

LAPORAN TUGAS BESAR 2
IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Kelompok 46 - BrBaloni Lulilolli

Nama	NIM
Rafen Max Alessandro	13523031
Michael Alexander Angkawijaya	13523102
Aria Judhistira	13523112

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
BAB 1: Deskripsi Tugas.....	5
BAB 2: Landasan Teori.....	7
2.1 Dasar Teori.....	7
2.2 Program dan Website.....	11
BAB 3: Analisis Pemecahan Masalah.....	12
3.1. Langkah-langkah Pemecahan Masalah.....	12
3.2. Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma DFS dan BFS.....	13
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	16
3.4. Contoh Ilustrasi Kasus.....	19
BAB 4: Implementasi dan Pengujian.....	23
4.1. Spesifikasi Teknis Program.....	23
4.2. Penjelasan Tata Cara Penggunaan Program.....	28
4.3. Hasil Pengujian.....	30
4.4. Analisis Hasil Pengujian.....	36
BAB 5: Kesimpulan, Saran, dan Refleksi.....	38
5.1 Kesimpulan.....	38
5.2 Saran.....	38
5.3 Refleksi.....	39
Lampiran.....	40
1. Tautan Repository.....	40
2. Tautan Aplikasi Web.....	40
3. Tautan Video.....	40
4. Tabel Checklist.....	40
Daftar Pustaka.....	42

DAFTAR GAMBAR

Gambar 1. Little Alchemy 2 (sumber: https://www.thegamer.com).....	5
Gambar 2. Elemen dasar pada Little Alchemy 2.....	6
Gambar 3. Algoritma IDS sebagai uninformed search.....	7
Gambar 4. Algoritma A* sebagai informed search.....	8
Gambar 5. Cara kerja algoritma BFS.....	9
Gambar 6. Cara kerja algoritma DFS.....	10
Gambar 7. Contoh hasil representasi graf menggunakan struktur data GraphNode dan GraphEdge....	15
Gambar 8. Graf pemetaan kombinasi pembentuk elemen sand.....	21
Gambar 9. Graf pemetaan kombinasi valid pembentuk elemen sand.....	21
Gambar 10. Graf hasil pencarian DFS untuk elemen sand.....	22
Gambar 11. Graf hasil pencarian BFS untuk elemen sand.....	23
Gambar 12. Tampilan laman Home aplikasi web.....	29
Gambar 13. Tampilan laman Main aplikasi web.....	30
Gambar 14. Pemilihan elemen pada laman Main.....	31
Gambar 15. Tampilan laman Main ketika mencari resep Sand.....	31
Gambar 16. Tampilan hasil pencarian DFS satu resep elemen aquarium.....	32
Gambar 17. Tampilan hasil pencarian DFS empat resep elemen aquarium.....	32
Gambar 18. Tampilan hasil pencarian BFS elemen aquarium.....	33
Gambar 19. Tampilan hasil pencarian BFS elemen grass.....	33
Gambar 20. Tampilan hasil pencarian DFS elemen grass.....	33
Gambar 21. Tampilan hasil pencarian DFS tiga resep elemen grass.....	34
Gambar 22. Tampilan hasil pencarian BFS elemen mountain grass.....	34
Gambar 23. Tampilan hasil pencarian DFS elemen mountain grass.....	35
Gambar 24. Tampilan hasil pencarian BFS elemen eruption dua resep.....	35
Gambar 25. Tampilan hasil pencarian DFS elemen eruption dua resep.....	36
Gambar 26. Error handler menangani masukan elemen tidak dikenali.....	36
Gambar 27. Error handler menangani tidak memberikan masukan elemen.....	37
Gambar 28. Error handler menangani tidak memberikan masukan algoritma.....	37

DAFTAR TABEL

Tabel 1. Tabel Checklist.....	41
-------------------------------	----

BAB 1: Deskripsi Tugas



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

Little Alchemy 2 merupakan permainan berbasis website / aplikasi yang dikembangkan oleh Recloak dan dirilis pada tahun 2017. Permainan ini bertujuan untuk membentuk 720 elemen dari 4 elemen dasar yang tersedia, yaitu *air*, *earth*, *fire*, dan *water*. Permainan ini merupakan sekuel dari permainan sebelumnya yakni Little Alchemy 1 yang dirilis tahun 2010.

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

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

Komponen-komponen dari permainan ini antara lain:

1. Elemen dasar

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



Gambar 2. Elemen dasar pada Little Alchemy 2

2. Elemen turunan

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

3. *Combine Mechanism*

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

BAB 2: Landasan Teori

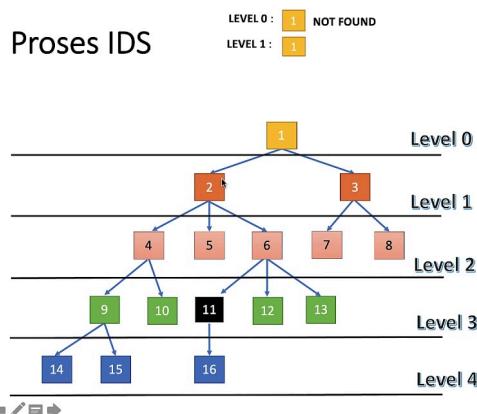
2.1 Dasar Teori

2.1.1 Algoritma Penjelajahan Graf

Graf adalah suatu jenis struktur data yang terdiri dari sekumpulan simpul (*node*) yang merepresentasikan sebuah nilai dan sisi (*edge*) yang menghubungkan simpul-simpul tersebut. Sisi sendiri dapat memiliki nilai yang menyatakan bobot dan bisa berupa berarah atau tidak berarah. Jenis-jenis graf dapat dikategorikan berdasarkan arah sisinya sebagai graf tak berarah dan graf berarah, berdasarkan bobot sisinya sebagai graf berbobot dan graf tidak berbobot, atau berdasarkan struktur khusus, seperti graf melingkar, graf bipartit, dan sebagainya.

Algoritma penjelajahan graf merupakan suatu algoritma yang mengunjungi simpul-simpul pada graf secara sistematis. Algoritma ini bekerja dengan asumsi bahwa setiap simpul terhubung. Tujuan utama dari algoritma ini adalah untuk memecahkan sebuah persoalan yang solusinya dapat ditemukan pada graf. Terdapat dua jenis utama algoritma pencarian solusi pada graf, yakni pencarian solusi tanpa informasi dan pencarian solusi dengan informasi.

- Algoritma penjelajahan graf tanpa informasi

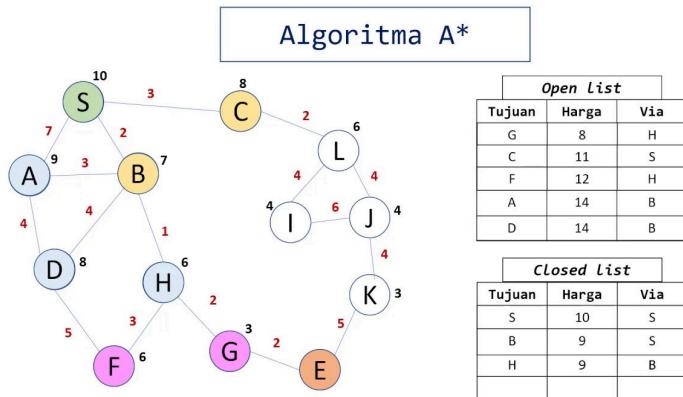


Gambar 3. Algoritma IDS sebagai *uninformed search*

Algoritma *uninformed search* tidak memberikan informasi tambahan selain informasi yang terkandung pada graf itu sendiri. Dengan demikian, algoritma dilakukan dengan mengiterasi kemungkinan jalur graf (*exhaustive*

search) berdasarkan prosedur tertentu untuk mencari posisi dari kombinasi jalur graf yang diinginkan. Beberapa strategi algoritma yang merupakan *uninformed search* adalah *Depth First Search* (DFS), *Breadth First Search* (BFS), *Depth Limited Search* (DLS), *Iterative Deepening Search* (IDS), dan *Uniform Cost Search*.

- b. Algoritma penjelajahan graf dengan informasi

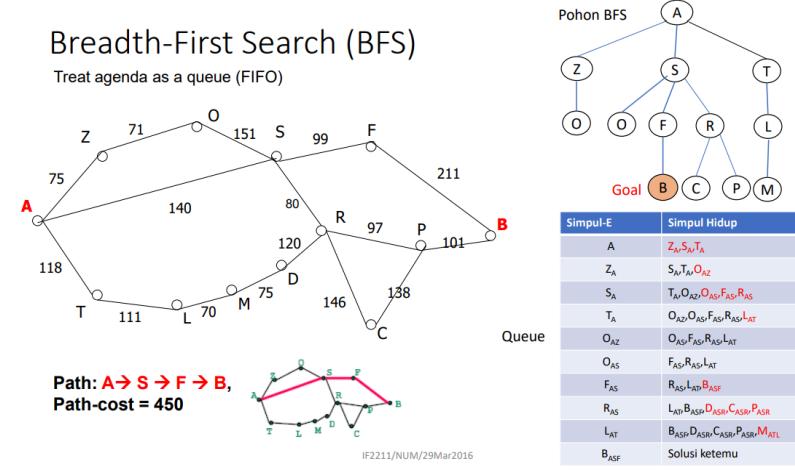


Gambar 4. Algoritma A* sebagai *informed search*

Algoritma *informed search* melakukan pencarian suatu tujuan tertentu menggunakan basis heuristik dan *non-goal state* yang memengaruhi cara kerja algoritma. Beberapa strategi algoritma yang merupakan *informed search* adalah *Best First Search* dan A^* .

2.1.2 *Breadth First Search (BFS)*

Algoritma *breadth first search* (BFS) merupakan suatu algoritma traversal graf tanpa informasi yang mencari solusi secara melebar. Pada algoritma BFS, penjelajahan simpul-simpul graf mengutamakan simpul yang bertetanggaan atau terhubung dengan simpul awal. Setelah mengunjungi semua simpul yang terdekat, pencarian dilanjutkan pada simpul berikutnya yang terhubung dengan salah satu simpul yang sudah dikunjungi, hingga solusi optimal ditemukan atau setiap simpul telah terkunjung.



