

LAPORAN TUGAS BESAR 2
IF2211 STRATEGI ALGORITMA



Disusun oleh kelompok 51 LetUsCook

Muhammad Ghifary Komara Putra 13523066

Sabilul Huda 13523072

Diyah Susan Nugrahani 13523080

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

BAB I	3
DESKRIPSI TUGAS	3
BAB II	5
LANDASAN TEORI	5
2.1 Penjelajahan Graf	5
2.2 BFS	5
2.3 DFS	6
2.4 Website	8
2.5. Web Scraping	9
BAB III	10
ANALISIS PEMECAHAN MASALAH	10
3.1. Langkah-langkah Pemecahan Masalah	10
3.2 Proses Pemetaan Masalah	12
3.3 Fitur Fungsional dan Arsitektur Website	14
3.4 Ilustrasi Kasus	15
BAB IV	20
IMPLEMENTASI DAN PENGUJIAN	20
4.1 Struktur Data	20
4.1.1. Struktur Data Umum	20
4.1.2. Struktur Data untuk bfs_multiple.go:	20
4.1.3. Struktur Data untuk bfs_single.go	21
4.1.4. Struktur Data untuk dfs_multiple.go	21
4.1.4. Struktur Data untuk bidirectional_single.go	21
4.2 Fungsi dan Prosedur	22
4.2.1 BFS (Single)	22
4.2.2 BFS (Multiple)	25
4.2.3 DFS (Fungsi Pembantu)	29
4.2.4 DFS (Single)	30
4.2.5 DFS (Multiple)	31
4.2.6 Scraping	34
4.2.7 Fungsi utama (Main)	39
4.2.8. Bidirectional (Single)	43
4.2.9 Bidirectional (Multiple)	46
4.2.10 Bidirectional (Pembantu)	49
4.3 Tata Cara Penggunaan	50
4.3.1 Penggunaan di Lokal	50
4.3.2 Penggunaan Dengan Docker	50
4.3.3 Penggunaan Melalui Website	51
4.4 Hasil Pengujian	51
4.5 Analisis Hasil Pengujian	59

BAB V	62
KESIMPULAN	62
5.1 Kesimpulan	62
5.2 Saran	62
5.3 Refleksi	62
LAMPIRAN	63
DAFTAR PUSTAKA	64

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.



Gambar 1. Elemen dasar pada Little Alchemy 2

2. Elemen turunan

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

3. *Combine Mechanism*

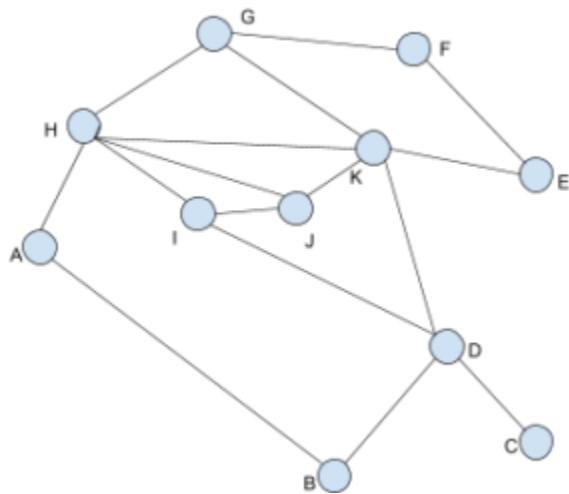
Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

BAB II

LANDASAN TEORI

2.1 Penjelajahan Graf

Graf digunakan untuk merepresentasikan objek-objek dan hubungan antara objek tersebut. Representasi visual dari graf adalah dengan menyatakan objek sebagai simpul atau *node* dan hubungan antara objek sebagai sisi atau *edge*. Simpul-simpul pada graf tidak boleh merupakan himpunan kosong, sedangkan sisi-sisi pada graf boleh merupakan himpunan kosong.



Gambar 2. Contoh Representasi Graf (sumber :
<https://www.geeksforgeeks.org/welsh-powell-graph-colouring-algorithm/>)

Penjelajahan graf merupakan algoritma pencarian solusi persoalan yang direpresentasikan dengan graf. Dalam penjelajahan graf, suatu persoalan akan direpresentasikan sebagai graf yang terhubung. Solusi yang dicari dalam penjelajahan graf didapatkan dengan mengunjungi simpul-simpul di dalam graf dengan beberapa pendekatan algoritma.

Algoritma pencarian solusi berbasis graf dapat dilakukan dengan dua pendekatan, yaitu dengan menggunakan graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan. Untuk melakukan pencarian di graf statis, graf akan direpresentasikan sebagai struktur data. Graf dinamis adalah graf yang terbentuk saat proses pencarian solusi. Di graf dinamis, pada awalnya graf tidak tersedia dan baru dibangun selama proses pencarian dilakukan. Dalam mencari solusi persoalan di graf, terdapat dua algoritma yang dapat digunakan yaitu *Breadth First Search (BFS)* dan *Depth First Search (DFS)*.

2.2 BFS

Pencarian menyebar atau biasa disebut *breadth-first search* (BFS) merupakan suatu algoritma traversal graf, tepatnya algoritma pencarian solusi berbasis graf yang bersifat tanpa informasi (*uninformed/blind search*). Pencarian dapat dilakukan pada graf statis (graf sudah

terbentuk sebelum proses pencarian dilakukan) maupun pada graf dinamis (graf terbentuk saat proses pencarian dilakukan). Pada suatu graf statis, algoritma BFS yang diawali pada simpul v memiliki algoritma sebagai berikut:

1. Kunjungi simpul v
2. Bangun sebuah queue
3. Push node-node yang bertetangga dengan v ke dalam queue
4. Traversal setiap elemen queue, lakukan langkah 3 pada tiap elemen queue sampai queue habis.

Hal serupa terjadi pada pencarian pada graf dinamis, dengan penambahan langkah berupa pembentukan pohon (atau, secara umum, graf) secara dinamis. Beberapa istilah yang relevan dalam pencarian ini adalah sebagai berikut:

- Pohon ruang status (*state space tree*)
- Simpul sebagai *problem state* (apakah layak membentuk solusi atau tidak)
 - Simpul akar sebagai *initial state*
 - Simpul daun sebagai *goal state*
- Cabang sebagai operator/langkah dalam persoalan
- Ruang status (*state space*) sebagai himpunan semua simpul
- Ruang solusi sebagai himpunan status solusi
- Solusi sebagai rute menuju status solusi

Pembangkitan status pada pencarian graf dinamis bekerja dengan mengaplikasikan operator atau langkah legal kepada status (simpul) pada rute tertentu. Jalur dari simpul akar ke simpul daun merupakan rangkaian operator yang mengarah pada solusi persoalan.

2.3 DFS

Pencarian mendalam atau biasa disebut depth-first search (DFS) merupakan suatu algoritma traversal graf, tepatnya algoritma pencarian solusi berbasis graf yang bersifat tanpa informasi (*uninformed/blind search*). Pencarian dapat dilakukan pada graf statis (graf sudah terbentuk sebelum proses pencarian dilakukan) maupun pada graf dinamis (graf terbentuk saat proses pencarian dilakukan). Pada suatu graf statis, algoritma DFS yang diawali pada simpul v memiliki algoritma sebagai berikut:

1. Kunjungi simpul v
2. Kunjungi simpul w , yaitu simpul yang bertetangga dengan v
3. Ulangi DFS mulai dari simpul w
4. Ketika pencarian mencapai suatu simpul u sedemikian hingga semua simpul yang bertetangga dengan u telah dikunjungi, pencarian akan melakukan *backtrack* (runut balik) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul yang belum dikunjungi
5. Pencarian berakhir bila simpul yang dicari telah ditemukan atau tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi

Hal serupa terjadi pada pencarian pada graf dinamis, dengan penambahan langkah berupa pembentukan pohon (atau, secara umum, graf) secara dinamis. Beberapa istilah yang relevan dalam pencarian ini adalah sebagai berikut:

- Pohon ruang status (*state space tree*)
- Simpul sebagai *problem state* (apakah layak membentuk solusi atau tidak)
 - Simpul akar sebagai *initial state*
 - Simpul daun sebagai *goal state*
- Cabang sebagai operator/langkah dalam persoalan
- Ruang status (*state space*) sebagai himpunan semua simpul
- Ruang solusi sebagai himpunan status solusi
- Solusi sebagai rute menuju status solusi

Pembangkitan status pada pencarian graf dinamis bekerja dengan mengaplikasikan operator atau langkah legal kepada status (simpul) pada rute tertentu. Jalur dari simpul akar ke simpul daun merupakan rangkaian operator yang mengarah pada solusi persoalan.

2.3. Bidirectional Searching

Pencarian dua arah atau biasa disebut *Bidirectional Searching* merupakan suatu algoritma traversal graf, tepatnya algoritma pencarian solusi berbasis graf yang bersifat tanpa informasi (*uninformed/blind search*). Pencarian dapat dilakukan pada graf statis (graf sudah terbentuk sebelum proses pencarian dilakukan) maupun pada graf dinamis (graf terbentuk saat proses pencarian dilakukan). Pada algoritma ini, dibutuhkan informasi node awal dan node akhir. Pada suatu graf statis, algoritma *Bidirectional* yang diawali pada simpul v memiliki algoritma sebagai berikut:

1. Bangun dua *queue*, *forwardQueue* dan *backwardQueue*.
2. Isi kedua *queue* tersebut dengan node-node yang diketahui
3. Selama kedua node tersebut tidak kosong, lakukan proses pada node tersebut dan bangkitkan node tetangga node terkait dan tambahkan ke *queue* yang terkait.

Hal serupa terjadi pada pencarian pada graf dinamis, dengan penambahan langkah berupa pembentukan pohon (atau, secara umum, graf) secara dinamis. Beberapa istilah yang relevan dalam pencarian ini adalah sebagai berikut:

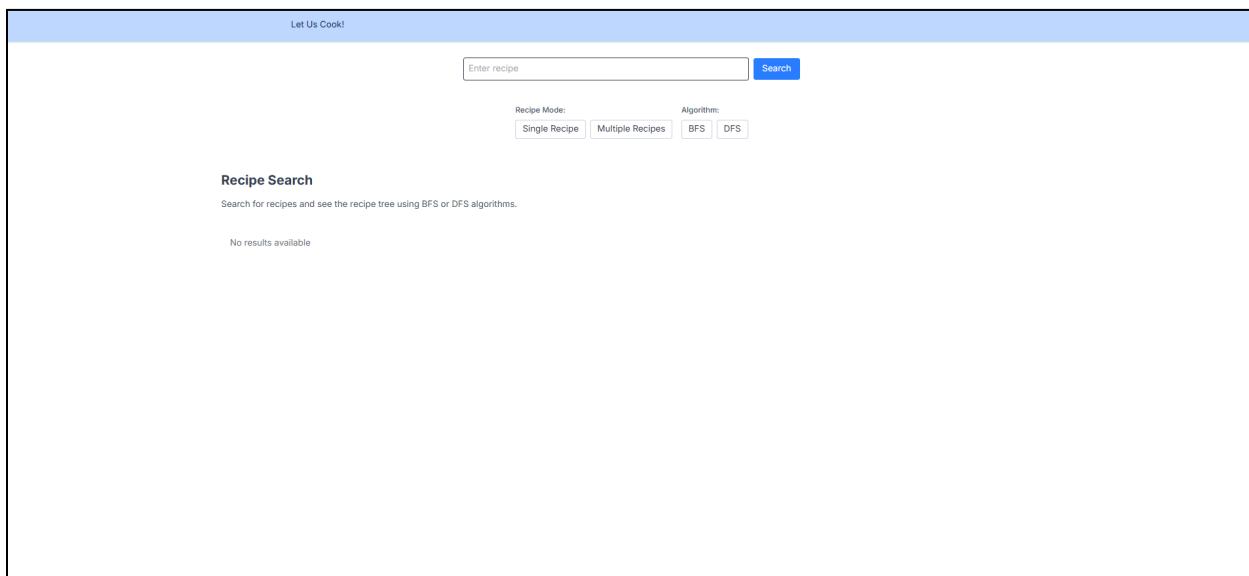
- Pohon ruang status (*state space tree*)
- Simpul sebagai *problem state* (apakah layak membentuk solusi atau tidak)
 - Simpul akar sebagai *initial state*
 - Simpul daun sebagai *goal state*
- Cabang sebagai operator/langkah dalam persoalan
- Ruang status (*state space*) sebagai himpunan semua simpul
- Ruang solusi sebagai himpunan status solusi
- Solusi sebagai rute menuju status solusi
- *ForwardQueue* sebagai antrian pemrosesan maju
- *backwardQueue* sebagai antrian pemrosesan maju

Pembangkitan status pada pencarian graf dinamis bekerja dengan mengaplikasikan operator atau langkah legal kepada status (simpul) pada rute tertentu. Jalur dari simpul akar ke simpul daun merupakan rangkaian operator yang mengarah pada solusi persoalan

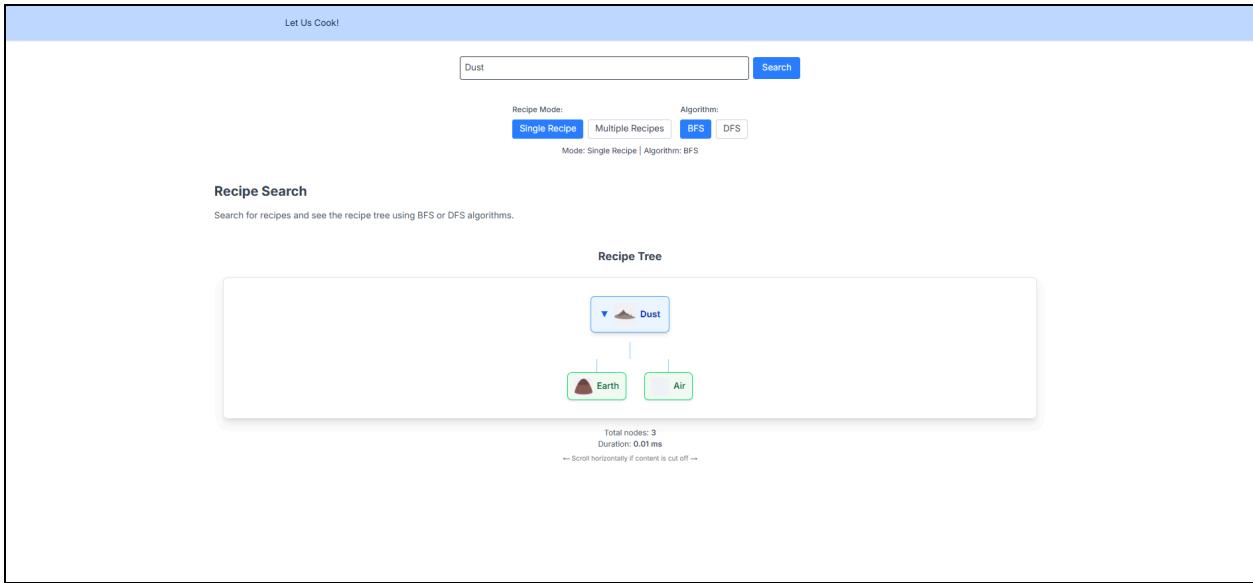
2.4 Website

Website adalah sebuah *platform* yang dapat digunakan untuk menampilkan informasi kepada pengguna. Arsitektur *website* dibagi menjadi dua bagian utama yaitu *front end* dan *back end*. Kedua bagian tersebut saling berhubungan satu sama lain untuk menghasilkan fungsionalitas *website*. *Front end* adalah bagian website yang menampilkan informasi dan dapat dilihat secara langsung oleh pengguna. Pada bagian ini pengguna dapat berinteraksi dengan cara memberikan *input query* dan kemudian akan mendapatkan *output* yang sesuai. Tujuan utama dari *front end* adalah untuk memberikan *user experience (UX)* yang baik dengan membuat tampilan *website* menjadi mudah diakses, responsive, dan dapat mudah dimengerti. *Back end* adalah server yang bertugas untuk mengolah data dan *request* yang diberikan oleh pengguna. Pengguna tidak berinteraksi secara langsung dengan *back end*, melainkan input yang diberikan pengguna akan dikirim dari *front end* ke *back end* kemudian kembali lagi ke *front end* untuk menampilkan informasi yang diminta.

