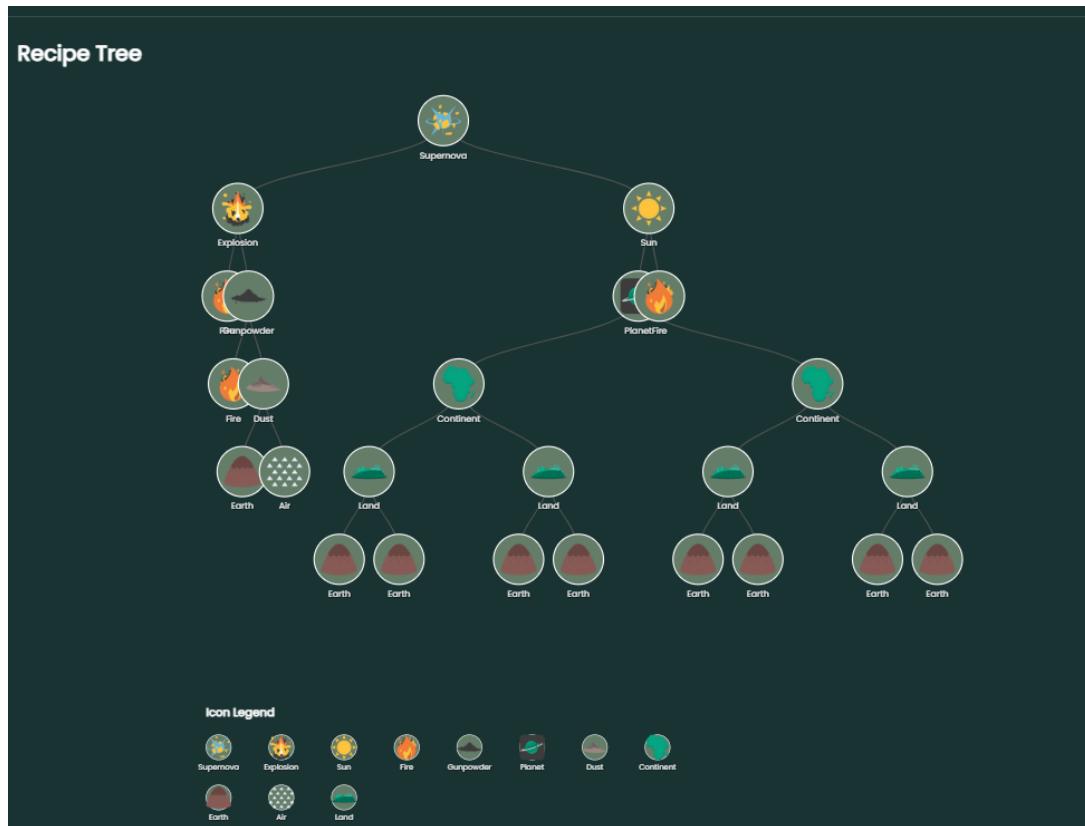
Supernova

Execution time: 11 ms    Visited nodes: 31

Path #1

Recipe steps

- Explosion + Sun → Supernova
- Fire + Gunpowder → Explosion
- Planet + Fire → Sun
- Fire + Dust → Gunpowder
- Continent + Continent → Planet
- Earth + Air → Dust
- Land + Land → Continent
- Land + Land → Continent
- Earth + Earth → Land

