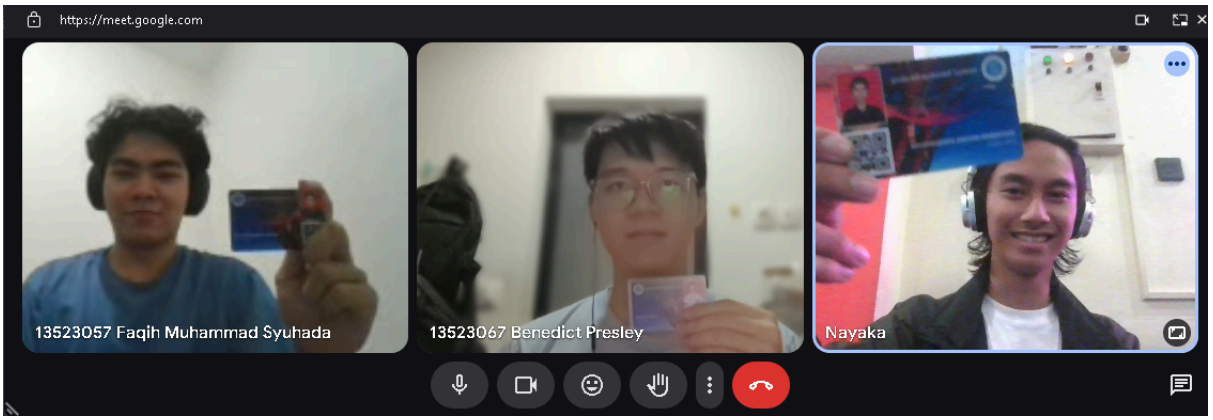
Laporan Tugas Besar 2
IF2211 Strategi Algoritma



Disusun oleh:
2 Pendiklat 1 Coach

Faqih Muhammad Syuhada	13523057
Benedict Presley	13523067
Zulfaqqar Nayaka Athadiansyah	13523094

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung

2025

Daftar Isi

1. Deskripsi Tugas

1.1. Gambaran Umum

Gambar 1. Poster gim Little Alchemy 2. (sumber: https://www.thegamer.com)

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Recloak dan 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**.

1.2. Komponen Permainan

1.2.1. 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 digabungkan menjadi elemen turunan yang berjumlah 720 elemen.

Gambar 2. Elemen dasar pada Little Alchemy 2

1.2.2. 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. Daftar elemen lengkap beserta *recipe*-nya dapat diakses di sini. Terdapat pengecualian terhadap beberapa elemen spesial seperti Time yang tidak dapat dihasilkan dari elemen lainnya, serta elemen-elemen mitologi dan monster (*Myths and Monsters*) yang tidak ada pada laman tadi.

1.2.3. Mekanisme Penggabungan Elemen

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.

2. Landasan Teori

2.1. Penjelajahan Graf

Penjelajahan graf adalah proses mengunjungi satu per satu simpul dalam sebuah graf secara sistematis, dengan asumsi grafnya terhubung. Penjelajahan graf dapat bertujuan untuk mencari simpul tertentu, memeriksa struktur graf (misalnya memeriksa apakah grafnya bipartite, siklik, dan/atau planar), menemukan jalur dari suatu simpul ke simpul lainnya, dan lain-lain. Konsep ini diterapkan dalam banyak hal yang membutuhkan pemodelan dengan graf, misalnya jaringan sosial (*social network*), jaringan komputer, pencarian jalur (*pathfinding*), dan masih banyak lagi. Ada sejumlah pendekatan yang dapat digunakan untuk melakukan penjelajahan graf, yakni *depth first search* (DFS) dan *breadth first search* (BFS).

2.2. Depth First Search (DFS)

Algoritma DFS, seperti namanya, mengutamakan kedalaman terlebih dahulu dalam menjelajahi graf dengan menelusuri suatu percabangan dalam graf sedalam mungkin. Tiap kali DFS menemukan suatu simpul yang bertetangga dengan lebih dari satu simpul lainnya, DFS akan memilih salah satu tetangga yang belum dikunjunginya, dan proses ini diteruskan hingga DFS mencapai simpul terdalam. Ketika hal ini tercapai, barulah dilakukan *backtracking*, yakni berjalan mundur hingga mencapai simpul dengan simpul tetangga yang belum ditelusuri. DFS diterapkan pada simpul tetangga tersebut, dan proses ini terus dilanjutkan hingga seluruh simpul sudah dikunjungi atau simpul yang dicari sudah ditemukan. Berikut adalah kode semu (*pseudocode*) dari DFS dengan pendekatan rekursif dan fungsi `doSomething(`v`)` adalah aksi yang dilakukan pada simpul v.

prosedur DFS(graph, u, visited)

KAMUS
G : graf yang dijelajahi, berupa senarai ketetanggaan
u : simpul yang sekarang diproses
visited : map untuk melacak simpul yang sudah dikunjungi

ALGORITMA
```
doSomething(u)
visited[u] ← true
for v in tetangga(G, u) do
    if (not visited[v]) then
        DFS(G, v, visited)
    endif
endfor
```

Pada graf dengan banyak simpul $|V|$ dan banyak sisi $|E|$ yang direpresentasikan dengan senarai ketetanggaan (*adjacency list*), kompleksitas waktu dari DFS adalah $O(|V| + |E|)$.

2.3. Breadth First Search (BFS)

Berbeda dengan DFS, algoritma BFS mengutamakan pencarian secara meluas dalam menjelajahi graf dengan menelusuri seluruh simpul tetangga dari suatu simpul pertama. Prosedur yang sama diterapkan pada tiap simpul tetangganya, dan terus demikian hingga seluruh simpul sudah dikunjungi atau simpul yang dicari sudah ditemukan. Dengan demikian, BFS menelusuri *layer* demi *layer* (bukan *brick by brick*), menjelajahi seluruh simpul pada kedalaman yang sama secara bertahap. Berikut adalah kode semu (*pseudocode*) dari BFS dengan pendekatan rekursif dan fungsi `doSomething(`v`)` adalah aksi yang dilakukan pada simpul v.

procedure BFS(G, u):

KAMUS
G : graf yang dijelajahi, berupa senarai ketetanggaan
u : simpul yang sekarang diproses
visited : map untuk melacak simpul yang sudah dikunjungi
Q : antrian (*queue*)

ALGORITMA

```
doSomething(u)
visited[u] ← true
enqueue(Q, u)

while (length(Q) ≠ 0):
    dequeue(Q, u)
    for v in tetangga(G, u) do
          if (not visited[v]) then
              doSomething(v)
              enqueue(Q, v)
              visited[v] ← true
          endif
    endfor
endwhile
```

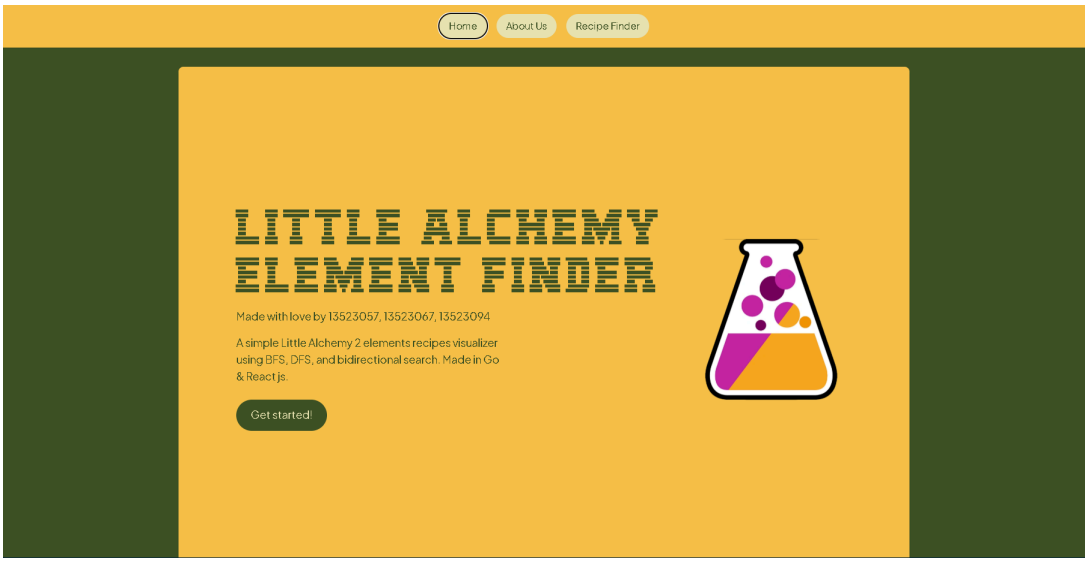
Pada graf dengan banyak simpul $|V|$ dan banyak sisi $|E|$ yang direpresentasikan dengan senarai ketetanggaan (*adjacency list*), kompleksitas waktu dari BFS adalah $O(|V| + |E|)$.

2.4. Aplikasi yang Dibangun

Aplikasi berbasis web yang diimplementasikan untuk tugas besar ini menggunakan *framework* pengembangan web Next.js dan bahasa pemrograman JavaScript untuk *frontend* serta bahasa pemrograman Go untuk *backend*. Bahasa pemrograman Go mendukung konkurensi melalui fitur `goroutine`, yang dapat mempercepat kerja aplikasi dengan *multithreading*.

Aplikasi ini dapat menampilkan pohon *recipes* untuk menyusun elemen yang diberikan oleh pengguna sebagai masukan beserta waktu pencarian yang dibutuhkan dan banyak simpul pada pohon *recipe* yang dikunjungi sepanjang pencarian. Pengguna dapat memilih algoritma

yang digunakan oleh aplikasi: DFS, BFS, atau *bidirectional search*. Pengguna juga dapat memilih untuk melihat jalur *recipe* terpendek atau sejumlah *recipe* dengan jumlah yang ditentukan oleh pengguna sendiri. Penjelasan mendalam dari aplikasi ini beserta fitur-fiturnya akan diberikan pada subbab Fitur dan Arsitektur Aplikasi.



3. Analisis Pemecahan Masalah

3.1. Langkah-langkah Pemecahan Masalah

3.1.1. Melakukan *Scraping*

Web scraping dilakukan untuk mendapatkan seluruh data elemen beserta *recipes* dan ikon SVG-nya dengan menggunakan pustaka `goquery` oleh Martin Angers. Ikon disimpan di direktori `src/fe/public/icons`, sementara data elemen dan *recipes*-nya disimpan di `src/be/data/elements.json` dalam bentuk sebagai berikut.

```
[
  {
      “element”: <nama-elemen>,
      “tier”: <0-15>,
      “recipes”: [
                    <bahan-1>, <bahan-2>
       ],
       …
  },
  …
]
```

3.1.2. Membangun Graf *Recipes*

Data dari `elements.json` disusun menjadi suatu senarai ketetanggaan yang dideklarasikan sebagai `AdjList = make([][][2]int, 0)` di Go. Representasi graf ini cocok dengan struktur `elements.json` dan tergolong relatif efisien untuk melakukan *traversal* karena mengakses *recipes* dari suatu elemen mempunyai kompleksitas algoritma $O(1)$. Representasi ini juga hemat memori, khususnya untuk graf *sparse* yang sesuai dengan objek permasalahan dari aplikasi ini karena tidak semua elemen di Little Alchemy 2 bisa sembarangan dikombinasikan.

3.1.3. Implementasi Algoritma Pencarian

Pengguna memasukkan nama elemen yang dicari beserta algoritma pencarian yang diinginkan. Berdasarkan masukan ini, pencarian pada graf akan dijalankan oleh program, baik dengan DFS atau BFS sambil mengkalkulasikan durasi dari pencarian dan jumlah simpul yang dikunjungi. Dua parameter ini, beserta jalur *recipes* yang ditemukan akan disimpan dalam struktur data `PathResult` yang dirincikan pada bagian Struktur Data. Algoritma DFS dan BFS yang diimplementasikan dapat mencari *recipe* paling pendek (yang memerlukan penggabungan paling sedikit) ataupun mencari beberapa *recipes* berbeda. Untuk melakukan pencarian, pertama dibangun semua pohon

recipe yang mungkin, lalu ditentukan pohon yang akan diambil. Dilakukan juga optimasi menggunakan *caching*.

3.1.4. Menampilkan Hasil

Data berbentuk `PathResult` akan di-*encode* menjadi JSON lalu dioper menuju *frontend*. Pohon *recipes* dari elemen yang dicari akan ditampilkan di layar pengguna beserta durasi pencarian dan jumlah simpul yang dikunjungi.

