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

Disusun oleh:
Kelompok Bagikopiciumbuleuit
1. Muhammad Fithra Rizki/13523049
2. Sakti Bimasena/13523053
3. Aliya Husna Fayyaza/13523062

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

**2025**

# Bab 1
# Deskripsi Tugas

Gambar 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

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.

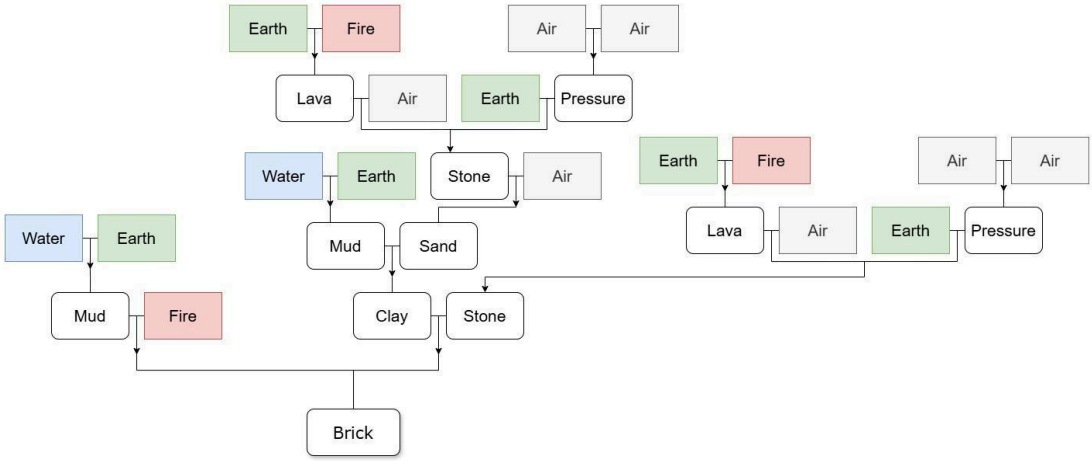
3. *Combine Mechanism*
   Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

Spesifikasi Wajib

- Buatlah aplikasi pencarian *recipe* elemen dalam permainan Little Alchemy 2 dengan menggunakan strategi **BFS dan DFS**.
- Tugas dikerjakan berkelompok dengan anggota **minimal 2 orang** dan **maksimal 3 orang**, boleh lintas kelas dan lintas kampus.
- Aplikasi berbasis **web**, untuk *frontend* dibangun menggunakan bahasa **Javascript** dengan *framework* **Next.js atau React.js**, dan untuk *backend* menggunakan bahasa **Golang**.
- Untuk *repository frontend* dan *backend* diperbolehkan **digabung** maupun **dipisah.**
- Untuk data elemen beserta resep dapat diperoleh dari ***scraping*** *website* Fandom Little Alchemy 2.
- Terdapat opsi pada *aplikasi* untuk **memilih algoritma** BFS atau DFS (juga *bidirectional* jika membuat bonus)
- Terdapat *toggle button* untuk memilih untuk menemukan **sebuah *recipe*** (Silahkan yang mana saja) atau **mencari banyak *recipe (multiple recipe)*** menuju suatu elemen tertentu. Apabila pengguna ingin mencari banyak *recipe* maka terdapat cara bagi pengguna untuk **memasukkan parameter banyak *recipe*** maksimal yang ingin dicari. Aplikasi boleh

mengeluarkan *recipe* apapun asalkan berbeda dan memenuhi banyak yang diinginkan pengguna (apabila mungkin).

- Mode pencarian *multiple recipe* wajib dioptimasi menggunakan ***multithreading.***
- Elemen yang digunakan pada suatu *recipe* harus berupa elemen dengan ***tier* lebih rendah** dari elemen yang ingin dibentuk.
- Aplikasi akan **memvisualisasikan *recipe*** yang ditemukan sebagai sebuah *tree* yang menunjukkan kombinasi elemen yang diperlukan dari elemen dasar. Agar lebih jelas perhatikan contoh berikut




Gambar 3. Contoh visualisasi *recipe* elemen

Gambar diatas menunjukkan contoh visualisasi *recipe* dari elemen *Brick*. Setiap elemen bersebelahan menunjukkan elemen yang perlu dikombinasikan. Amati bahwa *leaf* dari *tree* selalu berupa elemen dasar. Apabila dihitung, gambar diatas menunjukkan 5 buah *recipe* untuk *Brick* (karena *Brick* dapat dibentuk dengan kombinasi *Mud+Fire* atau *Clay+Stone*, begitu pula Stone yang dapat dibentuk oleh kombinasi *Lava+Air* atau *Earth+Pressure*). Visualisasi pada aplikasi tidak perlu persis seperti contoh diatas, tetapi pastikan bahwa *recipe* ditampilkan dengan jelas.

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

# Bab 2
# Landasan Teori

## 2.1 Penjelajahan Graf

Penjelajahan graf adalah proses sistematis untuk menelusuri simpul-simpul dalam struktur graf yang terdiri dari node (simpul) dan edge (sisi). Dalam konteks aplikasi *Little Alchemy 2*, setiap simpul mewakili sebuah elemen, sementara sisi menyatakan hubungan antara dua elemen yang dapat dikombinasikan menjadi elemen baru. Tujuan utama dari penelusuran graf pada proyek ini adalah untuk menemukan jalur kombinasi dari elemen-elemen dasar hingga menghasilkan elemen target. Karena banyaknya kemungkinan kombinasi antar elemen, diperlukan strategi penelusuran yang efisien untuk menemukan hasil yang diinginkan secara cepat dan akurat. Pemilihan algoritma penjelajahan graf yang tepat menjadi krusial dalam menentukan performa dan kualitas solusi yang ditemukan.

### 2.1.1 Algoritma Breadth First Search (BFS)

Breadth First Search (BFS) adalah algoritma penjelajahan graf yang mengeksplorasi simpul secara mendatar dari simpul awal, menjelajahi semua tetangga terlebih dahulu sebelum berpindah ke simpul yang lebih dalam. BFS menggunakan struktur antrian (*queue*) untuk mencatat urutan eksplorasi dan menjamin bahwa simpul dengan jarak minimum dari titik awal dievaluasi terlebih dahulu. Dalam konteks penelusuran graf, BFS memiliki keunggulan dalam hal *completeness*—yaitu selalu menemukan solusi selama faktor branching (b) terbatas—dan *optimality*, yaitu menjamin jalur terpendek ditemukan jika setiap langkah memiliki bobot biaya yang sama.

Kompleksitas waktu BFS adalah O(b^d), di mana b adalah jumlah rata-rata cabang (branching factor) dan *d* adalah kedalaman solusi. Ini terjadi karena BFS menjelajahi seluruh simpul pada level 0 hingga level *d*. Namun, kelemahan utama BFS terletak pada konsumsi memori yang besar, karena dalam skenario terburuk juga memerlukan ruang sebesar O(b^d), yang berarti seluruh simpul pada level terdalam perlu disimpan dalam memori secara simultan. Dengan demikian, walaupun BFS menjamin solusi optimal, efisiensi memori menjadi tantangan dalam graf yang sangat luas dan dalam.

### 2.1.2 Algoritma Depth First Search (DFS)

Depth First Search (DFS) adalah algoritma penjelajahan graf yang bekerja dengan prinsip menyelam sedalam mungkin ke jalur tertentu sebelum mundur dan mencoba jalur lain. DFS menggunakan struktur data *stack*, baik secara eksplisit maupun melalui rekursi, untuk menyimpan jejak penelusuran. Algoritma ini cocok digunakan ketika ingin menemukan solusi yang berada jauh dari simpul awal atau saat memori sangat terbatas, karena DFS hanya menyimpan satu jalur eksplorasi aktif pada satu waktu.

Dalam hal *completeness*, DFS tetap lengkap selama faktor percabangan (*branching factor*) *b* terbatas dan ada penanganan terhadap jalur yang redundan agar tidak terjebak dalam siklus atau eksplorasi yang sama berulang kali. Namun, DFS tidak menjamin *optimality*—artinya jalur pertama yang ditemukan bukan selalu jalur terpendek. Kompleksitas waktu DFS adalah O(b^m), di mana *m* adalah kedalaman maksimum dari graf. Dalam kasus terburuk, DFS bisa menjelajahi semua simpul hingga kedalaman maksimal sebelum menemukan solusi. Kompleksitas ruang DFS lebih hemat dibanding BFS, yakni sebesar O(bm), karena hanya perlu menyimpan satu jalur dari akar ke simpul terdalam yang sedang dieksplorasi. Oleh karena itu, DFS lebih efisien secara memori, namun berisiko menemukan solusi yang tidak optimal dan membutuhkan banyak waktu dalam kasus graf yang dalam atau dengan banyak cabang.

### 2.1.3 Algoritma Bidirectional Search

Algoritma *Bidirectional Search* merupakan teknik penelusuran graf yang dilakukan dari dua arah sekaligus, yaitu dari simpul awal (elemen dasar) menuju target, dan dari simpul target kembali ke elemen dasar. Pendekatan dua arah ini bertujuan untuk mempercepat proses pencarian dengan memperkecil ruang pencarian secara signifikan dibandingkan pencarian satu arah. Dalam proyek ini, algoritma bidirectional diimplementasikan dalam dua variasi, yaitu menggunakan Breadth First Search (BFS) dan Depth First Search (DFS). Pada *Bidirectional BFS*, kedua arah pencarian dilakukan dengan pendekatan melebar, sehingga seluruh kemungkinan kombinasi pada level tertentu dieksplorasi terlebih dahulu sebelum melangkah ke level berikutnya. Sedangkan pada *Bidirectional DFS*, pencarian dilakukan secara mendalam dari kedua arah, menelusuri jalur kombinasi hingga sedalam mungkin sebelum kembali dan mencoba jalur lain. Kombinasi dari pencarian dua arah ini memungkinkan sistem untuk mendeteksi titik temu (intersection) antara dua pencarian dan merekonstruksi jalur yang valid dari elemen awal hingga elemen target dengan lebih efisien.

## 2.2 Deployment

Deployment adalah proses pengiriman dan penyebaran aplikasi dari lingkungan pengembangan ke lingkungan produksi agar dapat digunakan oleh pengguna akhir. Tujuan utama dari deployment adalah memastikan bahwa aplikasi berjalan dengan stabil, dapat diakses sesuai kebutuhan, dan memenuhi ekspektasi performa. Proses deployment biasanya mencakup konfigurasi server, manajemen dependensi, pengaturan basis data, serta monitoring performa aplikasi setelah dirilis. Dalam pengembangan modern, deployment dapat dilakukan secara manual, semi-otomatis, atau otomatis menggunakan Continuous Integration/Continuous Deployment (CI/CD). Metode deployment yang baik dapat meningkatkan efisiensi pengiriman produk serta mengurangi risiko kesalahan yang muncul akibat perbedaan lingkungan antara pengembangan dan produksi.

