

TUGAS BESAR 2
STRATEGI ALGORITMA IF2211
Penerapan Algoritma Breadth-First Search dan Depth-First
Search untuk Pencarian Resep pada Gim Little Alchemy 2



Kelompok 27
Tabel Periodik

Julius Arthur	13523030
David Bakti Lodianto	13523083
Nadhif Al Rozin	13523076

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

2025

DAFTAR ISI

DAFTAR ISI	1
BAB I: DESKRIPSI TUGAS	2
BAB II: LANDASAN TEORI	3
1. Landasan Teori	3
2. Desain Aplikasi Web	5
a. Frontend	5
b. Backend	5
BAB III: ANALISIS PEMECAHAN MASALAH	6
1. Deskripsi Persoalan	6
2. Pemetaan Masalah ke Elemen-Elemen Algoritma DFS & BFS	6
3. Fitur Fungsional dan Arsitektur Aplikasi Web	7
BAB IV: IMPLEMENTASI DAN PENGUJIAN	9
1. Spesifikasi Teknis Program	9
a. Struktur data	9
b. Fungsi dan Prosedur	10
i. Scraping	10
ii. BFS	14
iii. DFS	
iv. Bidirectional	17
2. Tata Cara Penggunaan	25
3. Hasil dan Analisis	26
BAB V: KESIMPULAN DAN SARAN	30
1. Kesimpulan	30
2. Saran	30
3. Refleksi	30
LAMPIRAN	31
DAFTAR REFERENSI	32

BAB I

DESKRIPSI TUGAS

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. Pada Tugas Besar pertama Strategi Algoritma ini, kami 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 1 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

1. Landasan Teori

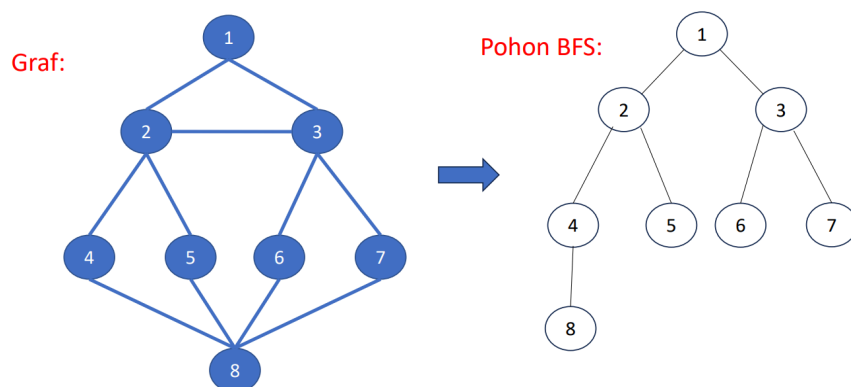
a. Spesifikasi Wajib

Algoritma penjelajahan graf (*graph traversal algorithm*) adalah algoritma yang digunakan untuk mengunjungi tiap-tiap simpul dari graf dengan metode yang sistematis. Hal ini berguna dalam pemecahan masalah-masalah graf atau permasalahan yang dapat direpresentasikan dalam graf, seperti pencarian jalur pada papan catur sebagai bidak kuda.

Berdasarkan informasi yang ada, algoritma penjelajahan graf dapat dibagi menjadi dua, yaitu *Uninformed/ Blind Search* dan *Informed Search*. Laporan ini hanya akan membahas *Uninformed Search*, karena *Informed Search* kurang relevan dalam penyelesaian masalah yang diinginkan. *Uninformed Search* secara umum memiliki 2 strategi, yaitu BFS dan DFS.

i. Breadth-First Search (BFS)

Breadth-First Search adalah algoritma penjelajahan graf dengan menjelajahi setiap simpul yang bertetangga dengan simpul awal terlebih dahulu sebelum melanjutkan menjelajah simpul dengan kedalaman selanjutnya yang terhubung dengan simpul yang sudah dikunjungi. Maka, setiap simpul pada tingkat kedalaman tertentu (relatif terhadap simpul awal) akan dikunjungi semua terlebih dahulu sebelum melanjutkan ke tingkat kedalaman yang lebih dalam. Karena penjelajahan ini mengunjungi setiap simpul pada suatu kedalaman, BFS selalu menghasilkan rute terpendek dari simpul awal ke simpul tujuan. Hasil penjelajahan ini dapat direpresentasikan ke dalam suatu *tree* (pohon) sesuai contoh gambar berikut, dengan nomor pada simpulnya menandakan urutan simpulnya dikunjungi.

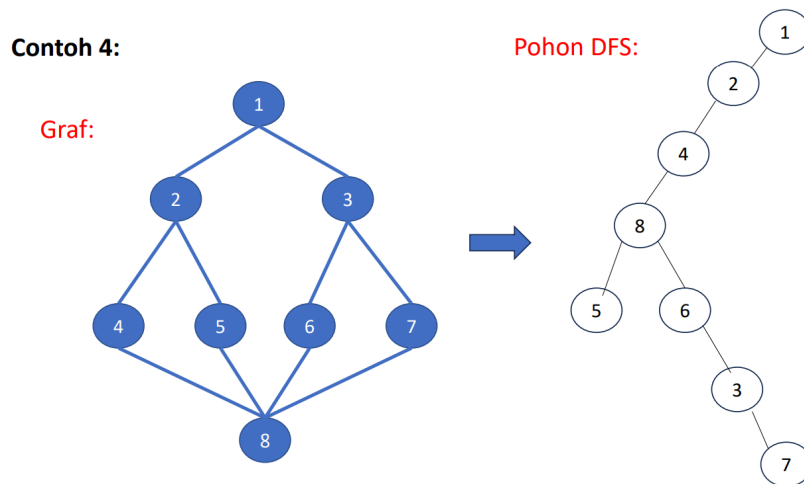


Gambar 2 Ilustrasi urutan eksekusi BFS

Dalam implementasinya, BFS menggunakan struktur data *Queue* untuk menyimpan simpul-simpul yang akan dijelajahi, karena pada dasarnya urutannya bersifat FIFO (*First-In First-Out*).

ii. Depth-First Search (DFS)

Depth-First Search adalah algoritma penjelajahan graf yang cukup mirip dengan BFS, namun lebih memprioritaskan kedalaman terlebih dahulu dibandingkan dengan menjelajahi tiap simpul pada satu tingkat kedalaman. Sehingga, algoritma ini akan menjelajah simpul tetangga pertama yang ditemui, hingga mencapai tujuan atau mencapai simpul buntu (tidak ada simpul lagi untuk dikunjungi). Ketika mencapai simpul buntu, DFS dapat kembali ke simpul sebelumnya dan melanjutkan penjelajahan ke simpul tetangga selanjutnya. Karena sifatnya yang tidak mencoba semua kemungkinan terlebih dahulu, DFS tidak pasti dalam menghasilkan rute terpendek, hanya memastikan bahwa rute tercapai ke tujuan akhir. Hasil penjelajahannya juga dapat direpresentasikan ke dalam pohon, seperti pada gambar berikut.



Gambar 3 Ilustrasi urutan eksekusi DFS

Berbeda dengan BFS dalam implementasinya, DFS seringkali menggunakan struktur data Stack untuk menyimpan simpul-simpul yang akan dikunjungi selanjutnya atau bisa juga diimplementasikan dalam pemanggilan fungsi yang rekursif, karena sifat urutan LIFO (*Last-In First-Out*).

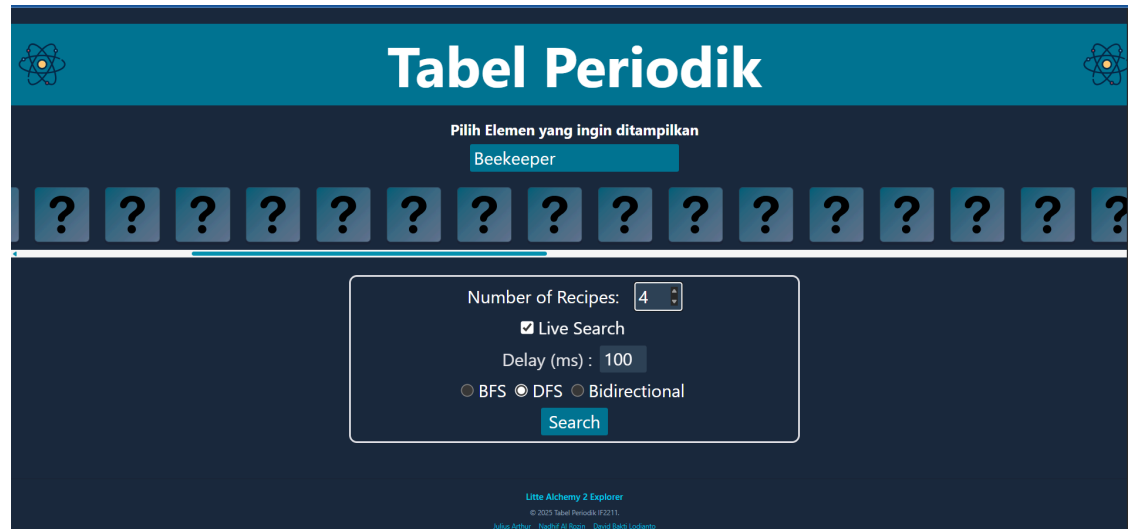
b. Spesifikasi Bonus

i. *Bidirectional Search*

Bidirectional Search adalah algoritma penjelajahan graf yang menggunakan dua algoritma penjelajahan graf sekaligus, dengan satu algoritma mulai dari simpul awal dan yang lain mulai dari ujung akhir dari graf. Sehingga, penjelajahan akan berakhir ketika kedua algoritma ini bertemu pada simpul yang sama.