Pada tugas ini, *website* dibuat dengan *framework* react dengan bahasa TypeScript. Tampilan *website* dibuat sederhana dengan satu halaman saja. Semua fitur yang ada pada *website* langsung ditampilkan dalam halaman tersebut. Untuk bagian algoritma pencarian solusi digunakan bahasa go. Dalam menghubungkan antara *front end* dan *back end* digunakan *framework* gin. *Repository* untuk *website* dipisah menjadi *front end* dan *back end* dengan alasan karena akan dibuat *docker*.



Gambar 3. Tampilan Awal *Website*



Gambar 4. Tampilan Website Setelah *Query*

2.5. Web Scraping

Web scraping adalah metode yang digunakan untuk mengekstrak data dari suatu website secara otomatis. Tujuan utama dari penggunaan *web scraping* adalah mengumpulkan informasi yang diinginkan lalu mengolah atau memvisualisasikannya sesuai kebutuhan. Berikut adalah beberapa hal penting terkait *web scraping*:

1. Struktur Dokumen HTML: *Web scraping* sangat bergantung pada pemahaman struktur HTML, karena data yang ditampilkan di halaman web biasanya tersimpan dalam elemen-elemen HTML seperti `<div>`, `<table>`, ``, `<a>`, dan sebagainya. Struktur HTML bersifat hierarkis dan disusun dalam bentuk Document Object Model (DOM). Scraper akan mencari elemen berdasarkan tag, class, ID, atau atribut lainnya.
2. HTTP dan Protokol Web: Untuk mengakses halaman web, *scraper* perlu memahami protokol komunikasi HTTP/HTTPS. Permintaan dikirim melalui HTTP *request* (GET, POST, dll), dan jawaban dari server dikembalikan dalam bentuk HTTP *response* yang berisi data HTML.
3. Parsing HTML: Parsing adalah proses menguraikan dokumen HTML agar bisa dimanipulasi. Dalam scraping, parser digunakan untuk mengekstraksi data dari elemen tertentu.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-langkah Pemecahan Masalah

A. Langkah-langkah komunikasi *Back-end* dengan *Front-end*

Website yang dibuat untuk tugas ini terdiri dari *front end* dan *back end*. Pengguna akan berinteraksi melalui *front end* dan dapat menginputkan *query* resep yang ingin dicari beserta dengan pilihan mode dan algoritma. Setelah pengguna menekan tombol *search*, data dari *front end* akan dikirim ke *backend*. Untuk menghubungkan antara keduanya digunakan framework gin. Data yang telah dikirim ke *back end* kemudian akan diproses dengan algoritma sesuai yang diinginkan oleh pengguna. Setelah algoritma selesai, akan dikembalikan data berupa pohon solusi resep serta jumlah node yang dikunjungi. Data tersebut dikirim dalam bentuk json dan nantinya akan ditampilkan oleh *front end*. Untuk tampilan icon element, dilakukan scrapping nama elemen dan link gambar yang diubah ke bentuk json supaya mempermudah ketika akan ditampilkan di *front end*.

B. Langkah-langkah algoritma BFS

Elaborasi lebih lanjut mengenai struktur data yang digunakan tertera pada bab 3.2 dan 4.1. Berikut merupakan langkah-langkah pemecahan masalah dengan algoritma BFS.

Single recipe

1. Kunjungi simpul akar v , yaitu elemen yang akan dicari
2. Bangkitkan seluruh simpul $v.Children$ dengan batasan: tier children harus kurang dari tier orangtuanya
3. Diantara anak-anak tersebut, pilih yang paling tinggi skornya, perhitungan score berdasarkan seberapa banyak elemen dari pair terkait yang merupakan elemen dasar.
4. Tambahkan node-node hasil resep terbaik ke pohon dan queue, jika belum dikunjungi atau ditemukan jalur lebih pendek
5. Simpan informasi resep terbaik (`bestRecipes`) untuk setiap node
6. Setelah BFS selesai, bangun kembali pohon resep optimal secara rekursif (DFS) dari target ke elemen-elemen dasar

Multiple recipe

1. Buat root node berdasarkan nama dari indeks nama elemen inputan user. Buat antrian BFS dan tanda kunjungan (`visited`) untuk menyimpan node yang telah dikunjungi.
2. Masukkan node root dengan depth 0 ke dalam antrian BFS.
3. Selama antrian tidak kosong, ambil node dari antrian, lewati node yang sudah pernah dikunjungi, dan jika indeks node ≤ 4 , anggap node dasar dan lanjutkan ke node berikutnya.

4. Ambil semua resep dari `MapperIdxToRecipes` berdasarkan indeks saat ini, untuk setiap resep, periksa apakah tier dari bahan-bahan tidak lebih tinggi dari node saat ini, jika iya lewati.
5. Jika batas jumlah resep belum terlampaui, buat dua node anak berdasarkan dua bahan resep. Tambahkan kedua node anak sebagai `Pair_recipe` ke dalam node saat ini.
6. Masukkan kedua node anak ke dalam antrian untuk diproses selanjutnya. Lakukan pemangkasan terhadap node-node yang bukan terminal dengan memanggil `PruneNonTerminalParallel` sebanyak satu kali.
7. Jika root belum lengkap atau tidak merupakan node dasar, cari jalur terpendek dengan `FindShortestPath`. Jika sudah lengkap, rekursi ke setiap anak untuk melengkapi node secara rekursif.
8. Kembalikan root pohon hasil BFS dan total jumlah node yang berhasil dibentuk.

C. Langkah-langkah algoritma DFS

Elaborasi lebih lanjut mengenai struktur data yang digunakan tertera pada bab 3.2 dan 4.1. Berikut merupakan langkah-langkah pemecahan masalah dengan algoritma DFS.

Single recipe

1. Kunjungi simpul akar v , yaitu elemen yang akan dicari
2. Kunjungi simpul w , yaitu simpul anak pertama dari v yang memenuhi semua batasan (tidak mengandung elemen dengan *tier* lebih tinggi dari v dan tidak mengandung elemen *time*), yang merupakan pasangan elemen yang membentuk v . Simpul w akan memiliki dua simpul anak, yaitu $x1$ dan $x2$
3. Ulangi DFS mulai dari simpul $x1$
4. Ulangi DFS mulai dari simpul $x2$
5. Ketika pencarian mencapai suatu simpul u sedemikian hingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian akan melakukan runut balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul yang belum dikunjungi
6. Pencarian berakhir ketika simpul merupakan elemen dasar (*water, fire, earth, air*)

Multiple recipe

1. Kunjungi simpul akar v , yaitu elemen yang akan dicari
2. Kunjungi simpul w , yaitu simpul anak dari v , yang merupakan pasangan elemen yang membentuk v . Simpul w akan memiliki dua simpul anak, yaitu $x1$ dan $x2$
3. Ulangi DFS mulai dari simpul $x1$. Langkah ini berjalan bersamaan dengan (4)
4. Ulangi DFS mulai dari simpul $x2$. Langkah ini berjalan bersamaan dengan (3)
5. Hitung banyaknya kombinasi *recipe* dari w , yaitu hasil perkalian dari banyaknya kombinasi *recipe* untuk membuat $x1$ dengan banyaknya kombinasi *recipe* untuk membuat $x2$

6. Untuk menghitung banyaknya kombinasi *recipe* dari suatu elemen, jumlahkan banyaknya kombinasi *recipe* dari w_1, \dots, w_n , yaitu seluruh pasangan kombinasi elemen yang telah dibangkitkan semasa pencarian.
7. Simpan banyaknya *recipe* yang diperlukan untuk membuat v
8. Ketika banyak *recipe* yang diperlukan untuk membuat v sudah mencapai atau melewati batas yang diinginkan, seluruh pencarian pada simpul y yang masih berlangsung hanya akan mencari satu *recipe* untuk membuat y
9. Ketika pencarian mencapai suatu simpul u sedemikian hingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian akan melakukan runut balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul yang belum dikunjungi
10. Pencarian berakhir ketika simpul merupakan elemen dasar (*water, fire, earth, air*)

D. Langkah-langkah algoritma *Bidirectional Searching*

Elaborasi lebih lanjut mengenai struktur data yang digunakan tertera pada bab 3.2 dan 4.1. Berikut merupakan langkah-langkah pemecahan masalah dengan algoritma DFS.

Single recipe

1. Cek apakah input nama elemen valid.
2. Cek apakah *tier* elemen lebih dari satu. Jika tidak fungsi akan mengembalikan *root null* dan *contNode 0*
3. Inisiasi *ForwardQueue*, *BackwardQueue*, *ForwardVisited*, *BackwardVisited*, dan variabel lain yang diperlukan
4. Memasukkan beberapa node yang sudah diketahui pada *ForwardQueue*, *BackwardQueue*, *ForwardVisited*, *BackwardVisited*
5. Selama (*looping*) *ForwardQueue*, *BackwardQueue* tidak kosong lakukan proses berikut:
 - a. Jika node tersebut sudah dikunjungi, lewati proses
 - b. Proses elemen pertama dari tiap *ForwardQueue*, *BackwardQueue*, dengan menyimpan *path* untuk mencapai node tersebut. Jika node tersebut pernah dikunjungi pada proses lawannya (jika saat ini proses yang berlangsung adalah *forward*, maka yang dimaksud adalah proses *backward*, dan sebaliknya) maka satukan *path* dari proses *forward* dan *backward* untuk mendapat *path* akhir.
 - c. Keluar dari *looping* tersebut.

3.2 Proses Pemetaan Masalah

Masalah pemanfaatan algoritma BFS dan DFS dalam pencarian *recipe* pada permainan Little Alchemy 2 dapat dipetakan menjadi masalah pencarian tanpa informasi (*uninformed/blind*

search) pada graf dinamis. Perhatikan bahwa informasi yang diperoleh pada proses *scrapping* terbatas pada himpunan elemen yang terdapat pada permainan serta himpunan informasi yang berkaitan dengan elemen tersebut, yaitu *tier* dari elemen yang bersangkutan dan himpunan kombinasi *recipe* yang membentuknya. Representasi pohon dinamis pada masalah ini dapat dipetakan sebagai berikut:

- Simpul

Terdapat dua jenis simpul dalam permasalahan ini, yaitu simpul elemen (RecipeTree) dan simpul kombinasi resep (Pair_recipe). RecipeTree merepresentasikan suatu elemen dan dapat memiliki anak berupa Pair_recipe. Suatu Pair_recipe merepresentasikan pasangan elemen yang dapat membentuk RecipeTree yang menjadi induknya. Setiap Pair_recipe akan memiliki dua anak, masing-masing merupakan RecipeTree yang merepresentasikan elemen yang terdapat pada Pair_recipe tersebut.

- Pohon ruang status

Pohon ruang status pada permasalahan ini adalah pohon yang terbentuk dari simpul-simpul RecipeTree dan Pair_recipe. Anak dari suatu RecipeTree adalah Pair_recipe yang akan membentuknya dan anak dari suatu Pair_recipe adalah RecipeTree dari elemen yang tertera pada Pair_recipe tersebut

- Simpul akar

Simpul akar pada permasalahan ini adalah elemen yang akan dicari *recipe* yang membentuknya (berupa RecipeTree)

- Simpul daun

Simpul daun atau status solusi pada permasalahan ini adalah empat elemen dasar pada setiap permainan, yaitu *water*, *fire*, *earth*, dan *air*.

- Cabang

Cabang atau operator/langkah dalam persoalan ini adalah pengecekan Pair_recipe apa sajakah yang dapat menjadi anak dari suatu RecipeTree. Informasi ini diperoleh berdasarkan data hasil scraping. Terdapat pula syarat tambahan dalam percabangan ini, yaitu *tier* dari elemen pada Pair_recipe harus lebih rendah dari *tier* dari elemen pada RecipeTree yang akan menjadi induknya dan elemen *time* akan diabaikan (tidak dianggap sebagai langkah valid) baik pada RecipeTree maupun pada Pair_recipe

- Ruang status

Himpunan semua simpul, baik RecipeTree maupun Pair_recipe hasil pembentukan graf dinamis

- Ruang solusi

Himpunan semua status solusi (berupa RecipeTree dari empat elemen dasar)

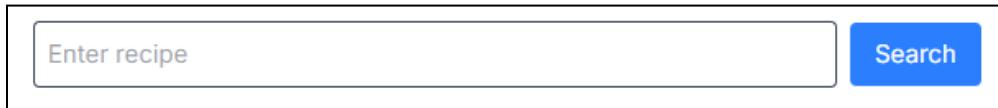
- Solusi

Rute dari simpul akar (elemen yang hendak dicari resepnya) ke seluruh simpul daun pada pohon yang telah terbentuk

3.3 Fitur Fungsional dan Arsitektur Website

3.3.1 Search Bar

Fitur *search bar* digunakan untuk mencari resep. Pengguna dapat memasukkan resep yang ingin dicari di kolom pencarian lalu memilih opsi pencarian yang tersedia kemudian menekan tombol *search*.



Gambar 5. *Search Bar*

3.3.2 Opsi Pencarian

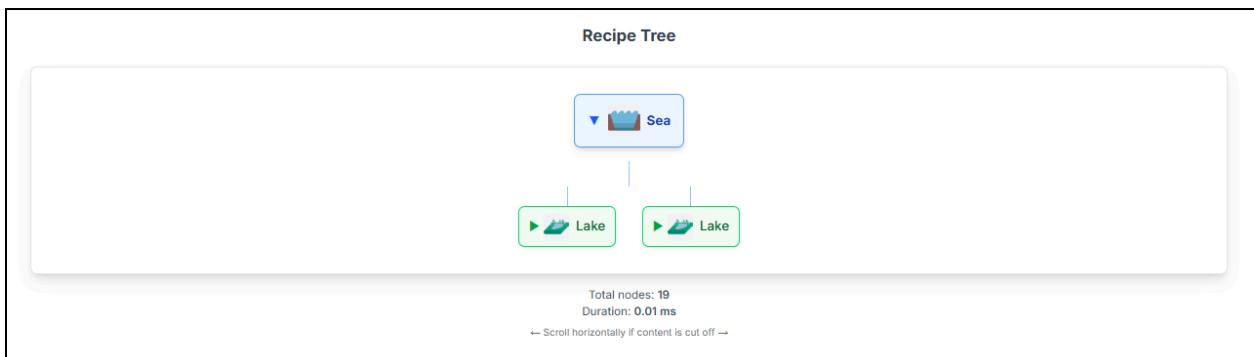
Terdapat beberapa opsi pencarian dengan dua mode yaitu *single recipe* dan *multiple recipes*. Ketika memilih mode *multiple recipes*, akan muncul kolom input jumlah resep yang ingin dicari. Selain itu terdapat tiga opsi algoritma yang dapat dipilih, yaitu algoritma BFS dan DFS.

