

LAPORAN TUGAS BESAR 2

IF2211 STRATEGI ALGORITMA



oleh:
Kelompok astimatism

Andi Farhan Hidayat	13523128
Ahmad Syafiq	13523135
Rafael Marchel Darma Wijaya	13523146

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I	
DESKRIPSI TUGAS.....	3
BAB II	
LANDASAN TEORI.....	4
2.1. Algoritma Depth First Search.....	4
2.2. Algoritma Breadth First Search.....	5
2.3. Pembangunan Aplikasi Web.....	7
BAB III	
ANALISIS PEMECAHAN MASALAH.....	9
3.1. Langkah-Langkah Pemecahan Masalah.....	9
3.2. Mapping Persoalan Menjadi Elemen Algoritma DFS dan BFS.....	10
3.2.1. DFS.....	11
3.2.2. BFS.....	14
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web Astigmatism Recipe Finder.....	20
3.4.1. DFS.....	21
3.4.2. BFS.....	25
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	27
4.1. Spesifikasi Teknis Program.....	27
1. Struktur Data.....	27
4.2. Source Code.....	29
4.3. Penjelasan Tata Cara Penggunaan Program.....	64
4.4. Hasil Pengujian.....	64
4.5. Analisis Hasil Pengujian.....	66
BAB V	
KESIMPULAN.....	67
LAMPIRAN.....	69
6.1. Checklist Spesifikasi Program.....	69
6.1. Pranala.....	69
DAFTAR PUSTAKA.....	70

BAB I

DESKRIPSI TUGAS



Gambar 1.1. Little Alchemy 2
(sumber: <https://www.thegamer.com>)

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

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

Tugas Besar 2 Strategi Algoritma kali ini adalah membuat sebuah Little Alchemy Recipe Finder berbasis web dengan mengimplementasikan algoritma Breadth First Search dan Depth First Search. Program ini menerima masukan berupa pilihan metode dan elemen target kemudian program akan memberikan keluaran berupa visualisasi resep membuat elemen target, jumlah simpul yang ditelusuri, serta waktu pencarian.

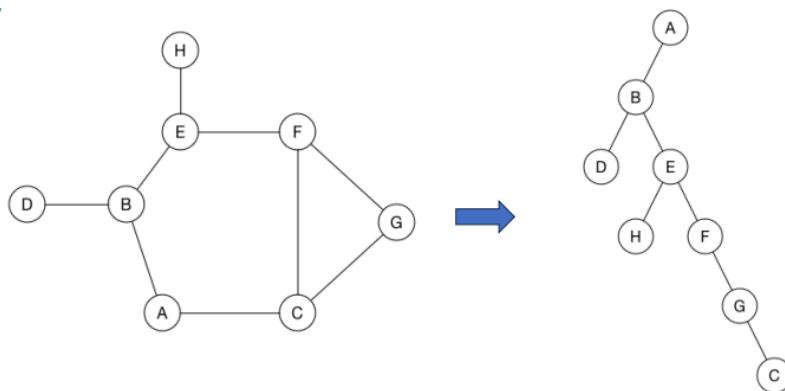
BAB II

LANDASAN TEORI

2.1. Algoritma Depth First Search

Depth-First Search (DFS) atau pencarian mendalam adalah salah satu algoritma pencarian graf yang digunakan untuk menjelajahi dan mencari simpul atau edge dalam sebuah graf. DFS dimulai dengan memilih satu simpul acak dan menandainya sebagai “dikunjungi”. Pada setiap langkah, algoritma ini mengunjungi simpul yang belum dikunjungi dan bersebelahan dengan simpul yang sedang diperiksa. Proses ini berlanjut sampai menemui jalan buntu, yaitu saat tidak ada lagi simpul tetangga yang belum dikunjungi. Kemudian, algoritma akan melakukan *backtracking* ke simpul sebelumnya untuk mencoba mengunjungi simpul yang belum dikunjungi dari sana.

DFS akan berhenti setelah mundur kembali ke simpul awal dan menemui jalan buntu. Pada titik ini, semua simpul dan komponen graf yang sama dengan simpul awal sudah dikunjungi. Jika masih ada simpul yang belum dikunjungi, pencarian harus dimulai lagi dari salah satu simpul di komponen graf lainnya. Dengan demikian, DFS juga dapat digunakan untuk mengidentifikasi komponen terhubung dalam graf tak berarah.



Gambar 2.1.1. Contoh DFS

(sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

Algoritma DFS menggunakan struktur data stack dengan prinsip Last-In, First-Out (LIFO). Stack digunakan untuk melacak simpul yang harus dikunjungi kembali setelah mencapai jalan buntu. Implementasi DFS juga dapat menggunakan pendekatan rekursif, yang secara implisit memanfaatkan call stack sebagai struktur data utamanya. Setiap kali DFS masuk lebih dalam ke simpul baru, simpul tersebut ditambahkan ke

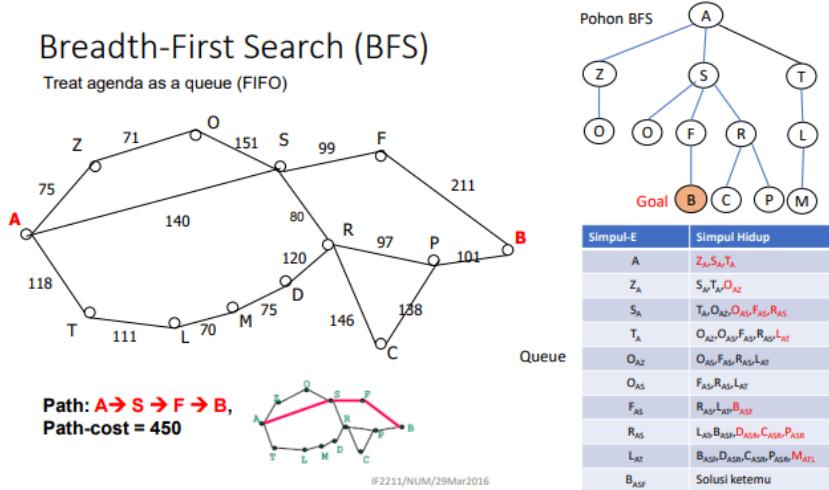
stack; jika menemui jalan buntu, algoritma akan mundur ke simpul sebelumnya yang masih memiliki tetangga belum dikunjungi.

Keunggulan utama DFS adalah efisiensinya dalam penggunaan memori pada graf yang tidak terlalu lebar, serta kemampuannya menjelajah hingga ke simpul terdalam secara cepat. DFS sangat cocok digunakan untuk menyelesaikan permasalahan seperti deteksi siklus dalam graf, topological sorting, dan pencarian komponen terhubung. Selain itu, DFS lebih mudah diadaptasi untuk berbagai varian permasalahan yang memerlukan eksplorasi mendalam terlebih dahulu.

Namun demikian, DFS juga memiliki beberapa kelemahan. Salah satunya adalah kemampuannya yang buruk dalam menemukan jalur terpendek dalam graf tak berbobot, karena ia tidak menjamin eksplorasi simpul terdekat terlebih dahulu. DFS juga dapat menyebabkan eksplorasi yang terlalu dalam pada graf besar, sehingga tidak efisien pada graf yang luas dan dalam. Pada implementasi rekursif, DFS juga rentan terhadap stack overflow jika kedalaman graf sangat besar.

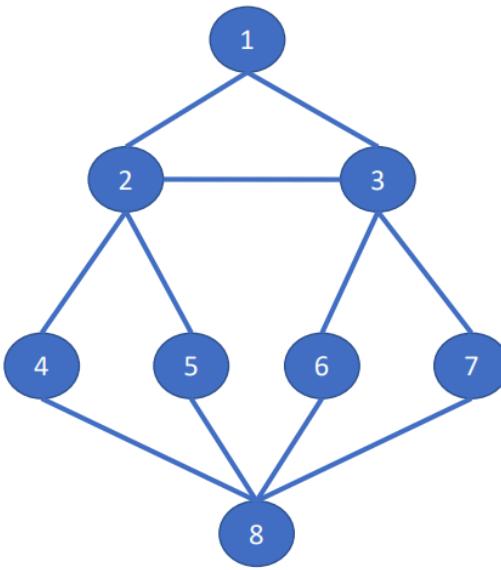
2.2. Algoritma Breadth First Search

Breadth-First Search (BFS) atau pencarian melebar adalah salah satu algoritma pencarian graf yang digunakan untuk menjelajahi dan mencari simpul atau edge dalam sebuah graf. BFS bekerja dengan mengeksplorasi simpul-simpul graf secara bertahap, dimulai dari simpul awal, kemudian mengunjungi semua simpul tetangga terdekat sebelum berpindah ke simpul-simpul yang lebih jauh. Pendekatan ini menjadikan algoritma BFS terlihat “berhati-hati” dibandingkan dengan algoritma DFS yang cenderung “berani” dalam menjelajahi simpul-simpul yang lebih dalam terlebih dahulu. Algoritma BFS sangat berguna dalam menemukan jalur terpendek pada graf tak berbobot.



Gambar 2.2.1. Contoh 1 BFS

(sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>)



Gambar 2.2.1. Contoh 2 BFS

(sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

Inti dari algoritma BFS adalah penggunaan struktur data queue (antrean) yang beroperasi dengan prinsip First-In, First-Out (FIFO). Proses dimulai dengan menandai simpul awal sebagai dikunjungi dan memasukkannya ke dalam queue. Kemudian, simpul di depan queue diambil, dan semua simpul tetangga yang belum dikunjungi dimasukkan ke dalam queue. Simpul-simpul tersebut kemudian ditandai sebagai dikunjungi untuk

mengurangi pengulangan. Proses ini terus berlanjut hingga queue kosong atau simpul tujuan telah ditemukan.

Struktur data umum dari BFS meliputi sebuah matriks ketetanggaan $A = [a_{ij}]$ yang berukuran $n \times n$, $a_{ij} = 1$ jika simpul i dan simpul j bertetangga, $a_{ij} = 0$ jika simpul i dan simpul j tidak bertetangga. Selain itu, terdapat antrian q untuk menyimpan simpul yang akan dikunjungi serta tabel boolean visited.

Keuntungan utama dari BFS adalah kemampuannya untuk menemukan jalur terpendek dalam graf tidak berbobot, karena BFS memastikan bahwa simpul yang lebih dekat dengan simpul awal dijelajahi terlebih dahulu. Hal ini membuat BFS sangat berguna dalam aplikasi pencarian rute di peta, penyelesaian puzzle, dan penyelesaian masalah yang melibatkan pemrograman graf. BFS juga dapat digunakan untuk menentukan apakah sebuah graf terhubung atau tidak dan untuk mencari semua komponen yang terhubung dalam sebuah graf.

BFS juga memiliki beberapa kelemahan. Salah satu kelemahan utama adalah penggunaan memori yang tinggi, terutama pada graf dengan banyak simpul dan tepi karena semua simpul yang belum dieksplorasi perlu disimpan dalam antrian. Selain itu, BFS juga tidak efisien untuk graf dengan kedalaman yang sangat besar atau masalah yang memerlukan eksplorasi lebih dalam sebelum solusi ditemukan.

2.3. Pembangunan Aplikasi Web



Gambar 2.3.1 Halaman utama astigmatism Little Alchemy Recipe Finder

Aplikasi web astigmatism dirancang untuk membantu menemukan resep dari elemen-elemen dalam permainan "Little Alchemy 2", sebuah permainan menggabungkan dua elemen menjadi elemen lain. Aplikasi ini menggunakan dua metode pencarian yaitu Breadth First Search (BFS) dan Depth First Search (DFS). Pengguna dapat memilih metode yang diinginkan dan memilih target elemen yang ingin dibuat. Pengguna juga

dapat memilih mode multiple recipe untuk menampilkan banyak resep sekaligus dengan masukan jumlah resep yang diinginkan. Setelah itu, sistem akan melakukan scraping data resep dari situs wiki fandom Little Alchemy, memprosesnya sesuai metode yang dipilih, dan menampilkan hasil berupa visualisasi pohon resep penggabungan dari elemen dasar hingga mendapatkan elemen target, jumlah simpul yang dilalui selama pencarian, serta durasi pencarian.

Aplikasi web ini dibangun dengan menggunakan next.js untuk antarmuka pengguna dan Golang untuk logika bisnisnya, termasuk pengambilan data dan pemrosesan algoritma pencarian.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Permasalahan utama yang ingin diselesaikan dalam proyek ini adalah menemukan resep pembuatan suatu elemen dalam permainan *Little Alchemy 2*, berdasarkan penggabungan dua elemen lain. Dalam konteks ini, setiap elemen hasil tidak tergantung pada satu elemen lain secara langsung, melainkan pada pasangan elemen tertentu. Oleh karena itu, permasalahan ini tidak dapat diselesaikan menggunakan graf konvensional yang hanya menghubungkan satu simpul ke simpul lain, melainkan membutuhkan struktur graf yang mencerminkan ketergantungan dua elemen sekaligus untuk membentuk elemen baru.

Langkah pertama dalam pemecahan masalah adalah memahami struktur data yang dibutuhkan untuk mewakili semua elemen dan resep di dalam permainan. Data elemen dan resep dikumpulkan melalui proses *scraping* dari situs wiki resmi *Little Alchemy 2*, kemudian diubah menjadi struktur yang merepresentasikan setiap elemen hasil sebagai simpul dalam graf, dengan nilai-nilai ketergantungan berupa daftar pasangan bahan pembentuknya. Dengan kata lain, setiap simpul mewakili satu elemen hasil, dan dihubungkan dengan daftar pasangan elemen yang bila digabung akan menghasilkan elemen tersebut.

Langkah berikutnya adalah membangun graf yang memetakan setiap elemen hasil ke satu atau lebih pasangan bahan pembentuk. Graf ini direpresentasikan dalam bentuk struktur data map id elemen ke senarai pasangan elemen.