2. Desain Aplikasi Web

a. Frontend



Kami menggunakan color palette Cyan dari tailwind sebagai warna utama dengan tambahan warna putih dan hitam.

Dari segi responsif dengan menggunakan **flex**, **rounded**, dan **hover effects** untuk pengalaman pengguna yang baik. Meskipun tidak mencakup keseluruhan ukuran, namun web masih dapat digunakan meski ukuran window di rubah rubah

Arsitektur code yang dibuat pada frontend menggunakan framework React. pengaturan modularitas digunakan dengan membuat component component yang ditulis pada folder yang sama.

State management dilakukan dengan `useEffect` dan `useState`. untuk mengatur variabel yang dirubah selama user menggunakan web seperti Live, Input, BFS/DFS/Bidirectional, Kondisi loading, Searching dll.

Untuk interaktivitas dengan pengguna, penggunaan image tree yang bisa didrag, di zoom dan list of element yang bisa di scroll dan ditekan. Selain itu penggunaan hover juga memberikan feedback langsung dan ekspektasi kepada pengguna apa yang akan terjadi saat menekan tombol

Selain itu penggunaan default di beberapa tempat untuk menangkap error yang mungkin terjadi selama user menggunakan web agar web tidak tampak rusak.

b. Backend






Backend website didesain untuk mengurus logika pencarian resep, dengan memiliki API yang dapat mengirim data hasil pencarian ke frontend. Namun sebelum itu, diperlukan melakukan scraping website wiki fandom Little Alchemy untuk mendapatkan semua *recipe* dari masing-masing *element*. Backend akan dibagi menjadi 5 API, yakni live-DFS, DFS, live-BFS, BFS, dan bidirectional.

BAB III

ANALISIS PEMECAHAN MASALAH

1. Deskripsi Persoalan

Pada permainan ini, satu elemen dapat dibentuk dari berbagai kombinasi dari 2 elemen. Persoalannya adalah mencari resep-resep dari salah satu elemen, dan membuat visualisasinya sebagai pohon. Sehingga, perlu dibuat program yang dapat mencari resep-resep dari elemen, menyimpan rutenya dan dari rute tersebut, digambarkan menjadi suatu pohon. Untuk permasalahan pertama dan kedua, hal ini sangat cocok untuk diselesaikan menggunakan algoritma BFS dan DFS. Namun, perlu dipetakan terlebih dahulu persoalan ini menjadi suatu graf agar dapat diimplementasikan dengan baik.

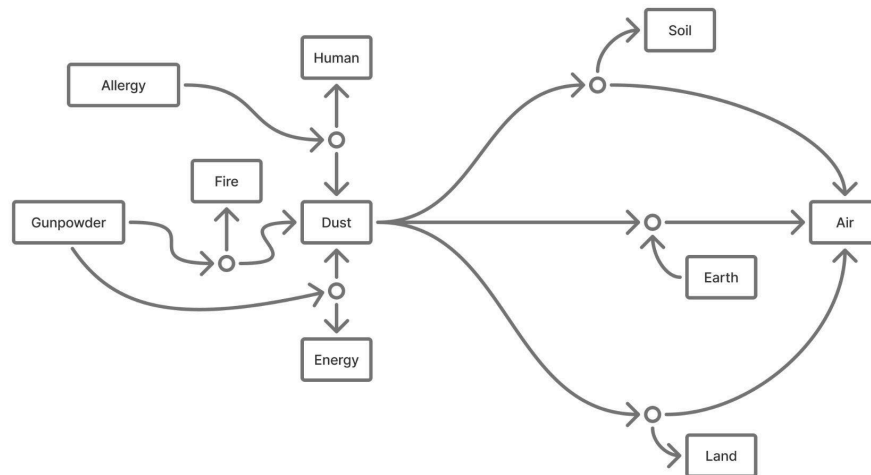
Element	Recipes
 Dust	<ul style="list-style-type: none"> Earth +  Air Land +  Air

Gambar 4 Contoh Resep Suatu Elemen di Gim Little Alchemy 2

2. Pemetaan Masalah ke Elemen-Element Algoritma DFS & BFS

Dari sistem permainannya, pemain dapat mendapatkan satu *element* dari menggabungkan dua *element*. Melihat dari daftar *recipe* dari tiap-tiap *element* yang disediakan pada laman *wiki fandom*-nya, hal ini sangat menyerupai *adjacency list* berarah dalam suatu graf. Maka, persoalan ini dengan mudah dapat dipetakan menjadi suatu graf, dengan setiap simpulnya merepresentasikan *element-element* yang ada. Selanjutnya, sisi dari kedua simpul akan merepresentasikan bahwa element dapat dibentuk dari element yang terhubung dengannya. Karena sifat dari sisinya yang tidak bolak-balik, maka sisinya bersifat berarah.

Akan tetapi, suatu resep elemen harus terdiri dari dua elemen. Jika *element* langsung dihubungkan satu sama lain, tidak bisa dibedakan gabungan dari 2 simpul *child* yang mana saja yang dapat menghasilkan simpul *parent*-nya. Maka, perlu dikelompokkan dari *recipe*-nya, sehingga *element* tidak langsung terhubung ke *element*. Hal ini bisa diimplementasikan dengan membuat 2 jenis simpul yang berbeda, yaitu simpul *element* dan simpul *recipe*. Simpul *element* akan terhubung dengan simpul-simpul *recipe*, dan simpul *recipe* akan terhubung hanya ke tiga simpul *element*, dua yang membentuk, dan satu yang dibentuk.



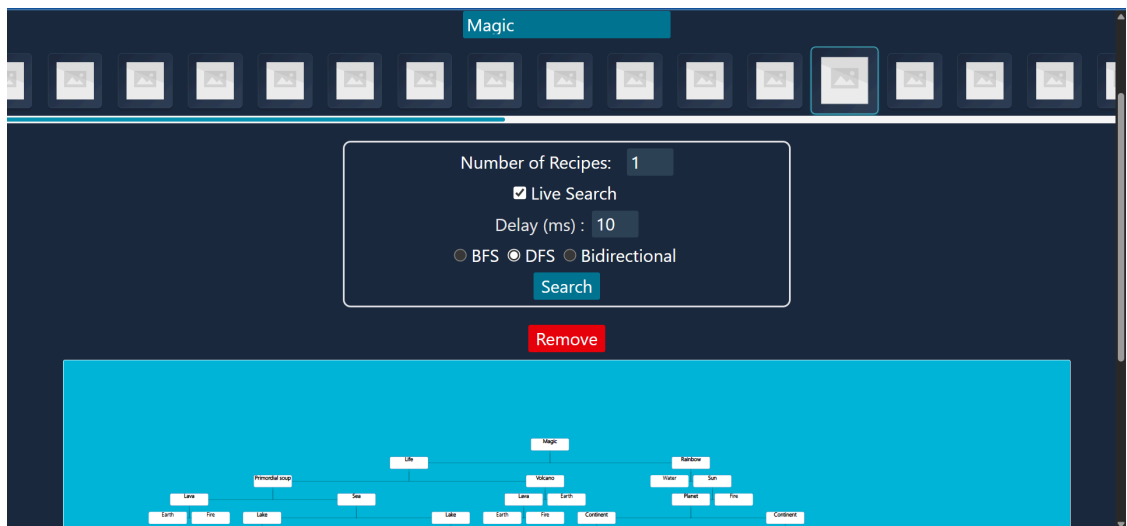
Gambar 5 Ilustrasi Graf Hasil Pemetaan untuk *Element* “Dust”

Pada ilustrasi di atas, simpul lingkaran menandakan simpul *recipe*, dan simpul persegi panjang adalah simpul *element*. Arah dari sisi menandakan bahwa simpul awal dibentuk dari simpul tujuan. Dengan model ini, algoritma penjelajahan graf sudah dapat digunakan.