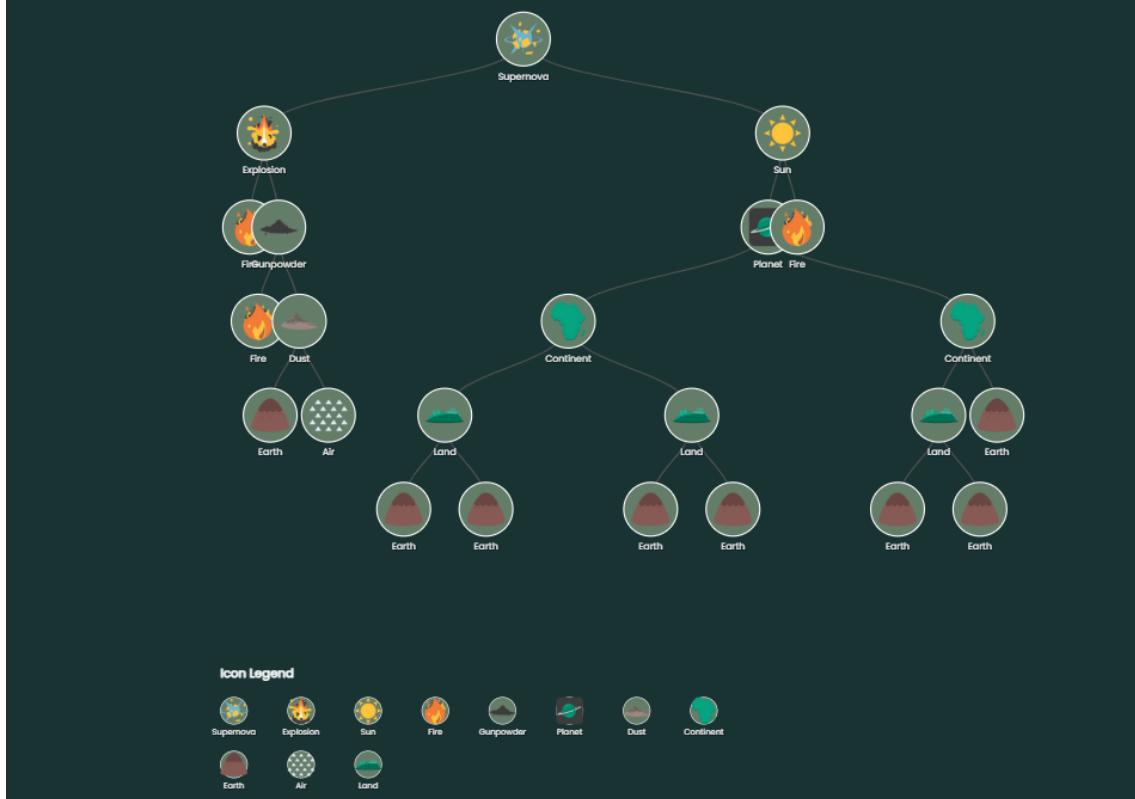


## Path #2

### Recipe steps

- Explosion + Sun → Supernova
- Fire + Gunpowder → Explosion
- Planet + Fire → Sun
- Fire + Dust → Gunpowder
- Continent + Continent → Planet
- Earth + Air → Dust
- Land + Land → Continent
- Land + Earth → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Earth + Earth → Land

### Recipe Tree



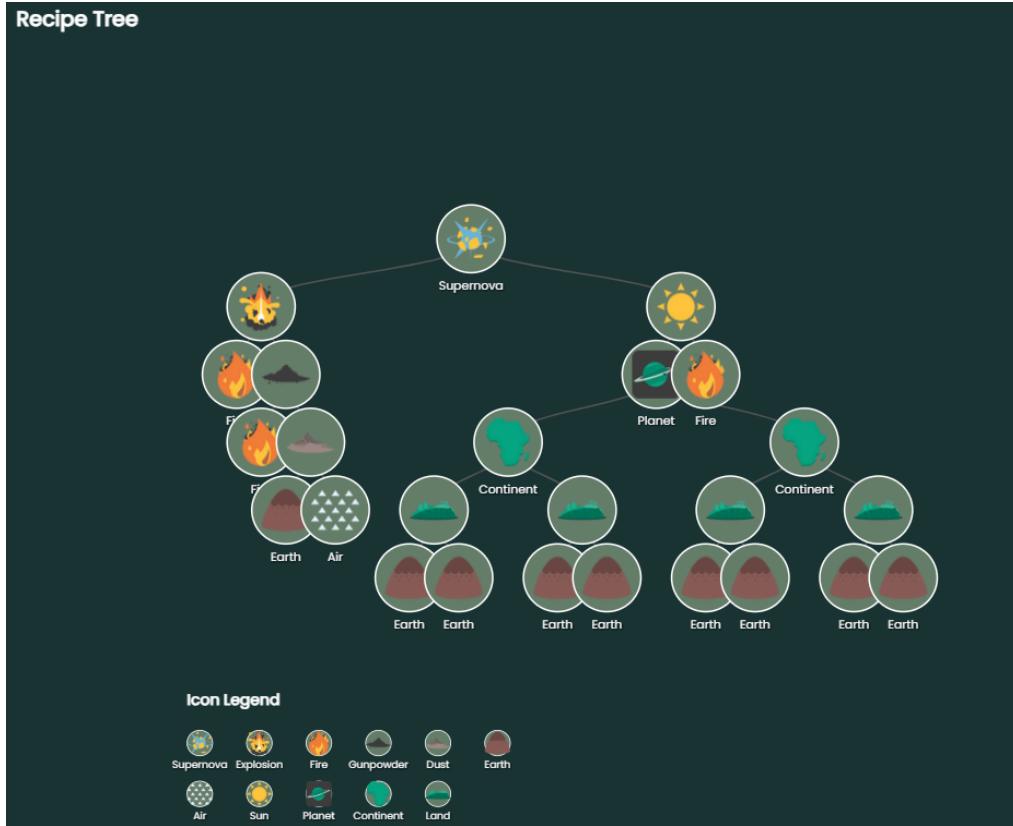
#### 4.4.2.4. Multiple Recipe - DFS

The screenshot shows a web interface for a search or recipe application. The URL in the browser is `localhost:3000/multiple-recipe/result?element=Supernova&algo=dfs&max=2`. The interface has a dark theme with a navigation bar at the top. The main content area displays the results for the element **Supernova**, stating "Found 2 recipes for Supernova". Below this, a large button labeled **Supernova** is shown. Performance metrics are listed: "Execution time: 16 ms" and "Visited nodes: 41". A section titled "Path #1" details the "Recipe steps" required to reach the target element:

**Recipe steps**

- Explosion + Sun → **Supernova**
- Fire + Gunpowder → **Explosion**
- Fire + Dust → **Gunpowder**
- Earth + Air → **Dust**
- Planet + Fire → **Sun**
- Continent + Continent → **Planet**
- Land + Land → **Continent**
- Earth + Earth → **Land**
- Earth + Earth → **Land**
- Land + Land → **Continent**
- Earth + Earth → **Land**
- Earth + Earth → **Land**

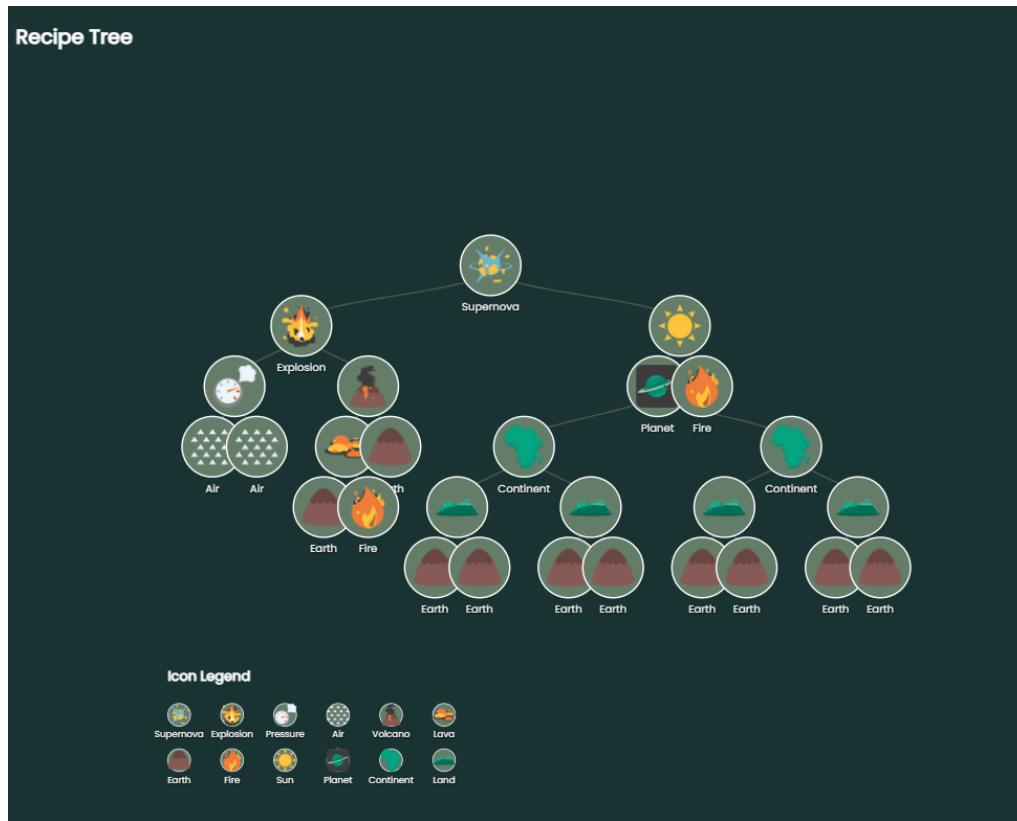
### Recipe Tree



### Path #2

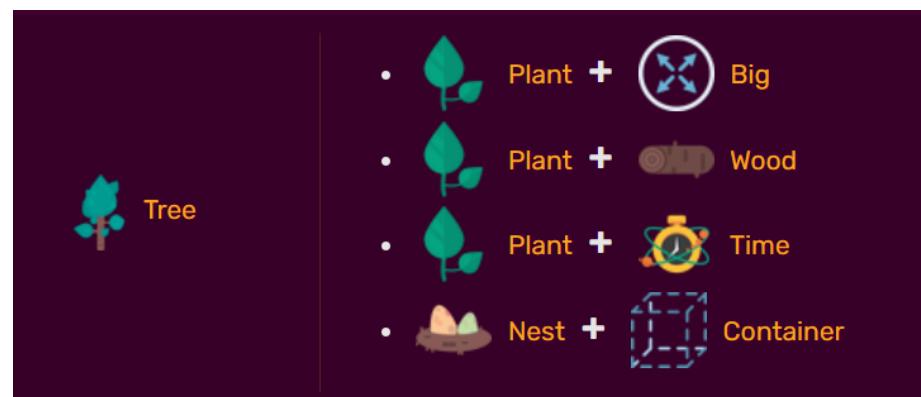
#### Recipe steps

- Explosion + Sun → Supernova
- Pressure + Volcano → Explosion
- Air + Air → Pressure
- Lava + Earth → Volcano
- Earth + Fire → Lava
- Planet + Fire → Sun
- Continent + Continent → Planet
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land
- Land + Land → Continent
- Earth + Earth → Land
- Earth + Earth → Land