Gambar 5. Cara kerja algoritma BFS

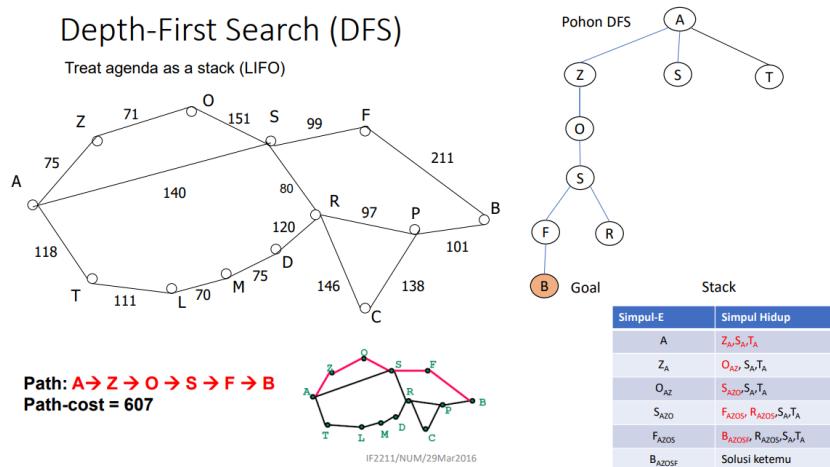
Cara kerja dari algoritma BFS adalah sebagai berikut:

1. Dibentuk sebuah *queue* yang menerima simpul sebagai elemennya;
2. Masukkan akar sebagai simpul awal ke dalam *queue*;
3. Lakukan *dequeue*, jika *head* dari *queue* adalah solusi, kembalikan solusi (dapat berupa simpul tersebut, atau urutan bagaimana simpul didapatkan);
4. Jika *head* dari *queue* bukan sebuah solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut ke dalam *queue*;
5. Jika didapati bahwa *queue* kosong, maka pencarian dinyatakan tidak menghasilkan solusi;
6. Ulangi langkah untuk elemen selanjutnya dari *queue* dari langkah kedua.

Algoritma BFS menjamin solusi persoalan dapat ditemukan selama nilai *branching factor* atau jumlah maksimum percabangan setiap simpul terbatas. Meskipun BFS memakan ruang yang banyak pada memori dengan kompleksitas ruang yang tinggi, algoritma ini memungkinkan pencarian solusi optimal pada graf berbobot, seperti pencarian rute terpendek, dan memiliki kompleksitas waktu yang dapat lebih rendah dibandingkan dengan algoritma *depth first search* (DFS).

2.1.3 Depth First Search (DFS)

Algoritma *depth first search* (DFS) merupakan suatu algoritma traversal graf tanpa informasi yang mencari solusi secara mendalam. Algoritma ini menggunakan pendekatan rekursi untuk mencari solusi pada simpul yang bertetanggaan dengan simpul awal. Artinya, penjelajahan dilakukan pada suatu simpul yang terhubung dengan simpul awal, kemudian diteruskan pada simpul tersebut dengan simpul yang terhubung dengannya. Ketika pencarian mencapai simpul yang bertetanggaan dengan simpul yang sebelumnya telah dikunjungi, maka algoritma akan melakukan *backtracking* kepada simpul sebelumnya untuk mencari dan menjelajahi simpul lain yang belum pernah dikunjungi. Pencarian akan berakhir ketika mencapai solusi atau ketika tidak ada simpul yang belum pernah dikunjungi lagi.



Gambar 6. Cara kerja algoritma DFS

Cara kerja dari algoritma DFS adalah sebagai berikut:

1. Dibentuk sebuah *stack* yang menerima simpul sebagai elemennya;
2. Masukkan akar sebagai simpul awal ke dalam *stack*;
3. Lakukan *pop*, jika *head* dari *stack* adalah solusi, kembalikan solusi (dapat berupa simpul tersebut, atau urutan bagaimana simpul didapatkan);
4. Jika *head* dari *queue* bukan sebuah solusi, *push* seluruh simpul yang bertetangga dengan simpul tersebut ke dalam *queue*;

5. Jika didapati bahwa *stack* kosong, maka pencarian dinyatakan tidak menghasilkan solusi;
6. Ulangi langkah untuk elemen selanjutnya dari *stack* dari langkah kedua.

Selama *branching factor* terbatas dan terdapat penanganan untuk jalur yang berulang-ulang, algoritma DFS menjamin bahwa solusi dapat ditemukan. Namun, DFS tidak menjamin solusi yang optimal dan dapat berjalan lebih lama dibandingkan BFS dengan kompleksitas waktu yang lebih tinggi, meskipun algoritma ini lebih baik secara kompleksitas ruang.

2.2 Program dan Website

Program *website* untuk tugas besar ini terdiri atas *front end* dan *back end* yang dibuat terpisah agar penggerjaan keduanya dapat diparalelkan dalam satu waktu bersamaan, dan diintegrasikan ketika keduanya telah dipastikan berjalan dengan baik.

1. Back End

Program untuk tugas besar ini menggunakan bahasa pemrograman Golang dalam membentuk *backend*. *Backend* bertanggung jawab dalam melakukan *web scraping* pada website Fandom Little Alchemy 2 yang memiliki informasi mengenai setiap resep menuju suatu elemen. Informasi mengenai resep elemen kemudian dimanfaatkan untuk membentuk struktur *tree* yang menunjukkan rute elemen-elemen yang dilewati menuju elemen yang diinginkan, dengan pembentukan pohon dan pencarian elemen dilakukan menggunakan algoritma BFS, DFS, dan *bidirectional search*.

2. Front End

Antarmuka program bertanggung jawab dalam menampilkan pohon solusi memanfaatkan *framework* Next.js dengan Javascript sebagai bahasa pemrograman utamanya. Konten pada website terdiri dari komponen-komponen React dan menggunakan HTML untuk menyusun konten dan CSS untuk *styling* dan aksesoris website.

BAB 3: Analisis Pemecahan Masalah

3.1. Langkah-langkah Pemecahan Masalah

Setelah disederhanakan, permasalahan pada tugas besar kali ini adalah mencari rute dari suatu elemen masukan pengguna menuju elemen-elemen penyusunnya, baik elemen dasar atau elemen turunan. Jika elemen tersebut merupakan elemen turunan, maka pencarian rute akan terus dilakukan hingga ditemukan kombinasi yang dibentuk hanya oleh elemen-elemen dasar. Dengan demikian, pemecahan masalah dimulai dengan melakukan *scraping website* untuk mengambil data elemen dan mengubahnya menjadi struktur data yang dapat diproses oleh program, lalu menerima masukan dari pengguna, dan menggunakan algoritma BFS maupun DFS untuk melakukan algoritma pencarian graf yang telah dibentuk, untuk mengembalikan hasil pencarian kepada pengguna.

Langkah pertama yang krusial bagi program adalah mengumpulkan data elemen melalui *scraping website* [Little Alchemy 2 Wiki](#). *Scraping* dilakukan dengan kakas *goquery* dan menghasilkan struktur data Element dan Combination. Struktur data Element disusun atas Name (string), CanCreate (array of string), Combinations(array of pointer to Combination), dan Tier (int), sedangkan struktur data Combination disusun atas ResultName (string), LeftName (string), dan RightName (string). Kedua struktur data ini berperan dalam memetakan data elemen dan kombinasi yang memungkinkan ke dalam struktur data yang dapat diproses oleh program.

Lalu, pengguna tidak hanya memberikan masukan elemen yang dibutuhkan, tetapi juga konfigurasi-konfigurasi yang dibutuhkan oleh program. Setelah pengguna memasukkan masukan berupa elemen target, dibutuhkan konfigurasi metode algoritma yang digunakan dalam mencari resep dan jumlah resep yang diinginkan untuk ditemukan. Masukan-masukan tersebut kemudian digunakan sebagai parameter dalam pembentukan graf dan pemanggilan fungsi dalam program, meliputi fungsi pencarian graf dengan implementasi algoritma BFS atau DFS

BFS diimplementasikan menggunakan metode iteratif sedangkan DFS menggunakan metode rekursif. Dengan demikian, BFS akan melakukan pencarian per level sebanyak jumlah kombinasi yang dimiliki oleh elemen target, dan DFS akan melakukan pencarian per kombinasi yang memungkinkan secara mendalam. Pencarian oleh kedua algoritma bertujuan untuk menemukan jalur kombinasi yang dapat dinyatakan hanya menggunakan elemen dasar.

Program kemudian direalisasikan ke dalam bentuk *website* dengan *frontend* menggunakan *framework* Next.js menggunakan bahasa Javascript dan Typescript serta *backend* menggunakan bahasa pemrograman Go. Setelah *website* dapat dijalankan, program menjalankan keseluruhan solusi dari permasalahan pencarian elemen dalam permainan Little Alchemy 2 dengan *interface* sederhana, interaktif, dan informatif. *Interface website* menampilkan graf yang memetakan jalur kombinasi dalam membentuk elemen masukan dari pengguna, banyaknya simpul graf yang diteliti dalam melakukan pencarian, dan lamanya pencarian dilakukan.

