

TUGAS BESAR II IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Ghaisan Zaki Pratama (10122078)

Qodri Azkarayan (13523010)

Clarissa Nethania Tambunan (13523016)

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Dr. Ir. Rinaldi Munir, M.T.

Menterico Adrian, S.T, M.T.

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

2025

BAB I

DESKRIPSI TUGAS

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

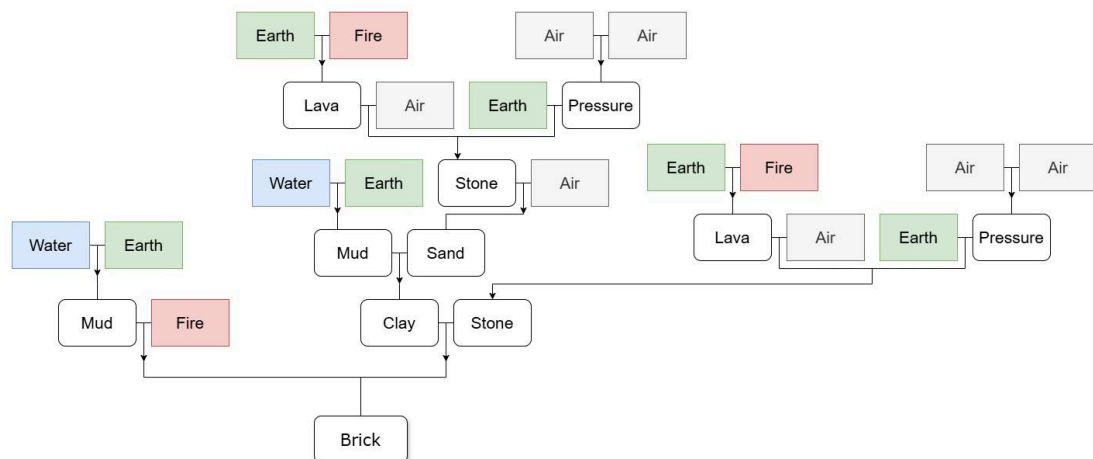
Untuk mendapatkan elemen turunan pemain dapat melakukan *combine* antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

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](https://www.fandom.com/wiki/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.

Spesifikasi Bonus

- Membuat video tentang aplikasi BFS dan DFS pada permainan Little Alchemy 2 di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll. **Semakin menarik video, maka semakin banyak poin yang diberikan.**
- Menambahkan algoritma bukan hanya DFS dan BFS, tetapi juga [strategi bidirectional](#)

BAB II

LANDASAN TEORI

2.1 Dasar Teori

2.1.1 Penjelajahan Graf

Penjelajahan graf (*graph traversal*) adalah teknik untuk mengunjungi setiap simpul dan sisi dalam graf dengan cara yang sistematis untuk menemukan jalur, memeriksa keterhubungan, atau membangun pohon solusi (Levitin, 2003). Representasi graf—baik sebagai daftar tetangga maupun matriks ketetanggaan—menentukan efisiensi traversal dengan *adjacency list*, kompleksitas traversal adalah $O(V + E)$ dengan V ialah banyak simpul dan E ialah banyak sisi (Bhardwaj & Verma, 2017). Traversal graf artinya melakukan pencarian solusi persoalan yang direpresentasikan dengan graf. Pada konteks aplikasi algoritma pencarian, traversal graf menjadi fondasi bagi BFS, DFS, dan metode lainnya (Munir, 2025). Beberapa algoritma tersebut merupakan algoritma pencarian solusi berbasis graf tanpa informasi (*uninformed/blind search*) karena dalam penjelajahannya tidak ada informasi tambahan yang disediakan. Dalam proses pencarian solusi, terdapat dua pendekatan:

1. Graf Statis

Graf yang sudah terbentuk sebelum proses pencarian dilakukan.

2. Graf Dinamis

Graf yang dibentuk saat proses pencarian dilakukan.

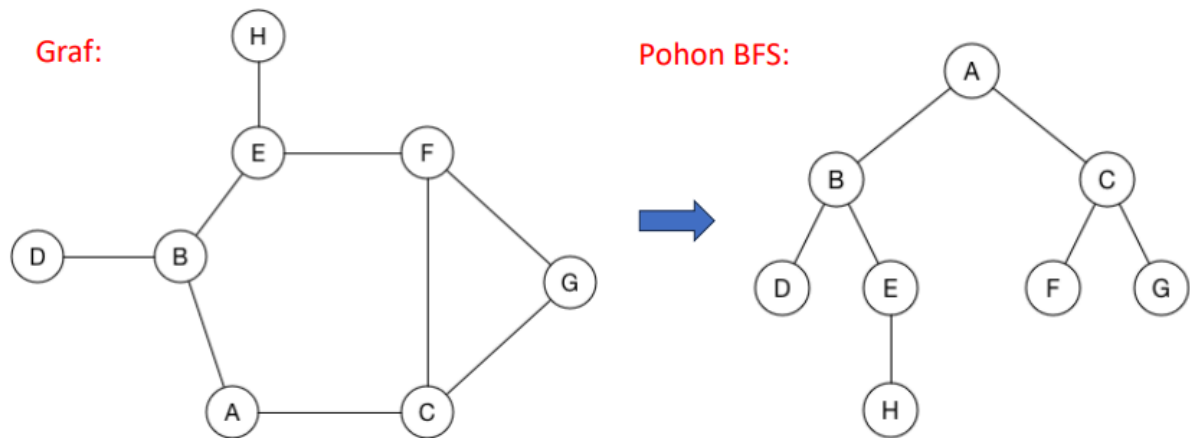
2.1.2 Breadth First Search

BFS bekerja lapis demi lapis mulai dari simpul akar, mengunjungi semua tetangga pada kedalaman d sebelum ke kedalaman $d + 1$ (Cormen et al., 1992). Hal ini menjamin BFS akan menemukan jalur terpendek pada graf tak berbobot (Coursera, 2019). Dilakukan dengan antrian (queue), kompleksitas waktu dan ruangnya adalah $O(b^d)$, dengan b adalah *branching factor* (maksimum pencabangan yang mungkin dari suatu simpul) dan d adalah *depth* (kedalaman dari solusi terbaik yaitu cost terendah) (Levitin, 2003). Selama nilai b terbatas BFS menjamin ditemukannya solusi jika memang ada, tetapi tidak menjamin mendapatkan solusi yang optimal (Munir, 2025). Traversal dimulai dari simpul v dengan algoritma sebagai berikut

1. Kunjungi simpul v .
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.

3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Traversal graf secara BFS dapat digambarkan sebagai pohon BFS:



Urutan simpul-simpul yang dikunjungi secara BFS dari A \rightarrow A, B, C, D, E, F, G, H

Gambar 1 : Contoh 2 dari slide kuliah BFS-DFS Bagian 1 (Munir, 2025).

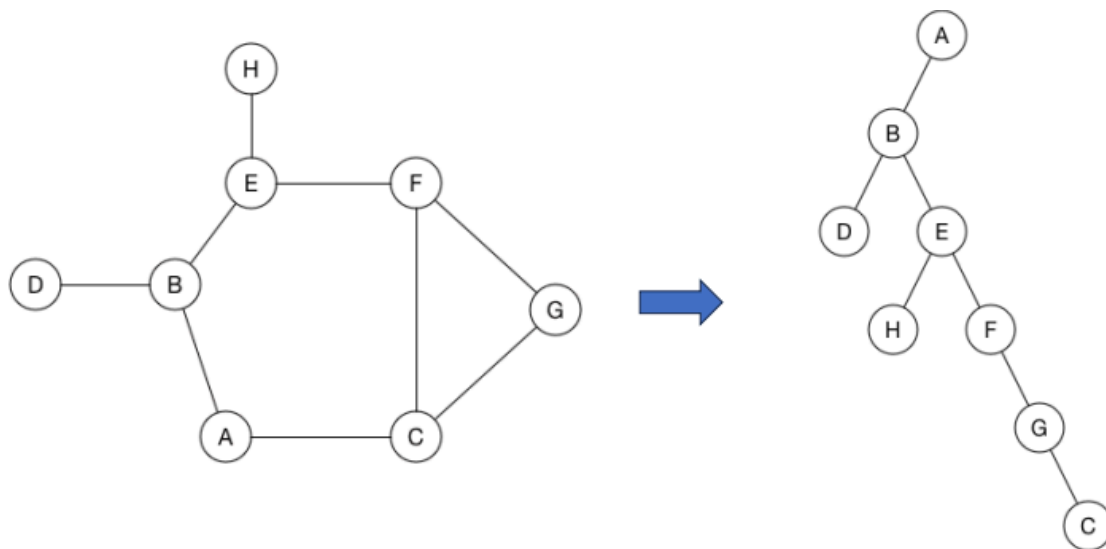
2.1.3 Depth First Search

DFS menelusuri sedalam mungkin melalui satu jalur sebelum melakukan *backtrack* (Levitin, 2003). Dilakukan dengan tumpukan (stack) atau rekursi, kompleksitas waktunya adalah $O(b^m)$ dan kompleksitas ruangnya adalah $O(bm)$ dengan m adalah maksimum kedalaman dari ruang status yang bisa bernilai tak hingga (Cormen et al., 1992). DFS tidak menjamin jalur terpendek, tetapi efisien untuk mengeksplorasi seluruh graf dan menemukan komponen terhubung atau siklus (Russel & Norvig, 2016). Selama nilai b terbatas dan ada penanganan 'redundant paths' dan 'repeated states' DFS menjamin ditemukannya solusi jika memang ada, tetapi tidak menjamin mendapatkan solusi yang optimal (Munir, 2025). Karakteristik *backtracking* di dalam algoritma DFS berguna untuk memecahkan persoalan pencarian solusi yang memiliki banyak alternatif pilihan selama pencarian. Solusi diperoleh dengan mengekspansi pencarian menuju *goal* dengan aturan "depth-first". *Backtracking* di dalam algoritma DFS adalah cara yang metodologis mencoba beberapa sekuens keputusan sampai menemukan sekuens yang "bekerja". Traversal dimulai dari simpul v dengan algoritma:

1. Kunjungi simpul v .
2. Kunjungi simpul w yang bertetangga dengan simpul v .
3. Ulangi DFS mulai dari simpul w .

4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Traversal graf secara DFS dapat digambarkan sebagai pohon DFS:



Urutan simpul-simpul yang dikunjungi secara DFS dari A \rightarrow A, B, D, E, H, F, G, C

Gambar 2 : Contoh 4 dari slide kuliah BFS-DFS Bagian 1 (Munir, 2025).