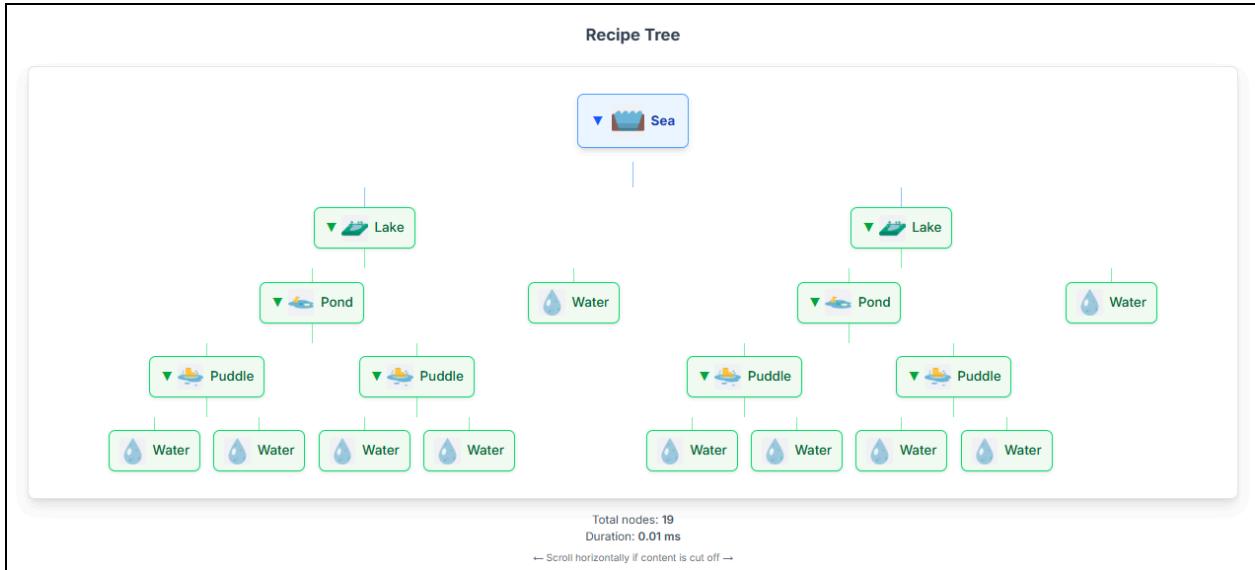


Gambar 6. Opsi Pencarian

3.3.3 Tampilan Tree

Tampilan pohon hasil pencarian dapat *di-expand* sehingga pengguna dapat melihat keseluruhan resep hingga elemen dasar. *Container* juga dapat digeser secara horizontal apabila ada resep yang terpotong.

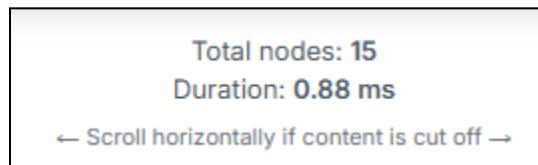




Gambar 7. Hasil Pohon Solusi

3.3.4 Informasi Tambahan

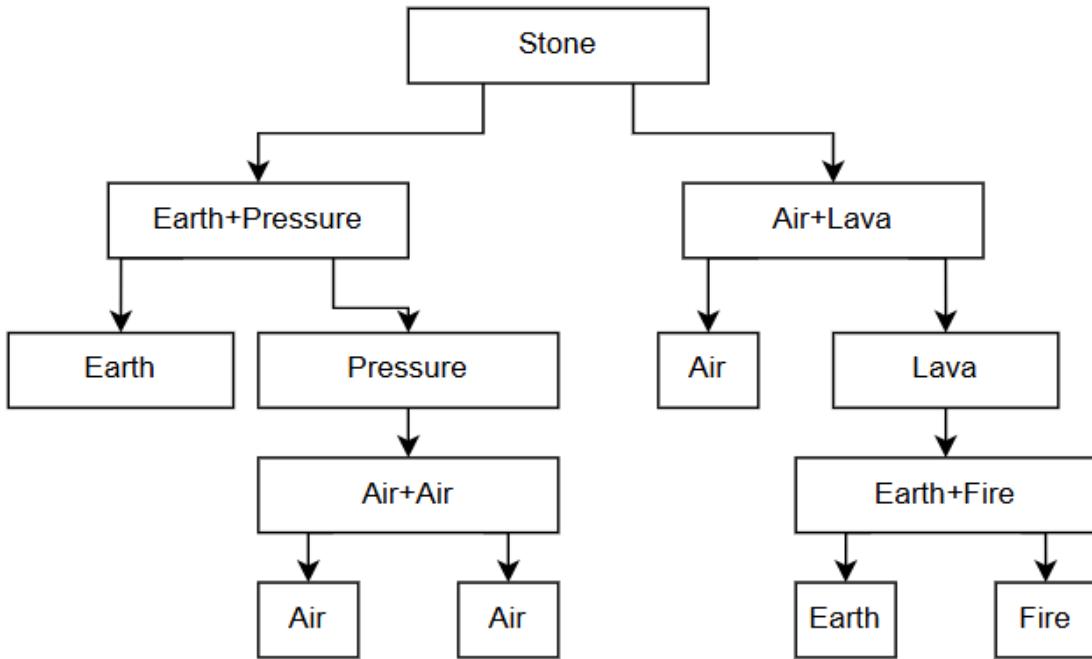
Setiap hasil pencarian akan disertai dengan informasi terkait jumlah *nodes* yang dilalui untuk mencapai resep tersebut serta lama durasi pencarian.



Gambar 8. Informasi Tambahan

3.4 Ilustrasi Kasus

Misal hendak dicari suatu elemen stone dengan seluruh kemungkinan resep seperti berikut



Gambar 9. Ilustrasi Kasus

3.4.1 BFS (Single)

Head	Queue	Visited Node	Solution Path
Stone	[Earth, Pressure, Air, Lava]	[]	[]
Earth	[Pressure, Air, Lava]	[Stone]	[[Stone, Earth]]
Pressure	[Air, Lava, Air, Air]	[Stone, Earth]	[[Stone, Earth]]
Air	[Lava, Air, Air]	[Stone, Earth, Pressure]	[[Stone, Earth], [Stone, Air]]
Lava	[Air, Air, Earth, Fire]	[Stone, Earth, Pressure, Air]	[[Stone, Earth], [Stone, Air]]
Air	[Air, Earth, Fire]	[Stone, Earth, Pressure, Air, Lava]	[[Stone, Earth], [Stone, Air]]

Air	[Earth, Fire]	[Stone, Earth, Pressure, Air, Lava]	[[Stone, Earth], [Stone, Air], [Stone, Pressure, Air], [Stone, Pressure, Air]]
(Rute menuju sebuah resep telah ditemukan)			

[
[Stone, Earth],
[Stone, Pressure, Air],
[Stone, Pressure, Air]
]

3.4.2 BFS (Multiple)

Asumsikan akan dicari 2 *recipe* untuk membuat Stone. Dua elemen pada sebuah head berarti bahwa kedua head tersebut diproses bersamaan dengan *multithreading*.

Head	Queue	Visited Node	Solution Path	Banyak Resep yang Ditemukan
Stone	[Earth, Pressure, Air, Lava]	[]	[]	0
Earth Pressure Air Lava	[Air, Air, Earth, Fire]	[Stone]	[[Stone, Earth], [Stone, Air]]	0
Air Air Earth Fire	[]	[Stone, Earth, Pressure, Air, Lava]	[[Stone, Earth], [Stone, Air], [Stone, Pressure, Air], [Stone, Pressure, Air], [Stone, Lava, Earth], [Stone, Lava, Fire]]	2
(Sudah ditemukan 2 alternatif metode membuat A)				
Solution path untuk setiap alternatif:				
[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air]]				

dan
[
[Stone, Air],
[Stone, Lava, Earth],
[Stone, Lava, Fire]
]

3.4.3 DFS (Single)

Head	Stack	Visited Node	Solution Path
Stone	[Earth, Pressure, Air, Lava]	[]	[]
Earth	[Pressure, Air, Lava]	[Stone]	[[Stone, Earth]]
Pressure	[Air, Air, Air, Lava]	[Stone, Earth]	[[Stone, Earth]]
Air	[Air, Air, Lava]	[Stone, Earth, Pressure]	[[Stone, Earth], [Stone, Pressure, Air]]
Air	[Air, Lava]	[Stone, Earth, Pressure, Air]	[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air]]

(Rute menuju sebuah resep telah ditemukan)

Solution path akhir:

[
[Stone, Earth],
[Stone, Pressure, Air],
[Stone, Pressure, Air]
]

3.4.4 DFS (Multiple)

Asumsikan akan dicari 2 *recipe* untuk membuat A. Dua elemen pada sebuah head berarti bahwa kedua head tersebut diproses bersamaan dengan *multithreading*.

Head	Stack	Visited Node	Solution Path	Banyak Resep

				yang Ditemukan
Stone	[Earth, Pressure, Air, Lava]	[]	[]	0
Earth Pressure	[Air, Air, Air, Lava]	[Stone]	[[Stone, Earth]]	0
Air Air	[Air, Lava]	[Stone, Earth, Pressure]	[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air]]	1
Air Lava	[Earth, Fire]	[Stone, Earth, Pressure, Air, Air]	[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air], [Stone, Air]]	1
Earth Fire	[]	[Stone, Earth, Pressure, Air, Air, Lava]	[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air], [Stone, Air], [Stone, Lava, Earth], [Stone, Lava, Fire]]	2
(Sudah ditemukan 2 alternatif metode membuat A)				
Solution path untuk setiap alternatif:				
<p>[[Stone, Earth], [Stone, Pressure, Air], [Stone, Pressure, Air]]</p> <p>dan</p> <p>[[Stone, Air], [Stone, Lava, Earth], [Stone, Lava, Fire]]</p>				

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Struktur Data

4.1.1. Struktur Data Umum

- a. `AlchemyEntry` : merepresentasikan hasil scraping secara keseluruhan. `Name` menyimpan nama elemen dan `Combines` menyimpan *array of recipe* dengan format string "`{Recipe1} + {Recipe2}`".

```
type AlchemyEntry struct {
    Name      string      `json:"element"`
    Combines []string    `json:"recipes"`
}
```

- b. `Pair_recipe`: merepresentasikan pair resep dalam bentuk pointer. Ini digunakan sebagai *children* dari `RecipeTree`. `First` adalah elemen pertama dan `Second` adalah elemen kedua. Tidak berlaku komutatifnya.

```
type Pair_recipe struct {
    First  *RecipeTree
    Second *RecipeTree
}
```

- c. `RecipeTree`: merepresentasikan *tree* yang digunakan sebagai output dari tiap algoritma yang digunakan. `Name` adalah nama elemen dari *node* tersebut, sedangkan `Children` berisi *pointer to array* `Pair_recipe` yang menyimpan resep-resep yang membentuk elemen tersebut

```
type RecipeTree struct {
    Name      string      `json:"name"`
    Children []*Pair_recipe `json:"children,omitempty"`
}
```

- d. `Recipe`: merepresentasikan pair resep dalam bentuk *integer*. Ini digunakan pada `MapperIdxToRecipes`.

```
type Recipe struct {
    First int
    Second int
}
```

4.1.2. Struktur Data untuk `bfs_multiple.go`:

`bfsTask` : merepresentasikan *task* yang dikirim kepada *worker* dalam penerapan multithreading. `Idx` berisi index dari elemen saat ini, `Node` berisi `RecipeTree` untuk menyimpan node dari elemen saat ini, dan `depth` menyimpan kedalaman node terkait.

```
Type bfSTask struct {
    Idx    int
    Node   *data_type.RecipeTree
    Depth  int
}
```

4.1.3. Struktur Data untuk bfs_single.go

a. queueItem

Struktur data ini digunakan untuk menyimpan index resep dan tree node. Pendekatannya dilakukan dengan *queue* yang diappend ketika elemen tersebut merupakan bagian dari resep yang dicari.

```
type queueItem struct {
    idx      int
    node     *data_type.RecipeTree
    distance int
}
```

4.1.4. Struktur Data untuk dfs_multiple.go

a. Counter

Struktur data ini bertugas sebagai variabel global yang akan menyimpan informasi mengenai id dari elemen yang sedang dicari kombinasi resepnya, mutex untuk keamanan *multithreading*, count untuk menghitung banyak kombinasi resep yang telah ditemukan, dan limit untuk menyimpan banyak kombinasi resep yang hendak dicari.

```
type Counter struct {
    rootId int
    mu      sync.Mutex
    count   int
    limit   int
}
```

4.1.4. Struktur Data untuk bidirectional_single.go

a. Path_element

Ini digunakan sebagai elemen dari *path*, dengan *Idx* menyimpan indeks dari elemen pada posisi saat ini dan *Result* menyimpan indeks dari elemen hasil posisi saat ini dan posisi sebelumnya. Pengecualian untuk elemen pertama, nilai *Result* adalah indeks dirinya sendiri

```
Type Path_element struct {
    Idx int
    Result int
}
```

4.2 Fungsi dan Prosedur

4.2.1 BFS (Single)

4.2.1.1 func FindShortestPath(targetURL string) (*data_type.RecipeTree, int)

Fungsi ini bertugas untuk membentuk *tree*, mencari jalur terpendek dengan algoritma BFS.

```
func FindShortestPath(targetURL string) (*data_type.RecipeTree, int) {
    targetIdx, ok := scrapping.MapperNameToIdx[targetURL]
    if !ok || targetIdx == -1 {
        fmt.Println("Error: Invalid target URL")
        return nil, 0
    }

    // queue untuk BFS - menyimpan index resep dan tree node
    type queueItem struct {
        idx      int
        node     *data_type.RecipeTree
        distance int
        path    map[int]bool // track elemen dalam path saat ini
    }
    tier := scrapping.MapperIdxToTier[targetIdx]
    if tier == -1 || tier == 0 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[targetIdx], Children: nil}, 1
    }

    // map untuk menyimpan node yang sudah dikunjungi dan jaraknya
    visited := make(map[int]int)

    root := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[targetIdx]}

    // inisialisasi queue dengan target resep
    initialPath := make(map[int]bool)
    initialPath[targetIdx] = true // tandai target sebagai bagian dari
path

    queue := []queueItem{{idx: targetIdx, node: root, distance: 0, path:
initialPath}}
    visited[targetIdx] = 0

    // map untuk menyimpan resep terbaik untuk setiap elemen
    bestRecipes := make(map[int]*data_type.Recipe)

    // BFS untuk mencari jalur terpendek
    visitedCount := 1
    for len(queue) > 0 {
        // ambil item pertama dari queue
        current := queue[0]
        queue = queue[1:]

        currentIdx := current.idx
        currentNode := current.node
```

```

currentDistance := current.distance
currentPath := current.path

// jika sudah mencapai elemen dasar, stop
if currentIdx <= 4 {
    continue
}

// semua resep yang bisa membuat elemen saat ini
recipes := scrapping.MapperIdxToRecipes[currentIdx]

if len(recipes) == 0 {
    continue
}

// temukan resep terbaik (yang menggunakan elemen dari tier yang
lebih rendah)
var bestRecipe *data_type.Recipe
bestScore := -1

currentTier := scrapping.MapperIdxToTier[currentIdx]

for _, recipe := range recipes {
    firstIdx := recipe.First
    secondIdx := recipe.Second

    firstTier := scrapping.MapperIdxToTier[firstIdx]
    secondTier := scrapping.MapperIdxToTier[secondIdx]

    // pastikan kedua bahan berasal dari tier yang lebih
rendah
    if firstTier >= currentTier || secondTier >= currentTier {
        continue
    }

    // mencegah siklus dan mencegah menggunakan elemen target
sebagai bahan
    if currentPath[firstIdx] || currentPath[secondIdx] {
        continue
    }

    // hitung skor resep (prioritaskan elemen dasar)
    score := 0
    if firstIdx <= 4 {
        score++
    }
    if secondIdx <= 4 {
        score++
    }

    // pilih resep dengan skor tertinggi atau yang belum
pernah dikunjungi
    if score > bestScore || bestRecipe == nil {