Agar hasil pencarian dapat divisualisasikan dan dianalisis dengan baik, perlu ditentukan struktur data yang menyimpan jalur pencarian dalam bentuk pohon. Dalam hal ini, digunakan sebuah struktur *TreeNode*, yang merepresentasikan satu elemen hasil beserta relasinya dengan node-node pembentuknya. Setiap *TreeNode* menyimpan informasi elemen itu sendiri (*Element*), simpul induk (*Parent*), serta daftar anaknya (*Children*) dalam bentuk pasangan melalui *PairNode*. Untuk mencatat jumlah kombinasi resep yang telah ditelusuri, *TreeNode* juga perlu menyimpan jumlah tersebut. Selain itu, diperlukan atribut *visited* yang akan digunakan secara internal untuk membantu pencatatan siklus atau loop selama pencarian, dan tidak ditampilkan ke pengguna. Dengan desain seperti ini, setiap simpul pada pohon dapat menunjukkan dengan jelas dari pasangan mana ia terbentuk, serta memungkinkan traversal dan visualisasi struktur secara rekursif.

Setelah struktur graf dan pohon pencarian ditetapkan, pengguna dapat memasukkan input berupa elemen target yang ingin dicari resepnya, serta memilih metode pencarian yang digunakan: Breadth-First Search (BFS) atau Depth-First Search (DFS). Terdapat pula opsi "multiple recipe" yang memungkinkan pengguna melihat lebih dari satu alternatif kombinasi bahan pembentuk.

Sistem kemudian akan menelusuri graf tersebut dengan metode sesuai dengan pilihan pengguna untuk menemukan jalur-jalur pembentukan elemen target. Selama proses pencarian, sistem akan mencatat simpul yang dikunjungi, jalur kombinasi elemen yang dipilih, serta waktu eksekusi algoritma. Hasil pencarian ini kemudian divisualisasikan dalam bentuk pohon resep, di mana setiap node menunjukkan elemen hasil dan anak-anaknya menunjukkan bahan-bahan pembentuknya sesuai pasangan resep yang dipilih. Hasil visualisasi dan informasi pencarian seperti jumlah simpul yang diakses dan durasi pencarian ditampilkan ke pengguna melalui antarmuka web.

3.2. Mapping Persoalan Menjadi Elemen Algoritma DFS dan BFS

Permasalahan pencarian resep dalam Little Alchemy 2 dimodelkan sebagai masalah eksplorasi graf terarah yang merepresentasikan elemen-elemen sebagai simpul dan setiap edge menunjukkan relasi pembentukan elemen tersebut dari pasangan dua elemen lainnya. Pencarian resep dipandang sebagai proses menelusuri jalur dari elemen target ke elemen-elemen dasar, yaitu elemen yang tidak memiliki resep pembentuk. Artinya, pencarian dimulai dari elemen yang ingin dibentuk, lalu diturunkan ke pasangan-pasangan bahan yang dapat membentuknya, dan dilanjutkan secara rekursif ke bahan-bahan tersebut, hingga mencapai simpul-simpul yang tidak dapat dipecah lebih lanjut.

Hasil pencarian dicatat dalam bentuk sebuah pohon yang terdiri dari simpul-simpul. Pohon tersebut dimulai dari akar berupa elemen target, lalu diturunkan ke pasangan simpul elemen yang membentuknya berdasarkan resep hingga mencapai simpul elemen dasar. Setiap simpul merepresentasikan satu elemen sehingga memerlukan atribut pengidentifikasi elemen yang direpresentasikan. Setiap simpul pada pohon juga perlu menyimpan anaknya yang setiapnya merupakan bagian dari pasangan elemen yang membentuknya, sehingga memerlukan atribut yang dapat menyimpan banyak pasangan simpul.

Untuk mendukung fitur multiple recipe, setiap simpul juga perlu menyimpan banyaknya kombinasi resep valid untuk membentuk elemen tersebut supaya pencarian dapat dihentikan ketika jumlah kombinasi resep valid untuk membuat target sudah memenuhi masukan. Jumlah kombinasi resep valid dihitung dengan menjumlahkan hasil kali kombinasi resep valid tiap pasangan anak (sum of product).

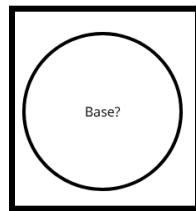
Intinya, elemen target dipetakan sebagai akar pohon pencarian. Setiap resep direpresentasikan sebagai pasangan simpul anak dalam bentuk PairNode, dan simpul leaf merepresentasikan elemen dasar yang tidak memiliki resep. Jumlah kombinasi resep dihitung secara rekursif dari leaf ke akar menggunakan pendekatan *sum of product*.

Adapun rincian mapping persoalan menjadi elemen algoritma DFS dan BFS secara spesifik akan dijelaskan lebih lanjut.

3.2.1. DFS

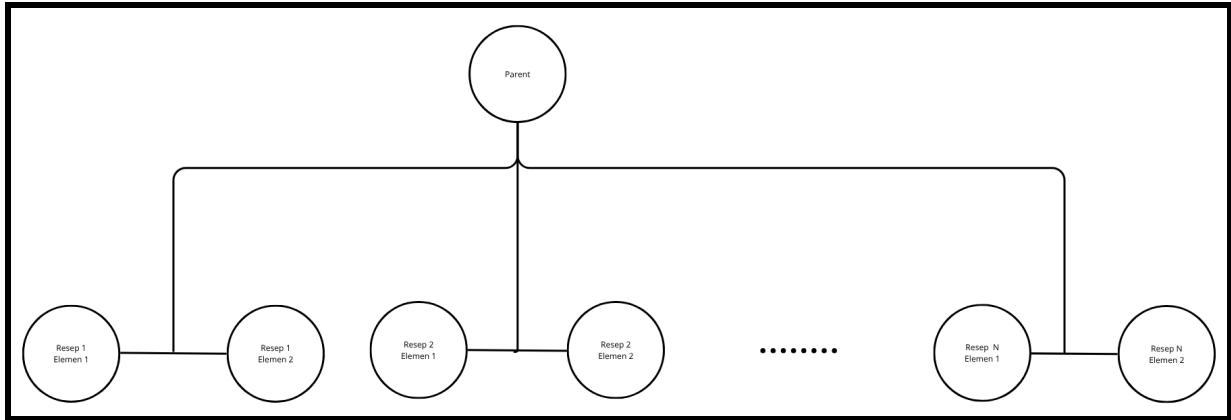
Pencarian resep valid dengan metode Depth First Search dilakukan secara rekursif yang secara implisit memanfaatkan struktur data stack. Resep yang valid ditemukan ketika leaf berupa elemen dasar telah ditemukan pada seluruh percabangannya. Suatu cabang pasti akan menemukan leaf berupa elemen dasar. DFS diterapkan dengan selalu menelusuri resep pertama terlebih dahulu. Pada level resep, DFS juga diterapkan dengan selalu menelusuri elemen pertama (elemen kiri) terlebih dahulu. Pada setiap rekursi, dilakukan:

1. Pengecekan terhadap elemen saat ini. Jika elemen adalah elemen dasar, beri nilai 1 untuk jumlah resep valid elemen tersebut, hentikan penelusuran lebih dalam pada cabang ini, dan kembalikan ke parent.

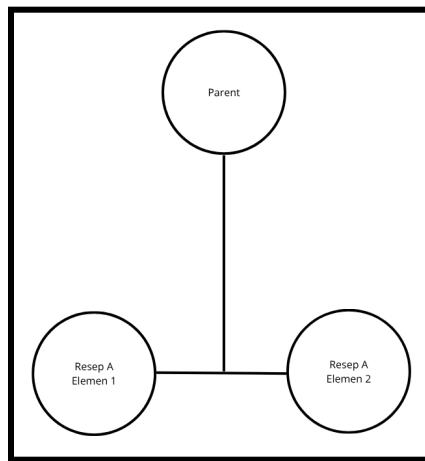


Gambar 3.2.1. Pengecekan elemen saat ini
(sumber: dokumentasi pribadi)

2. Pemilihan pasangan bahan pembentuk dari elemen saat ini. Untuk menghindari dependensi sirkuler dan menyesuaikan spesifikasi tugas, pasangan bahan yang memiliki bahan dengan tier lebih tinggi dari elemen saat ini akan dilompati.

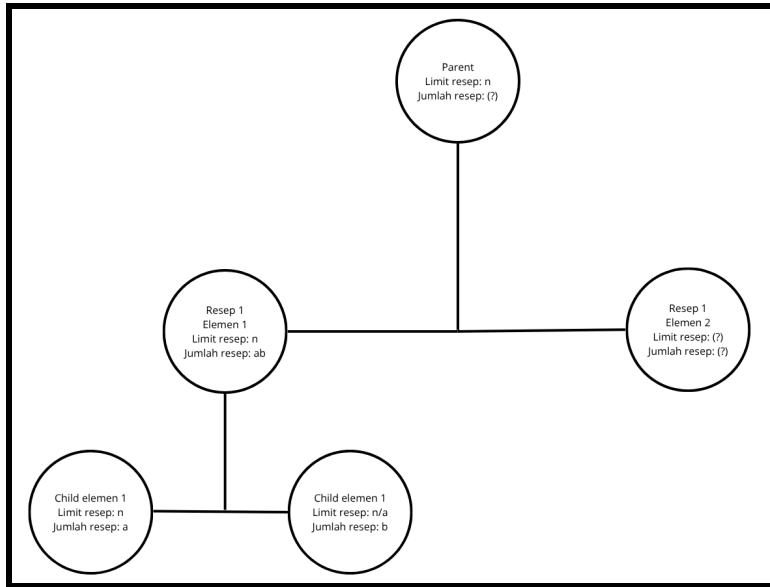


Gambar 3.2.1. Elemen dapat dibentuk dari N resep
(sumber: kpribadi)

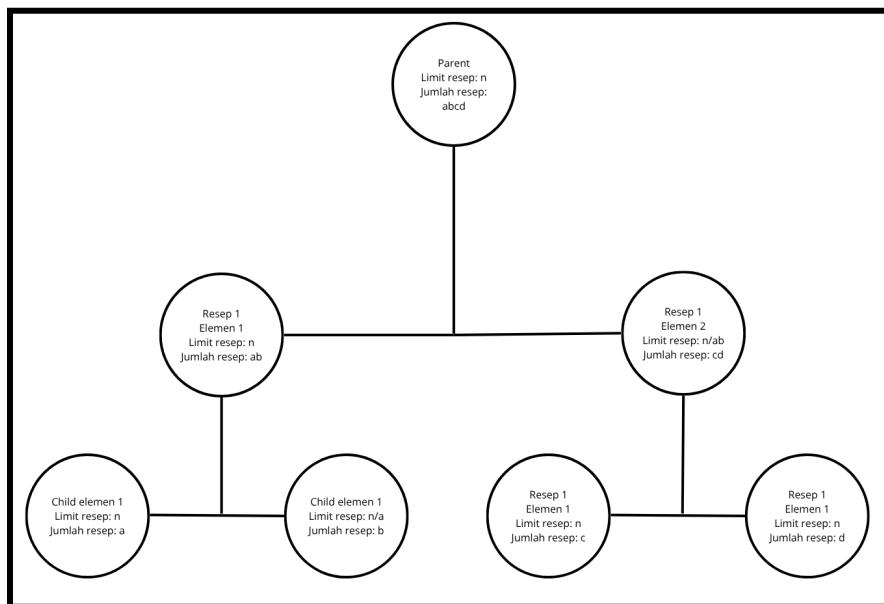


Gambar 3.2.2. Resep pertama dipilih
(sumber: dokumentasi pribadi)

3. Penelusuran secara DFS ke masing-masing elemen dalam pasangan. DFS pertama dilakukan untuk elemen pertama dan hasilnya akan digunakan untuk menentukan alokasi limit pencarian elemen kedua.

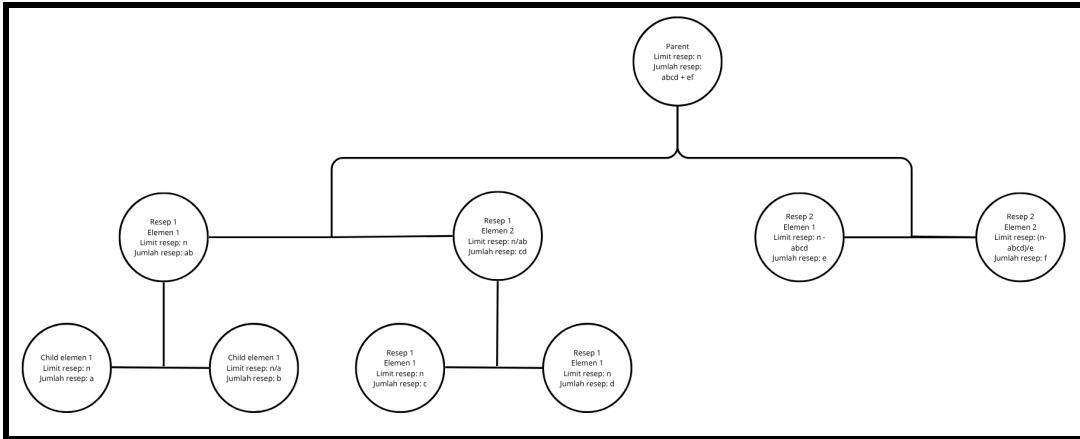


Gambar 3.2.3. DFS ke elemen pertama
(sumber: dokumentasi pribadi)



Gambar 3.2.4. Lanjutan DFS ke elemen kedua
(sumber: dokumentasi pribadi)

4. Penghitungan jumlah kombinasi resep valid dari pasangan bahan. Jumlah resep dari hasil DFS pada kedua bahan pembentuk dikalikan kemudian ditambahkan ke jumlah resep elemen saat ini. Jika jumlah kombinasi resep valid belum memenuhi target, tambahkan resep lain.



Gambar 3.2.5. Lanjutan DFS ke resep kedua
 (sumber: dokumentasi pribadi)

Dengan pendekatan ini, pada setiap elemen, akan terbentuk pohon yang menggambarkan sejumlah cara untuk membentuk elemen tersebut. DFS dihentikan ketika seluruh cabang berakhir di elemen dasar dan tidak dapat/perlu dibuat cabang lagi. Dalam hal ini, pohon dengan akar berupa elemen target akan berisi sejumlah resep yang jumlahnya menyesuaikan masukan pengguna atau merupakan seluruh kemungkinan kombinasi resep yang membentuk elemen target. Pohon tersebut yang akan dikembalikan sebagai hasil dari proses pencarian resep. Sayangnya, pendekatan ini tidak selalu menghasilkan jumlah resep yang akurat sesuai masukan karena tidak bisa mengontrol kombinasi resep yang terbentuk sehingga resep yang diberikan terkadang sedikit melebihi jumlah resep yang diminta.