3.2. Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma DFS dan BFS

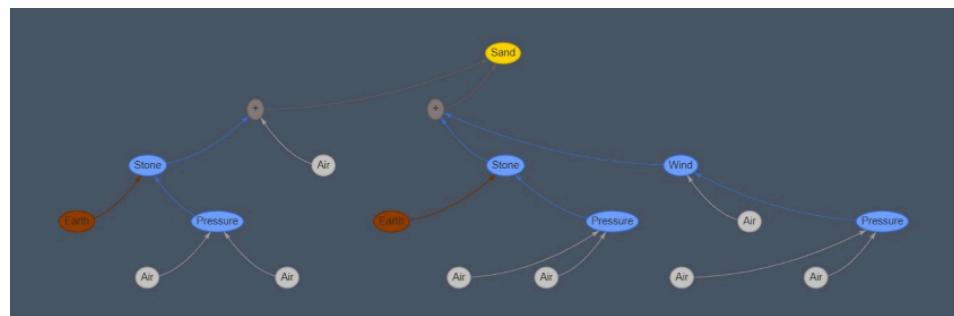
Agar persoalan dapat diselesaikan dengan algoritma DFS dan BFS, diperlukan proses pemetaan komponen persoalan menjadi graf agar dapat diproses ke dalam algoritma DFS dan BFS. Kelompok setuju untuk melakukannya ke dalam dua tahap, yang dapat dilakukan secara bersamaan (seperti pada DFS) atau tidak (seperti pada BFS).

1. Melakukan pencarian jalur kombinasi menggunakan graf masukan

Data hasil *website scraping* yang diproses ke dalam struktur data Element dan Combination digunakan dalam proses melakukan pencarian jalur kombinasi. Tahapan ini dilakukan dengan meneliti *list of Combination* yang mungkin dari elemen target sebagai tetangga dari sebuah Element, menelusuri *list of Combination* tersebut (menggunakan algoritma BFS dan DFS), dan berhenti ketika didapat jalur kombinasi yang dapat sepenuhnya dibentuk hanya menggunakan elemen dasar.

2. Membentuk *node* dan *edge* untuk graf keluaran

Setelah didapatkan jalur kombinasi yang sepenuhnya dibentuk hanya menggunakan elemen dasar, dilakukan perubahan jalur kombinasi tersebut menjadi *node* dan *edge* dalam satu graf baru menggunakan struktur data *GraphNode* dan *GraphEdge*. *GraphNode* bertanggung jawab untuk membentuk simpul dengan ID unik dan nama yang merepresentasikan elemen, sedangkan *GraphEdge* bertanggung jawab untuk menyatakan sisi tetangga antara dua *GraphEdge*. Kelompok juga menyetujui untuk membedakan struktur graf dalam merepresentasikan elemen yang hanya memiliki satu kombinasi, dan elemen yang memiliki lebih dari satu kombinasi sebagai berikut.



Gambar 7. Contoh hasil representasi graf menggunakan struktur data *GraphNode* dan *GraphEdge*

Elemen dengan satu kombinasi langsung bertetangga dengan kedua elemen yang membentuknya, seperti elemen *stone* yang bertetangga langsung dengan *earth* dan *pressure*. Sedangkan elemen dengan lebih dari satu kombinasi, seperti elemen *sand* yang dapat dibentuk oleh *stone* dan *air* atau *stone* dan *wind*, bertetangga dengan *GraphNode* bernama “+” sejumlah jumlah kombinasi yang dimiliki. Hal ini ditujukan untuk meningkatkan *readability* dalam menafsirkan maksud dari graf.

Menggunakan tahapan tersebut, rancangan algoritma DFS dan BFS untuk menyelesaikan permasalahan dibentuk sebagai berikut.

a. Algoritma DFS

Algoritma DFS menggunakan pendekatan rekursivitas dalam melakukan pencarian kombinasi elemen. Elemen masukan dari pengguna akan dimasukkan sebagai parameter dari fungsi, yang kemudian memanggil fungsi yang sama untuk elemen komponen dari setiap kombinasi elemen sebelumnya. Fungsi akan mengembalikan nilai berupa urutan jalur kombinasi yang dapat membentuk elemen parameter fungsi dari elemen dasar jika didapati kombinasi yang tepat, atau nilai *nil* jika tidak didapati kombinasi tersebut. Hal ini mungkin terjadi menyesuaikan spesifikasi spek yang menyatakan bahwa setiap elemen hanya dapat dibentuk oleh elemen dengan Tier lebih rendah, dan terdapat elemen pada permainan Little Alchemy 2 yang dibentuk hanya menggunakan elemen dengan Tier yang setara.

Dengan demikian, secara umum, algoritma DFS akan melakukan pencarian secara mendalam untuk salah satu kombinasi yang mungkin dari elemen target. Walaupun proses ini tangkis dalam konteks penggunaan ruang, proses ini dapat memberikan penggunaan waktu yang jauh lebih tidak efektif karena hanya terpaku pada satu Combination.

b. Algoritma BFS

Algoritma BFS menggunakan pendekatan iteratif dalam melakukan pencarian kombinasi elemen. Elemen masukan dari pengguna akan dimasukkan sebagai parameter dari fungsi, yang kemudian diolah sebagai *queue* of Combination berdasarkan *list* of Combination dari elemen tersebut. Algoritma BFS kemudian melakukan langkah berikut.

1. *Dequeue* Combination pertama dalam *queue*, lakukan pengecekan apakah setiap Element yang berperan dalam Combination tersebut merupakan elemen dasar atau tidak;
2. Jika iya, maka kembalikan Combination tersebut sebagai jalur;
3. Jika tidak, maka lakukan *expand* pada Combination tersebut dengan menghilangkan elemen dasar dengan mengganti elemen turunan dengan elemen komponen penyusunnya, untuk setiap kombinasi yang

dimiliki oleh elemen turunan tersebut. Hal ini sejalan dengan konsep ‘memasukkan tetangga dari simpul dalam’ yang dimiliki oleh BFS;

4. *Enqueue* Combination hasil *expand* ke dalam *queue*;
5. Ulangi langkah langkah pertama hingga ditemukan Combination yang memenuhi langkah kedua.

Berbeda dengan DFS, jika elemen masukan memiliki kombinasi dengan tahapan yang lebih sedikit dibandingkan kombinasi lainnya, maka BFS dapat dipastikan akan mengembalikan jalur kombinasi tersebut. Namun, BFS memiliki kelemahan akan kebutuhan ruang yang jauh lebih berat dibandingkan DFS karena melakukan pencarian secara melebar. Kebutuhan ruang untuk BFS dapat berkembang secara eksponensial menyesuaikan jumlah *branching factor* yang dimiliki pada setiap kombinasi awal dan turunan.

3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

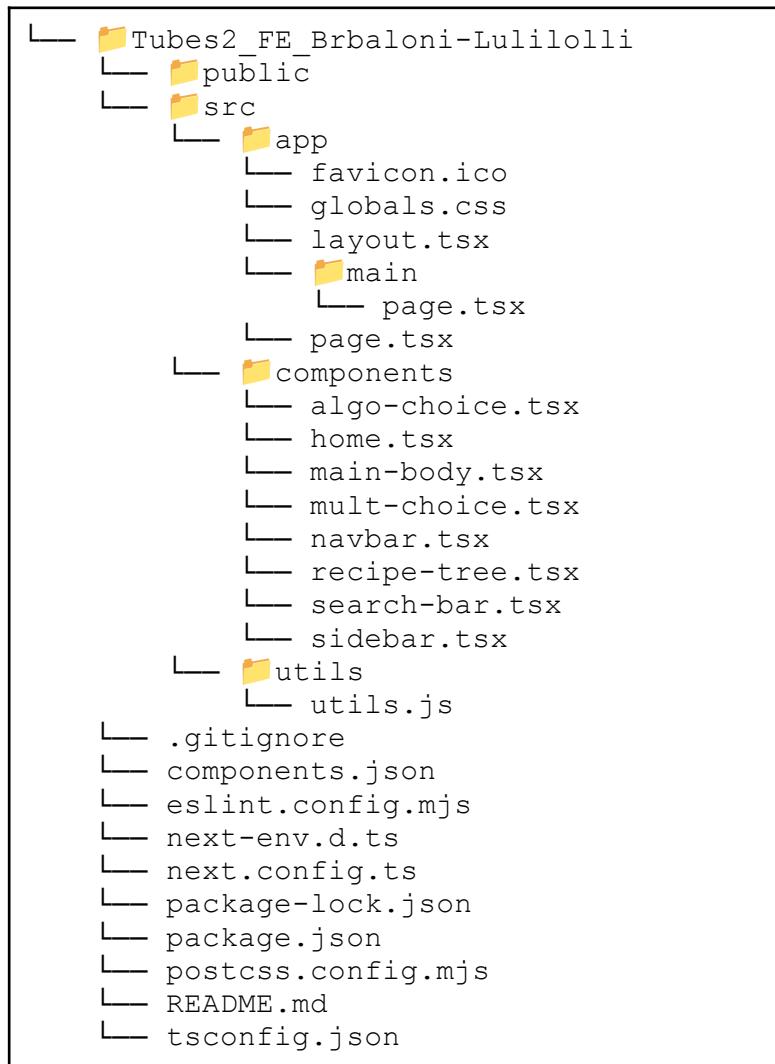
Aplikasi web dibangun dengan fitur fungsional utama berupa pencarian resep menuju suatu elemen. Pencarian resep tersebut dapat memanfaatkan algoritma BFS atau DFS. Selain itu, aplikasi web juga menyediakan pilihan untuk mencari jalur resep menuju suatu elemen yang banyak (lebih dari 1).

Pengembangan aplikasi web dibagi menjadi dua bagian, yaitu bagian *frontend* dan bagian *backend*, yang masing-masing dikembangkan pada repositori yang berbeda. Bagian *frontend* aplikasi web berfungsi sebagai antarmuka program dengan pengguna. Bagian ini pun berfungsi sebagai penghubung yang menjembatani *input* dari pengguna kepada program dan menampilkan hasilnya kembali kepada pengguna. Di sisi lainnya, bagian *backend* berfungsi untuk menjalankan program sesungguhnya, yaitu algoritma pencarian resep suatu elemen.