2.1.4 Bidirectional

Bidirectional search menjalankan dua BFS sekaligus—satu dari sumber dan satu dari tujuan—hingga mereka bertemu di tengah (GeeksforGeeks. 2025). Secara teori, kompleksitasnya turun menjadi $O(b^{d/2})$ per arah, sehingga total lebih cepat dibanding BFS tunggal (Progančić, 2019). Namun, efektivitasnya bergantung pada kemampuan melakukan traversal balik.

2.2 Penjelasan Singkat Mengenai Aplikasi Web yang Dibangun

Aplikasi web yang dibangun bernama “Little Alchemy 2 Recipe Finder” yang memudahkan pengguna untuk mencari resep-resep dalam permainan Little Alchemy 2 berdasarkan elemen target yang dimasukkan ketika pertama kali mengakses aplikasi web. Aplikasi web ini memiliki tiga pilihan algoritma pencarian resep target elemen tersebut yaitu BFS (*Breadth First Search*), DFS (*Depth First Search*), atau Bidirectional untuk menemukan cara mengkombinasikan elemen-elemen dasar hingga mencapai elemen target. Sebelum mencari

target elemen maka pengguna memiliki dua pilihan apakah ingin melakukan pencarian resep tunggal atau banyak resep sekaligus. Setelah dilakukan pencarian, aplikasi akan menampilkan visualisasi pohon resep dan juga akan menampilkan panel statistik dari resep tersebut.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Penjelasan umum alur pemecahan masalah adalah sebagai berikut: pertama, kami kumpulkan data elemen, tier, dan resep dengan scraping dari web; kedua, kami merepresentasikan data tersebut dalam struktur graph yang efisien untuk pencarian; ketiga, kami menerapkan algoritma BFS, DFS, Bidirectional (baik untuk mencari *single recipe* maupun *multiple recipe*) pada *graph* ini untuk membangun pohon resep lengkap; dan terakhir kami *expose* proses tersebut melalui API HTTP.

1. Scraping Data

Panggil `scraper.ScrapeToFile` yang akan:

- Fetch halaman wiki Little Alchemy 2.
- Parse setiap section tier (Starting, Tier 1, Tier 2, ...) dengan goquery.
- Ekstrak nama elemen, tier, dan pasangan elemen yang membangun elemen tersebut.
- Simpan ke `scraped_recipes.json` dalam format:

```
{
  "ElementName": {
    "tier": 2,
    "recipes": [["Left", "Right"], ...]
  }, ...
}
```

2. Inisialisasi Struktur Graph

Panggil `utils.NewGraph` yang melakukan:

- Baca file `scraped_recipes.json`.
- Unmarshal JSON ke map intermediate.
- Buat `Graph.Elements` map `name→tier`.
- Buat `Graph.Recipes` map `name→list of ingredient pairs`.
- Validasi bahwa semua bahan di resep memiliki entry tier (untuk fallback).

3. Implementasi Fungsi Utilitas

- `Tier(name)` : Mengembalikan level dasar elemen.
- `RecipesFor(target, enforceTier)` : Mengambil resep valid dengan filter tier bahan.

4. Implementasi Single Search: BFS

Fungsi `SearchBFS(g, target)`:

- Inisialisasi timer `start`, map `visited`, map `pre`, dan queue dengan `target`.
- Iterasi: Dequeue `curr`, increment `steps`.
- Jika `tier(curr)==0`, skip expand (elemen dasar yaitu *water*, *earth*, *fire*, dan *air*).
- Ambil `combos:=RecipeFor(curr, strict)`; jika kosong, `fallback:=RecipeFor(curr, false)`.
- Simpan resep pertama di `pre[curr]=[left, right]`.
- Tandai kedua bahan `visited`, enqueue.
- Ulangi hingga queue kosong.
- Bangun `TreeNode` dari `BuildTreeFromPre(g, target, pre)`.
- Hitung durasi dan kembalikan `SearchResult`.

5. Implementasi Single Search: DFS

Fungsi `SearchDFS(g, target)`:

- Inisialisasi timer, map `pre`, dan map `inPath` untuk detect siklus.
- DFS rekursif:
 - Base: jika `tier(curr)==0`, return `true`.
 - Tandai `inPath[curr]=true`, increment `steps`.
 - Loop `combos strict→fallback`.
 - Jika `dfs(left)&&dfs(right)` : simpan `pre[curr]=[left, right]`, clear `inPath[curr]`, return `true`.
 - Backtrack: `inPath[curr]=false`, return `false`.
- Panggil `dfs(target)`.
- Jika ditemukan path, rekonstruksi tree dan hitung durasi.

6. Implementasi Multiple Search: BFS

Fungsi `SearchBFSMultiple(g, target, maxRecipes)`:

- Ambil `combos` awal untuk `target strict→fallback`.
- Buat channel goroutine, sync primitives (WG, Mutex, context).
- Untuk tiap combo spawn goroutine yang memanggil `SearchBFSPre(g, target, pre)`:
 - Lakukan BFS lokal seeded dengan salah satu resep awal.
 - Kirim `SearchResult` via channel jika unik (map signature JSON).

- Cancel semua goroutine saat `len(results) == maxRecipes`.
- Kumpulkan hasil, kembalikan.

7. Implementasi Multiple Search: DFS

Fungsi `SearchDFSMultiple(g, target, maxRecipes)`:

- Spawn goroutine per resep awal.
- Panggil `SearchDFSPre(g, target, pre)` seeded.
- Deduplikasi dan batasi `maxRecipes`.

8. Implementasi Single Search: Bidirectional

Ekspansi simultan dari target (maju) dan semua elemen dasar (mundur), bertemu di tengah, lalu menggabungkan peta pendahulu untuk membangun pohon.

9. Implementasi Multiple Search: Bidirectional

BFS seeded dua arah per resep awal, memanggil `SearchBidirectionalWithSeed`, mengumpulkan pohon unik hingga `maxRecipes`.

10. Rekonstruksi Pohon Resep

- Membentuk `*Node` rekursif berdasarkan `pre map[string][2]string`.
- Versi *safe* (`BuildTreeFromSafe`) melakukan pelacakan `visited` untuk mencegah loop pada Bidirectional Search.

11. Routing dan API

Buat `router.SetupRouter` dengan satu endpoint POST `/search`:

- Bind JSON ke struct `RequestBody`.
- Validasi `target`, `algorithm`, `mode`, `max_recipes`.
- Dispatch ke salah satu dari fungsi `search`.
- Kembalikan JSON array `[]SearchResult`.

12. Bootstrap dan Server

`main.go` menjalankan:

- `scraper.ScrapeToFile`.
- `utils.NewGraph` untuk load graph.
- `router.SetupRouter` Build Gin engine.
- `r.Run(":8080")` untuk start server.

3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma DFS dan BFS

| Tahap | BFS | DFS |
|---------------------------|---|---|
| Inisialisasi | Queue FIFO (<i>First In First Out</i>), visited map, pre map kosong | Rekursi, inPath dan pre map kosong |
| Kondisi Dasar | Leaf dengan tier = 0 (elemen dasar) | Tier = 0: return true |
| Ekspansi Node | Dequeue curr, loop resep: strict→fallback, enqueue | Cek setiap combo, recuse pada left & right |
| Pencatatan Resep | Simpan pre[curr]=combo[0], combo[1] saat expand | Catat pre[curr] hanya di branch yang berhasil fully |
| Traversal | Breadth-first → semua level terurut | Depth-first→eksplorasi mendalam sebelum sibling |
| Backtracking | Tidak diperlukan | Keluarkan inPath[curr] = false jika gagal branch |
| Rekontruksi Tree | BuildTreeFromPre menelusuri pre secara rekursif | BuildTreeFromPre menelusuri pre secara rekursif |
| Langkah Paralel | spawn goroutine per resep awal | spawn goroutine per resep awal |
| Unikasi Hasil | seen map signature JSON | seen map signature JSON |
| Hentian Multi-Mode | context.Cancel saat len(results)>=maxRecipes | context.Cancel saat len(results)>=maxRecipes |

3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

1. Fitur Fungsional Aplikasi Web

- Form input untuk elemen target agar pengguna dapat memasukkan elemen target yang ingin dicari resepnya.