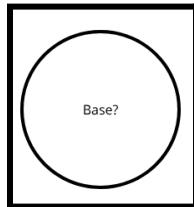
3.2.2. BFS

Pencarian resep valid dengan metode Breadth First Search dilakukan dengan memanfaatkan struktur data queue. Resep yang valid ditemukan ketika leaf berupa elemen dasar telah ditemukan pada seluruh percabangannya. Proses pencarian BFS dimulai dari elemen target, kemudian menelusuri semua kombinasi resep yang dapat membentuk elemen tersebut, selapis demi selapis. Karena BFS menelusuri secara level-order, setiap kemungkinan dikembangkan hingga menemukan leaf node yang merupakan elemen dasar, memastikan bahwa pencarian tetap berada dalam batas dan menghasilkan solusi dengan kedalaman minimum. Dengan pendekatan ini, BFS menjamin bahwa resep valid akan ditemukan dalam urutan level yang paling dangkal terlebih dahulu, yang artinya solusi yang dihasilkan BFS adalah solusi paling sederhana.

BFS memanfaatkan struktur data queue untuk mengelola urutan kunjungan simpul dalam pohon sehingga penelusuran dilakukan selapis demi selapis. BFS dimulai dengan memasukkan akar pohon yang berupa elemen target ke dalam antrian. Kemudian,

simpul diproses urut berdasarkan antrian hingga antrian kosong. Pada setiap proses, dilakukan:

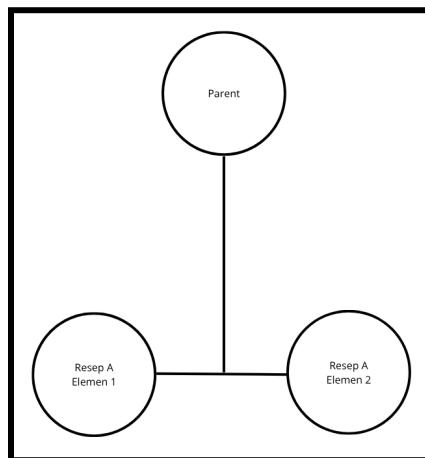
1. Pengecekan terhadap elemen saat ini. Jika elemen adalah elemen dasar, tidak ada proses yang perlu dilakukan sehingga elemen dapat dilompati.



Gambar 3.2.6. Pengecekan elemen saat ini

(sumber: dokumentasi pribadi)

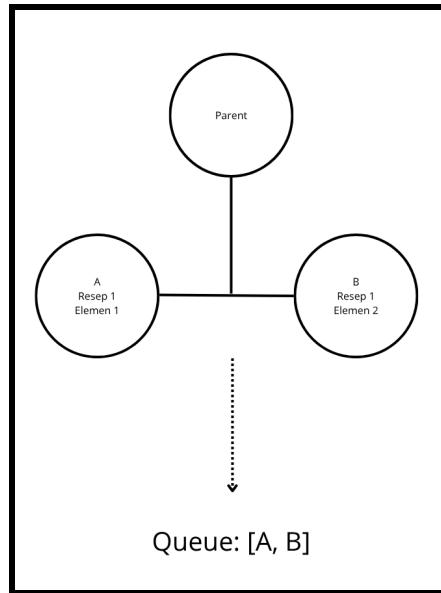
2. Pemilihan resep pertama dari elemen saat ini. Untuk menghindari dependensi sirkuler dan menyesuaikan spesifikasi tugas, pasangan bahan yang memiliki bahan dengan tier lebih tinggi dari elemen saat ini akan dilompati dan dilanjutkan ke resep selanjutnya. Kemudian, karena spesifikasi sudah menyaring resep yang memiliki bahan dengan tier lebih tinggi, dependensi sirkuler tidak akan terjadi sehingga atribut visited tidak diperlukan. Namun, kami tetap menambahkannya supaya tetap mengikuti algoritma BFS klasik.



Gambar 3.2.7. Pemilihan resep pertama

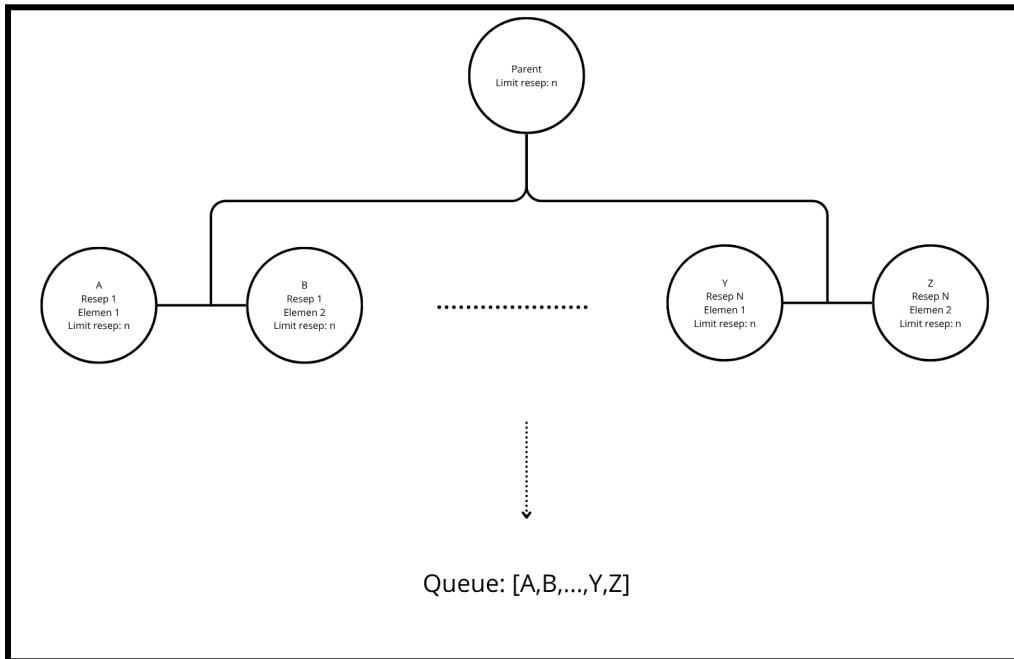
(sumber: dokumentasi pribadi)

3. Penciptaan simpul kedua elemen resep sebagai anak. Simpul kemudian dimasukkan ke dalam antrian untuk selanjutnya diproses.



Gambar 3.2.8. Memasukkan elemen ke antrian
(sumber: dokumentasi pribadi)

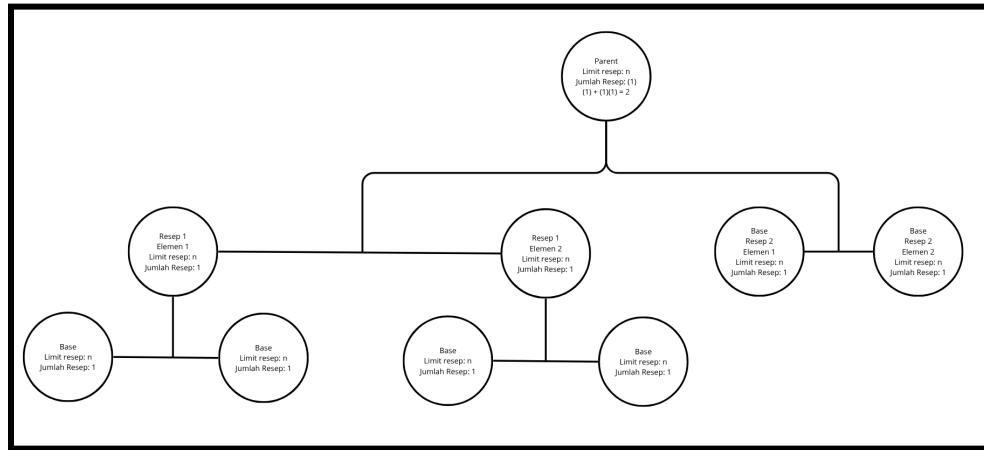
4. Perlakuan yang sama dilakukan ke setiap resep yang lain kecuali resep yang terdiri dari bahan dengan tier yang lebih tinggi sehingga antrian diisi semua simpul elemen dari tiap resep yang dapat membentuk parent.



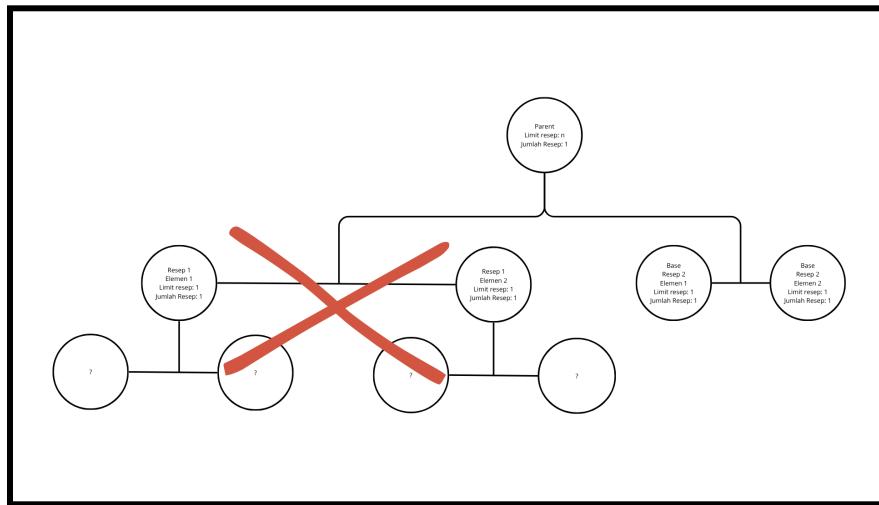
Gambar 3.2.9. Memasukkan semua elemen ke antrian
(sumber: dokumentasi pribadi)

Penelusuran selapis demi selapis pada akhirnya akan selesai setelah setiap cabang berakhir di leaf yang berupa elemen dasar. Penelusuran juga dapat berakhir lebih cepat

ketika terbentuk resep valid yang jumlahnya lebih dari batasan sesuai masukan pengguna. Pada kasus kedua, cabang yang belum lengkap (belum menemukan daun elemen dasar) akan dipotong.



Gambar 3.2.10. Penelusuran BFS yang selesai karena semua cabang berakhir di daun elemen dasar
(sumber: dokumentasi pribadi)

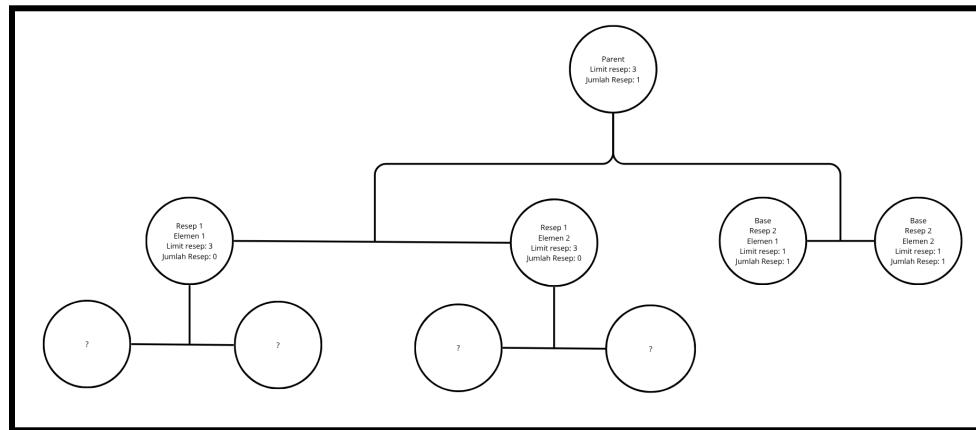


Gambar 3.2.11. Penelusuran BFS yang selesai karena jumlah resep valid sudah terpenuhi (1)
(sumber: dokumentasi pribadi)

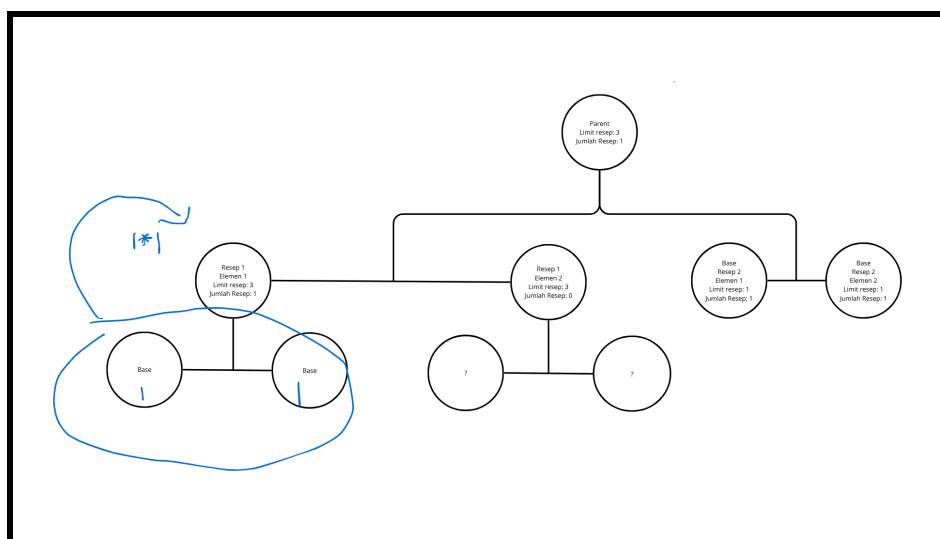
Untuk menghentikan pencarian ketika telah terbentuk resep valid sejumlah masukan, diperlukan sistem supaya simpul akar dapat mengetahui banyak resep yang telah terbentuk. Sistem tersebut dibuat dengan memanggil metode penghitungan jumlah resep ketika resep yang terdiri dari dua elemen dasar diproses.