## 2.3 Docker

Docker adalah platform open-source yang memungkinkan developer untuk mengemas aplikasi beserta semua dependensinya ke dalam sebuah unit yang disebut container. Container merupakan lingkungan terisolasi yang dapat dijalankan secara

konsisten di berbagai sistem operasi yang memiliki Docker Engine. Dengan menggunakan Docker, pengembang tidak perlu khawatir terhadap perbedaan versi pustaka, sistem operasi, atau konfigurasi server yang dapat menyebabkan aplikasi gagal berjalan. Docker mempermudah proses pengembangan, testing, hingga deployment karena memastikan bahwa aplikasi selalu berjalan dalam lingkungan yang sama dari awal hingga ke tahap produksi. Selain itu, Docker juga mendukung orkestrasi dan manajemen container secara skala besar melalui teknologi seperti Docker Compose dan Kubernetes, yang sangat berguna dalam membangun sistem berbasis microservices.

## 2.3 Penjelasan Singkat Web

Web berjudul Kejucraft adalah web yang dapat digunakan untuk mencari resep dari elemen-elemen yang ada di game Little Alchemy 2. Website ini dapat diakses melalui dua cara, yaitu langsung melalui situs web di https://kejucraft.vercel.app/site/formpage?type=bfs atau dijalankan secara lokal menggunakan Docker. Setelah aplikasi terbuka, pengguna dapat memilih algoritma pencarian yang diinginkan, yaitu BFS, DFS, atau Bidirectional. Jika memilih Bidirectional, pengguna akan diminta menentukan apakah algoritma yang ingin digunakan berbasis BFS atau DFS. Selanjutnya, pilih mode pencarian apakah ingin mencari satu jalur *crafting* (*single recipe*) atau beberapa jalur alternatif (*multiple recipe*). Masukkan nama elemen target yang ingin dicari jalur pembentukannya. Jika mode multiple dipilih, pengguna juga diminta mengisi parameter jumlah maksimal jalur yang ingin ditemukan. Setelah semua pilihan dikonfigurasi, klik tombol *Search* untuk memulai pencarian. Untuk melihat pembentukan tree secara langsung dan berulang, pengguna dapat mengaktifkan mode live update dengan menekan tombol merah.

# Bab 3
# Analisis Pemecahan Masalah

## 3.1 *Preprocessing* Data

Preprocessing data dalam proyek ini dimulai dari proses scraping halaman wiki *Little Alchemy 2* menggunakan pustaka goquery. Tujuannya adalah mengambil informasi setiap elemen yang tersedia dalam permainan, termasuk nama elemen, gambar ilustrasinya, kombinasi bahan (resep), dan tingkatan atau *tier* yang menunjukkan seberapa jauh elemen tersebut berada dalam progres permainan. Fungsi getDoc akan mengambil dokumen HTML dari URL target, lalu dilakukan pencarian terhadap berbagai elemen HTML seperti <h2>, <h3>, <table>, dan <ul> untuk mengidentifikasi nama elemen dan resepnya. Proses ini dilakukan secara bertahap, dengan menyusun informasi secara terstruktur ke dalam struct ElementRecipe. Nama-nama elemen dinormalisasi dengan fungsi normalizeName agar bisa disamakan formatnya, menghindari duplikasi akibat perbedaan kapitalisasi atau karakter khusus.

Selain nama dan resep, program juga mengambil URL gambar elemen dari atribut data-src, srcset, atau src dalam tag <img>. Untuk elemen yang berada di tier dasar seperti "Earth", ada penambahan manual karena tidak semua tier atau resep selalu konsisten tersedia di halaman yang di-scrape. Seluruh hasil akhir kemudian dikumpulkan ke dalam slice bertipe []ElementRecipe dan disimpan dalam file recipes.json dengan format JSON rapi menggunakan json.Encoderagar bisa dibaca ulang oleh program utama.

Setelah data tersimpan, fungsi LoadElements akan membaca file JSON recipes.json, lalu melakukan *unmarshal* agar data tersebut menjadi slice bertipe []ElementData. Data inilah yang akan menjadi dasar bagi fungsi PrepareElementMaps, yang kemudian menyusun tiga struktur penting: recipeMap (pemetaan elemen ke kombinasi bahan), tierMap (pemetaan elemen ke tingkatnya), dan basicElements (map boolean berisi elemen dasar seperti Air, Fire, Earth, dan Water). Dengan ketiga struktur tersebut, program memiliki fondasi yang kuat untuk melakukan pencarian jalur pembuatan elemen menggunakan algoritma seperti DFS, BFS, atau Bidirectional Search. Proses preprocessing ini memastikan bahwa data yang digunakan sudah bersih, konsisten, dan efisien untuk ditelusuri oleh algoritma di tahap selanjutnya.

## 3.2 Penerapan Penjelajahan Graf

### 3.2.1 Metode DFS (*Depth First Search)*

Algoritma DFS (Depth-First Search) yang digunakan dalam program ini bertujuan untuk mencari jalur pembuatan sebuah elemen target dari bahan-bahan dasar yang tersedia, dengan menelusuri kemungkinan kombinasi resep yang bisa menghasilkan elemen tersebut. Proses dimulai dari elemen target sebagai akar pencarian, lalu secara rekursif ditelusuri ke bawah ke arah bahan-bahannya, dan seterusnya hingga mencapai elemen-elemen dasar seperti Air, Earth, Fire, dan Water yang tidak perlu dibuat. Langkah-langkah *crafting* yang berhasil dikonstruksi dikumpulkan dalam bentuk Step, yang menyimpan dua bahan dan hasilnya, dan disusun menjadi

Path yang mewakili satu jalur lengkap. Algoritma ini menggunakan sistem validasi berdasarkan *tier*, yaitu tingkat kemajuan elemen dalam hierarki *crafting*. Maka dari itu, sebelum mencoba satu kombinasi resep, DFS akan memastikan bahwa *tier* dari elemen hasil lebih tinggi dari kedua bahan penyusunnya, sehingga jalur *crafting* selalu bergerak naik dan tidak pernah kembali ke bawah. Ini membuat kemungkinan siklus menjadi tidak mungkin terjadi. Selain itu, algoritma ini juga menggunakan struktur memo untuk mencatat apakah suatu elemen sudah pernah terbukti bisa atau tidak bisa dibuat dalam cabang sebelumnya. Jika suatu elemen sudah gagal dibuat sebelumnya, maka DFS bisa langsung melewati pencarian ulang tersebut, mempercepat proses pencarian dan mencegah eksplorasi redundan. Fungsi *findPathDFS* akan berhenti pada jalur valid pertama yang ditemukan, lalu mengembalikannya sebagai satu-satunya solusi.

Selanjutnya, untuk memenuhi tugas mencari satu resep dan banyak resep, digunakan dua fungsi utama yaitu *FindSingleRecipeDFS* dan *FindMultipleRecipesConcurrent*. Fungsi *FindSingleRecipeDFS* digunakan ketika hanya dibutuhkan satu jalur crafting dari elemen-elemen dasar menuju elemen target. Fungsi ini bekerja dengan menjalankan algoritma DFS sekali, dan langsung mengembalikan jalur pertama yang valid tanpa mengeksplorasi kemungkinan lain. Sementara itu, untuk menemukan lebih dari satu resep yang berbeda, digunakan fungsi *FindMultipleRecipesConcurrent*. Fungsi ini menjalankan pencarian secara paralel dengan beberapa variasi dataset resep yang telah diacak melalui *createRecipeVariation*. Dengan pendekatan *multithreading* ini, sistem mampu menemukan jalur-jalur crafting yang unik dengan cepat dan efisien. Setiap *thread* dijalankan secara terpisah dengan urutan resep berbeda, sehingga memungkinkan DFS menjelajahi jalur alternatif yang tidak ditemukan dalam pencarian tunggal. Hasil akhirnya dikumpulkan, di-filter agar tidak duplikat, lalu ditampilkan sebagai daftar solusi yang bervariasi.

Sebagai ilustrasi, misalnya kita ingin mencari jalur menuju elemen Barn, yang secara logika *crafting* berasal dari House + Field (satu-satunya kombinasi yang memenuhi). DFS pertama mungkin menemukan jalur seperti Water + Earth = Mud → Mud + Fire = Brick → Brick + Brick = Wall → Wall + Wall = House, lalu Air + Air = Pressure → Earth + Pressure = Stone → Stone + Fire = Metal → Earth + Metal = Plow → Plow + Earth = Field, hingga akhirnya House + Field = Barn. Namun, perlu dicatat bahwa baik House maupun Field bisa dibuat melalui berbagai urutan crafting yang berbeda. Misalnya, Wall sebagai bahan House bisa juga didapat dari jalur lain selain Brick, dan Field bisa dibuat dari jalur yang berbeda dalam pembentukan Plow atau Metal. Maka dari itu, dibutuhkan fungsi *createRecipeVariation* untuk mengacak peletakan *thread* pada simpul anak karena DFS akan selalu menyusuri kombinasi pertama yang tersedia dalam resep, tanpa pengacakan maka ia akan menghasilkan jalur yang sama setiap kali dijalankan. Dengan mengacak urutan resep pada setiap elemen menggunakan *seed* tertentu, DFS yang berjalan di tiap thread akan mengeksplorasi cabang *crafting* yang berbeda lebih dulu. Akibatnya, jalur yang ditemukan pun bisa bervariasi.

### 3.2.2 Metode Bidirectional BFS (*Breadth First Search)*

Fungsi BiSearchBFS mengimplementasikan algoritma pencarian dua arah (*bidirectional search*) dengan pendekatan Breadth-First Search (BFS). Pencarian dilakukan secara simultan dari dua arah yaitu maju dari elemen-elemen dasar melalui forwardQueue, dan mundur dari elemen target melalui backwardQueue. Setiap elemen yang dikunjungi ditandai melalui dua map forwardVisited dan backwardVisited, dan informasi jalur disimpan dalam bentuk parent serta langkah pembuatan.

Pencarian maju (*forward*) memproses setiap elemen dasar dan mencoba semua kombinasi resep (elements) yang mungkin bisa dibuat dari bahan-bahan yang telah tersedia. Sebuah elemen hasil (result) hanya akan dibuat jika dua bahan pembentuknya (ing1 dan ing2) telah ditemukan sebelumnya, dan tier dari kedua bahan tersebut lebih rendah dari tier hasil. Ketika result berhasil dibuat, sistem menyimpannya ke antrian BFS dan mencatat langkah *crafting*-nya. Jika result juga telah ditemukan dari arah backward, maka titik temu ditemukan dan jalur bisa direkonstruksi. Pencarian mundur (*backward*) berjalan dari elemen target ke bahan-bahan pembentuknya dengan memeriksa semua resep yang menghasilkan elemen saat ini. Jika bahan-bahan tersebut valid dan memiliki *tier* lebih rendah dari hasil, maka sistem menelusuri mundur ke bahan-bahan tersebut dan menyimpannya ke antrian BFS. Jika bahan tersebut telah ditemukan dari arah *forward*, maka jalur dapat digabung dan diproses untuk direkonstruksi menggunakan fungsi reconstructPath. Proses dijalankan secara bergantian (*alternating* BFS), dan akan berhenti saat ditemukan titik temu antara dua arah pencarian. Fungsi reconstructPath akan menyusun urutan crafting secara topologis dari elemen dasar menuju target berdasarkan semua langkah *crafting* (steps) yang telah dikumpulkan dari dua arah.