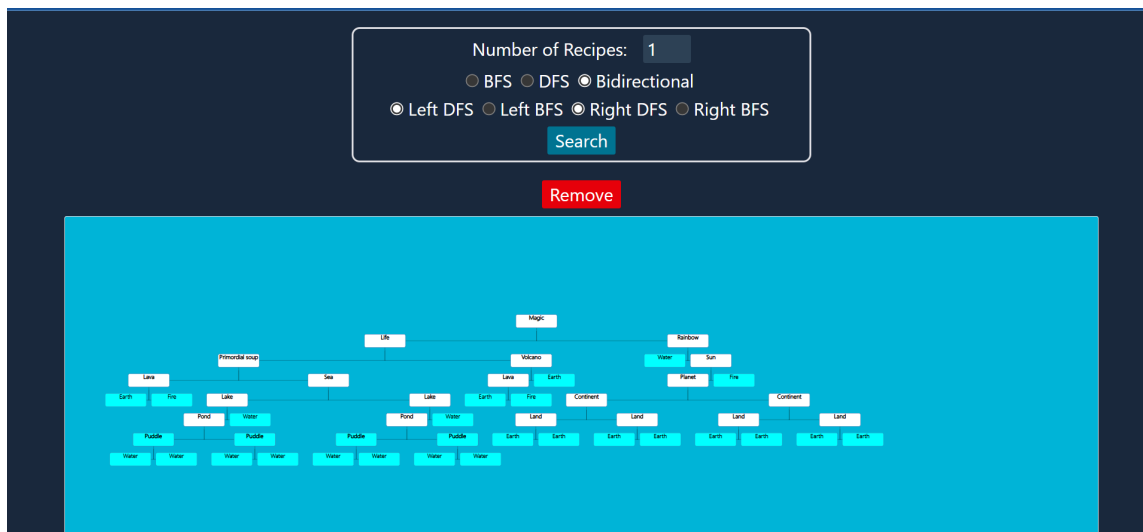
3. Fitur Fungsional dan Arsitektur Aplikasi Web

Fitur yang disediakan dalam web adalah mencari resep dari suatu elemen dengan jumlah yang kita masukkan. Pengguna dapat memilih untuk mencari resep secara langsung (dengan satu kali respons) atau dengan menggunakan live update dengan delay yang bisa diatur. Terdapat tiga metode pencarian yang dapat dipilih untuk mencari elemen, yakni BFS, DFS, serta bidirectional search. Hasil pencarian akan divisualisasikan dalam bentuk tree yang memiliki daun berupa elemen terendah (air, tanah, udara, dan api). Selain itu, untuk memperlihatkan fitur *live update*, website akan menggunakan SSE (Server-Side Events) untuk menjaga koneksi satu HTTP *request* agar dapat mengirim data terus-menerus.

Arsitektur web yang dipilih adalah dengan menggunakan front end dan back end dengan front end bertugas untuk berinteraksi dengan user, baik menerima input atau mengeluarkan output. Sedangkan, back end digunakan untuk semua proses yang dirasa tidak perlu ditunjukkan kepada user seperti scraping, hasil json data, proses debug, serta menjalankan algoritma pencarian resep yang dipilih. Berikut contoh penggunaan aplikasi kami.



Gambar DFS Live search dengan delay 10ms



Gambar Bidirectional Left DFS (Bawah, biru) dan Right DFS (Atas, putih)

BAB IV

IMPLEMENTASI DAN PENGUJIAN

1. Spesifikasi Teknis Program

a. Struktur data

Sebelum mengurus pencarian resep, aplikasi akan melakukan scraping website terlebih dahulu untuk mendapatkan *recipe* lengkap untuk setiap *element*. Hal ini dilakukan menggunakan struktur data Element sebagai berikut untuk diubah ke dalam format JSON.

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

Terdapat dua struktur data yang digunakan untuk membentuk graf persoalan ini, yaitu *element* dan *recipe*. Berikut adalah struktur data *element*.

```
type ElementNode struct {
    IsVisited bool
    Name      string
    ImgSrc    string
    Left      bool
    Tier      int
    Children  []*RecipeNode
}
```

ElementNode merepresentasikan sebuah elemen yang ada pada Little Alchemy 2. ElementNode menyimpan identitas elemen tersebut, seperti nama dan *tier*. ElementNode juga menyimpan sebuah logika boolean IsVisited. Nilai ini digunakan dalam algoritma pencarian yang menandakan bahwa sebuah elemen sudah dikunjungi. Selain itu, sebuah elemen dapat dibentuk dari satu atau lebih kombinasi elemen lainnya. Untuk membedakan kombinasi tersebut, terdapat struktur data *recipe*. Berikut adalah struktur data *recipe*.

```
type RecipeNode struct {
    Result      string
    Ingredient1 *ElementNode
    Ingredient2 *ElementNode
}
```

RecipeNode menyimpan tiga bagian penting, yaitu hasil kombinasi dari resep itu sendiri, serta dua elemen yang digunakan untuk membentuk resep tersebut. Karena sebuah elemen bisa saja dibuat dari lebih dari satu resep, maka ElementNode menyimpan sebuah larik yang terdiri dari pointer menuju RecipeNode pada Children.

Untuk menyimpan dalam JSON, terdapat dua struktur data yang mirip dengan kedua struktur data di atas.

```
type ExportableElement struct {
    Name      string          `json:"name"`
    ImgSrc     string          `json:"img_src"`
    Attributes map[string]string `json:"attributes"`
    Children   []ExportableRecipe `json:"children"`
}
```

```
type ExportableElement struct {
    Name      string          `json:"name"`
    ImgSrc     string          `json:"img_src"`
    Attributes map[string]string `json:"attributes"`
    Children   []ExportableRecipe `json:"children"`
}
```

Terdapat nilai tambahan yang disimpan pada kedua struktur data di atas. ImgSrc menunjukkan peta gambar yang tersimpan pada perangkat. Sedangkan, attributes sebuah merupakan *map* yang menunjukkan tipe node (resep atau elemen) serta sisi (kiri dan kanan).

- b. Fungsi dan Prosedur
 - i. Scraping

```
func scrape() []Element {
    c := colly.NewCollector()
    var recipeMap []Element
    var currentTier int

    c.OnHTML("tr, h3", func(e *colly.HTMLElement) {
        if e.Name == "h3" {
            // Save the current <h3> tier title
            tier :=
strings.TrimSpace(e.DOM.Find("span.mw-headline").Text())
            if len(tier) == 0 {
                return
            }
            if tier == "Starting elements" {
                fmt.Println("Starting elements")
                currentTier = 0
            } else {
```

```

                                currentTier = int(tier[5] - '0')
                                if tier[6] != ' ' {
                                    currentTier = currentTier*10 +
int(tier[6]-'0')
                                }
                                fmt.Println("Current tier:", currentTier)
                            }
                            return
                        }

                        tds := e.DOM.Find("td")
                        if tds.Length() < 2 {
                            return
                        }

                        var elmt Element
                        if tds.Eq(0).Find("a[title]").First().AttrOr("title", "")
== "Elements (Little Alchemy 1)" {
                            return
                        }

                        elmt.Name =
tds.Eq(0).Find("a[title]").First().AttrOr("title", "")
                        if elmt.Name == "Time" {
                            fmt.Println("Skipping Time element")
                            return
                        }

                        imgSrc, exists :=
tds.Eq(0).Find("img").First().Attr("data-src")
                        if !exists {
                            imgSrc =
tds.Eq(0).Find("img").First().AttrOr("src", "")
                        }
                        elmt.ImgSrc = imgSrc

                        // fmt.Printf("Element: %s, ImgSrc: %s\n", elmt.Name,
imgSrc)

                        elmt.Tier = currentTier
                        tds.Eq(1).Find("li").Each(func(i int, li
*goquery.Selection) {
                            var ingredients [2]string
                            li.Find("a[title]").Each(func(j int, a
*goquery.Selection) {
                                if a.AttrOr("title", "") == "Time" {
                                    fmt.Println("Skipping Time recipe for
element:", elmt.Name)
                                    return
                                }
                                ingredients[j] = a.AttrOr("title", "")
                            })
                            if ingredients[0] != "" && ingredients[1] != "" {
                                elmt.Recipes = append(elmt.Recipes,
ingredients)
                            }
                        })
                        recipeMap = append(recipeMap, elmt)
                    })
                })

```

```

        c.OnScraped(func(r *colly.Response) {
        })

        c.Visit("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alche
        my_2)")

        return recipeMap
    }

    func convertToJson(recipes []Element) ([]byte, error) {
        jsonBytes, err := json.MarshalIndent(recipes, "", " ")
        if err != nil {
            fmt.Println("Error marshalling to JSON:", err)
            return nil, err
        }
        if jsonBytes == nil {
            fmt.Println("jsonBytes is nil")
            return nil, fmt.Errorf("jsonBytes is nil")
        }
        return jsonBytes, nil
    }
}

```

Scraping dilakukan dengan mengambil setiap elemen HTML dengan atribut `tr` atau `h3`. Atribut `tr` digunakan untuk mengambil setiap baris pada suatu tabel, dengan kolom pertama mengandung *element*, dan kolom kedua mengandung daftar *recipe*-nya. Dengan pola itu, didapatkan *recipe* dari setiap *element*. Atribut `h3` digunakan untuk mengambil nilai Tier dari tiap-tiap *element*.