- Pemilihan algoritma untuk dapat memilih algoritma pencarian resep yang diinginkan pengguna yaitu BFS, DFS, dan Bidirectional.
- Mode *single* atau *multiple recipe* untuk dapat memilih antara mode pencarian resep tunggal (*single*) atau beberapa resep sekaligus (*multiple*). Jika pengguna memilih *multiple* maka terdapat batas maksimal resep yang ingin ditampilkan yaitu 1-6.
- Visualisasi pohon resep (*RecipeTree*) yang menampilkan visualisasi pohon untuk menggambarkan hubungan antar elemen dari *starting element* ke *target element* yang merupakan hasil akhir dari pencarian resep.
- Panel Statistik (*StatsPanel*) yang menyediakan statistik terkait pencarian yaitu jumlah node yang dikunjungi, waktu pencarian/durasi, dan jumlah resep yang ditemukan sesuai dengan mode pencarian yang dipilih.
- *Error Handling* yang menangani error saat data tidak ditemukan atau ketika terjadi kesalahan dalam proses pencarian resep.
- Update resep secara dinamis yang menampilkan hasilnya secara dinamis termasuk visualisasi dan statistik yang berubah berdasarkan input pengguna atau data yang diterima tanpa perlu memuat ulang halaman secara keseluruhan.

2. Arsitektur Aplikasi Web

Arsitektur aplikasi web yang dibangun terdiri dari dua bagian utama, yaitu *frontend* dan *backend* yang masing-masing memiliki fungsinya sendiri. Berikut adalah struktur direktori aplikasi web yang dibangun.

```
src/
├── backend/
│   ├── main.go
│   ├── router/
│   │   └── router.go
│   ├── search/
│   │   ├── bfs.go
│   │   ├── dfs.go
│   │   ├── common.go
│   │   ├── multi_bfs.go
│   │   └── multi_dfs.go
│   ├── scraper/
│   │   └── scraper.go
│   └── utils/
```

```

|   |   └─ graph.go
|   └─ go.mod
|   └─ go.sum
|   └─ scraped_recipes.json
└─ frontend/
    └─ pages/
        │   └─ _app.js
        │   └─ index.jsx
    └─ components/
        │   └─ RecipeForm.jsx
        │   └─ RecipeTree.jsx
        │   └─ StatsPanel.jsx
    └─ utils/
        │   └─ api.js
    └─ styles/
        │   └─ global.css
    └─ package.json
    └─ package-lock.json
    └─ .next/

```

Dalam bagian *frontend* ini, aplikasi web dibangun untuk bertanggung jawab atas semua interaksi dengan pengguna dan tampilan antarmuka. Berikut adalah penjelasan struktur *frontend* tersebut.

- `_app.js` : file ini untuk menginisialisasi aplikasi Next.js dan mengimpor `global.css` untuk seluruh aplikasi
- `index.jsx` : halaman utama yang menampilkan form input elemen target dan algoritma pencarian serta menampilkan hasil diagram pohon resep dan statistik yang dihasilkan.
- `components/` : folder ini berisi komponen-komponen yang membangun bagian antarmuka pengguna di mana setiap komponen bertanggung jawab untuk menampilkan bagian-bagian spesifik dari aplikasi.
 - `RecipeForm.jsx` : form input utama dimana pengguna dapat memasukkan elemen target yang ingin dicari, memilih algoritma pencarian yang diinginkan (BFS, DFS, Bidirectional), serta memilih *single* atau *multiple recipe*.

- `RecipeTree.jsx` : menampilkan hasil pohon resep yang menunjukkan elemen dasar yang dihasilkan untuk mencapai elemen target menggunakan diagram Mermaid.js untuk menggambarkan hubungan antar elemen secara visual.
- `StatsPanel.jsx` : menampilkan statistik terkait pencarian resep seperti jumlah node yang dikunjungi, waktu pencarian/durasi, dan jumlah resep yang ditemukan.
- `utils/` : folder ini berisi file utilitas yang digunakan di berbagai bagian aplikasi.
 - `api.js` : file ini berisi fungsi untuk berkomunikasi dengan *backend* melalui API. Fungsi `fetchRecipes` digunakan untuk mengirim permintaan resep ke *backend* dan menerima data resep yang dikembalikan oleh server.
- `styles/` : folder ini berisi file CSS untuk aplikasi.
 - `global.css` : file yang digunakan untuk mendesain elemen-elemen antarmuka pengguna seperti form input, *toggle switch*, dan tampilan diagram.
- `package.json` : file konfigurasi untuk dependensi Node.js yang digunakan dalam folder *frontend*.
- `package-lock.json` : menyimpan versi terbaru dari dependensi yang diinstal untuk memastikan kesesuaian antara instalasi di lingkungan pengembangan yang berbeda
- `.next/` : folder yang secara otomatis dihasilkan oleh Next.js saat proyek di-build yang berisi file-file yang diperlukan untuk menjalankan aplikasi di server produksi.

3.4 Contoh Ilustrasi Kasus

1. Masalah yang dihadapi pengguna misalnya yaitu seorang pengguna ingin mencari cara untuk membuat elemen baru dalam permainan Little Alchemy 2, seperti Barn. Pengguna tidak mengetahui resep atau kombinasi elemen yang diperlukan untuk membuat Barn.
2. Solusi yang diberikan aplikasi yaitu dengan pengguna memasukkan “Barn” sebagai *Target Element* pada form pencarian resep yang kemudian mencari resep untuk membuat “Barn” dan menampilkan hasil pencarian berupa kombinasi elemen yang dibutuhkan.
3. Hasilnya yaitu menampilkan diagram pohon yang menunjukkan urutan kombinasi elemen dari elemen dasar mencapai elemen target Barn dan menampilkan statistik panel resep.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur Data Program

1. Graph (utils.Graph)

- `Elements map[string]*Element`

Menyimpan tier tiap elemen.

- `Recipes map[string][[]string]`

Daftar resep asli per elemen (setiap combo berupa slice of 2 nama bahan).

2. SearchResult (search.SearchResult)

- `Recipe *Node`

Pointer ke akar pohon resep yang ditemukan.

- `NodesVisited int`

Jumlah node (elemen) yang dikunjungi selama pencarian.

- `Duration string`

Waktu pencarian dalam format milidetik.

3. Node (search.Node)

- `Name string`

Nama elemen.

- `Combines []*Node`

Slice node anak (dua bahan), kosong untuk elemen dasar.

4. Predecessor Map (map[string][2]string)

- Menyimpan resep terpilih untuk membangun pohon:

`pre[X] = ["Left", "Right"]` berarti X dibentuk dari Left dan Right.

5. Bagian Frontend

- `target` : Menyimpan elemen target yang akan dicari resepnya.

- `algo` : Menyimpan algoritma yang dipilih (BFS, DFS, Bidirectional).

- `mode` : Menyimpan mode pencarian (single atau multiple).

- `max` : Menyimpan jumlah maksimum resep yang akan dicari jika mode adalah "multiple".

- `error` : Menyimpan pesan error jika ada kesalahan dalam proses pencarian.

- `recipeData` : Menyimpan data resep yang diterima dari backend.

4.1.2 Fungsi dalam Program

Backend

| Nama Fungsi | Lokasi | Deskripsi |
|---------------------|-----------------------------|--|
| NewGraph | utils/graph.g o | Baca JSON, inisialisasi Graph.Elements dan Graph.Recipes. |
| RecipesFor | utils/graph.g o | Kembalikan resep raw; jika enforceTier, filter bahan dengan tier < target. |
| SearchBFS | search/bfs.go | BFS lengkap; build pohon resep, return SearchResult. |
| SearchDFS | search/dfs.go | DFS lengkap; backtracking untuk temukan satu resep, return SearchResult. |
| SearchBFSWithPre | search/multi_ bfs.go | BFS lokal seeded; digunakan oleh SearchBFSMultiple. |
| SearchBFSMultiple | search/multi_ bfs.go | Paralel BFS per combo awal; deduplikasi hingga maxRecipes. |
| SearchDFSWithPre | search/multi_ dfs.go | DFS lokal seeded; digunakan oleh SearchDFSMultiple. |
| SearchDFSMultiple | search/multi_ dfs.go | Paralel DFS per combo awal; deduplikasi hingga maxRecipes. |
| BuildTreeFromPre | search/common .go | Rekonstruksi *Node tree berdasarkan pre map. |
| SearchBidirectional | search/bidire ctional.go | Dua arah BFS dari target dan elemen dasar, bangun pohon jika bertemu. |

| | | |
|-----------------------------|-------------------------------|---|
| SearchBidirectionalWithSeed | search/multi_bidirectional.go | Dua arah BFS per seed; paralelisasi dan deduplikasi hingga maxRecipes. |
| BuildTreeFromPre | search/common.go | Rekonstruksi *Node tree berdasarkan pre map. |
| BuildTreeFromPreSafe | search/common.go | Versi aman dari rekonstruksi tree; mencegah loop pada bidirectional. |
| SetupRouter | router/router.go | Definisikan endpoint /search, bind JSON, dispatch ke fungsi search, kembalikan JSON result. |

Frontend

| Nama Fungsi | Lokasi | Deskripsi |
|-------------|---------------------------|---|
| RecipeTree | components/RecipeTree.jsx | Fungsi ini merender diagram pohon resep menggunakan Mermaid.js berdasarkan data yang diberikan. |

4.1.3 Prosedur dalam Program

Backend

- **Startup**
 - main.go memanggil scraper.ScrapeToFile, lalu utils.NewGraph, kemudian router.SetupRouter, dan menjalankan HTTP server.
- **Permintaan Pencarian**
 - Client kirim POST /search dengan JSON payload.
 - Gin mem-ShouldBindJSON ke RequestBody.
 - Validasi input (target ada di graph, mode & algorithm valid).
- **Penanganan Single-Mode**
 - Jika mode=="single", panggil SearchBFS, SearchDFS, atau SearchDFS.
 - Hasil tunggal ([]SearchResult{ ... }) direct di-c.JSON.
- **Penanganan Multiple-Mode**
 - Jika mode=="multiple", validasi max_recipes>0.