Berikut adalah uraian arsitektur aplikasi web yang dibagi menjadi bagian *frontend* dan *backend*:

- a. *Frontend*

Bagian *frontend* aplikasi web memanfaatkan *framework* Next.js sebagai *framework* utama. Pada *framework* ini, bahasa pemrograman yang digunakan untuk konfigurasi halaman dan integrasi dengan *backend* adalah Typescript dan Javascript, serta Tailwind CSS untuk melakukan *styling* UI aplikasi web.



Terdapat dua halaman pada aplikasi web, yakni laman *Home* dan laman *Main*. Laman *Home* berfungsi sebagai *landing page* aplikasi web yang berisi petunjuk penggunaan program pada aplikasi web. Pengguna dapat navigasi dan berpindah antara laman dengan memanfaatkan *navigation bar* yang

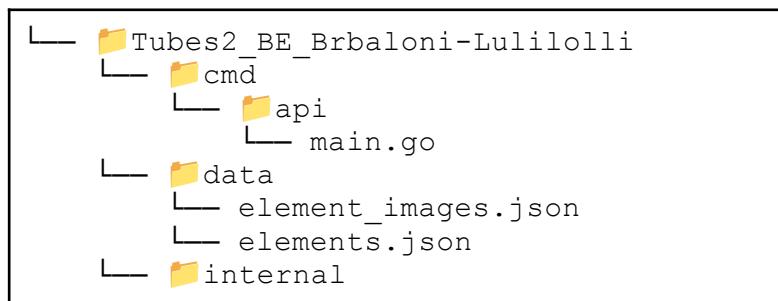
tersedia di setiap halaman. Komponen-komponen laman, seperti *sidebar*, *navbar*, dan lainnya disimpan pada folder *src/components*.

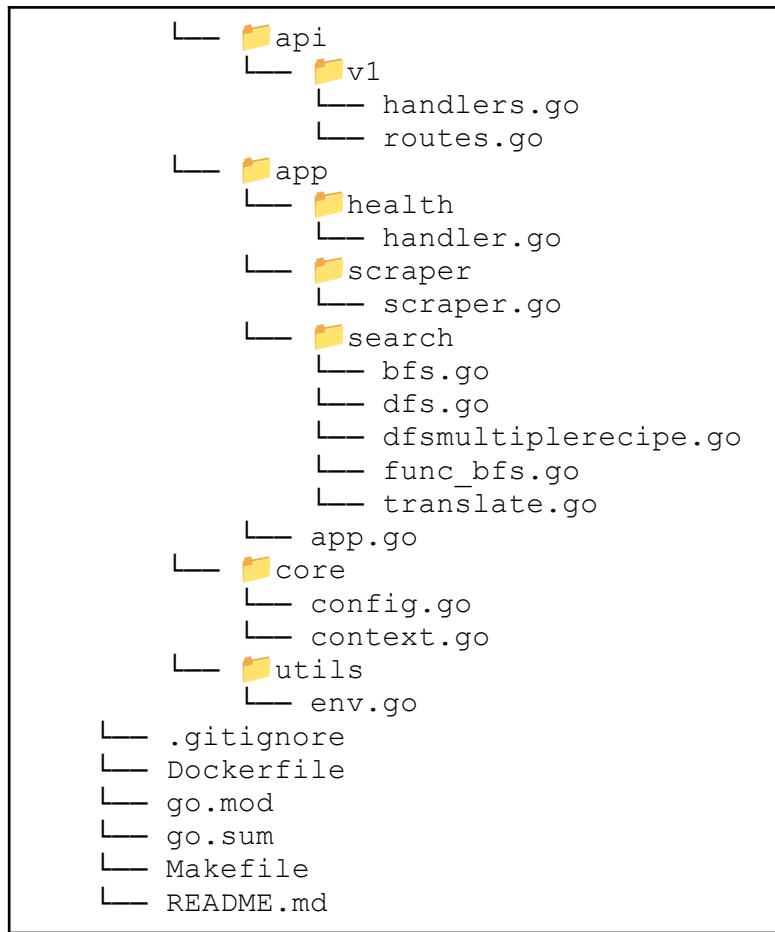
Lama *Main* bertujuan sebagai tempat menjalankan fungsi aplikasi web, yakni pencarian resep menuju suatu elemen. Pengguna dapat memasukkan semua data *input* untuk melakukan pencarian resep. Jika pencarian berhasil, maka program akan menampilkan hasil pohon resep pada blok berwarna abu-abu di tengah laman beserta waktu pencarinya dan jumlah *node* pada pohon.

Integrasi algoritma *backend* dengan antarmuka *frontend* dilakukan dengan bantuan API. Bagian *frontend* akan mengambil data pohon melalui API *endpoint* yang ditentukan berdasarkan data *input* pengguna. Lalu, bagian *frontend* akan mengolah data tersebut untuk menampilkan pohon menggunakan perpustakaan Vis.js Network. Bagian *frontend* juga mengambil data semua nama elemen beserta gambar elemen tersebut melalui API *endpoint* yang telah ditentukan oleh bagian *backend* untuk mempermudah pencarian.

b. Backend

Bagian backend aplikasi web ini dibangun menggunakan bahasa pemrograman Go (Golang) dan memanfaatkan framework Gin sebagai HTTP web framework utama untuk pembuatan REST API. Backend bertanggung jawab atas seluruh logika utama aplikasi, termasuk proses scraping data, pencarian jalur pembuatan elemen menggunakan algoritma pencarian, serta penyediaan API untuk diakses oleh frontend.



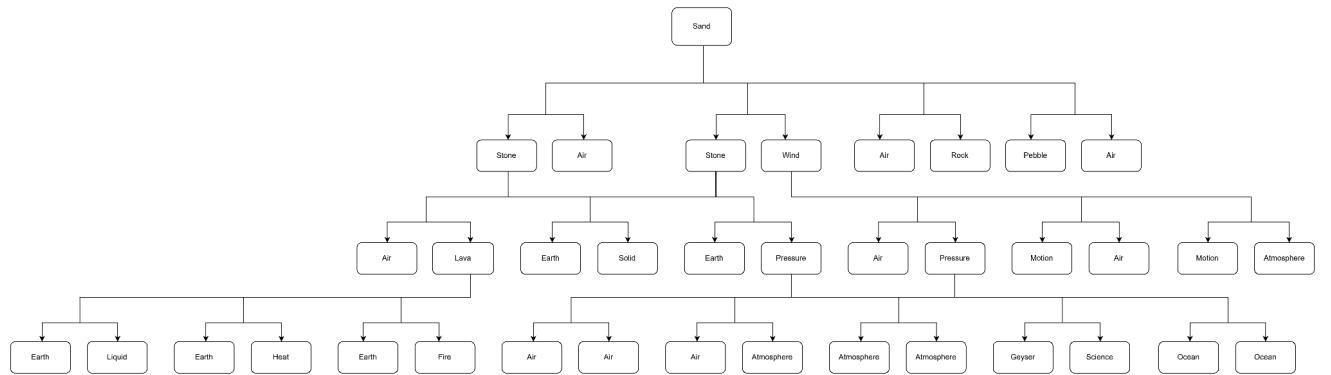


Aplikasi backend menyajikan tiga API endpoint:

1. /api/v1/scrape : untuk melakukan scraping (selalu dijalankan oleh backend ketika inisialisasi awal)
2. /api/v1/images : untuk mengambil daftar semua elemen dan gambarnya.
3. /api/v1/search : untuk melakukan pencarian resep berdasarkan input target, algoritma, dan jumlah resep.

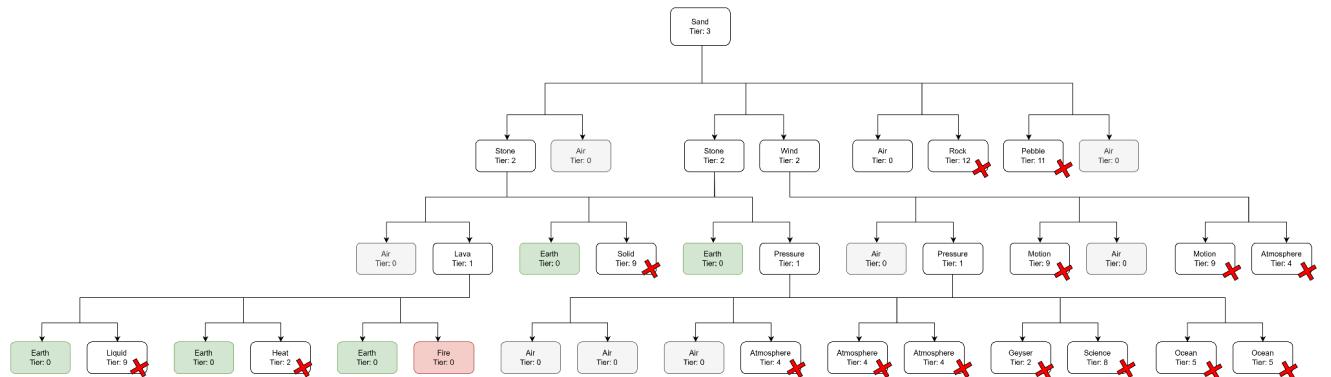
3.4. Contoh Ilustrasi Kasus

Sebagai ilustrasi kasus, salah satu data elemen yang didapatkan melalui *website scraping* pada permainan Little Alchemy 2 adalah elemen *sand* (pasir), yang jika dipetakan ke dalam graf berdasarkan kombinasi elemen penyusunnya membentuk graf sebagai berikut.