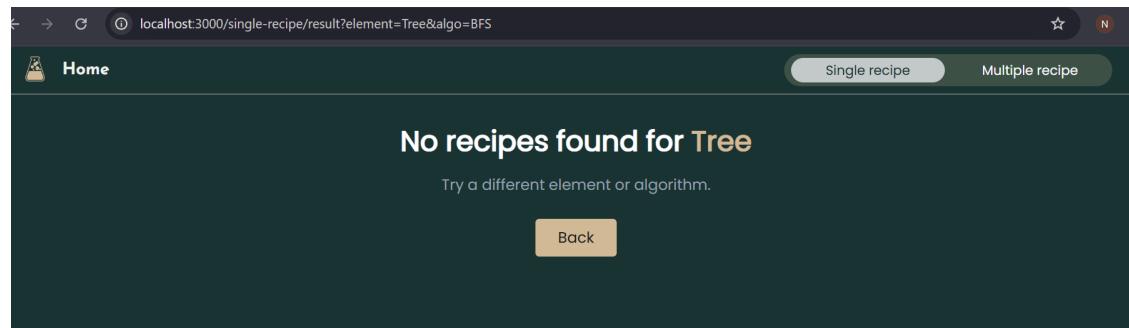


#### 4.3.3. Test Case 3

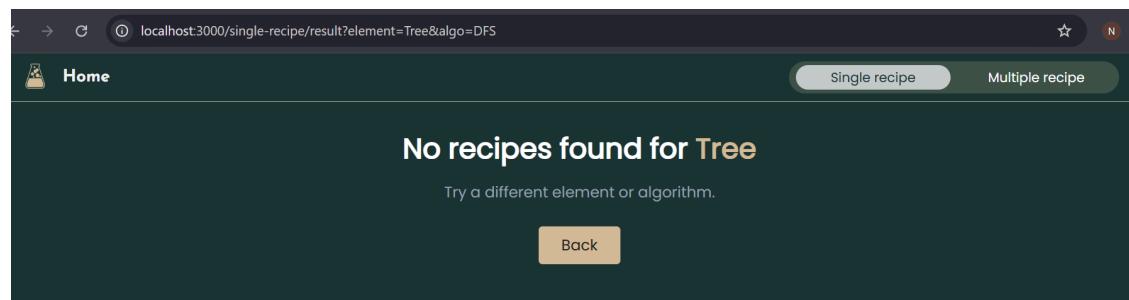
Pada test case ini kami melakukan penelusuran resep dari elemen ‘Tree’ yang merupakan tier 9. Dalam pengujian ini, ditemukan kasus khusus dimana tidak terdapat resep valid. Hal ini disebabkan oleh keberadaan elemen tier 10 (wood, big, container) dan elemen spesial seperti time yang menghambat konstruksi resep yang memenuhi syarat, yaitu harus dibentuk dari elemen dengan tier yang lebih rendah.