```

```

        bestRecipe = &recipe
        bestScore = score
    }

}

if bestRecipe == nil {
    continue
}

bestRecipes[currentIdx] = bestRecipe

firstIdx := bestRecipe.First
secondIdx := bestRecipe.Second

visitedCount += 2
// buat node untuk bahan pertama dan kedua
firstNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[firstIdx]}
secondNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[secondIdx]}

// tambahkan ke dalam tree
pair := &data_type.Pair_recipe{First: firstNode, Second:
secondNode}
currentNode.Children = append(currentNode.Children, pair)

newDistance := currentDistance + 1

// buat copy dari path saat ini untuk digunakan di jalur baru
newPathFirst := make(map[int]bool)
for k, v := range currentPath {
    newPathFirst[k] = v
}
newPathFirst[firstIdx] = true

newPathSecond := make(map[int]bool)
for k, v := range currentPath {
    newPathSecond[k] = v
}
newPathSecond[secondIdx] = true

if dist, found := visited[firstIdx]; !found || newDistance <
dist {
    visited[firstIdx] = newDistance
    queue = append(queue, queueItem{idx: firstIdx, node:
firstNode, distance: newDistance, path: newPathFirst})
}

if dist, found := visited[secondIdx]; !found || newDistance <
dist {
    visited[secondIdx] = newDistance
    queue = append(queue, queueItem{idx: secondIdx, node:
secondNode, distance: newDistance, path: newPathSecond})
}

```

```

        }
    }

    // rekonstruksi jalur optimal
    optimalPath := ConstructOptimalPath(targetIdx, bestRecipes)
    return optimalPath, visitedCount
}

```

4.2.1.2 func ConstructOptimalPath(targetIdx int, bestRecipes map[int]*data_type.Recipe) *data_type.RecipeTree

Fungsi ini bertugas untuk merekonstruksi jalur optimal dari suatu *tree* yang telah terbentuk..

```

func ConstructOptimalPath(targetIdx int, bestRecipes
map[int] *data_type.Recipe) *data_type.RecipeTree {
    root := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[targetIdx]}

    // DFS untuk membangun pohon resep
    BuildRecipeTree(root, targetIdx, bestRecipes, make(map[int]bool))

    return root
}

```

4.2.2 BFS (Multiple)

4.2.2.1 func Bfs_multiple_recipe(url string, bound int) (*data_type.RecipeTree, int)

Fungsi ini bertugas untuk melakukan pencarian *multiple recipe* dengan BFS secara *multithreading*.

```

func Bfs_multiple_recipe(url string, bound int) (*data_type.RecipeTree, int) {
    idx := scrapping.MapperNameToIdx[url]
    if idx == -1 {
        fmt.Println("Error: Invalid URL")
        return nil, 0
    }
    tier := scrapping.MapperIdxToTier[idx]
    if tier == -1 || tier == 0 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[idx], Children: nil}, 1
    }

    root := &data_type.RecipeTree{Name: scrapping.MapperIdxToName[idx]}
    queue := make(chan bfsTask, 1000)
    visited := make([]bool, 720)
    var visitedMu sync.Mutex

    countNode := 1
    var count int
    var countMu sync.Mutex

```

```

var wg sync.WaitGroup
queue <- bfsTask{Idx: idx, Node: root, Depth: 0}
wg.Add(1)

for i := 0; i < 8; i++ {
    go func() {
        for task := range queue {
            idx := task.Idx
            node := task.Node
            depth := task.Depth

            visitedMu.Lock()
            if visited[idx] {
                visitedMu.Unlock()
                wg.Done()
                continue
            }
            visited[idx] = true
            visitedMu.Unlock()

            if idx <= 4 {
                wg.Done()
                continue
            }

            recipes := scrapping.MapperIdxToRecipes[idx]
            currentTier := scrapping.MapperIdxToTier[idx]
            for _, recipe := range recipes {
                firstIdx := recipe.First
                secondIdx := recipe.Second
                if currentTier <
scrapping.MapperIdxToTier[firstIdx] || currentTier <
scrapping.MapperIdxToTier[secondIdx] {
                    continue
                }

                countMu.Lock()
                if count >= bound {
                    countMu.Unlock()
                    break
                }
                if countRecipe(root) >= bound {
                    count++
                }
                countMu.Unlock()
                countNode += 2
                firstNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[firstIdx]}
                secondNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[secondIdx]}
                pair := &data_type.Pair_recipe{First:
firstNode, Second: secondNode}
                node.Children = append(node.Children, pair)
            }
        }
    }
}

```

```

        wg.Add(2)
        queue <- bfsTask{Idx: firstIdx, Node:
firstNode, Depth: depth + 1}
        queue <- bfsTask{Idx: secondIdx, Node:
secondNode, Depth: depth + 1}
    }
    wg.Done()
}
}()

wg.Wait()
close(queue)

PruneNonTerminalParallel(root)
PruneNonTerminalParallel(root)
newRoot, nodeAddition := completeTheRoot(root)
root = newRoot
countNode += nodeAddition

return root, countNode
}

```

4.2.2.2 func countRecipe(t *data_type.RecipeTree) int

Fungsi ini bertugas untuk menghitung banyaknya kombinasi resep untuk membuat *root node* dari *tree* yang diberikan.

```

func countRecipe(t *data_type.RecipeTree) int {
    currentId := scrapping.MapperNameToIdx[t.Name]

    if currentId == 0 || currentId == 1 || currentId == 2 || currentId ==
3 {
        return 1
    }

    totalWays := 0
    for i := 0; i < len(t.Children); i++ {
        firstRecipe := t.Children[i].First
        secondRecipe := t.Children[i].Second

        countFirst := countRecipe(firstRecipe)
        countSecond := countRecipe(secondRecipe)

        totalWays += countFirst * countSecond
    }
    return totalWays
}

```

4.2.2.3 func completeTheRoot(root *data_type.RecipeTree) (*data_type.RecipeTree, int)

Fungsi ini bertugas untuk melengkapi *node-node* yang belum mencapai elemen dasar ketika pencarian BFS telah menemukan jumlah *recipe* yang sesuai.

```
func completeTheRoot(root *data_type.RecipeTree) (*data_type.RecipeTree, int) {
    idx := scrapping.MapperNameToIdx[root.Name]
    if idx > 4 || len(root.Children) == 0 {
        newRoot, nodeCount := FindShortestPath(root.Name)
        return newRoot, nodeCount
    } else {
        nodeCount := 0
        for i := 0; i < len(root.Children); i++ {
            firstRecipe, firstNodeCount :=
completeTheRoot(root.Children[i].First)
            secondRecipe, secondNodeCount :=
completeTheRoot(root.Children[i].Second)

            root.Children[i].First = firstRecipe
            root.Children[i].Second = secondRecipe
            nodeCount += firstNodeCount + secondNodeCount
        }
        return root, nodeCount
    }
}
```

4.2.2.4 Pruning

Fungsi-fungsi ini bertugas untuk memangkas node-node dengan children yang tidak memiliki pasangan resep yang keduanya merupakan elemen dasar.

```
func PruneNonTerminalParallel(root *data_type.RecipeTree) {
    pruneTreeParallel(root)
}

func pruneTreeParallel(node *data_type.RecipeTree) bool {
    if node == nil {
        return false
    }

    if len(node.Children) == 0 {
        return isBasicElement(node)
    }

    var wg sync.WaitGroup
    mu := sync.Mutex{}
    validChildren := make([]*data_type.Pair_recipe, 0)

    for _, pair := range node.Children {
        wg.Add(1)
        go func(pair *data_type.Pair_recipe) {
            defer wg.Done()
            firstValid := pruneTreeParallel(pair.First)
            secondValid := pruneTreeParallel(pair.Second)
```

```

        if len(pair.First.Children) == 0 &&
len(pair.Second.Children) == 0 {
            if isBasicElement(pair.First) &&
isBasicElement(pair.Second) {
                mu.Lock()
                validChildren = append(validChildren, pair)
                mu.Unlock()
            }
        } else if firstValid || secondValid {
            mu.Lock()
            validChildren = append(validChildren, pair)
            mu.Unlock()
        }
    } (pair)
}

wg.Wait()
node.Children = validChildren
return len(validChildren) > 0
}

```

4.2.2.5 func isBasicElement(node *data_type.RecipeTree) bool

Fungsi ini bertugas untuk mengecek apakah suatu elemen merupakan elemen dasar.

```

func isBasicElement(node *data_type.RecipeTree) bool {
    idx, ok := scrapping.MapperNameToIdx[node.Name]
    return ok && idx <= 4
}

```

4.2.3 DFS (Fungsi Pembantu)

4.2.3.1 func CreateRecipeTreeFromName(name string) *data_type.RecipeTree

Fungsi ini bertugas sebagai konstruktor RecipeTree dengan masukan berupa nama elemen yang bersangkutan.

```

func CreateRecipeTreeFromName(name string) *data_type.RecipeTree {
    rootChild := [] *data_type.Pair_recipe{}
    return &data_type.RecipeTree{Name: name, Children: rootChild}
}

```

4.2.3.2 func CreateRecipeTreeFromId(id int) *data_type.RecipeTree

Fungsi ini bertugas sebagai konstruktor RecipeTree dengan masukan berupa id elemen yang bersangkutan.

```

func CreateRecipeTreeFromId(id int) *data_type.RecipeTree {
    rootChild := [] *data_type.Pair_recipe{}
    name := scrapping.MapperElmIdx[id]
    return &data_type.RecipeTree{Name: name, Children: rootChild}
}

```

4.2.3.3 func NodeCount(t *data_type.RecipeTree) int

Fungsi ini bertugas untuk menghitung jumlah node pada *tree* yang telah terbentuk.

```
func NodeCount(t *data_type.RecipeTree) int {
    currentId := scrapping.MapperIdxElm[t.Name]
    if currentId == 0 || currentId == 1 || currentId == 2 || currentId == 3 || getTier(currentId) == 9999 {
        return 1
    }

    total := 1
    for i := 0; i < len(t.Children); i++ {
        total += NodeCount(t.Children[i].First)
        total += NodeCount(t.Children[i].Second)
    }
    return total
}
```

4.2.4.4 func getTier(id int) int

Fungsi ini bertugas untuk mengembalikan tier dari suatu elemen dengan masukan berupa id elemen tersebut.

```
func getTier(id int) int {
    return scrapping.MapperIdxToTier[id]
```

4.2.4 DFS (Single)

4.2.4.1 func DFSSingle(t *data_type.RecipeTree)

Prosedur ini bertugas untuk membentuk *tree* hasil pencarian *recipe* dengan *t* sebagai *root node*.

```
func DFSSingle(t *data_type.RecipeTree) {
    currentId := scrapping.MapperNameToIdx[t.Name]

    // basis
    if currentId == 0 || currentId == 1 || currentId == 2 || currentId == 3 {
        return
    }

    // rekurens
    childrenList := scrapping.MapperIdxToRecipes[currentId]
    for i := 0; i < len(childrenList); i++ {
        idFirst := childrenList[i].First
        idSecond := childrenList[i].Second

        // skip yang tidak memenuhi kriteria
        if getTier(idFirst) >= getTier(currentId) || getTier(idSecond) >= getTier(currentId) {
            continue
        }

        // proses
        DFSSingle(&childrenList[i])
    }
}
```

```

    }

    firstRecipe := CreateRecipeTreeFromId(idFirst)
    secondRecipe := CreateRecipeTreeFromId(idSecond)

    currentPair := &data_type.Pair_recipe{First: firstRecipe,
Second: secondRecipe}
    t.Children = append(t.Children, currentPair)
    DFSSingle(firstRecipe)
    DFSSingle(secondRecipe)
    break
}

}

```

4.2.4.2 func DFSSingleEntryPoint(url string) (*data_type.RecipeTree, int)

Fungsi ini merupakan penghubung antara algoritma DFS dengan fungsi utama pada bagian *backend*.

```

func DFSSingleEntryPoint(url string) (*data_type.RecipeTree, int) {
    idx := scrapping.MapperNameToIdx[url]
    if idx == -1 {
        fmt.Println("Error: Invalid URL")
        return nil, 0
    }
    tier := scrapping.MapperIdxToTier[idx]
    if tier == -1 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[idx], Children: nil}, 1
    }
    root := &data_type.RecipeTree{Name: scrapping.MapperIdxToName[idx]}

    DFSSingle(root)
    return root, NodeCount(root)
}

```

4.2.5 DFS (Multiple)

4.2.5.1 Fungsi Terkait GlobalCounter

GlobalCounter merupakan suatu variabel global yang bertugas untuk menyimpan data jumlah resep yang sudah ditemukan sejauh pencarian. Terdapat beberapa fungsi pembantu untuk memanipulasi GlobalCounter, di antaranya fungsi setter, fungsi getter, fungsi TryAdd (menambahkan nilai GlobalCounter secara *thread safe*) dan fungsi isLimitReached (mengecek, secara *thread safe*, apakah nilai GlobalCounter sudah melewati batas yang diinginkan).

```

var GlobalCounter = Counter{rootId: 0, count: 0, limit: 0}

func (c *Counter) SetCounter(id int, limit int) {
    c.rootId = id
    c.count = 0
    c.limit = limit
}

```

```

}

func (c *Counter) GetCount() int {
    return c.count
}

func (c *Counter) TryAdd(n int) bool {
    c.mu.Lock()
    defer c.mu.Unlock()

    if c.count >= c.limit {
        return false
    }
    c.count += n
    return c.count < c.limit
}

func (c *Counter) IsLimitReached() bool {
    c.mu.Lock()
    defer c.mu.Unlock()
    return c.count >= c.limit
}

```

4.2.5.2 func DFSMultiple(t *data_type.RecipeTree, wg *sync.WaitGroup) int

Fungsi ini bertugas untuk membentuk *tree* hasil pencarian *recipe* secara *multithreading* dengan *t* sebagai *root node*, juga akan mengembalikan banyak alternatif kombinasi *recipe* yang ditemukan.

```

func DFSMultiple(t *data_type.RecipeTree, wg *sync.WaitGroup) int {
    if wg != nil {
        defer wg.Done()
    }

    currentId := scrapping.MapperNameToIdx[t.Name]

    // basis
    if currentId == 0 || currentId == 1 || currentId == 2 || currentId ==
3 {
        return 1
    }

    // rekurens
    childrenList := scrapping.MapperIdxToRecipes[currentId]

    // multithreading
    totalWays := 0
    for i := 0; i < len(childrenList); i++ {
        if i > 0 && GlobalCounter.IsLimitReached() {
            return 1
        }
        if totalWays >= GlobalCounter.limit {

```

```

        return totalWays
    }

    idFirst := childrenList[i].First
    idSecond := childrenList[i].Second
    if getTier(idFirst) >= getTier(currentId) || getTier(idSecond)
    >= getTier(currentId) {
        continue
    }

    firstRecipe := CreateRecipeTreeFromId(idFirst)
    secondRecipe := CreateRecipeTreeFromId(idSecond)
    currentPair := &data_type.Pair_recipe{First: firstRecipe,
Second: secondRecipe}
    t.Children = append(t.Children, currentPair)

    channel1 := make(chan int, 1)
    channel2 := make(chan int, 1)

    var childWg sync.WaitGroup
    childWg.Add(2)

    go func() {
        defer childWg.Done()
        channel1 <- DFSMultiple(firstRecipe, nil)
    }()
    go func() {
        defer childWg.Done()
        channel2 <- DFSMultiple(secondRecipe, nil)
    }()
}

childWg.Wait()

countFirst := <-channel1
countSecond := <-channel2
totalWays += countFirst * countSecond

if currentId == GlobalCounter.rootId {
    GlobalCounter.TryAdd(countFirst * countSecond)
}
}

return totalWays
}

```

4.2.5.3 func DFSMultipleEntryPoint(url string) (*data_type.RecipeTree, int)