Untuk memenuhi kebutuhan pencarian jalur tunggal maupun multipel, digunakan dua fungsi utama yaitu FindSingleRecipedan BiSearchMultipleBFS. FindSingleRecipe berfungsi sebagai pembungkus yang memanggil salah satu dari dua algoritma (BiSearchBFS atau BiSearchDFS) tergantung parameter. Hasilnya berupa satu jalur *crafting* dari elemen dasar menuju target. Selanjutnya, BiSearchMultipleBFS digunakan untuk menemukan beberapa jalur *crafting* yang unik menuju elemen target. Karena BFS dua arah akan selalu menghasilkan jalur terpendek yang sama bila dijalankan berulang-ulang, BiSearchMultipleBFS mengatasi hal ini dengan melakukan pengacakan urutan resep (melalui shuffleRecipes) sebelum memulai pencarian. Dengan mengacak urutan resep dan menjalankan pencarian secara paralel menggunakan goroutine, fungsi ini dapat mengeksplorasi banyak kemungkinan jalur alternatif.

Setiap thread hasil eksplorasi akan mengirimkan kandidat jalurnya melalui channel. Untuk menjamin keunikan, setiap jalur dikonversi ke bentuk string menggunakan hashPath, yang menjadi kunci untuk mengecek duplikasi dalam pathSignatures. Hanya jalur dengan signature unik yang disimpan. Proses ini berlanjut hingga jumlah maksimum jalur unik (maxPaths) tercapai.Sebagai ilustrasi, forward BFS dapat memulai dari elemen dasar seperti Water, Earth, dan Fire, lalu menyusun elemen-elemen lanjutan seperti Mud, Brick, atau Wall. Di saat yang sama, backward BFS dari elemen target seperti Barn akan mundur melalui kombinasi seperti House + Field, dan seterusnya. Jika dua arah bertemu pada satu elemen yang sama, maka pencarian berhenti dan jalur bisa disusun ulang.

### 3.2.3 Metode Bidirectional DFS (*Depth Search First)*

Fungsi BiSearchDFS merupakan implementasi algoritma pencarian dua arah yang menggunakan pendekatan DFS secara bergantian, di mana sistem menelusuri jalur *crafting* dari dua arah secara bergantian pada setiap iterasi. Algoritma ini menggunakan struktur list sebagai *stack* untuk menyimpan status eksplorasi (SearchState) yang sedang dijalankan. Ada dua stack yang digunakan yaitu forwardStack untuk pencarian dari elemen-elemen dasar, dan backwardStack untuk pencarian dari elemen target. Struktur SearchState menyimpan elemen aktif yang sedang ditelusuri, bahan-bahan yang sudah tersedia (Available), serta langkah-langkah *crafting* (Steps) yang telah terbentuk sejauh ini. Pada inisialisasi pencarian, semua elemen dasar dimasukkan ke dalam forwardStack, sementara elemen target dimasukkan ke backwardStack. Selanjutnya, proses pencarian berjalan dalam satu *loop* tunggal yang akan terus berlangsung selama setidaknya salah satu dari stack masih memiliki elemen.

Pada proses *forward*, sistem mengambil satu elemen dari forwardStack dan mengecek apakah sudah ditemukan dari sisi backward. Jika sudah, maka langkah-langkah *crafting* dari kedua arah digabung dan divalidasi. Jika jalurnya lengkap dan valid, maka fungsi reconstructPath digunakan untuk menyusun urutan crafting dari awal ke target. Jika belum ditemukan oleh sisi lawan, sistem mencoba semua resep yang memungkinkan menggunakan elemen-elemen yang tersedia. Setiap resep yang valid akan menghasilkan state baru, yang kemudian ditambahkan ke stack untuk dieksplorasi lebih lanjut. Pada proses *backward* yang dijalankan setelah satu langkah forward, algoritma mengambil satu elemen dari backwardStack dan proses berjalan dari target ke bahan-bahan yang dapat membentuknya. Sistem mencari semua resep yang dapat menghasilkan elemen saat ini, kemudian memeriksa apakah bahan-bahannya valid dan belum dikunjungi. Jika memenuhi syarat, maka dibuat state baru dan bahan tersebut dimasukkan ke stack backward. Sama seperti pada proses forward, jika ditemukan titik temu di mana sebuah elemen telah dikunjungi oleh kedua arah, maka langkah-langkah dari kedua sisi digabung, divalidasi, dan diproses menjadi jalur *crafting* lengkap.

Selanjutnya, untuk memenuhi tugas pencarian satu maupun beberapa jalur *crafting*, digunakan beberapa fungsi tambahan. Untuk pencarian satu jalur saja, digunakan fungsi FindSingleRecipe, yang hanya bertugas memilih dan memanggil BiSearchDFS atau BiSearchBFS sesuai parameter. Selanjutnya, fungsi BiSearchMultipleDFS digunakan untuk menemukan beberapa jalur *crafting* unik menuju elemen target dengan menjalankan BiSearchDFS secara berulang. Setiap hasil yang ditemukan akan diperiksa keunikannya menggunakan signature dari urutan elemen dalam path (hashPath), dan hanya jalur yang belum pernah muncul sebelumnya yang disimpan. Untuk memastikan jalur yang dibentuk memang valid, digunakan fungsi isStepsComplete, yang mengecek bahwa semua bahan dalam langkah-langkah crafting berasal dari elemen dasar atau memiliki jalur *crafting*-nya sendiri. Jalur yang lolos validasi akan direkonstruksi secara topologis menggunakan fungsi reconstructPath. Sebagai ilustrasi, pada iterasi pertama, forward DFS mungkin mengeksplorasi Water dan menambahkan state untuk Water + Earth = Mud ke stack. Kemudian pada langkah backward di iterasi yang sama, algoritma mungkin menelusuri Barn ke House + Field. Pada iterasi berikutnya, forward DFS mungkin melanjutkan ke Mud + Stone = Clay, diikuti oleh backward DFS yang menelusuri House ke Brick + Wood. Dengan pola *alternating* ini, kedua pencarian dapat bertemu di suatu titik tengah (misalnya di Brick), sehingga menemukan jalur lengkap dari elemen dasar ke target.

### 3.2.4 Metode BFS (*Breadth First Search)*

Algoritma Breadth-First Search (BFS) dalam program ini digunakan untuk menemukan urutan langkah crafting untuk membentuk sebuah elemen *target* dari elemen-elemen dasar yang tersedia. Namun, berbeda dari pendekatan BFS klasik yang berjalan dari *start* ke *goal*, algoritma ini berjalan secara *reverse*, dimulai dari elemen *target*, dan dipecah mundur menjadi bahan-bahan penyusunnya berdasarkan data resep (*recipes*). Pada tiap langkahnya, algoritma menyimpan status berupa node yang merepresentasikan elemen-elemen yang masih harus dipecah (*Remaining*), langkah-langkah crafting yang sudah terkumpul (*Steps*), serta pencatatan elemen yang sudah dikunjungi untuk mencegah eksplorasi berulang dan siklus.

Proses pencarian dilakukan dengan memanfaatkan struktur antrian (*queue*) BFS, di mana pada setiap iterasi, algoritma mengambil satu node dan memeriksa apakah seluruh elemen di dalam *Remaining* sudah merupakan elemen dasar. Jika iya, maka proses crafting dianggap selesai dan jalur crafting dikembalikan sebagai solusi. Jika tidak, maka elemen non-dasar pertama dari *Remaining* akan dipilih untuk diekspansi. Elemen ini akan dicari semua resepnya (kombinasi dua bahan yang bisa menghasilkan elemen tersebut), lalu tiap kombinasi diuji kelayakannya berdasarkan aturan *tier*: elemen hasil hanya bisa dibuat dari bahan dengan tier yang lebih rendah. Bila valid, maka elemen yang dipilih digantikan oleh dua bahan penyusunnya (jika belum dasar), dan langkah crafting baru ditambahkan ke node yang akan dimasukkan ke antrian BFS untuk dilanjutkan pencariannya.

Proses ini memastikan bahwa pencarian menyebar secara merata dari elemen target ke bahan-bahan dasar, dan akan menemukan jalur crafting terpendek terlebih dahulu (dalam jumlah langkah crafting). Untuk mencegah eksplorasi berulang pada elemen yang sama dalam satu jalur, program menjaga pencatatan *Visited* per node, serta menggunakan *StepSet* untuk menghindari duplikasi langkah. Karena arah BFS-nya *reverse*, langkah-langkah crafting yang ditemukan perlu dibalik urutannya sebelum dikembalikan sebagai hasil akhir.

Dalam versi BFS paralel, seperti yang digunakan oleh fungsi *FindMultipleRecipesBFSConcurrent*, pencarian dilakukan secara serentak dengan memanfaatkan *multithreading*. Beberapa thread diluncurkan untuk menelusuri jalur crafting berbeda berdasarkan variasi kombinasi bahan. Pendekatan ini sangat efektif dalam menghasilkan banyak solusi alternatif dengan cepat, karena setiap thread mengeksplorasi kemungkinan pembuatan bahan secara independen. Untuk menghindari duplikasi hasil, sistem mencatat *signature* unik dari setiap jalur crafting dan hanya menyimpan jalur-jalur yang benar-benar berbeda. Penggunaan *memoization* terbatas juga diterapkan, misalnya dengan mencatat bahan-bahan yang sudah gagal dibuat, sehingga thread tidak perlu melakukan pencarian yang sama berulang kali.

Contoh sederhananya:
Jika ingin membuat Barn, maka: Barn bisa dibuat dari House + Field. House bisa dari Wall + Wall, Field dari Earth + Tool. Proses akan terus menurunkan ke bahan dasar seperti Earth, Fire,

dll. Karena BFS akan melakukan pengecekan setiap turun kedalaman, maka umumnya jalur yang pertama ditemukan adalah jalur tercepat/terpendek.

# Bab 4
# Implementasi dan Pengujian

## 4.1 Spesifikasi Teknik Program

### 4.1.1 Struktur Data

#### 4.1.1.1 DFS (Depth Search First)

1. ElementRecipe
   Struktur data yang merepresentasikan satu elemen beserta semua resep pembentuknya, gambar, dan *tier*-nya.
2. Step dan Path
   Menyimpan urutan kombinasi elemen yang dilakukan untuk membentuk elemen target (Step adalah langkah tunggal dari Path)

#### 4.1.1.2 BFS (Breadth Search First)

1. BFSNode
   Struktur data yang merepresentasikan satu state pada pencarian BFS. Setiap BFSNode menyimpan semua informasi yang dibutuhkan untuk mendeskripsikan satu titik pada pohon pencarian BFS. Remaining adalah array of string yang menunjukkan apa saja elemen yang belum di expand dalam queue pencarian. Steps adalah array of Step dari struct dfs. Visited adalah map antara string dan boolean untuk menandakan apakah suatu elemen sudah pernah di expand, sehingga tidak perlu di expand lagi. StepSet adalah mapping antara suatu step dan boolean, untuk menandakan apakah suatu set unik apa tidak.