- Panggil `SearchBFSMultiple`, `SearchDFSMultiple`, `SearchBidirectionalMultiple`.
- Kumpulkan hingga `maxRecipes` unik, kembalikan array.
- **Pengembalian Hasil**
 - Server mengembalikan `200 OK + JSON list SearchResult`.
 - Error selama proses → `500 Internal Server Error + pesan`.

Frontend

| Nama Prosedur | Lokasi | Deskripsi |
|-----------------------------------|--|--|
| <code>handleSubmit</code> | <code>components/RecipeForm.jsx</code> | Fungsi untuk menangani submit formulir pencarian dan mengirimkan permintaan ke backend dan menyimpan hasilnya. |
| <code>handleToggleChange</code> | <code>components/RecipeForm.jsx</code> | Fungsi untuk mengganti mode pencarian antara "single" dan "multiple". |
| <code>handleAlgorithmClick</code> | <code>components/RecipeForm.jsx</code> | Fungsi untuk memilih algoritma pencarian (BFS, DFS, atau Bidirectional). |
| <code>StatsPanel</code> | <code>components/StatsPanel.jsx</code> | Prosedur yang digunakan untuk menampilkan informasi statistik seperti jumlah node yang dikunjungi, waktu yang dibutuhkan, dan jumlah resep yang ditemukan. |
| <code>Home</code> | <code>pages/index.jsx</code> | Prosedur utama yang merender halaman utama aplikasi dan menerima hasil pencarian dari <code>RecipeForm</code> dan menampilkannya. |

4.2 Penjelasan Tata Cara Penggunaan Program

4.2.1 Interface Program



The screenshot shows the 'Little Alchemy 2 Recipe Finder' web interface. At the top, the title 'Little Alchemy 2 Recipe Finder' is displayed in a pink font. Below the title, there is a small icon of a water drop and a fire flame with the text 'Air dan Api' between them. The interface includes a 'Target Element:' text input field. Below this is an 'Algoritma:' section with three buttons: 'BFS' (highlighted in pink), 'DFS', and 'Bidirectional'. There is also a 'Multiple Recipe:' toggle switch, currently set to 'off'. A pink 'Search' button is located below the toggle. In the bottom left corner, there is a small circular icon with the letter 'N'.

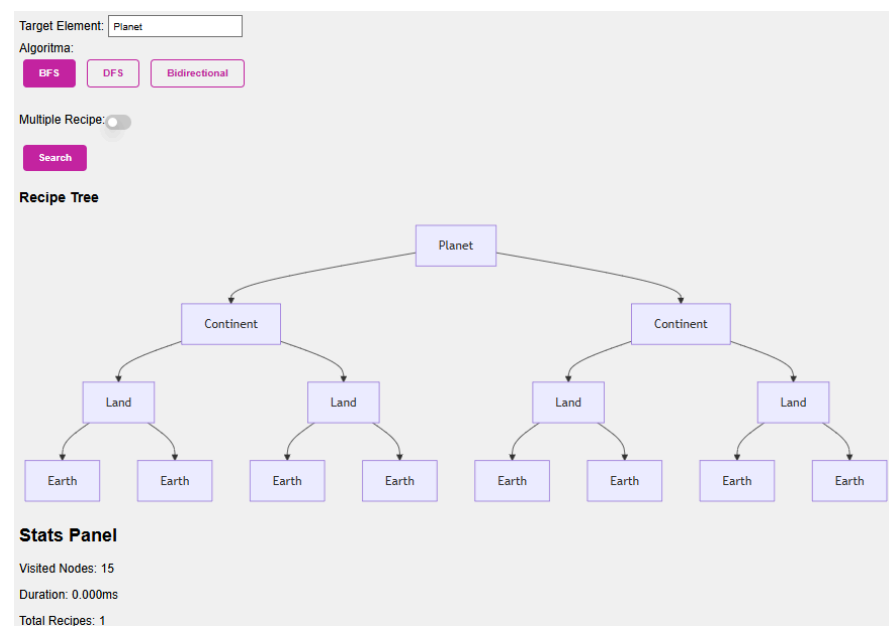
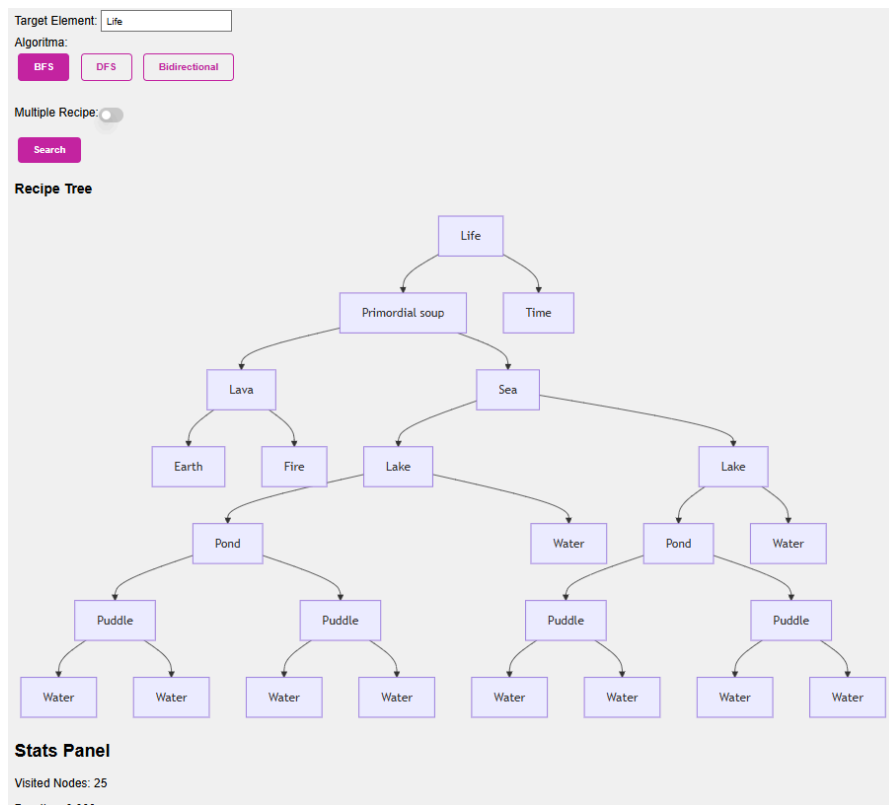
4.2.2 Fitur-Fitur Program

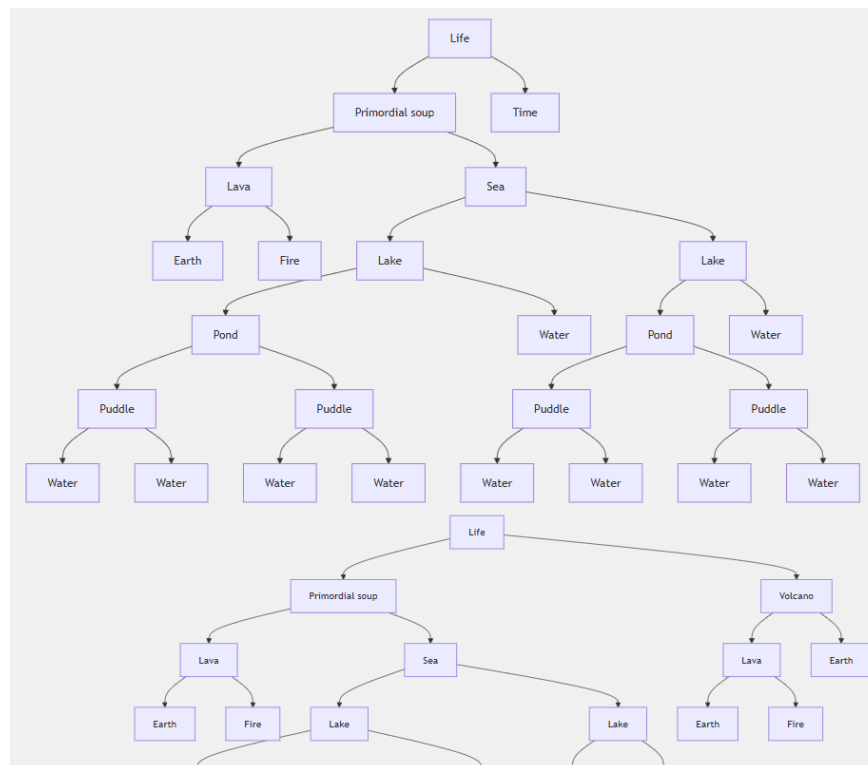
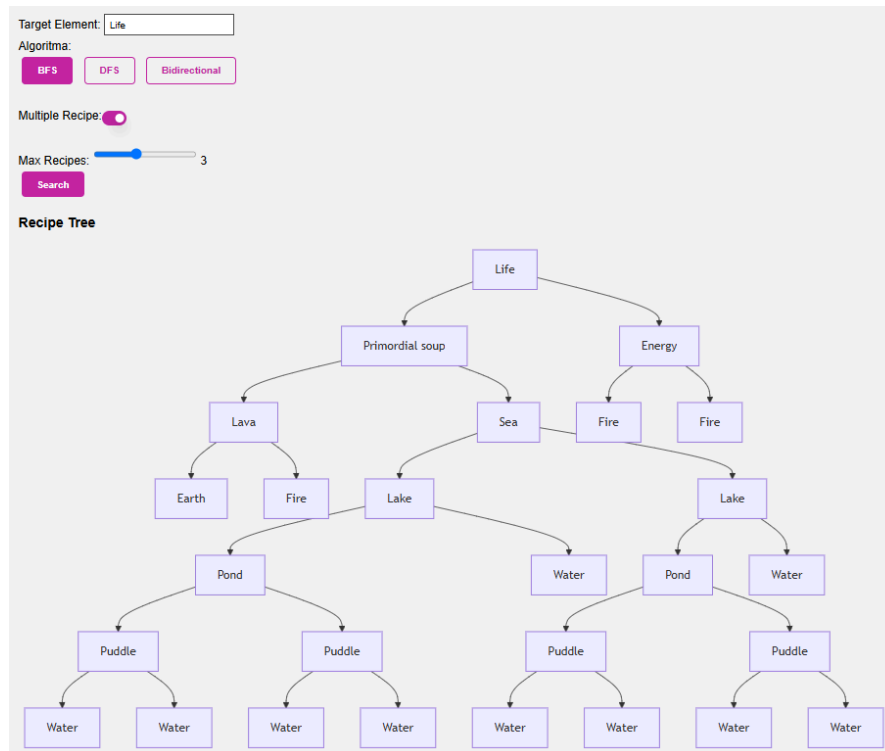
Program ini membantu mencari resep untuk membuat elemen tertentu di game *Little Alchemy 2* berdasarkan algoritma pencarian seperti BFS, DFS, dan Bidirectional Search. Berikut adalah fitur-fitur dan fungsinya.