ii. BFS

```

func bfs(root *ElementNode, elements map[string]*ElementNode, recipes
[]*RecipeNode, limitRecipe int, ch chan int) {
    visited := make(map[string]bool)
    var mu sync.Mutex
    var ru sync.Mutex
    var recipeMu sync.Mutex

    mu.Lock()
    visited[root.Name] = true
    root.IsVisited = true
    mu.Unlock()

    recipeCount := make(map[string]int)
    currentLevel := []*ElementNode{root}
    count := 0

    for len(currentLevel) > 0 {
        var wg sync.WaitGroup
        var nextLevel []*ElementNode

        if ch != nil {
            fmt.Println("Level: ", currentLevel[0].Tier)
            ch <- currentLevel[0].Tier
        }

        for _, current := range currentLevel {

```

```

current.Children = []*RecipeNode{}
wg.Add(1)

go func(current *ElementNode) {
    defer wg.Done()

    for _, recipe := range recipes {
        if recipe.Result != current.Name ||
recipe.Ingredient1 == nil || recipe.Ingredient2 == nil {
            continue
        }

        base1, ok1 :=
elements[recipe.Ingredient1.Name]
        base2, ok2 :=
elements[recipe.Ingredient2.Name]
        if !ok1 || !ok2 || base1.Tier >=
current.Tier || base2.Tier >= current.Tier {
            continue
        }

        mu.Lock()
        ru.Lock()

        exist := false
        for _, c := range current.Children {
            if base1.Name ==
c.Ingredient1.Name && base2.Name == c.Ingredient2.Name {
                exist = true
                break
            }
        }

        if exist {
            ru.Unlock()
            mu.Unlock()
            continue
        }

        if current == root {
            if count >= limitRecipe {
                ru.Unlock()
                mu.Unlock()
                return
            }
            count++
            current.Children =
append(current.Children, &RecipeNode{
                Result:
current.Name,
                Ingredient1:
elements[base1.Name],
                Ingredient2:
elements[base2.Name],
            })
            recipeMu.Lock()
            recipeCount[current.Name] += 1
            recipeMu.Unlock()
        } else {
            res := RecipeNode{

```

```

current.Name,
elements[base1.Name],
elements[base2.Name],

Result:
Ingredient1:
Ingredient2:
}

recipeMu.Lock()
recipeCount[current.Name] += 1

current_count := 1
for _, v := range recipeCount

    current_count *= v
}

if current_count > limitRecipe

{
recipeCount[current.Name] -= 1

    recipeMu.Unlock()
    ru.Unlock()
    mu.Unlock()
    break
}
recipeMu.Unlock()

current.Children =

append(current.Children, &res)
fmt.Printf("Appending recipe
for %s, %s + %s\n", current.Name, base1.Name, base2.Name)

current.Tier)

if ch != nil {
    fmt.Println("Level: ",

    ch <- current.Tier
}

}

ru.Unlock()
mu.Unlock()

mu.Lock()
if !base1.IsVisited {
    nextLevel = append(nextLevel,

    visited[base1.Name] = true
    base1.IsVisited = true
}
if !base2.IsVisited {
    nextLevel = append(nextLevel,

    visited[base2.Name] = true
    base2.IsVisited = true
}
mu.Unlock()

}
}(current)
}

```

```

        wg.Wait()
        fmt.Println("Level: ", currentLevel[0].Tier)
        currentLevel = nextLevel
    }

    if ch != nil {
        fmt.Println("DONE")
        close(ch)
    }
}

```

Fungsi BFS adalah metode pencarian *Breadth First Search* pada sebuah graf resep. Root/titik mulai dari graf ini adalah elemen yang ingin dicari. Pencarian *multiple recipe* dibatasi dengan `limitRecipe`. Terdapat pula channel untuk mendukung konkurensi dan menampilkan *live update* pada frontend.

Fungsi memproses semua anak dari semua node yang ada pada suatu level, layaknya algoritma BFS. Iterasi pada suatu level ditunjukkan oleh *for* `currentLevel`. Setiap node yang ada pada `currentLevel` akan diproses secara konkuren melalui *goroutine* yang menerima elemen tersebut. Perhatikan bahwa terdapat waiting group yang di inisialisasi untuk setiap larik `currentLevel`. Waiting group ini bertambah setiap kali *goroutine* dijalankan. Setiap anak pada node di `currentLevel` (element yang merupakan bagian dari resep untuk membuat elemen pada `currentLevel`) akan diproses untuk ditambahkan ke queue. Fungsi mengecek setiap elemen pembentuk pada suatu resep. Resep dianggap valid, jika kedua elemen pembentuk/ *ingredient* merupakan elemen dengan *tier* yang lebih rendah dari pada elemen root, hingga dicapai elemen dengan *tier* 0. Selain itu, fungsi juga membatasi pencarian pada total kombinasi resep yang boleh dicari. Jika, batas sudah tercapai, maka fungsi tidak akan menambahkan kombinasi baru untuk membuat sebuah elemen. Untuk mengurangi kompleksitas waktu, fungsi menandai setiap node yang sudah pernah dilewati. Jika terdapat node yang sudah pernah dilewati sebelumnya, node tidak akan ditambahkan kembali ke larik. Jika seluruh *goroutine* sudah selesai (waiting group = 0), maka fungsi akan mengirim data ke channel untuk divisualisasikan.

iii. DFS

```

func DFS_Multiple(
    current *ElementNode,
    wg *sync.WaitGroup,
    elements map[string]*ElementNode,

```



```

depthChan chan int,
barrier *sync.WaitGroup,
) {
    defer func() {
        if depthChan != nil {
            fmt.Printf("DFS_Multiple: %s\n", current.Name)
            depthChan <- current.Tier
        }
    }()
    visitMu.Lock()
    if current.IsVisited {
        visitMu.Unlock()
        barrier.Done()
        return
    }
    current.IsVisited = true
    visitMu.Unlock()

    if current.Tier == 0 {
        barrier.Done()
        return
    }

    count := atomic.AddInt32(&numberVisit, 1)
    fmt.Printf("Visiting node Multi (%d): %s Tier: %d\n", count, current.Name, current.Tier,
    atomic.LoadInt32(&recipeLeft))

    // fmt.Printf("Recipe len: %d", len(current.Children))
    ALLrecipes := make([]*RecipeNode, len(current.Children))
    copy(ALLrecipes, current.Children)
    current.Children = []*RecipeNode{}

    // fmt.Printf("Recipe len: %d", len(ALLrecipes))
    barrier.Done()
    barrier.Wait()

    if depthChan != nil {
        fmt.Printf("DFS_Multiple: %s\n", current.Name)
        depthChan <- current.Tier
    }
}

```

```

fistAdd := true
for _, recipe := range ALLrecipes {
    if recipe.Result != current.Name {
        continue
    }

    // fmt.Printf("Processing recipe for %s\n",
current.Name)

    ing1 := recipe.Ingredient1
    ing2 := recipe.Ingredient2
    if ing1 == nil || ing2 == nil {
        continue
    }

    if ing1.Tier >= current.Tier || ing2.Tier >=
current.Tier {
        continue
    }

    if !fistAdd {
        if atomic.LoadInt32(&recipeLeft) <= 0 {
            continue
        }
        atomic.AddInt32(&recipeLeft, -1)
    }
    fistAdd = false

    visitMu.Lock()
    current.Children = append(current.Children, recipe)
    fmt.Printf("Appending recipe Multi for %s, %s +
%s\n", current.Name, ing1.Name, ing2.Name)
    visitMu.Unlock()

    visitMu.Lock()
    if ing1.Tier == 0 {
        ing1.IsVisited = true
    }
    if ing2.Tier == 0 {
        ing2.IsVisited = true
    }
    visitMu.Unlock()

```

```

        if ing1.Tier == 0 && ing2.Tier == 0 {
            continue
        }
        barrier.Add(1)
        select {
        case sem <- struct{}{}:
            wg.Add(1)
            go func(n *ElementNode) {
                defer wg.Done()

                DFS_Multiple(n, wg, elements, depthChan,
barrier)

                <-sem // release slot
            }(ing1)
        default:
            DFS_Multiple(ing1, wg, elements, depthChan,
barrier)
        }
        barrier.Add(1)
        select {
        case sem <- struct{}{}:
            wg.Add(1)
            go func(n *ElementNode) {
                defer wg.Done()
                DFS_Multiple(n, wg, elements, depthChan,
barrier)

                <-sem // release slot
            }(ing2)
        default:
            DFS_Multiple(ing2, wg, elements, depthChan,
barrier)
        }
    }

    // if len(current.Children) == 0 {
    //     DFS_Single(current, wg, elements, depthChan)
    // }
    visitMu.Lock()
    current.IsVisited = true
    visitMu.Unlock()
}