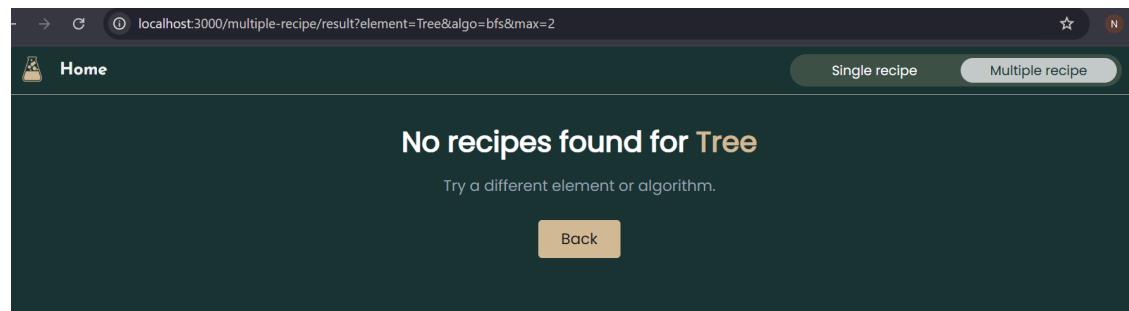
#### 4.3.3.1. Single Recipe - BFS



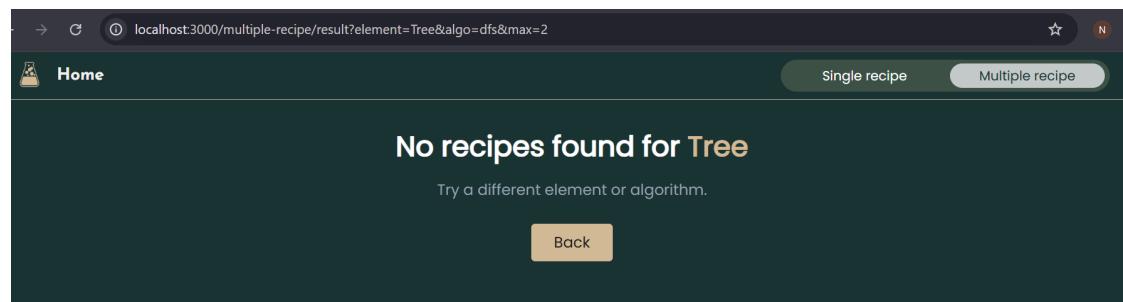
#### 4.3.3.2. Single Recipe - DFS



#### 4.3.3.3. Multiple Recipe - BFS

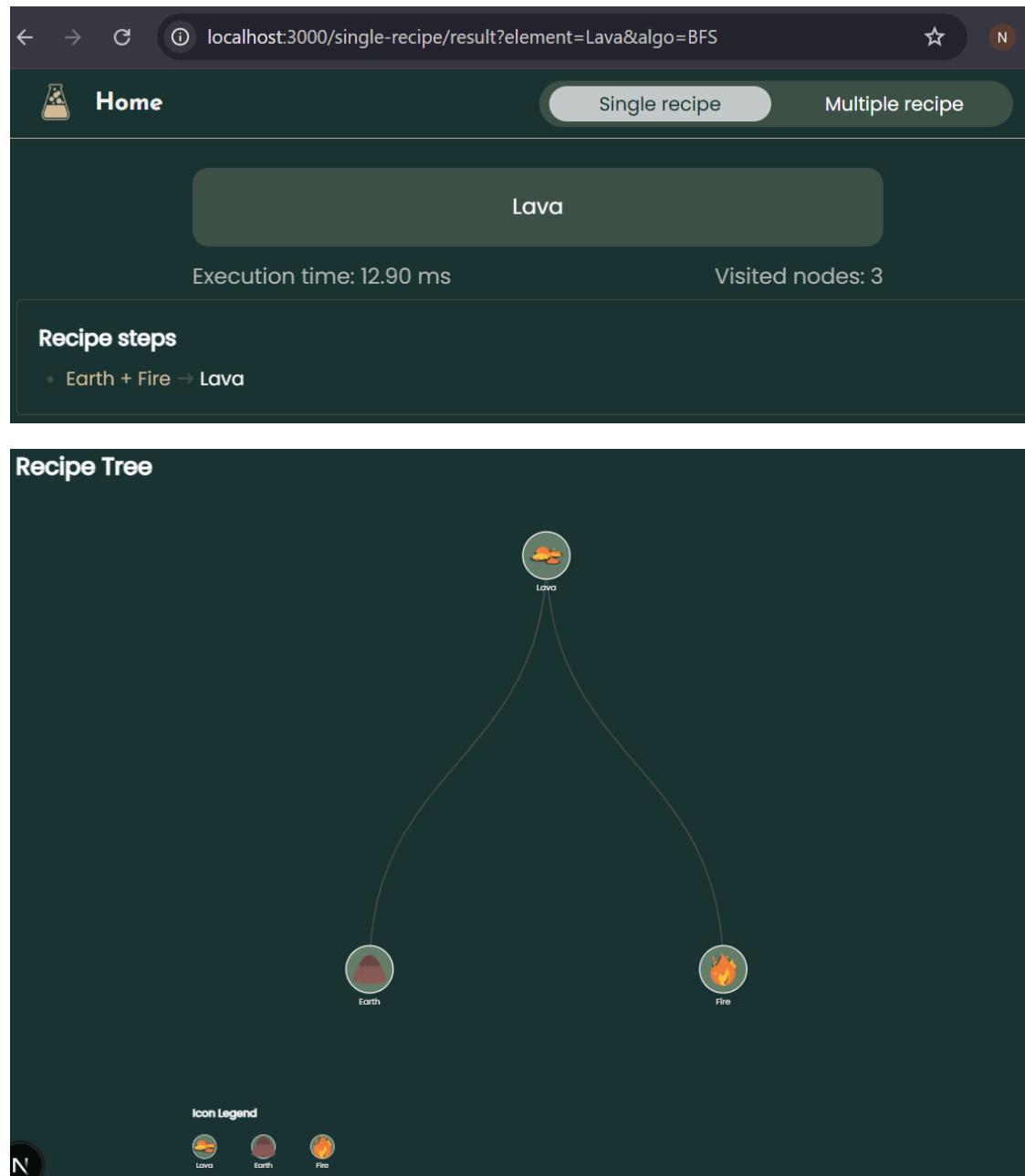


#### 4.3.3.4. Multiple Recipe - DFS



#### 4.3.4. Test Case 4

##### 4.3.4.1. Single Recipe - BFS



#### 4.3.4.2. Single Recipe - DFS

localhost:3000/single-recipe/result?element=Lava&algo=DFS

Home Single recipe Multiple recipe

Lava

Execution time: 16.50 ms Visited nodes: 3

**Recipe steps**

- Earth + Fire → Lava

#### 4.3.4.3. Multiple Recipe - BFS

localhost:3000/multiple-recipe/result?element=Lava&algo=bfs&max=5

Home Single recipe Multiple recipe

Found 1 recipes for Lava

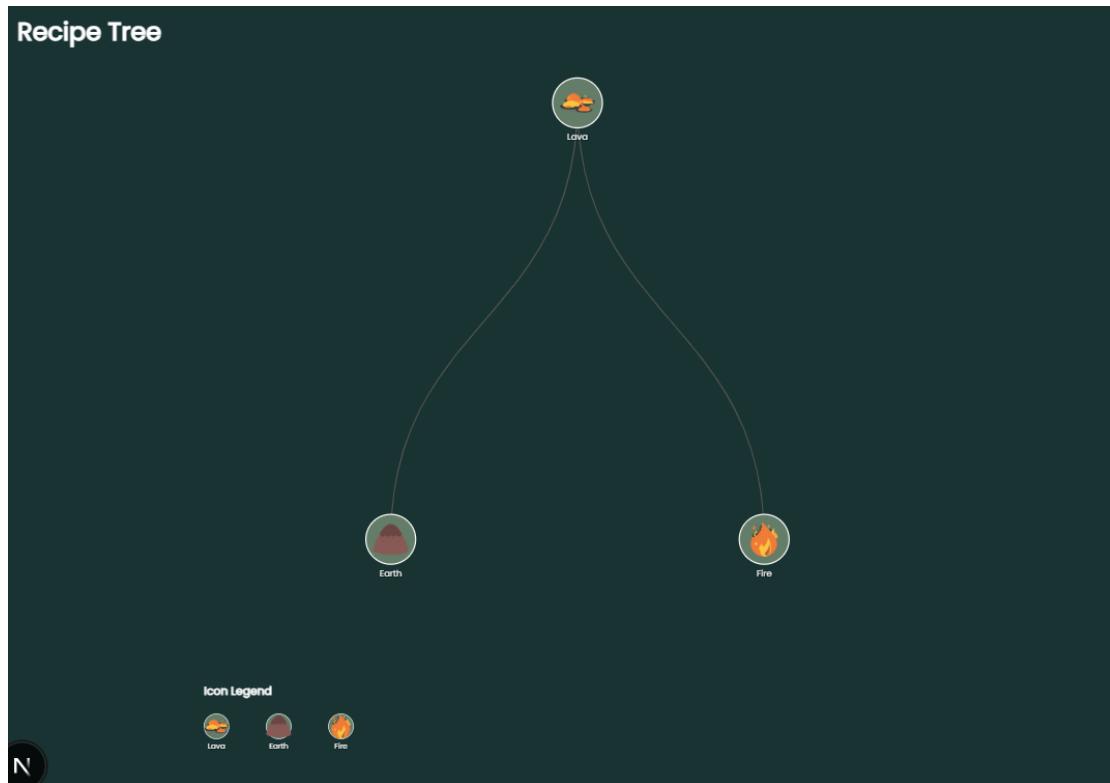
Lava

Execution time: 10 ms Visited nodes: 3

**Path #1**

**Recipe steps**

- Earth + Fire → Lava



#### 4.3.4.4. Multiple Recipe - DFS

localhost:3000/multiple-recipe/result?element=Lava&algo=dfs&max=5

Home Single recipe Multiple recipe

Found 1 recipes for Lava

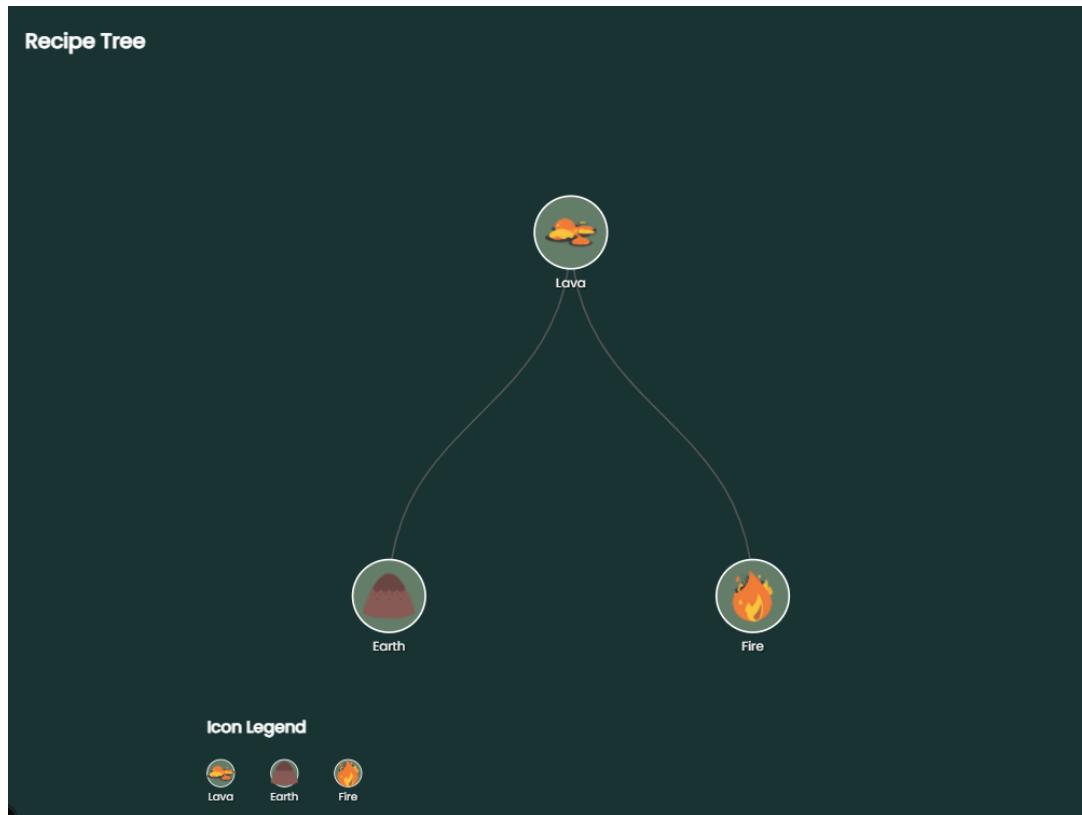
Lava

Execution time: 12 ms Visited nodes: 3

Path #1

**Recipe steps**

- Earth + Fire → Lava