Ketika resep yang terdiri dari dua elemen dasar diproses, jumlah resep milik elemen yang merupakan parent dari resep tersebut diperbarui dengan menghitung ulang sum of products anak-anaknya. Ketika jumlah resep elemen tersebut diperbarui, parent

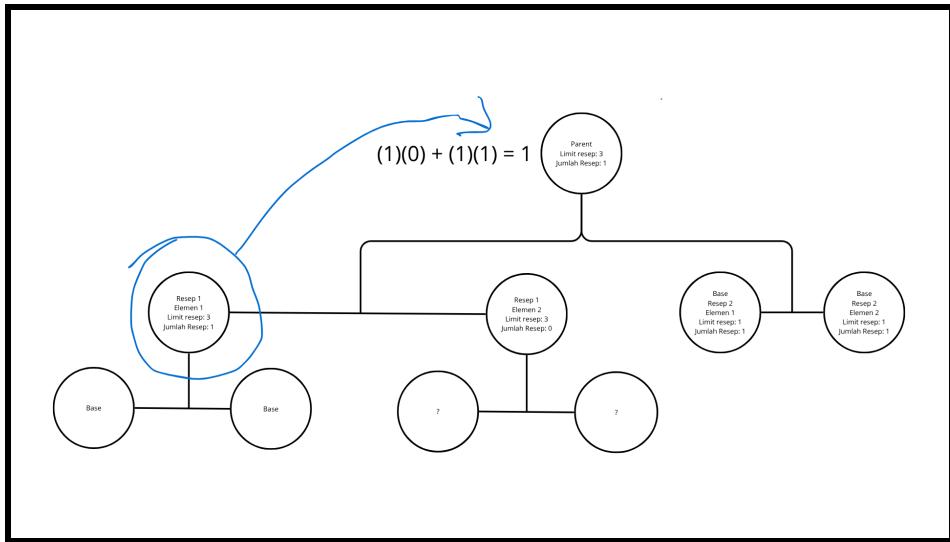
dari elemen tersebut juga perlu diperbarui sehingga metode perhitungan perlu berjalan secara rekursif hingga akar.



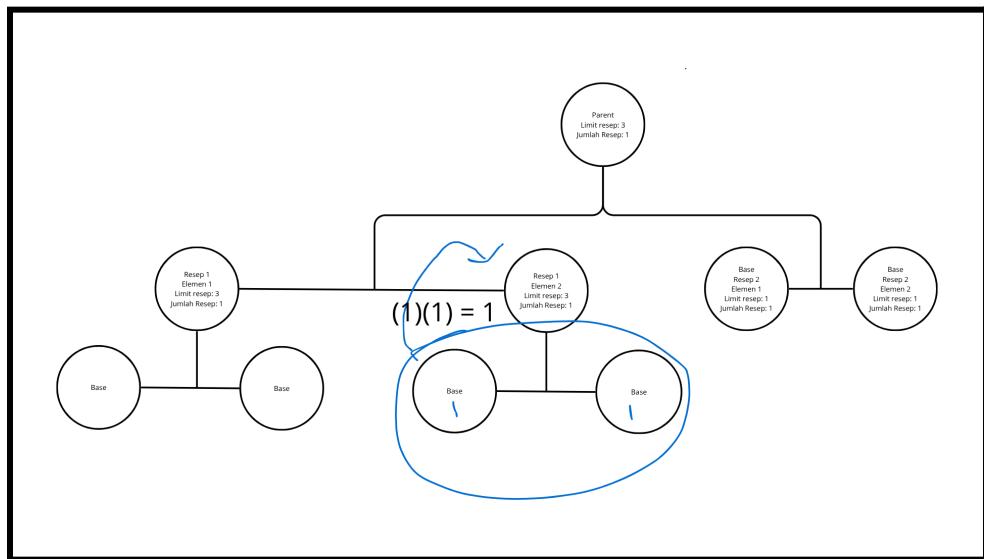
Gambar 3.2.12. Keadaan setelah penelusuran BFS di level kedua
(sumber: dokumentasi pribadi)



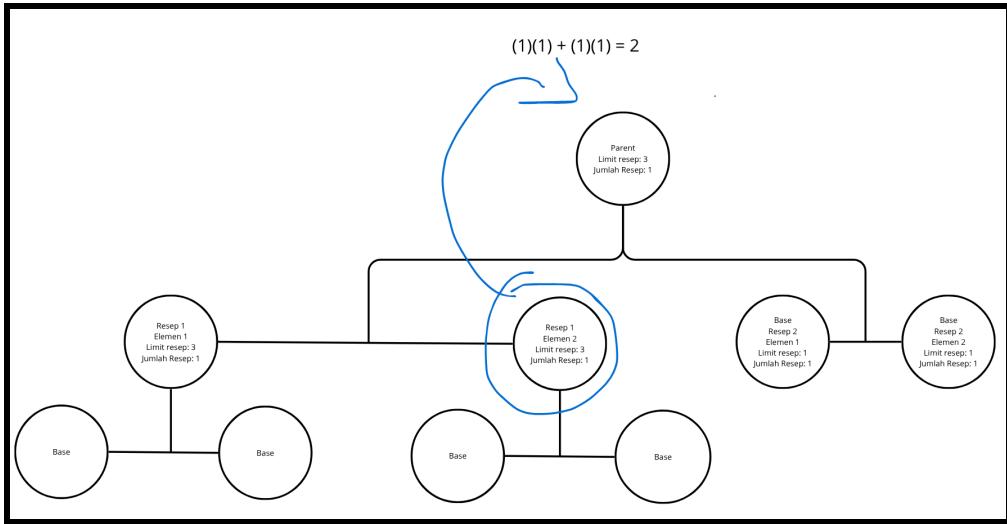
Gambar 3.2.13. Keadaan setelah menemukan resep yang terdiri dari dua elemen dasar
(sumber: dokumentasi pribadi)



Gambar 3.2.14. Pemanggilan pembaruan (perhitungan ulang) jumlah resep parent
(sumber: dokumentasi pribadi)



Gambar 3.2.15. Keadaan setelah menemukan resep yang terdiri dari dua elemen dasar
(sumber: dokumentasi pribadi)



Gambar 3.2.16. Pemanggilan pembaruan (perhitungan ulang) jumlah resep parent
(sumber: dokumentasi pribadi)

3.3. Fitur Fungsional dan Arsitektur Aplikasi Web Astigmatism Recipe Finder

Aplikasi web Astigmatism Recipe Finder memiliki fitur fungsional utama berupa sistem Start Menu yang meniru tampilan dan interaksi seperti pada sistem operasi Windows. Melalui Start Menu ini, pengguna dapat mengatur berbagai parameter pencarian, seperti memilih elemen target yang ingin dicari resepnya, menentukan algoritma pencarian (misalnya Breadth First Search atau Depth First Search), serta menetapkan batasan jumlah resep atau kedalaman pencarian. Selain itu, aplikasi ini juga dilengkapi dengan sistem window interaktif yang memungkinkan setiap hasil pencarian atau visualisasi pohon resep ditampilkan dalam jendela tersendiri. Jendela ini bersifat resizable dan dapat dipindah-pindah, memberi pengguna fleksibilitas untuk mengatur tampilan antarmuka sesuai kebutuhan mereka, layaknya menggunakan sistem operasi grafis pada desktop. Kombinasi dari kedua fitur ini memberikan pengalaman pengguna yang intuitif, dinamis, dan interaktif dalam mengeksplorasi resep elemen di Little Alchemy 2.

3.4. Contoh Ilustrasi Kasus

Untuk memperjelas analisis pemecahan masalah, kami akan membuat sebuah ilustrasi kasus dengan penyelesaiannya. Tinjau graf sampel berikut:

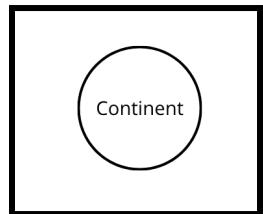
Elemen	Resep
Continent	(Land, Land), (Earth, Land)
Land	(Earth, Earth)

Earth	-
-------	---

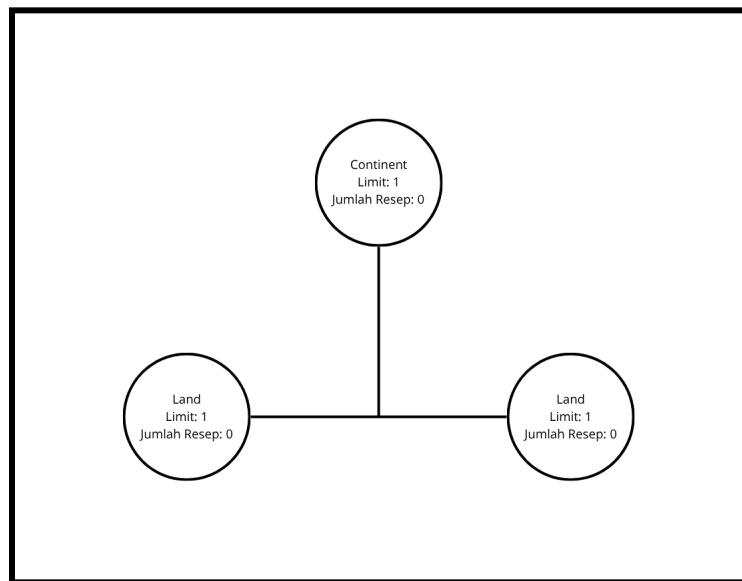
Pada contoh ilustrasi kasus ini, pengguna memasukkan Continent sebagai target elemen yang resepnya ingin dicari sebanyak satu.

3.4.1. DFS

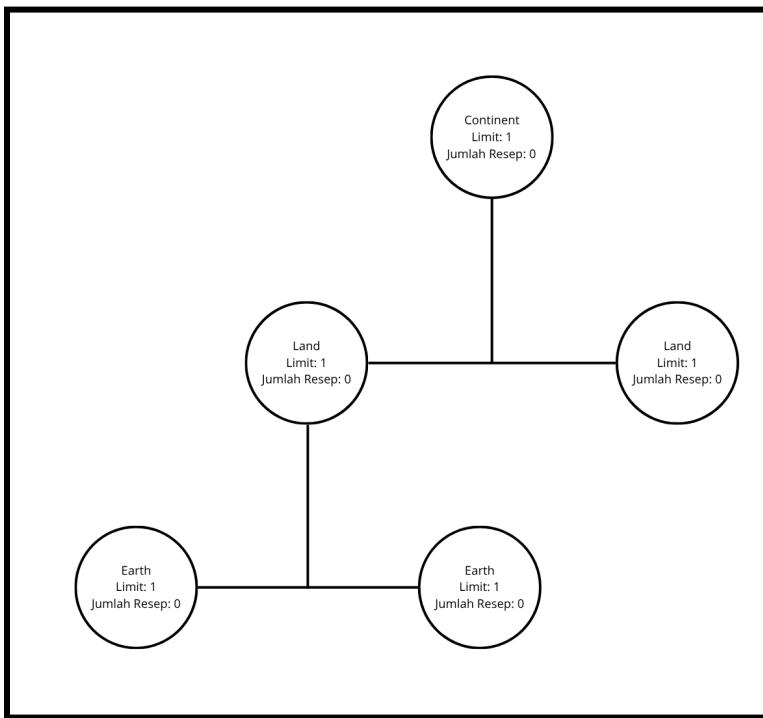
1. Continent menjadi root dari pohon yang akan dibuat



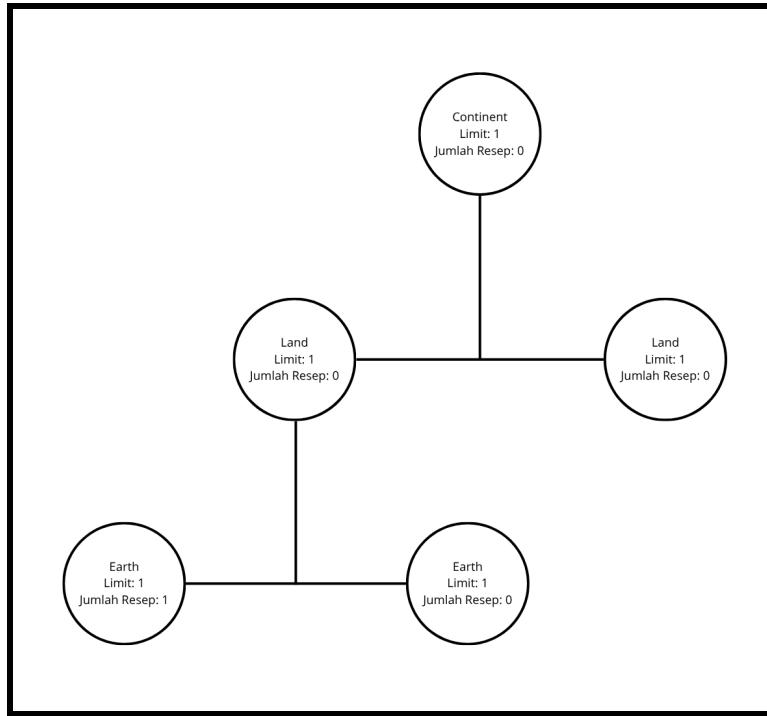
2. Resep pertama dari Continent dipilih



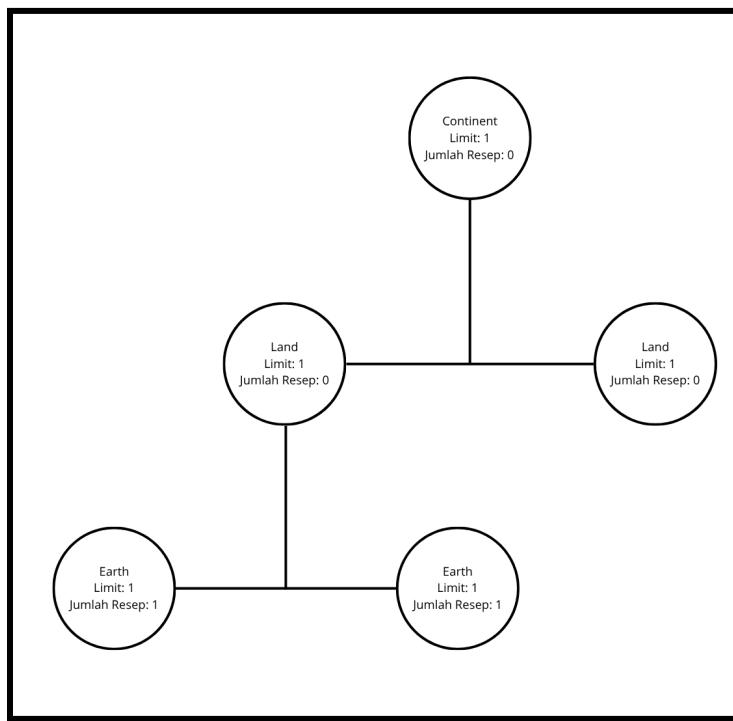
3. Elemen pertama (Land) dari resep tersebut diproses, resep pertama elemen (Land) tersebut dipilih



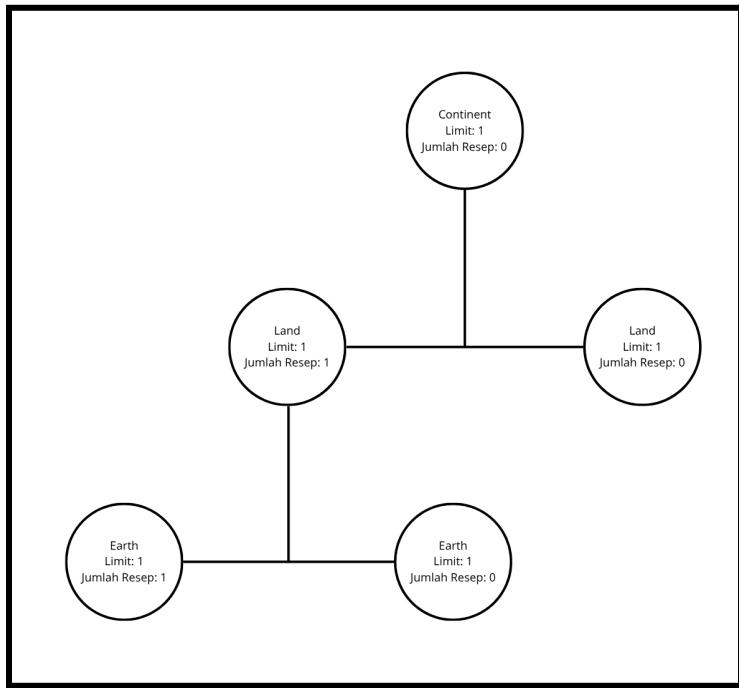
4. Elemen pertama (Earth) dari resep elemen Land diproses. Karena Earth merupakan elemen dasar, jumlah resepnya adalah satu.