3.2. Pemetaan Masalah

Pohon *recipe* yang dibangkitkan oleh aplikasi ini tersusun atas tiap elemen dalam permainan sebagai simpul. Sepasang sisi yang menghubungkan suatu elemen dengan sepasang elemen menyatakan bahwa sepasang elemen tersebut merupakan *recipe* penyusun dari elemen semula. Misalnya, simpul "Mud" terhubung dengan simpul "Water" dan "Earth" karena "Mud" dihasilkan dari kombinasi kedua elemen tadi. Terakhir, graf dari seluruh elemen dinyatakan dalam bentuk senarai ketetanggaan (*adjacency list*).

3.3. Fitur

3.3.1. Pencarian Pohon *Recipe* Elemen dengan Visualisasi *Real-time*

Pengguna dapat mencari pohon *recipe* yang mungkin untuk menyusun suatu bahan dari bahan-bahan lain, dimulai dari bahan paling dasar hingga bahan-bahan lainnya dengan *tier* yang lebih rendah dari bahan yang dicari. Pencarian ini divisualisasikan secara tahap demi tahap.

3.3.2. Pemilihan Algoritma Pencarian

Pengguna dapat memilih algoritma yang ingin digunakan, baik DFS maupun BFS.

3.3.3. Pemilihan Mode Pencarian

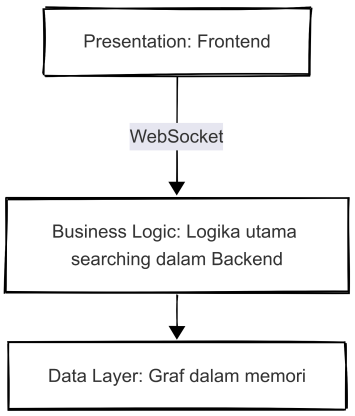
Pengguna dapat memilih untuk mencari pohon *recipes* dari satu recipe saja (mode *single recipe*) atau banyak *recipes* dengan jumlah yang sesuai dengan masukan dari pengguna.

3.4. Arsitektur Aplikasi

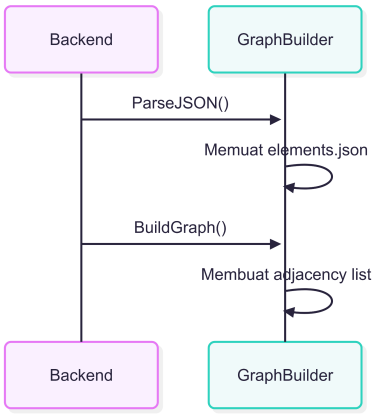
Aplikasi dibuat dengan React.js untuk *frontend*, bahasa pemrograman Go untuk *backend,* dan pustaka Gorilla *websocket* untuk melakukan pengiriman pesan secara kontinu dari *backend* ke *frontend* dan sebaliknya. Berikut adalah struktur folder dari aplikasi yang dibuat.

```
├───src
│   ├───be
│   │   ├───data
│   │   │   └───elements.json
│   │   ├───internal
│   │   │   ├───handler
│   │   │   │   └───websocket.go
│   │   │   ├───scraper
│   │   │   │   └───scraper.go
│   │   │   ├───tools
│   │   │   │   ├───graphBuilder.go
│   │   │   │   └───search.go
│   │   └───main.go
│   └───fe
│       ├───node_modules
│       ├───public
│       │   ├───fonts
│       │   └───icons
│       └───src
│           ├───assets
│           └───components
├docker-compose.yml
├Dockerfile.backend
└Dockerfile.frontend
```

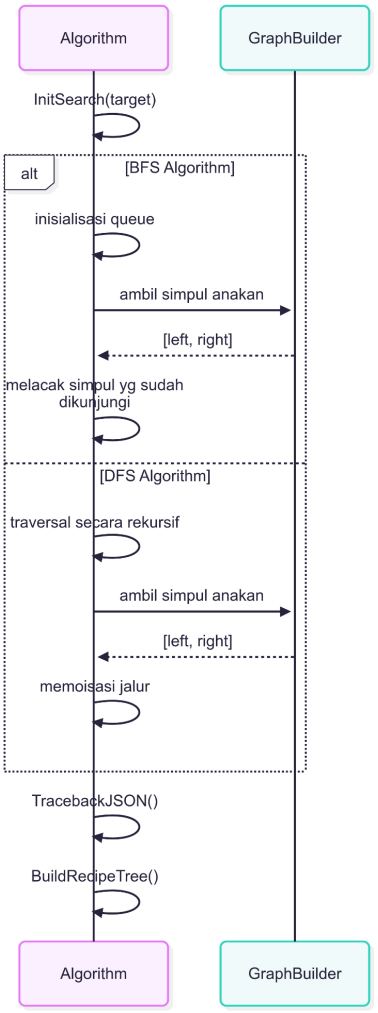
Aplikasi yang dikembangkan menggunakan pola pengembangan perangkat lunak *layered architecture* yang membedah aplikasi menjadi tiga *layer*: *presentation* (*frontend*), *business logic* (logika utama aplikasi, *event listener,* dll), dan *data* (memparsing `elements.json` dan menyimpannya sebagai graf dalam memori).



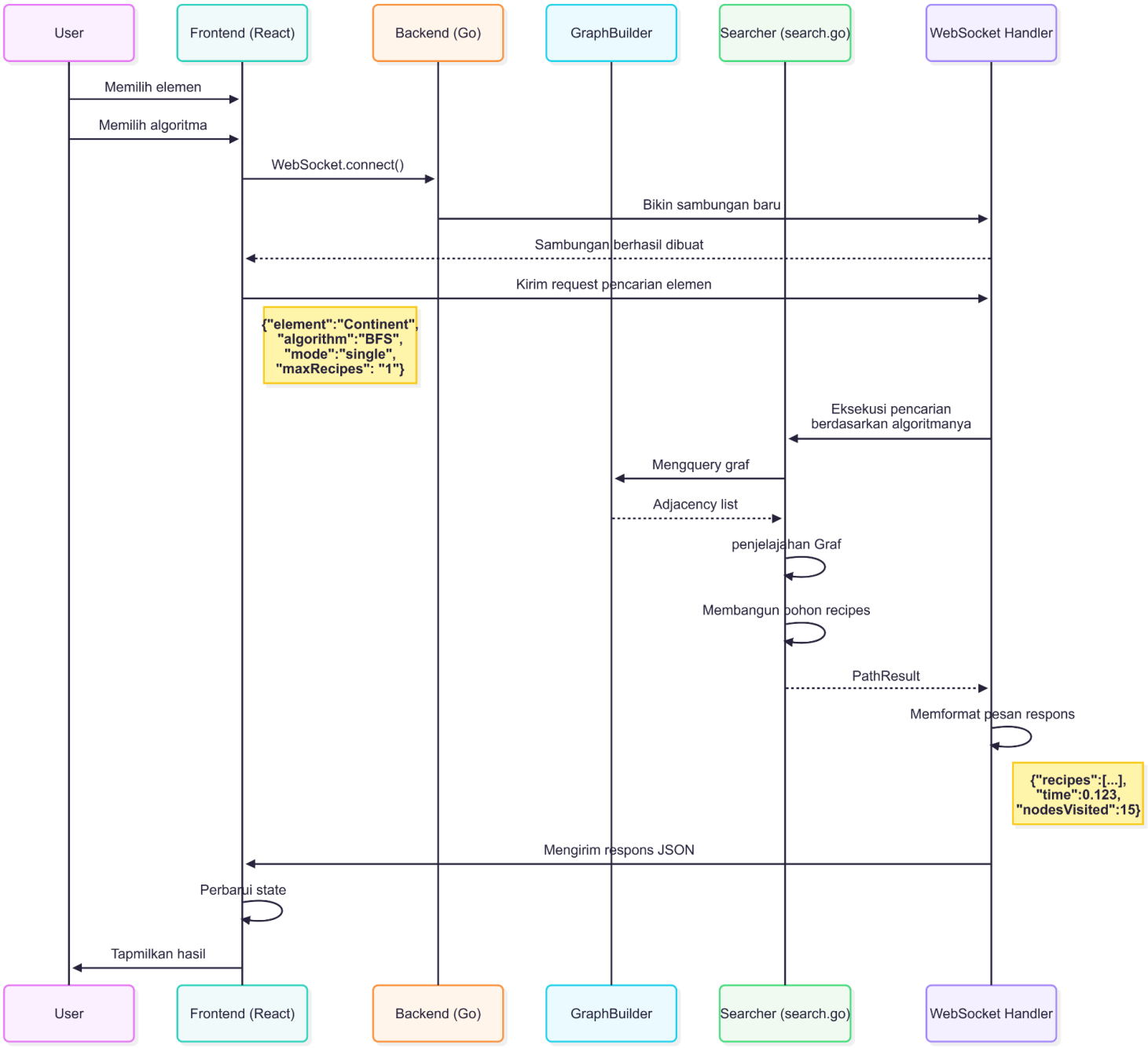
Mula-mula, dilakukan inisialisasi untuk meng-*scraping* data dari laman wiki Little Alchemy 2 dan menyimpannya dalam bentuk JSON, memuat dan mem-*parsing*-nya ke dalam aplikasi, dan membangun graf.



Secara umum, alur pencarian BFS dan DFS adalah sebagai berikut.



Alur kerja aplikasi dideskripsikan dengan diagram sekuens berikut.



3.5. Skenario

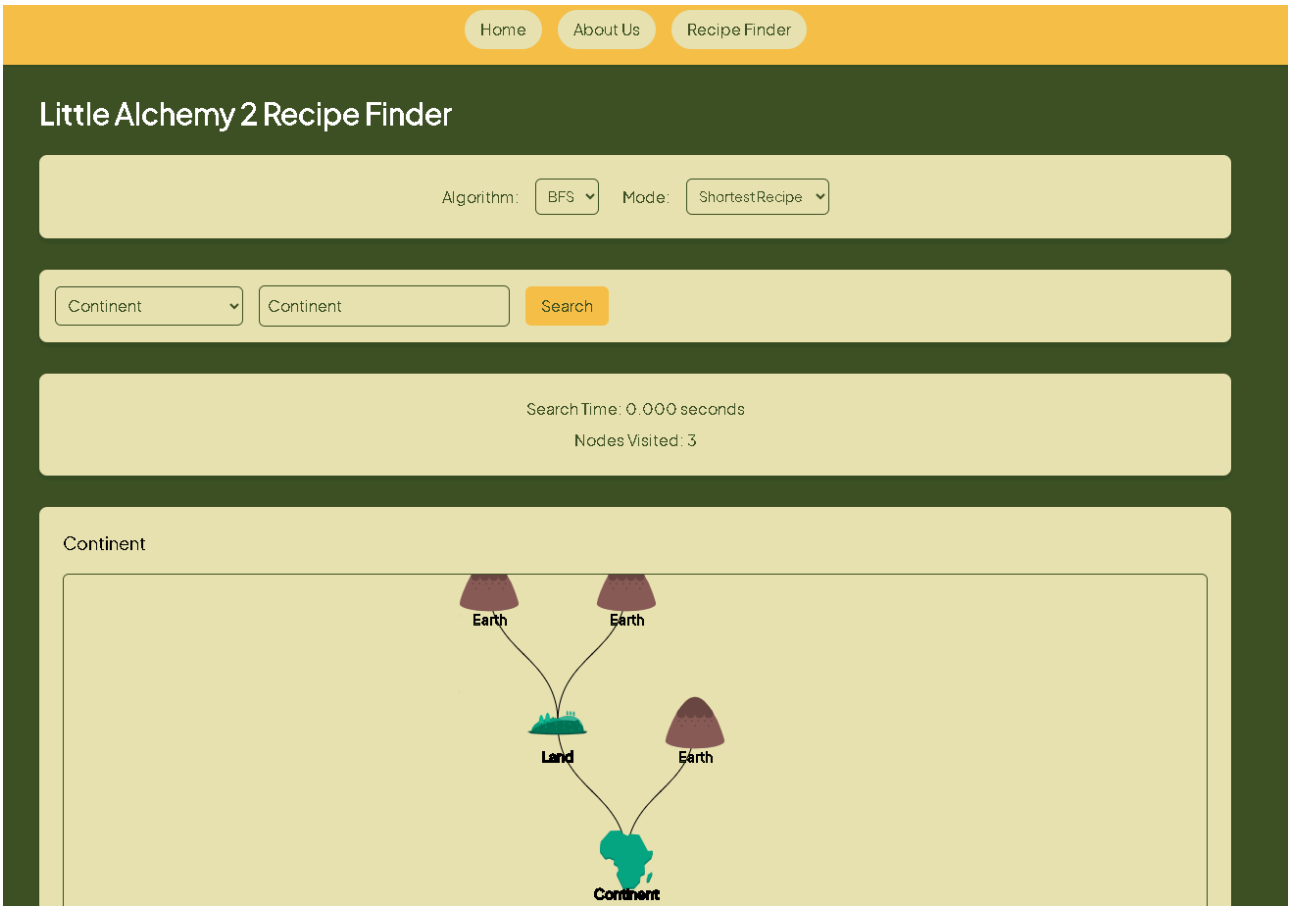
Ambil contoh kita ingin mencari elemen “Continent” dengan algoritma BFS dan mode *single recipe*. *Frontend* akan menyimpan kueri pencarian kita sebagai pesan JSON:

```
{"element":"Continent",
 "algorithm":"BFS",
 "mode":"single"}
```

Jika belum ada sambungan *websocket*, sambungan ini akan diinisialisasi kemudian pesan JSON tadi akan dikirim ke *backend* untuk ditangani oleh *handler* `websocket.go` yang akan memanggil fungsi pencarian `runBFS()` yang menghasilkan suatu data `PathResult` yang diterjemahkan ke JSON sebagai berikut.

```
{
    "recipes": [
        {
            "name": "Continent",
            "children": [
                {
                    "name": "Land",
                    "children": [
                        {"name": "Earth"},
                        {"name": "Earth"}
                    ]
                },
                {"name": "Earth"}
            ]
        }
    ],
    "time": 0.000121206,
    "nodesVisited": 3
}
```

Websocket handler kemudian mengirimkan pesan ini ke *frontend* untuk kemudian diproses dan divisualisasikan menjadi sebuah *tree* menggunakan React D3 Tree serta menampilkan durasi pencarian dan jumlah simpul yang dikunjungi.