Fungsi ini merupakan penghubung antara algoritma DFS dengan fungsi utama pada bagian *backend*.

```

func DFSMultipleEntryPoint(url string) (*data_type.RecipeTree, int) {
    idx := scrapping.MapperNameToIdx[url]
    if idx == -1 {

```

```

        fmt.Println("Error: Invalid URL")
        return nil, 0
    }
    tier := scrapping.MapperIdxToTier[idx]
    if tier == -1 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[idx], Children: nil}, 1
    }
    root := &data_type.RecipeTree{Name: scrapping.MapperIdxToName[idx]}
    GlobalCounter.SetCounter(scrapping.MapperIdxElm[root.Name], 100)

    var wg sync.WaitGroup
    wg.Add(1)
    go func() {
        DFSMultiple(root, &wg)
    }()
    wg.Wait()
    return root, NodeCount(root)
}

```

4.2.6 Scraping

4.2.6.1 func normalizeText(input string) string

Fungsi ini bertugas untuk membersihkan spasi berlebih dari suatu teks

```

func normalizeText(input string) string {
    pattern := regexp.MustCompile(`\s+`)
    return strings.TrimSpace(pattern.ReplaceAllString(input, " "))
}

```

4.2.6.2 func extractCombinations(td *goquery.Selection) []string

Fungsi ini bertugas untuk mengurai cell tabel menjadi daftar resep

```

func extractCombinations(td *goquery.Selection) []string {
    var results []string

    td.Find("li").Each(func(_ int, item *goquery.Selection) {
        content := normalizeText(item.Text())
        if content != "" {
            results = append(results, content)
        }
    })

    // fallback jika tidak ada <li>
    if len(results) == 0 {
        fallback := normalizeText(td.Text())
        if fallback != "" {
            results = append(results, fallback)
        }
    }
}

```

```
        return results
    }
```

4.2.6.3 func FetchAllData() ([]AlchemyEntry, error)

Prosedur ini bertugas untuk memuat semua data dari file JSON ke slice

```
func FetchAllData() ([]AlchemyEntry, error) {
    source, err := os.Open("little_alchemy_elements.json")
    if err != nil {
        return nil, errors.New("failed to open saved data file")
    }
    defer source.Close()

    container := make(map[string][]AlchemyEntry)
    if err := json.NewDecoder(source).Decode(&container); err != nil {
        return nil, errors.New("JSON decoding failed")
    }

    var flatList []AlchemyEntry
    for _, group := range container {
        flatList = append(flatList, group...)
    }
    return flatList, nil
}
```

4.2.6.4 func StartScraper() error

Fungsi ini bertugas untuk menjalankan proses *scraping*, juga mengembalikan pesan kesalahan jika ada.

```
func StartScraper() error {
    targetURL :=
"https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"
    response, err := http.Get(targetURL)
    if err != nil {
        return fmt.Errorf("connection error: %w", err)
    }
    defer response.Body.Close()

    if response.StatusCode != http.StatusOK {
        return fmt.Errorf("received bad status code: %d %s",
response.StatusCode, response.Status)
    }

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

    idxCounter := 0
```

```

doc.Find("h3").Each(func(_ int, header *goquery.Selection) {
    span := header.Find("span.mw-headline")
    if span.Length() == 0 {
        return
    }

    category := normalizeText(span.Text())
    FinalData[category] = []AlchemyEntry{}

    // Tentukan tier
    var tier int
    switch strings.ToLower(category) {
    case "special element":
        tier = -1
    case "starting elements":
        tier = 0
    default:
        _, err := fmt.Sscanf(strings.ToLower(category), "tier %d
elements", &tier)
        if err != nil {
            tier = -999 // fallback tier
        }
    }

    // Cari tabel berikutnya setelah h3
    tableNode := header.Next()
    for tableNode != nil && goquery.NodeName(tableNode) != "table" {
        tableNode = tableNode.Next()
    }
    if tableNode == nil {
        return
    }

    tableNode.Find("tr").Each(func(rowIndex int, row
*goquery.Selection) {
        if rowIndex == 0 {
            return // skip header
        }

        cells := row.Find("td")
        if cells.Length() >= 2 {
            elementName := normalizeText(cells.Eq(0).Text())
            recipeList := extractCombinations(cells.Eq(1))

            FinalData[category] = append(FinalData[category],
AlchemyEntry{
                Name:      elementName,
                Combines: recipeList,
            })

            // Isi mapper jika belum ada
            if _, exists := MapperNameToIdx[elementName];

```

```

!exists {
    MapperNameToIdx[elementName] = idxCounter
    MapperIdxToName[idxCounter] = elementName
    MapperIdxToTier[idxCounter] = tier
    idxCounter++
}
}

// Bangun MapperIdxToRecipes setelah FinalData selesai
for _, entries := range FinalData {
    for _, entry := range entries {
        resultIdx, ok := MapperNameToIdx[entry.Name]
        if !ok {
            continue
        }
        for _, recipe := range entry.Combines {
            parts := strings.Split(recipe, "+")
            if len(parts) != 2 {
                continue
            }
            first := normalizeText(parts[0])
            second := normalizeText(parts[1])
            firstIdx, ok1 := MapperNameToIdx[first]
            secondIdx, ok2 := MapperNameToIdx[second]
            pair := data_type.Recipe{First: firstIdx, Second:
secondIdx}
            if ok1 && ok2 {
                MapperIdxToRecipes[resultIdx] =
append(MapperIdxToRecipes[resultIdx], pair)
            }
        }
    }
}

FinalDataSaveToFile()
MapperNameToIdxSaveToFile()
MapperIdxToNameSaveToFile()
MapperIdxToTierSaveToFile()
MapperIdxToRecipesSaveToFile()

return errors.New("scraping completed")
}

```

4.2.6.5 Save

Prosedur-prosedur berikut bertugas untuk menyimpan data hasil scraping ke suatu file JSON.

```

func FinalDataSaveToFile() {
    // Simpan FinalData ke file JSON
    output, err := os.Create("scraper/JSON/little_alchemy_elements.json")
    if err != nil {

```

```

        fmt.Println("Error creating JSON file:", err)
        return
    }
    defer output.Close()

    writer := json.NewEncoder(output)
    writer.SetIndent("", " ")
    if err := writer.Encode(FinalData); err != nil {
        fmt.Println("Error writing JSON content:", err)
        return
    }
    fmt.Println("Final data saved to little_alchemy_elements.json")
}

func MapperNameToIdxSaveToFile() {
    // Simpan MapperNameToIdx ke file JSON
    output, err := os.Create("scraper/JSON/MapperNameToIdx.json")
    if err != nil {
        fmt.Println("Error creating JSON file:", err)
        return
    }
    defer output.Close()

    writer := json.NewEncoder(output)
    writer.SetIndent("", " ")
    if err := writer.Encode(MapperNameToIdx); err != nil {
        fmt.Println("Error writing JSON content:", err)
        return
    }
    fmt.Println("Mapper saved to MapperNameToIdx.json")
}

func MapperIdxToNameSaveToFile() {
    // Simpan MapperIdxToName ke file JSON
    output, err := os.Create("scraper/JSON/MapperIdxToName.json")
    if err != nil {
        fmt.Println("Error creating JSON file:", err)
        return
    }
    defer output.Close()

    writer := json.NewEncoder(output)
    writer.SetIndent("", " ")
    if err := writer.Encode(MapperIdxToName); err != nil {
        fmt.Println("Error writing JSON content:", err)
        return
    }
    fmt.Println("Mapper saved to MapperIdxToName.json")
}

func MapperIdxToTierSaveToFile() {
    // Simpan MapperIdxToTier ke file JSON
    output, err := os.Create("scraper/JSON/MapperIdxToTier.json")
}

```

```

        if err != nil {
            fmt.Println("Error creating JSON file:", err)
            return
        }
        defer output.Close()

        writer := json.NewEncoder(output)
        writer.SetIndent("", "    ")
        if err := writer.Encode(MapperIdxToTier); err != nil {
            fmt.Println("Error writing JSON content:", err)
            return
        }
        fmt.Println("Mapper saved to MapperIdxToTier.json")
    }

func MapperIdxToRecipesSaveToFile() {
    // Simpan MapperIdxToRecipes ke file JSON
    output, err := os.Create("scraper/JSON/MapperIdxToRecipes.json")
    if err != nil {
        fmt.Println("Error creating JSON file:", err)
        return
    }
    defer output.Close()

    writer := json.NewEncoder(output)
    writer.SetIndent("", "    ")
    if err := writer.Encode(MapperIdxToRecipes); err != nil {
        fmt.Println("Error writing JSON content:", err)
        return
    }
    fmt.Println("Mapper saved to MapperIdxToRecipes.json")
}

```

4.2.7 Fungsi utama (Main)

```

func main() {
    scrapping.StartScraper()
    r := gin.Default()

    r.Use(cors.New(cors.Config{
        AllowOrigins:     []string{"http://localhost:3000"},
        AllowMethods:     []string{"GET", "POST", "PUT", "DELETE",
"OPTIONS"},
        AllowHeaders:     []string{"Origin", "Content-Type", "Accept",
"Authorization"},
        ExposeHeaders:    []string{"Content-Length"},
        AllowCredentials: true,
        MaxAge:           12 * time.Hour,
    }))

    r.GET("/api/ping", func(c *gin.Context) {
        c.JSON(http.StatusOK, gin.H{

```

```

        "message": "pong",
    })
}

r.POST("/api/search", func(c *gin.Context) {
    var req SearchRequest
    if err := c.ShouldBindJSON(&req); err != nil {
        log.Printf("Error binding JSON: %v", err)
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    log.Printf("Received search request: Query=%s, Mode=%s,
Algorithm=%s, CountRicipe=%d", req.Query, req.Mode, req.Algorithm,
req.CountRicipe)

    if req.Query == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Query cannot
be empty"})
        return
    }
    if req.Mode != "single" && req.Mode != "multiple" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Mode must be
'single' or 'multiple'"})
        return
    }
    if req.Algorithm != "bfs" && req.Algorithm != "dfs" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Algorithm
must be 'bfs', 'dfs', or 'bidirectional'"})
        return
    }
}

var result interface{}

if req.Algorithm == "bfs" {
    if req.Mode == "multiple" {
        log.Printf("Calling BFS multiple recipe with query:
%s", req.Query)

        // hitung durasi
        startTime := time.Now()
        tree, count :=
bfs_multiple_recipe.Bfs_multiple_recipe(req.Query, req.CountRicipe)
        duration := time.Since(startTime)
        if tree == nil {
            log.Printf("BFS returned nil for query: %s",
req.Query)
            c.JSON(http.StatusInternalServerError,
gin.H{"error": "Failed to process recipe"})
            return
        }
    }
}

```

```

        result = gin.H{
            "tree": tree,
            "count": count,
            "duration": duration.Seconds(),
        }

        // simpan hasil ke file JSON
        jsonBytes, err := json.MarshalIndent(result, "", "")
    )

        if err != nil {
            log.Printf("Gagal mengubah ke JSON: %v", err)
        } else {
            err := os.WriteFile("output_recipe.json",
        jsonBytes, 0644)
            if err != nil {
                log.Printf("Gagal menulis file JSON:
%v", err)
            } else {
                log.Println("Berhasil menyimpan hasil
pencarian ke output_recipe.json")
            }
        }

    } else {
        log.Printf("Calling BFS multiple recipe with query:
%s", req.Query)
        // hitung durasi
        startTime := time.Now()
        tree, count :=
bfs_shortest.FindShortestPath(req.Query)
        duration := time.Since(startTime)

        if tree == nil {
            log.Printf("BFS returned nil for query: %s",
req.Query)
            c.JSON(http.StatusInternalServerError,
gin.H{"error": "Failed to process recipe"})
            return
        }
        result = gin.H{
            "tree": tree,
            "count": count,
            "duration": duration.Seconds(),
        }

        // simpan hasil ke file JSON
        jsonBytes, err := json.MarshalIndent(result, "", "")
    )

        if err != nil {
            log.Printf("Gagal mengubah ke JSON: %v", err)
        } else {
            err := os.WriteFile("output_recipe.json",
        jsonBytes, 0644)

```

```

                if err != nil {
                    log.Printf("Gagal menulis file JSON: %v", err)
                } else {
                    log.Println("Berhasil menyimpan hasil pencarian ke output_recipe.json")
                }
            }
        }
    } else if req.Algorithm == "dfs" {
        if req.Mode == "multiple" {
            log.Printf("Calling BFS multiple recipe with query: %s", req.Query)

            // hitung durasi
            startTime := time.Now()
            tree, count := dfs.DFSMultipleEntryPoint(req.Query)
            duration := time.Since(startTime)
            if tree == nil {
                log.Printf("BFS returned nil for query: %s", req.Query)
                c.JSON(http.StatusInternalServerError,
gin.H{"error": "Failed to process recipe"})
                return
            }
            result = gin.H{
                "tree": tree,
                "count": count,
                "duration": duration.Seconds(),
            }
            // simpan hasil ke file JSON
            jsonBytes, err := json.MarshalIndent(result, "", "")
            if err != nil {
                log.Printf("Gagal mengubah ke JSON: %v", err)
            } else {
                err := os.WriteFile("output_recipe.json",
jsonBytes, 0644)
                if err != nil {
                    log.Printf("Gagal menulis file JSON: %v", err)
                } else {
                    log.Println("Berhasil menyimpan hasil pencarian ke output_recipe.json")
                }
            }
        } else {
            log.Printf("Calling BFS multiple recipe with query: %s", req.Query)

            // hitung durasi
            startTime := time.Now()
            tree, count := dfs.DFSSingleEntryPoint(req.Query)
            duration := time.Since(startTime)

```

```

        if tree == nil {
            log.Printf("BFS returned nil for query: %s",
req.Query)
            c.JSON(http.StatusInternalServerError,
gin.H{"error": "Failed to process recipe"})
            return
        }
        result = gin.H{
            "tree": tree,
            "count": count,
            "duration": duration.Seconds(),
        }

        // simpan hasil ke file JSON
        jsonBytes, err := json.MarshalIndent(result, "", "")
        if err != nil {
            log.Printf("Gagal mengubah ke JSON: %v", err)
        } else {
            err := os.WriteFile("output_recipe.json",
jsonBytes, 0644)
            if err != nil {
                log.Println("Gagal menulis file JSON:
%v", err)
            } else {
                log.Println("Berhasil menyimpan hasil
pencarian ke output_recipe.json")
            }
        }
    }
} else if req.Algorithm == "bidirectional"{
    if req.Mode == "multiple"{
        // disini bidirectional
    }
}

log.Printf("Sending response with result: %+v", result)
c.JSON(http.StatusOK, gin.H{
    "success": true,
    "data": result,
})
}

log.Println("Server is running on port 8080")
r.Run(":8080")
}

