# Laporan Tugas Besar 2
# IF2211 Strategi Algoritma 2025/2026

Disusun oleh:

Kelompok 20 - AVATAR

| | |
|---|---|
| Dave Daniell Yanni | (13523003) |
| Bryan Ho | (13523029) |
| Daniel Pedrosa Wu | (13523099) |

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


2025

# DAFTAR ISI


**DAFTAR ISI** **2**
**BAB I** **3**
**DESKRIPSI TUGAS** **3**
**BAB II** **5**
**LANDASAN TEORI** **5**
2.1 Penjelajahan Graf 5
2.2 Breadth First Search (BFS) 5
2.3 Depth First Search (DFS) 6
2.4 Penjelasan Aplikasi Web yang Dibangun 6
**BAB III** **7**
**ANALISIS PEMECAHAN MASALAH** **7**
3.1 Langkah-langkah Pemecahan Masalah 7
3.2 Proses pemetaan masalah menjadi elemen-elemen algoritma DFS dan BFS 7
3.3 Fitur Fungsional 9
3.4 Arsitektur Website (Front-End) 10
3.5 Arsitektur Program (Back-End) 10
3.6 Contoh ilustrasi kasus 11
**BAB IV** **12**
**IMPLEMENTASI DAN PENGUJIAN** **12**
4.1 Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun) 12
4.2 Penjelasan Tata Cara Penggunaan Program 35
4.3 Hasil Pengujian 37
4.4 Analisis Hasil Pengujian 43
**BAB V** **44**
**KESIMPULAN, SARAN, DAN REFLEKSI** **44**
5.1 Kesimpulan 44
5.2 Saran 44
5.3 Refleksi 45
**LAMPIRAN** **46**
1. Daftar Pustaka 46
2. Tautan Repository 46
3. Tautan Video 46
4. Tabel Evaluasi 47

# BAB I
# 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.

# BAB II
# LANDASAN TEORI

## 2.1 Penjelajahan Graf

Penjelajahan graf (*graph traversal*) adalah proses sistematis dalam mengunjungi setiap simpul (*node*) dan sisi (*edge*) dalam sebuah graf, dengan tujuan untuk mengeksplorasi struktur graf atau menyelesaikan permasalahan yang direpresentasikan oleh graf tersebut.

Terdapat dua metode dasar dalam penjelajahan graf:

1. Pencarian Melebar (*Breadth First Search* / BFS)
   BFS menjelajahi graf secara bertingkat, dimulai dari simpul awal, kemudian mengunjungi semua simpul yang bertetangga langsung (level 1), dilanjutkan dengan simpul-simpul tetangga dari simpul level 1 (level 2), dan seterusnya.

2. Pencarian Mendalam (*Depth First Search* / DFS)
   DFS menjelajahi graf secara mendalam dengan memilih satu simpul tetangga untuk dijelajahi terlebih dahulu hingga mencapai simpul tanpa tetangga yang belum dikunjungi. Setelah itu, penjelajahan kembali ke simpul sebelumnya (backtracking) dan melanjutkan ke jalur lainnya.

Graf sering digunakan sebagai representasi berbagai jenis permasalahan, seperti pencarian rute, analisis jaringan sosial, pemrosesan gambar, dan lain sebagainya. Oleh karena itu, penjelajahan graf menjadi langkah penting dalam menemukan solusi atas persoalan-persoalan tersebut.

## 2.2 *Breadth First Search* (BFS)

*Breadth First Search* merupakan salah satu algoritma dasar dalam penelusuran graf. Algoritma ini bekerja dengan memulai penelusuran dari sebuah simpul awal, kemudian mengunjungi seluruh simpul yang bertetangga secara langsung terlebih dahulu. Setelah seluruh simpul tetangga pada tingkat pertama ($n$) dikunjungi, penelusuran dilanjutkan ke simpul-simpul pada tingkat berikutnya ($n$+1), yaitu tetangga dari simpul-simpul yang telah dikunjungi sebelumnya.

BFS memiliki karakteristik berbeda dengan *Depth First Search* (DFS), di mana penelusuran dilakukan dengan mendahulukan simpul-simpul yang paling dekat dari simpul awal. Dengan demikian, BFS secara umum menelusuri graf secara bertingkat (level by level).

Beberapa algoritma graf yang terkenal, seperti algoritma pencarian jalur terpendek Dijkstra, Algoritma Kahn untuk topological sorting, serta Algoritma Prim untuk pencarian Minimum Spanning Tree, merupakan algoritma yang berbasis pada konsep BFS.

## 2.3 *Depth First Search* (DFS)

*Depth First Search* juga merupakan salah satu algoritma dasar dalam penelusuran graf. Dalam metode DFS, penelusuran dilakukan dengan menjelajahi simpul-simpul yang berdekatan secara mendalam (*depth-wise*). Artinya, dari satu simpul awal, algoritma akan menelusuri salah satu simpul tetangganya, kemudian dari simpul tersebut menelusuri tetangganya lagi, dan seterusnya, hingga mencapai simpul yang tidak memiliki tetangga yang belum dikunjungi. Setelah itu, penelusuran kembali ke simpul sebelumnya (*backtracking*) untuk melanjutkan penelusuran ke simpul tetangga lainnya yang belum dikunjungi.

## 2.4 Penjelasan Aplikasi Web yang Dibangun

Aplikasi ini merupakan platform berbasis web yang dirancang untuk membantu pengguna mencari resep kombinasi elemen dalam permainan Little Alchemy 2. Sistem memungkinkan pencarian resep menggunakan dua strategi penelusuran graf utama, yaitu *Breadth First Search* (BFS) dan *Depth First Search* (DFS).

Data elemen dan resep diperoleh melalui proses web scraping dari situs Fandom Little Alchemy 2 (https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)), dan akan disimpan serta diolah dalam sistem backend.

Fitur utama aplikasi meliputi:

- Pemilihan algoritma pencarian (BFS dan DFS).
- Parameter pencarian untuk menentukan jumlah maksimum resep yang ingin ditemukan.
- Menyediakan visualisasi pertumbuhan pohon secara real-time.
- Memiliki parameter penundaan waktu untuk mengatur kecepatan pertumbuhan pohon.
- Optimasi multithreading untuk pencarian multiple recipe agar proses lebih efisien.
- Validasi tier yang memastikan bahwa elemen pembentuk selalu memiliki tier lebih rendah dari elemen yang dibentuk.
- Visualisasi hasil pencarian dalam bentuk tree yang menggambarkan kombinasi elemen dari level dasar hingga elemen target.
- Tampilan statistik hasil pencarian seperti waktu pencarian dan jumlah simpul (*node*) yang dikunjungi.

# BAB III
# ANALISIS PEMECAHAN MASALAH

## 3.1 Langkah-langkah Pemecahan Masalah

Sistem ini dirancang untuk menemukan resep pembentukan suatu elemen berdasarkan struktur data graf. Berikut adalah langkah-langkah pemecahan masalah:

1. Memahami Struktur Masalah
    - Setiap element memiliki recipes (kombinasi 2 bahan).
    - Membentuk target, yaitu menemukan cara membentuk suatu element menggunakan kombinasi bahan dasar (tier 0) melalui berbagai resep. Resep ini dapat dimodelkan sebagai suatu pohon biner.
2. Membangun Representasi Data
    - Kumpulan elemen diletakkan di dalam sebuah ElementGraph yang berisi relasi antara elemen-elemen.
    - Graf tersebut akan berisi simpul-simpul yang merupakan ElementNode.
    - Relasi antara suatu ElementNode dengan ElementNode lain direpresentasikan dengan ElementRelation dengan satu simpul yang menjadi simpul TargetNode dan banyak simpul SourceNode
3. Memilih Algoritma Pencarian
    - DFS untuk melakukan traversal dalam
    - BFS untuk melakukan traversal level-by-level
4. Menentukan Validitas
    - Validasi tier untuk mencegah elemen lebih tinggi membentuk elemen yang lebih rendah.
    - Batasi jumlah resep (misal, maksimal 5) untuk efisiensi.
5. Optimasi Proses
    - Menggunakan multithreading dan caching hasil pencarian.
    - Tree cloning dan pooling untuk menghemat alokasi memori.
6. Visualisasi Hasil
    - Menyusun hasil dalam bentuk tree (TreeNode) yang menggambarkan hirarki resep.

## 3.2 Proses pemetaan masalah menjadi elemen-elemen algoritma DFS dan BFS

Ada tiga prasyarat yang diletakkan untuk memudahkan proses pencarian ini, yaitu bahwa suatu *recipe* harus terdiri dari elemen-elemen dengan *tier* lebih rendah daripada elemen yang ingin dibentuk, sebuah *recipe* tidak akan selalu memiliki rute terpendek,

dan harus bisa memungkinkan *multithreading*. Melalui *scrapping* yang dilakukan pada wiki Little Alchemy 2, tiap elemen dipetakan sebagai suatu simpul dalam graf berarah yang di dalamnya juga mengandung informasi mengenai elemen-elemen apa saja yang dapat membentuknya. Graf ini dapat ditelusuri untuk menentukan *recipe* dari elemen *tier* 0 hingga elemen yang ditujukan.

Karena sudah dipastikan bahwa suatu *recipe* terdiri dari elemen-elemen dari *tier* yang lebih rendah, maka permasalahan ini dapat dimodelkan sebagai suatu pohon, yakni sebuah graf terhubung yang asiklik. Hasil yang dicari adalah satu atau lebih *recipe* yang independen dari elemen dasar ke elemen yang ditujukan. Dengan menggunakan fakta bahwa suatu resep terdiri dari tepat dua elemen, maka suatu resep independen adalah suatu pohon biner dengan daunnya sepenuhnya terdiri dari elemen dasar (*tier* 0).

Untuk mencari resep tersebut, ada dua cara yang dapat digunakan yaitu bermulai dari elemen yang ditujukan atau bermulai dari elemen dasar. Karena permasalahannya dapat dimodelkan dengan pohon biner dan suatu elemen terbentuk dari dua elemen dengan *tier* yang lebih rendah maka lebih cocok bermulai dari elemen yang ditujukan. Dengan memulai dari elemen yang ditujukan, penelusuran dapat dilakukan ke *recipe-recipe*-nya, dan *sub recipe* dari *recipe* tersebut juga ditelusuri hingga mencapai elemen dasar.

Untuk menemukan suatu resep, algoritma yang dapat digunakan adalah algoritma Depth-First Search dan Breadth-First Search. Elemen yang ditujukan dijadikan sebagai simpul akar. Dengan menggunakan algoritma DFS ataupun BFS, setiap *recipe* unik dari elemen simpul akan ditelusuri. Tiap *recipe* terdiri dari dua elemen. Jika elemen tersebut bukan elemen dasar, maka telusuri lagi *sub recipe* tersebut hingga mencapai elemen dasar. Tiap cabang unik yang daun-daunnya adalah elemen dasar akan dijadikan satu *recipe* independen.