```

Fungsi DFS dimulai dari root yang akan mencari recipe yang dimilikinya apakah ingredient yang dipilih sesuai (Tier tidak lebih tinggi atau sama dengan tier root) jika benar akan diproses, jika termasuk tier 0 akan langsung di labeli visit dan di skip. Jika bukan 0 akan membuat go routine dari channel yang tersedia. jika tidak ada channel lagi yang tersedia akan masuk ke fungsi biasa dan ingredient ke 2 nya akan menunggu hingga ingredient pertama selesai. Pada gambar proses pencarian akan selalu ke kiri terlebih dahulu, jika dalam live update, proses akan selalu ke kiri jika tidak ada channel lain yang tersisa, jika ada kedua branch akan berjalan. Untuk mempermudah penghitungan total recipe yang dibuat digunakan DFS single yang berguna untuk menangkap semua element dari DFS_Multiple jika tidak ada lagi recipe yang tersedia dan hanya berfokus untuk memetakan element itu sampai tier 0 (dead end). DFS_Multiple akan mengukur jumlah recipe yang ada dengan memberikan recipe 1 dari suatu element pengecualian dan mengurangi 1 setiap ada recipe baru yang dijabarkan. Pada pencarian penandaan node yang sudah pernah dikunjungi dilakukan dengan isVisited hal ini agar menghindari redundancy dan memberikan konsistensi pada hasil.

iv. Bidirectional

```
func Bidirect_Right_DFS(
    root *ElementNode,
    wg *sync.WaitGroup,
    elementMap map[string]*ElementNode,
    depthChan chan int,
    doneChan chan struct{},
) {
    wg.Add(1)
    go func() {
        defer wg.Done()
        DFS_Multiple(root, wg, elementMap, depthChan)
        fmt.Println("[DFS Right] Done")
        close(doneChan) // Notify BFS
    }()
}

func Bidirect_Right_BFS(
    root *ElementNode,
    limitRecipe int,
    wg *sync.WaitGroup,
    elementMap map[string]*ElementNode,
```

```

    allRecipes []*RecipeNode,
    depthChan chan int,
    doneChan chan struct{},
) {
    go func() {
        bfs(root, elementMap, allRecipes, limitRecipe,
depthChan)
        fmt.Println("[BFS Right] Done")
        close(doneChan) // Notify BFS
    }()
}

```

Sisi Right dari bidirect adalah sisi yang berasal dari element tujuan ke element dasar. Oleh karena itu pengimplementasiannya menggunakan fungsi yang sudah dibuat sebelumnya dengan pemberian done Channel untuk memberitahu left side jika right side sudah selesai. Karena kondisi bidirect selesai atau tidaknya dipengaruhi penuh oleh right side.

```

func Bidirect_Left_BFS(
    basic []*ElementNode,
    target *ElementNode,
    allElement map[string]*ElementNode,
    allRecipes []*RecipeNode,
    doneChan <-chan struct{},
) {
    fmt.Println("[BFS] Bidirect_Left_BFS started")

    discovered := make(map[string]*ElementNode)
    tierElements := make(map[int][]*ElementNode)
    for _, el := range basic {
        discovered[el.Name] = el
        el.IsVisited = true
        el.Left = true
        fmt.Printf("[BFS] Added basic element: %s (tier
%d)\n", el.Name, el.Tier)
        tierElements[el.Tier] =
append(tierElements[el.Tier], el)
    }
}

```

```

ingredient := make(chan *ElementNode, 100)
var wg sync.WaitGroup
var mu sync.Mutex

// Worker
worker := func(id int, recipes []*RecipeNode) {
    defer wg.Done()
    fmt.Printf("[Worker %d] Started with %d recipes\n",
id, len(recipes))
    for _, recipe := range recipes {
        select {
        case <-doneChan:
            fmt.Printf("[Worker %d] Received doneChan,
exiting early\n", id)
            return
        default:
        }
        mu.Lock()
        in1 := recipe.Ingredient1
        in2 := recipe.Ingredient2
        result := allElement[recipe.Result]

        if result.IsVisited {
            fmt.Printf("[Worker %d] Skipped %s (already
visited)\n", id, result.Name)
            mu.Unlock()
            continue
        }
        if !in1.IsVisited || !in2.IsVisited {
            // fmt.Printf("[Worker %d] Skipped %s
(missing ingredients)\n", id, result.Name)
            mu.Unlock()
            continue
        }
        if result.Tier >= target.Tier {
            fmt.Printf("[Worker %d] Skipped %s (tier too
high)\n", id, result.Name)
            mu.Unlock()
            continue
        }
        if result.Tier <= in1.Tier || result.Tier <=

```

```

in2.Tier {
    fmt.Printf("[Worker %d] Skipped %s (tier not
increasing)\n", id, result.Name)
    mu.Unlock()
    continue
}

    fmt.Printf("[Worker %d] Checking recipe: %s (%d)
+ %s (%d) -> %s\n", id, in1.Name, in1.Tier, in2.Name,
in2.Tier, result.Name)

    result.IsVisited = true
    result.Left = true
    result.Children = make([]*RecipeNode, 0)
    result.Children = append(result.Children,
recipe)

    discovered[result.Name] = result
    tierElements[result.Tier] =
append(tierElements[result.Tier], result)
    fmt.Printf("[Worker %d] Discovered new element:
%s (tier %d)\n", id, result.Name, result.Tier)
    mu.Unlock()
    ingredient <- result
}
fmt.Printf("[Worker %d] Finished\n", id)
}

for currentTier := range target.Tier {
    fmt.Printf("[BFS] Processing tier %d -> %d\n",
currentTier, currentTier+1)
    select {
    case <-doneChan:
        fmt.Println("[BFS] Cancelled by DFS (doneChan
closed)")
        return
    default:
    }

    // Filter recipes that produce elements to next tier
    nextTier := currentTier + 1
    candidates := make([]*RecipeNode, 0)
    for _, recipe := range allRecipes {

```

```

        result := allElement[recipe.Result]
        if result.Tier == nextTier {
            candidates = append(candidates, recipe)
        }
    }

    fmt.Printf("[BFS] Tier %d: %d recipe candidates
found\n", nextTier, len(candidates))

    // Start workers
    numWorkers := 4
    chunkSize := (len(candidates) + numWorkers - 1) /
numWorkers
    for i := range numWorkers {
        start := i * chunkSize
        end := min(start + chunkSize, len(candidates))
        if start >= end {
            continue
        }
        wg.Add(1)
        go worker(i, candidates[start:end])
    }

    wg.Wait()

    // Drain ingredients
    drained := false
    count := 0
    for !drained {
        select {
        case <-doneChan:
            fmt.Println("[BFS] Received doneChan signal
during draining. Exiting early.")
            return

            case newEl := <-ingredient:
                fmt.Printf("[BFS] -> New element added: %s
(tier %d)\n", newEl.Name, newEl.Tier)
                count++
                if newEl.Tier == target.Tier {
                    fmt.Printf("[BFS] Target tier %d reached
with element %s\n", target.Tier, newEl.Name)
                    return
                }
            }
        }
    }
}

```



```

    }

    case <-time.After(500 * time.Millisecond):
        drained = true
    }
}

fmt.Printf("[BFS] Tier %d complete, %d new elements
discovered\n", currentTier, count)
}

fmt.Println("[BFS Left] Finished")
}

```

Fungsi Left side BFS dilakukan dengan membuat sebuah channel bernama *ingredient* yang dapat ditarik oleh fungsi untuk mencari recipe yang merupakan kombinasi dari ingredient tersedia dengan element lain dari pool *allRecipe* untuk tier selanjutnya. BFS left akan berhenti saat ada pesan pada *done* Channel oleh right side atau jika semua recipe pada tier yang sama dengan tier target sudah dijabarkan. Mekanisme penjabaran dilakukan dengan melakukan drain pada pool recipe yang sudah diambil oleh para worker yang dibagi recipe. jika ada recipe yang tidak sesuai (tier ingredient lebih tinggi dari tier hasil, atau hasil recipe sudah di visit, atau tier hasil recipe melebihi target tier) maka akan di skip, jika sesuai di labeli sebagai visit dan diberikan tepat satu recipe untuk node tersebut.

```

func Bidirect_Left_DFS(
    basic []*ElementNode,
    target *ElementNode,
    allElement map[string]*ElementNode,
    allRecipes []*RecipeNode,
    doneChan <-chan struct{},
) {
    stack := make([]*ElementNode, 0)
    var mu sync.Mutex

    // basic elements
    for _, el := range basic {
        stack = append(stack, el)
        el.IsVisited = true
    }
}