```

4.2.8. Bidirectional (Single)

Fungsi ini melakukan seluruh proses *Bidirectional Searching* mode single

```

func BidirectionalSingle(targetURL string) (*data_type.RecipeTree, int) {
    targetIdx, ok := scrapping.MapperNameToIdx[targetURL]
    if !ok || targetIdx == -1 {
        fmt.Println("Error: Invalid target URL")
        return nil, 0
    }

    tier := scrapping.MapperIdxToTier[targetIdx]
    if tier == -1 || tier == 0 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[targetIdx], Children: nil}, 1
    }

    ForwardQueue := [] []Path_element{}
    BackwardQueue := [] []Path_element{}
    ForwardVisited := make(map[int] []Path_element) // ForwardVisited[x]
menyimpan path yang menghasilkan x dari elemen dasar
(ForwardVisited[x] [].last() != x karena akan menghasilkan x)
    BackwardVisited := make(map[int] []Path_element) //
BackwardVisited[(x,y)] menyimpan path yang dimulai dari pair (x,y) hingga
menghasilkan targetIdx

    recipeMap:= scrapping.MapperPairToIdxs
    reverseMap := scrapping.MapperIdxToRecipes
    nodeCount := 0

    for i:=0; i<4; i++{
        if recipeMap[i] != nil {
            for key, val := range recipeMap[i] {
                result := val
                r1 := Path_element{Idx: i, Result: i}
                r2 := Path_element{Idx: key, Result: result}
                ForwardQueue = append(ForwardQueue,
[]Path_element{r1, r2})
                ForwardVisited[i] = []Path_element{r1}
                ForwardVisited[key] = []Path_element{r2}
                ForwardVisited[result] = []Path_element{r1, r2}
                nodeCount += 2
            }
        }
    }

    targetElmt := Path_element{Idx: targetIdx, Result: targetIdx}
    BackwardQueue = append(BackwardQueue, []Path_element{targetElmt})
    nodeCount += 1

    for len(ForwardQueue) > 0 && len(BackwardQueue) > 0 {
        // Proses Forward
        ForwardPath := ForwardQueue[0]
        ForwardQueue = ForwardQueue[1:]
        fwdlst := ForwardPath[len(ForwardPath)-1]
        fwdlstRes := fwdlst.Result

```

```

nextElmts := recipeMap[fwdlastRes]
for key, res := range nextElmts {
    if _, visited := ForwardVisited[res]; visited {
        continue
    }
    nextPathElmt := Path_element{Idx: key, Result: res}
    tempForwardPath := append(ForwardPath, nextPathElmt)
    ForwardQueue = append(ForwardQueue, tempForwardPath)
    ForwardVisited[res] = tempForwardPath
    nodeCount ++
    if BackwardVisited[res] != nil {
        finalPath := []Path_element{}
        finalPath = append(finalPath, ForwardPath...)
        finalPath = append(finalPath,
reversePath(BackwardVisited[res])...)
        finalRoot := pathTotree(finalPath)
        return finalRoot, nodeCount
    }
}

// Backward process
BackwardPath := BackwardQueue[0]
BackwardQueue = BackwardQueue[1:]
bwlast := BackwardPath[len(BackwardPath)-1]
bwlastIdx := bwlast.Idx
recipes := reverseMap[bwlastIdx]
for _, recipe := range recipes {
    if _, visited1 := BackwardVisited[recipe.First]; visited1 {
        continue
    }
    if _, visited2 := BackwardVisited[recipe.Second]; visited2 {
        continue
    }
    firstPathElmt := Path_element{Idx: recipe.First, Result:
bwlastIdx}
    secondPathElmt := Path_element{Idx: recipe.Second, Result:
bwlastIdx}
    firstPathElmtBasic := Path_element{Idx: recipe.First,
Result: recipe.First}
    secondPathElmtBasic := Path_element{Idx: recipe.Second,
Result: recipe.Second}
    newBackwardPath := BackwardPath[:len(BackwardPath)-1]
    newBackwardPath1 := append(newBackwardPath, firstPathElmt,
secondPathElmtBasic)
    newBackwardPath2 := append(newBackwardPath,
secondPathElmt, firstPathElmtBasic)
    BackwardQueue = append(BackwardQueue, newBackwardPath1)
    BackwardQueue = append(BackwardQueue, newBackwardPath2)
    BackwardVisited[recipe.Second] =
append(BackwardVisited[recipe.Second], newBackwardPath1...)
    BackwardVisited[recipe.First] =
append(BackwardVisited[recipe.First], newBackwardPath2...)

```

```

        nodeCount += 2
        if ForwardVisited[recipe.First] != nil {
            finalPath := []Path_element{}
            finalPath = append(finalPath,
ForwardVisited[recipe.First]...)
            finalPath = append(finalPath, newBackwardPath2...)
            finalRoot := pathTotree(finalPath)
            return finalRoot, nodeCount
        }
        if ForwardVisited[recipe.Second] != nil {
            finalPath := []Path_element{}
            finalPath = append(finalPath,
ForwardVisited[recipe.Second]...)
            finalPath = append(finalPath, newBackwardPath1...)
            finalRoot := pathTotree(finalPath)
            return finalRoot, nodeCount
        }
    }
    return nil, nodeCount
}

```

4.2.9 Bidirectional (Multiple)

Fungsi ini melakukan seluruh proses *Bidirectional Searching* mode multiple

```

func BidirectionalMultiple(targetURL string, bound int)
(*data_type.RecipeTree, int) {
    targetIdx, ok := scrapping.MapperNameToIdx[targetURL]
    if !ok || targetIdx == -1 {
        fmt.Println("Error: Invalid target URL")
        return nil, 0
    }

    tier := scrapping.MapperIdxToTier[targetIdx]
    if tier == -1 || tier == 0 {
        return &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[targetIdx], Children: nil}, 1
    }

    countRecipe := 0
    var allRecipes = [] []Path_element{}
    ForwardQueue := [] []Path_element{}
    BackwardQueue := [] []Path_element{}
    ForwardVisited := make(map[int] []Path_element) // ForwardVisited[x]
menyimpan path yang menghasilkan x dari elemen dasar
(ForwardVisited[x] [].last() != x karena akan menghasilkan x)
    BackwardVisited := make(map[int] []Path_element) //
BackwardVisited[(x,y)] menyimpan path yang dimulai dari pair (x,y) hingga

```

```

menghasilkan targetIdx

recipeMap:= scrapping.MapperPairToIdxs
reverseMap := scrapping.MapperIdxToRecipes
nodeCount := 0

for i:=0; i<4; i++{
    if recipeMap[i] != nil {
        for key, val := range recipeMap[i] {
            result := val
            r1 := Path_element{Idx: i, Result: i}
            r2 := Path_element{Idx: key, Result: result}
            ForwardQueue = append(ForwardQueue,
                []Path_element{r1, r2})
            ForwardVisited[i] = []Path_element{r1}
            ForwardVisited[key] = []Path_element{r2}
            ForwardVisited[result] = []Path_element{r1, r2}
            nodeCount += 2
        }
    }
}

targetElmt := Path_element{Idx: targetIdx, Result: targetIdx}
BackwardQueue = append(BackwardQueue, []Path_element{targetElmt})
nodeCount += 1

for len(ForwardQueue) > 0 && len(BackwardQueue) > 0 {
    // Proses Forward
    ForwardPath := ForwardQueue[0]
    ForwardQueue = ForwardQueue[1:]
    fwdlast := ForwardPath[len(ForwardPath)-1]
    fwdlastRes := fwdlast.Result
    nextElmts := recipeMap[fwdlastRes]
    for key, res := range nextElmts {
        if _, visited := ForwardVisited[res]; visited {
            continue
        }
        nextPathElmt := Path_element{Idx: key, Result: res}
        tempForwardPath := append(ForwardPath, nextPathElmt)
        ForwardQueue = append(ForwardQueue, tempForwardPath)
        ForwardVisited[res] = tempForwardPath
        nodeCount ++
        if BackwardVisited[res] != nil {
            finalPath := []Path_element{}
            finalPath = append(finalPath, ForwardPath...)
            finalPath = append(finalPath,
                reversePath(BackwardVisited[res])...)
            allRecipes = append(allRecipes, finalPath)
            countRecipe ++
            if (countRecipe >= bound) {
                break
            }
        }
    }
}

```

```

        }
    }

    // Backward process
    BackwardPath := BackwardQueue[0]
    BackwardQueue = BackwardQueue[1:]
    bwlast := BackwardPath[len(BackwardPath)-1]
    bwlastIdx := bwlast.Idx
    recipes := reverseMap[bwlastIdx]
    for _, recipe := range recipes {
        if _, visited1 := BackwardVisited[recipe.First]; visited1 {
            continue
        }
        if _, visited2 := BackwardVisited[recipe.Second]; visited2 {
            continue
        }
        firstPathElmt := Path_element{Idx: recipe.First, Result:
bwlastIdx}
        secondPathElmt := Path_element{Idx: recipe.Second, Result:
bwlastIdx}
        firstPathElmtBasic := Path_element{Idx: recipe.First,
Result: recipe.First}
        secondPathElmtBasic := Path_element{Idx: recipe.Second,
Result: recipe.Second}
        newBackwardPath := BackwardPath[:len(BackwardPath)-1]
        newBackwardPath1 := append(newBackwardPath, firstPathElmt,
secondPathElmtBasic)
        newBackwardPath2 := append(newBackwardPath,
secondPathElmt, firstPathElmtBasic)
        BackwardQueue = append(BackwardQueue, newBackwardPath1)
        BackwardQueue = append(BackwardQueue, newBackwardPath2)
        BackwardVisited[recipe.Second] =
append(BackwardVisited[recipe.Second], newBackwardPath1...)
        BackwardVisited[recipe.First] =
append(BackwardVisited[recipe.First], newBackwardPath2...)
        nodeCount += 2
        if ForwardVisited[recipe.First] != nil {
            finalPath := []Path_element{}
            finalPath = append(finalPath,
ForwardVisited[recipe.First]...)
            finalPath = append(finalPath, newBackwardPath2...)
            allRecipes = append(allRecipes, finalPath)
            countRecipe ++
            if (countRecipe >= bound) {
                break
            }
        }
        if ForwardVisited[recipe.Second] != nil {
            finalPath := []Path_element{}
            finalPath = append(finalPath,
ForwardVisited[recipe.Second]...)
            finalPath = append(finalPath, newBackwardPath1...)
        }
    }
}

```

```

        allRecipes = append(allRecipes, finalPath)
        countRecipe ++
        if (countRecipe >= bound) {
            break
        }
    }
}

if (len(allRecipes) == 0) {
    return nil, 0
}
baseRoot := pathTotree(allRecipes[0])
for i := 1; i < len(allRecipes); i++ {
    combinePathToRoot(baseRoot, allRecipes[i], false)
}
baseRoot, newNodeCount := completeTheRoot(baseRoot)
nodeCount += newNodeCount
return baseRoot, nodeCount
}

```

4.2.10 Bidirectional (Pembantu)

Berikut adalah fungsi-fungsi yang membantu proses *bidirectional searching*

```

func combinePathToRoot(root *data_type.RecipeTree, recipe []Path_element,
done bool) {
    if (done) {
        return
    }
    if (root.Name != scrapping.MapperIdxToName[recipe[0].Idx]) {
        for _, child := range root.Children {
            combinePathToRoot(child.First, recipe, done)
        }
    } else {
        newRecipe := pathTotree(recipe)
        root.Children = append(root.Children, newRecipe.Children...)
    }
}
func BuildRecipeTree(node *data_type.RecipeTree, idx int, bestRecipes
map[int]*data_type.Recipe, visited map[int]bool) {
    if idx <= 4 || visited[idx] {
        return
    }

    visited[idx] = true
    recipe, found := bestRecipes[idx]
    if !found {
        return
    }
}

```

```

        firstIdx := recipe.First
        secondIdx := recipe.Second

        firstNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[firstIdx]}
        secondNode := &data_type.RecipeTree{Name:
scrapping.MapperIdxToName[secondIdx]}

        pair := &data_type.Pair_recipe{First: firstNode, Second: secondNode}
node.Children = append(node.Children, pair)

        if firstIdx > 4 {
            BuildRecipeTree(firstNode, firstIdx, bestRecipes, visited)
        }

        if secondIdx > 4 {
            BuildRecipeTree(secondNode, secondIdx, bestRecipes, visited)
        }
    }
}

```

4.3 Tata Cara Penggunaan

4.3.1 Penggunaan di Lokal

Untuk menjalankan *website* di lokal, hal yang perlu dipersiapkan adalah install go dan node. Setelah semua hal telah siap, *clone repository front end* dan *back end* yang terlampir di lampiran. Untuk menjalankan website, masuk ke direktori src kemudian jalankan command berikut

```

npm install
npm run dev

```

Setelah itu, jalankan server *backend* dengan menjalankan command

```

go run main.go

```

Setelah keduanya berjalan dengan baik, *website* siap untuk digunakan.

Langkah-langkah untuk mencari resep antara lain :

1. Masukkan *query* resep yang ingin dicari di *search bar*.
2. Pilih mode pencarian antara *single* atau *multiple*.
3. Pilih algoritma pencarian antara BFS atau DFS.
4. Setelah itu klik tombol *search*.
5. Setelah pencarian selesai, akan muncul hasil pohon resep beserta dengan detail informasi jumlah *node* yang dikunjungi serta lama durasi pencarian.

4.3.2 Penggunaan Dengan Docker

Untuk menjalankan *website* menggunakan docker, hal yang perlu dipersiapkan adalah unduh docker desktop. Setelah itu, lanjutkan dengan melakukan langkah-langkah berikut :

1. Letakkan folder *front end* dan *back end* di root yang sama.

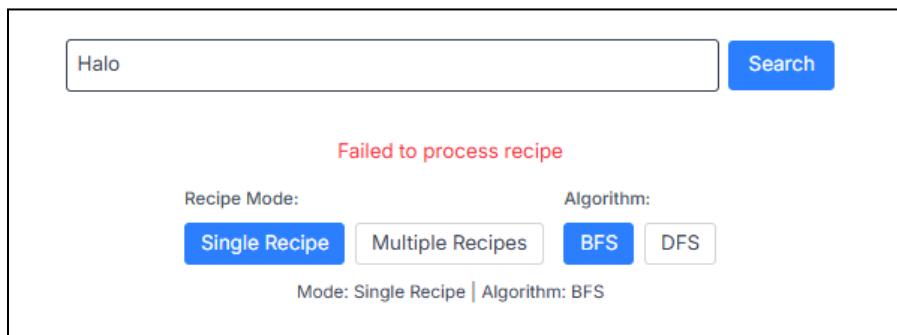
2. Pindahkan file docker-compose.txt ke root direktori lalu ubah menjadi docker.compose.yml
 3. Setelah itu jalankan docker dengan menjalankan *command* docker-compose-up
- Setelah docker berjalan, website dapat digunakan untuk melakukan pencarian resep. Langkah-langkah pencarian resep sama seperti yang telah dijelaskan di 4.3.1.

4.3.3 Penggunaan Melalui Website

Untuk mengakses website dapat diakses melalui tautan [Let Us Cook](#). Website tersebut sudah di-deploy dan dapat digunakan untuk mencari resep. Langkah-langkah pencarian resep sama seperti yang telah dijelaskan di 4.3.1.

4.4 Hasil Pengujian

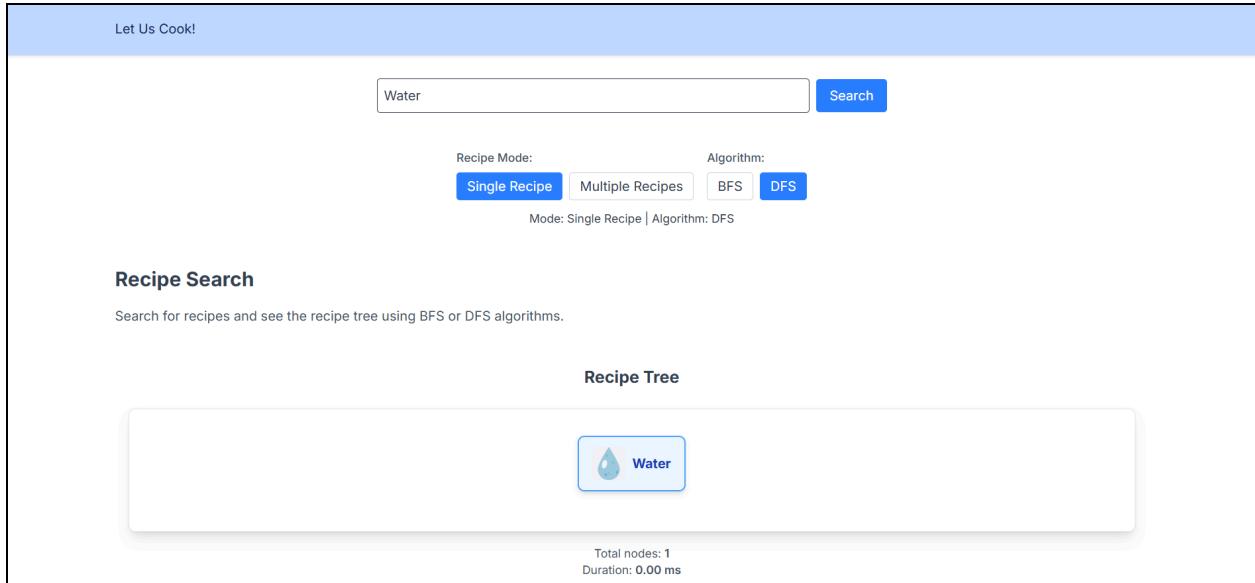
- a. Input Bukan merupakan elemen yang valid



Gambar 10. Kasus Input Tidak Valid

Tidak dapat memproses resep karena elemen yang dicari tidak valid.