#### 4.5. Analisis hasil pengujian

Berdasarkan hasil pengujian program, algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) menunjukkan performa yang relatif serupa dari segi waktu eksekusi pada kasus-kasus umum. Kedua algoritma ini dapat menyelesaikan pencarian dengan efisiensi yang sebanding pada banyak skenario. Namun, terdapat perbedaan signifikan pada kasus pencarian multiple recipe, di mana DFS cenderung membutuhkan waktu lebih lama untuk menyelesaikan pencarian. Hal ini disebabkan oleh sifat DFS yang menelusuri cabang secara mendalam sebelum beralih ke cabang lain, sehingga ketika dihadapkan dengan banyak kemungkinan resep yang perlu ditelusuri, DFS membutuhkan lebih banyak iterasi dan waktu pemrosesan.

Menariknya, pencarian multi recipe dapat menunjukkan performa yang lebih cepat dibandingkan single recipe karena adanya implementasi konsep multithreading. Dengan pendekatan multithreading, program dapat melakukan pencarian secara paralel untuk beberapa resep sekaligus, sehingga memanfaatkan kapasitas pemrosesan komputer secara optimal. Proses pencarian yang sebelumnya harus dilakukan secara sekuensial dapat dibagi menjadi beberapa thread yang berjalan bersamaan, menghasilkan peningkatan kecepatan yang signifikan terutama pada sistem dengan multiple core processor.