#### 4.1.1.3 Bidirectional

1. map[string][][]string elements
   Struktur utama yang menyimpan semua resep. Setiap elemen hasil (*key*) memiliki satu atau lebih pasangan bahan pembentuk
2. map[string][]string steps
   Digunakan untuk menyimpan langkah-langkah *crafting*, di mana setiap elemen hasil dikaitkan dengan dua bahan pembentuknya
3. map[string]bool basicElements
   Digunakan untuk menandai elemen-elemen dasar yang tersedia sejak awal (seperti Water, Fire, Earth).
4. map[string]int tier
   Digunakan untuk menyimpan informasi *tier* dari setiap elemen, digunakan untuk validasi *crafting* (bahan harus tier lebih rendah dari hasil).
5. SearchState struct
   Struktur yang digunakan dalam DFS bidirectional untuk menyimpan status eksplorasi saat ini

### 4.1.2 Fungsi dan Prosedur

#### 4.1.2.1 DFS (Depth Search First)

1. findPathDFS(...)
   Fungsi DFS rekursif dengan memoization untuk mencari satu jalur *crafting* dari elemen dasar ke target. Menghindari perhitungan ulang dengan memo (jika elemen sudah pernah dibuat, tidak perlu ditelusuri lagi), dan menyatukan langkah dari dua bahan jika jalur ke keduanya valid.
2. generateSignature(path Path)
   Menghasilkan string unik dari satu jalur untuk mengecek duplikasi jalur *crafting*.
3. createRecipeVariation(...)
   Membuat versi acak dari daftar resep dengan cara men-*shuffle* urutan resep dalam setiap elemen. Fungsi ini membantu menciptakan jalur *crafting* yang berbeda-beda dalam DFS.
4. FindSingleRecipeDFS(...)
   Fungsi pembungkus yang membaca file JSON resep dan menjalankan findPathDFS.
5. FindMultipleRecipesDFSConcurrent(...)
   Menjalankan DFS rekursif secara paralel dengan beberapa variasi resep. Menggunakan goroutine dan mutex untuk mendapatkan hasil pencarian dari berbagai thread, serta memfilter jalur berdasarkan *signature* agar tidak duplikat.

#### 4.1.2.2 BFS (Breadth Search First)

1. findPathBFS(...)
   Fungsi findPathBFS melakukan reverse breadth-first search untuk menemukan urutan langkah pembuatan (resep) yang valid yang menghasilkan elemen target dari serangkaian elemen dasar tertentu. Ia menelusuri dari elemen target ke belakang, mencoba semua kombinasi resep yang valid hingga semua elemen direduksi menjadi elemen dasar awal. Ia mengembalikan jalur pembuatan terpendek yang valid (jika ada), waktu yang dibutuhkan, dan jumlah elemen unik yang dikunjungi selama pencarian.
2. FindSingleRecipeBFS(...)
   Fungsi ini menemukan satu jalur pembuatan yang memungkinkan untuk membuat targetElement dari sekumpulan startingElement yang diberikan, menggunakan Breadth-First Search. Fungsi memuat semua resep dari file. Lalu menjalankan findPathBFS untuk menghitung jalur terpendek. Jika ditemukan, mencetak dan mengembalikan jalur tersebut, jumlah node yang dikunjungi, dan waktu yang dibutuhkan. Jika tidak ada jalur, ia melaporkan kegagalan.
3. FindMultipleRecipesBFSConcurrent(...)
   Fungsi ini menemukan beberapa jalur beragam untuk membuat targetElement menggunakan BFS paralel pada berbagai versi set resep. Fungsi memuat resep dan mengidentifikasi kombinasi yang valid untuk membuat target. Lalu menerapkan beberapa variasi resep acak untuk mengeksplorasi berbagai strategi. Menggunakan goroutine (thread bersamaan) dengan batas waktu dan semaphore untuk:
   - Mencari jalur bahan.
   - Menggabungkan jalur tersebut menjadi solusi lengkap untuk target.
   - Menghindari jalur duplikat menggunakan signature.

Lalu, mengumpulkan dan mengembalikan hingga maxRecipes jalur yang valid dan unik, diurutkan berdasarkan langkah paling sedikit.

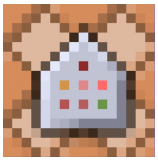
#### 4.1.2.3 Bidirectional

1. BiSearchBFS(...)
   Mencari jalur crafting menggunakan algoritma Breadth First Search dua arah (dari elemen dasar dan dari target). Menggunakan dua antrian (forwardQueue dan backwardQueue) untuk menemukan titik temu, kemudian merekonstruksi jalur lengkap dari kedua sisi.
2. BiSearchDFS(...)
   Versi dua arah dari pencarian dengan Depth First Search. Menggunakan dua stack (forwardStack dan backwardStack) dan menyimpan eksplorasi dalam bentuk SearchState. Jalur dibentuk dengan cara menelusuri ke satu arah sampai mentok sebelum beralih ke jalur lain.
3. reconstructPath(...)
   Membangun kembali urutan *crafting* dari *basic elements* hingga target berdasarkan langkah-langkah (steps) yang dikumpulkan. Implementasinya menggunakan *topological sort*.
4. isStepsComplete(steps, basicElements)
   Mengecek apakah semua bahan dalam langkah-langkah crafting dapat ditelusuri secara valid ke elemen dasar. Jika ada bahan yang tidak memiliki jalur, maka dianggap tidak lengkap.
5. cloneMap(map[string]bool)
   Fungsi tambahan untuk menyalin map boolean. Digunakan saat membuat SearchState baru agar state eksplorasi tidak saling mengganggu.
6. isValidElement(element, elements, tiers)
   Mengecek apakah suatu elemen valid (punya *tier* dan terdaftar dalam elemen hasil atau bahan di recipes).
7. FindSingleRecipe(...)
   Fungsi pembungkus yang memilih antara BiSearchDFS dan BiSearchBFS berdasarkan parameter input untuk mencari satu jalur *crafting* dari elemen dasar ke target.
8. BiSearchMultipleBFS(...) / BiSearchMultipleDFS(...)
   Menjalankan BFS/DFS berulang kali secara acak untuk menghasilkan beberapa jalur *crafting* yang unik. Pengacakan dilakukan pada urutan resep, dan hasil dievaluasi keunikannya dengan *hashing signature* (hashPath) untuk BFS agar tidak ada duplikasi.

## 4.2 Tata Cara Penggunaan Program

1. Jika ingin mengakses melalui web, masuk ke situs https://kejucraft.vercel.app
2. Jika ingin menggunakan docker, ketik docker network create kejucraft-net di terminal lalu docker compose up untuk kedua docker (*front end* dan *back end*)

3. Pilih ingin menggunakan algoritma BFS, DFS, atau Bidirectional. Untuk opsi Bidirectional, pilih ingin BFS atau DFS. Berikut panduan dalam website untuk pengubahan mode-mode yang ada
4. Lakukan scraping terlebih dahulu jika belum memiliki file scraping
5. Pilih ingin mencari *single* atau *multiple recipe*

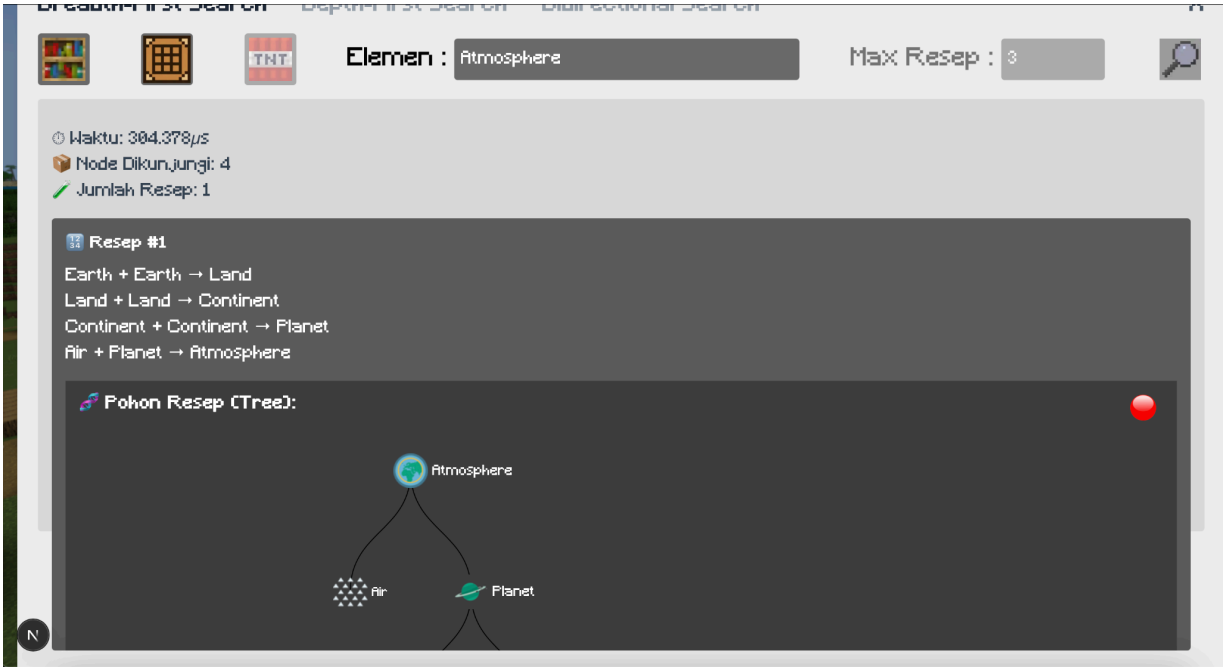
| | |
|---|---|
| | Button untuk melakukan scraping |
| | Switch button (bersama dengan furnace), yakni sedang dalam mode single threading |
| | Switch button (bersama dengan crafting table), yakni sedang dalam mode multi threading |
| | Switch button (bersama dengan command block), yakni sedang dalam mode Bidirectional BFS |
|  | Switch button (bersama dengan TNT), yakni sedang dalam mode Bidirectional DFS |

6. Masukkan nama elemen yang ingin dicari
7. Jika memilih mode *multiple recipe*, masukkan parameter dari banyak resep yang ingin dicari
8. Klik tombol *search*
9. Jika ingin melihat *live update* dari pembuatan *tree* secara berulang, klik tombol merah pada bagina hasil