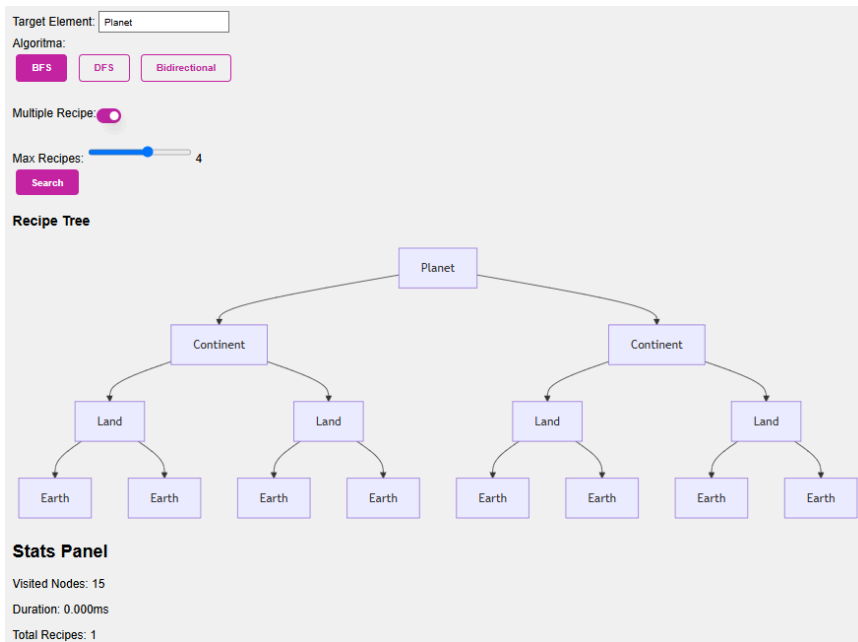
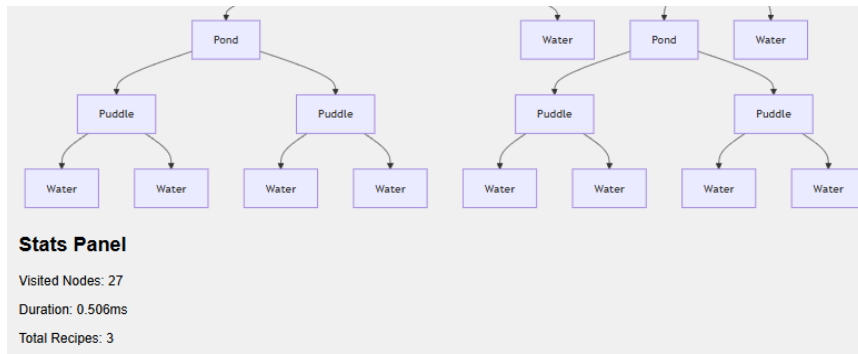
- Target Element : kolom ini digunakan untuk memasukkan nama elemen yang ingin dicari resepnya. Contohnya “Meteoroid”, “Avalanche”, dan lain-lain.
- Algoritma (Tombol Pilihan) : memilih algoritma pencarian yang digunakan yaitu BFS (Breadth-First Search) biasanya lebih cepat untuk menemukan jalur terpendek, DFS (Depth-First Search) dengan menjelajah hingga ke kedalaman terlebih dahulu, atau Bidirectional dengan mencari dari dua arah sekaligus biasanya lebih efisien.
- Multiple Recipe (*toggle switch*) : jika off (default), maka hanya menampilkan satu resep untuk membuat elemen target. Sedangkan, jika on, maka menampilkan semua kemungkinan resep yang tersedia untuk elemen tersebut.
- Search : setelah mengisi Target Element dan memilih algoritma, klik tombol ini untuk memulai pencarian resep dan nantinya hasil akan ditampilkan di bawahnya.