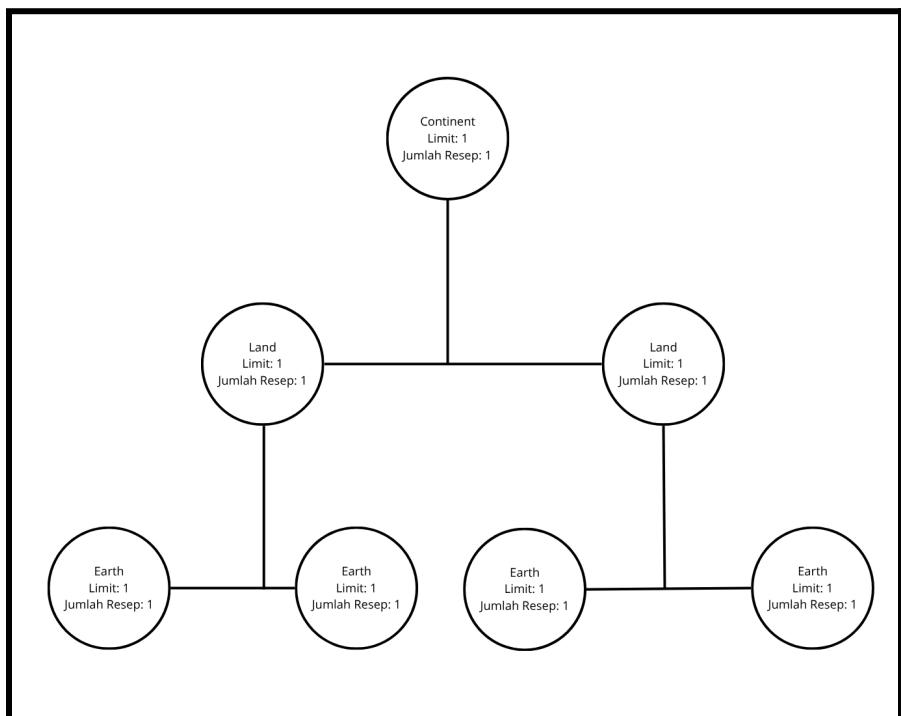
5. Elemen kedua (Earth) dari resep elemen Land diproses. Karena Earth merupakan elemen dasar, jumlah resepnya adalah satu.



6. Jumlah resep valid elemen pertama (Land) dihitung dari sum of products anak-anaknya.



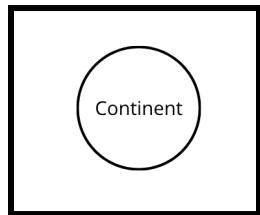
7. Elemen kedua (Land) dari resep Continent diproses, resep pertama elemen (Land) tersebut dipilih. Langkah-langkah selanjutnya sama seperti perlakuan elemen pertama (Land). Terakhir, jumlah resep Continent diperbarui.



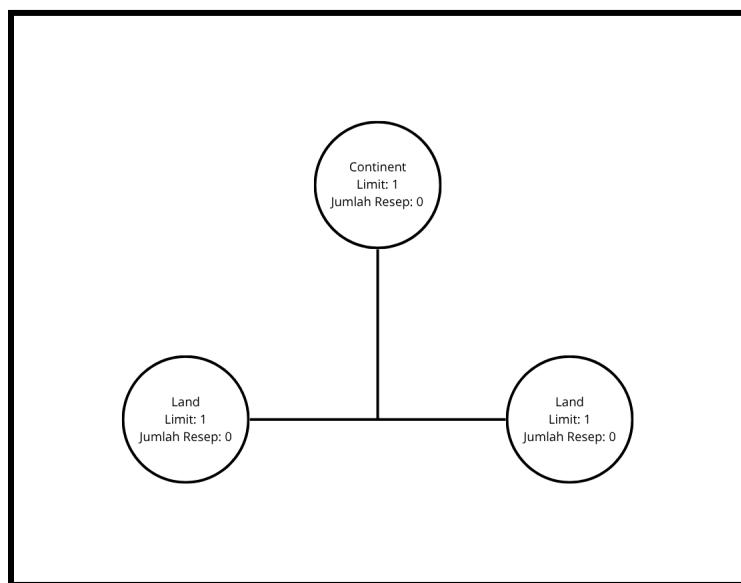
8. Pohon tersebut dikembalikan sebagai hasil penelusuran resep

3.4.2. BFS

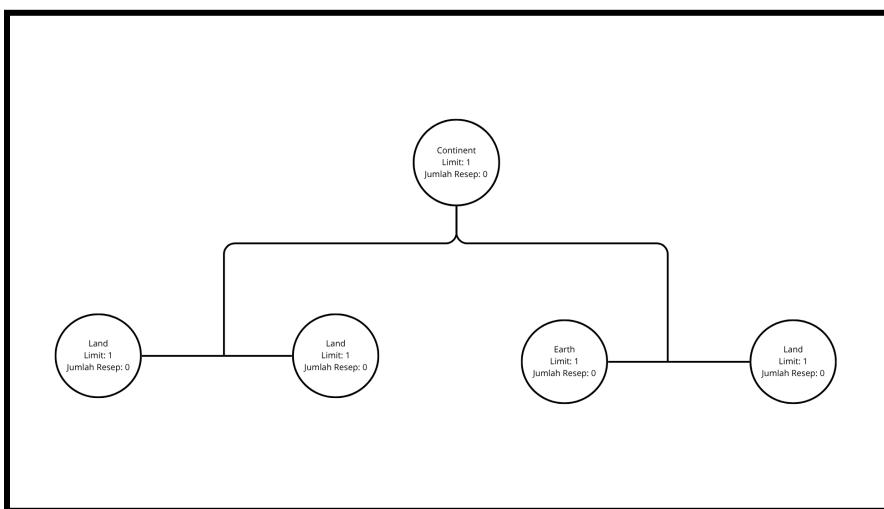
1. Iterasi pertama: Continent menjadi root dari pohon yang akan dibuat, dimasukkan ke queue kosong dan langsung diproses.



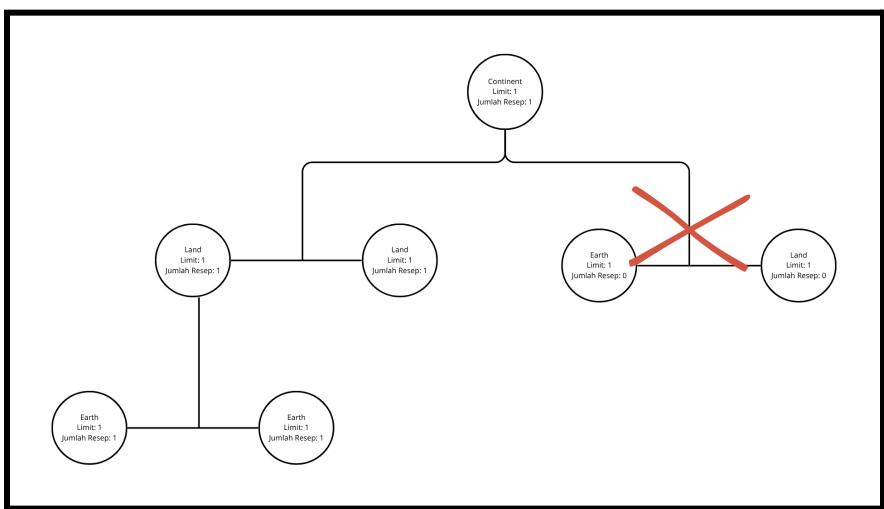
2. Iterasi kedua



3. Iterasi kedua



4. Iterasi ketiga



BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

1. Struktur Data

Struktur Data	Penjelasan
<pre>type RecipeType struct { ElementId int Element string ImgUrl string ImgUrl1 string ImgUrl2 string IngredientId1 int Ingredient1 string IngredientId2 int Ingredient2 string Tier int }</pre>	RecipeType digunakan untuk menyimpan resep setelah scraping data.
<pre>type PairElement struct { Element1 int Element2 int }</pre>	PairElement digunakan untuk mengelompokkan 2 elemen yang membentuk suatu resep
<pre>type ResultType struct { Tree *TreeNode `json:"Tree"` AccessedNodes uint64 `json:"AccessedNodes"` Time uint64 `json:"Time"` }</pre>	ResultType digunakan untuk mengembalikan hasil penelusuran
<pre>type MetaMapType struct {</pre>	MetaMapType digunakan untuk

```

    ElementList []string
`json:"ElementList"`
    IdNameMap map[int]string
`json:"IdNameMap"`
    NameIdMap map[string]int
`json:"NameIdMap"`
    NameImgMap map[string]string
`json:"NameImgMap"`
    IdImgMap map[int]string
`json:"IdImgMap"`
    NameTierMap map[string]int
`json:"NameTierMap"`
    IdTierMap map[int]int
`json:"IdTierMap"`
}

```

memetakan id elemen dengan metadatanya

```

type TreeNode struct {
    Parent      *TreeNode
`json:"-"`
    Element     int
`json:"element"`
    Visited     *VisitedPath
`json:"-"`
    Children    []*PairNode
`json:"children,omitempty"`
    PossibleRecipes uint64
`json:"-"`
}

```

TreeNode adalah struktur data utama yang merepresentasikan elemen pada tiap simpul di pohon hasil penelusuran. TreeNode menyimpan pointer ke parent untuk memanggil fungsi kalkulasi ulang jumlah resep, id elemen yang direpresentasikan, children dalam pasang-pasangan, dan jumlah resep valid yang terbentuk

```

type PairNode struct {
    Element1 *TreeNode
`json:"element1"`
    Element2 *TreeNode
`json:"element2"`
}

```

PairNode menyimpan pasangan TreeNode untuk merepresentasikan sebuah resep

4.2. Source Code

```
main.go
package main

import (
    "fmt"
    "net/http"

    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/server"
)

func main() {
    server := server.NewServer()

    err := server.ListenAndServe()
    if err != nil && err != http.ErrServerClosed {
        panic(fmt.Sprintf("http server error: %s", err))
    }
}
```

```
bfshandler.go
package handlers

import (
    "encoding/json"
    "log"
    "net/http"
    "os"
    "strconv"

    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/logic"
    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/models"
    "github.com/coder/websocket"
    "github.com/gin-gonic/gin"
)
```

```
func LimitedBFSTree(ctx *gin.Context) {
    // If user already scraped recipes
    // hasScraped, err := ctx.Cookie("scraped")
    // if err != nil || hasScraped != "true" {
    //     ctx.JSON(http.StatusUnauthorized, gin.H{"error": "You must scrape the
    data first"})
    //     return
    // }

    // Read recipes from JSON file
    filePath := "data/recipes.json"
    fileBytes, err := os.ReadFile(filePath)
    if err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to read
    recipes file"})
        return
    }

    // Deserialize recipes
    var recipes []models.RecipeType
    if err := json.Unmarshal(fileBytes, &recipes); err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to
    unmarshal recipes"})
        return
    }

    // Build graph
    graph := logic.BuildGraph(recipes)

    // Build Tier Map
    tierMap := logic.BuildTierMap(recipes)

    // Get target
    targetId, err := strconv.Atoi(ctx.DefaultQuery("target", ""))
    if targetId == 0 {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Target element is
    required"})
        return
    }
}
```

```
    } else if err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Problem when convert
target to integer"})
        return
    }

    // Get limit
    limit, err := strconv.ParseUint(ctx.DefaultQuery("limit", ""), 10, 64)
    if limit == 0 {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Limit element is
required"})
        return
    } else if err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Problem when convert
limit to integer"})
        return
    }

    result := logic.BuildLimitedBFSTree(targetId, graph, tierMap, limit)

    ctx.JSON(http.StatusOK, gin.H{
        "data": result,
    })
}

// For Live Update
func WebsocketBFSTreeHandler(c *gin.Context) {
    w := c.Writer
    r := c.Request

    socket, err := websocket.Accept(w, r, nil)
    if err != nil {
        log.Printf("could not open websocket: %v", err)
        http.Error(w, "could not open websocket",
http.StatusInternalServerError)
        return
    }
```

```
    defer socket.Close(websocket.StatusNormalClosure, "closing websocket")

    ctx := r.Context()

    targetId, err := strconv.Atoi(c.Query("target"))
    if err != nil || targetId == 0 {
        socket.Write(ctx, websocket.MessageText, []byte(`{"error":"Invalid or
missing target"}))
        return
    }

    limit, err := strconv.ParseUint(c.Query("limit"), 10, 64)
    if err != nil || limit == 0 {
        socket.Write(ctx, websocket.MessageText, []byte(`{"error":"Invalid or
missing limit"}))
        return
    }

    fileBytes, err := os.ReadFile("data/recipes.json")
    if err != nil {
        socket.Write(ctx, websocket.MessageText, []byte(`{"error":"Failed to
read recipes file"}))
        return
    }

    var recipes []models.RecipeType
    if err := json.Unmarshal(fileBytes, &recipes); err != nil {
        socket.Write(ctx, websocket.MessageText, []byte(`{"error":"Failed to
unmarshal recipes"}))
        return
    }

    graph := logic.BuildGraph(recipes)
    tierMap := logic.BuildTierMap(recipes)

    logic.StreamLimitedBFSTree(ctx, socket, targetId, graph, tierMap, limit)
}
```