Perbedaan DFS dan BFS akan menyebabkan cara penelusuran dari graf yang berbeda. Pada DFS, algoritma akan memiliki satu simpul anak dan akan menelusuri anak dari simpul anak tersebut hingga mencapai elemen dasar. Kemudian algoritma akan *backtrack* ke cabang sebelum-sebelumnya jika selesai menelusuri suatu jalur. Berbeda dengan DFS, BFS akan menelusuri *level by level*. Pada tiap simpul, algoritma akan menelusuri tiap *recipe* yang berada langsung di bawahnya. Setelah selesai, maka ia akan berlanjut ke level berikutnya dan menelusuri seluruh level tersebut hingga pada akhirnya menemukan apa yang dicari. Alhasil, pohon pencarian yang dihasilkan DFS akan dalam sementara pohon perncarian yang dihasilkan BFS akan lebar.

### 3.3 Fitur Fungsional

Aplikasi ini menyediakan sejumlah fitur utama yang dirancang untuk membantu pengguna mengeksplorasi dan memahami cara pembentukan elemen secara interaktif dan efisien. Berikut adalah fitur-fitur fungsional aplikasi yang dirancang untuk mendukung pencarian dan eksplorasi resep elemen.

1. Pencarian Resep
   Aplikasi memungkinkan pengguna untuk mencari jalur pembentukan suatu elemen target dari elemen-elemen dasar. Sistem akan secara otomatis menelusuri berbagai kombinasi resep yang memungkinkan, lalu menyusun jalur pembuatan yang valid sesuai dengan struktur data yang ada.

2. Pemilihan Algoritma Pencarian
   Pengguna dapat memilih algoritma yang digunakan untuk proses pencarian, yaitu antara DFS (Depth-First Search) atau BFS (Breadth-First Search). DFS cocok untuk menemukan berbagai variasi jalur, sementara BFS ideal untuk menemukan jalur paling pendek.

3. Dukungan Multi-solusi
   Fitur ini memungkinkan pengguna untuk meminta lebih dari satu alternatif resep menuju elemen yang sama. Hal ini bermanfaat saat pengguna ingin mengeksplorasi seluruh kemungkinan atau membandingkan efisiensi kombinasi yang berbeda.

4. Visualisasi Pohon Resep
   Hasil pencarian resep ditampilkan dalam bentuk visual pohon, yang memperlihatkan bagaimana setiap elemen disusun dari bahan-bahan dasarnya. Visualisasi ini membantu pengguna memahami struktur hierarki kombinasi secara intuitif dan menarik.

5. Metrik Performa Pencarian
   Setiap proses pencarian disertai statistik performa berupa jumlah node (simpul) yang dikunjungi serta waktu yang dibutuhkan untuk menyelesaikan pencarian. Data ini memberikan gambaran efisiensi dari metode yang dipilih.

6. Tier List Elemen
   Elemen-elemen dikelompokkan berdasarkan tingkat kompleksitas atau “tier”-nya. Dengan adanya pengelompokan ini, pengguna dapat melihat elemen mana yang bersifat dasar dan mana yang memerlukan kombinasi lanjutan, membantu dalam menyusun strategi kombinasi elemen secara lebih terstruktur.

### 3.4 Arsitektur Website (Front-End)

Website ini dibangun menggunakan React dan Vite dengan TypeScript untuk mengelola logika interaktif di sisi pengguna. Untuk tampilan antarmuka (UI), digunakan Tailwind CSS, yang memungkinkan styling yang efisien dan responsif.

Antarmuka pengguna terdiri dari beberapa komponen utama, yaitu:

- Input Bar: Tempat pengguna memasukkan nama elemen yang ingin dicari.
- Toggle Button: Digunakan untuk memilih metode pencarian yang diinginkan, yaitu antara BFS (*Breadth-First Search*) atau DFS (*Depth-First Search*).
- Max Recipes: Input angka untuk menentukan jumlah maksimal resep/solusi yang ingin ditampilkan.
- Visualisasi Pohon: Menampilkan struktur pencarian dalam bentuk tree, yang menunjukkan langkah-langkah pembentukan elemen.

Struktur source code pada bagian front-end disederhanakan dalam satu folder utama, yaitu src, dengan komponen utama berada dalam App.tsx. Seluruh logika interaksi, pemilihan metode pencarian, serta integrasi API dilakukan dalam komponen ini.

### 3.5 Arsitektur Program (Back-End)

Bagian back-end dibangun menggunakan bahasa pemrograman Go (Golang), dengan fokus pada algoritma pencarian graf BFS dan DFS. Data elemen Little Alchemy 2 diperoleh melalui proses scraping dari situs resmi wiki-nya.

Struktur folder pada backend dibagi menjadi lima bagian utama, yaitu:

- controllers: Berisi implementasi algoritma BFS dan DFS, serta logika pembentukan pohon pencarian untuk visualisasi front-end.
- models: Menyimpan logika pemrosesan data, termasuk filtering (misalnya, sebuah elemen hanya bisa dibuat dari elemen dengan tier lebih rendah).
- data: Menyimpan hasil scraping dalam bentuk data mentah atau terstruktur, yang akan digunakan oleh algoritma pencarian.
- routes: Mengatur endpoint dan routing dari REST API, termasuk endpoint pencarian elemen dan pengambilan data visualisasi pohon.
- scraper: Berisi kode untuk melakukan scraping data dari https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)
- websocket: Memungkinkan visualisasi proses pencarian elemen (BFS/DFS) secara bertahap di sisi frontend seiring proses berjalan di backend.

### 3.6 Contoh ilustrasi kasus

Contoh kasus pencarian untuk elemen “Brick”.

- Menggunakan BFS
    1. Mulai dari elemen target ("Brick")
    2. Eksplorasi resep tingkat pertama, yaitu "Brick" bisa dibuat dari "Mud" dan "Fire"
    3. Eksplorasi semua elemen di tingkat ini, yaitu "Mud" dan "Fire"
    4. Untuk setiap elemen yang bukan elemen dasar, eksplorasi resepnya
    5. "Mud" dibuat dari "Earth" dan "Water", dan "Fire" adalah elemen dasar
    6. Eksplorasi semua elemen di tingkat ini: "Earth" dan "Water"
    7. Jika "Earth" dan "Water" adalah elemen dasar, pencarian selesai

- Menggunakan DFS:
    1. Mulai dari elemen target ("Brick")
    2. Eksplorasi resep tingkat pertama, yaitu "Brick" bisa dibuat dari "Mud" dan "Fire"
    3. Lanjut eksplorasi "Mud" terlebih dahulu (prioritas kedalaman)
    4. "Mud" dibuat dari "Earth" dan "Water"
    5. Eksplorasi "Earth" (prioritas kedalaman)
    6. "Earth" adalah elemen dasar, kembali ke level atas
    7. Eksplorasi "Water"
    8. "Water" adalah elemen dasar, kembali ke level atas
    9. Eksplorasi "Fire"
    10. "Fire" adalah elemen dasar, pencarian selesai

- Perbedaan Hasil:
    1. BFS cenderung menemukan resep dengan jumlah langkah minimal (pohon lebih lebar)
    2. DFS cenderung menemukan resep dengan jalur yang lebih dalam (pohon lebih dalam)

# BAB IV
# IMPLEMENTASI DAN PENGUJIAN

## 4.1 Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun)

Spesifikasi teknik program dibagi menjadi dua, yaitu bagian *scraper* dan bagian *solver*. *Scraping* adalah proses mengambil data dari sumber web. Proses ini dilakukan saat aplikasi web pertama kali dijalankan. *Scraper* sepenuhnya terletak di *scraper.go*. *Scraper* ini menggunakan library Colly untuk melakukan *web scraping* dan menggunakan Goquery untuk manipulasi DOM. *Scraper* mengekstraksi elemen-elemen yang berisi informasi mengenai nama elemen, tingkat (tier), serta resep yang dapat membentuk elemen tersebut. Elemen tersebut dipetakan dalam struktur data Elemen dan Recipe.

```go
type Recipe struct {
    Ingredients []string `json:"ingredients"`
}

type Element struct {
    Name    string   `json:"name"`
    Tier    int      `json:"tier"`
    Recipes []Recipe `json:"recipes"`
}
```

Dengan menggunakan fungsi `scrape`, data yang terkumpul kemudian diubah menjadi format JSON yang lebih mudah untuk diproses lebih lanjut oleh sistem. Dalam hal ini, data yang dikumpulkan mencakup nama elemen, tier-nya, dan kombinasi resep yang dapat menghasilkan elemen tersebut. Setelah scraping selesai, data ini disimpan dalam file JSON yang akan digunakan oleh bagian solver untuk proses perhitungan dan pencarian resep.

```go
func Scrape(filePath string) {
    c := colly.NewCollector(
        colly.UserAgent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"),
        colly.AllowedDomains("little-alchemy.fandom.com"),
        colly.AllowURLRevisit(),
    )

    var elements []Element
    sectionTiers := make(map[string]int)

    sectionCollector := c.Clone()
    sectionCollector.OnHTML("h3 .mw-headline", func(e *colly.HTMLElement) {
        sectionTitle := e.Text
        sectionID := e.Attr("id")

        var tier int
        switch sectionTitle {
        case "Starting elements", "Special element":
            tier = 0
        default:
            if sectionTitle != "" {
                rawSectionTitle := strings.Split(sectionTitle, " ")
                if len(rawSectionTitle) == 3 {
                    tierStr := rawSectionTitle[1]
                    if t, err := strconv.Atoi(tierStr); err == nil {
                        tier = t
                    } else {
                        fmt.Println("Error converting tier string to int:", err)
                        tier = 999
                    }
                } else {
                    fmt.Println("Unexpected section title format:", sectionTitle)
                    tier = 999
                }
            }
        }

        sectionTiers[sectionID] = tier
    })

    if err := sectionCollector.Visit("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"); err != nil {
        log.Fatal(err)
    }

    c.OnHTML("h3", func(h3 *colly.HTMLElement) {
        sectionID := h3.ChildAttr(".mw-headline", "id")
        tier, exists := sectionTiers[sectionID]
        if !exists {
            tier = 999
        }

        h3.DOM.NextUntil("h3").Each(func(i int, s *goquery.Selection) {
            if s.Is("table.list-table") {
                s.Find("tr").Each(func(_ int, row *goquery.Selection) {
                    if row.Find("td:nth-of-type(1)").Text() == "" {
                        return
                    }

```

```go
                    name := row.Find("td:nth-of-type(1)").Text()
                    var recipes []Recipe

                    recipeText := row.Find("td:nth-of-type(2)").Text()
                    if recipeText != "" {
                        rawRecipes := strings.Split(recipeText, "\n")
                        for _, recipe := range rawRecipes {
                            recipe = strings.TrimSpace(recipe)
                            if strings.Contains(recipe, "+") {
                                parts := strings.Split(recipe, "+")
                                var ingredients []string
                                for _, part := range parts {
                                    ingredient := strings.TrimSpace(part)
                                    if ingredient != "" {
                                        ingredients = append(ingredients, ingredient)
                                    }
                                }
                                if len(ingredients) >= 2 {
                                    recipes = append(recipes, Recipe{Ingredients: ingredients})
                                }
                            }
                        }
                    }

                    if name != "" {
                        elements = append(elements, Element{
                            Name:    strings.TrimSpace(name),
                            Tier:    tier,
                            Recipes: recipes,
                        })
                    }
                })
            }
        })
    })

    if err := c.Visit("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"); err != nil {
        log.Fatal(err)
    }

    jsonData, err := json.MarshalIndent(elements, "", "  ")
    if err != nil {
        log.Fatal("Error marshalling to JSON:", err)
    }

    if err := os.WriteFile(filePath, jsonData, 0644); err != nil {
        log.Fatal("Error writing JSON to file:", err)
    }

    fmt.Println("Data successfully saved to", filePath)
}
```