4. Implementasi dan Pengujian

4.1. Spesifikasi Teknis

4.1.1. Struktur Data

Hasil *scraping* elemen dan *recipe* dari laman Little Alchemy 2 disimpan dalam *struct* `Element`.

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

Tiap simpul disimpan sebagai `Node`.

```
type Node struct {
    ID       int       `json:"id,omitempty"`
    Name     string    `json:"name"`
    Children []*Node   `json:"children,omitempty"`
```

```
}
```

Sementara itu, *request* dari *frontend* yang dikirim sebagai JSON yang dapat diterjemahkan sebagai *struct* berikut.

```
var request struct {
    Element string `json:"element"`
    Algo    string `json:"algorithm"`
    Mode    string `json:"mode"`
    MaxRecipes int `json:"maxRecipes"`
}
```

```
{
  “element”: <nama elemen>
  “algorithm”: <BFS/DFS>
  “mode”: <single/multiple>
  “maxRecipes”: <jumlah resep maksimum yang diminta, 1 untuk mode
single>
}
```

Pesan ini kemudian dibalas dengan respons JSON yang dapat direpresentasikan sebagai *struct* berikut.

```
type PathResult struct {
    Recipes      []*Node `json:"recipes"`
    Time         float64 `json:"time"`
    NodesVisited int     `json:"nodesVisited"`
}
```

```
{
  “recipes”: [
    {
      “name”: <nama-elemen>,
      “children”: [
        ...
      ]
    }
  ...
  ],
  “time”: <waktu dalam sekon>,
  “nodesVisited”: <jumlah simpul yang dikunjungi>
}
```

4.1.2. Subprogram

Berikut adalah *pseudocode snippet* dari sejumlah subprogram inti dalam aplikasi, khususnya dari *backend*.

src\be\internal\scraper\scraper.go

Melakukan *web scraping* seluruh elemen dan *recipes*-nya (dan ikon SVG-nya juga) pada laman wiki Little Elements 2 lalu

menyimpannya ke dalam `elements.json`.

```go
package scraper

import (
    "encoding/json"
    "io"
    "log"
    "net/http"
    "os"
    "strconv"
    "strings"
    "sync"
    "path/filepath"

    "github.com/PuerkitoBio/goquery"
)

// struktur data untuk menyimpan nama elemen beserta recipe penyusunnya
type Element struct {
    Name    string     `json:"element"`
    Tier    int        `json:"tier"`
    Recipes [][]string `json:"recipes"`
}

// fungsi untuk donlot file, di sini dipake untuk donlot SVG
func donlotFile(url, path string) error {
    resp, err := http.Get(url)
    if err ≠ nil {
        return err
    }
    defer resp.Body.Close()

    out, err := os.Create(path)
    if err ≠ nil {
        return err
    }
    defer out.Close()

    _, err = io.Copy(out, resp.Body)
    return err
}

// fungsi buat ngehapus elemen dari slice
func deleteElmt[T any](arr []T, idx int) []T {
    if idx < 0 || idx ≥ len(arr) {
        return arr
    }
    return append(arr[:idx], arr[idx+1:]...)
}

// fungsi untuk ngehapus elemen sesuai di parameter fungsinya
func deleteElmtByName(elmts []Element, name string) []Element {
    for i, elmt := range elmts {
        if elmt.Name == name {
            return deleteElmt(elmts, i)
        }
    }
    return elmts
}

// fungsi untuk ngehapus recipe yang melibatkan bahan dengan nama yang ada di parameter fungsinya
func deleteRecipeByName(elmts []Element, name string) []Element {
    for i, elmt := range elmts {
        for j := len(elmt.Recipes) - 1; j ≥ 0; j-- {
            recipe := elmt.Recipes[j]
            if recipe[0] == name || recipe[1] == name {
                elmt.Recipes = deleteElmt(elmt.Recipes, j)
                elmts[i] = elmt
            }
        }
    }
    return elmts
}

func Scrape() {
```

```
os.Mkdir("data/icons", 0755)

// get request ke halaman wiki Little Alchemy 2
res, err := http.Get("https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)")
if err ≠ nil {
    log.Fatal(err)
}
defer res.Body.Close()

// parsing HTML-nya
doc, err := goquery.NewDocumentFromReader(res.Body)
if err ≠ nil {
    log.Fatal(err)
}

var elements []Element
var wg sync.WaitGroup
var mu sync.Mutex

inElements := make(map[string]bool)

var currentTier int = 0

// ambil semua elemen dari tabel
doc.Find("h3, table.list-table").Each(func(i int, s *goquery.Selection) {
    // menentukan tier
    if goquery.NodeName(s) == "h3" {
        header := strings.ToLower(s.Text())
        if strings.Contains(header, "tier") {
            // pecah berdasarkan spasi
            // elemen kedua menyatakan tier
            parts := strings.Fields(header)
            if len(parts) ≥ 2 {
                if tierNum, err := strconv.Atoi(parts[1]); err == nil {
                    currentTier = tierNum
                }
            }
        }
    } else if goquery.NodeName(s) == "table" {
        s.Find("tr").Each(func(i int, row *goquery.Selection) {
            if i == 0 {
                return
            }

            // ambil nama elemen di kolom pertama
            element := strings.TrimSpace(row.Find("td:nth-child(1) a").Text())
            if element == "" {
                return
            }

            // cek apakah elemen sudah diproses
            mu.Lock()
            if inElements[element] {
                mu.Unlock()
                return
            }
            inElements[element] = true
            mu.Unlock()

            // ambil URL dari SVG di kolom pertama
            svgURL := ""
            imgTag := row.Find("td:nth-child(1) .icon-hover a").First()
            if imgHref, exists := imgTag.Attr("href"); exists {
                svgURL = imgHref
            }

            // ambil semua recipe dari kolom kedua
            var recipes [][]string
            row.Find("td:nth-child(2) li").Each(func(j int, li *goquery.Selection) {
                recipe := strings.TrimSpace(li.Text())
                parts := strings.Split(recipe, " + ")
                if len(parts) == 2 {
                    recipes = append(recipes, []string{strings.TrimSpace(parts[0]), strings.TrimSpace(parts[1])})
                }
            })

            wg.Add(1)
```

```
            go func(element, svgURL string, recipes [][]string, tier int) {
                defer wg.Done()
                // donlot SVG
                svgPath := filepath.Join("data/icons", element+".svg")
                if err := donlotFile(svgURL, svgPath); err ≠ nil {
                    log.Printf("Failed to download SVG for %s: %v", element, err)
                    return
                }

                // tambahin elemen ke slice
                mu.Lock()
                elements = append(elements, Element{
                    Name:    element,
                    Tier:    tier,
                    Recipes: recipes,
                })
                mu.Unlock()
            }(element, svgURL, recipes, currentTier)
        })
    }
})

// tunggu semua goroutine selesai
wg.Wait()

// ngehapusin elemen yang gaperlu berdasarkan QnA
https://docs.google.com/spreadsheets/d/1SVCNEBOYS0_eKShaHFIrx_5YVOg-V1uiBX-fAHpypxg
// klo ga salah: time, ruins, archeologist, dan elemen yg muncul sbg bahan recipe
// tapi gaada di kolom elements dari laman ini https://little-alchemy.fandom.com/wiki/Elements_(Myths_and_Monsters)

// maaf yh ini hardcode dikit soalnya gaada cara lain aowkaowkaokw
elements = deleteElmtByName(elements, "Ruins")
elements = deleteElmtByName(elements, "Archeologist")
elements = deleteRecipeByName(elements, "Ruins")
elements = deleteRecipeByName(elements, "Archeologist")

// ngeparsing laman myths and monsters
hapusin := []string{
    // "Time",
    "Ruins",
    "Archeologist"}

res, err = http.Get("https://little-alchemy.fandom.com/wiki/Elements_(Myths_and_Monsters)")
if err ≠ nil {
    log.Fatal(err)
}
defer res.Body.Close()
doc, err = goquery.NewDocumentFromReader(res.Body)
if err ≠ nil {
    log.Fatal(err)
}

doc.Find("table.list-table").Each(func(i int, s *goquery.Selection) {
    s.Find("tr").Each(func(i int, row *goquery.Selection) {
        if i == 0 {
            return
        }
        element := strings.TrimSpace(row.Find("td:nth-child(1) a").Text())
        if element == "" {
            return
        } else {
            // append ke slice hapusin
            hapusin = append(hapusin, element)
        }
    })
})

for _, elmt := range hapusin {
    elements = deleteElmtByName(elements, elmt)
    elements = deleteRecipeByName(elements, elmt)
}

// simpen data ke file JSON
file, err := os.Create("data/elements.json")
if err ≠ nil {
    log.Fatal(err)
```

```
    }
    defer file.Close()

    encoder := json.NewEncoder(file)
    encoder.SetIndent("", "    ")
    if err := encoder.Encode(elements); err ≠ nil {
        log.Fatal(err)
    }
}
```

src\be\internal\tools\graphBuilder.go

Membangun graf berupa senarai ketetanggaan dari `elements.json`.

```
package tools

import (
    "encoding/json"
    "log"
    "os"
)

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

var (
    TierMap   = make(map[string]int)
    NameToID  = make(map[string]int)
    IDToName  []string
    Frequency = make(map[string]int)
    AdjList   = make([][][2]int, 0)
    elements  []Element
)

func ParseJSON() {
    file, err := os.Open("data/elements.json")
    if err ≠ nil {
        log.Fatalf("Failed to open elements.json: %v", err)
    }
    defer file.Close()

    decoder := json.NewDecoder(file)
    if err := decoder.Decode(&elements); err ≠ nil {
        log.Fatalf("Failed to decode JSON: %v", err)
    }
}

func BuildGraph() {
    var counter int = 0
    for _, element := range elements {
        if _, exists := NameToID[element.Name]; !exists {
            NameToID[element.Name] = counter
            IDToName = append(IDToName, element.Name)
            counter++
        }
        TierMap[element.Name] = element.Tier
    }

    AdjList = make([][][2]int, len(NameToID))

    for _, element := range elements {
        product := element.Name
        productID := NameToID[product]

        for _, recipe := range element.Recipes {
            if len(recipe) ≠ 2 {
                continue
            }
```

```
            left, right := recipe[0], recipe[1]

            leftID := NameToID[left]
            rightID := NameToID[right]

            AdjList[productID] = append(AdjList[productID], [2]int{leftID, rightID})
        }
    }
}
```