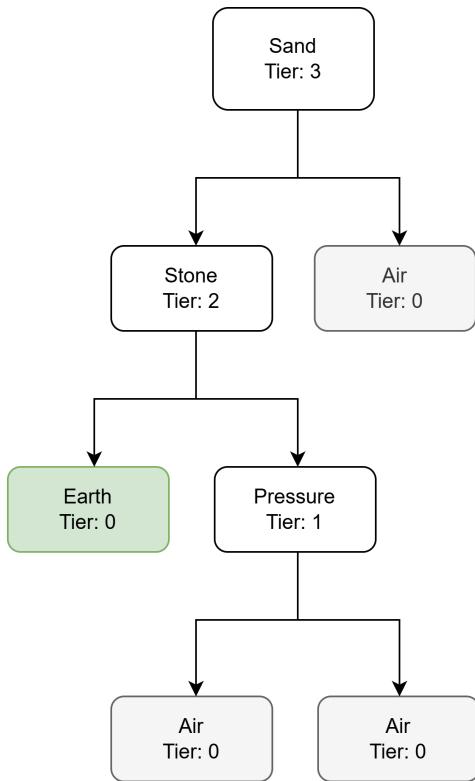
Gambar 8. Graf pemetaan kombinasi pembentuk elemen *sand*

Menyesuaikan spesifikasi pada spek, maka informasi Tier setiap elemen diperhitungkan untuk mengeliminasi kombinasi yang didapatkan dari elemen dengan Tier tidak lebih kecil dibandingkan elemen yang dibentuk oleh kombinasi tersebut.



Gambar 9. Graf pemetaan kombinasi valid pembentuk elemen *sand*

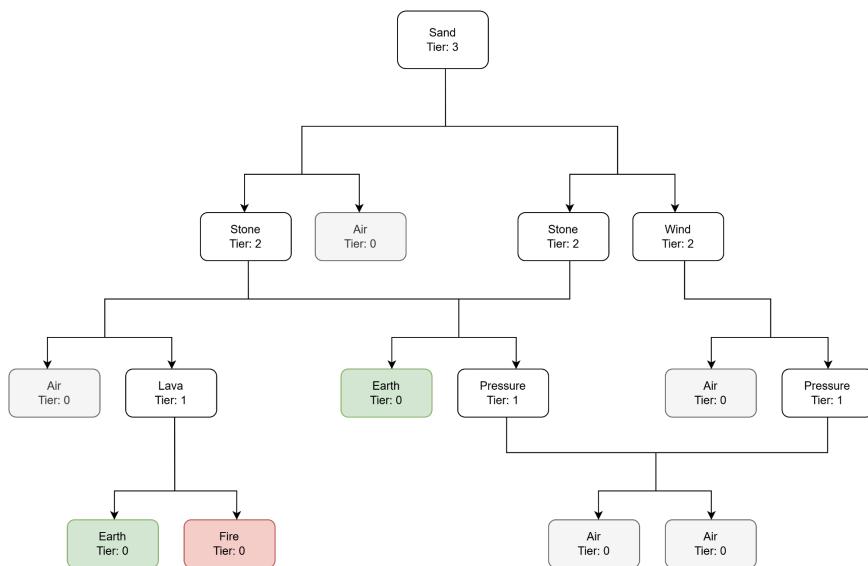
Dengan graf tersebut, algoritma pencarian DFS dan BFS akan memberikan hasil yang berbeda sebagai berikut.



Gambar 10. Graf hasil pencarian DFS untuk elemen *sand*

DFS melakukan pencarian secara mendalam terhadap satu per satu kombinasi pembentuk elemen *sand*. Kombinasi yang pertama diteliti adalah kombinasi *stone - air*, yang karena *stone* bukan merupakan elemen dasar, maka akan lanjut untuk diteliti menggunakan fungsi yang sama.

Penelusuran jalur kemudian selesai ketika elemen *pressure* dapat dibentuk oleh dua elemen dasar. Karena dinyatakan bahwa didapatkan jalur kombinasi yang dibentuk hanya menggunakan komponen dasar, maka jalur tersebut dikembalikan sebagai hasil dari pencarian DFS, dengan hasil seperti gambar di samping.



Gambar 11. Graf hasil pencarian BFS untuk elemen *sand*

Di sisi lain, BFS melakukan pencarian secara melebar, melakukan iterasi kepada setiap kombinasi yang valid dari elemen *sand*, sebelum meneliti lebih dalam untuk setiap elemen pembentuk elemen *sand* yang bukan merupakan elemen dasar. Hal ini menyebabkan graf hasil pencarian BFS menunjukkan banyak jalur kombinasi, sebagai hasil dari natur algoritma BFS yang meneliti satu level dengan menyeluruh sebelum lanjut ke level selanjutnya.

BAB 4: Implementasi dan Pengujian

4.1. Spesifikasi Teknis Program

Struktur data yang digunakan meliputi empat struktur data major berikut.

1. Element

Struktur data Element merepresentasikan elemen dalam permainan Little Alchemy 2. Struktur ini dibentuk menggunakan *struct* sebagai kesatuan dari dari ‘Name’, ‘CanCreate’, ‘Combinations’, dan ‘Tier’. Name merepresentasikan nama elemen, Combinations memetakan setiap kombinasi yang dapat membentuk elemen tersebut, dan Tier menyatakan nilai *tier* dari elemen sesuai dengan data yang didapatkan dari hasil *website scraping* Fandom Little Alchemy 2. CanCreate disiapkan sebagai dasar data dalam membentuk algoritma *bi-directional*, yang sayangnya tidak sempat untuk diimplementasikan karena keterbatasan waktu.

```
type Element struct {
    Name      string
    CanCreate []string
    Combinations []*Combination
    Tier      int
}
```

2. Combination

Struktur data Combination merepresentasikan kombinasi sepasang elemen yang dapat membentuk elemen lain. Struktur ini dibentuk menggunakan *struct* sebagai kesatuan dari ‘ResultName’, ‘LeftName’, dan ‘RightName’. LeftName dan RightName menyatakan komponen elemen yang digunakan, dan menghasilkan elemen dengan nama yang disimpan dalam ResultName.

```
type Combination struct {
    ResultName string
    LeftName   string
    RightName  string
}
```

3. GraphNode

Struktur data GraphNode merepresentasikan simpul sebagai komponen dari keluaran graf hasil pencarian oleh program. Struktur ini dibentuk menggunakan *struct* sebagai kesatuan dari ‘ID’ dan ‘Label’. ID menyatakan bilangan unik sebagai identifikasi suatu simpul, dan Label menyatakan nama elemen yang direpresentasikan oleh simpul tersebut.

```
type GraphNode struct {
    ID      int `json:"id"`
    Label  string `json:"label"`
}
```

4. GraphEdge

Struktur data GraphEdge merepresentasikan sisi sebagai komponen dari keluaran graf hasil pencarian oleh program. Struktur ini dibentuk menggunakan *struct* sebagai kesatuan dari ‘From’ dan ‘To’, yang keduanya menyatakan nilai ID dari simpul yang dihubungkan sebagai tetangga melalui sisi tersebut.

```
type GraphEdge struct {
    From int `json:"from"`
    To   int `json:"to"`
}
```

Fungsi dan prosedur yang digunakan dikelompokkan berdasarkan penggunaannya dalam algoritma.

1. Implementasi Algoritma DFS

Nama Fungsi	Deskripsi
DFS(string) → (interface{}, error, int)	Fungsi <i>helper</i> untuk menjalankan algoritma DFS dalam menjelajahi graf dan menemukan resep. Fungsi menerima masukan nama elemen yang ingin dicari untuk memberikan keluaran jalur kombinasi yang valid, <i>error message</i> jika terjadi error, dan jumlah elemen yang dilakukan pengecekan untuk dapat memberikan hasil.

findCombinationRouteDFS(*Element) → (interface{}, int)	Fungsi algoritma DFS yang dibentuk dengan pendekatan rekursif. Fungsi menerima masukan struktur data Element sesuai dengan masukan pengguna dan mengembalikan jalur kombinasi yang valid serta jumlah elemen yang dilakukan pengecekan. Fungsi akan membalikan <i>nil</i> jika tidak mendapatkan hasil
TranslateOutputPathToGraph(interface{}, string) → ([]GraphNode, []GraphEdge, err)	Fungsi <i>helper</i> untuk mentranslasikan hasil keluaran dari fungsi DFS satu resep.
convertModifiedToGraphRecursive(interface{}, int, *[]GraphNode, *[]GraphEdge, *int) → err	Fungsi translasi hasil keluaran dari fungsi DFS satu resep yang dibentuk dengan pendekatan rekursif menjadi struktur data yang dapat diterima oleh <i>frontend</i> berupa GraphNode dan GraphEdge.
TranslateMultipleRecipeOutputToGraph(string, []interface{}) → ([]GraphNode, []GraphEdge, err)	Fungsi <i>helper</i> untuk mentranslasikan hasil keluaran dari fungsi DFS banyak resep.
processRecipeAlternativesInternal([]interface{}, string, int, *[]GraphNode, *[]GraphEdge, *int, string) → err	Fungsi <i>helper</i> untuk mentranslasikan hasil keluaran dari fungsi DFS banyak resep.
processSingleRecipePathIngredientsInternal	Fungsi translasi hasil keluaran dari fungsi DFS banyak resep yang dibentuk dengan pendekatan rekursif menjadi struktur data yang dapat diterima oleh <i>frontend</i> berupa GraphNode dan GraphEdge.