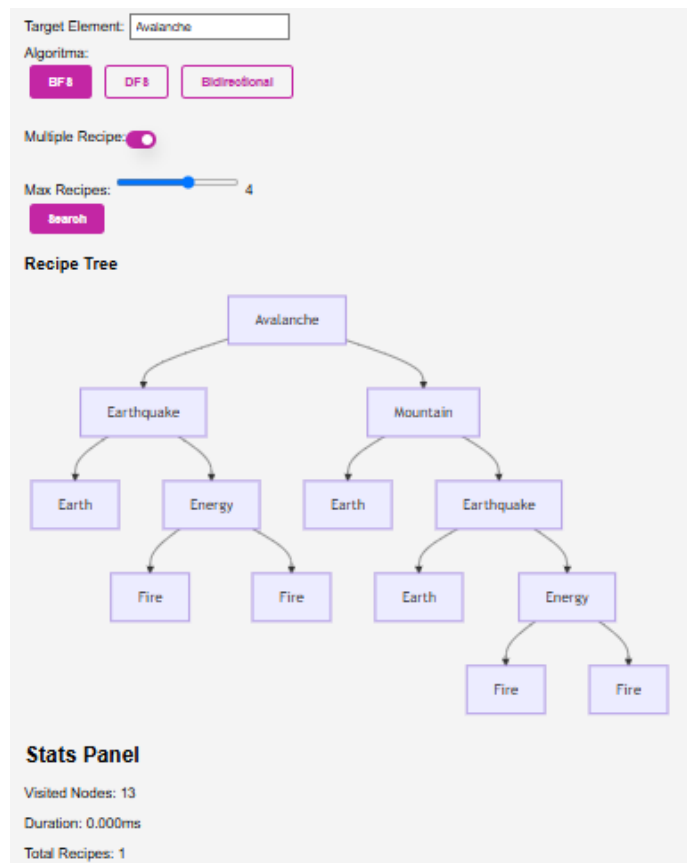
4.3 Hasil Pengujian

1. Single BFS

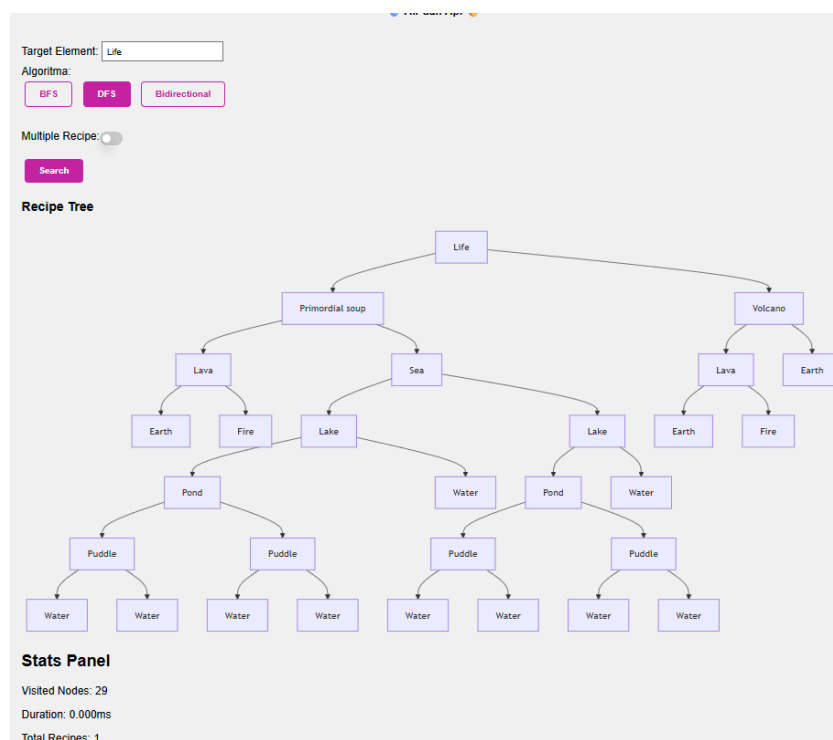


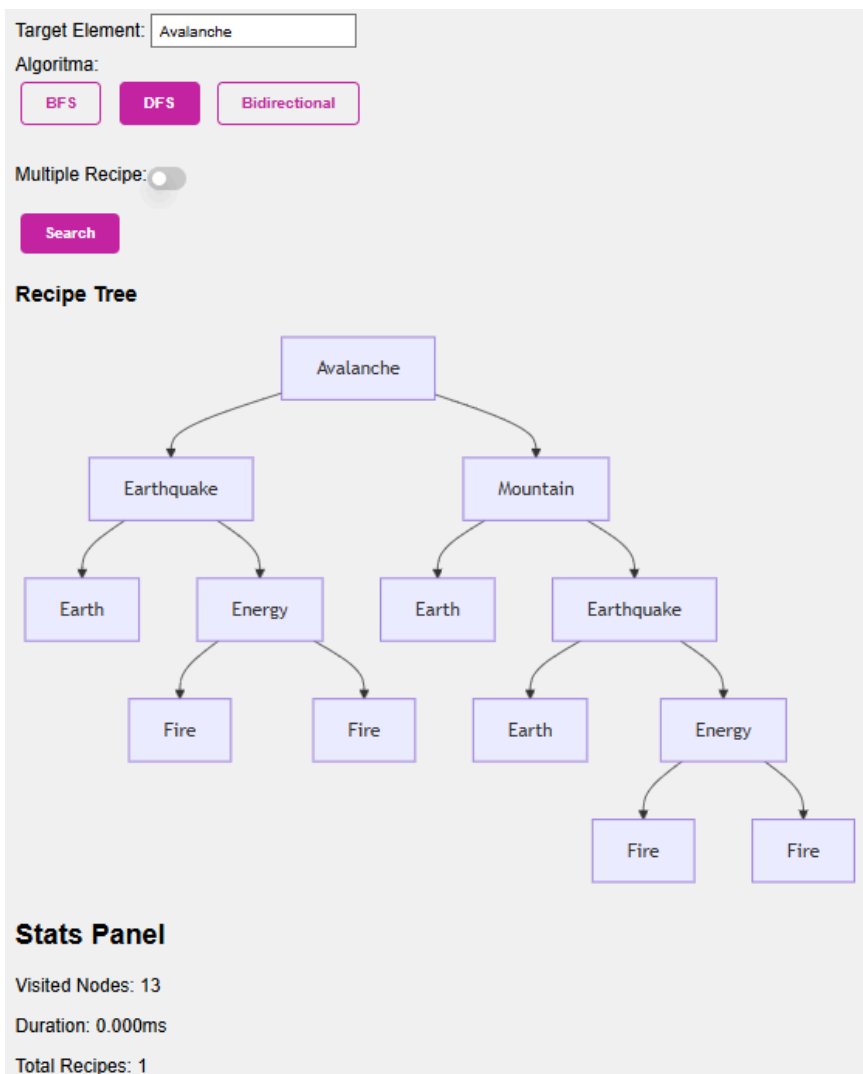
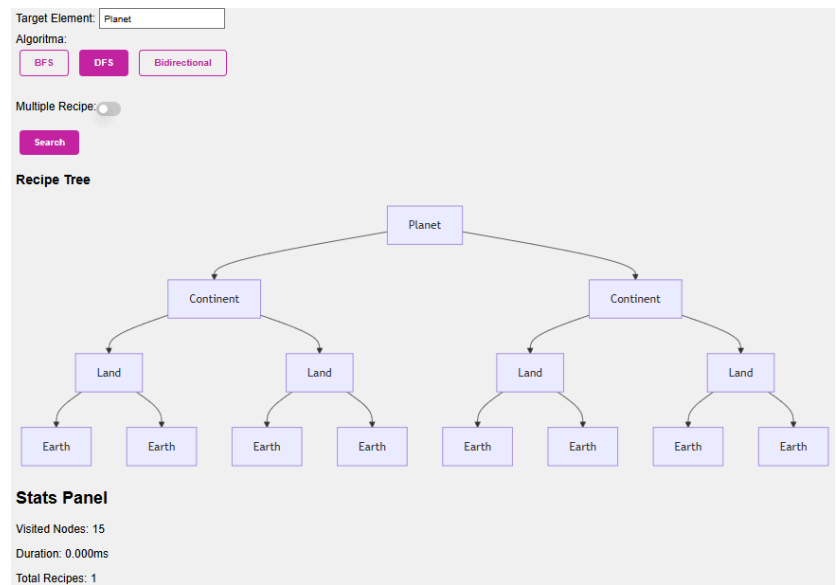






3. Single DFS





Target Element:

Algorithm:

Multiple Recipe: ☐

Recipe Tree

Stats Panel

Visited Nodes: 111
Duration: 0.000ms
Total Recipes: 1

4. Multiple DFS

Little Alchemy 2 Recipe Finder

🔥 Air dan Api 🔥

Target Element:

Algorithm:

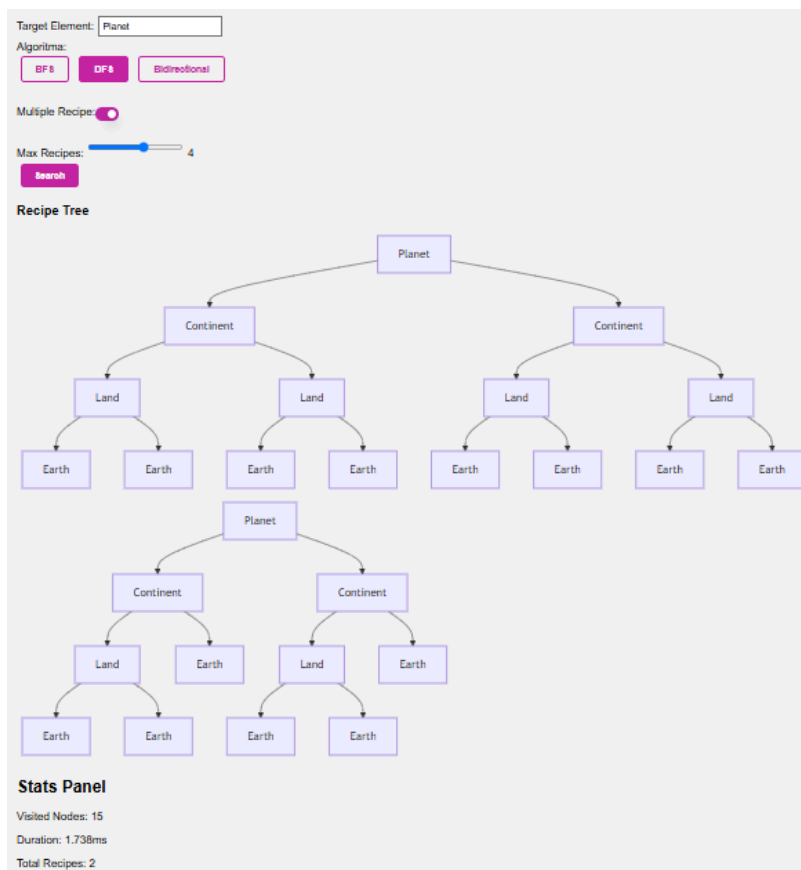
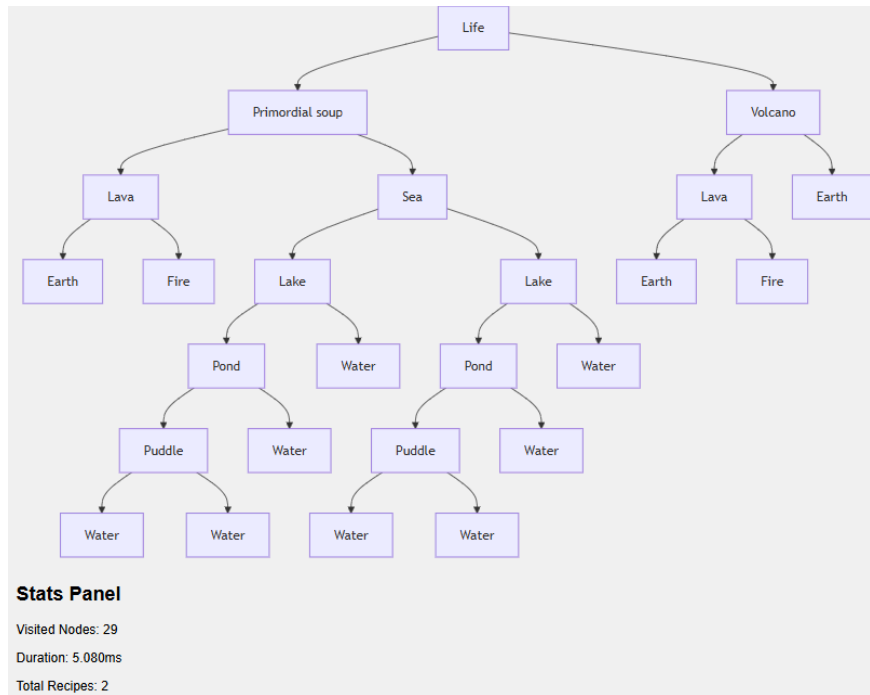
Multiple Recipe: ☒