```

```

        el.Left = true
    }

    // DFS Loop
    for len(stack) > 0 {
        select {
            case <-doneChan:
                fmt.Println("[DFS] Received doneChan signal, exiting early.")
                return
            default:
                currentElement := stack[len(stack)-1]
                stack = stack[:len(stack)-1] // Pop

                fmt.Printf("[DFS] Processing element: %s (tier %d)\n", currentElement.Name, currentElement.Tier)

                if currentElement.Tier == target.Tier {
                    fmt.Printf("[DFS] Target tier %d reached with element %s\n", target.Tier, currentElement.Name)
                    return
                }

                for _, recipe := range allRecipes {
                    if currentElement.Name !=
recipe.Ingredient1.Name && currentElement.Name !=
recipe.Ingredient2.Name {
                        continue
                    }

                    newElement := allElement[recipe.Result]
                    if newElement == nil ||
newElement.IsVisited || !recipe.Ingredient1.IsVisited ||
!recipe.Ingredient2.IsVisited || newElement.Tier >=
target.Tier {
                        continue
                    }

                    // push
                    mu.Lock()
                    newElement.IsVisited = true
                    newElement.Left = true

```

```

        newElement.Children = make([]*RecipeNode, 0)
        newElement.Children =
append(newElement.Children, recipe)
        mu.Unlock()

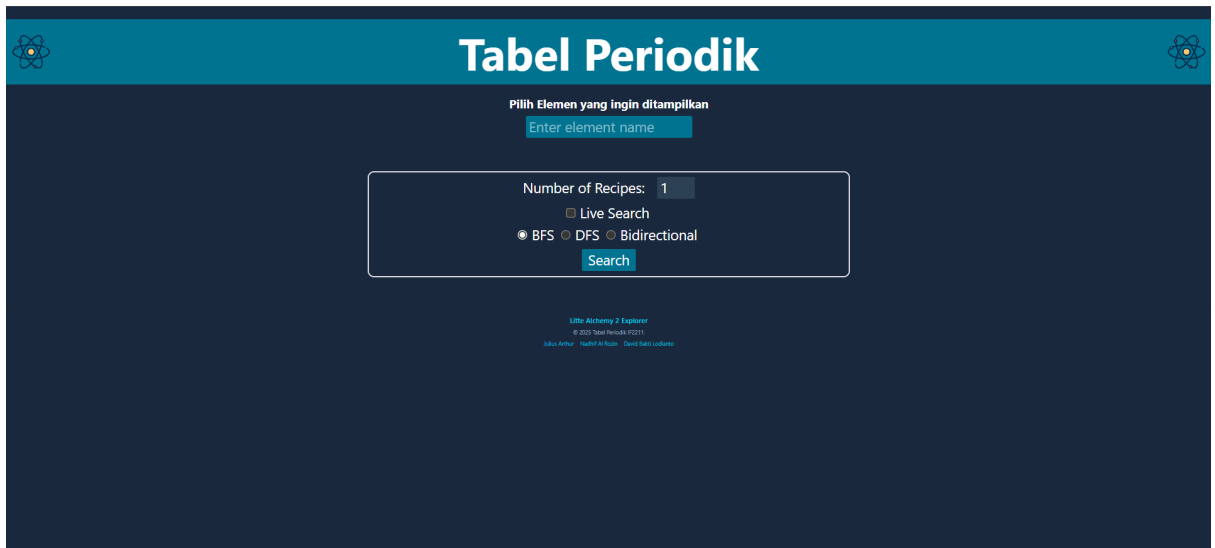
        stack = append(stack, newElement)
        fmt.Printf("[DFS] Discovered new element: %s
(tier %d)\n", newElement.Name, newElement.Tier)
    }
}

fmt.Println("[DFS Left] No more elements to process.
Exiting DFS.")
}

```

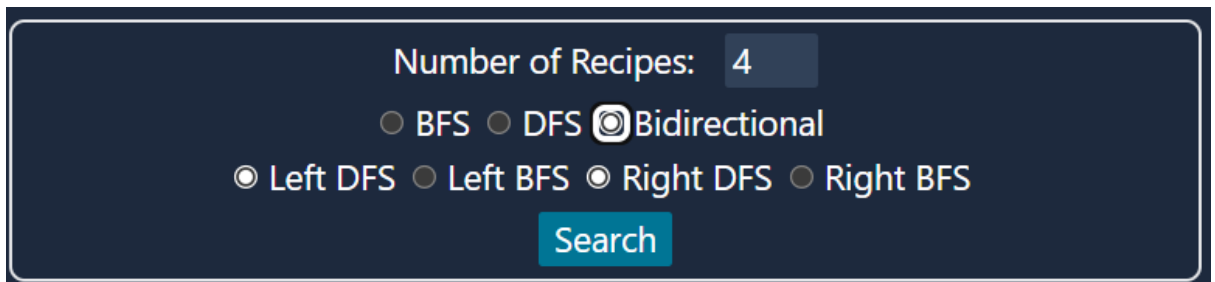
Fungsi Left side DFS dilakukan dengan membuat sebuah stack yang akan di pop dan push oleh fungsi, pop dilakukan untuk mengambil elemen yang ingin dijabarkan dan push dilakukan untuk menambahkan hasil dari recipe yang dipilih sebagai kandidat ingredient. DFS left akan berhenti saat ada pesan pada done Channel oleh right side atau jika semua recipe pada tier yang sama dengan tier target sudah dijabarkan. Pemilihan recipe dilakukan dengan mencari apakah ada recipe yang mempunyai ingredient element ini, jika ada di cek apakah sudah divisit, apa ingredient lain belum di visit dan apakah target tier lebih rendah dari tier hasil. jika salah satunya iya, maka recipe tersebut akan dikecualikan. jika tidak ada recipe yang ditemukan element akan hilang dari stack dan melakukan back track ke element di bawahnya. jika masih ditemukan recipe yang sesuai akan di push elemen baru yang merupakan hasil dari recipe. dan elemen lama dibiarkan hilang.

2. Tata Cara Penggunaan



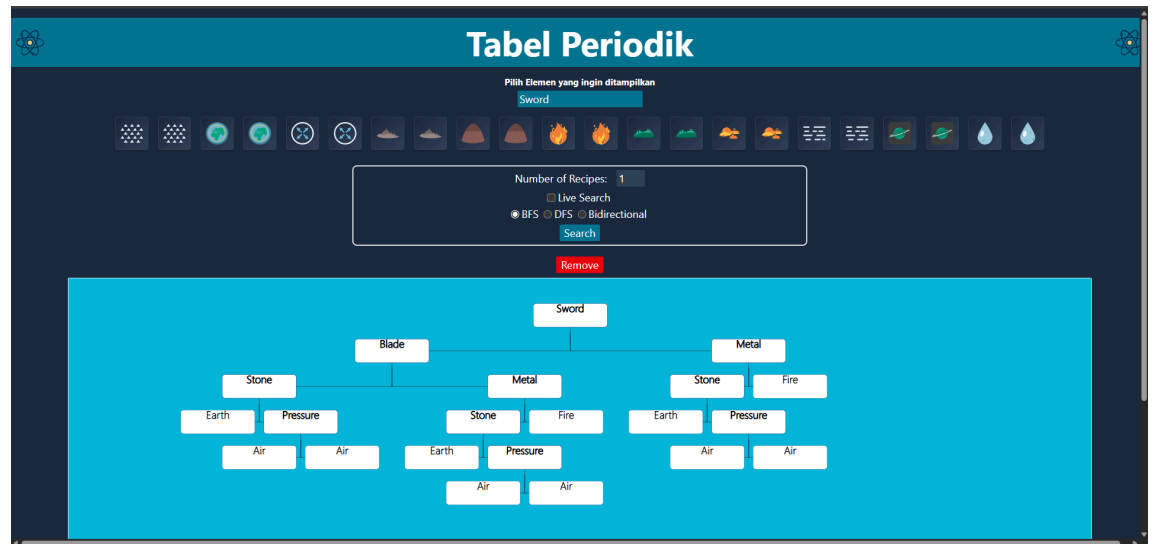
Pengguna dapat memasukkan elemen yang dicari pada box berwarna biru di atas. Kemudian, pengguna dapat memilih jumlah resep yang ingin dicari pada box Number of Recipes. Nilai lebih dari satu menandakan bahwa pengguna mencari multiple recipe. Terdapat toggle Live Search yang dapat dinyalakan. Dengan fitur Live Search, program akan memperbarui visualisasi hasil pencarian secara berkala. Pada fitur ini, pengguna juga dapat memilih waktu tunda untuk setiap pembaruan pohon pada visualisasi.

Kemudian, terdapat tiga algoritma pencarian yang dapat dipilih oleh pengguna, yakni BFS, DFS, dan Bidirectional.