2. Implementasi Algoritma BFS

Nama Fungsi	Deskripsi
stringToElement(map[string]*Element, string) → *Element	Fungsi untuk mengubah nama sebuah elemen menjadi struktur data Element dengan nama tersebut. Fungsi menerima masukan berupa <i>map</i> hasil pengolahan data <i>website scraping</i> dan nama elemen yang akan diubah.
isBasicElementByName(string) → bool	Fungsi untuk menentukan apakah suatu nama elemen merupakan elemen dasar, yaitu Water, Earth, Fire, atau Air.

isAllBasicElement([]string) → bool	Fungsi untuk menentukan apakah kumpulan nama terdiri atas hanya elemen-elemen dasar, menggunakan fungsi isBasicElementByName dalam implementasinya.
findCombinationRouteBFS(map[string]*Element, string, int) → ([]GraphNode, []GraphEdge, int)	Fungsi algoritma BFS yang dibentuk dengan pendekatan iteratif. Fungsi menerima <i>map</i> hasil pengolahan data <i>website scraping</i> , nama elemen masukan pengguna, serta jumlah minimal resep yang akan ditampilkan sesuai dengan masukan pengguna dan mengembalikan jalur kombinasi yang valid (langsung dalam bentuk graf) serta jumlah elemen yang dilakukan pengecekan. Fungsi akan membalikan <i>nil</i> jika tidak mendapatkan hasil.
createFixList([][]string) → ([][]string, []string, map[string][][]string)	Fungsi menerima hasil kasar dari jalur kombinasi dan melakukan perhitungan untuk memastikan hasil yang ditampilkan merupakan hasil minimal dari algoritma.
reduceMap(map[string][][]string, string, []string) → map[string][][]string	Fungsi yang mereduksi <i>map</i> untuk memastikan jumlah resep yang dihasilkan minimal
limitRecipe(map[string][]int, map[string][][]string, string, int) → map[string][][]string	Fungsi <i>helper</i> dalam mereduksi <i>map</i>
expandNode(map[string][][]string, string, int, int) → ([]GraphNode, []GraphEdge)	Fungsi menerima <i>map</i> berisi hasil bersih jalur kombinasi dan membentuk keluaran graf.
createBFSNode(map[string][][]string, []string) → ([]GraphNode, []GraphEdge)	Fungsi <i>helper</i> dalam membentuk keluaran graf.
BFS(string, int) → ([]GraphNode, []GraphEdge, int, err)	Fungsi <i>helper</i> untuk menjalankan algoritma BFS dalam menjelajahi graf dan menemukan resep. Fungsi menerima masukan nama elemen yang ingin dicari dan jumlah resep yang ingin diketahui untuk memberikan keluaran graf, <i>error message</i> jika terjadi error, dan jumlah elemen yang dilakukan pengecekan untuk dapat memberikan hasil.

3. Implementasi *multithreading*

Mengenai *multithreading*, kelompok setuju untuk melakukan *multithreading* pada DFS yang secara natur melakukan penjelajahan terhadap satu jalur tanpa memertimbangkan jalur lain sebelum jalur yang bersangkutan memberikan hasil. Dengan demikian, untuk setiap elemen yang memiliki lebih dari satu kombinasi untuk menciptakan elemen tersebut, akan dibentuk *thread* baru untuk setiap kombinasi sehingga proses DFS dapat dijalankan secara paralel yang meningkatkan efisiensi algoritma DFS dalam program..

Namun, hal ini berbeda dengan BFS yang kelompok implementasikan. Karena BFS mengutamakan penjelajahan setiap kemungkinan kombinasi di level yang sama terlebih dahulu, sangat memungkinkan bahwa setelah satu tahapan BFS menghasilkan sejumlah resep sekaligus. Maka, fungsi dasar BFS telah mengakomodasi kebutuhan akan jumlah resep lebih dari satu, dan tidak dibutuhkan *multithreading* dalam mencari *multiple recipe*.

Nama Fungsi	Deskripsi
DFSMultipleRecipe(string, int) → ([]interface{}, error, int, int)	Fungsi <i>helper</i> dalam melakukan <i>multithreading</i> terhadap pencarian DFS untuk elemen masukan pengguna. Fungsi menerima masukan nama elemen yang akan dijelajahi dan jumlah resep yang ingin untuk ditampilkan.
findMultipleRouteDFS(*Element, int) → ([]interface{}, int, int)	Fungsi <i>multithreading</i> terhadap pencarian DFS, menggunakan <i>mutex</i> untuk memastikan tidak terjadi <i>race condition</i> antara <i>thread</i> yang satu dengan yang lain, karena setiap <i>thread</i> akan melakukan perubahan terhadap sumber daya yang sama dalam membentuk keluaran graf

4. Implementasi *website scraper*

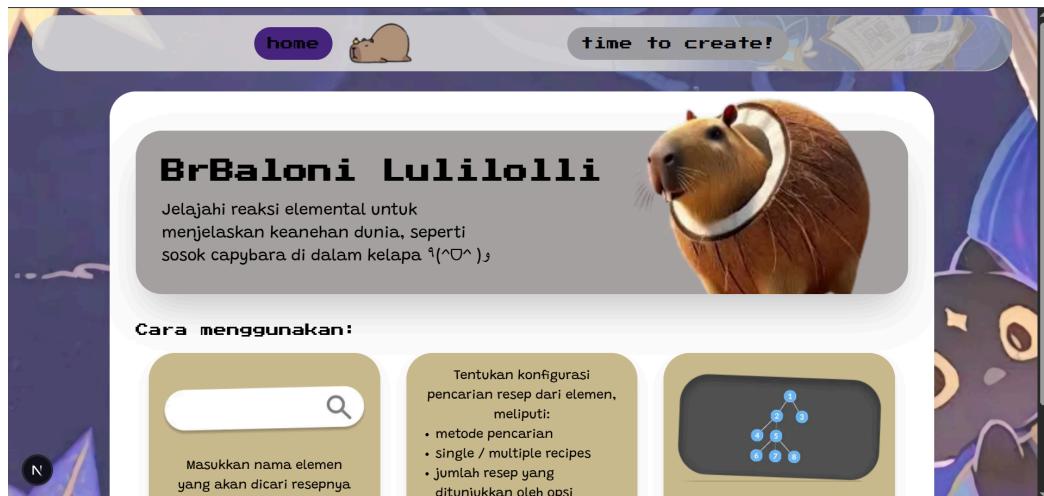
Nama Fungsi	Deskripsi
ScrapeElements() → map[string]*Element, error	Fungsi untuk scraping data elemen dari website Little Alchemy 2, menyimpan data tersebut dalam file elements.json dan membentuk <i>map</i> yang memetakan nama elemen kepada struktur data Element untuk elemen tersebut

LoadElementsFromFile()
→ map[string]*Element, error

Fungsi *helper* untuk melakukan *website scraping* data setiap elemen dalam *website* Little Alchemy 2

4.2. Penjelasan Tata Cara Penggunaan Program

Saat membuka aplikasi web, pengguna akan masuk ke dalam laman *Home* terlebih dahulu. Laman ini bertujuan sebagai *landing page* aplikasi web dan sebagai pemberi petunjuk mengenai tata cara penggunaan aplikasi web.



Gambar 12. Tampilan laman *Home* aplikasi web

Penggunaan program sendiri dilakukan pada laman *Main*. Pengguna dapat memanfaatkan *navigation bar* pada bagian atas aplikasi untuk pindah ke laman tersebut dengan menekan tombol dengan tulisan “*time to create!*” pada bagian atas web.



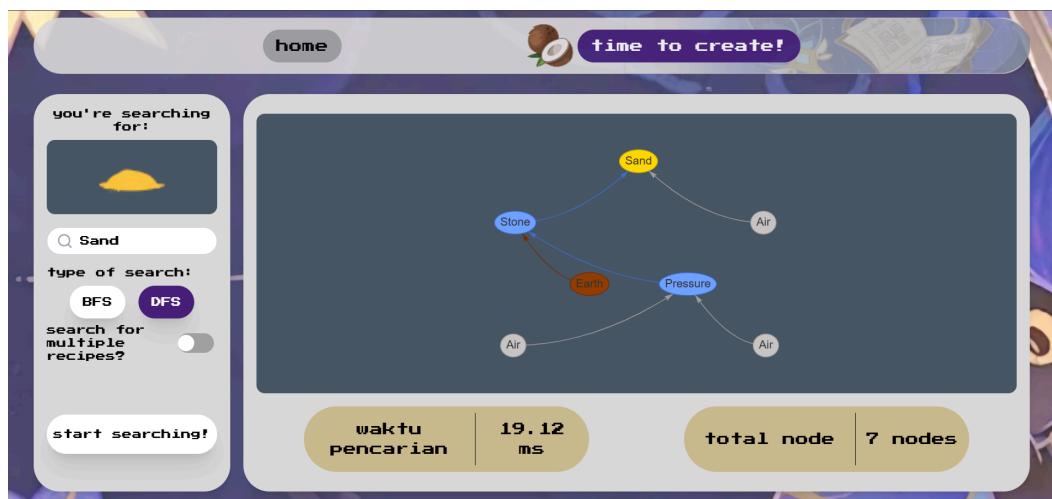
Gambar 13. Tampilan laman *Main* aplikasi web

Laman *Main* merupakan laman utama penggunaan program. Untuk melakukan pencarian, pengguna harus terlebih dahulu memasukkan data-data *input* yang dibutuhkan pada *sidebar* aplikasi web. Pengguna perlu memasukkan nama elemen yang resepnya ingin dicari pada *search bar*. Aplikasi telah melakukan *scraping* data nama-nama elemen beserta gambar elemen tersebut sehingga ketika mengklik *search bar*, akan muncul *dropdown* yang berisi daftar elemen. Ketika pengguna telah memasukkan nama elemen yang terdapat pada daftar tersebut, aplikasi akan menampilkan gambarnya juga di atas *search bar*.



Gambar 14. Pemilihan elemen pada laman *Main*

Selanjutnya, pengguna perlu memilih algoritma pencarian yang diinginkan. Aplikasi menyediakan pilihan antara algoritma BFS atau DFS. Kemudian, pengguna dapat memilih untuk mencari banyak resep. Pilihan ini bersifat opsional sehingga pengguna dapat langsung melakukan pencarian. Namun, jika pengguna memilih ini, maka pengguna dapat memasukkan jumlah resep yang diinginkan. Terakhir, untuk memulai pencarian pohon resep setelah memasukkan semua data *input* yang dibutuhkan, pengguna dapat menekan tombol di bawah *sidebar* dengan tulisan “start searching!” Program akan menerima data dari bagian *backend* dan mengolahnya menjadi pohon resep untuk ditampilkan. Berikut adalah contoh penggunaannya pada pencarian pohon resep untuk elemen *Sand* dengan pencarian DFS.