## 4.3 Pengujian dan Analisisnya

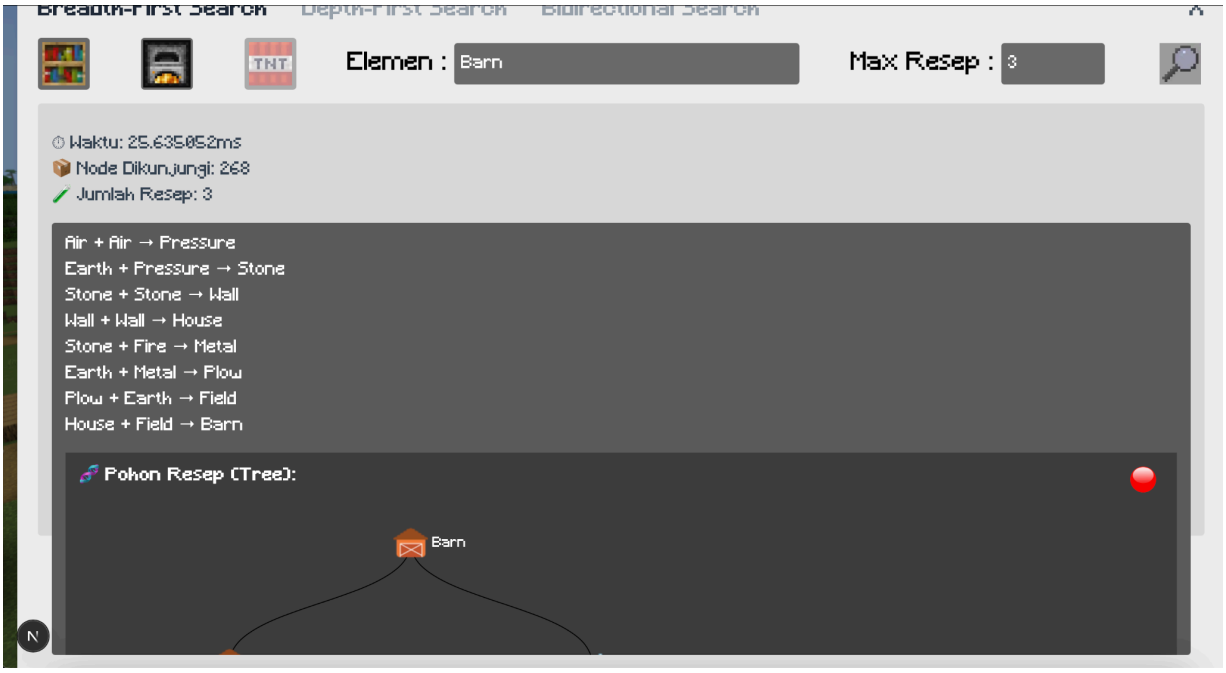
### 4.3.1 Pengujian 1

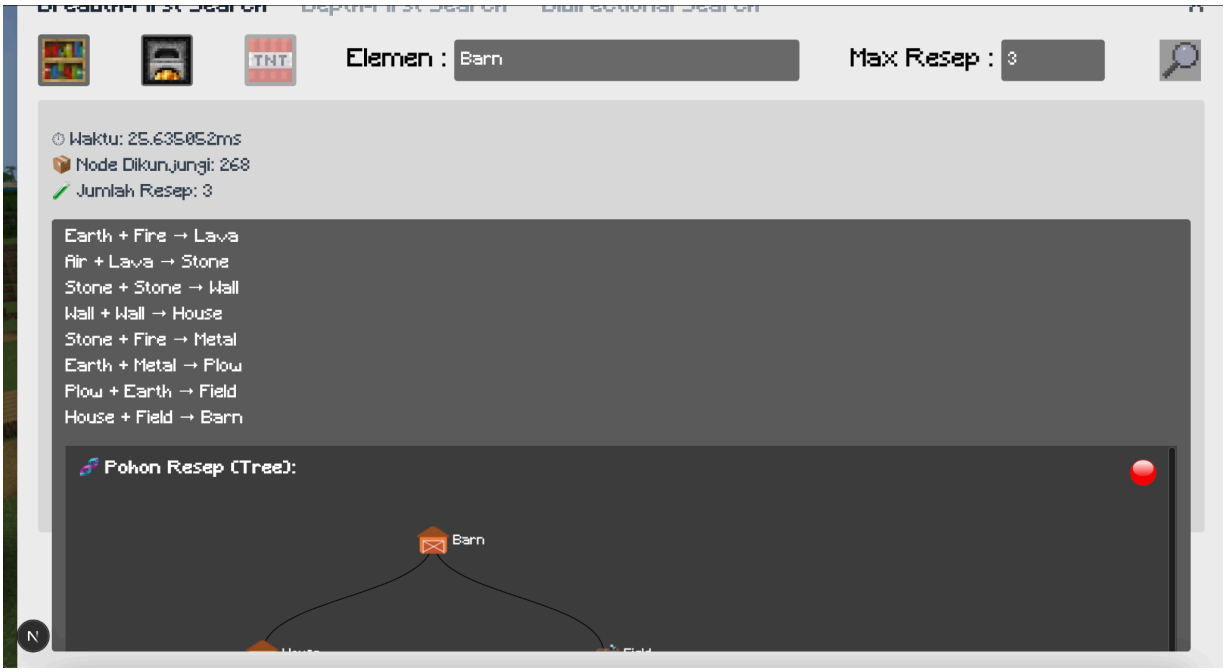
| | |
|---|---|
| Test Case : | Pencarian elemen Atmosphere dengan metode BFS single-threading |

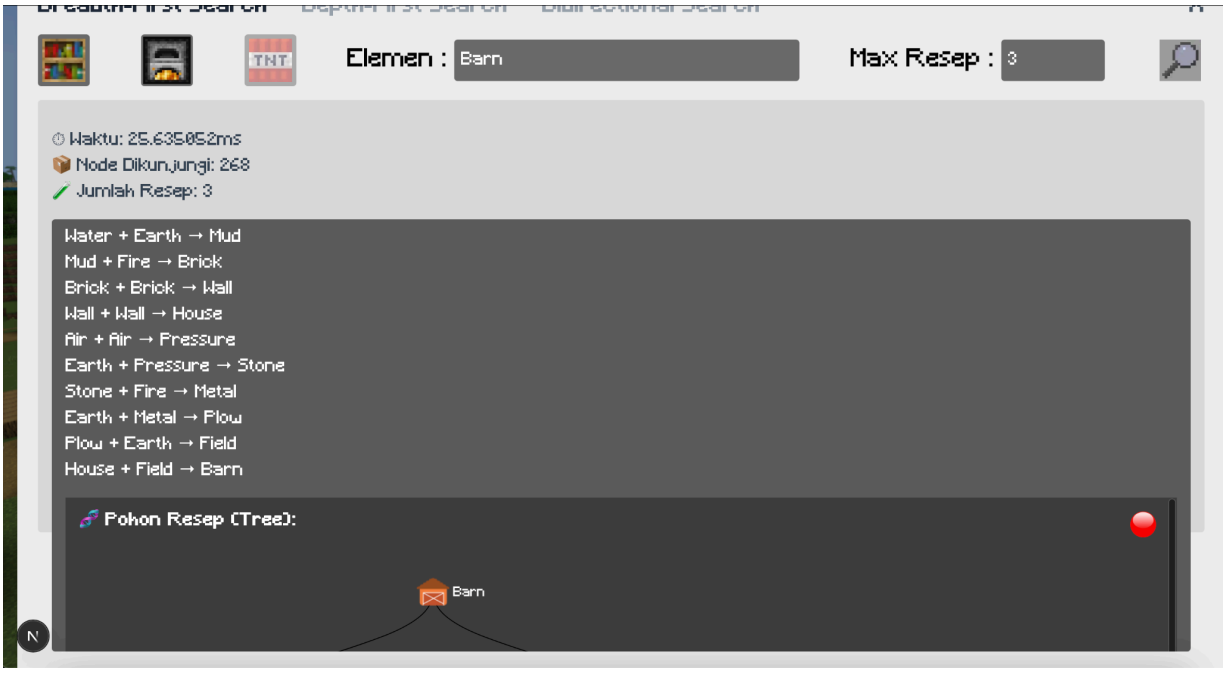
| Hasil Pengujian |
|---|
|  |
| Analisis Hasil |
| Program dapat menemukan 1 resep dari elemen Atmosphere. Dengan waktu membutuhkan 0.3 ms dan mengunjungi 4 node. |

### 4.3.2 Pengujian 2

| Test Case : | Pencarian 3 resep elemen Barn dengan metode BFS multithreading |
|---|---|
| Hasil Pengujian | |


Elemen : Barn
Max Resep : 3
Waktu: 25.635052ms
Node Dikunjungi: 268
Jumlah Resep: 3
Air + Air → Pressure
Earth + Pressure → Stone
Stone + Stone → Wall
Wall + Wall → House
Stone + Fire → Metal
Earth + Metal → Flow
Flow + Earth → Field
House + Field → Barn
Pohon Resep (Tree):
Barn

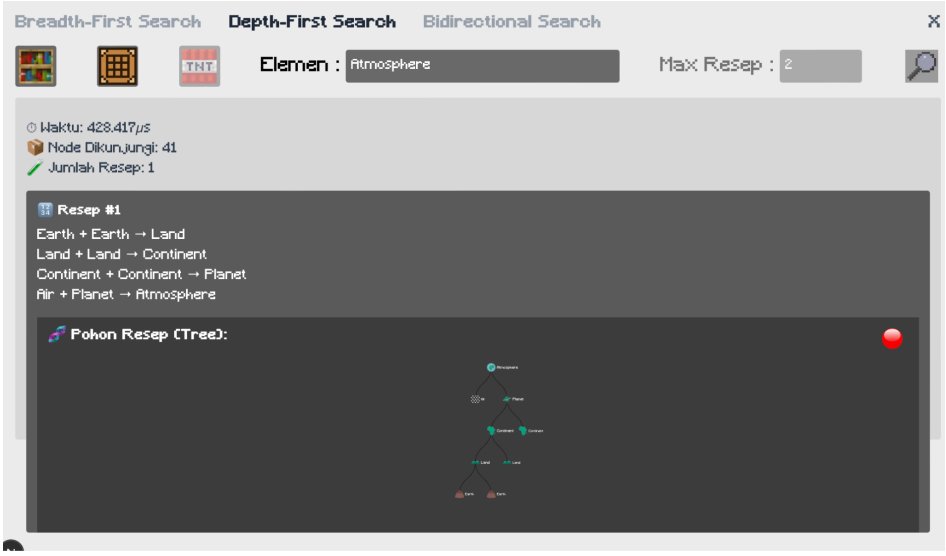

Elemen : Barn
Max Resep : 3
Waktu: 25.635052ms
Node Dikunjungi: 268
Jumlah Resep: 3
Earth + Fire → Lava
Air + Lava → Stone
Stone + Stone → Wall
Wall + Wall → House
Stone + Fire → Metal
Earth + Metal → Flow
Flow + Earth → Field
House + Field → Barn
Pohon Resep (Tree):
Barn




| Analisis Hasil |
|---|
| Program dapat berhasil menemukan 3 resep berbeda dari elemen Barn dengan membutuhkan waktu 25 ms dan mengunjungi 268 node. |