Dari segi penggunaan memori, BFS teridentifikasi lebih boros dalam penggunaan data karena karakteristiknya yang mengharuskan penyimpanan seluruh node pada level yang sama dalam antrian (queue). Algoritma BFS perlu menyimpan semua node tetangga sebelum menelusuri lebih dalam, yang berpotensi mengakibatkan penggunaan memori yang besar terutama pada graf dengan faktor percabangan tinggi. Sebaliknya, DFS cenderung lebih hemat memori karena hanya perlu menyimpan path dari root ke node saat ini, meskipun trade-off-nya adalah waktu yang lebih lama dalam beberapa kasus pencarian.

## **BAB V**

### **KESIMPULAN, SARAN, DAN REFLEKSI**

#### **5.1. Kesimpulan**

Pada tugas besar ini, kami berhasil membuat aplikasi berbasis web yang memiliki fitur utama pencarian resep suatu elemen pada Little Alchemy 2. Dalam implementasinya, kami menggunakan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS). Bahasa pemrograman yang digunakan adalah Go untuk pengembangan backend dan Typescript dengan framework next.js dan tailwindcss untuk pengembangan frontend.

Implementasi kedua algoritma penjelajahan graf tersebut memungkinkan pengguna untuk menemukan berbagai kombinasi elemen yang diperlukan untuk membuat elemen target dengan efisien. BFS menawarkan solusi terpendek dalam hal jumlah langkah kombinasi, sementara DFS menyediakan alternatif pencarian yang dapat mengeksplorasi jalur yang lebih dalam dan lebih hemat memori. Melalui pengujian yang kami lakukan, algoritma ... cenderung lebih optimal karena ...