Gambar 15. Tampilan laman *Main* ketika mencari resep *Sand*

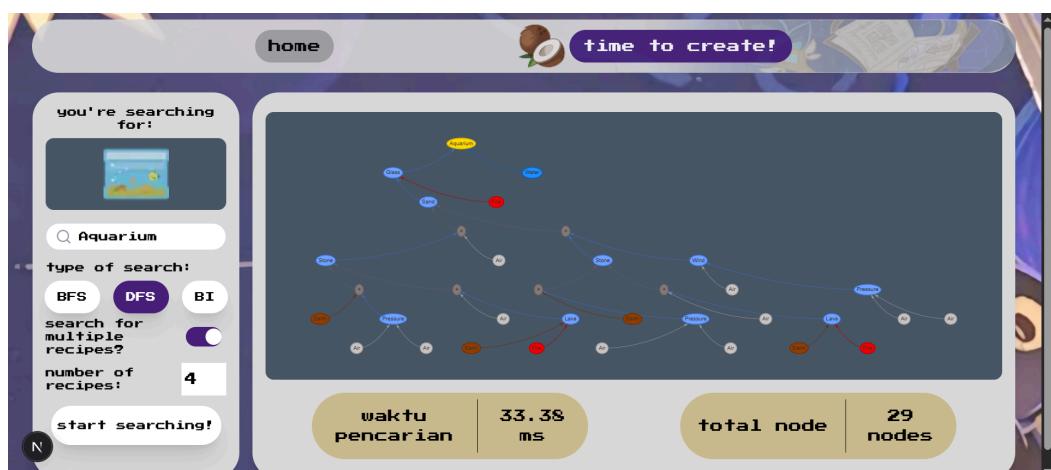
4.3. Hasil Pengujian

- Mencari *Aquarium* dengan algoritma DFS satu resep



Gambar 16. Tampilan hasil pencarian DFS satu resep elemen aquarium

b. Mencari *Aquarium* dengan algoritma DFS empat resep



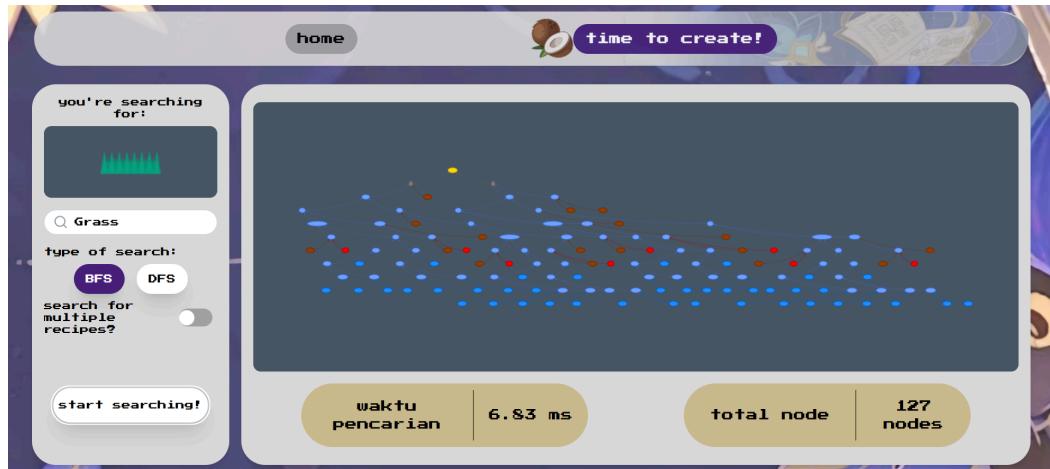
Gambar 17. Tampilan hasil pencarian DFS empat resep elemen aquarium

c. Mencari *Aquarium* dengan algoritma BFS



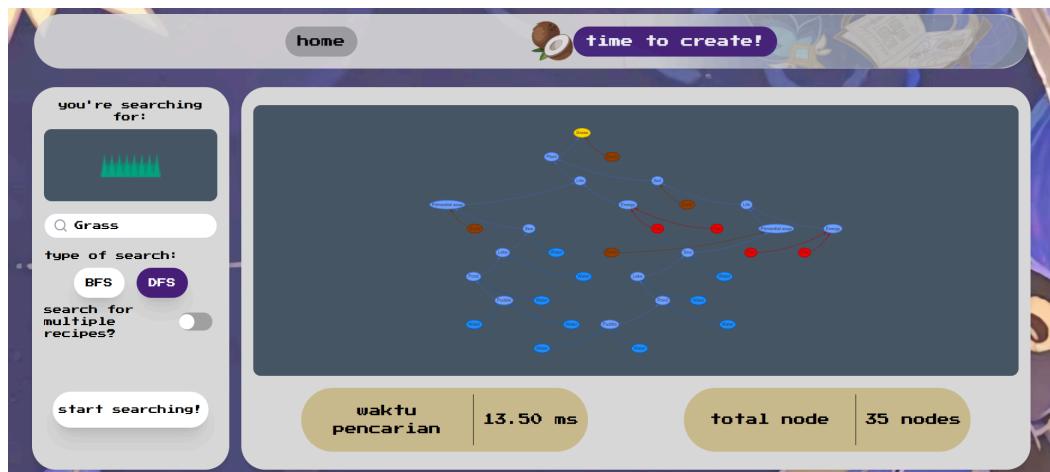
Gambar 18. Tampilan hasil pencarian BFS elemen aquarium

d. Mencari *Grass* dengan algoritma BFS



Gambar 19. Tampilan hasil pencarian BFS elemen grass

e. Mencari *Grass* dengan algoritma DFS



Gambar 20. Tampilan hasil pencarian DFS elemen grass

f. Mencari *Grass* dengan algoritma DFS tiga resep



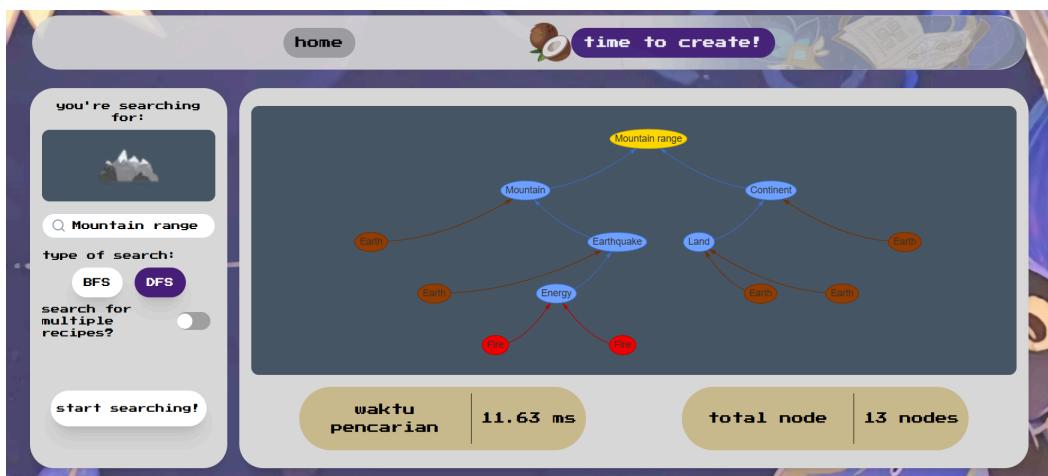
Gambar 21. Tampilan hasil pencarian DFS tiga resep elemen grass

g. Mencari *Mountain Range* dengan algoritma BFS



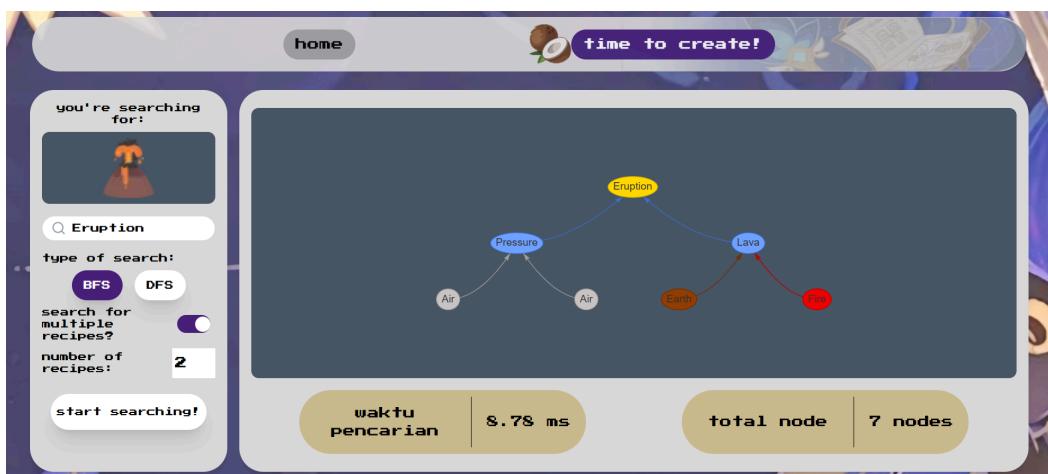
Gambar 22. Tampilan hasil pencarian BFS elemen mountain grass

h. Mencari *Mountain Range* dengan algoritma DFS



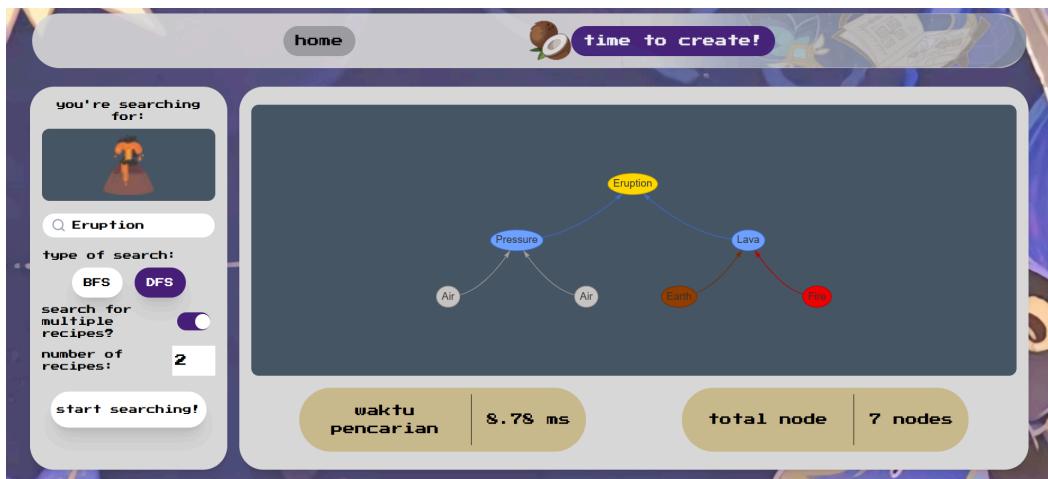
Gambar 23. Tampilan hasil pencarian DFS elemen mountain grass