[dfshandler.go](#)

package handlers

```
import (
    "encoding/json"
    "net/http"
    "os"
    "strconv"

    "github.com/andi-frame/Tubes2_astimatism/backend/internal/logic"
    "github.com/andi-frame/Tubes2_astimatism/backend/internal/models"
    "github.com/gin-gonic/gin"
)

func DFSTree(ctx *gin.Context) {
    // hasScraped, err := ctx.Cookie("scraped")
    // if err != nil || hasScraped != "true" {
    //     ctx.JSON(http.StatusUnauthorized, gin.H{"error": "You must scrape the
    data first"})
    //     return
    // }

    fileBytes, err := os.ReadFile("data/recipes.json")
    if err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to read
recipes file"})
        return
    }

    var recipes []models.RecipeType
    if err := json.Unmarshal(fileBytes, &recipes); err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to
unmarshal recipes"})
        return
    }

    // Build graph dan metadata
```

```
graph := logic.BuildGraphDFS(recipes)
metaMap := logic.BuildElementMetaMap(recipes)

// Get target
targetIdStr := ctx.DefaultQuery("target", "")
targetId, err := strconv.Atoi(targetIdStr)
if err != nil {
    ctx.JSON(http.StatusBadRequest, gin.H{"error": "Target must be integer
(element id)"})
    return
}

tree := logic.BuildLimitedDFSTree(targetId, graph, metaMap, 1)

ctx.JSON(http.StatusOK, gin.H{
    "data": tree,
})
}

func LimitedDFSTree(ctx *gin.Context) {
    // hasScraped, err := ctx.Cookie("scraped")
    // if err != nil || hasScraped != "true" {
    //     ctx.JSON(http.StatusUnauthorized, gin.H{"error": "You must scrape the
data first"})
    //     return
    // }

    fileBytes, err := os.ReadFile("data/recipes.json")
    if err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to read
recipes file"})
        return
    }

    var recipes []models.RecipeType
    if err := json.Unmarshal(fileBytes, &recipes); err != nil {
        ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to
unmarshal recipes"})
    }
}
```

```
        return
    }

    // Build graph dan metadata
    graph := logic.BuildGraphDFS(recipes)
    metaMap := logic.BuildElementMetaMap(recipes)

    // Get target and limit
    targetIdStr := ctx.DefaultQuery("target", "")
    targetId, err := strconv.Atoi(targetIdStr)
    if err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Target must be integer
(element id)"})
        return
    }

    limit, err := strconv.ParseUint(ctx.DefaultQuery("limit", ""), 10, 64)
    if limit <= 0 {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Limit must be a
positive integer"})
        return
    } else if err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Problem when convert
limit to integer"})
        return
    }

    trees := logic.BuildLimitedDFSTree(targetId, graph, metaMap, limit)

    ctx.JSON(http.StatusOK, gin.H{
        "data": trees,
    })
}
```

Meta_map_handler.go

```
package handlers

import (
    "net/http"

    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/logic"
    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/models"
    "github.com/gin-gonic/gin"
)

type RecipeRequest struct {
    Recipes []models.RecipeType `json:"recipes"`
}

func MetaMap(ctx *gin.Context) {
    var req RecipeRequest

    if err := ctx.BindJSON(&req); err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request body"})
        return
    }

    metaMap := logic.BuildMetaMapFromRecipes(req.Recipes)

    ctx.JSON(http.StatusOK, gin.H{
        "data": metaMap,
    })
}
```

Scrapers_handler.go

```
package handlers

import (
    "fmt"
    "net/http"
```

```
"strings"

"github.com/andi-frame/Tubes2_astigmatism/backend/internal/models"
"github.com/gin-gonic/gin"
"github.com/gocolly/colly"
)

func getTier(index int) int {
    switch index {
    case 1:
        return 0
    case 2:
        // Skip (Ruins/Archeologist)
        return -1
    case 3:
        return 1
    case 4:
        return 2
    case 5:
        return 3
    case 6:
        return 4
    case 7:
        return 5
    case 8:
        return 6
    case 9:
        return 7
    case 10:
        return 8
    case 11:
        return 9
    case 12:
        return 10
    case 13:
        return 11
    case 14:
        return 12
```

```
        case 15:
            return 13
        case 16:
            return 14
        case 17:
            return 15
        default:
            return -1
    }
}

func ScrapeHandler(ctx *gin.Context) {
    mainURL :=
    "https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"
    mythsURL :=
    "https://little-alchemy.fandom.com/wiki/Elements_(Myths_and_Monsters)"

    var recipes []models.RecipeType
    mapTier := make(map[string]int)
    mapId := make(map[string]int)
    mapImgUrl := make(map[string]string)
    mapMythMonster := make(map[string]bool)

    elementCounter := 0
    recipeCounter := 0

    // Scrape Myths and Monsters
    mythCollector :=
    colly.NewCollector(colly.AllowedDomains("little-alchemy.fandom.com"))

    mythCollector.OnHTML("table.list-table", func(table *colly.HTMLElement) {
        // each element
        table.ForEach("tbody tr", func(_ int, h *colly.HTMLElement) {
            element := strings.TrimSpace(h.ChildText("td:first-of-type a"))
            if element == "" {
                return
            }
        })
    })
}
```

```
        mapMythMonster[element] = true
    })
})

mythCollector.OnRequest(func(r *colly.Request) {
    fmt.Println("Visiting ", r.URL)
})

mythCollector.OnError(func(r *colly.Response, e error) {
    fmt.Println("Error:", e.Error())
})

if err := mythCollector.Visit(mythsURL); err != nil {
    fmt.Printf("Error scraping Myths and Monsters: %s\n", err.Error())
}

// Scrape main elements, ignore Myths and Monsters
mainCollector :=
colly.NewCollector(colly.AllowedDomains("little-alchemy.fandom.com"))
tableIndex := 0

// each table (starting and tiers)
mainCollector.OnHTML("table.list-table", func(table *colly.HTMLElement) {
    tableIndex++
    tier := getTier(tableIndex)
    if tier < 0 || tier > 17 {
        return
    }

    // each element generated
    table.ForEach("tbody tr", func(_ int, h *colly.HTMLElement) {
        element := strings.TrimSpace(h.ChildText("td:first-of-type a"))
        if element == "" || element == "Time" || element == "Ruins" ||
element == "Archeologist" {
            return
        }

        // Skip Myths and Monsters
    })
})
```

```
if mapMythMonster[element] {
    // fmt.Printf("Skipping Myth/Monster element: %s\n", element)
    return
}

elementImgUrl := ""
if imgTag := h.DOM.Find("td:first-of-type span img");
imgTag.Length() > 0 {
    elementImgUrl, _ = imgTag.Attr("data-src")
}

elementCounter++
mapTier[element] = tier
mapId[element] = elementCounter
mapImgUrl[element] = elementImgUrl
// fmt.Printf("\nElement[%v]: %-10s | %v\n", elementCounter,
element, tier)

if element == "Earth" {
    r := models.RecipeType{
        ElementId: elementCounter,
        Element: element,
        ImgUrl: elementImgUrl,
        Tier: tier,
    }
    recipes = append(recipes, r)
    return
}

// each recipe to the element generated
h.ForEach("td:nth-of-type(2) li", func(_ int, li
*colly.HTMLElement) {
    aTags := li.DOM.Find("a")

    if aTags.Length() < 2 {
        return
    }
```

```

        imgUrl1, _ := aTags.Eq(0).Find("img").Attr("data-src")
        imgUrl2, _ := aTags.Eq(2).Find("img").Attr("data-src")
        ingredient1 := strings.TrimSpace(aTags.Eq(1).Text())
        ingredient2 := strings.TrimSpace(aTags.Eq(3).Text())

        if ingredient1 == "Time" || ingredient2 == "Time" ||
ingredient1 == "Ruins" || ingredient2 == "Ruins" || ingredient1 ==
"Archeologist" || ingredient2 == "Archeologist" {
            return
        }

        // Skip Myths and Monsters
        if mapMythMonster[ingredient1] || mapMythMonster[ingredient2] {
            // fmt.Printf("Skipping recipe with Myth/Monster
ingredients: %s + %s\n", ingredient1, ingredient2)
            return
        }

        recipeCounter++

        ingId1 := mapId[ingredient1]
        ingId2 := mapId[ingredient2]
        ing1 := ingredient1
        ing2 := ingredient2
        img1 := imgUrl1
        img2 := imgUrl2

        if ingId1 > ingId2 {
            ingId1, ingId2 = ingId2, ingId1
            ing1, ing2 = ing2, ing1
            img1, img2 = img2, img1
        }

        r := models.RecipeType{
            ElementId:      elementCounter,
            Element:       element,
            ImgUrl:        mapImgUrl[element],
            ImgUrl1:       img1,
    
```

```
        ImgUrl2:      img2,
        IngredientId1: ingId1,
        Ingredient1:   ing1,
        IngredientId2: ingId2,
        Ingredient2:   ing2,
        Tier:          tier,
    }
    recipes = append(recipes, r)

    // Testing
    // fmt.Printf("Recipe[%v]: %s + %s\n", recipeCounter,
r.Ingredient1, r.Ingredient2)
    // fmt.Printf("ImgUrl: %s\n", r.ImgUrl)
    // fmt.Printf("ImgUrl1: %s\n", r.ImgUrl1)
    // fmt.Printf("ImgUrl2: %s\n", r.ImgUrl2)
}
})
})
}

mainCollector.OnRequest(func(r *colly.Request) {
    fmt.Print("Visiting ", r.URL)
})

mainCollector.OnError(func(r *colly.Response, e error) {
    fmt.Println("Error:", e.Error())
})

if err := mainCollector.Visit(mainURL); err != nil {
    fmt.Print("Error scraping main URL: ", err.Error())
    ctx.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to
scrape main URL"})
    return
}

// Fix missing ingredient IDs
for i := range recipes {
    // Check IngredientId1
    if recipes[i].IngredientId1 == 0 && recipes[i].Ingredient1 != "" {
```

```
        if id, exists := mapId[recipes[i].Ingredient1]; exists {
            recipes[i].IngredientId1 = id
            // fmt.Printf("Fixed missing ID for ingredient: %s = %d\n",
recipes[i].Ingredient1, id)
        } else {
            elementCounter++
            mapId[recipes[i].Ingredient1] = elementCounter
            recipes[i].IngredientId1 = elementCounter
            // fmt.Printf("Assigned new ID for ingredient: %s = %d\n",
recipes[i].Ingredient1, elementCounter)
        }
    }

    // Check IngredientId2
    if recipes[i].IngredientId2 == 0 && recipes[i].Ingredient2 != "" {
        if id, exists := mapId[recipes[i].Ingredient2]; exists {
            recipes[i].IngredientId2 = id
            // fmt.Printf("Fixed missing ID for ingredient: %s = %d\n",
recipes[i].Ingredient2, id)
        } else {
            elementCounter++
            mapId[recipes[i].Ingredient2] = elementCounter
            recipes[i].IngredientId2 = elementCounter
            // fmt.Printf("Assigned new ID for ingredient: %s = %d\n",
recipes[i].Ingredient2, elementCounter)
        }
    }
}

// Set cookie for 1 day
ctx.SetCookie("scraped", "true", 86400, "/", "localhost", false, true)

// Return recipes data
ctx.JSON(http.StatusOK, gin.H{
    "data": recipes,
    "stats": gin.H{
        "totalElements": elementCounter,
        "totalRecipes": len(recipes),
    }
})
```

```
        "mythsFiltered": len(mapMythMonster),
    },
})
}
```

```
Base_logic.go
package logic

import (
    "sort"

    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/models"
)

/*
Graph used for Depth First Search
Output is
map<id, vector<pair<id,id>>
*/
func BuildGraphDFS(recipes []models.RecipeType) map[int][]models.PairElement {
    result := make(map[int][]models.PairElement)
    for _, r := range recipes {
        pair := models.PairElement{
            Element1: r.IngredientId1,
            Element2: r.IngredientId2,
        }
        result[r.ElementId] = append(result[r.ElementId], pair)
    }
    return result
}

// Bottom-Up Version:
// func BuildGraph(recipes []models.RecipeType) map[models.PairElement]int {
//     result := make(map[models.PairElement]int)
//     for _, r := range recipes {
```

```

//      pair := models.PairElement{
//          Element1: r.IngredientId1,
//          Element2: r.IngredientId2,
//      }
//      result[pair] = r.ElementId
//  }
//  return result
// }

// Top-Down Version:
func BuildGraph(recipes []models.RecipeType) map[int][]models.PairElement {
    result := make(map[int][]models.PairElement)
    for _, r := range recipes {
        pair := models.PairElement{
            Element1: r.IngredientId1,
            Element2: r.IngredientId2,
        }
        result[r.ElementId] = append(result[r.ElementId], pair)
    }
    return result
}

func BuildElementMetaMap(recipes []models.RecipeType)
map[int]models.ElementMeta {
    meta := make(map[int]models.ElementMeta)
    for _, r := range recipes {
        if _, exists := meta[r.ElementId]; !exists {
            meta[r.ElementId] = models.ElementMeta{
                Name:    r.Element,
                ImgUrl: r.ImgUrl,
                Tier:   r.Tier,
            }
        }
    }
    return meta
}

func IsBaseElement(id int) bool {