src\be\internal\tools\tree.go

Modul pembantu untuk pemrosesan *tree*.

```
package tools

import (
    "fmt"
    "strings"
)

func ParseTree(s string) (*Node, error) {
    s = strings.TrimSpace(s)

    if len(s) == 0 {
        return nil, fmt.Errorf("empty string")
    }

    i := 0
    for i < len(s) && s[i] != '(' {
        i++
    }

    if i == len(s) {
        return &Node{Name: s}, nil
    }

    name := s[:i]

    node := &Node{
        Name:     name,
        Children: make([]*Node, 0, 2),
    }

    content := s[i:]
    level := 0
    start := 1

    for j := 1; j < len(content); j++ {
        if content[j] == '(' {
            level++
        } else if content[j] == ')' {
            level--
        }

        if (level == 0 && j < len(content)-1 && content[j+1] == '(') || (level == -1) {
            if j > start {
                child, err := ParseTree(content[start : j+1])
                if err != nil {
                    return nil, err
                }
                node.Children = append(node.Children, child)
            }

            if level == -1 {
                break
            }

            start = j + 1
        }
    }
```

```
    return node, nil
}

func CountNodes(node *Node) int {
    if node == nil {
        return 0
    }

    count := 1
    for _, child := range node.Children {
        count += CountNodes(child)
    }

    return count
}

func SortTree(node *Node) *Node {
    if node == nil || len(node.Children) < 2 {
        return node
    }

    for i := range node.Children {
        node.Children[i] = SortTree(node.Children[i])
    }

    if len(node.Children) == 2 {
        leftCount := CountNodes(node.Children[0])
        rightCount := CountNodes(node.Children[1])

        if rightCount > leftCount {
            node.Children[0], node.Children[1] = node.Children[1], node.Children[0]
        }
    }

    return node
}
```

src\be\internal\tools\search.go

Logika utama untuk pencarian DFS dan BFS.

```
package tools

import (
    "encoding/json"
    "fmt"
    "time"
)

const INF = 1000000000

var (
    DFSIndex     = 0
    Aux          = make([][]int, 0)
    RAux         = make([][]int, 0)
    NextHop      = make([]int, 0)
    Label        = make([]string, 0)
    NodesVisited = 0
    DP           = make([]int, 0)
    IDToNode     = make([]int, 0)
)

type Node struct {
    ID       int     `json:"id,omitempty"`
    Name     string  `json:"name"`
    Children []*Node `json:"children,omitempty"`
}

type PathResult struct {
    Recipes       []*Node `json:"recipes"`
```

```
    Time          float64 `json:"time"`
    TimeFormatted string  `json:"timeFormatted"`
    NodesVisited  int     `json:"nodesVisited"`
}

type Queue struct {
    items [][2]int
    head  int
    tail  int
    size  int
}

func NewQueue(capacity int) *Queue {
    return &Queue{
        items: make([][2]int, capacity),
        head:  0,
        tail:  0,
        size:  0,
    }
}

func (q *Queue) Enqueue(item [2]int) {
    if q.size == len(q.items) {
        q.resize()
    }

    q.items[q.tail] = item
    q.tail = (q.tail + 1) % len(q.items)
    q.size++
}

func (q *Queue) Dequeue() [2]int {
    if q.size == 0 {
        return [2]int{}
    }

    item := q.items[q.head]
    q.head = (q.head + 1) % len(q.items)
    q.size--
    return item
}

func (q *Queue) IsEmpty() bool {
    return q.size == 0
}

func (q *Queue) resize() {
    newItems := make([][2]int, len(q.items)*2)
    for i := 0; i < q.size; i++ {
        newItems[i] = q.items[(q.head+i)%len(q.items)]
    }
    q.items = newItems
    q.head = 0
    q.tail = q.size
}

type IntQueue struct {
    items []int
    head  int
    tail  int
    size  int
}

func NewIntQueue(capacity int) *IntQueue {
    return &IntQueue{
        items: make([]int, capacity),
        head:  0,
        tail:  0,
        size:  0,
    }
}

func (q *IntQueue) Enqueue(item int) {
    if q.size == len(q.items) {
        q.resize()
    }
```

```
    q.items[q.tail] = item
    q.tail = (q.tail + 1) % len(q.items)
    q.size++
}

func (q *IntQueue) Dequeue() int {
    if q.size == 0 {
        return -1
    }

    item := q.items[q.head]
    q.head = (q.head + 1) % len(q.items)
    q.size--
    return item
}

func (q *IntQueue) IsEmpty() bool {
    return q.size == 0
}

func (q *IntQueue) resize() {
    newItems := make([]int, len(q.items)*2)
    for i := 0; i < q.size; i++ {
        newItems[i] = q.items[(q.head+i)%len(q.items)]
    }
    q.items = newItems
    q.head = 0
    q.tail = q.size
}

func InitSearch(targetElem string) {
    DFSIndex = 0
    Aux = make([][]int, 1)
    RAux = make([][]int, 1)
    NextHop = make([]int, 1)
    Label = make([]string, 1)
    NodesVisited = 0

    Label[0] = targetElem

    elementCount := len(IDToName)
    DP = make([]int, elementCount)
    IDToNode = make([]int, elementCount)

    for i := 0; i < elementCount; i++ {
        DP[i] = INF
        IDToNode[i] = -1
    }
}

func DFS(current int, adjIndex int) int {
    NodesVisited++
    name := IDToName[adjIndex]

    IDToNode[adjIndex] = current

    if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
        DP[adjIndex] = 0
        return 0
    }

    if DP[adjIndex] != INF {
        return DP[adjIndex]
    }

    productTier := TierMap[name]

    for _, combination := range AdjList[adjIndex] {
        leftID, rightID := combination[0], combination[1]
        leftName, rightName := IDToName[leftID], IDToName[rightID]
        leftTier, rightTier := TierMap[leftName], TierMap[rightName]

        if leftTier >= productTier || rightTier >= productTier {
            continue
        }

        DFSIndex++
```

```
        Aux = append(Aux, []int{})
        NextHop = append(NextHop, 0)
        Label = append(Label, "!")
        temp := DFSIndex

        Aux[current] = append(Aux[current], temp)

        var countLeft, countRight int

        if DP[leftID] == INF {
            DFSIndex++
            Aux = append(Aux, []int{})
            NextHop = append(NextHop, 0)
            Label = append(Label, leftName)
            Aux[temp] = append(Aux[temp], DFSIndex)
            countLeft = DFS(DFSIndex, leftID)
        } else {
            Aux[temp] = append(Aux[temp], IDToNode[leftID])
            countLeft = DP[leftID]
        }

        if DP[rightID] == INF {
            DFSIndex++
            Aux = append(Aux, []int{})
            NextHop = append(NextHop, 0)
            Label = append(Label, rightName)
            Aux[temp] = append(Aux[temp], DFSIndex)
            countRight = DFS(DFSIndex, rightID)
        } else {
            Aux[temp] = append(Aux[temp], IDToNode[rightID])
            countRight = DP[rightID]
        }

        totalCost := countLeft + countRight + 1
        if totalCost < DP[adjIndex] {
            DP[adjIndex] = totalCost
            NextHop[current] = temp
        }
    }

    return DP[adjIndex]
}

func BFS(targetElem string) int {
    targetID := NameToID[targetElem]
    BFSIndex := 0

    deg := make([]int, 1)
    minimum := make([]int, 1)
    p := NewIntQueue(100)
    q := NewQueue(100)

    for i := 0; i < len(IDToName); i++ {
        IDToNode[i] = -1
    }

    q.Enqueue([2]int{0, targetID})
    minimum[0] = INF
    Label[0] = targetElem
    IDToNode[targetID] = 0

    for !q.IsEmpty() {
        NodesVisited++
        pair := q.Dequeue()
        current := pair[0]
        adjIndex := pair[1]

        name := IDToName[adjIndex]

        if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
            minimum[current] = 0
            p.Enqueue(current)
            continue
        }

        productTier := TierMap[name]
```

```
    for _, combination := range AdjList[adjIndex] {
        leftID, rightID := combination[0], combination[1]
        leftName, rightName := IDToName[leftID], IDToName[rightID]
        leftTier, rightTier := TierMap[leftName], TierMap[rightName]

        if leftTier ≥ productTier || rightTier ≥ productTier {
            continue
        }

        BFSIndex++
        Aux = append(Aux, []int{})
        RAux = append(RAux, []int{})
        NextHop = append(NextHop, 0)
        deg = append(deg, 2)
        minimum = append(minimum, 1)
        Label = append(Label, "!")
        temp := BFSIndex

        Aux[current] = append(Aux[current], temp)
        RAux[temp] = append(RAux[temp], current)
        deg[current]++

        if IDToNode[leftID] == -1 {
            BFSIndex++
            Aux = append(Aux, []int{})
            RAux = append(RAux, []int{})
            NextHop = append(NextHop, 0)
            deg = append(deg, 0)
            minimum = append(minimum, INF)
            Label = append(Label, leftName)
            IDToNode[leftID] = BFSIndex

            Aux[temp] = append(Aux[temp], BFSIndex)
            RAux[BFSIndex] = append(RAux[BFSIndex], temp)
            q.Enqueue([2]int{BFSIndex, leftID})
        } else {
            Aux[temp] = append(Aux[temp], IDToNode[leftID])
            RAux[IDToNode[leftID]] = append(RAux[IDToNode[leftID]], temp)
        }

        if IDToNode[rightID] == -1 {
            BFSIndex++
            Aux = append(Aux, []int{})
            RAux = append(RAux, []int{})
            NextHop = append(NextHop, 0)
            deg = append(deg, 0)
            minimum = append(minimum, INF)
            Label = append(Label, rightName)
            IDToNode[rightID] = BFSIndex

            Aux[temp] = append(Aux[temp], BFSIndex)
            RAux[BFSIndex] = append(RAux[BFSIndex], temp)
            q.Enqueue([2]int{BFSIndex, rightID})
        } else {
            Aux[temp] = append(Aux[temp], IDToNode[rightID])
            RAux[IDToNode[rightID]] = append(RAux[IDToNode[rightID]], temp)
        }
    }
}

for !p.IsEmpty() {
    current := p.Dequeue()
    moves := minimum[current]

    if current == 0 {
        continue
    }

    for _, elem := range RAux[current] {
        deg[elem]--

        if Label[elem] == "!" {
            minimum[elem] += moves
        } else {
            if minimum[elem] > moves {
                minimum[elem] = moves
                NextHop[elem] = current
```

```
                }
            }

            if deg[elem] == 0 {
                p.Enqueue(elem)
            }
        }
    }

    return minimum[0]
}

func DFSBuildRecipes(current int, need int) []string {
    name := Label[current]
    result := make([]string, 0)

    if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
        result = append(result, "("+name+")")
        return result
    }

    if name != "!" {
        for _, candidate := range Aux[current] {
            possible := DFSBuildRecipes(candidate, need)
            for _, s := range possible {
                result = append(result, "("+name+s+")")
                need--
                if need <= 0 {
                    break
                }
            }
            if need <= 0 {
                break
            }
        }
        return result
    }

    leftRecipes := DFSBuildRecipes(Aux[current][0], need)
    leftLen := len(leftRecipes)

    rightNeeded := (need + leftLen - 1) / leftLen
    if rightNeeded < 1 {
        rightNeeded = 1
    }
    rightRecipes := DFSBuildRecipes(Aux[current][1], rightNeeded)

    for _, s := range rightRecipes {
        for _, t := range leftRecipes {
            result = append(result, t+s)
            need--
            if need <= 0 {
                break
            }
        }
        if need <= 0 {
            break
        }
    }

    return result
}

func BFSBuildRecipes(startNode int, need int) []string {
    memo := make([][]string, len(Aux))
    for i := range memo {
        memo[i] = make([]string, 0)
    }

    queue := NewIntQueue(100)
    deg := make([]int, len(Aux))

    for i := 0; i < len(Aux); i++ {
        name := Label[i]
        if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
            queue.Enqueue(i)
        }
```

```
    deg[i] = len(Aux[i])
}

for !queue.IsEmpty() {
    node := queue.Dequeue()
    name := Label[node]

    if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
        memo[node] = append(memo[node], "("+name+")")
        for _, nextNode := range RAux[node] {
            deg[nextNode]--
            if deg[nextNode] == 0 {
                queue.Enqueue(nextNode)
            }
        }
        continue
    }

    if name ≠ "!" {
        recipes := make([]string, 0)
        for _, recipeNode := range Aux[node] {
            if len(memo[recipeNode]) == 0 {
                continue
            }

            remaining := need - len(recipes)
            if remaining ≤ 0 {
                break
            }

            copyCount := 0
            if remaining < len(memo[recipeNode]) {
                copyCount = remaining
            } else {
                copyCount = len(memo[recipeNode])
            }
            for i := 0; i < copyCount; i++ {
                recipes = append(recipes, "("+name+memo[recipeNode][i]+")")
            }
        }
        memo[node] = recipes

        for _, nextNode := range RAux[node] {
            deg[nextNode]--
            if deg[nextNode] == 0 {
                queue.Enqueue(nextNode)
            }
        }
        continue
    }

    combined := make([]string, 0)
    left := Aux[node][0]
    right := Aux[node][1]

    if len(memo[left]) > 0 && len(memo[right]) > 0 {
        for _, l := range memo[left] {
            if len(combined) ≥ need {
                break
            }

            remaining := need - len(combined)
            copyCount := min(remaining, len(memo[right]))

            for i := 0; i < copyCount; i++ {
                combined = append(combined, l+memo[right][i])
            }
        }
    }

    memo[node] = combined

    for _, nextNode := range RAux[node] {
        deg[nextNode]--
        if deg[nextNode] == 0 {
            queue.Enqueue(nextNode)
        }
```

```
        }
    }

    return memo[startNode]
}

func TracebackJSON(current int, recipeId int) *Node {
    name := Label[current]

    if name == "Fire" || name == "Water" || name == "Earth" || name == "Air" || name == "Time" {
        return &Node{
            Name: name,
        }
    }

    if name == "!" {
        left := TracebackJSON(Aux[current][0], 0)
        right := TracebackJSON(Aux[current][1], 0)

        children := []*Node{left, right}

        return &Node{
            Children: children,
        }
    }

    node := &Node{
        Name: name,
    }

    if NextHop[current] != 0 {
        recipe := TracebackJSON(NextHop[current], 0)
        node.Children = recipe.Children
    }

    return node
}

func BuildRecipeTree(current int) []*Node {
    recipes := []*Node{}

    mainRecipe := &Node{
        ID:   0,
        Name: Label[current],
    }

    if NextHop[current] != 0 {
        recipe := TracebackJSON(NextHop[current], 0)
        mainRecipe.Children = recipe.Children
    }

    recipes = append(recipes, mainRecipe)

    return recipes
}

func RunDFS(targetElem string) (int, string) {
    startTime := time.Now()

    InitSearch(targetElem)

    targetID := NameToID[targetElem]
    steps := DFS(0, targetID)

    elapsedTime := time.Since(startTime)
    elapsedNano := elapsedTime.Nanoseconds()

    timeFormatted := fmt.Sprintf("%.4f µs", float64(elapsedNano)/1000)

    recipe := BuildRecipeTree(0)
    recipe[0] = SortTree(recipe[0])

    result := PathResult{
        Recipes:       recipe,
        Time:          elapsedTime.Seconds(),
        TimeFormatted: timeFormatted,
        NodesVisited:  NodesVisited,
```

```
    }

    jsonData, err := json.Marshal(result)
    if err != nil {
        return steps, "{\"error\": \"Failed to generate JSON\"}"
    }

    return steps, string(jsonData)
}

func RunDFSMultiple(targetElem string, need int) string {
    startTime := time.Now()

    InitSearch(targetElem)

    targetID := NameToID[targetElem]
    DFS(0, targetID)

    recipeList := DFSBuildRecipes(0, need)

    recipes := make([]*Node, len(recipeList))
    var err error

    for i, recipe := range recipeList {
        recipes[i], err = ParseTree(recipe)
        recipes[i] = SortTree(recipes[i])
        if err != nil {
            return "{\"error\": \"Failed to parse recipe tree\"}"
        }
    }

    elapsedTime := time.Since(startTime)
    elapsedNano := elapsedTime.Nanoseconds()

    timeFormatted := fmt.Sprintf("%.4f µs", float64(elapsedNano)/1000)

    result := PathResult{
        Recipes:       recipes,
        Time:          elapsedTime.Seconds(),
        TimeFormatted: timeFormatted,
        NodesVisited:  NodesVisited,
    }

    jsonData, err := json.Marshal(result)
    if err != nil {
        return "{\"error\": \"Failed to generate JSON\"}"
    }

    return string(jsonData)
}

func RunBFS(targetElem string) (int, string) {
    startTime := time.Now()

    InitSearch(targetElem)

    steps := BFS(targetElem)

    elapsedTime := time.Since(startTime)
    elapsedNano := elapsedTime.Nanoseconds()

    timeFormatted := fmt.Sprintf("%.4f µs", float64(elapsedNano)/1000)

    recipe := BuildRecipeTree(0)
    recipe[0] = SortTree(recipe[0])

    result := PathResult{
        Recipes:       recipe,
        Time:          elapsedTime.Seconds(),
        TimeFormatted: timeFormatted,
        NodesVisited:  NodesVisited,
    }

    jsonData, err := json.Marshal(result)
    if err != nil {
        return steps, "{\"error\": \"Failed to generate JSON\"}"
    }
```

```
    return steps, string(jsonData)
}

func RunBFSMultiple(targetElem string, need int) string {
    startTime := time.Now()

    InitSearch(targetElem)

    BFS(targetElem)

    recipeList := BFSBuildRecipes(0, need)

    recipes := make([]*Node, len(recipeList))
    var err error

    for i, recipe := range recipeList {
        fmt.Println(recipe)
        recipes[i], err = ParseTree(recipe)
        recipes[i] = SortTree(recipes[i])
        if err ≠ nil {
            return "{\"error\": \"Failed to parse recipe tree\"}"
        }
    }

    elapsedTime := time.Since(startTime)
    elapsedNano := elapsedTime.Nanoseconds()

    timeFormatted := fmt.Sprintf("%.4f µs", float64(elapsedNano)/1000)

    result := PathResult{
        Recipes:       recipes,
        Time:          elapsedTime.Seconds(),
        TimeFormatted: timeFormatted,
        NodesVisited:  NodesVisited,
    }

    jsonData, err := json.Marshal(result)
    if err ≠ nil {
        return "{\"error\": \"Failed to generate JSON\"}"
    }

    return string(jsonData)
}
```