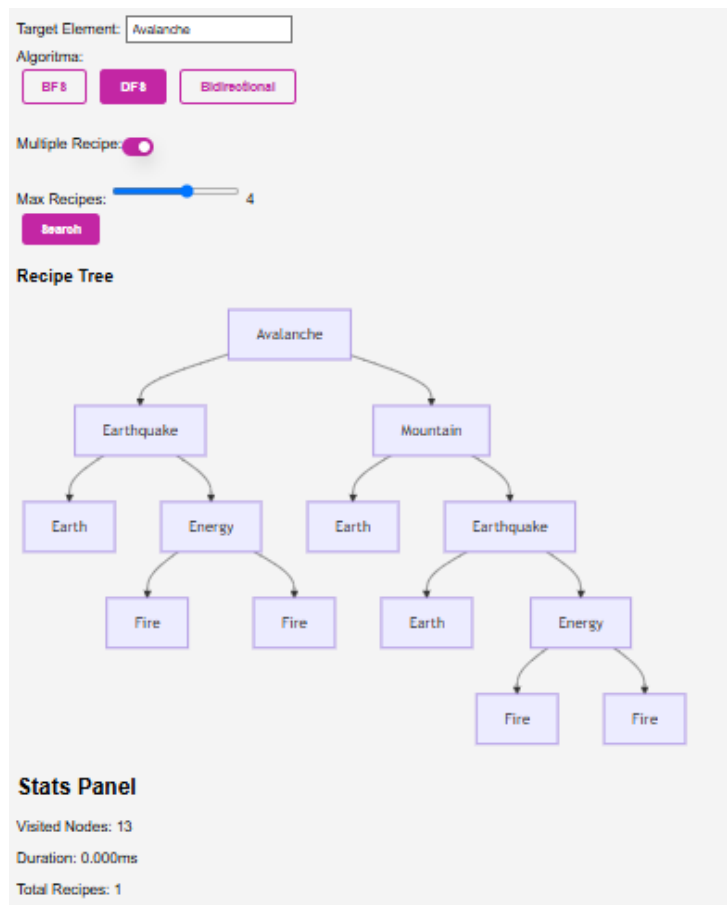
Max Recipes:

Recipe Tree

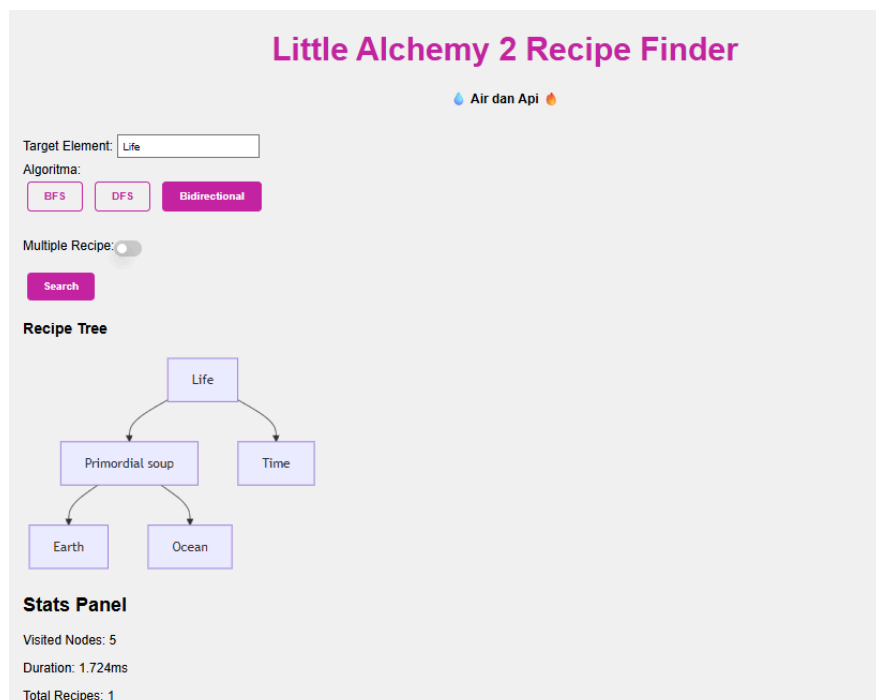
```

graph TD
    Life --> Primordial_soup[Primordial soup]
    Life --> Volcano
    Primordial_soup --> Lava
    Primordial_soup --> Sea
    Volcano --> Lava
    Volcano --> Earth
    Lava --> Earth
    Lava --> Fire
    Sea --> Lake
    Sea --> Lake
    Lake --> Pond
    Lake --> Water
    Pond --> Puddle
    Pond --> Puddle
    Water --> Puddle
    Water --> Puddle
    Puddle --> Water
    Puddle --> Water
    Puddle --> Water
    Puddle --> Water
  
```





5. Single Bidirectional



Target Element: Planet

Algoritma:

BFS

DFS

Bidirectional

Multiple Recipe: ☐

Search

Recipe Tree

```

graph TD
    Planet --> Earth
    Planet --> Space

```

Stats Panel

Visited Nodes: 3

Duration: 0.511ms

Total Recipes: 1

Target Element: Avalanche

Algoritma:

BFS

DFS

Bidirectional

Multiple Recipe: ☐

Search

Recipe Tree

```

graph TD
    Avalanche --> Earthquake
    Avalanche --> Mountain
    Mountain --> Earth
    Mountain --> Earthquake

```

Stats Panel

Visited Nodes: 5

Duration: 1.708ms

Total Recipes: 1

Target Element: Lawn mower

Algoritma:

BFS

DFS

Bidirectional

Multiple Recipe: ☐

Search

Recipe Tree

```

graph TD
    Lawn_mower[Lawn mower] --> Grass
    Lawn_mower --> Tool
    Grass --> Plant
    Grass --> Earth
    Plant --> Seed
    Plant --> Earth