Selain itu, kami juga mempelajari konsep multithreading untuk mengeksekusi suatu proses dalam beberapa thread secara bersamaan atau paralel. Konsep ini membantu untuk mempercepat waktu pencarian dalam graf. Beberapa aspek dalam algoritma pencarian graf dalam kasus ini dapat dilakukan secara paralel karena beberapa bagian proses bersifat independen. Ini dapat memangkas waktu pencarian dari kasus single thread di mana satu bagian proses harus menunggu eksekusi bagian proses sebelumnya meskipun tidak ada keterkaitan atau ketergantungan dengan bagian proses sebelumnya.

#### **5.2. Saran**

Berikut beberapa saran oleh kelompok FullRustAlchemist yang berguna untuk memaksimalkan potensi pengembangan selanjutnya:

- a. Mengoptimalkan struktur pembagian tugas serta meningkatkan kerjasama tim untuk mengoptimalkan hasil program.
- b. Meningkatkan komunikasi antar anggota kelompok untuk mencegah adanya kesalahpahaman dalam memaknai cara kerja maupun algoritma program

### 5.3. Refleksi

Dalam tugas ini, kami belajar mengenai algoritma pencarian BFS dan DFS dengan graf yang memiliki struktur unik, yaitu graf yang merepresentasikan kombinasi elemen-elemen pada permainan *Little Alchemy 2*. Setiap simpul dalam graf merepresentasikan satu elemen, dan setiap sisi menunjukkan hasil dari penggabungan dua elemen yang membentuk elemen baru. Melalui implementasi BFS dan DFS, kami dapat mencari urutan kombinasi yang efisien untuk membentuk elemen target dari elemen dasar. Tugas ini meningkatkan pemahaman kami mengenai algoritma BFS dan DFS dan membuat kami menyadari bahwa algoritma ini memiliki lingkup implementasi yang sangat luas, bahkan banyak digunakan dalam kehidupan sehari-hari.

Tugas ini menjadi wadah eksplorasi program multithreading. Pengerajan tugas ini menggunakan fitur multithreading go berupa goroutine dan menggunakan alat sinkronisasi seperti Channel, sync.Mutex, dan sync.WaitGroup. Tugas ini juga memberikan contoh untuk perbandingan konsep konkurensi dengan paralelisme. Sebelumnya, sempat diimplementasikan algoritma DFS dimana thread dibangkitkan secara rekursif tetapi setiap thread sangat bergantung kepada satu state yang aksesnya terbagi secara global. Untuk mencegah race condition diperlukan sebuah mekanisme mutual exclusion (sync.Mutex). Karena routine sangat sering bergantung pada state ini, routine yang sebenarnya melakukan eksekusi secara efektif hanyalah satu yaitu routine yang sedang memiliki akses eksklusif pada state global. Ini sama saja dengan pemrosesan single threading hanya saja dengan ilusi paralelisme.

Selain itu, implementasi algoritma penelusuran graf dengan multithreading masih perlu ditinjau kembali. Contohnya, pada algoritma BFS, perlu ditinjau kembali distribusi task ke thread secara lebih merata dan optimal. BFS juga membutuhkan waktu sinkronisasi yang dapat dioptimasi pada tiap akhir iterasi. Kemudian pada algoritma DFS, kebanyakan thread hanya berada pada state menunggu. Thread akan aktif untuk ketika menerima sinyal dan kemudian menunggu kembali hingga routine yang dikontrol memberikan hasil. Ditambah lagi, routine yang dibangkitkan untuk elemen kedua untuk kebanyakan waktu akan tidur, harus menunggu elemen pertama selesai. Kesimpulannya, perlu dipikirkan kembali desain algoritma yang memiliki utilisasi thread yang maksimal dengan cara distribusi task yang lebih baik.

## LAMPIRAN

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.	✓	

- Tautan reposik <https://little-alchemy2-recipe-finder.vercel.app/tory> GitHub :  
[https://github.com/naylzhra/Tubes2\\_FullRustAlchemist](https://github.com/naylzhra/Tubes2_FullRustAlchemist)
- Tautan youtube :  
<https://youtu.be/f1-PvPU5c1U>
- Tautan deploy:  
<https://little-alchemy2-recipe-finder.vercel.app/>

## DAFTAR PUSTAKA

Munir, R. *BFS - DFS - Bagian I*, 2025. [Daring]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf) [Diakses: 11-Mei-2025].

Munir, R. *BFS - DFS - Bagian I*, 2025. [Daring]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf) [Diakses: 11-Mei-2025].

Parallel breadth first search. Tersedia: [https://en.wikipedia.org/wiki/Parallel\\_breadth-first\\_search](https://en.wikipedia.org/wiki/Parallel_breadth-first_search) [Diakses: 13-Mei-2025].

Go User Manual. [Daring]. Tersedia: <https://go.dev/doc/>  
Next.js Documentation. <https://nextjs.org/docs>