- Mencari *Eruption*, elemen dengan hanya satu resep, dengan algoritma BFS dua resep



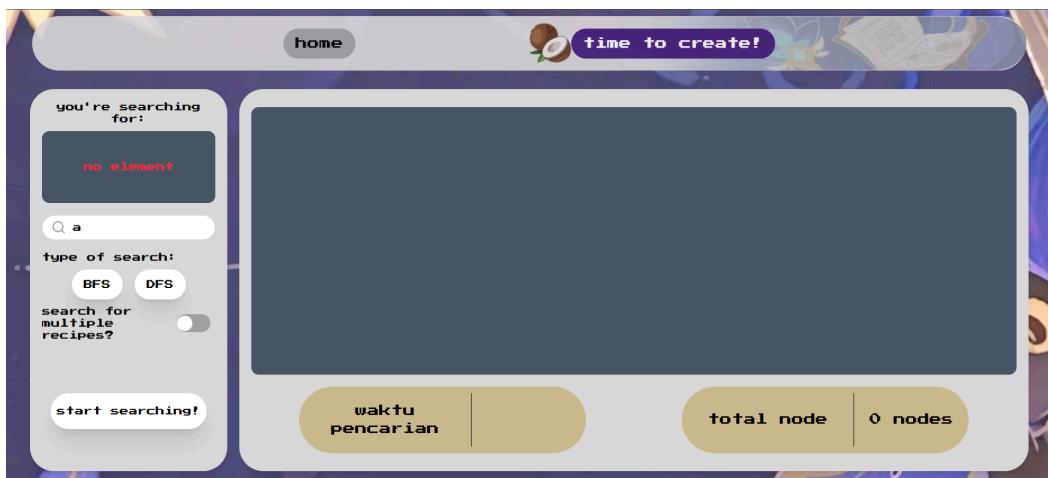
Gambar 24. Tampilan hasil pencarian BFS elemen eruption dua resep

- Mencari *Eruption*, elemen dengan hanya satu resep, dengan algoritma DFS dua resep



Gambar 25. Tampilan hasil pencarian DFS elemen eruption dua resep

- Error handler menangani masukan elemen tidak dikenali



Gambar 26. Error handler menangani masukan elemen tidak dikenali

- Error handler menangani tidak memberikan masukan elemen



Gambar 27. Error handler menangani tidak memberikan masukan elemen

m. Error handler menangani tidak memberikan masukan algoritma



Gambar 28. Error handler menangani tidak memberikan masukan algoritma

4.4. Analisis Hasil Pengujian

Berdasarkan pengujian yang dilakukan, diperoleh hasil berupa waktu pencarian dan jumlah node yang dikunjungi selama proses pencarian jalur resep menuju elemen target. Terlihat bahwa dengan algoritma DFS, jumlah node yang dikunjungi lebih sedikit dibandingkan dengan algoritma BFS, karena DFS langsung menelusuri jalur dalam hingga menemukan solusi pertama, sedangkan BFS mengeksplorasi semua kemungkinan di setiap level sebelum melanjutkan ke level berikutnya.

Meskipun jumlah node yang dikunjungi lebih banyak, rata-rata waktu pencarian dengan BFS lebih cepat daripada waktu pencarian dengan DFS. BFS selalu mendapatkan

hasil yang paling optimal, sedangkan DFS akan memberikan solusi pertama yang ditemukannya, yang belum tentu merupakan jalur terpendek atau paling efisien. Selain itu, kompleksitas waktu DFS juga dipengaruhi oleh *overhead* yang besar dari proses rekursifnya. Meskipun demikian, BFS membutuhkan memori yang jauh lebih banyak dari DFS, terlihat dari perbedaan jumlah node yang dikunjungi secara signifikan.

Untuk waktu pencarian DFS dengan banyak resep, waktu pencarinya lebih lama, dan jumlah node yang dikunjungi lebih banyak dari DFS satu resep. Namun, perbedaan waktu pencarian tidak begitu besar karena penggunaan *multithreading*, optimasi pilihan kombinasi, serta *caching*.

BAB 5: Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Tugas Besar II Strategi Algoritma menyajikan permasalahan pencarian jalur-jalur kombinasi / resep pembentukan elemen dalam permainan Little Alchemy 2. Elemen dalam permainan Little Alchemy 2 dibentuk sebagai kombinasi dari elemen-elemen yang telah ditemukan, dan dimulai dengan empat elemen dasar, yaitu *water*, *earth*, *air*, dan *fire*. Proses pencarian menggunakan algoritma BFS dan DFS untuk sesuai dengan jumlah resep yang diinginkan oleh pengguna, dan hasil dari pencarian jalur dipetakan ke dalam graf, beserta jumlah elemen yang dilakukan pengecekan berdasarkan algoritma yang ditentukan dan waktu yang dibutuhkan program dalam memberikan hasil.

Keseluruhan program diimplementasikan sebagai *website* yang menyediakan *interface* sederhana dalam menggunakan program. Pembentukan *website* memanfaatkan Go sebagai bahasa pemrograman dalam pembentukan *backend*, Next.js sebagai *framework* dalam pembentukan *frontend*, beserta React, HTML, dan CSS untuk memberikan pengalaman baik dalam menggunakan *website*. Seluruh pemilihan komponen telah dijelaskan pada Bab 3.3 dalam laporan.

Segala macam bentuk pengerjaan, seperti dasar teori, hasil pembentukan *source code*, alasan penggunaan algoritma tertentu, pengimplementasian fungsi, serta tampilan layar *website* hasil kerja kelompok telah kelompok dokumentasikan secara terstruktur melalui laporan ini. Meskipun masih jauh dari kata sempurna, dengan berbangga hati kelompok menyatakan bahwa kelompok telah berhasil membentuk *website* yang mampu memberikan jalur kombinasi / resep menuju masukan elemen menggunakan pendekatan algoritma BFS dan DFS sebagai hasil akhir dari tugas besar II Strategi Algoritma.

5.2 Saran

Kelompok menyadari bahwa hasil akhir yang dibentuk oleh kelompok masih jauh dari kata maksimal. Terdapat beberapa perbaikan-perbaikan yang dapat diusahakan lebih lanjut, seperti dalam pembentukan algoritma yang lebih efektif, implementasi *website*, dan *benchmark performance* dari kinerja *website* keseluruhan. Namun, mengingat waktu yang

minimal akibat kesibukan masing-masing anggotanya, pengembangan lebih lanjut diserahkan sebagai implementasi lanjutan untuk dikerjakan di waktu luang sebagai projek pribadi.

5.3 Refleksi

Kelompok mengucapkan terima kasih yang sebesar-besarnya kepada Bu Ulfah dan Pak Rinaldi selaku dosen pengampu mata kuliah Strategi Algoritma pada kampus Ganesha serta seluruh asisten yang telah memberikan perannya dalam pelaksanaan tugas besar kedua atas kesempatan yang telah diberikan untuk mengerjakan tugas besar mata kuliah Strategi Algoritma. Melalui tugas ini, kelompok telah mendapatkan beragam bentuk ilmu-ilmu dalam lingkup keinformatikaan, pengalaman dalam membentuk proyek informatika berbasis *website*, serta latihan keterampilan-keterampilan yang relevan bagi seorang mahasiswa informatika, seperti kemampuan bekerja sama dan berkomunikasi dalam kelompok, kemampuan mengoperasikan *git* sebagai instrumen kolaborasi, kemampuan *debugging* dan melakukan *handle* terhadap *error* dalam kode pemrograman, dan keterampilan-keterampilan relevan lainnya. Kiranya apa yang didapatkan melalui penggerjaan tugas besar ini menjadi bekal yang baik bagi anggota untuk melangsungkan keseharian perkuliahan di jurusan Teknik Informatika kedepannya.

Lampiran

1. Tautan Repository

Repository Frontend: https://github.com/TukangLas21/Tubes2_FE_Brbaloni-Lulilolli

Repository Backend: https://github.com/angkaberapa/Tubes2_BE_BrBaloni-Lulilolli

2. Tautan Aplikasi Web

<https://tubes2-fe-brbaloni-lulilolli.vercel.app/>

3. Tautan Video

<https://www.youtube.com/watch?v=Klx4UGEPMcg>

4. Tabel Checklist

Tabel 1. Tabel Checklist

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.	✓	

Daftar Pustaka

Informatika.stei.itb.ac.id. (2025). Breadth/Depth First Search (BFS/DFS) (Bagian 1). Diakses pada 8 Mei 2025, dari [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)

Informatika.stei.itb.ac.id. (2025). Breadth/Depth First Search (BFS/DFS) (Bagian 2). Diakses pada 8 Mei 2025, dari [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)

GO. Go Documentation. Diakses 8 Mei 2025 melalui <https://go.dev/doc/>

Network. Vis.js Network Documentation. Diakses 9 Mei 2025 melalui <https://visjs.github.io/vis-network/docs/network/>

Dimitrov, S., Minchev, M., & Zhuang, Y. (2024). BFS versus DFS for random targets in ordered trees. arXiv preprint arXiv:2404.05664.