Setelah *scraping* dan disimpan dalam suatu file JSON, program akan membaca JSON tersebut untuk mendapatkan data-data yang diperlukan untuk membangun graf relasi antara elemen. Pemrosesan ini terletak di *elements.go*. Data yang ada di JSON tersebut akan dibaca oleh program dan dipetakan ke dalam struktur data-struktur data ElementNode, ElementGraph, ElementRelation, dan Recipe.

```go
type ElementNode struct {
    Element  *Element
    Children []*ElementRelation
    Parents  []*ElementRelation
}

type ElementRelation struct {
    TargetNode  *ElementNode
    SourceNodes []*ElementNode
    Recipe      Recipe
}

type Recipe struct {
    Ingredients []string `json:"ingredients"`
}

type ElementGraph struct {
    RootNode   *ElementNode
    AllNodes   map[string]*ElementNode
    Tier0Nodes []*ElementNode
}
```

Setiap struktur data memiliki fungsi yang berbeda. ElementNode berfungsi untuk menyimpan simpul-simpul yang ada dalam graf relasi elemen. Relasi ini digambarkan dengan ElementRelation yang menyimpan relasi apa yang ada antara 3 elemen yaitu 1 simpul target dan 2 simpul bahan. ElementGraph menyimpan semua simpul yang ada di graf.

```go
type TreeNode struct {
    Name   string
    Recipe []*TreeNode
}
```

Fungsi utama dari program ini terdapat pada *ElementsController.go*. Modul ini menyimpan fungsi-fungsi yang digunakan untuk melakukan traversal pohon secara DFS dan BFS. BFS dan DFS dijalankan dalam fungsi mereka sendiri, yaitu `dfs()` dan `bfs()`. Fungsi DFS bekerja secara rekursif. Artinya fungsi tersebut memanggil dirinya sendiri secara terus menerus. Biasanya pada fungsi rekursif terdapat kasus basis yang menjadi penghenti pemanggilan dirinya secara terus menerus.

```go
if len(target.Parents) <= 0 || target.Element.Tier == 0 {
    node := &TreeNode{
        Name: target.Element.Name,
    }
    return []*TreeNode{node}
}
```

Pada fungsi ini, kasus basis terjadi saat elemen tidak memiliki anak atau merupakan elemen dasar, maka ia akan mengembalikan elemen tersebut. Jika tidak berada dalam kasus basis, maka ia akan lanjut ke rekurens.

```go
for _, recipe := range target.Parents {
    if len(recipe.SourceNodes) < 2 {
        continue
    }

    atomic.AddInt64(&NodesVisited, 1)
    leftTrees := dfs(recipe.SourceNodes[0], n, treeChan)
    rightTrees := dfs(recipe.SourceNodes[1], n, treeChan)

    for _, left := range leftTrees {
        for _, right := range rightTrees {
            node := &TreeNode{
                Name:   target.Element.Name,
                Recipe: []*TreeNode{left, right},
            }

            treeChan <- node

            results = append(results, node)
            if len(results) >= int(n) {
                return results
            }
        }
        if len(results) >= int(n) {
            return results
        }
    }
```

Dalam kasus rekurens, fungsi akan mengiterasikan setiap resep yang membentuk elemen target. Secara rekursif, ia akan memanggil fungsi `dfs()` untuk elemen yang menjadi elemen pembentuknya. Ini akan membangun level di dalamnya terus menerus. Fungsi kemudian akan menggabungkan tree yang dibentuk dengan node anaknya yang tepat. Karena ini terjadi secara rekursif, maka lama kelamaan ini akan kembali ke elemen awal yang dicari.

Berbeda dengan DFS, BFS bekerja dengan cara yang berbeda. Karena sifatnya yang level by level, implementasinya tidak bisa menggunakan fungsi yang rekursif. Untuk mengelola simpulnya, implementasi menggunakan suatu *queue*. Pada setiap level, fungsi akan memasukkan tiap elemen yang merupakan *ingredients* ke dalam queue.

```go
for _, recipe := range target.Parents {
    currentQueue = append(currentQueue,
        &QueueItem{Element: recipe.Recipe.Ingredients[0]},
        &QueueItem{Element: recipe.Recipe.Ingredients[1]},
    )
}

```

```go
for len(currentQueue) > 0 {
    nextQueue := []*QueueItem{}
    for len(currentQueue) > 0 {
        current := currentQueue[0]
        currentQueue = currentQueue[1:]
```

Elemen akan dimasukkan ke dalam *queue* untuk diproses hingga tidak ada elemen yang berada di dalam *queue*. Untuk menyimpan struktur pohon dari elemen, disimpan menggunakan suatu *map* yang menyimpan TreeNode. Jika elemen belum diketahui seluruh resepnya, maka akan diproses juga di queue selanjutnya.

```go
allReady := true
for _, recipe := range currentNode.Parents {
    if len(recipe.SourceNodes) < 2 {
        continue
    }
    if !processedElements[recipe.Recipe.Ingredients[0]] {
        nextQueue = append(nextQueue, &QueueItem{Element: recipe.Recipe.Ingredients[0]})
        allReady = false
    }
    if !processedElements[recipe.Recipe.Ingredients[1]] {
        nextQueue = append(nextQueue, &QueueItem{Element: recipe.Recipe.Ingredients[1]})
        allReady = false
    }
}
```

Setelah selesai diproses, maka akan pindah ke level selanjutnya untuk diproses.

```go
currentQueue = nextQueue
```

Fungsi akan dijalankan melalui fungsi pembantu yaitu `startDFS()` dan `startBFS()`.

```go
func StartDFS(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := dfs(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}

func StartBFS(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := bfs(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}
```

Berikut adalah kode sumber untuk bagian teknis dari program ini.
scrapper.go

```go
package scraper

import (
    "encoding/json"
    "fmt"
    "log"
    "os"
    "strconv"
    "strings"

    "github.com/PuerkitoBio/goquery"
    "github.com/gocolly/colly/v2"
)

type Recipe struct {
    Ingredients []string `json:"ingredients"`
}

type Element struct {
    Name    string   `json:"name"`
    Tier    int      `json:"tier"`
    Recipes []Recipe `json:"recipes"`
}

func Scrape(filePath string) {
    c := colly.NewCollector(
        colly.UserAgent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"),
        colly.AllowedDomains("little-alchemy.fandom.com"),
        colly.AllowURLRevisit(),
    )

    var elements []Element
    sectionTiers := make(map[string]int)

    sectionCollector := c.Clone()
    sectionCollector.OnHTML("h3 .mw-headline", func(e *colly.HTMLElement) {
        sectionTitle := e.Text
        sectionID := e.Attr("id")

        var tier int
        switch sectionTitle {
        case "Starting elements", "Special element":
            tier = 0
        default:
            if sectionTitle != "" {
                rawSectionTitle := strings.Split(sectionTitle, " ")
                if len(rawSectionTitle) == 3 {
                    tierStr := rawSectionTitle[1]
                    if t, err := strconv.Atoi(tierStr); err == nil {
                        tier = t
                    } else {
                        fmt.Println("Error converting tier string to int:", err)
                        tier = 999
                    }
                } else {
                    fmt.Println("Unexpected section title format:", sectionTitle)
                    tier = 999
                }
            }
        }

        sectionTiers[sectionID] = tier
    })

    if err := sectionCollector.Visit("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"); err != nil {
        log.Fatal(err)
    }
```

```go
    c.OnHTML("h3", func(h3 *colly.HTMLElement) {
        sectionID := h3.ChildAttr(".mw-headline", "id")
        tier, exists := sectionTiers[sectionID]
        if !exists {
            tier = 999
        }

        h3.DOM.NextUntil("h3").Each(func(i int, s *goquery.Selection) {
            if s.Is("table.list-table") {
                s.Find("tr").Each(func(_ int, row *goquery.Selection) {
                    if row.Find("td:nth-of-type(1)").Text() == "" {
                        return
                    }

                    name := row.Find("td:nth-of-type(1)").Text()
                    var recipes []Recipe

                    recipeText := row.Find("td:nth-of-type(2)").Text()
                    if recipeText != "" {
                        rawRecipes := strings.Split(recipeText, "\n")
                        for _, recipe := range rawRecipes {
                            recipe = strings.TrimSpace(recipe)
                            if strings.Contains(recipe, "+") {
                                parts := strings.Split(recipe, "+")
                                var ingredients []string
                                for _, part := range parts {
                                    ingredient := strings.TrimSpace(part)
                                    if ingredient != "" {
                                        ingredients = append(ingredients, ingredient)
                                    }
                                }
                                if len(ingredients) >= 2 {
                                    recipes = append(recipes, Recipe{Ingredients: ingredients})
                                }
                            }
                        }
                    }

                    if name != "" {
                        elements = append(elements, Element{
                            Name:    strings.TrimSpace(name),
                            Tier:    tier,
                            Recipes: recipes,
                        })
                    }
                })
            }
        })
    })

    if err := c.Visit("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"); err != nil {
        log.Fatal(err)
    }

    jsonData, err := json.MarshalIndent(elements, "", "  ")
    if err != nil {
        log.Fatal("Error marshalling to JSON:", err)
    }

    if err := os.WriteFile(filePath, jsonData, 0644); err != nil {
        log.Fatal("Error writing JSON to file:", err)
    }

    fmt.Println("Data successfully saved to", filePath)
}
```

elements.go

```go
package elementsModel

import (
    "encoding/json"
    "errors"
    "os"
    "sync"
)

type ElementNode struct {
    Element  *Element
    Children []*ElementRelation
    Parents  []*ElementRelation
}

type ElementRelation struct {
    TargetNode  *ElementNode
    SourceNodes []*ElementNode
    Recipe      Recipe
}

type Recipe struct {
    Ingredients []string `json:"ingredients"`
}

type Element struct {
    Name    string   `json:"name"`
    Tier    int      `json:"tier"`
    Recipes []Recipe `json:"recipes"`
}

type ElementGraph struct {
    RootNode   *ElementNode
    AllNodes   map[string]*ElementNode
    Tier0Nodes []*ElementNode
}

type ElementsService struct {
    elements    []Element
    elementsMap map[string]*Element
    graph       *ElementGraph
    filePath    string
    initialized bool
    mutex       sync.RWMutex
}

var (
    instance *ElementsService
    once     sync.Once
)

func GetInstance() *ElementsService {
    once.Do(func() {
        instance = &ElementsService{
            elementsMap: make(map[string]*Element),
        }
    })
    return instance
}
```

```go
func (es *ElementsService) Initialize(filePath string) error {
    es.mutex.Lock()
    defer es.mutex.Unlock()

    if es.initialized {
        return nil
    }

    file, err := os.Open(filePath)
    if err != nil {
        return err
    }
    defer file.Close()

    data, err := os.ReadFile(filePath)
    if err != nil {
        return err
    }

    var elements []Element
    err = json.Unmarshal(data, &elements)
    if err != nil {
        return err
    }

    for i := range elements {
        es.elementsMap[elements[i].Name] = &elements[i]
    }

    es.elements = elements
    es.filePath = filePath

    es.buildElementGraph()

    es.initialized = true
    return nil
}

func (es *ElementsService) buildElementGraph() {
    graph := &ElementGraph{
        RootNode: &ElementNode{