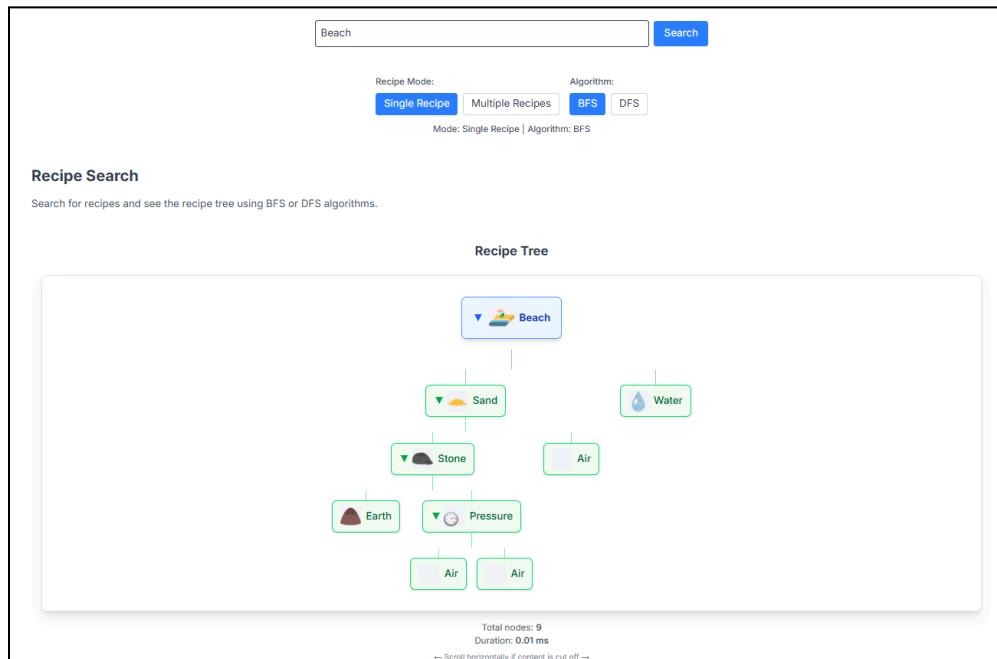
- b. Input adalah elemen dasar



Gambar 11. Input Elemen Dasar

Total node yang dikunjungi adalah 1 karena elemen dasar langsung didapatkan. Untuk durasinya sangat kecil sehingga di tampilan website tidak muncul presisinya.

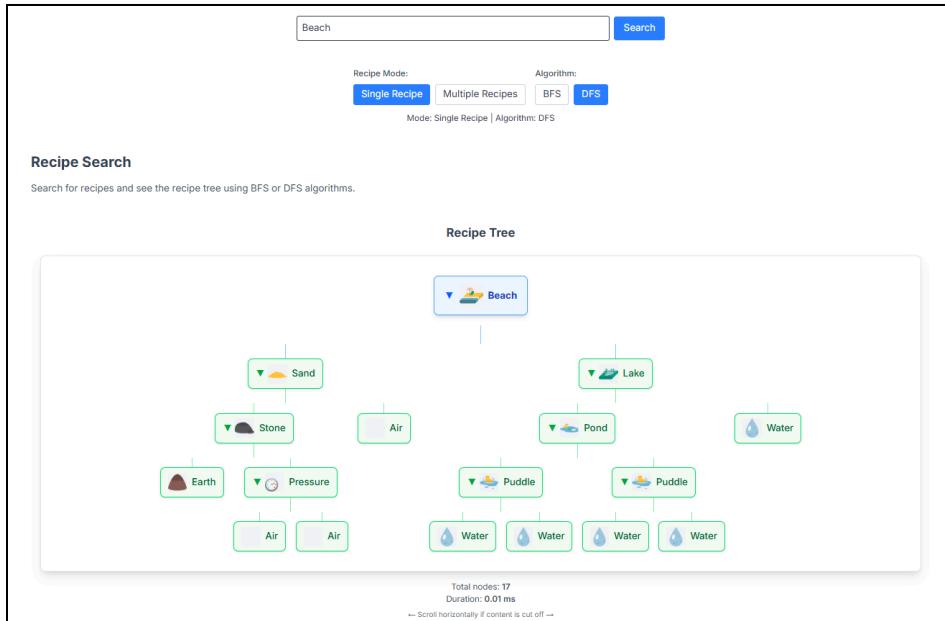
c. Test BFS mode single



Gambar 12. BFS single

Resep mencari beach dengan BFS single adalah sand + water, dengan total nodes yang dikunjungi 9 dan total durasi pencarian 0,01 ms.

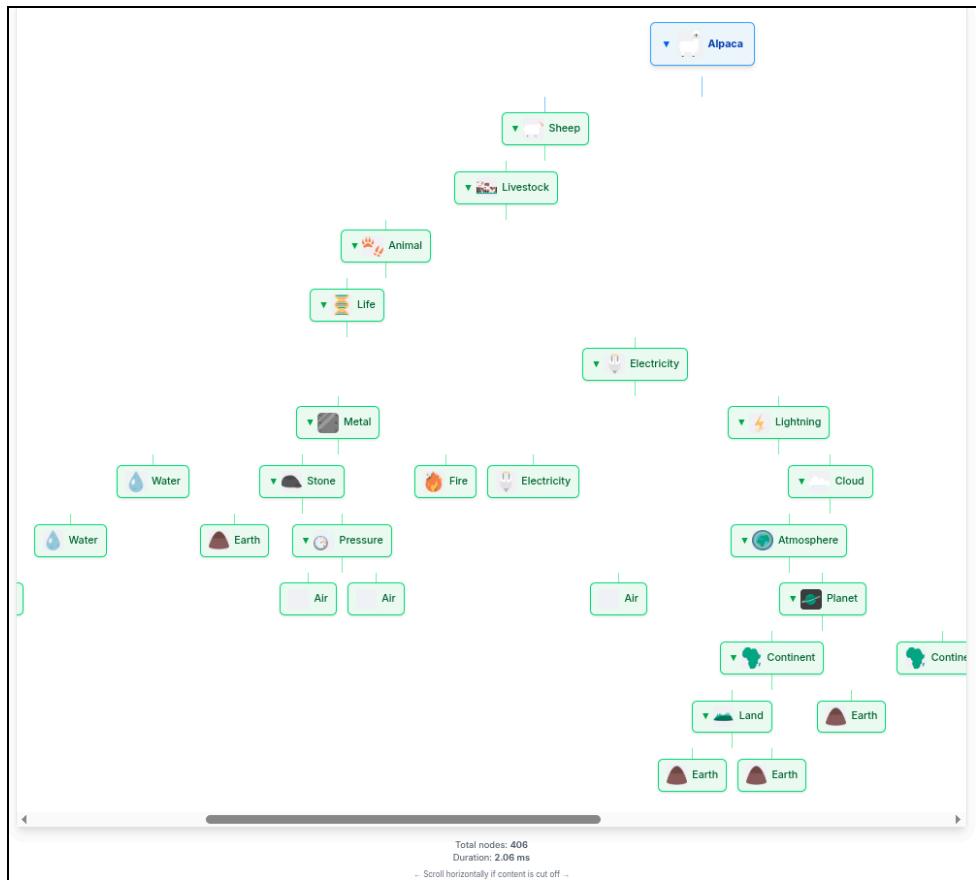
d. Test DFS mode single



Gambar 13. DFS single

Resep mencari beach dengan DFS single adalah sand + lake, dengan total nodes yang dikunjungi 17 dan total durasi 0,01 ms.

e. Test BFS mode multiple dengan bound = 3



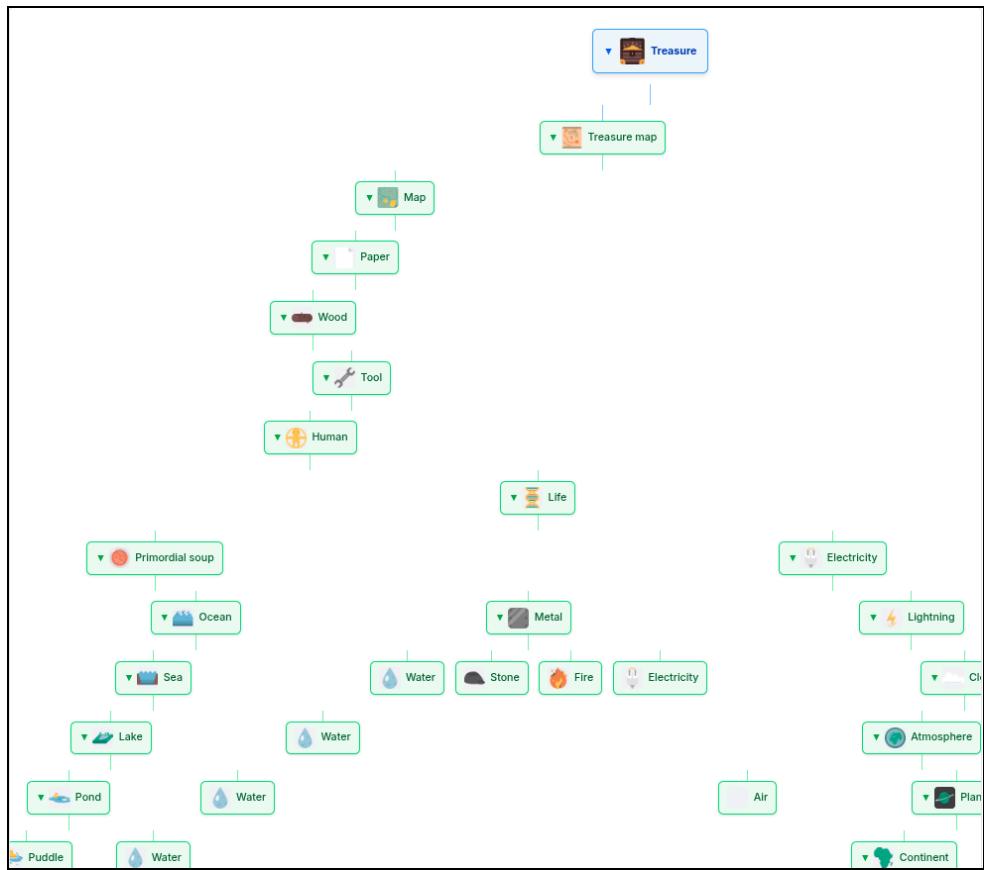
Gambar 14. Testing BFS mode multiple dengan bound = 3

Total nodes: 406

Duration: 2.06 ms

Resep mencari *Alpaca* dengan BFS multiple dengan bound = 3 menggunakan 406 nodes dan waktu runtime 2.06 ms. Hasil resep (dalam bentuk tree json) terlampir.

f. Test BFS mode multiple dengan bound = 15



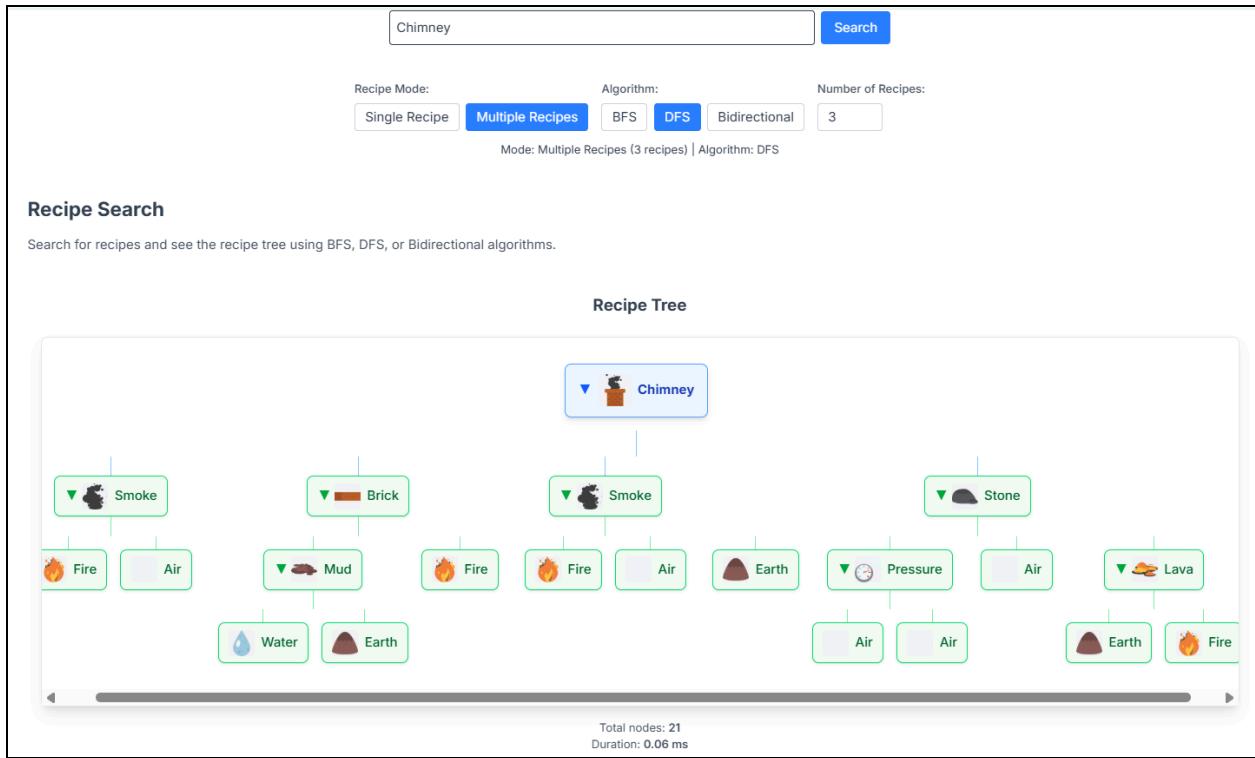
Gambar 15 Testing BFS mode multiple dengan bound = 15

Total nodes: 1040

Duration: 9.16 ms

Resep mencari *Treasure* dengan BFS multiple dengan bound = 15 menggunakan 1040 nodes dan waktu runtime 9.16 ms. Hasil resep (dalam bentuk *tree json*) terlampir.