src\be\internal\handler\websocket.go

Mengatur sambungan *websocket* antara *frontend* dengan *backend*.

```
package handler

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

    "github.com/gorilla/websocket"
    "github.com/BP04/Tubes2_2Pendiklat1Coach/internal/tools"
)

type Node struct {
    ID       int      `json:"id,omitempty"`
    Name     string   `json:"name"`
    Children []*Node  `json:"children,omitempty"`
}

type PathResult struct {
    Recipes      []*Node `json:"recipes"`
    Time         float64 `json:"time"`
    NodesVisited int     `json:"nodesVisited"`
```

```
}

var upgrader = websocket.Upgrader{
    CheckOrigin: func(r *http.Request) bool {
        return true
    },
}

// fungsi untuk ngoper ./data/elements.json ke frontend,
func GetElements(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Content-Type", "application/json")
    file, err := os.Open("data/elements.json")
    if err ≠ nil {
        http.Error(w, "Failed to open elements.json", http.StatusInternalServerError)
        return
    }
    defer file.Close()

    var elements []tools.Element
    if err := json.NewDecoder(file).Decode(&elements); err ≠ nil {
        http.Error(w, "Failed to decode JSON", http.StatusInternalServerError)
        return
    }

    if err := json.NewEncoder(w).Encode(elements); err ≠ nil {
        http.Error(w, "Failed to encode JSON", http.StatusInternalServerError)
    }
}

func WebSocketHandler(w http.ResponseWriter, r *http.Request) {
    ws, err := upgrader.Upgrade(w, r, nil)
    if err ≠ nil {
        log.Println("Error upgrading connection:", err)
        return
    }
    defer ws.Close()
    for {
        var request struct {
            Element string `json:"element"`
            Algo    string `json:"algorithm"`
            Mode    string `json:"mode"`
            MaxRecipes int `json:"maxRecipes"`
        }
        if err := ws.ReadJSON(&request); err ≠ nil {
            log.Println("Error reading JSON:", err)
            break
        }

        var response PathResult

        var path string
        switch request.Mode {
        case "single":
            if request.Algo == "BFS" {
                _, path = tools.RunBFS(request.Element)
            } else if request.Algo == "DFS" {
                _, path = tools.RunDFS(request.Element)
            }
        case "multiple":
            if request.Algo == "BFS" {
                path = tools.RunBFSMultiple(request.Element, request.MaxRecipes)
            } else if request.Algo == "DFS" {
                path = tools.RunDFSMultiple(request.Element, request.MaxRecipes)
            }
        }

        if err := json.Unmarshal([]byte(path), &response); err ≠ nil {
            log.Println("Error unmarshalling JSON:", err)
            break
        }

        jsonResponse, err := json.Marshal(response)
        if err ≠ nil {
            log.Println("Error marshalling JSON:", err)
            break
        }
```

```
        if err := ws.WriteMessage(websocket.TextMessage, jsonResponse); err ≠ nil {
            log.Println("Error writing message:", err)
            break
        }
    }
}
```

Terakhir, ini adalah program utama untuk menjalankan *backend*:

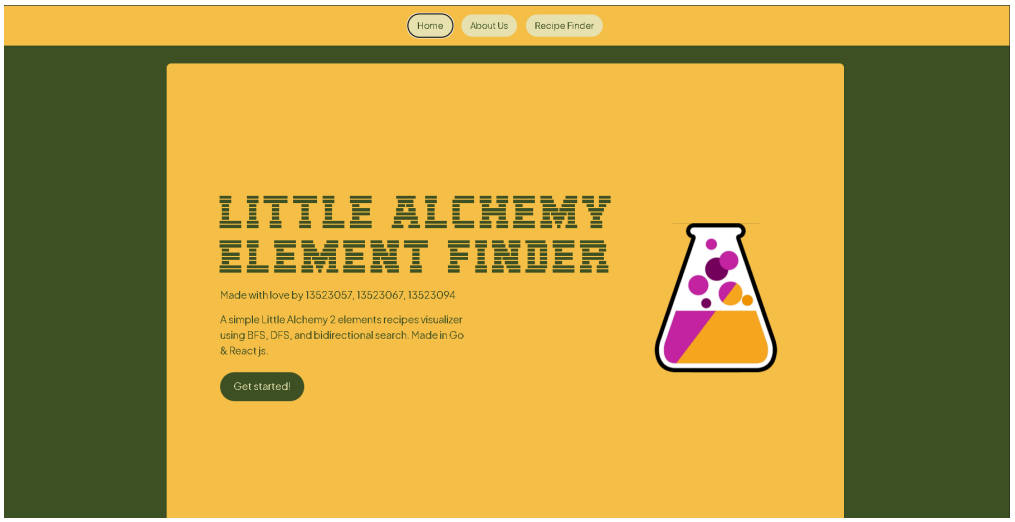
src\be\main.go

Mengatur sambungan *websocket* antara *frontend* dengan *backend*.

```
package main

import (
    "log"
    "net/http"

    "github.com/BP04/Tubes2_2Pendiklat1Coach/internal/handler"
    "github.com/BP04/Tubes2_2Pendiklat1Coach/internal/scraper"
    "github.com/BP04/Tubes2_2Pendiklat1Coach/internal/tools"
)

func main() {
    scraper.Scrape()
    tools.ParseJSON()
    tools.BuildGraph()

    http.HandleFunc("/ws", handler.WebSocketHandler)
    http.HandleFunc("/elements", handler.GetElements)

    log.Println("Server started on :8080")
    err := http.ListenAndServe(":8080", nil)
    if err ≠ nil {
        log.Fatal("Error starting server:", err)
    }
}
```

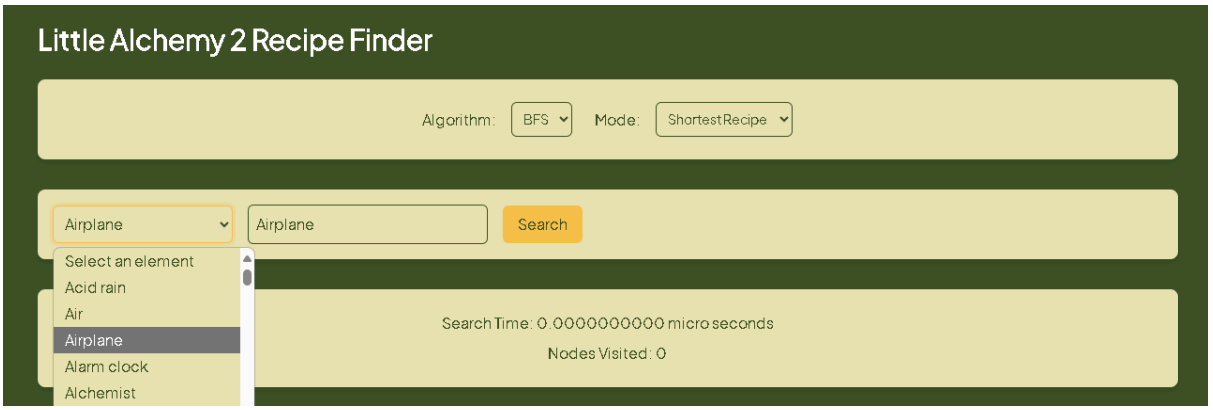
4.2. Tata Cara Penggunaan Aplikasi

Mula-mula pengguna akan tiba di *landing page*. Cukup tekan tombol "Get Started!" untuk berpindah ke laman program utama.