```

```go
    for _, node := range graph.AllNodes {
        if node.Element.Recipes == nil {
            continue
        }

        for _, recipe := range node.Element.Recipes {
            relation := &ElementRelation{
                TargetNode:  node,
                SourceNodes: []*ElementNode{},
                Recipe:      recipe,
            }

            shouldAddParent := true
            for _, ingredientName := range recipe.Ingredients {
                if ingredientNode, exists := graph.AllNodes[ingredientName]; exists {
                    if ingredientNode.Element.Tier >= node.Element.Tier {
                        shouldAddParent = false
                    } else {
                        relation.SourceNodes = append(relation.SourceNodes, ingredientNode)
                        ingredientNode.Children = append(ingredientNode.Children, relation)
                    }
                }
            }

            if shouldAddParent {
                node.Parents = append(node.Parents, relation)
            }
        }
    }

    for _, tier0Node := range graph.Tier0Nodes {
        rootRelation := &ElementRelation{
            TargetNode:  tier0Node,
            SourceNodes: []*ElementNode{graph.RootNode},
            Recipe:      Recipe{Ingredients: []string{"BasicElement"}},
        }
        graph.RootNode.Children = append(graph.RootNode.Children, rootRelation)
    }

    es.graph = graph
}

func (es *ElementsService) GetElementGraph() *ElementGraph {
    es.mutex.RLock()
    defer es.mutex.RUnlock()
    return es.graph
}

func (es *ElementsService) GetAllElements() []Element {
    es.mutex.RLock()
    defer es.mutex.RUnlock()

    return es.elements
}

func (es *ElementsService) GetElementByName(name string) (*Element, error) {
    es.mutex.RLock()
    defer es.mutex.RUnlock()

    if element, exists := es.elementsMap[name]; exists {
        return element, nil
    }

    return nil, errors.New("element not found")
}

func (es *ElementsService) GetElementNode(name string) (*ElementNode, error) {
    es.mutex.RLock()
    defer es.mutex.RUnlock()

    if node, exists := es.graph.AllNodes[name]; exists {
        return node, nil
    }

    return nil, errors.New("element node not found")
```

ElementsController.go

```go
package elementsController

import (
    elementsModel "backend/models"
    "fmt"
    "sort"
    "strconv"
    "sync"
    "sync/atomic"
    "time"
)

type ElementController struct {
}

type TreeNode struct {
    Name   string
    Recipe []*TreeNode
}

var NodesVisited int64

func NewElementController(filePath string) (*ElementController, error) {
    err := elementsModel.GetInstance().Initialize(filePath)
    if err != nil {
        return nil, err
    }
    return &ElementController{}, nil
}

func (ec *ElementController) GetAllElementsTiers() (map[string][]string, error) {
    elements := elementsModel.GetInstance().GetAllElements()

    tierGroups := make(map[string][]string)

    for _, element := range elements {
        tierStr := strconv.Itoa(element.Tier)
        tierGroups[tierStr] = append(tierGroups[tierStr], element.Name)
    }

    for _, elements := range tierGroups {
        sort.Strings(elements)
    }

    return tierGroups, nil
}

func StartDFS(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := dfs(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}

func StartBFS(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := bfs(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}
```

```go
func StartDFSMulti(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := dfsMulti(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}

func StartBFSMulti(targetName string, n int, treeChan chan *TreeNode) (*TreeNode, int64, time.Duration) {
    atomic.StoreInt64(&NodesVisited, 0)
    start := time.Now()
    node, err := elementsModel.GetInstance().GetElementNode(targetName)
    if err != nil {
        return nil, NodesVisited, 0
    }

    trees := bfsMulti(node, int64(n), treeChan)
    return mergeTree(trees), NodesVisited, time.Since(start)
}

func dfs(target *elementsModel.ElementNode, n int64, treeChan chan *TreeNode) []*TreeNode {
    if target == nil {
        return nil
    }

    if len(target.Parents) <= 0 || target.Element.Tier == 0 {
        node := &TreeNode{
            Name: target.Element.Name,
        }
        return []*TreeNode{node}
    }

    var results []*TreeNode

    for _, recipe := range target.Parents {
        if len(recipe.SourceNodes) < 2 {
            continue
        }

        atomic.AddInt64(&NodesVisited, 1)
        leftTrees := dfs(recipe.SourceNodes[0], n, treeChan)
        rightTrees := dfs(recipe.SourceNodes[1], n, treeChan)

        for _, left := range leftTrees {
            for _, right := range rightTrees {
                node := &TreeNode{
                    Name:   target.Element.Name,
                    Recipe: []*TreeNode{left, right},
                }

                treeChan <- node

                results = append(results, node)
                if len(results) >= int(n) {
                    return results
                }
            }
            if len(results) >= int(n) {
                return results
            }
        }
        if len(results) >= int(n) {
            return results
        }
    }

    return results
}
```

```go
func dfsMulti(target *elementsModel.ElementNode, limit int64, treeChan chan *TreeNode) []*TreeNode {
    if target == nil {
        return nil
    }

    if len(target.Parents) <= 0 || target.Element.Tier == 0 {
        node := &TreeNode{
            Name: target.Element.Name,
        }
        return []*TreeNode{node}
    }

    var results []*TreeNode
    var resultsMutex sync.Mutex
    var wg sync.WaitGroup

    for _, recipe := range target.Parents {
        wg.Add(1)
        go func(recipe *elementsModel.ElementRelation) {
            defer wg.Done()
            if len(recipe.SourceNodes) < 2 {
                return
            }

            atomic.AddInt64(&NodesVisited, 1)
            leftTrees := dfsMulti(recipe.SourceNodes[0], limit, treeChan)
            rightTrees := dfsMulti(recipe.SourceNodes[1], limit, treeChan)

            localResults := []*TreeNode{}
            for _, left := range leftTrees {
                for _, right := range rightTrees {
                    node := &TreeNode{
                        Name:   target.Element.Name,
                        Recipe: []*TreeNode{left, right},
                    }

                    treeChan <- node
                    localResults = append(localResults, node)

                    resultsMutex.Lock()
                    if len(results) >= int(limit) {
                        resultsMutex.Unlock()
                        return
                    }
                    resultsMutex.Unlock()
                }
            }

            resultsMutex.Lock()
            results = append(results, localResults...)
            if len(results) > int(limit) {
                results = results[:int(limit)]
            }
            resultsMutex.Unlock()
        }(recipe)
    }

    wg.Wait()
    return results
}
```

```go
func bfs(target *elementsModel.ElementNode, n int64, treeChan chan *TreeNode) []*TreeNode {
    if target == nil {
        return nil
    }

    type QueueItem struct {
        Element string
    }

    if len(target.Parents) <= 0 || target.Element.Tier == 0 {
        node := &TreeNode{
            Name: target.Element.Name,
        }
        return []*TreeNode{node}
    }

    elementToTree := make(map[string][]*TreeNode)
    processedElements := make(map[string]bool)

    currentQueue := []*QueueItem{}

    for _, recipe := range target.Parents {
        currentQueue = append(currentQueue,
            &QueueItem{Element: recipe.Recipe.Ingredients[0]},
            &QueueItem{Element: recipe.Recipe.Ingredients[1]},
        )
    }

    var results []*TreeNode

    for len(currentQueue) > 0 {
        nextQueue := []*QueueItem{}
        for len(currentQueue) > 0 {
            current := currentQueue[0]
            currentQueue = currentQueue[1:]

            if processedElements[current.Element] {
                continue
            }

            currentNode, err := elementsModel.GetInstance().GetElementNode(current.Element)
            if err != nil || currentNode == nil {
                continue
            }

            atomic.AddInt64(&NodesVisited, 1)

            if currentNode.Element.Tier == 0 || len(currentNode.Parents) == 0 {
                tree := &TreeNode{
                    Name: currentNode.Element.Name,
                }

                treeChan <- tree

                elementToTree[currentNode.Element.Name] = []*TreeNode{tree}
                processedElements[currentNode.Element.Name] = true
            }

            allReady := true
            for _, recipe := range currentNode.Parents {
                if len(recipe.SourceNodes) < 2 {
                    continue
                }
                if !processedElements[recipe.Recipe.Ingredients[0]] {
                    nextQueue = append(nextQueue, &QueueItem{Element: recipe.Recipe.Ingredients[0]})
                    allReady = false