g. Test DFS mode multiple dengan bound = 3



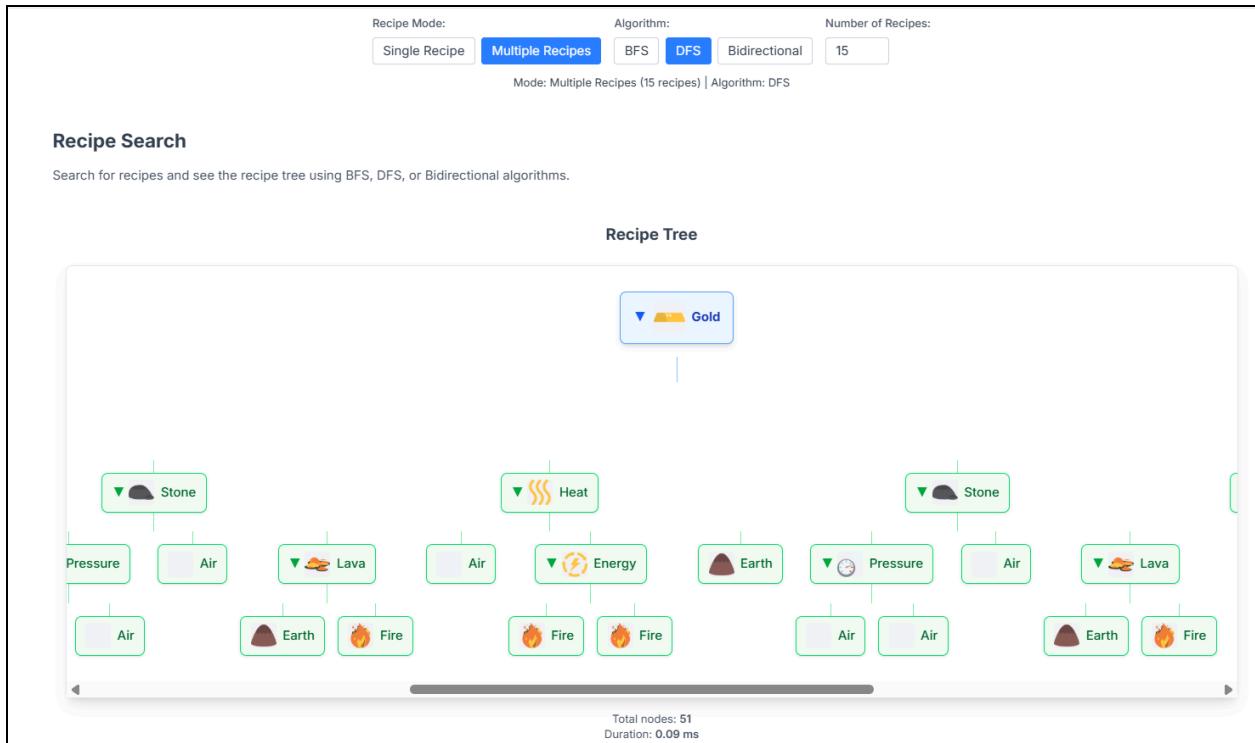
Gambar 16 Testing DFS mode multiple dengan bound = 3

Total nodes: 21

Duration: 0.06 ms

Resep mencari *Chimney* dengan DFS multiple dengan bound = 3 menggunakan 21 nodes dan waktu runtime 0.06 ms. Hasil resep (dalam bentuk tree json) terlampir.

h. Test DFS mode multiple dengan bound = 15



Gambar 17 Testing DFS mode multiple dengan bound = 15

Total nodes: 51

Duration: 0.09 ms

Resep mencari *Gold* dengan DFS multiple dengan bound = 15 menggunakan 51 nodes dan waktu runtime 0.109 ms. Hasil resep (dalam bentuk tree json) terlampir.

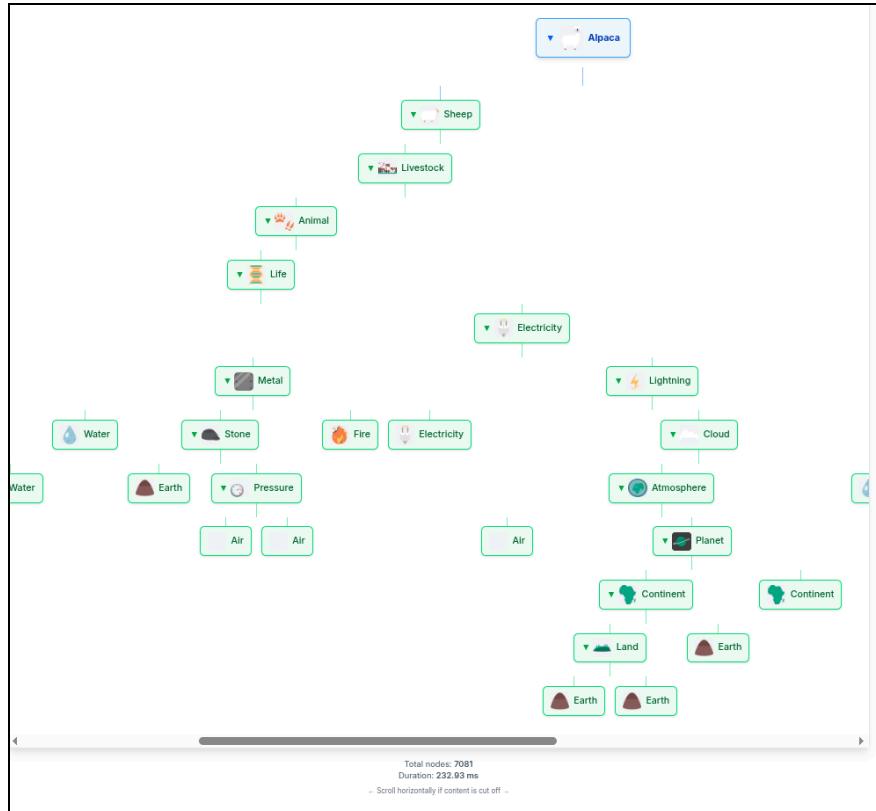
i. Test Bidirectional searching mode single



Gambar 18. Testing *Bidirectional Searching* dengan mode *single*

Resep mencari *Treasure* dengan *Bidirectional Searching single* menggunakan 337 nodes dan waktu runtime 0.80 ms. Hasil resep (dalam bentuk *tree json*) terlampir.

j. Test Bidirectional searching mode multiple dengan bound = 3



Gambar 19. Testing *Bidirectional Searching* dengan mode *multiple* dan bound = 3

Resep mencari *Alpaca* dengan *Bidirectional Searching multiple* menggunakan 7081 nodes dan waktu runtime 232.93 ms. Hasil resep (dalam bentuk *tree json*) terlampir.

4.5 Analisis Hasil Pengujian

a. Input Bukan merupakan elemen yang valid

Ketika pengguna memasukkan input elemen yang tidak valid, maka proses tidak dapat dilakukan karena seluruh resep yang ada merupakan resep yang berasal dari website little alchemy yang telah di scraping.

b. Input adalah elemen dasar

Ketika pengguna menginputkan resep yang ingin dicari adalah elemen dasar maka hasil *query* nya akan menampilkan satu elemen saja yaitu elemen itu sendiri. Jumlah node yang dikunjungi adalah satu karena resepnya adalah elemen itu sendiri.

c. Test BFS mode single

Untuk pencarian dengan algoritma BFS, terlihat bahwa hasil resepnya berbeda dengan hasil resep ketika menggunakan algoritma DFS. Terlihat bahwa resepnya lebih pendek dan jumlah nodes yang dikunjungi lebih sedikit, yaitu sembilan. Hal tersebut dikarenakan algoritma ini sifatnya menyebar sehingga dia akan mencoba kemungkinan jalur yang terhubung dengan node terakhirnya secara langsung. Hal ini membuat peluang mendapatkan resep terpendek menjadi lebih tinggi.

d. Test DFS mode single

Berbeda dengan pencarian menggunakan algoritma BFS, terlihat bahwa hasil resep yang ditampilkan lebih panjang dengan jumlah nodes yang dikunjungi sebanyak 17. Hal tersebut terjadi karena algoritma DFS melakukan pencarian secara mendalam ke salah satu cabang terlebih dahulu sampai mendapatkan resep yang sesuai. Ketika sampai ujung dan tidak mendapatkan resep yang dicari maka baru akan *backtrack* dan mencari ke cabang lainnya.

e. Test BFS mode multiple dengan bound = 3

Pada kasus ini, program berhasil mencari 3 alternatif resep yang dapat digunakan untuk mencari elemen *alpaca*. Alternatif solusinya diperoleh dari empat cara pembuatan metal yang berbeda. Program menemukan alternatif resep dalam durasi 2,06 ms dengan jumlah nodes yang dikunjungi adalah 406.

f. Test BFS mode multiple dengan bound = 15

Pada kasus ini, program berhasil mencari 16 alternatif resep yang dapat digunakan untuk membentuk elemen *treasure*. Perhatikan bahwa terdapat 16 cara untuk membuat *tool*, berasal dari 4 cara membuat *human* dan 4 cara membuat *metal*. Perhatikan bahwa selain elemen yang telah disebutkan, elemen lain pada *tree* hanya memiliki satu alternatif resep saja. Hal ini merupakan salah satu sifat dari pencarian multiple recipe dengan *multithreading*. Pemrosesan secara bersamaan memungkinkan perhitungan untuk sedikit melebihi batas yang dikehendaki pengguna.

g. Test DFS mode multiple dengan bound = 3

Pada kasus ini, program berhasil mencari 3 alternatif resep yang dapat digunakan untuk membentuk elemen *chimney*, yaitu *smoke + brick* (1 cara) dan *smoke + stone* (2 cara). Program pun dapat menemukan alternatif resep tersebut dengan waktu yang cukup cepat, yaitu 0.06 ms dan dengan total *node* yang dikunjungi sebanyak 21 *node*.

h. Test DFS mode multiple dengan bound = 15

Pada kasus ini, program berhasil mencari 16 alternatif resep yang dapat digunakan untuk membentuk elemen *gold*, yaitu *metal + sand* ($4 \times 4 = 16$ alternatif resep). Perhatikan bahwa nilai tersebut berlebih ketimbang masukan pengguna. Hal ini merupakan salah satu sifat dari

pencarian multiple recipe dengan *multithreading*. Karena pemrosesan *metal* dan *sand* dilakukan bersamaan, maka keduanya akan menemukan 4 resep untuk membentuk diri mereka masing-masing secara bersamaan pula. Meskipun terdapat validasi terhadap GlobalCounter, perhatikan bahwa tidak akan validasi yang melebihi angka lima belas sebelum ditemukan 16 resep. Angka terbesar yang memungkinkan adalah 8 (4 resep pada *metal* dan 2 resep pada *sand* atau sebaliknya). Program berhasil melakukan pencarian dengan waktu 0,09 ms dengan mengunjungi 51 *node*.

i. Test Bidirectional searching mode single

Pada kasus ini, program berhasil melakukan pencarian tepat satu resep terhadap elemen *treasure* dengan menggunakan algoritma bidirectional. Pencarian dilakukan dengan mengunjungi 303 *node* dengan waktu pencarian sebesar 0,57 ms. Perhatikan bahwa jumlah *node* yang cukup banyak ini melebihi bahkan jumlah *node* pada pencarian 15 resep pada uji kasus (h) untuk elemen yang sama. Hal ini merupakan sifat algoritma bidirectional yang juga melakukan pencarian, dan membangun pohon, dari elemen dasar. Sehingga, akan ada cukup banyak hasil dari pencarian yang tidak digunakan dan tidak pula ditampilkan pada pohon akhir yang terbentuk.

j. Test Bidirectional searching mode multiple dengan bound = 3

Pada kasus ini, program dapat melakukan pencarian alternatif resep untuk *alpaca* dengan jumlah nodes yang dikunjungi sebanyak 7081 dengan durasi pencarian 232,93 ms. Tiga alternatif resep tersebut didapatkan dengan empat alternatif resep pembuatan *metal* yang berbeda.

BAB V

KESIMPULAN

5.1 Kesimpulan

Dari tugas ini, kami berhasil menerapkan algoritma BFS (*Breadth First Search*) dan DFS (*Depth First Search*) dalam proses *searching recipes* pada permainan *Little Alchemy*. Selain itu, kami belajar banyak hal baru terkait *web development* dan *web scraping*. Pada kesempatan tugas ini kami menggunakan kakas *Gin Web FrameWork* sebagai kakas *BackEnd*, *NodeJS* sebagai kakas *FrontEnd* dan *GoQuery* sebagai kakas *web scraping*. Kami juga menggunakan Docker sebagai *platform* kontainer terisolasi untuk membangun aplikasi yang lebih portabel dan konsisten di berbagai lingkungan. Dalam penggeraan algoritma BFS dan DFS multiple recipe, kami menggunakan *GoConcurrency* sebagai alat untuk melakukan *multithreading* pada kode program, agar program dapat berjalan dengan lebih optimal.

5.2 Saran

Berikut merupakan beberapa saran pengembangan dari program yang telah dikembangkan:

1. Mengembangkan algoritma pencarian dengan heuristik tertentu ketimbang *blind search*
2. Mengembangkan skema *multithreading* atau pemrograman paralel yang lebih mumpuni, efektif, dan efisien
3. Mengembangkan fitur pencarian tanpa restriksi tier disertai *handling* terhadap pembentukan siklus pada tree yang sedang dibentuk

5.3 Refleksi

Setelah menyelesaikan tugas ini, ada beberapa hal yang dapat kami jadikan sebagai bahan refleksi untuk kedepannya. Hal pertama adalah sisihkan cukup waktu untuk menentukan konsep dasar dan struktur data yang ingin digunakan secara matang supaya tidak terjadi perubahan di tengah penggeraan. Kedua, buat metode yang modular supaya dapat digunakan kembali di fungsi lain sehingga kode bisa lebih bersih dan mudah dibaca. Pembagian kerja tugas juga sudah diusahakan seimbang satu sama lain. Komunikasi dalam penggeraan tugas ini juga sudah cukup baik dan kami dapat bekerja sama dengan baik satu sama lain.

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 *repository* :

Front End (https://github.com/DiyahSusan/TUBES2_FE_LetUsCook)

Back End (https://github.com/bill2247/TUBES2_BE_LetUsCook)

Tautan video : <https://youtu.be/fzz3jAhVNJQ?feature=shared>

Tautan website : <https://let-us-cook-new.vercel.app>

Tautan hasil testing :

<https://drive.google.com/drive/folders/1TmfU90LeqK93fXPKgRBMcPxoDle7fbp?usp=sharing>

DAFTAR PUSTAKA

Amazon Web Services. 2025. What is an API? diakses pada tanggal 11 Mei 2025 pada tautan <https://aws.amazon.com/what-is/api/>

Fandom. 2025. All elements in Little Alchemy 2 diakses pada tanggal 11 Mei 2025 pada tautan [https://little-alchemy.fandom.com/wiki/Elements_\(Little_Alchemy_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2))

Gin Web Framework. 2025. Gin diakses pada tanggal 11 Mei 2025 pada tautan <https://gin-gonic.com/>

Little Alchemy 2. 2025. Little Alchemy 2 diakses pada tanggal 11 Mei 2025 pada tautan <https://littlealchemy2.com/>

Munir, Rinaldi. 2025. *Breadth/Depth First Search* diakses pada tanggal 11 Mei 2025 pada tautan [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)

Mozilla Developer Network. 2025. Client-Server Architecture diakses pada tanggal 11 Mei 2025 pada tautan https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview

Node.js. 2025. Node.js Documentation diakses pada tanggal 11 Mei 2025 pada tautan <https://nodejs.org/docs/latest/api/>

PuerkitoBio. 2025. goquery diakses pada tanggal 11 Mei 2025 pada tautan <https://github.com/PuerkitoBio/goquery>

The Go Programming Language. 2025. Effective Go: Concurrency diakses pada tanggal 11 Mei 2025 pada tautan https://go.dev/doc/effective_go#concurrency

The Go Programming Language. 2025. Golang Documentation diakses pada tanggal 11 Mei 2025 pada tautan <https://go.dev/doc/>

Vercel. 2025. Next Documentation diakses pada tanggal 11 Mei 2025 pada tautan <https://nextjs.org/docs>