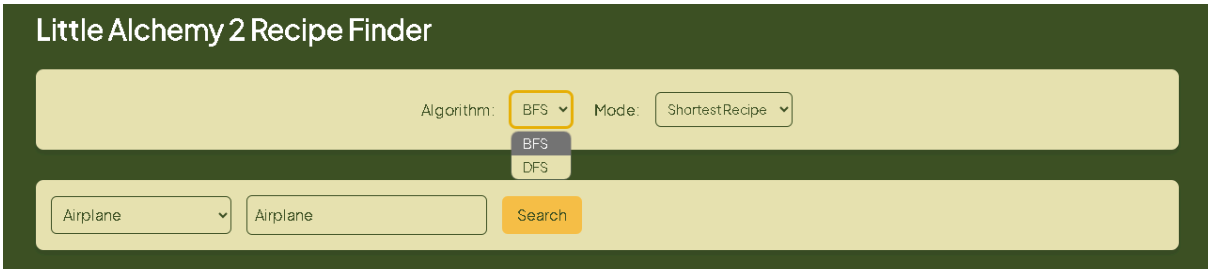


Di sini, pengguna dapat:

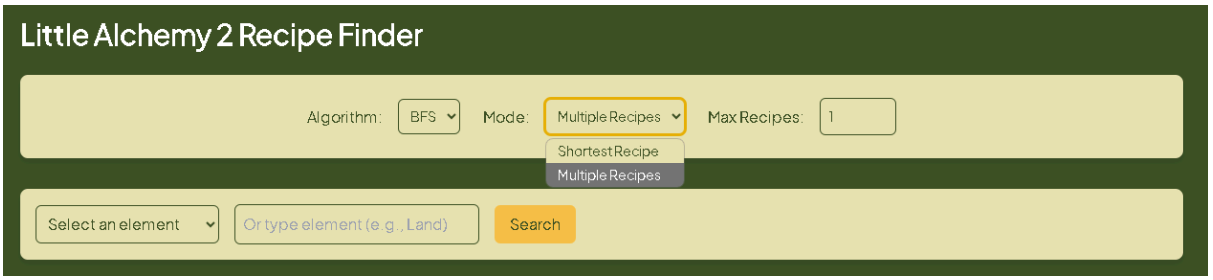
1. memasukkan elemen yang ingin dicari pada kolom pencarian (*search bar*) atau *dropdown* yang tersedia;



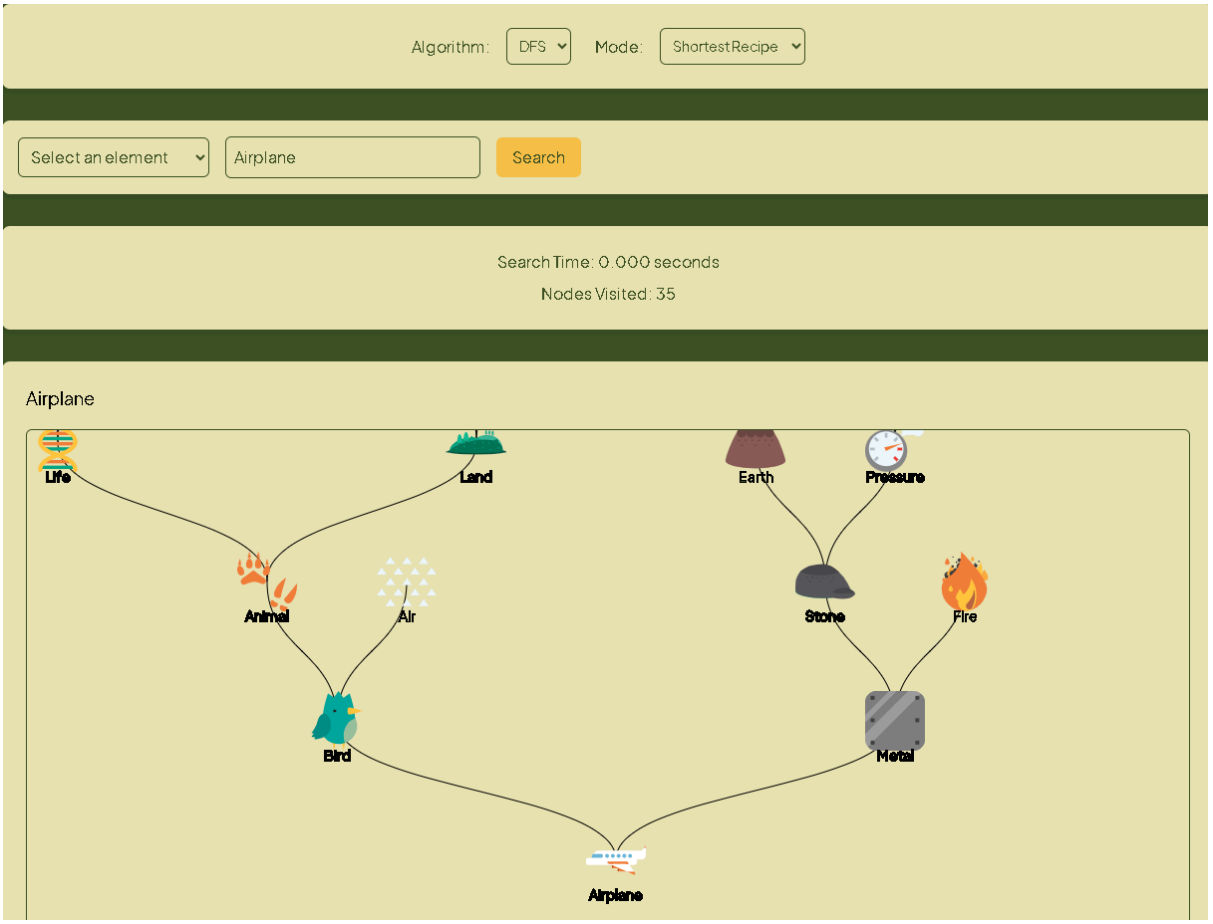
2. mengatur algoritma pencarian (DFS/BFS) melalui *dropdown*;



3. memilih mode *shortest recipe* atau *multiple recipes* beserta masukan jumlah *recipes* maksimum yang ditampilkan.



4. Melakukan pencarian *recipe(s)* berdasarkan kueri yang diberikan.



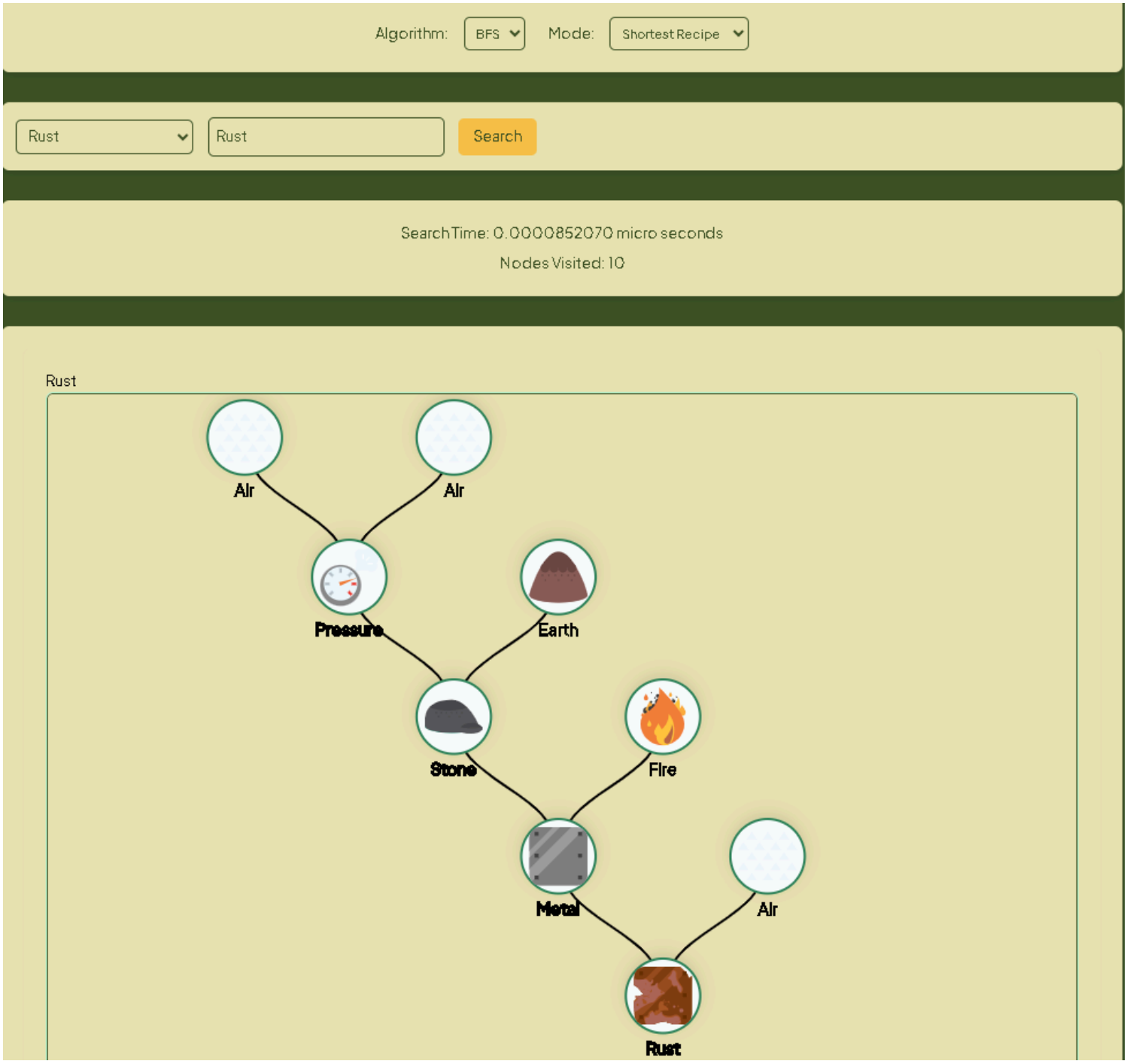
Recipe dari elemen yang dicari kemudian akan ditampilkan sebagai *card* di layar beserta keterangan durasi pencarian dan banyaknya simpul yang diperiksa.

4.3. Pengujian

Sangkalan: terkadang *search time* (durasi pencarian) memang menampilkan 0,000 *seconds* karena algoritma kami memang sebagus itu (kami menggunakan *caching*).