```

```go
            allReady := true
            for _, recipe := range currentNode.Parents {
                if len(recipe.SourceNodes) < 2 {
                    continue
                }
                if !processedElements[recipe.Recipe.Ingredients[0]] {
                    nextQueue = append(nextQueue, &QueueItem{Element: recipe.Recipe.Ingredients[0]})
                    allReady = false
                }
                if !processedElements[recipe.Recipe.Ingredients[1]] {
                    nextQueue = append(nextQueue, &QueueItem{Element: recipe.Recipe.Ingredients[1]})
                    allReady = false
                }
            }

            if !allReady {
                nextQueue = append(nextQueue, current)
                continue
            }

            var trees []*TreeNode
            for _, recipe := range currentNode.Parents {
                leftTrees := elementToTree[recipe.Recipe.Ingredients[0]]
                rightTrees := elementToTree[recipe.Recipe.Ingredients[1]]
                if leftTrees == nil || rightTrees == nil {
                    continue
                }
                for _, left := range leftTrees {
                    for _, right := range rightTrees {
                        node := &TreeNode{
                            Name:   currentNode.Element.Name,
                            Recipe: []*TreeNode{left, right},
                        }

                        treeChan <- node

                        trees = append(trees, node)
                    }
                }
            }
            if len(trees) > 0 {
                elementToTree[currentNode.Element.Name] = trees
                processedElements[currentNode.Element.Name] = true
            }
        }
        currentQueue = nextQueue
    }

    for _, recipe := range target.Parents {
        leftTrees := elementToTree[recipe.Recipe.Ingredients[0]]
        rightTrees := elementToTree[recipe.Recipe.Ingredients[1]]
        if leftTrees == nil || rightTrees == nil {
            continue
        }

        for _, left := range leftTrees {
            for _, right := range rightTrees {
                node := &TreeNode{
                    Name:   target.Element.Name,
                    Recipe: []*TreeNode{left, right},
                }
                results = append(results, node)
                if len(results) >= int(n) {
                    return results
                }
            }
        }
    }

    return results
}
```

```go
func bfsMulti(target *elementsModel.ElementNode, n int64, treeChan chan *TreeNode) []*TreeNode {
    if target == nil {
        return nil
    }

    type QueueItem struct {
        Element string
    }

    if len(target.Parents) <= 0 || target.Element.Tier == 0 {
        node := &TreeNode{
            Name: target.Element.Name,
        }
        return []*TreeNode{node}
    }

    elementToTree := make(map[string][]*TreeNode)
    var elementToTreeMutex sync.RWMutex

    processedElements := make(map[string]bool)
    var processedMutex sync.RWMutex

    currentQueue := []*QueueItem{}

    for _, recipe := range target.Parents {
        currentQueue = append(currentQueue,
            &QueueItem{Element: recipe.Recipe.Ingredients[0]},
            &QueueItem{Element: recipe.Recipe.Ingredients[1]},
        )
    }

    var results
```

```go
atomic.AddInt64(&NodesVisited, 1)

if currentNode.Element.Tier == 0 || len(currentNode.Parents) == 0 {
    tree := &TreeNode{
        Name: currentNode.Element.Name,
    }

    treeChan <- tree

    elementToTreeMutex.Lock()
    elementToTree[currentNode.Element.Name] = []*TreeNode{tree}
    elementToTreeMutex.Unlock()

    processedMutex.Lock()
    processedElements[currentNode.Element.Name] = true
    processedMutex.Unlock()
    return
}

allReady := true
nextItems := []*QueueItem{}

for _, recipe := range currentNode.Parents {
    if len(recipe.SourceNodes) < 2 {
        continue
    }
    processedMutex.RLock()
    leftProcessed := processedElements[recipe.Recipe.Ingredients[0]]
    rightProcessed := processedElements[recipe.Recipe.Ingredients[1]]
    processedMutex.RUnlock()

    if !leftProcessed {
        nextItems = append(nextItems, &QueueItem{Element: recipe.Recipe.Ingredients[0]})
        allReady = false
    }
    if !rightProcessed {
        nextItems = append(nextItems, &QueueItem{Element: recipe.Recipe.Ingredients[1]})
        allReady = false
    }
}
```

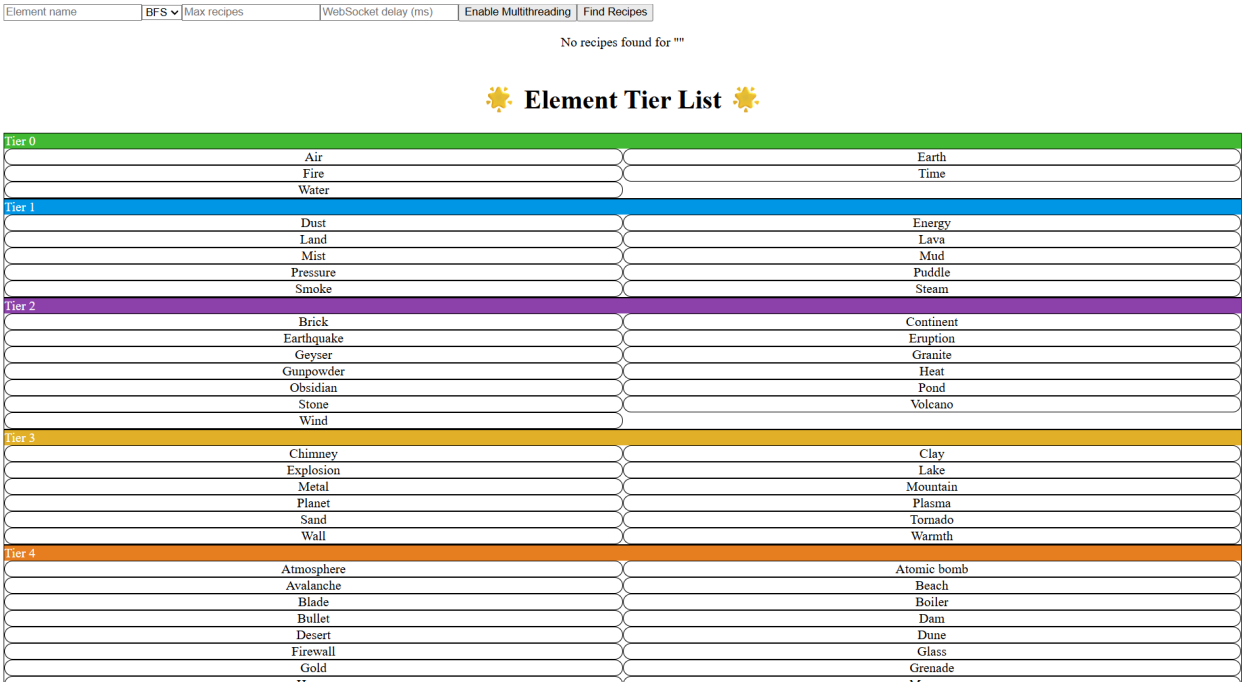
```go
            if !allReady {
                nextItems = append(nextItems, current)

                nextQueueMutex.Lock()
                nextQueue = append(nextQueue, nextItems...)
                nextQueueMutex.Unlock()
                return
            }

            var trees []*TreeNode
            for _, recipe := range currentNode.Parents {
                elementToTreeMutex.RLock()
                leftTrees := elementToTree[recipe.Recipe.Ingredients[0]]
                rightTrees := elementToTree[recipe.Recipe.Ingredients[1]]
                elementToTreeMutex.RUnlock()

                if leftTrees == nil || rightTrees == nil {
                    continue
                }

                for _, left := range leftTrees {
                    for _, right := range rightTrees {
                        node := &TreeNode{
                            Name:   currentNode.Element.Name,
                            Recipe: []*TreeNode{left, right},
                        }

                        treeChan <- node
                        trees = append(trees, node)
                    }
                }
            }

            if len(trees) > 0 {
                elementToTreeMutex.Lock()
                elementToTree[currentNode.Element.Name] = trees
                elementToTreeMutex.Unlock()

                processedMutex.Lock()
                processedElements[currentNode.Element.Name] = true
                processedMutex.Unlock()
            }
        }(current)
    }

    wg.Wait()
    currentQueue = nextQueue
}
```

```go
    var targetWg sync.WaitGroup
    for _, recipe := range target.Parents {
        targetWg.Add(1)
        go func(recipe *elementsModel.ElementRelation) {
            defer targetWg.Done()

            elementToTreeMutex.RLock()
            leftTrees := elementToTree[recipe.Recipe.Ingredients[0]]
            rightTrees := elementToTree[recipe.Recipe.Ingredients[1]]
            elementToTreeMutex.RUnlock()

            if leftTrees == nil || rightTrees == nil {
                return
            }

            localResults := []*TreeNode{}
            for _, left := range leftTrees {
                for _, right := range rightTrees {
                    node := &TreeNode{
                        Name:   target.Element.Name,
                        Recipe: []*TreeNode{left, right},
                    }
                    localResults = append(localResults, node)

                    resultsMutex.Lock()
                    if len(results) >= int(n) {
                        resultsMutex.Unlock()
                        return
                    }
                    resultsMutex.Unlock()
                }
            }

            resultsMutex.Lock()
            results = append(results, localResults...)
            if len(results) > int(n) {
                results = results[:int(n)]
            }
            resultsMutex.Unlock()
        }(recipe)
    }

    targetWg.Wait()
    return results
}
```

```go
func mergeTree(trees []*TreeNode) *TreeNode {
    root := &TreeNode{
        Name:   "Root",
        Recipe: trees,
    }
    return root
}

func PrintRecipeTree(tree *TreeNode, prefix string, isLast bool) {
    if tree == nil {
        return
    }

    connector := "├── "
    if isLast {
        connector = "└── "
    }

    fmt.Printf("%s%s%s", prefix, connector, tree.Name)

    if len(tree.Recipe) == 2 {
        fmt.Printf(" = %s + %s", tree.Recipe[0].Name, tree.Recipe[1].Name)
    }
    fmt.Println()

    newPrefix := prefix
    if isLast {
        newPrefix += "    "
    } else {
        newPrefix += "│   "
    }

    for i, child := range tree.Recipe {
        PrintRecipeTree(child, newPrefix, i == len(tree.Recipe)-1)
    }
}

func (ec *ElementController) GetElementByName(name string) (*elementsModel.Element, error) {
    element, err := elementsModel.GetInstance().GetElementByName(name)
    if err != nil {
        return nil, fmt.Errorf("element with name '%s' not found: %v", name, err)
    }
    return element, nil
}
```

## 4.2 Penjelasan Tata Cara Penggunaan Program

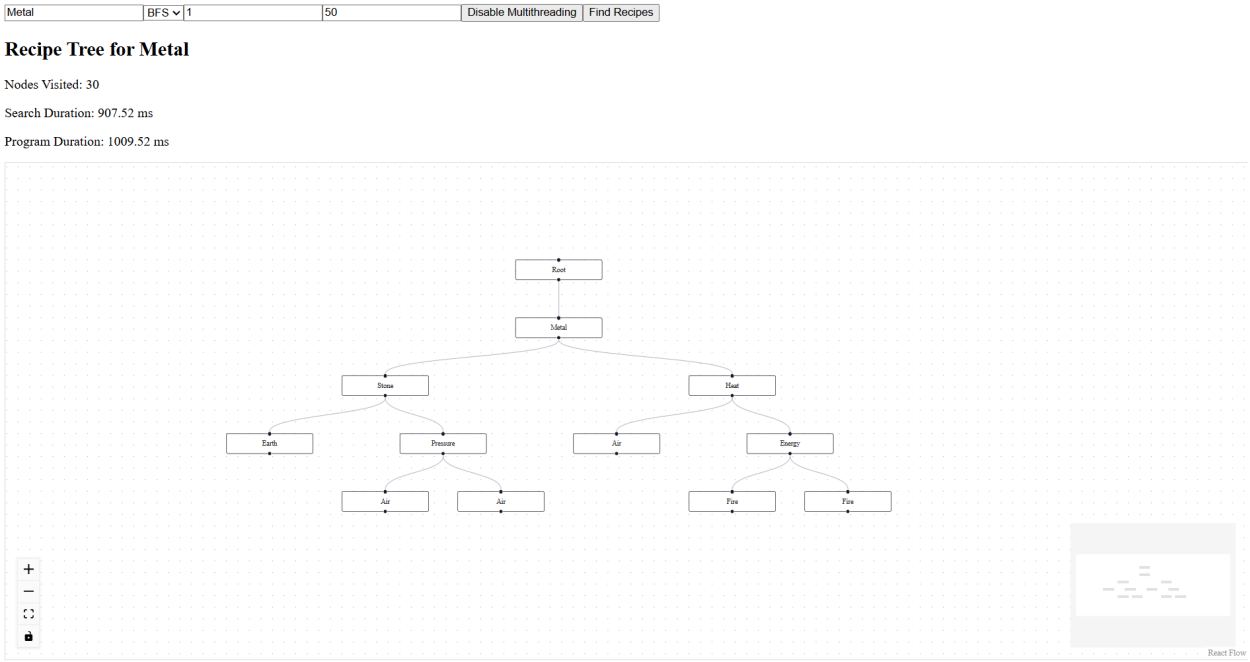
Tampilan halaman.