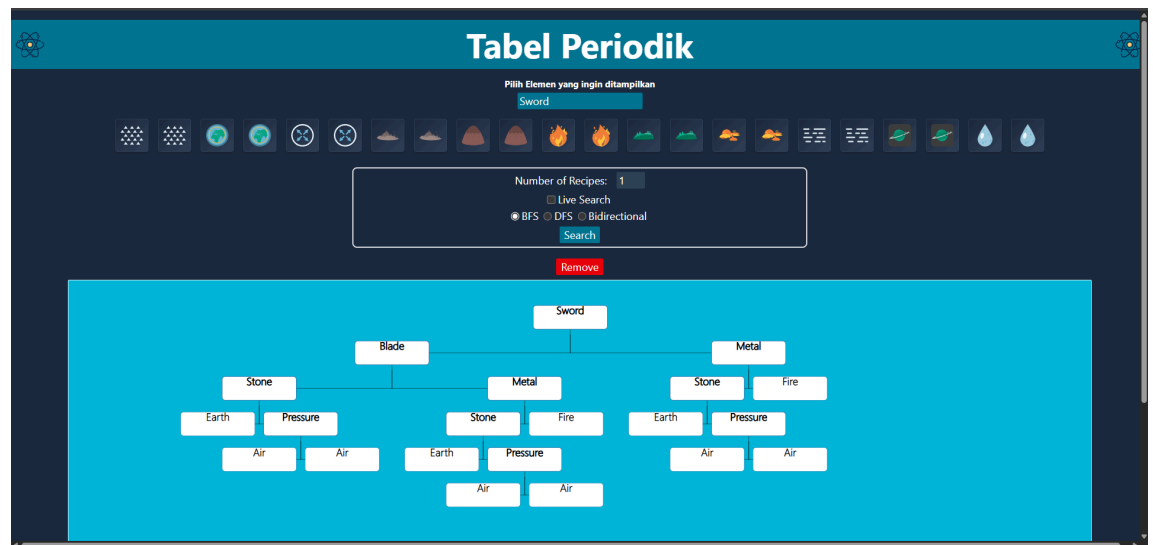
Jika memilih Bidirectional, terdapat 2 opsi yang harus ditentukan (Left DFS / Left BFS dan Right DFS/ Right BFS). Dengan Left DFS, program akan menelusuri sebuah elemen ke elemen dasarnya menggunakan DFS. Right DFS berarti program akan mencari dari elemen dasar ke elemen hasil menggunakan DFS. Hal yang serupa terjadi untuk Left BFS dan Right BFS.

Lalu, tekan tombol search untuk memulai pencarian. Berikut adalah contoh hasil pencarian sebuah elemen.



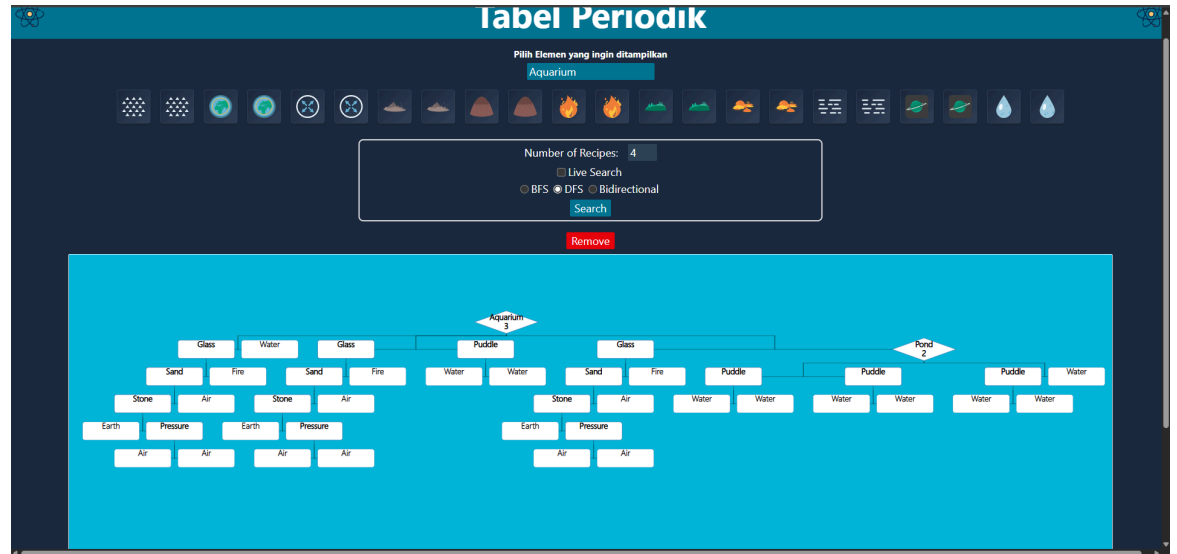
3. Hasil dan Analisis

- Pencarian elemen “Sword” dengan 1 resep dan tanpa fitur *live update* menggunakan algoritma BFS.



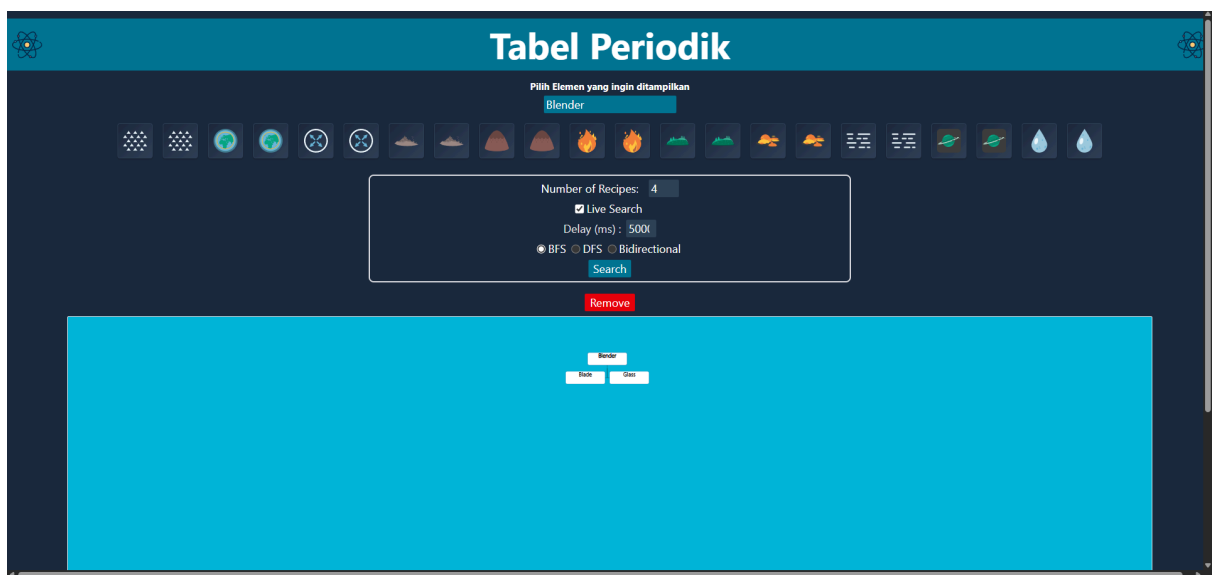
Program berhasil menghasilkan sebuah kombinasi resep untuk elemen “Sword”. Algoritma BFS berhasil menelusuri elemen secara bertingkat dan membatasinya pada 1 resep saja.