```

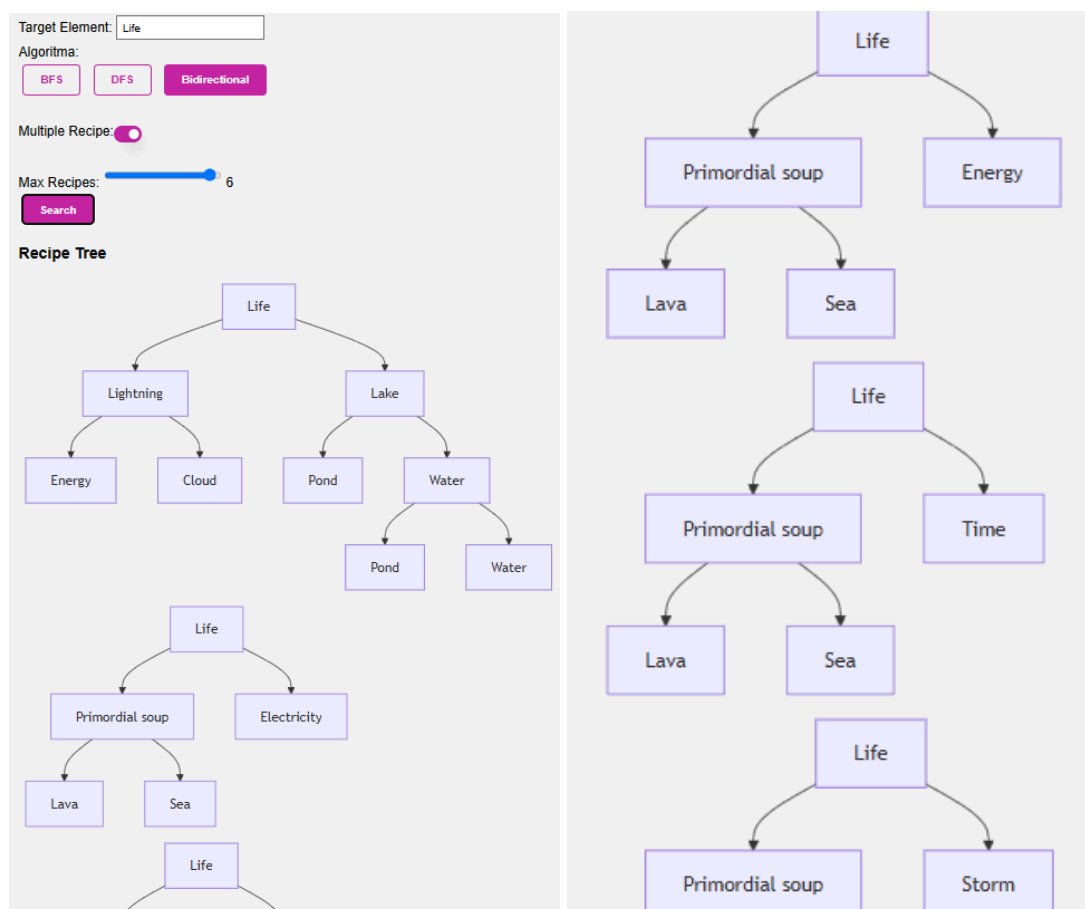
Stats Panel

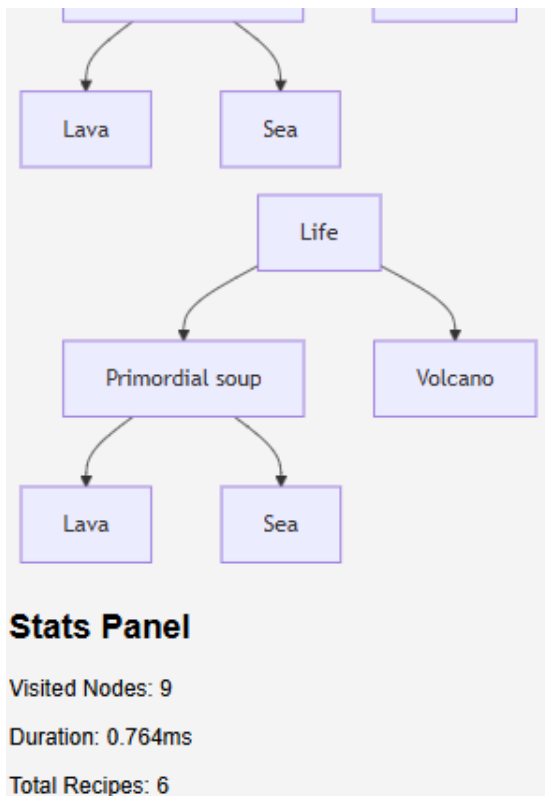
Visited Nodes: 7

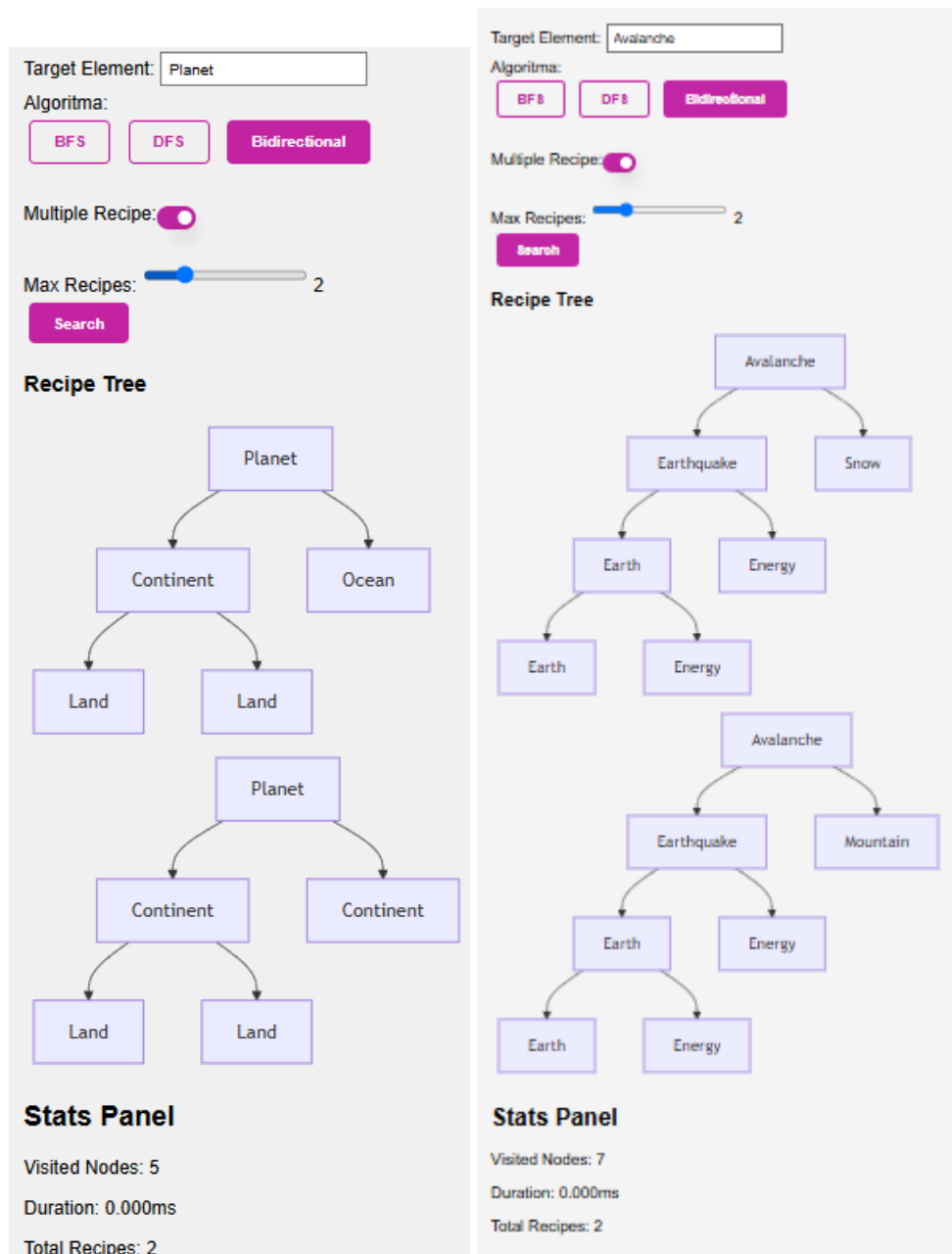
Duration: 0.000ms

Total Recipes: 1

6. Multiple Bidirectional







4.4 Analisis Hasil Pengujian

1. Single BFS

Pada pencarian dengan algoritma BFS, visited nodes sebanyak 25 menunjukkan bahwa BFS mencakup sejumlah node yang cukup banyak. Durasi yang sangat cepat (0.000ms) menunjukkan efisiensi dalam pencarian untuk satu resep saja. Meskipun jumlah node yang dikunjungi relatif banyak, pencarian ini tetap efisien.

2. Multiple BFS

Pada mode multiple BFS, 3 resep berhasil ditemukan setelah mengunjungi 27 node. Durasi pencarian sedikit lebih lama dibandingkan dengan pencarian satu resep (0.506ms), namun pencarian ini masih cukup efisien. Dapat dilihat bahwa pencarian dengan 3 resep tidak meningkatkan secara signifikan jumlah node yang dikunjungi atau waktu pencarian.

3. Single DFS

Dengan Depth-First Search (DFS), pencarian satu resep menghasilkan 29 node yang dikunjungi. Durasi pencarian tetap cepat (0.000ms), namun DFS mengunjungi lebih banyak node dibandingkan dengan BFS pada pencarian satu resep. Hal ini menandakan bahwa DFS lebih mendalam dan mengeksplorasi lebih banyak cabang dalam proses pencarian.

4. Multiple DFS

Pada mode multiple DFS, pencarian menghasilkan 2 resep dengan mengunjungi 29 node. Durasi pencarian meningkat cukup signifikan menjadi 5.080ms. Meskipun jumlah node yang dikunjungi sama dengan pencarian single DFS, durasi pencarian menjadi lebih lama karena pencarian dilakukan untuk dua resep.

5. Single Bidirectional

Dengan Bidirectional Search, pencarian hanya mengunjungi 5 node untuk menemukan 1 resep. Meskipun durasi pencarian sedikit lebih lama (1.724ms), pencarian hanya memerlukan sedikit node yang dikunjungi dibandingkan dengan BFS atau DFS. Ini menunjukkan bahwa Bidirectional Search lebih efisien dalam hal pencarian, terutama jika hanya mencari satu resep.

6. Multiple Bidirectional

Dalam mode multiple Bidirectional, algoritma ini berhasil menemukan hingga 6 resep dengan 9 node yang dikunjungi. Durasi pencarian sangat cepat (0.764ms). Meskipun pencarian dilakukan untuk 6 resep, algoritma Bidirectional tetap efisien dengan mengunjungi hanya sedikit node.

Berdasarkan analisis dari enam jenis pengujian di atas, algoritma untuk pencarian resep yang paling efisien adalah Bidirectional dalam hal jumlah node yang dikunjungi dan durasi sehingga pencarian dapat dilakukan lebih cepat dan mengunjungi lebih sedikit node dibandingkan algoritma lainnya. BFS dan DFS mengunjungi banyak node terutama DFS yang mengunjungi paling banyak node karena sifat pencarian yang mendalam. Pada pencarian *multiple recipe*, waktu pencarian meningkat cukup signifikan seiring dengan bertambahnya jumlah resep yang dicari.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1 Kesimpulan

Web yang kami buat dapat mengeluarkan resep dalam bentuk pohon untuk berbagai elemen yang ada di Little Alchemy 2. Pencarian dapat dilakukan dengan menggunakan BFS, DFS, dan Bidirectional. Resep yang dikeluarkan dapat single maupun multiple. Banyaknya node yang dikunjungi dalam pencarian dan durasi pencarian juga ditampilkan dalam webnya. Proses pencarian dilakukan dengan duration yang sangat cepat meskipun merupakan elemen dengan tier yang cukup tinggi. Tree yang didapat dari Bidirectional memberikan node yang jauh lebih sedikit dibandingkan BFS dan DFS karena proses pencarian yang dilakukan dua arah.

5.2 Saran

Untuk elemen dengan tier tinggi backend berhasil mengeluarkan resepnya tetapi ada beberapa elemen dengan tier tinggi yang tidak dapat ditampilkan tree nya karena terlalu banyak node yang dikunjungi dalam proses pencarian resepnya. Selain itu, frontend masih dapat dikembangkan agar tampilannya lebih menarik dan bisa menambahkan lebih banyak informasi dari resep-resep yang didapat. Selain itu, backend masih dapat dikembangkan karena masih belum banyak mengcover kasus-kasus khusus. Ketika run aplikasi dan terjadi error maka perlu run ulang backend dan frontend nya agar dapat kembali berhasil menampilkan aplikasi.

5.3 Refleksi

Pentingnya untuk selalu menyamakan ide agar saat proses integrasi backend dan frontend dapat berjalan lancar. Selain itu, pembagian tugas untuk backend dan frontend jika sekelompok bertiga perlu didiskusikan dengan jelas.

LAMPIRAN

Berikut lampiran *checklist*.

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

Berikut lampiran tautan.

| | |
|-------------------|---|
| Repository GitHub | https://github.com/qodriazka/Tubes2_Air-dan-API |
| Link YouTube | https://youtu.be/eTSIpx8-PMc |

DAFTAR PUSTAKA

- Bhardwaj, A., & Verma, P. (2017). *Design and Analysis of Algorithm*. Alpha Science.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (1992). *Introduction to Algorithms*. MIT Press.
- Coursera. (2019). Algorithmic Toolbox. Diambil dari <https://www.coursera.org/learn/algorithmic-toolbox>
- GeeksforGeeks. (2025). Bidirectional Search. Diambil dari <https://www.geeksforgeeks.org/bidirectional-search/>
- Levitin, A. (2003). *Introduction to the Design & Analysis of Algorithms*. Addison-Wesley.
- Munir, R. (2025). *Bahan Kuliah IF2251 Strategi Algoritmik*. ITB.
- Pogančić, M. V. (2019). The Branch and Bound Algorithm. *Towards Data Science*.
- Russell, S. J., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.