Input pada atas kiri halaman.

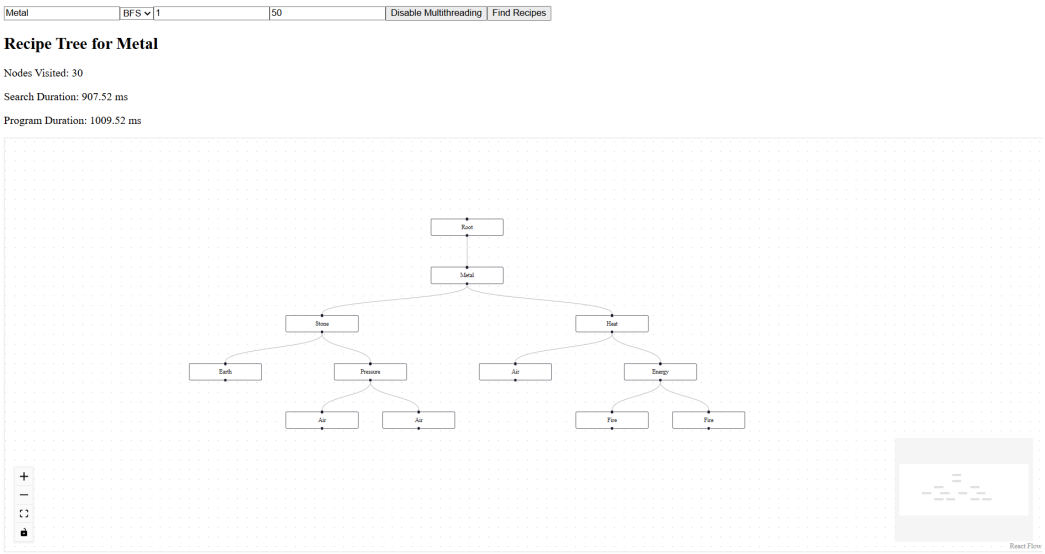
Setelah mengisi input, klik tombol “Find Recipes” untuk memulai mencari recipes. Fungsi ini dapat memilih antara pemilihan algoritma yang digunakan antara BFS dan DFS, memilih berapa resep yang dicari, delay animasi, dan apakah menggunakan multithreading.

Tampilan mencari elemen bernama “Metal” menggunakan BFS dan multithread.

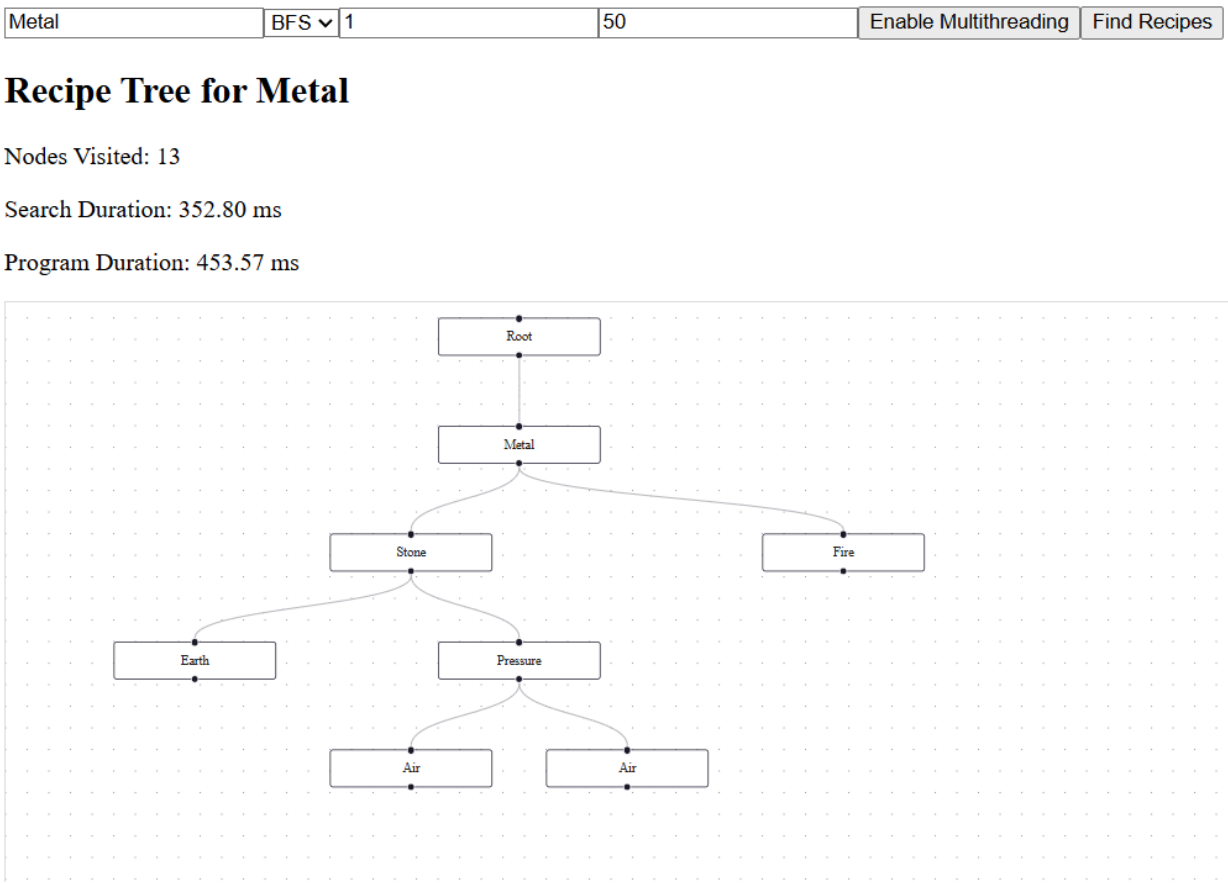



## 4.3 Hasil Pengujian

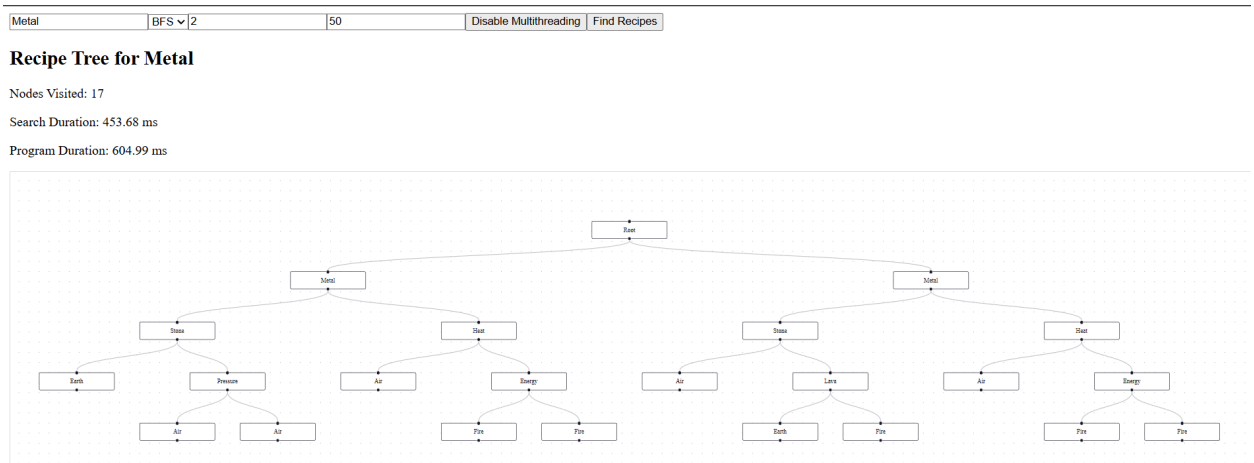
1. Metal, BFS, 1 recipe, multi thread.