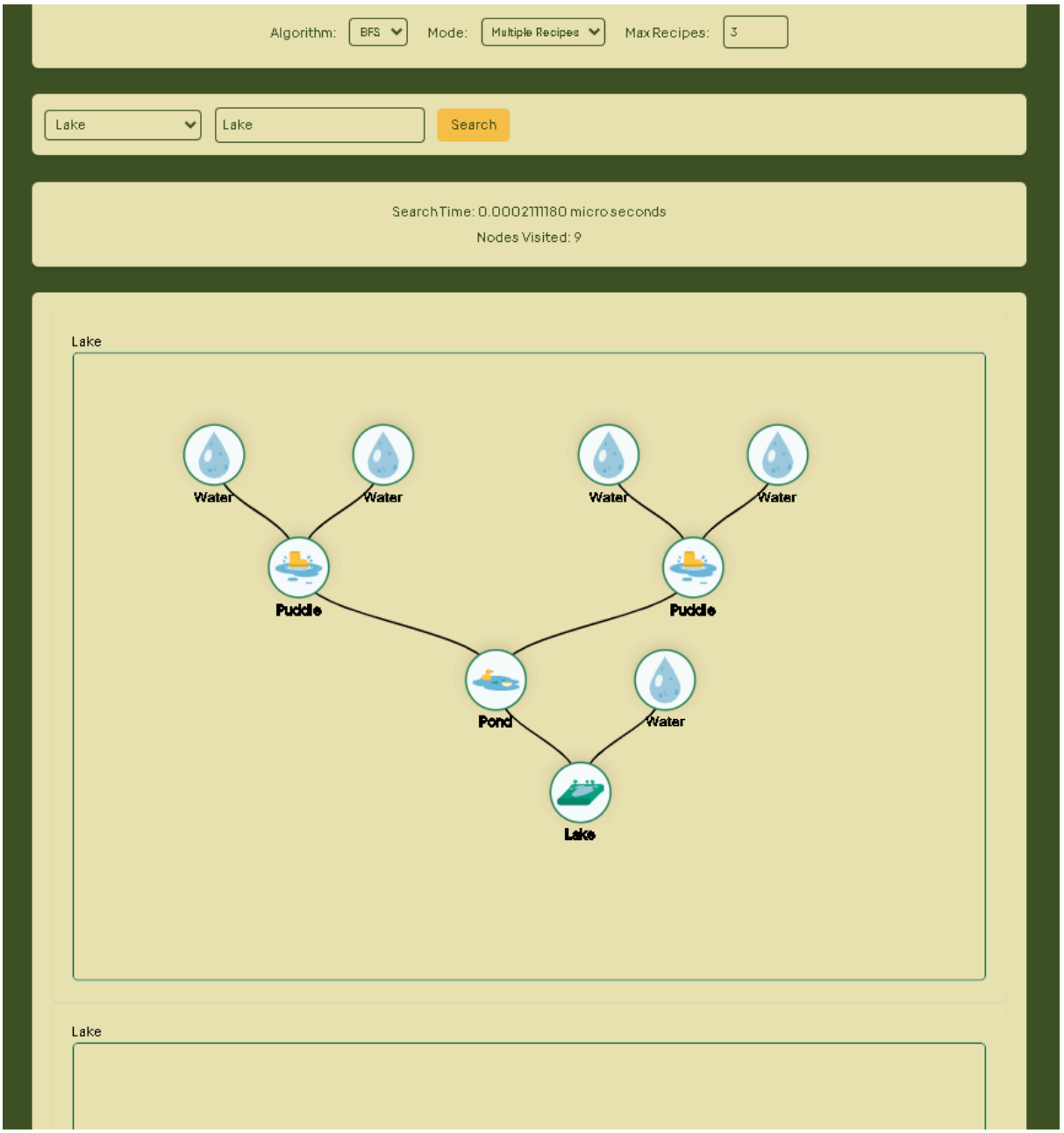
4.3.1. Kasus Uji 1: BFS, *single recipe*

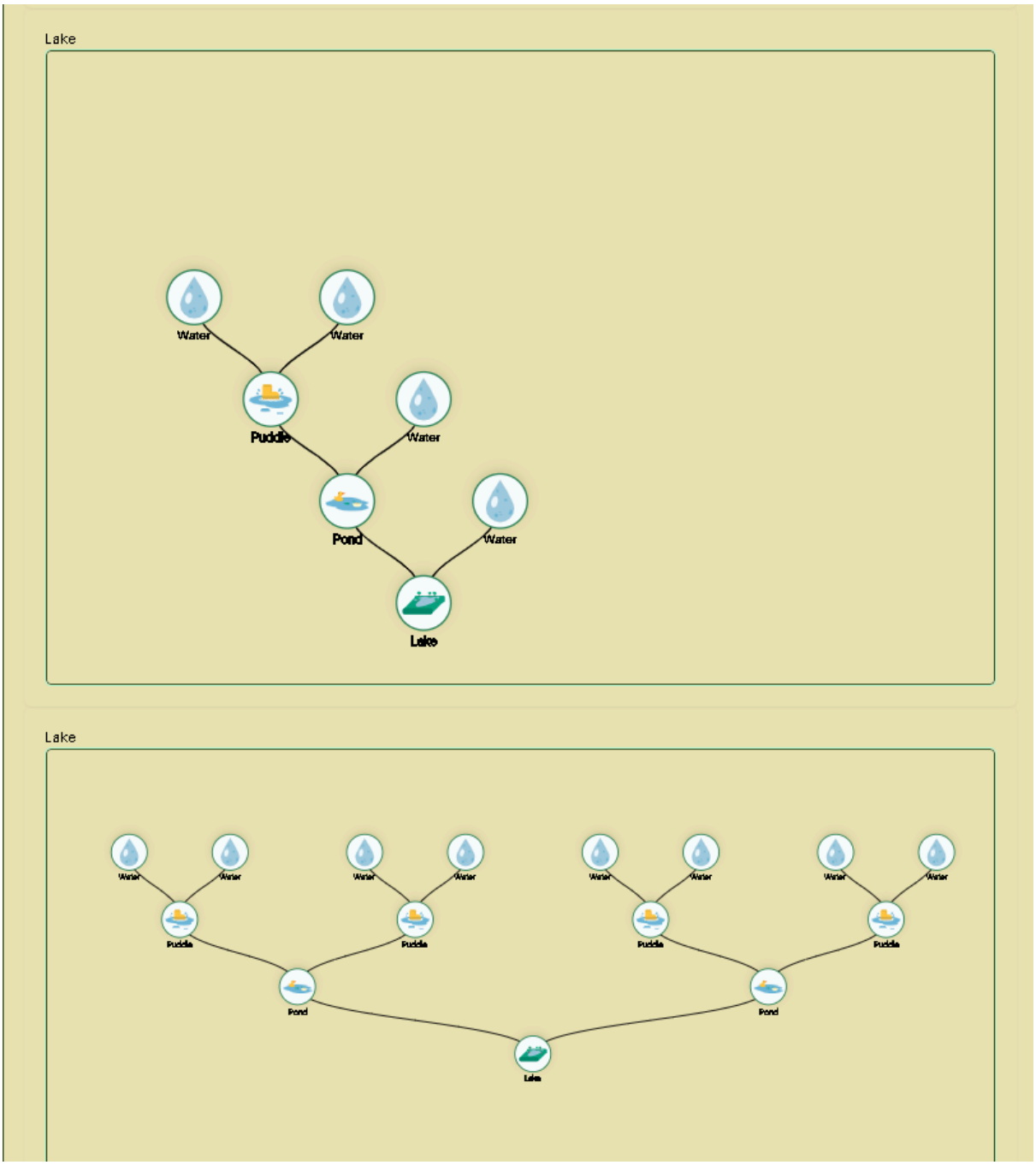
Dengan elemen kueri "Rust", didapatkan hasil sebagai berikut.



4.3.2. Kasus Uji 2: BFS, *multiple recipes* (3 *recipes*)

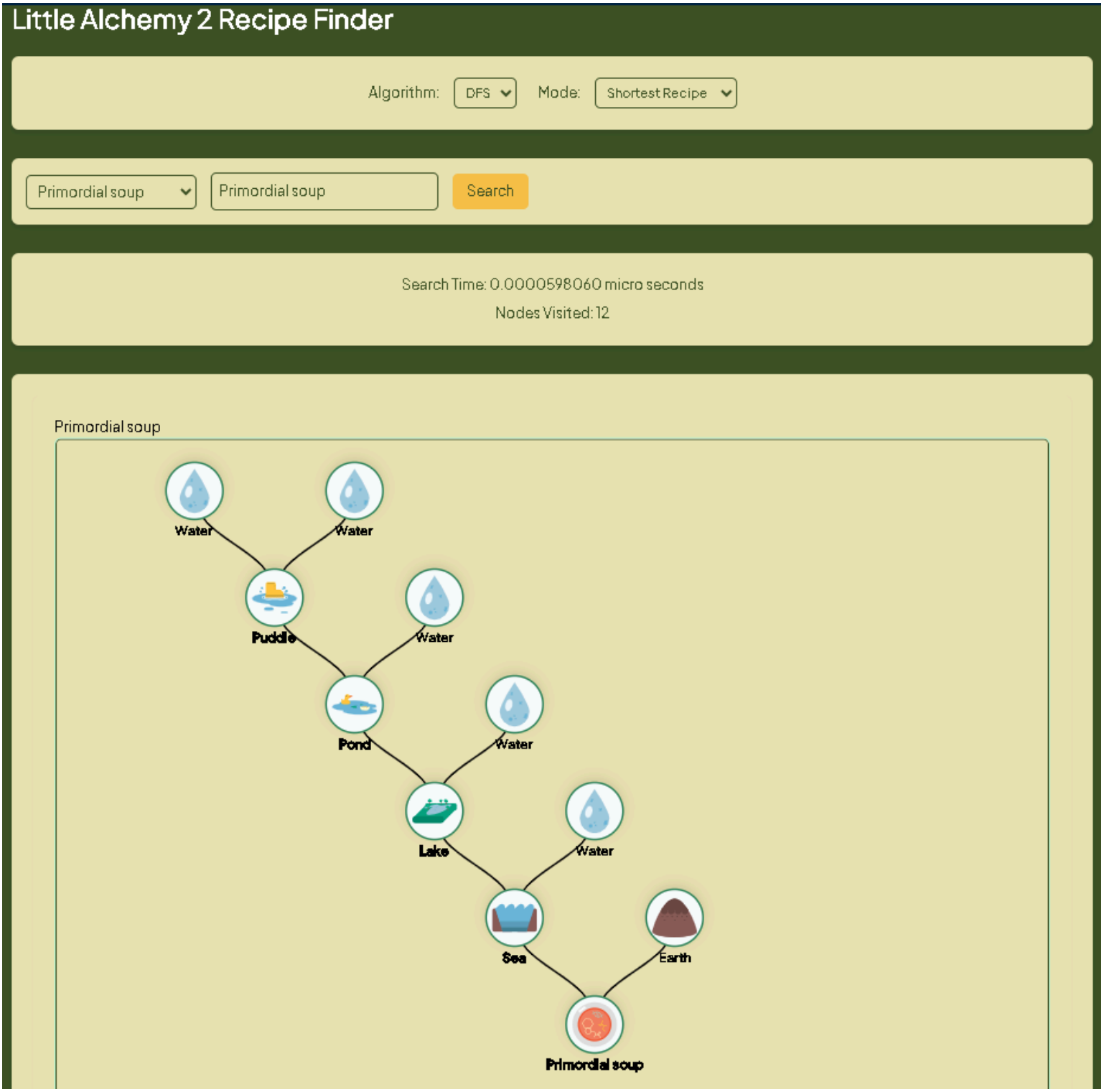
Dengan elemen kueri “Lake”, didapatkan hasil sebagai berikut.




Lake
Water
Water
Puddle
Water
Pond
Water
Lake
Lake

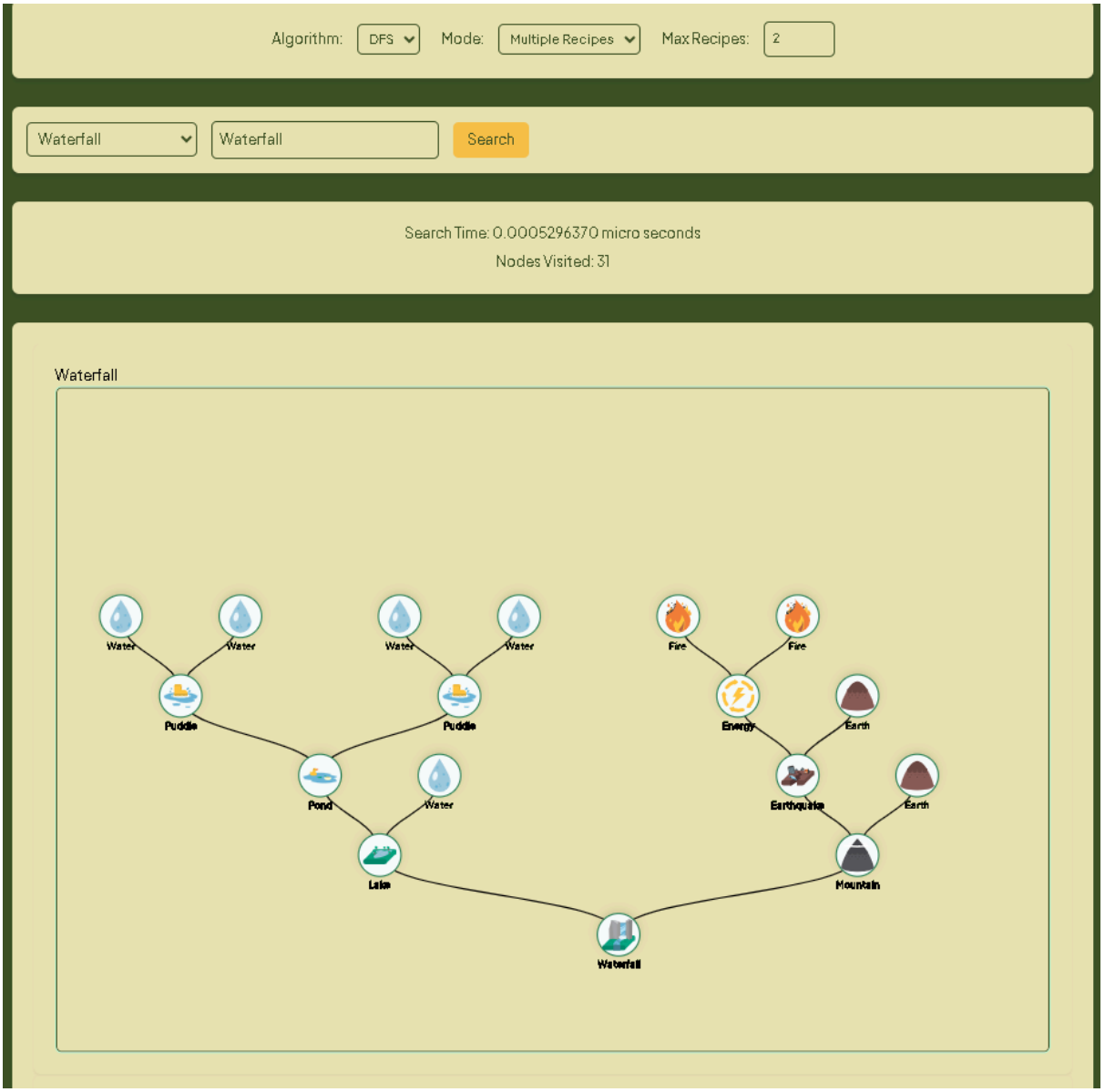
4.3.3. Kasus Uji 3: DFS, *single recipe*