### 4.3.3 Pengujian 3

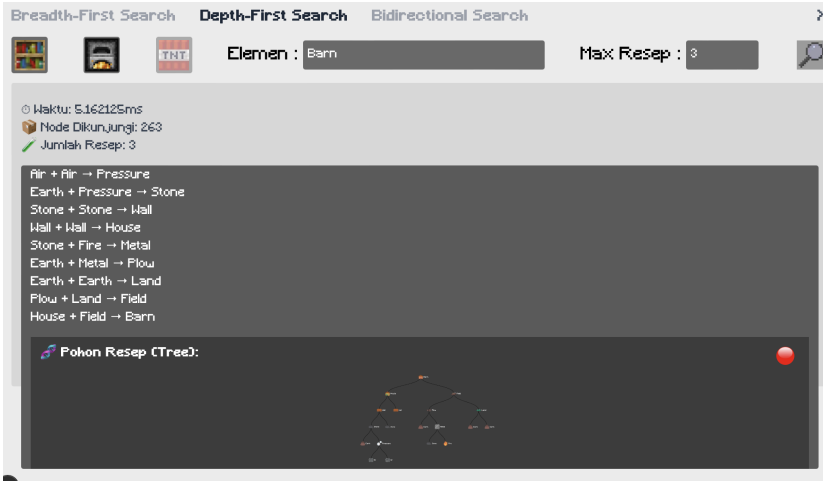
| Test Case : | Pencarian elemen Atmosphere dengan metode DFS single-threading |
|---|---|
| Hasil Pengujian | |

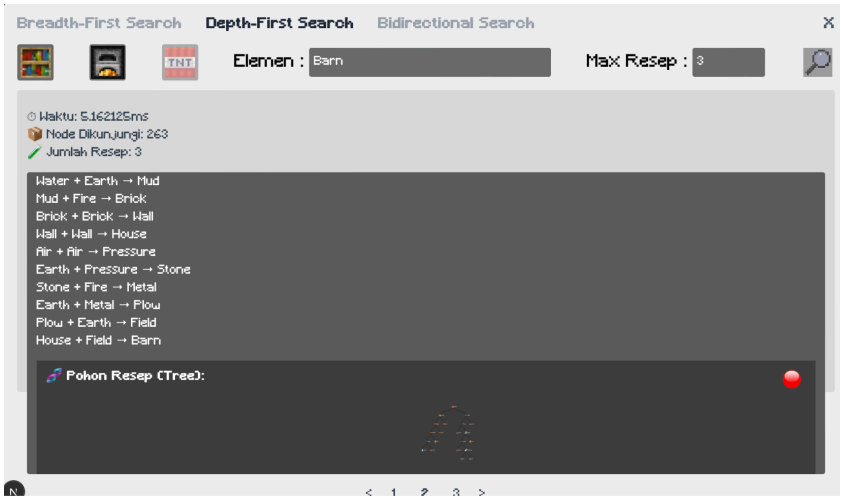


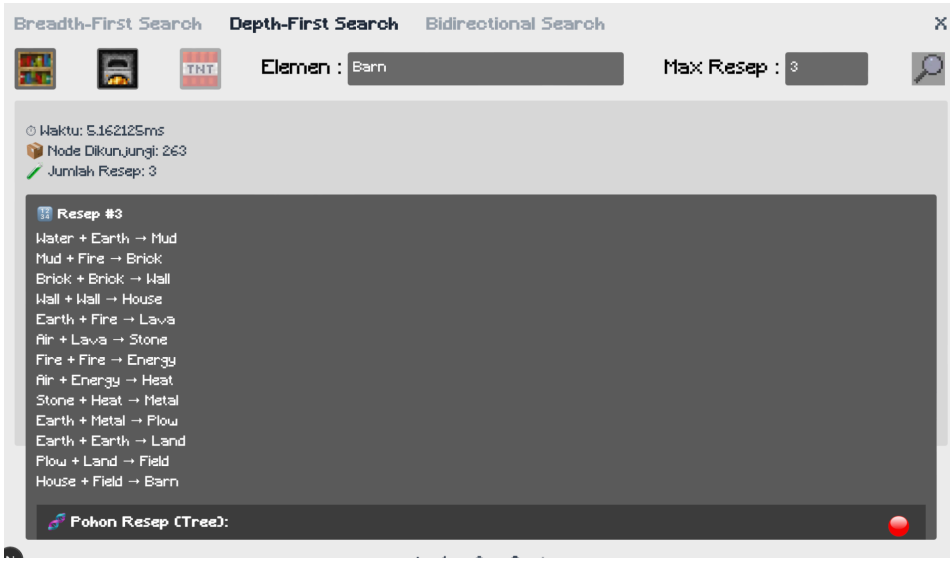

| Analisis Hasil |
|---|
| Program berhasil menemukan resep Atmosphere dengan membutuhkan 0.4 ms dan mengunjungi 41 node berbeda. |

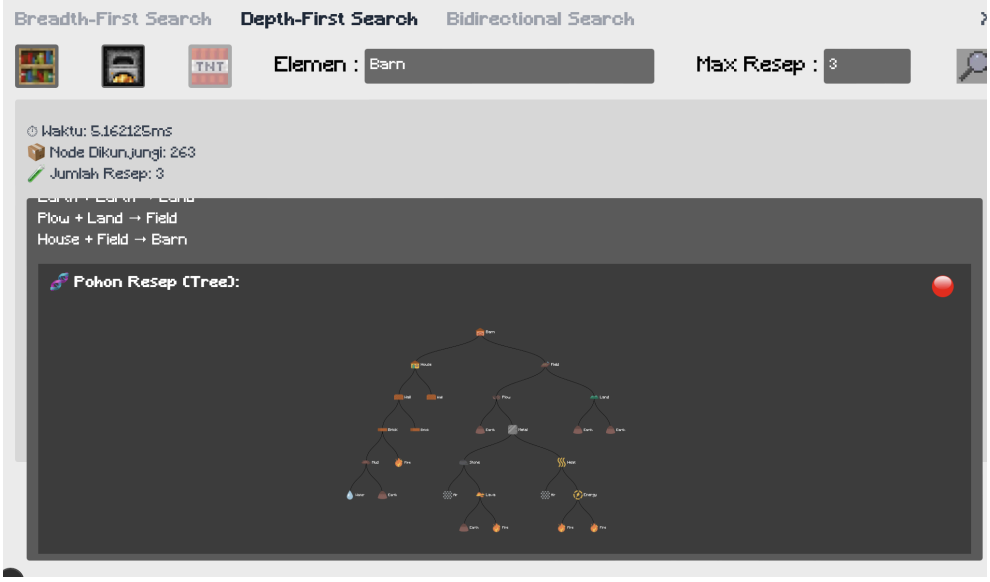
### 4.3.4 Pengujian 4

| Test Case : | Pencarian 3 resep elemen Barn dengan metode DFS multi-threading |
|---|---|
| Hasil Pengujian | |



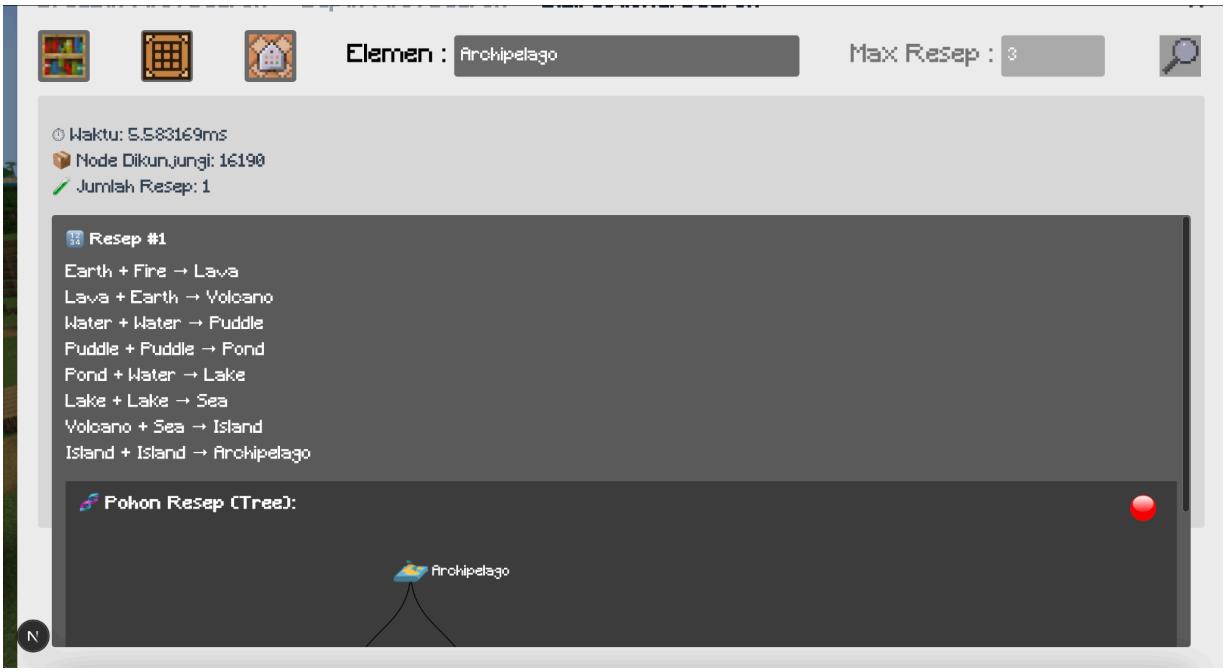

Breadth-First Search
Depth-First Search
Bidirectional Search
Elemen : Barn
Max Resep : 3
Waktu: 5.16212Sms
Node Dikunjungi: 263
Jumlah Resep: 3
Water + Earth → Mud
Mud + Fire → Brick
Brick + Brick → Wall
Wall + Wall → House
Air + Air → Pressure
Earth + Pressure → Stone
Stone + Fire → Metal
Earth + Metal → Plow
Plow + Earth → Field
House + Field → Barn
Pohon Resep (Tree):


Breadth-First Search
Depth-First Search
Bidirectional Search
Elemen : Barn
Max Resep : 3
Waktu: 5.16212Sms
Node Dikunjungi: 263
Jumlah Resep: 3
Resep #3
Water + Earth → Mud
Mud + Fire → Brick
Brick + Brick → Wall
Wall + Wall → House
Earth + Fire → Lava
Air + Lava → Stone
Fire + Fire → Energy
Air + Energy → Heat
Stone + Heat → Metal
Earth + Metal → Plow
Earth + Earth → Land
Plow + Land → Field
House + Field → Barn
Pohon Resep (Tree):