```

```
base := map[int]bool{
    1: true, 2: true, 3: true, 4: true,
}
return base[id]
}

func BuildTierMap(recipes []models.RecipeType) map[int]int {
    tierMap := make(map[int]int)

    for _, recipe := range recipes {
        tierMap[recipe.ElementId] = recipe.Tier
    }

    return tierMap
}

func BuildIdMap(recipes []models.RecipeType) map[string]int {
    idMap := make(map[string]int)

    for _, recipe := range recipes {
        idMap[recipe.Element] = recipe.ElementId
    }

    return idMap
}

func BuildMetaMapFromRecipes(recipes []models.RecipeType) models.MetaMapType {
    meta := models.MetaMapType{
        ElementList: make([]string, 0),
        IdNameMap:   make(map[int]string),
        NameIdMap:   make(map[string]int),
        NameImgMap:  make(map[string]string),
        IdImgMap:   make(map[int]string),
        NameTierMap: make(map[string]int),
        IdTierMap:   make(map[int]int),
    }

    for _, recipe := range recipes {
```

```

// Element
if _, exists := meta.IdNameMap[recipe.ElementId]; !exists {
    meta.IdNameMap[recipe.ElementId] = recipe.Element
    meta.NameIdMap[recipe.Element] = recipe.ElementId
    meta.NameImgMap[recipe.Element] = recipe.ImgUrl
    meta.IdImgMap[recipe.ElementId] = recipe.ImgUrl
    meta.NameTierMap[recipe.Element] = recipe.Tier
    meta.IdTierMap[recipe.ElementId] = recipe.Tier
}
}

for _, recipe := range recipes {
    // Ingredient 1
    if _, exists := meta.IdNameMap[recipe.IngredientId1]; !exists {
        meta.IdNameMap[recipe.IngredientId1] = recipe.Ingredient1
        meta.NameIdMap[recipe.Ingredient1] = recipe.IngredientId1
        meta.NameImgMap[recipe.Ingredient1] = recipe.ImgUrl1
        meta.IdImgMap[recipe.IngredientId1] = recipe.ImgUrl1
    }

    // Ingredient 2
    if _, exists := meta.IdNameMap[recipe.IngredientId2]; !exists {
        meta.IdNameMap[recipe.IngredientId2] = recipe.Ingredient2
        meta.NameIdMap[recipe.Ingredient2] = recipe.IngredientId2
        meta.NameImgMap[recipe.Ingredient2] = recipe.ImgUrl2
        meta.IdImgMap[recipe.IngredientId2] = recipe.ImgUrl2
    }
}

ids := make([]int, 0, len(meta.IdNameMap))
for id := range meta.IdNameMap {
    ids = append(ids, id)
}
sort.Ints(ids)

for _, id := range ids {
    if id == 0 {
        continue
    }
}

```

```
        }

        meta.ElementList = append(meta.ElementList, meta.IdNameMap[id])

    }

    return meta
}
```

```
Bfs_tree.go
package logic

import (
    "sync"
    "sync/atomic"
    "time"

    "github.com/andi-frame/Tubes2_astimatism/backend/internal/models"
)

// Memory pools
var treeNodePool = sync.Pool{
    New: func() any {
        return new(modelsTreeNode)
    },
}

var pairNodePool = sync.Pool{
    New: func() any {
        return new(modelsPairNode)
    },
}

func BuildLimitedBFSTree(targetId int, elementsGraph
map[int][]modelsPairElement, tierMap map[int]int, limit uint64)
*models.ResultType {
    start := time.Now()
```

```
var accessedNodes uint64 = 0
root := treeNodePool.Get().(*models.TreeNode)
*root = models.TreeNode{
    Element: targetId,
    Visited: &models.VisitedPath{Val: targetId, Prev: nil},
}

queue := []*models.TreeNode{root}
var mu sync.Mutex

// Visit each node
for len(queue) > 0 && root.PossibleRecipes < limit {
    var batch []*models.TreeNode
    mu.Lock()
    batchSize := min(len(queue), 10)
    batch = queue[:batchSize]
    queue = queue[batchSize:]
    mu.Unlock()

    // Process batch
    var wg sync.WaitGroup
    newNodesChan := make(chan *models.TreeNode, 100)

    for _, curr := range batch {
        atomic.AddUint64(&accessedNodes, 1)

        if IsBaseElement(curr.Element) {
            continue
        }

        wg.Add(1)
        go func(currentNode *models.TreeNode) {
            defer wg.Done()

            var localNewNodes []*models.TreeNode

            recipes := elementsGraph[currentNode.Element]
            currTier := tierMap[currentNode.Element]
```

```
        for _, recipe := range recipes {
            // Check Tier
            if currTier <= tierMap[recipe.Element1] || currTier <=
tierMap[recipe.Element2] {
                continue
            }

            // Check infinite loop
            mu.Lock()
            hasLoop := HasVisited(currentNode.Visited, recipe.Element1)
            ||
            HasVisited(currentNode.Visited, recipe.Element2)
            mu.Unlock()

            if hasLoop {
                continue
            }

            // First ingredient
            child1 := treeNodePool.Get().(*models.TreeNode)
            *child1 = models.TreeNode{
                Parent: currentNode,
                Element: recipe.Element1,
                Visited: &models.VisitedPath{Val: recipe.Element1,
Prev: currentNode.Visited},
            }
            isChild1Base := IsBaseElement(child1.Element)
            if isChild1Base {
                child1.PossibleRecipes = 1
            }

            // Second ingredient
            child2 := treeNodePool.Get().(*models.TreeNode)
            *child2 = models.TreeNode{
                Parent: currentNode,
                Element: recipe.Element2,
                Visited: &models.VisitedPath{Val: recipe.Element2,
```

```
Prev: currentNode.Visited},
    }
    isChild2Base := IsBaseElement(child2.Element)
    if isChild2Base {
        child2.PossibleRecipes = 1
    }

    // Create pair node
    pairNode := pairNodePool.Get().(*models.PairNode)
    *pairNode = models.PairNode{
        Element1: child1,
        Element2: child2,
    }

    // Add children to current node
    mu.Lock()
    originalLen := len(currentNode.Children)
    currentNode.Children = append(currentNode.Children,
pairNode)
    mu.Unlock()

    if isChild1Base && isChild2Base {
        mu.Lock()

        success := calculatePossibleRecipes(currentNode, limit)

        if !success {
            currentNode.Children =
currentNode.Children[:originalLen]
            calculatePossibleRecipes(currentNode, limit)
        }

        mu.Unlock()
    } else {
        if !isChild1Base {
            localNewNodes = append(localNewNodes, child1)
        }
        if !isChild2Base {
```

```
                localNewNodes = append(localNewNodes, child2)
            }
        }
    }

    for _, node := range localNewNodes {
        newNodesChan <- node
    }
}(curr)
}

go func() {
    wg.Wait()
    close(newNodesChan)
}()

// Collect all new nodes from channel
var newNodes []*models.TreeNode
for node := range newNodesChan {
    newNodes = append(newNodes, node)
}

// Add new nodes to queue
mu.Lock()
queue = append(queue, newNodes...)
mu.Unlock()
}

mu.Lock()
PruneTree(root)
mu.Unlock()

elapsed := time.Since(start)

return &models.ResultType{
    Tree:          root,
    AccessedNodes: accessedNodes,
    Time:          uint64(elapsed.Milliseconds()),
```

```
    }

}

func HasVisited(path *models.VisitedPath, id int) bool {
    for p := path; p != nil; p = p.Prev {
        if p.Val == id {
            return true
        }
    }
    return false
}

func calculatePossibleRecipes(node *modelsTreeNode, limit uint64) bool {
    if node.PossibleRecipes >= limit {
        return false
    }

    if node == nil {
        return true
    }

    if len(node.Children) == 0 {
        node.PossibleRecipes = 1
        return true
    }

    var total uint64 = 0
    for _, pair := range node.Children {
        total += pair.Element1.PossibleRecipes * pair.Element2.PossibleRecipes
    }

    node.PossibleRecipes = total

    if node.Parent != nil {
        return calculatePossibleRecipes(node.Parent, limit)
    }

    return true // ?
}
```

```
}

func PruneTree(node *modelsTreeNode) {
    if node == nil || len(node.Children) == 0 {
        return
    }

    prunedChildren := make([]modelsPairNode, 0)
    for _, pair := range node.Children {
        PruneTree(pair.Element1)
        PruneTree(pair.Element2)

        isE1Valid := IsBaseElement(pair.Element1.Element) ||
len(pair.Element1.Children) > 0
        isE2Valid := IsBaseElement(pair.Element2.Element) ||
len(pair.Element2.Children) > 0

        if isE1Valid && isE2Valid {
            prunedChildren = append(prunedChildren, pair)
        } else {
            treeNodePool.Put(pair.Element1)
            treeNodePool.Put(pair.Element2)
            pairNodePool.Put(pair)
        }
    }

    node.Children = prunedChildren
}
```

```
Bfs_ws.go
package logic

import (
    "context"
    "encoding/json"
```

```
"sync"
"sync/atomic"
"time"

"github.com/andi-frame/Tubes2_astimatism/backend/internal/models"
"github.com/coder/websocket"
)

func StreamLimitedBFSTree(ctx context.Context, conn *websocket.Conn, targetId
int, elementsGraph map[int][]models.PairElement, tierMap map[int]int, limit
uint64) {
    startTime := time.Now()
    var accessedNodes uint64 = 0

    root := treeNodePool.Get().(*models.TreeNode)
    *root = models.TreeNode{
        Element: targetId,
        Visited: &models.VisitedPath{Val: targetId, Prev: nil},
    }

    queue := []*models.TreeNode{root}
    var mu sync.Mutex

    for len(queue) > 0 && root.PossibleRecipes < limit {
        batchSize := min(len(queue), 10)
        batch := queue[:batchSize]
        queue = queue[batchSize:]

        var wg sync.WaitGroup
        newNodesChan := make(chan *models.TreeNode, 100)

        for _, curr := range batch {
            atomic.AddUint64(&accessedNodes, 1)
            if IsBaseElement(curr.Element) {
                continue
            }

            wg.Add(1)
```

```
go func(currentNode *models.TreeNode) {
    defer wg.Done()

    var localNewNodes []*models.TreeNode

    recipes := elementsGraph[currentNode.Element]
    currTier := tierMap[currentNode.Element]

    for _, recipe := range recipes {
        if currTier <= tierMap[recipe.Element1] || currTier <=
tierMap[recipe.Element2] {
            continue
        }

        if HasVisited(currentNode.Visited, recipe.Element1) ||
HasVisited(currentNode.Visited, recipe.Element2) {
            continue
        }

        child1 := treeNodePool.Get().(*models.TreeNode)
        *child1 = models.TreeNode{
            Parent: currentNode,
            Element: recipe.Element1,
            Visited: &models.VisitedPath{Val: recipe.Element1,
Prev: currentNode.Visited},
        }
        if IsBaseElement(child1.Element) {
            child1.PossibleRecipes = 1
        }

        child2 := treeNodePool.Get().(*models.TreeNode)
        *child2 = models.TreeNode{
            Parent: currentNode,
            Element: recipe.Element2,
            Visited: &models.VisitedPath{Val: recipe.Element2,
Prev: currentNode.Visited},
        }
        if IsBaseElement(child2.Element) {
```

```

        child2.PossibleRecipes = 1
    }

    pairNode := pairNodePool.Get().(*models.PairNode)
    *pairNode = models.PairNode{
        Element1: child1,
        Element2: child2,
    }

    mu.Lock()
    originalLen := len(currentNode.Children)
    currentNode.Children = append(currentNode.Children,
pairNode)
    mu.Unlock()

    if IsBaseElement(child1.Element) &&
IsBaseElement(child2.Element) {
        mu.Lock()
        success := calculatePossibleRecipes(currentNode, limit)
        if !success {
            currentNode.Children =
currentNode.Children[:originalLen]
            calculatePossibleRecipes(currentNode, limit)
        }
        mu.Unlock()
    } else {
        if !IsBaseElement(child1.Element) {
            localNewNodes = append(localNewNodes, child1)
        }
        if !IsBaseElement(child2.Element) {
            localNewNodes = append(localNewNodes, child2)
        }
    }
}

for _, node := range localNewNodes {
    newNodesChan <- node
}

```

```
        }(curr)
    }

    go func() {
        wg.Wait()
        close(newNodesChan)
    }()

    var newNodes []*models.TreeNode
    for node := range newNodesChan {
        newNodes = append(newNodes, node)
    }

    // Send partial result update
    elapsed := time.Since(startTime).Milliseconds()
    result := models.ResultType{
        Tree:           root,
        AccessedNodes: accessedNodes,
        Time:          uint64(elapsed),
    }
    payload, _ := json.Marshal(result)
    conn.Write(ctx, websocket.MessageText, payload)

    mu.Lock()
    queue = append(queue, newNodes...)
    mu.Unlock()
}

PruneTree(root)

// Send final result
elapsed := time.Since(startTime).Milliseconds()
result := models.ResultType{
    Tree:           root,
    AccessedNodes: accessedNodes,
    Time:          uint64(elapsed),
}
payload, _ := json.Marshal(result)
```

```
    conn.Write(ctx, websocket.MessageText, payload)
}
```

```
Dfs_tree.go
package logic

import (
    "time"
    "github.com/andi-frame/Tubes2_astigmatism/backend/internal/models"
)

func BuildLimitedDFSTree(
    targetId int,
    recipeGraph map[int][]models.PairElement,
    metaMap map[int]models.ElementMeta,
    limit uint64,
) models.ResultType {
    start := time.Now()

    var accessed uint64 = 0
    var tree *modelsTreeNode
    done := make(chan *modelsTreeNode, 1)

    go func() {
        tree = buildLimitedDFSTreeHelper(targetId, recipeGraph, metaMap, limit,
nil, &accessed)
        done <- tree
    }()
    <-done

    duration := time.Since(start)

    return models.ResultType{
        Tree:          tree,
        AccessedNodes: accessed,
        Time:         uint64(duration.Nanoseconds()),
    }
}
```

```
    }

}

func buildLimitedDFSTreeHelper(
    targetId int,
    recipeGraph map[int][]models.PairElement,
    metaMap map[int]models.ElementMeta,
    remainingLimit uint64,
    parent *modelsTreeNode,
    accessed *uint64,
) *modelsTreeNode {
    *accessed++

    node := &modelsTreeNode{
        Element: targetId,
        Parent: parent,
    }

    pairs, exists := recipeGraph[targetId]
    if !exists || len(pairs) == 0 || remainingLimit <= 0 {
        node.PossibleRecipes = 1
        return node
    }

    tierTarget := metaMap[targetId].Tier

    for _, pair := range pairs {
        if remainingLimit <= 0 {
            break
        }

        tier1 := metaMap[pair.Element1].Tier
        tier2 := metaMap[pair.Element2].Tier
        if tier1 >= tierTarget || tier2 >= tierTarget {
            continue
        }

        left := buildLimitedDFSTreeHelper(pair.Element1, recipeGraph, metaMap,
```

```

remainingLimit, node, accessed)
    remaining := (remainingLimit + left.PossibleRecipes - 1) /
left.PossibleRecipes
    right := buildLimitedDFSTreeHelper(pair.Element2, recipeGraph, metaMap,
remaining, node, accessed)

    leftCount := left.PossibleRecipes
    rightCount := right.PossibleRecipes
    newComb := leftCount * rightCount

    remainingLimit -= newComb
    node.Children = append(node.Children, &models.PairNode{
        Element1: left,
        Element2: right,
    })
    node.PossibleRecipes += newComb
}

if len(node.Children) == 0 {
    node.PossibleRecipes = 1
}

return node
}