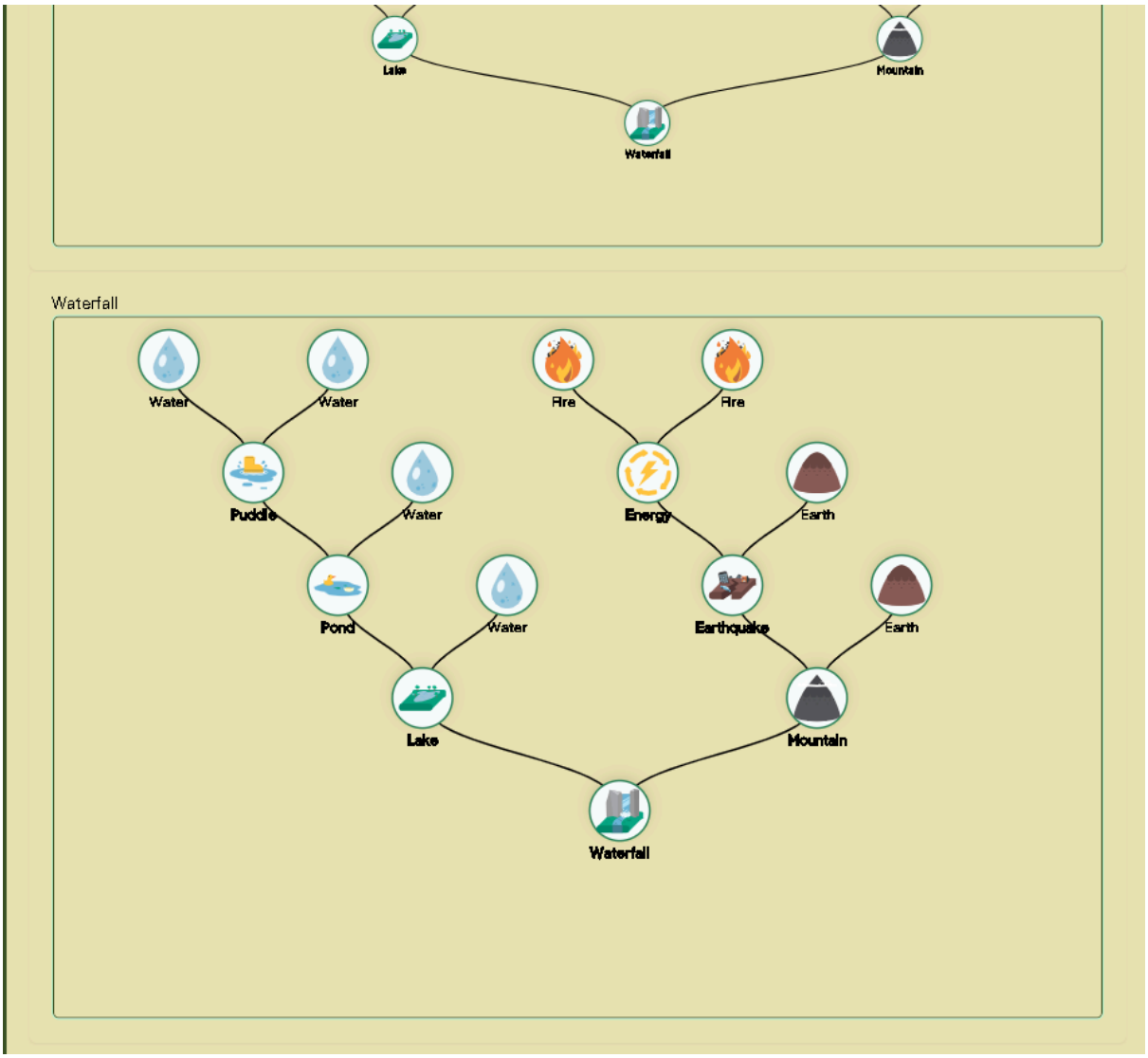
Dengan elemen kueri “Primordial soup”, didapatkan hasil sebagai berikut.



4.3.4. Kasus Uji 4: DFS, *multiple recipes* (2 *recipes*)

Dengan elemen kueri "Waterfall", didapatkan hasil sebagai berikut.





4.4. Analisis

Terdapat variasi hasil pada pencarian yang dilakukan. Algoritma BFS kami menggunakan pendekatan iteratif, sementara algoritma DFS-nya menggunakan pendekatan rekursif. Secara umum, algoritma DFS terhitung lebih cepat dibandingkan algoritma BFS. Untuk bisa mencari tree, BFS bukan hanya bergerak dari *source* ke *leaf* tetap juga *leaf* ke *source* lagi sehingga setiap sisi dilalui 2 kali sama seperti DFS. Selanjutnya overhead untuk queue terhitung lebih lambat dibandingkan untuk rekursi karena ukuran graf kecil. Terakhir, jumlah node yang dikunjungi relatif sama karena *traversal* dilakukan pada graf yang sama. Faktor-faktor ini menyebabkan algoritma DFS lebih cepat dari BFS khusus untuk permasalahan ini.

5. Penutup

5.1. Kesimpulan

Algoritma DFS dan BFS adalah algoritma pencarian yang sangat fleksibel dan dapat dimodifikasi untuk menyelesaikan berbagai persoalan. Dalam permainan Little Alchemy 2, DFS dan BFS dapat digunakan untuk mencari pohon terkecil pembentuk suatu elemen dan juga mencari pohon-pohon lain yang dapat membentuk elemen tersebut.

Untuk tugas ini digunakan teknologi react.js untuk tampilan depan serta bahasa Golang untuk melakukan pencarian di belakang layar. Untuk interaksi antara *front-end* dan *back-end* digunakan teknologi websocket.

5.2. Saran

Benedict Presley
Tugas besar ini jelek dan persiapannya tidak profesional. Saya mengerti sebagian besar ide tugas besar ini adalah ide asisten karena berpikir bahwa topiknya cocok. To a certain extent, benar topiknya cocok dan secara premis saya akui bagus, namun ada banyak kesalahan berpikir asisten dalam persiapan tugas besar ini sehingga membuat tugas besar ini menjadi tidak menyenangkan. Misal masalah looping yang sudah dibahas sejak awal namun asisten masih mengira ada cara untuk menyelesaikannya. Saya pikir asisten melakukan vibe coding untuk prototyping tugas besar. Jika tidak melakukan vibe coding, maka menurut saya ini adalah skill issue yang besar dari sisi asisten. Dalam menjawab QnA, menurut saya asisten sangat tidak peduli. Banyak jawaban yang kurang jelas atau bahkan tidak sesuai konteks pertanyaannya. Misalnya ketika saya bertanya tentang infinite loop yang membuat infinite recipe namun dijawab “Silahkan buat N recipe unik”. Silahkan memberi tahu saya bila saya yang bodoh dalam kasus ini. Saya tidak mengerjakan bonus bidirectional bukan karena tidak sempat atau tidak mampu namun saya menolak mengerjakan suatu tugas yang definisinya saja salah. Bidirectional searching adalah suatu bentuk search untuk mencari path dari a ke b dan searching dilakukan dari a dan b bersamaan (sesuai dengan link gfg yang dilampirkan di spesifikasi). Dalam tugas ini, yang kita cari adalah tree, bukan path. Hal ini saja sudah jelas menyalahi definisi bidirectional searching yang diberikan. Dan juga untuk bisa melakukan bidirectional searching diperlukan source dan end yang jelas. Dalam kasus ini, untuk bisa menemukan end, diperlukan traversal total dari source. Bidirectional search menjadi 2 kali lebih lambat daripada search biasa karena ujung-ujungnya melakukan search yang sama 2 kali.

Faqih Muhammad Syuhada

Saya setuju dengan Bene. Tugas besar ini simple tapi kurang jelas. Penjelasan yang diberikan kurang, seperti ada hal baru di QNA tetapi tidak di perbarui di spec dan beberapa hal lainnya. Saya tidak mengetahui apa maksud asisten dengan spek yang setengah-setengah, apakah memang harus berkreatifitas? Sehingga banyak perbedaan antar kelompok. Seharusnya spek yang di berikan jangan di revisi di dekat deadline walaupun diberikan waktu tambahan, hal tersebut tetap kurang profesional.

Zulfaqqar Nayaka Athadiansyah
Kalau revisi spek jangan mepet *deadline* bang T_T

5.3. Refleksi

Faqih Muhammad Syuhada
Tubes ini santai walau jika ingin *perfect* harusnya spek juga *perfect*. Sebagai yang memegang FE dan menyesuaikan isi dengan BE, ini terlalu santai untuk saya. Mau dengan vibe coding ataupun tidak ini mudah, harusnya siapapun malu jika vibe coding menjadi dasar FE ini. *Very happy to work with this team* yang *LGTM*. Kelompok yang keren karena menyatukan 3 pihak yang berbeda dengan keunikannya masing-masing. Walau di akhir waktu banyak bonus yang ingin dikejar tidak kekejar tetapi sudah puas dengan effort yang kecil hasilnya memuaskan. Terimakasih banyak untuk Bene dan Nayaka yang fleksibel.

Benedict Presley
Tugas besar yang sangat santai. Perlu berpikir cukup lama untuk menemukan struktur searching yang rapi dan implementasi yang bagus. 0 vibe coding karena AI/LLM tidak lebih ahli dalam hal algoritma dibanding aku, juga memang tidak diperbolehkan menggunakan AI/LLM. Very happy to work with an amazing team. Ini kelompok yang sangat tidak biasa, tapi aku senang uda ngajak kalian. Agak terlalu santai, jadi cenderung meremehkan dan chaos tipis pas di ujung.

Zulfaqqar Nayaka Athadiansyah
Overall tubesnya *chill*, sih. Sebagian besar faktornya karena temen sekelompokku gacor-gacor semua. Di tubes ini, aku nge-*handle scraper*, integrasi *frontend* dan *backend* lewat *websocket*, ngerjain sebagian besar dari laporan ini, nge-*deploy*, dan disuruh-suruh sama Faqih dan Ben (jujur ini emang karena mereka lebih jago, jadi aku nurut-nurut aja). Aku seneng bisa eksplorasi hal-hal baru kaya *websocket* dan serba-serbi DevOps. *It was fun working with you guys*! :D

Lampiran

Tautan repositori GitHub: https://github.com/BP04/Tubes2_2Pendiklat1Coach.git
Tautan aplikasi yang sudah di-*deploy*: https://frontend-service-966378866548.asia-southeast1.run.app
Tautan video bonus: https://youtu.be/7Q0jCTxzNak

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	☑	☐
2	Aplikasi dapat memperoleh data *recipe* melalui *scraping*.	☑	☐
3	Algoritma DFS dan BFS 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.	☑	☐

Tabel 1. Daftar pengerjaan tugas besar 2

Daftar Pustaka

S. Halim dan F. Halim, *Competitive Programming 3: The New Lower Bound of Programming Contests; Handbook for ACM ICPC and IOI Contestants*. 2013.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.