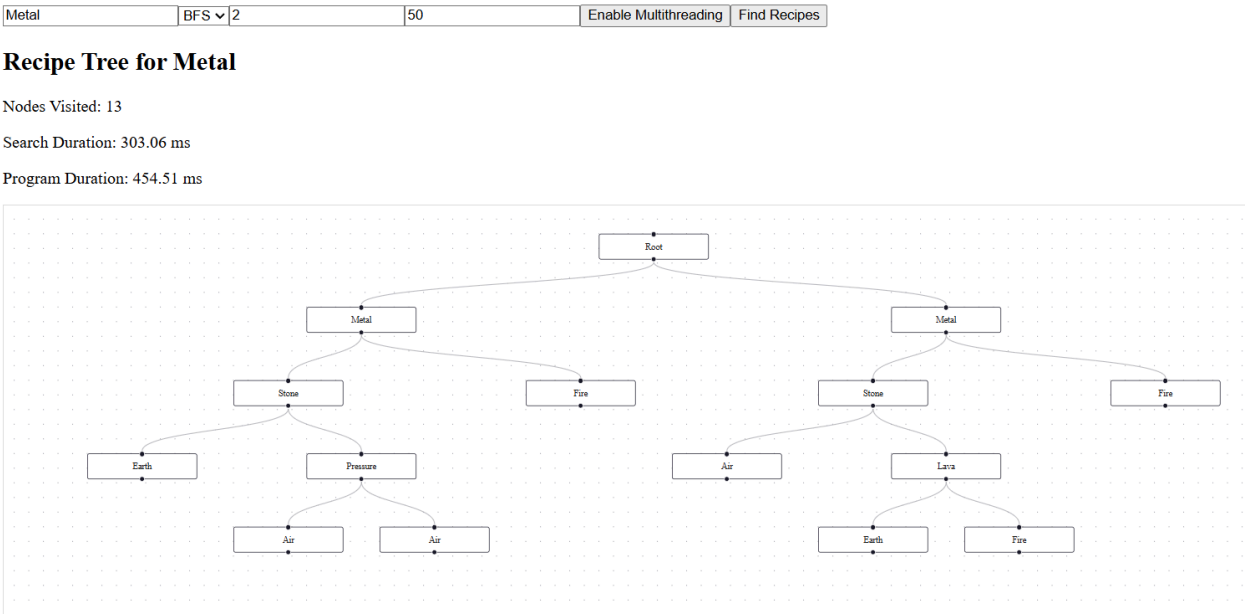

2. Metal, BFS, 1 recipe, non multi thread.




3. Metal, BFS, 2 recipes, multi thread.



4. Metal, BFS, 2 recipes, non multi thread.




5. Metal, DFS, 1 recipe, multi thread.

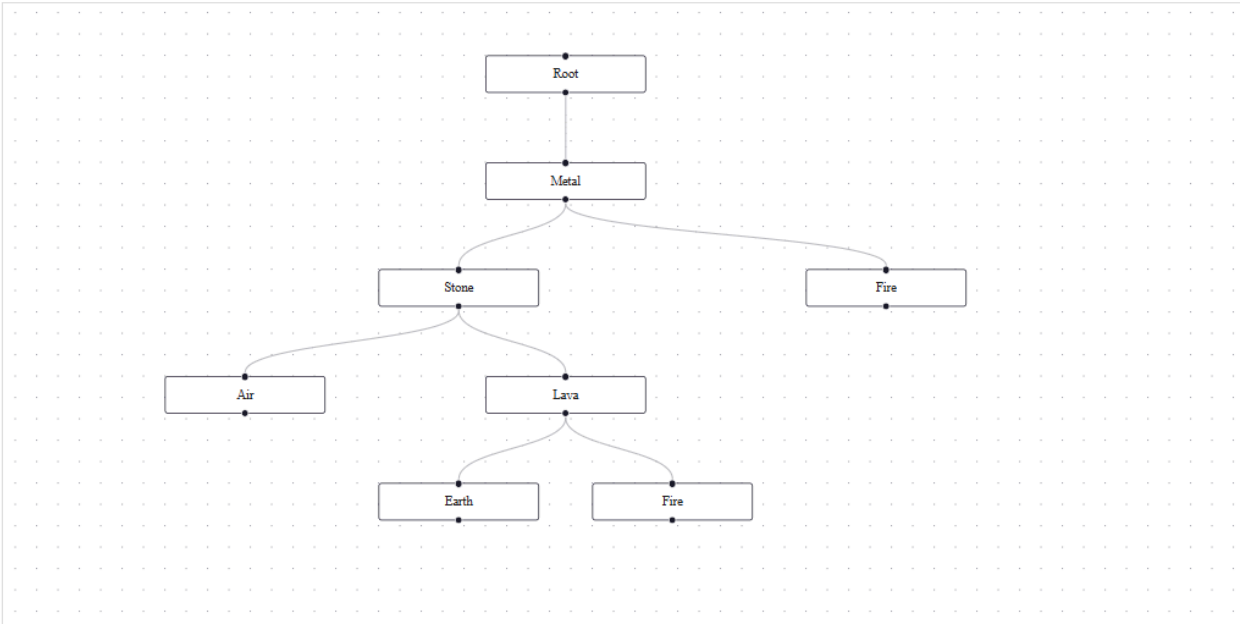
| Metal | DFS | 1 | 50 | Disable Multithreading | Find Recipes |
|---|---|---|---|---|---|

# Recipe Tree for Metal

Nodes Visited: 12

Search Duration: 504.09 ms

Program Duration: 604.83 ms




6. Metal, DFS, 2 recipes, multi thread.

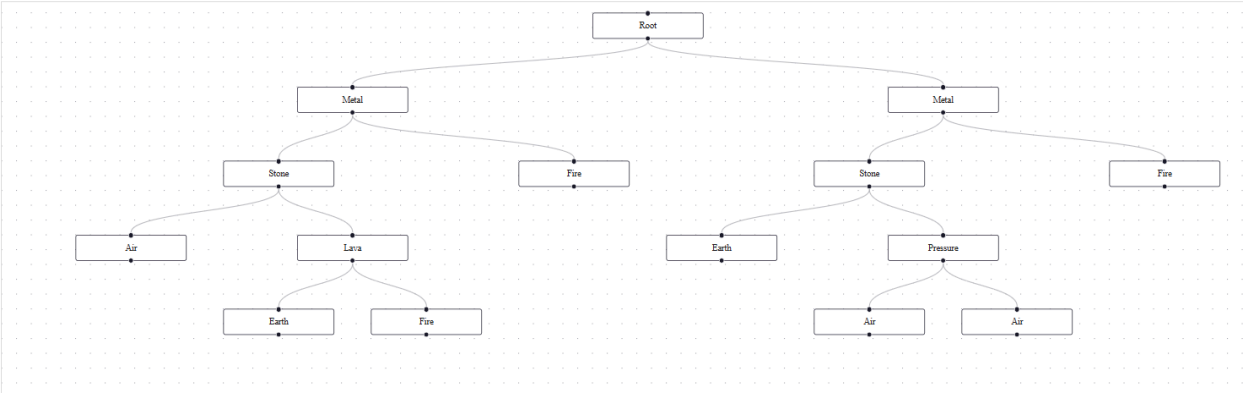
| Metal | DFS | 2 | 50 | Disable Multithreading | Find Recipes |
|---|---|---|---|---|---|

## Recipe Tree for Metal

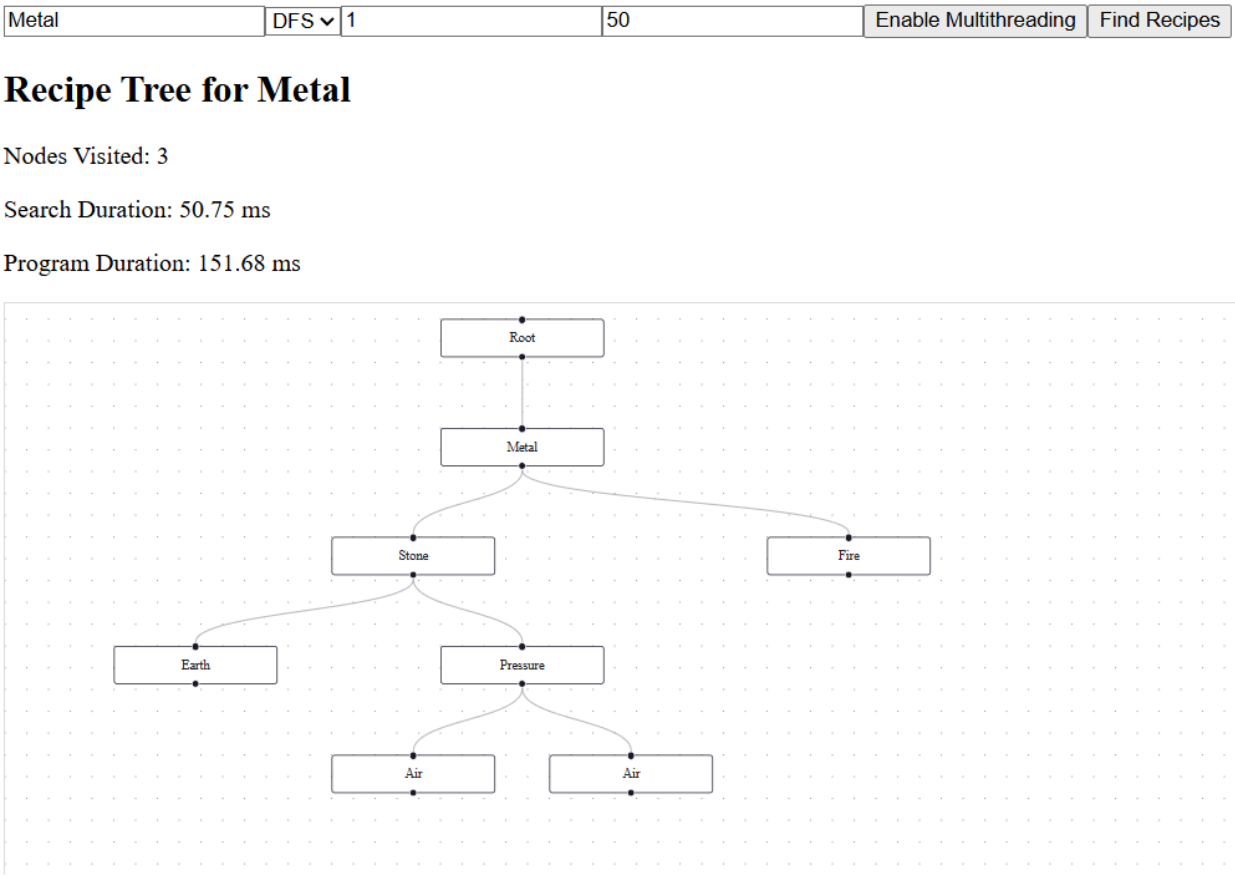
Nodes Visited: 12

Search Duration: 506.00 ms

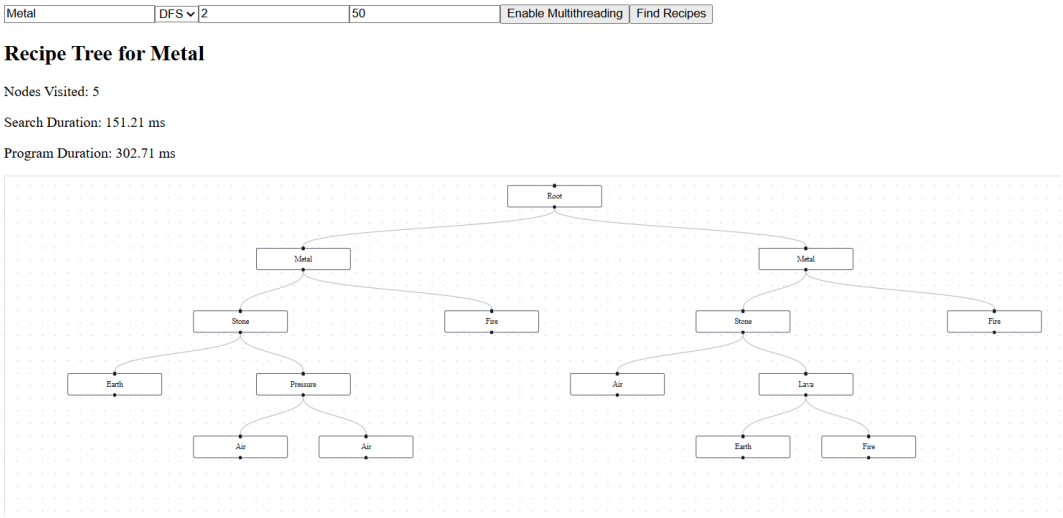
Program Duration: 657.26 ms



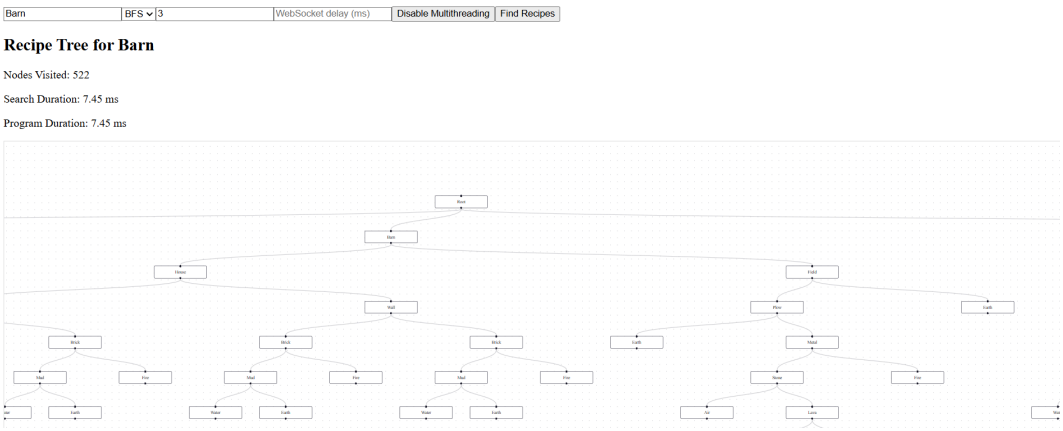
7. Metal, DFS, 1 recipe, non multi thread.