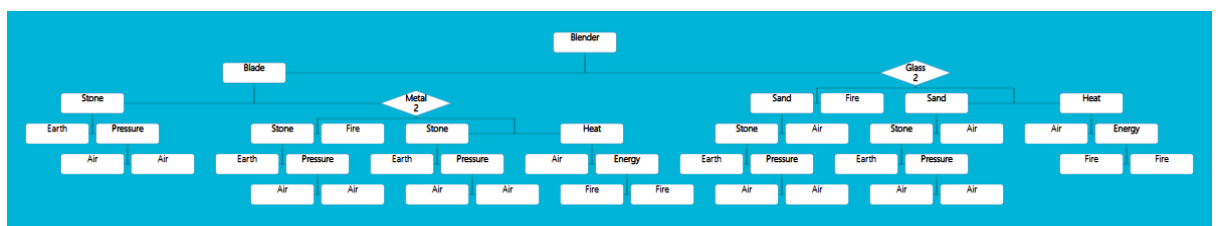
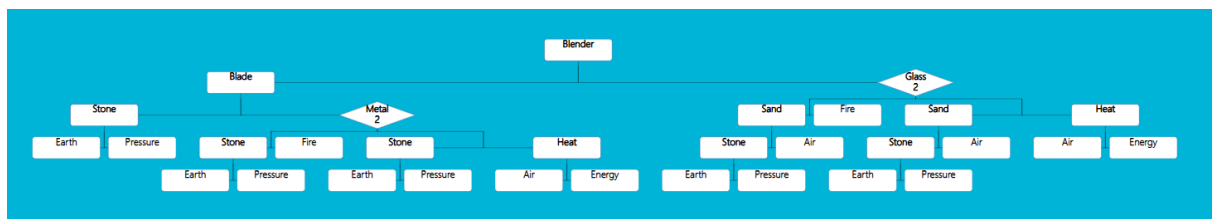
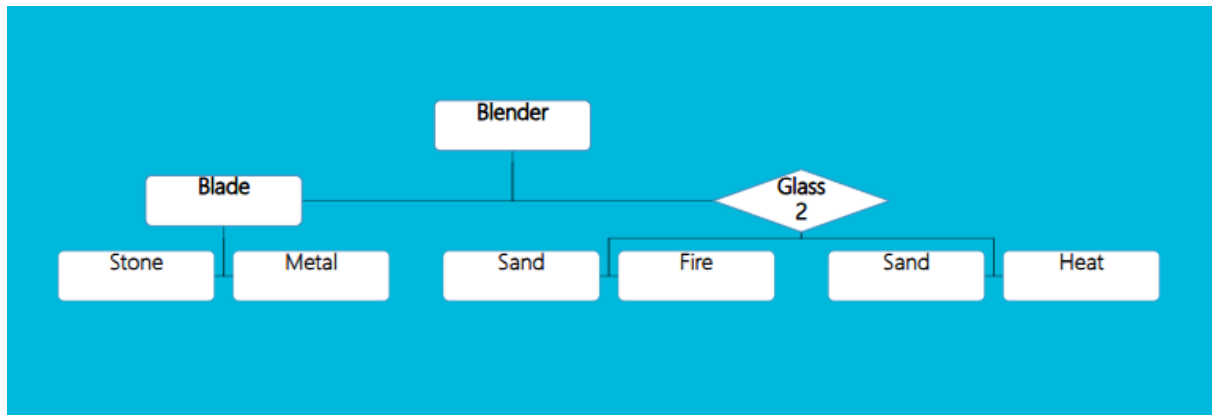
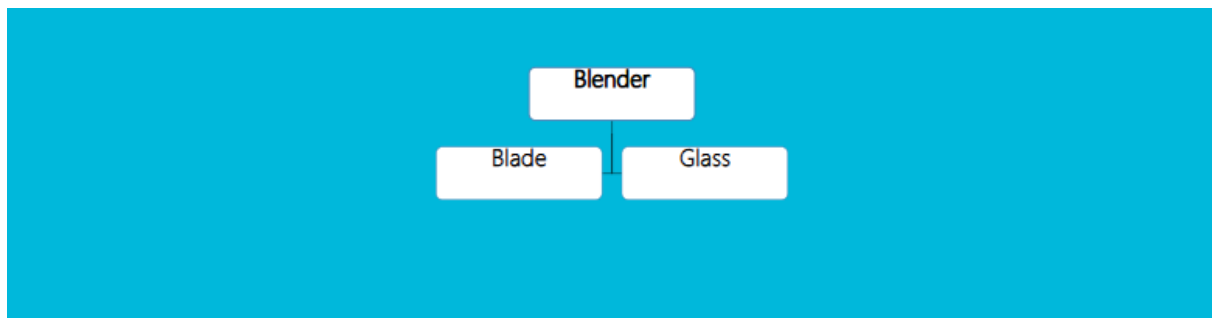
- b. Pencarian elemen “Aquarium” untuk 4 resep dan tanpa fitur *live update* menggunakan algoritma DFS.



Program berhasil menghasilkan sebuah kombinasi resep untuk elemen “Aquarium”. Algoritma DFS berhasil menelusuri elemen secara bertingkat dan membatasinya pada 4 resep saja. Terdapat 4 kombinasi untuk membuat Aquarium (*Glass + Water*, *Glass + Puddle*, dan *Glass + Pond*). Sedangkan, terdapat 2 kombinasi untuk membentuk sebuah Pond (*Puddle + Puddle* dan *Puddle + Water*). Maka secara total, terdapat 4 buah kombinasi membentuk elemen *Aquarium*.

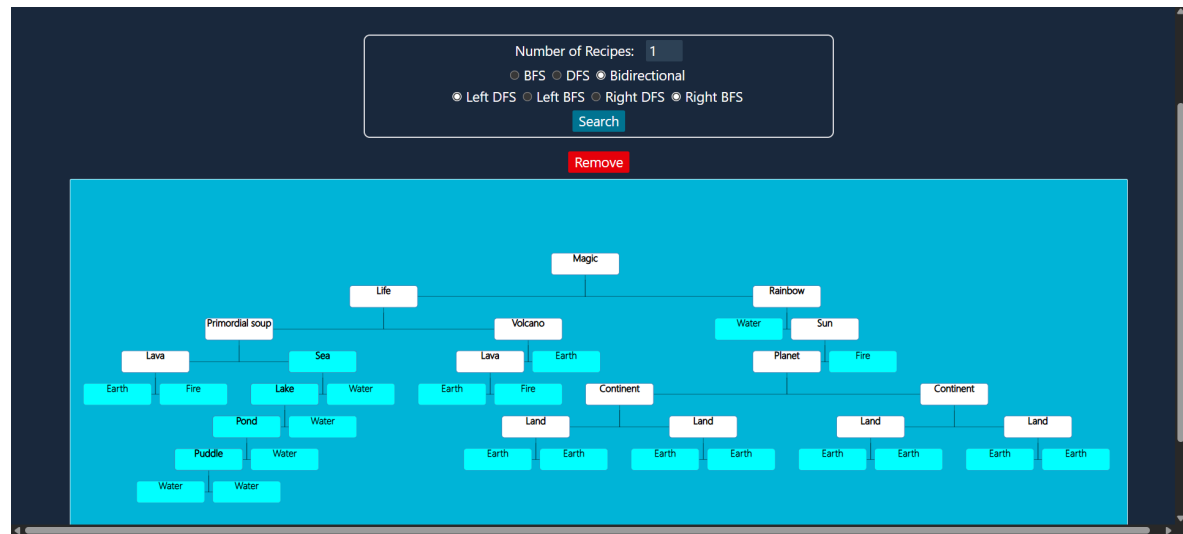
- c. Pencarian elemen “Blender” untuk 4 resep dan dengan fitur *live update* menggunakan algoritma BFS.





Program berhasil menghasilkan sebuah kombinasi resep untuk elemen “Blender” dan menampilkannya pada Frontend secara langsung/ *live*. Algoritma BFS berhasil menelusuri elemen secara bertingkat dan membatasinya pada 4 resep saja. Pada setiap *update*, visualisasi berhasil menunjukkan cara BFS melakukan iterasi dan pencarian pada graf yang ada. Setiap *update* menampilkan beberapa node baru untuk level yang baru. Hal ini terjadi secara konkuren dengan *goroutine* pada fungsi BFS.

- d. Pencarian elemen Magic untuk 1 resep dengan metode Bidirectional (Left DFS dan Right BFS).



Pencarian dimulai dari dua sisi yang berbeda, Magic dan elemen dasar (*Water*, *Earth*, dan *Fire*). Program berhasil menghasilkan sebuah resep untuk elemen “Magic”. Elemen dihasilkan dengan gabungan algoritma DFS dan BFS. BFS digunakan untuk mencari dari *right side* elemen, ditandai oleh elemen berwarna putih. Sedangkan, BFS digunakan pada *left side* elemen, ditunjukkan oleh elemen berwarna biru.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Penggunaan aplikasi berbasis web ini dalam mencari recipe yang dibutuhkan untuk membuat suatu elemen yang diinput oleh user berhasil memberikan hasil yang diharapkan. Baik dalam penggunaan metode BFS, DFS atau Bidirectional. Penggunaan fitur live update yang dapat memberikan perkembangan bagaimana pohon recipe dibuat juga berjalan dengan baik dan waktu delay yang digunakan sebagai kontrol kecepatan tiap snapshot dari tree juga bisa berjalan sebagaimana harapan kami.

2. Saran

Aplikasi ini masih bisa dikembangkan lagi mungkin dengan menambahkan metode search lain atau dengan mengkustomisasi tampilan tree yang dibuat, ataupun menyesuaikan dengan platform mobile. Selain itu ini juga bisa diberikan dokumentasi yang lebih menyeluruh baik di web ataupun dalam source code untuk user dan programmer lain yang ingin menggunakan projek ini

3. Refleksi

Kami sangat banyak belajar tentang tree, metode DFS, BFS, konkurensi, dan bahasa pemrograman Go selama kami mengerjakan tugas ini. Meskipun ada waktu dimana kami tidak bisa menemukan solusi untuk suatu masalah, tapi dengan waktu dan bimbingan yang cukup kami akhirnya bisa menyelesaikan proyek ini.

LAMPIRAN


- **Lampiran:** Tautan *repository* GitHub (dan video jika membuat)

Github Backend : https://github.com/koinen/Tubes2_BE_tabelPeriodik

Github Frontend : https://github.com/Narr21/Tubes2_FE_tabelPeriodik

Video : <https://www.youtube.com/watch?v=yIlg47J6uHw>

DAFTAR REFERENSI

- Munir, R. (2024). *Mata Kuliah Strategi Algoritmik - Semester I 2024/2025*. Institut Teknologi Bandung. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24-25.htm>
-  Concurrency in Go