Breadth-First Search
Depth-First Search
Bidirectional Search
Elemen : Barn
Max Resep : 3
Waktu: 5.16212Sms
Node Dikunjungi: 263
Jumlah Resep: 3
Plow + Land → Field
House + Field → Barn
Pohon Resep (Tree):

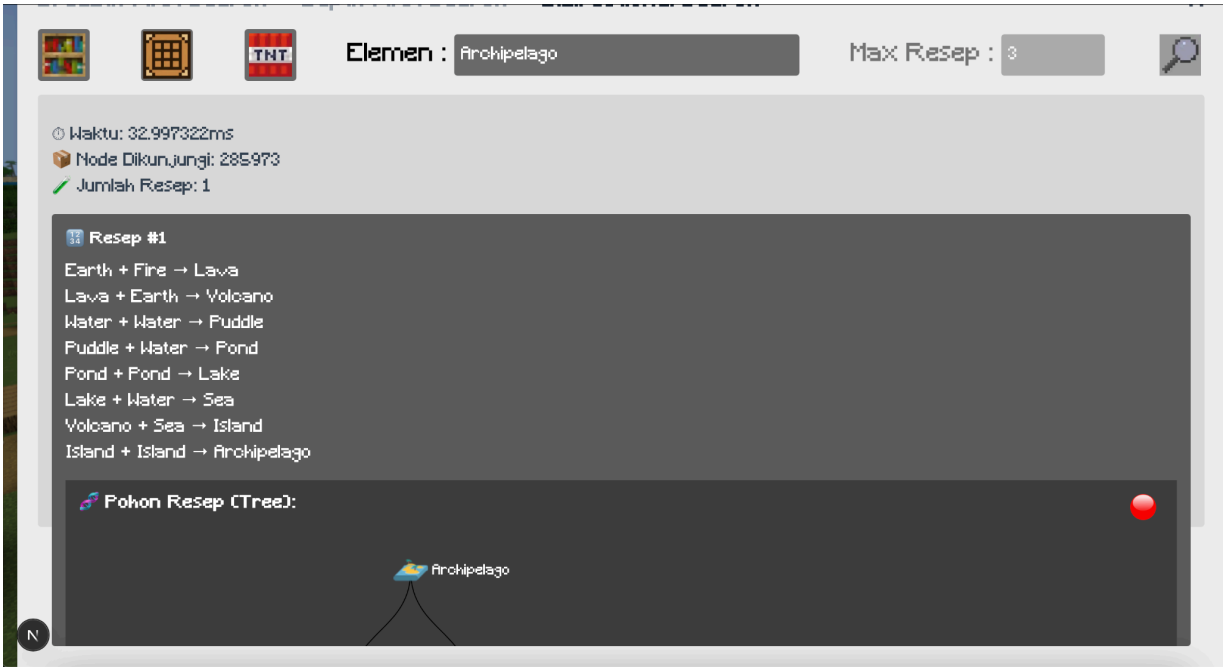
| Analisis Hasil |
| --- |
| Program berhasil menemukan 3 resep berbeda dari Barn dengan membutuhkan waktu 5 ms dan mengunjungi 263 node |

### 4.3.5 Pengujian 5

| Test Case : | Pencarian elemen Archipelago dengan metode Bidirectional BFS single-threading |
| --- | --- |
| Hasil Pengujian | |
|   | |
| Analisis Hasil | |
| Program berhasil menemukan 1 resep dari Archipelago dengan membutuhkan waktu 5 ms dan mengunjungi 16190 node | |

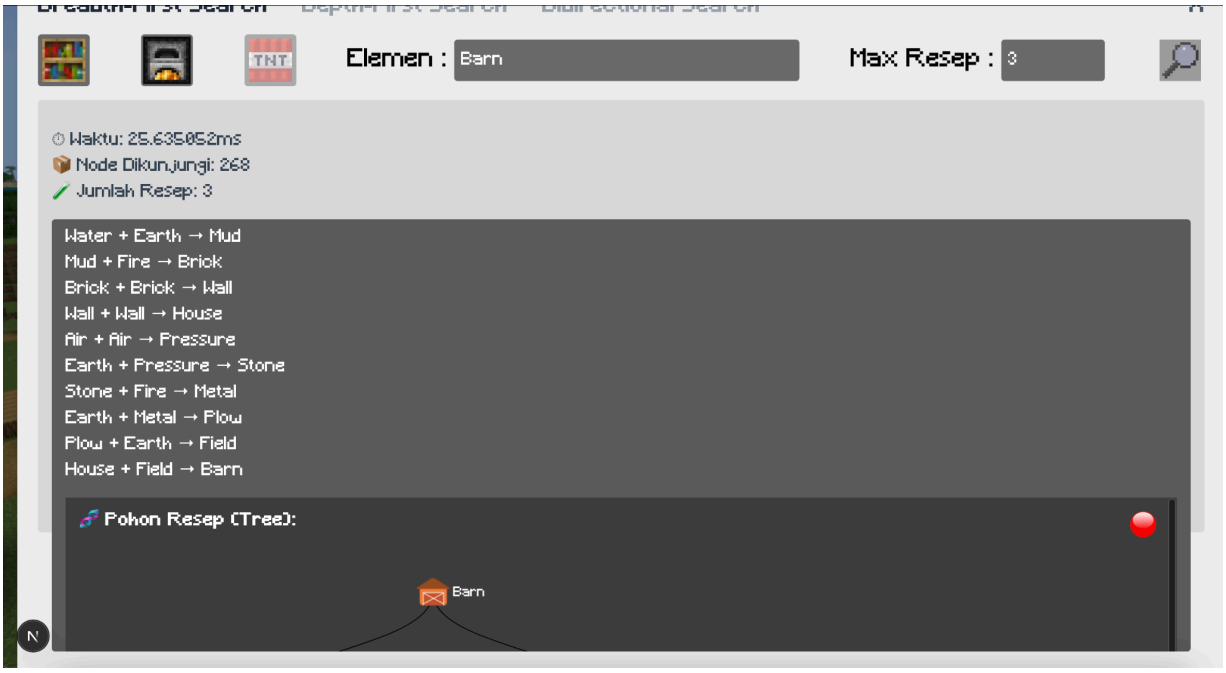
### 4.3.6 Pengujian 6

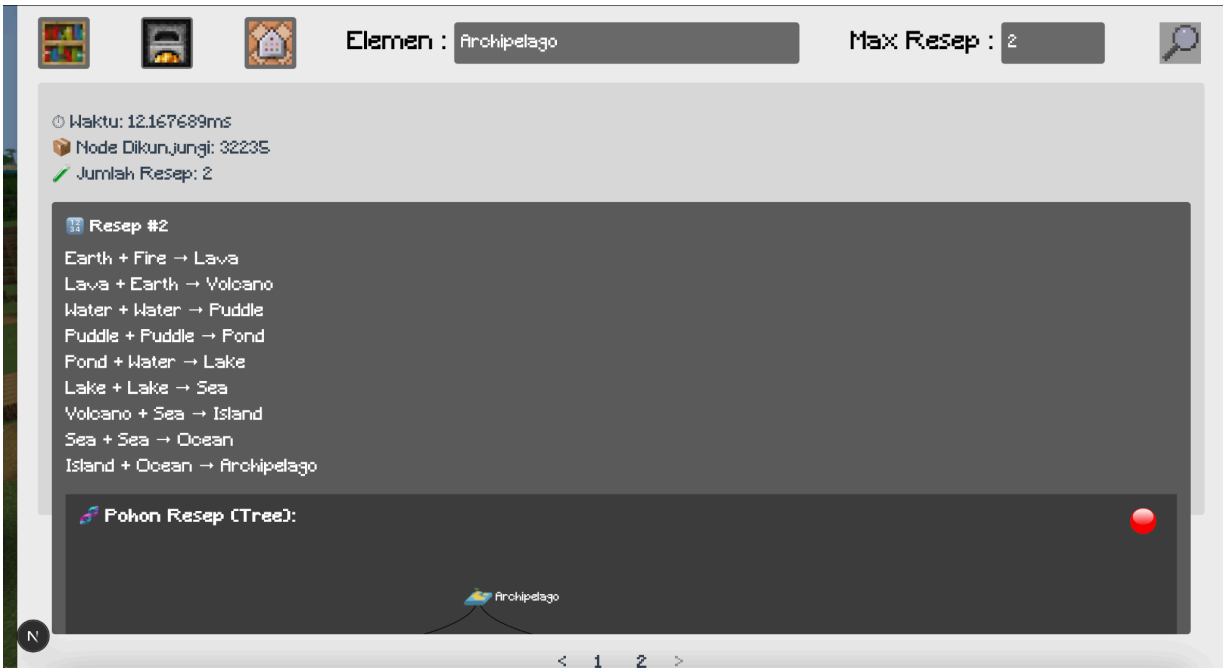
| Test Case : | Pencarian elemen Archipelago dengan metode Bidirectional DFS single-threading |
| --- | --- |

| Hasil Pengujian |
| --- |
|  |
| Analisis Hasil |
| Program berhasil menemukan 1 resep dari Archipelago dengan membutuhkan waktu 32 ms dan mengunjungi 285000 node |

### 4.3.7 Pengujian 7

| Test Case : | Pencarian elemen Archipelago dengan metode Bidirectional BFS multithreading |
| --- | --- |
| Hasil Pengujian | |