```

```

Recipe_tree.go
package models

type RecipeType struct {
    ElementId      int
    Element        string
    ImgUrl         string
    ImgUrl1        string
    ImgUrl2        string
    IngredientId1 int
    Ingredient1   string
}

```

```

    IngredientId2 int
    Ingredient2   string
    Tier          int
}

type PairElement struct {
    Element1 int
    Element2 int
}

type ResultType struct {
    Tree          *TreeNode `json:"Tree"`
    AccessedNodes uint64   `json:"AccessedNodes"`
    Time          uint64   `json:"Time"`
}

type MetaMapType struct {
    ElementList []string      `json:"ElementList"`
    IdNameMap   map[int]string `json:"IdNameMap"`
    NameIdMap   map[string]int  `json:"NameIdMap"`
    NameImgMap  map[string]string `json:"NameImgMap"`
    IdImgMap   map[int]string `json:"IdImgMap"`
    NameTierMap map[string]int  `json:"NameTierMap"`
    IdTierMap   map[int]int    `json:"IdTierMap"`
}

```

```

tree.go
package models

// Top-Down Version: (DFS)
type DFSNode struct {
    ElementId  int
    RecipeIndex int // indeks resep yang digunakan (dari 0 sampai
                    RecipeCount-1)
    RecipeCount int // total jumlah resep untuk elemen ini
    LeftChild   *DFSNODE
}

```

```
    RightChild *DFSNODE
    NodeCount int
}

type TreeNode struct {
    Parent      *TreeNode `json:"-"`
    Element     int       `json:"element"`
    Visited     *VisitedPath `json:"-"`
    Children    []*PairNode `json:"children,omitempty"`
    PossibleRecipes uint64   `json:"-"`
}

type PairNode struct {
    Element1 *TreeNode `json:"element1"`
    Element2 *TreeNode `json:"element2"`
}

type VisitedPath struct {
    Val int
    Prev *VisitedPath
}

// Bottom-Up Version: (unfinished)
// type TreeNode struct {
//     Elements []*RecipeNode `json:"elements"`
//     Children []*TreeNode   `json:"children"`
// }

type RecipeNode struct {
    Element     int       `json:"element"`
    Ingredient1 *RecipeNode `json:"ingredient1"`
    Ingredient2 *RecipeNode `json:"ingredient2"`
}

type ElementMeta struct {
    Name   string
    ImgUrl string
    Tier   int
}
```

```
}

type TreeResult struct {
    Element  string
    ImgUrl   string
    Children [] *PairResult
}

type PairResult struct {
    Element1 *TreeResult
    Element2 *TreeResult
}
```

4.3. Penjelasan Tata Cara Penggunaan Program

Berikut adalah antarmuka halaman utama program



1. Pengguna memilih elemen yang ingin dicari melalui kolom pencarian di bagian bawah.
2. Pengguna memilih metode pencarian *single* atau *multiple recipe*, menentukan jumlah resep yang diinginkan, serta memilih algoritma pencarian (BFS atau DFS)
3. Sistem memproses permintaan dan pengguna menunggu hasil pencarian ditampilkan.
4. Hasil ditampilkan

4.4. Hasil Pengujian

No	Input	Tangkapan Layar Hasil
----	-------	-----------------------

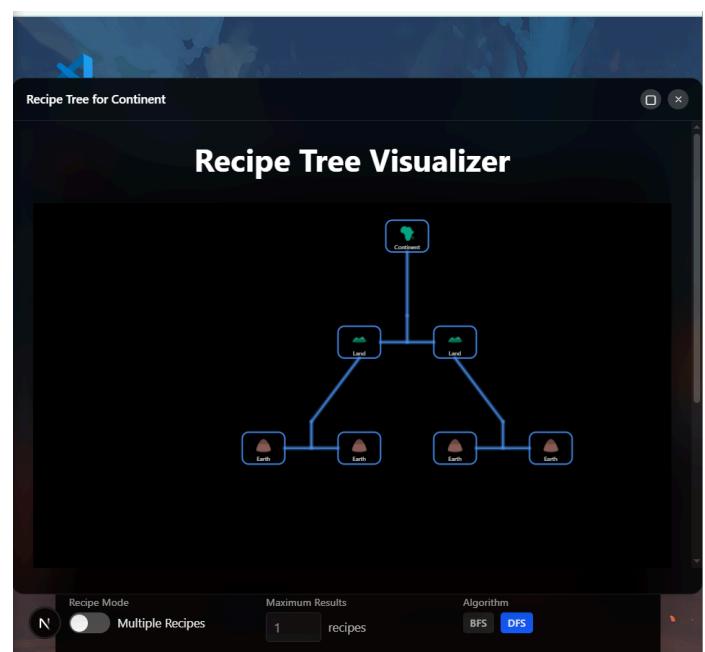
1

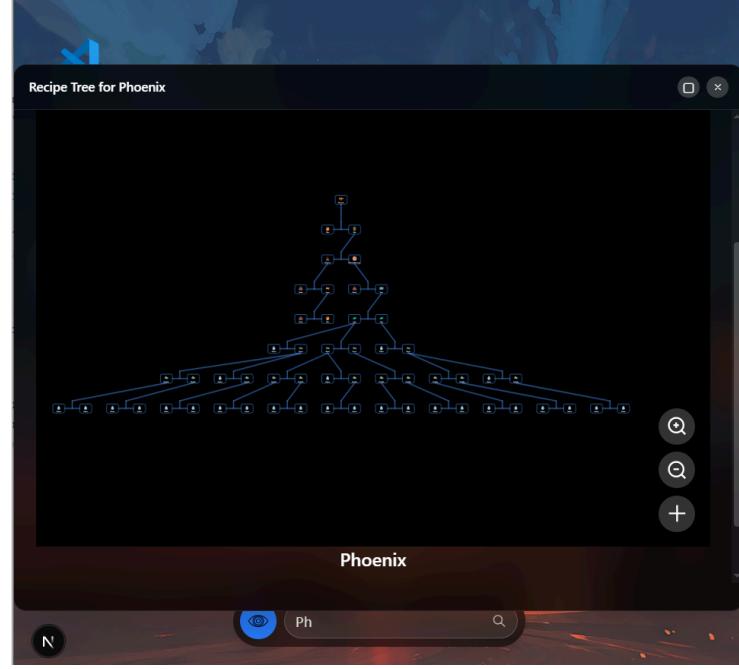
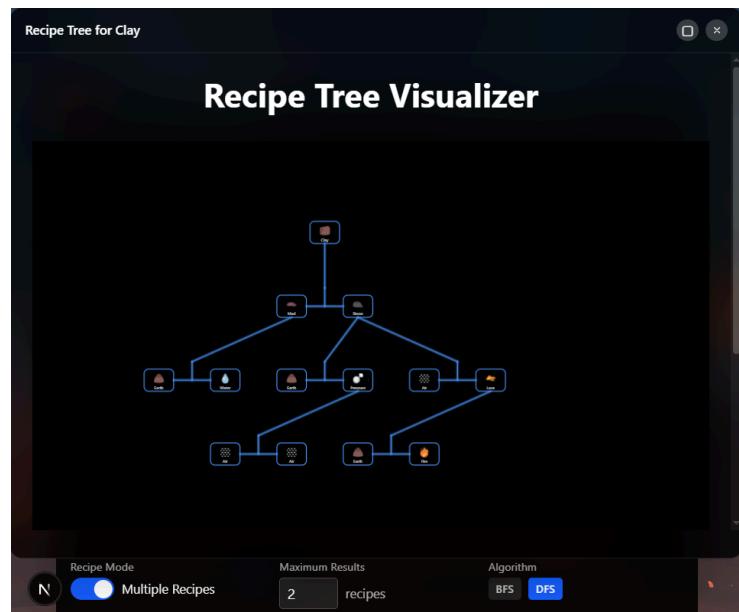
Continent, Single Recipe
BFS



2

Continent, Single Recipe
DFS



3	Phoenix, Multiple Recipe 5 BFS	
4	Clay, Multiple Recipe 2 DFS	

4.5. Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang dilakukan terhadap sistem pencarian resep elemen menggunakan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS), dapat disimpulkan bahwa kedua algoritma berjalan dengan lancar baik dalam mode *single recipe* maupun *multiple recipe*. Kedua pendekatan mampu menelusuri graf resep dan membangun pohon pencarian yang valid sesuai dengan batasan jumlah resep yang diminta pengguna.

Pada mode *single recipe*, baik BFS maupun DFS menunjukkan performa yang efisien dalam menemukan satu jalur resep menuju elemen target. BFS cenderung menemukan solusi dengan kedalaman minimum terlebih dahulu karena bersifat *level-order*, sehingga cocok untuk menemukan kombinasi resep yang paling sederhana atau paling pendek. Sementara itu, DFS menunjukkan keunggulan dalam pencarian yang bersifat mendalam terlebih dahulu, yang bisa berguna saat struktur graf sangat dalam namun tidak terlalu lebar.

Dalam mode *multiple recipe*, implementasi sistem yang mendukung pencarian paralel (*multithreaded*) berhasil memanfaatkan kekuatan dari kedua algoritma untuk mengeksplorasi berbagai kemungkinan resep dalam waktu yang relatif cepat. Sistem dapat menggabungkan beberapa hasil pencarian tanpa mengalami *overhead* signifikan, berkat penggunaan manajemen memori yang efisien dan struktur data seperti *memory pool* untuk node.

Secara keseluruhan, kedua algoritma telah berhasil diimplementasikan dengan baik dan mampu menghasilkan hasil pencarian yang akurat, cepat, dan optimal, sesuai dengan ekspektasi pengguna maupun tujuan dari sistem pencari resep dalam konteks aplikasi *Little Alchemy* ini.

BAB V KESIMPULAN

5.1. Kesimpulan

Berdasarkan implementasi dan analisis yang telah dilakukan dalam aplikasi yang telah kami buat, dapat disimpulkan beberapa hal berikut:

1. Permasalahan pencarian resep dalam permainan Little Alchemy 2 dapat dimodelkan secara efektif menggunakan *directed graph* berbasis *paired element*. Setiap simpul mewakili sebuah elemen hasil, dan setiap edge merepresentasikan pasangan bahan yang dapat membentuk elemen tersebut. Pemodelan ini memungkinkan aplikasi untuk menangani dependensi antar elemen yang bersifat dua arah.
2. Algoritma *Depth First Search* (DFS) berhasil diterapkan untuk menelusuri kemungkinan resep secara mendalam terlebih dahulu. DFS memberikan hasil yang eksploratif dan cocok untuk menemukan seluruh kombinasi resep. Namun, kontrol terhadap jumlah hasil resep sulit dilakukan karena sifatnya yang eksploratif.
3. Algoritma *Breadth First Search* (BFS) memberikan pendekatan yang lebih sistematis dan cocok untuk menemukan resep paling sederhana terlebih dahulu. BFS unggul dalam kontrol terhadap jumlah resep yang dihasilkan.
4. Aplikasi web yang dikembangkan memungkinkan pengguna untuk memilih metode pencarian, memilih target elemen, serta mengatur jumlah resep yang

diinginkan. Proses scraping data dari wiki dan penerapan logika algoritma berhasil diintegrasikan dengan baik.

5. Dari hasil pengujian, aplikasi mampu menangani berbagai permintaan pencarian resep dalam waktu yang relatif efisien. Namun, penggunaan memori dan waktu pencarian cenderung meningkat seiring dengan kedalaman dan kompleksitas elemen yang dicari.

5.2. Saran

Berikut beberapa saran yang dapat dipertimbangkan untuk mengembangkan dan meningkatkan aplikasi web kami:

1. BFS dan DFS dapat menjadi tidak efisien pada elemen yang sangat kompleks, perlu implementasi caching hasil pencarian atau teknik memoization agar menghindari pengulangan pencarian pada elemen yang sama.
2. Fitur penyimpanan hasil pencarian sebagai file (misalnya JSON atau PDF) akan sangat bermanfaat bagi pengguna yang ingin mendokumentasikan atau membagikan hasil pencarinya.

LAMPIRAN

6.1. Checklist Spesifikasi Program

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi dapat memperoleh data <i>recipe</i> melalui scraping.	✓	
3	Algoritma <i>Depth First Search</i> dan <i>Breadth First Search</i> dapat menemukan <i>recipe</i> elemen dengan benar.	✓	
4	Aplikasi dapat menampilkan visualisasi <i>recipe</i> elemen yang dicari sesuai dengan spesifikasi.	✓	
5	Aplikasi mengimplementasikan multithreading.	✓	
6	Membuat laporan sesuai dengan spesifikasi.	✓	
7	Membuat bonus video dan diunggah pada Youtube.	✓	
8	Membuat bonus algoritma pencarian <i>Bidirectional</i> .		✓
9	Membuat bonus <i>Live Update</i> .		✓
10	Aplikasi di- <i>containerize</i> dengan Docker.	✓	
11	Aplikasi di- <i>deploy</i> dan dapat diakses melalui internet.	✓	

6.1. Pranala

- Link Github Repository : [astimatism](#)
Link Deploy : tubes2-astimatism.vercel.app
Link Video : [video](#)

DAFTAR PUSTAKA

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

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