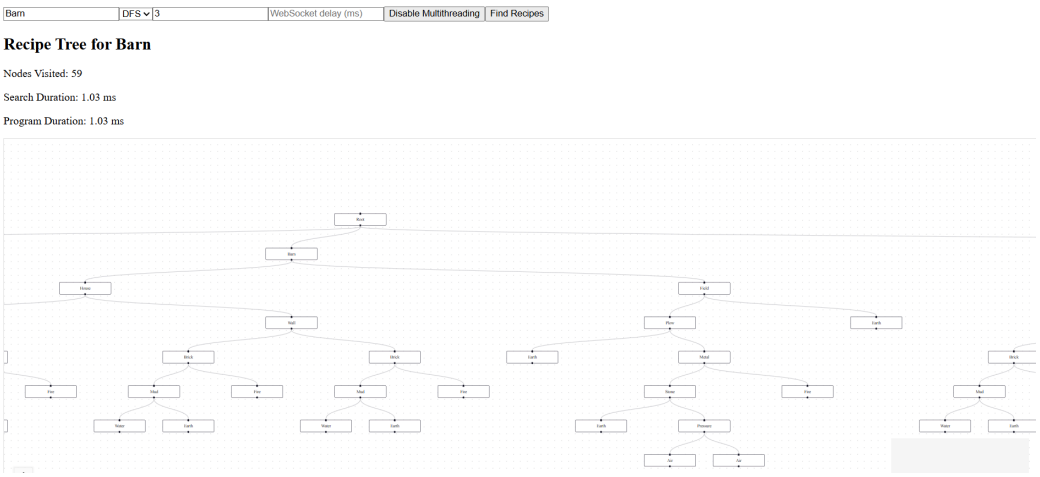

8. Metal, DFS, 2 recipes, non multi thread.



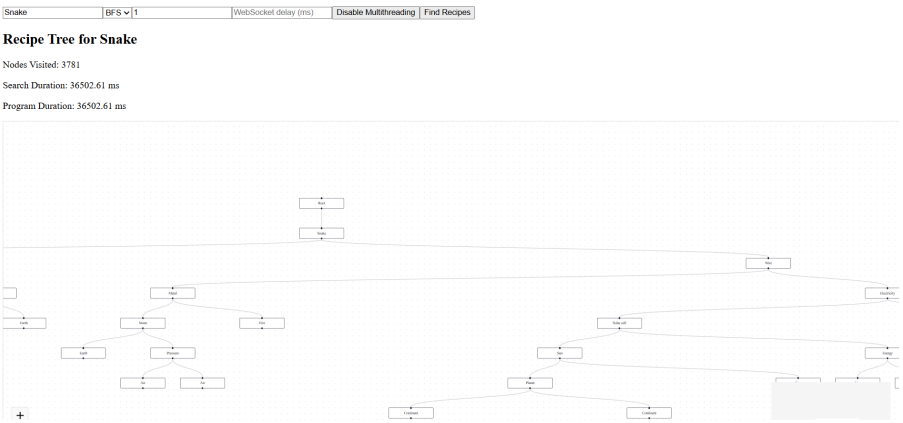

9. Barn, BFS, 3 recipes, multi thread.



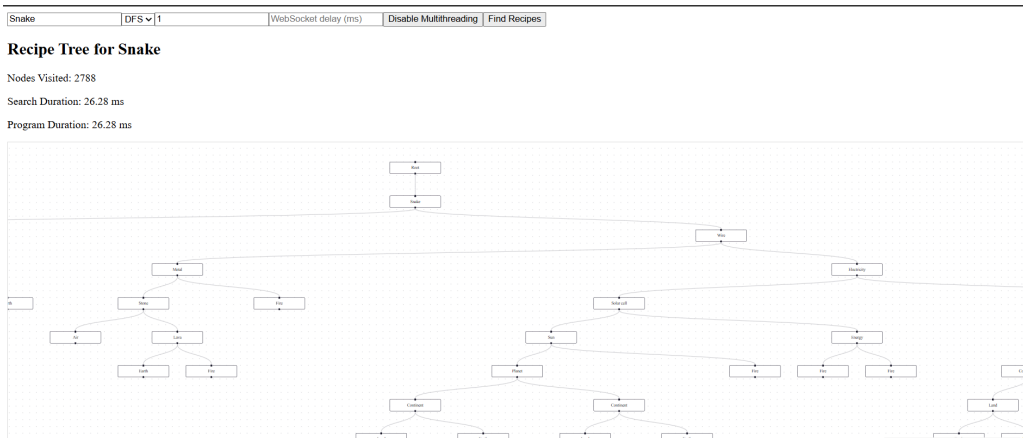

10. Barn, DFS, 3 recipes, multi thread.




11. Snake, BFS, 1 recipe, multi thread.



12. Snake, DFS, 1 recipe, multi thread.




## 4.4 Analisis Hasil Pengujian

Setelah dilakukan serangkaian pengujian terhadap algoritma pencarian Breadth-First Search (BFS) dan Depth-First Search (DFS), terdapat beberapa temuan menarik yang dapat disimpulkan. Pertama, dari segi waktu eksekusi, algoritma BFS cenderung membutuhkan waktu yang lebih lama dibandingkan dengan DFS. Hal ini kemungkinan besar disebabkan oleh sifat BFS yang harus mengeksplorasi setiap level node secara menyeluruh sebelum melanjutkan ke level berikutnya, sehingga jumlah node yang dikunjungi pun menjadi lebih banyak.

Kedua, jumlah node yang dikunjungi oleh BFS juga terbukti lebih banyak dibandingkan dengan DFS pada skenario pengujian yang dilakukan. DFS, dengan pendekatan eksplorasi mendalamnya, cenderung lebih cepat menemukan node target (tergantung pada posisi node tersebut dalam graf), sehingga traversalnya menjadi lebih efisien dalam beberapa kasus tertentu.

Ketiga, penerapan multithreading yang awalnya diharapkan dapat meningkatkan performa justru menunjukkan hasil yang sebaliknya. Penggunaan multithreading ternyata menambah overhead, baik dari sisi manajemen thread maupun sinkronisasi data antar thread. Akibatnya, waktu eksekusi secara keseluruhan justru menjadi lebih lambat dibandingkan dengan implementasi satu thread. Hal ini menunjukkan bahwa tidak semua algoritma pencarian cocok untuk dijalankan secara paralel, terutama jika struktur data yang digunakan tidak dirancang untuk akses konkuren secara efisien.

# BAB V
# KESIMPULAN, SARAN, DAN REFLEKSI

## 5.1 Kesimpulan

Melalui pengerjaan tugas besar ini, kami telah berhasil mengimplementasikan algoritma *breadth first search* (BFS) dan *depth first search* (DFS) dalam pembuatan aplikasi pencari recipe pada permainan Little Alchemy 2. Dalam tugas ini, kami menerapkan kedua algoritma tersebut untuk menelusuri kemungkinan kombinasi elemen dari elemen dasar hingga terbentuk elemen target yang diinginkan.

Aplikasi yang kami bangun dapat mencari multiple recipe menggunakan BFS dan DFS dengan batas jumlah yang dapat ditentukan pengguna, serta menampilkan visualisasi pohon kombinasi elemen secara jelas sehingga pengguna dapat memahami proses pembentukan elemen.

Dari hasil pengujian terhadap berbagai elemen, kami menemukan bahwa BFS sangat efektif dalam mencari solusi terpendek dengan waktu relatif cepat, sedangkan DFS lebih fleksibel dalam menelusuri semua kemungkinan kombinasi secara mendalam. Namun, performa keduanya sangat bergantung pada struktur graph recipe yang kompleks dan jumlah elemen yang sangat besar. Oleh karena itu, pemanfaatan multithreading pada pencarian multiple recipe sangat membantu dalam meningkatkan efisiensi pencarian.

## 5.2 Saran

Dalam pengerjaan tugas besar ini, terdapat beberapa saran yang akan kami perhatikan untuk dikembangkan kedepannya, antara lain:

1. Menstandarkan penamaan variabel dan fungsi agar lebih deskriptif dan seragam antara backend dan frontend, sehingga memudahkan kolaborasi dan meningkatkan keterbacaan kode.
2. Meningkatkan interaktivitas antarmuka dengan fitur pencarian yang menyarankan elemen secara otomatis berdasarkan input pengguna.
3. Mempelajari lebih lanjut tentang teknik web scraping yang lebih efisien dan tangguh agar pengambilan data dari sumber eksternal tetap stabil meskipun terjadi perubahan pada struktur halaman web.

## 5.3 Refleksi

Pengerjaan tugas besar ini memberikan banyak pelajaran bagi kami, khususnya terkait bagaimana memetakan masalah nyata ke dalam bentuk representasi graf yang dapat ditelusuri menggunakan algoritma pencarian.

Kami menyadari pentingnya kerja sama tim dalam menyelesaikan tugas kompleks seperti ini. Kami melakukan pembagian tugas yang jelas, komunikasi yang efektif, serta pemahaman yang sama sehingga dapat menyelesaikan tugas besar ini.

Melalui proses ini, kami tidak hanya mendapatkan pemahaman yang lebih dalam tentang BFS dan DFS, tetapi juga memperoleh pengalaman praktis dalam membangun aplikasi web dengan arsitektur client-server, serta pemanfaatan tools dan teknologi modern seperti React, Vite, Golang, dan multithreading.

# LAMPIRAN

## 1. Daftar Pustaka


Munir, Rinaldi. 2025. “Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1”
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf, diakses 9 Mei 2025)

Munir, Rinaldi. 2025. “Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 2”
(https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf, diakses 9 Mei 2025)

GeeksforGeeks. 2025. “Breadth First Search or BFS for a Graph”
(https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph, diakses 9 Mei 2025)

GeeksforGeeks. 2025. “Depth First Search or DFS for a Graph”
(https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph, diakses 9 Mei 2025)


## 2. Tautan Repository

Link repository dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok AVATAR adalah sebagai berikut:

https://github.com/d2v6/Tubes2_AVATAR

## 3. Tautan Video

Link video dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok AVATAR adalah sebagai berikut:

https://youtu.be/HVrtUm5QOzg

4. **Tabel Evaluasi**

| No | Poin | Ya | Tidak |
|---|---|---|---|
| 1 | Aplikasi dapat dijalankan. | ✓ | |
| 2 | Aplikasi dapat memperoleh data *recipe* melalui scraping. | ✓ | |
| 3 | Algoritma *Depth First Search* dan *Breadth First Search* dapat menemukan *recipe* elemen dengan benar. | ✓ | |
| 4 | Aplikasi dapat menampilkan visualisasi *recipe* 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 *Bidirectional*. | | ✓ |
| 9 | Membuat bonus *Live Update*. | ✓ | |
| 10 | Aplikasi di-*containerize* dengan Docker. | ✓ | |
| 11 | Aplikasi di-*deploy* dan dapat diakses melalui internet. | ✓ | |