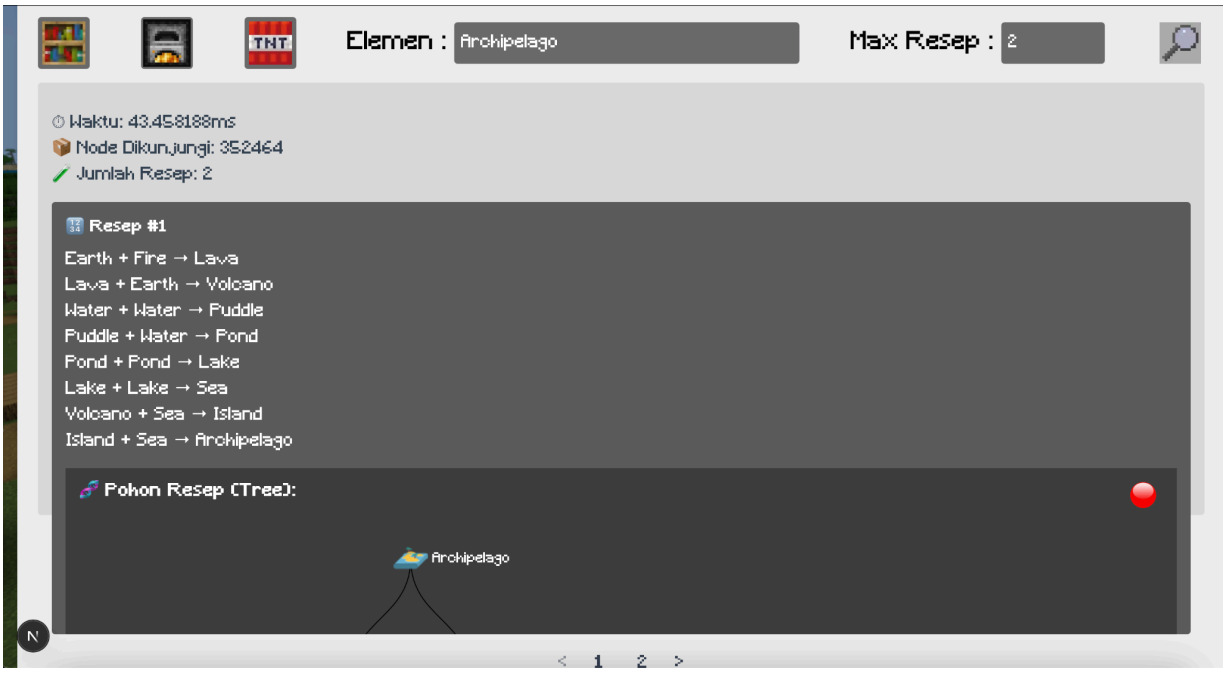



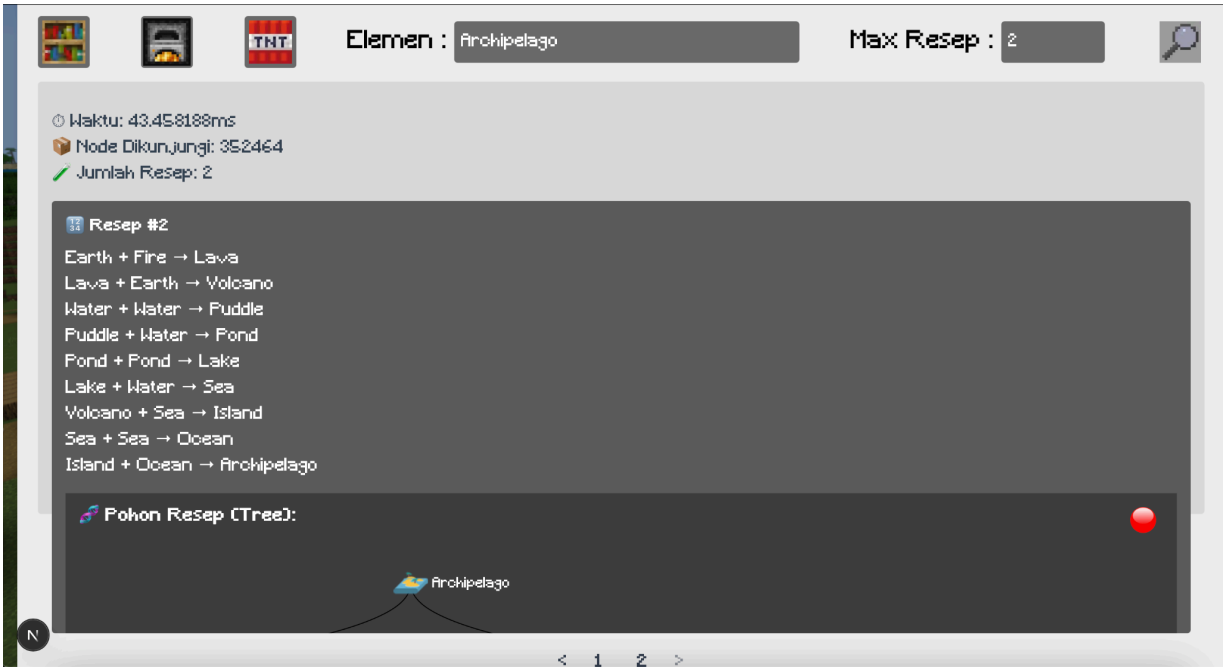

Analisis Hasil

Program berhasil menemukan 2 resep berbeda dari Archipelago dengan membutuhkan waktu 12 ms dan mengunjungi 32000 node

### 4.3.8 Pengujian 8

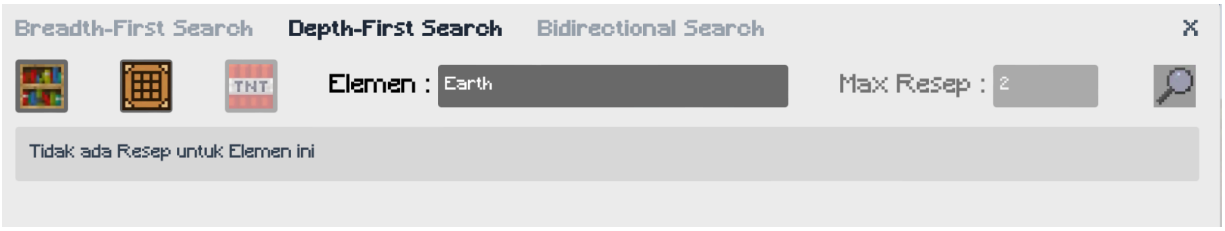
| Test Case : | Pencarian elemen Archipelago dengan metode Bidirectional DFS multithreading |
|---|---|
| Hasil Pengujian | |



| Analisis Hasil |
| --- |
| Program berhasil menemukan 2 resep berbeda dari Archipelago dengan membutuhkan waktu 43 ms dan mengunjungi 350000 node |

### 4.3.9 Pengujian 9

| Test Case : | Pencarian elemen dasar dan invalid elemen |
| --- | --- |
| Hasil Pengujian | |
|   | |
| Analisis Hasil | |
| Program tidak dapat menemukan path untuk tree karena tree membutuhkan elemen Time. Program juga tidak dapat menemukan path untuk elemen dasar karena tidak ada resepnya. | |

Secara keseluruhan, analisis hasil eksperimen menunjukkan pola dalam kinerja algoritma pencarian untuk masalah crafting yaitu DFS mengungguli BFS dengan mengunjungi lebih sedikit simpul dan menyelesaikan pencarian lebih cepat baik dalam implementasi *single recipe* maupun *multi recipe*, namun tren ini berbalik pada implementasi bidirectional search, di mana DFS justru mengunjungi lebih banyak simpul dan membutuhkan waktu lebih lama dibandingkan BFS. Hal ini disebabkan karena resep yang diminta adalah resep apapun, bukan resep terpendek sehingga DFS akan bisa menemukan resep terlebih dahulu karena algoritmanya yang menelusuri simpul secara mendalam.

Bidirectional BFS mengungguli bidirectional DFS dalam masalah *crafting* karena pola eksplorasi yang lebih efisien untuk menemukan titik temu. BFS secara sistematis memperluas semua simpul pada level yang sama sebelum pindah ke level berikutnya, meningkatkan

kemungkinan untuk kedua pencarian bertemu di tengah-tengah jalur dengan ekspansi yang seimbang. Sebaliknya, bidirectional DFS cenderung menjelajahi jalur-jalur secara mendalam yang mungkin sebenarnya tidak diperlukan secara berlebihan, termasuk cabang-cabang yang menyimpang jauh dari solusi, sebelum menemukan titik temu.

Secara teori, *bidirectional search* seharusnya memiliki kompleksitas O(b^(d/2)), lebih baik daripada BFS (O(b^d)) dan DFS (O(b^m)), tetapi hasil menunjukkan bahwa implementasi biasa (*non-bidirectional*) dari kedua algoritma justru lebih cepat dan mengunjungi lebih sedikit simpul. Hasil ini dapat dijelaskan oleh beberapa faktor yaitu *overhead* implementasi *bidirectional* yang memerlukan pelaksanaan dua pencarian dan struktur data tambahan untuk mendeteksi titik pertemuan serta menyambungkan kedua jalur, karakteristik masalah *crafting* dengan graf yang mungkin asimetris yaitu pencarian dari arah elemen dasar ke target mungkin lebih terstruktur dibanding arah sebaliknya, dan distribusi solusi yang mungkin lebih terkonsentrasi pada kedalaman tertentu. Hasil ini menunjukkan bahwa *overhead* komputasi dapat mengimbangi atau bahkan melebihi keuntungan teoretis dari pendekatan bidirectional, terutama untuk masalah dengan ruang pencarian yang tidak terlalu besar atau memiliki struktur khusus seperti dalam crafting game.

# Bab 5
# Kesimpulan, Saran, dan Refleksi

## 5.1 Kesimpulan

Melalui proyek ini, telah berhasil dibangun sebuah sistem pencari jalur crafting berbasis algoritma pencarian graf DFS, BFS, dan bidirectional. Sistem mampu mencari satu maupun banyak jalur pembuatan suatu elemen dari elemen dasar, dengan mempertimbangkan level (*tier*) antar elemen sebagai *constraint* utama. Implementasi juga telah mendukung pencarian *single recipe* dan *multiple recipe* menggunakan *multithreading* untuk mode *multiple recipe*

## 5.2 Saran

Dari sisi teknis, program telah berhasil menjalankan berbagai fungsi utama, namun terdapat beberapa aspek yang masih dapat dioptimalkan. Salah satunya adalah efisiensi waktu eksekusi, terutama pada implementasi DFS bidirectional untuk pencarian multiple recipe yang masih berjalan relatif lambat untuk beberapa elemen tertentu jika dibandingkan pendekatan lainnya. Hal ini dapat diperbaiki dengan mempercepat proses pencarian jalur alternatif dan mengurangi duplikasi eksplorasi pada banyak *thread*. Selain itu, penggunaan memori juga perlu diperhatikan, khususnya pada proses duplikasi struktur data seperti *map* dan *path* di tiap *thread* pencarian. Optimalisasi dari sisi waktu eksekusi dan penggunaan ruang memori menjadi hal yang penting untuk meningkatkan performa program, terutama jika digunakan pada dataset yang lebih besar atau lebih kompleks.

## 5.3 Refleksi

Tugas besar ini menjadi pengalaman yang penuh tantangan sekaligus pembelajaran. Salah satu hal yang paling signifikan adalah adanya perubahan spesifikasi yang cukup fundamental pada H-3 sebelum deadline sehingga menuntut adanya penyesuaian ulang terhadap seluruh struktur pencarian serta penerapan ulang konsep *tier*. Pengalaman ini mengajarkan pentingnya fleksibilitas teknis, serta bahwa menjaga semangat dan komunikasi adalah kunci bertahan dalam tekanan.

# Lampiran

Link *repository* github FE: https://github.com/sbimasena/Tubes2_FE_bagikopiciumbuleuit

Link *repository* github BE: https://github.com/sbimasena/Tubes2_BE_bagikopiciumbuleuit

Link video bonus: https://youtu.be/0rpWqJXmPWI

Link website: https://kejucraft.vercel.app/

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

# Daftar Pustaka

Munir, R. (2025). *Breadth First Search dan Depth First Search (Bagian 1)*. Program Studi Teknik Informatika, STEI ITB. Diakses dari Breadth/Depth First Search (BFS/DFS)

Munir, R. (2025). *Breadth First Search dan Depth First Search (Bagian 2)*. Program Studi Teknik Informatika, STEI ITB. Diakses dari Breadth/Depth First Search (BFS/DFS)