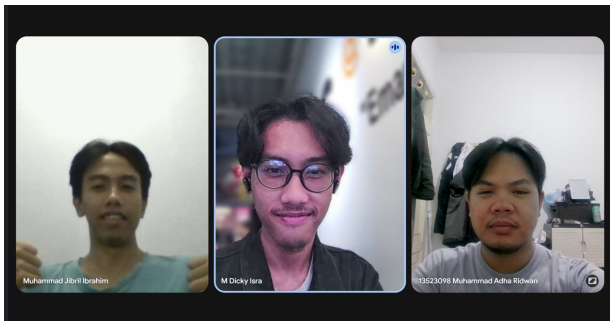
Laporan Tugas Besar 2

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

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2 (Genap) Tahun Akademik 2024/2025



Kelompok 58 (AshtonHallMorningRoutine)

Muhammad Dicky Isra	13523075
Muhammad Jibril Ibrahim	13523085
Muhammad Adha Ridwan	13523098

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

Daftar Isi

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. Permainan ini tersedia di web browser, Android atau iOS

Gambar 1: Little Alchemy 2

Komponen penting dari Little Alchemy 2 terdiri dari:

1. **Elemen Dasar**

   Dalam permainan Little Alchemy 2, terdapat 4 elemen dasar yang tersedia yaitu *water*, *fire*, *earth*, dan *air*, 4 elemen dasar tersebut nanti akan di-*combine* menjadi elemen turunan yang berjumlah 720 elemen.

Gambar 2: Elemen Dasar

2. Element 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.

Gambar 3: Elemen Turunan

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.

Gambar 4: Contoh *Combine Mechanism*

Pada Tugas Besar 2 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk membuat aplikasi web untuk menemukan resep elemen Little Alchemy 2.

Bab II
Landasan Teori

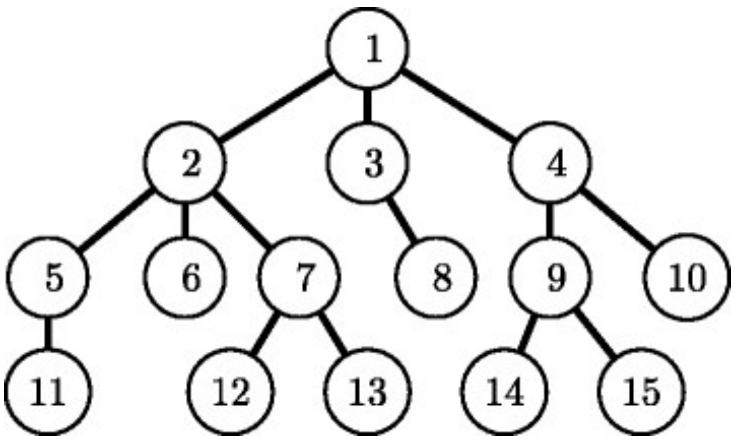
2.1. Graph Traversal

Graph Traversal adalah proses mengunjungi setiap simpul dalam sebuah graf dengan cara yang sistematis. Proses ini bertujuan untuk memeriksa dan/atau memperbarui setiap simpul dalam graf tersebut, dan dapat digunakan untuk berbagai tujuan seperti mencari simpul tertentu, menghitung panjang jalur terpendek, atau melakukan operasi lain pada graf.

Jika kita menggunakan graf sebagai representasi dari sebuah persoalan, maka *graph traversal* bisa disebut juga sebagai suatu pencarian solusi. Algoritma pada *graph traversal* terdiri dari berbagai algoritma, dua diantaranya adalah Breadth First Search (BFS) dan Depth First Search (DFS).

2.2. Breadth First Search

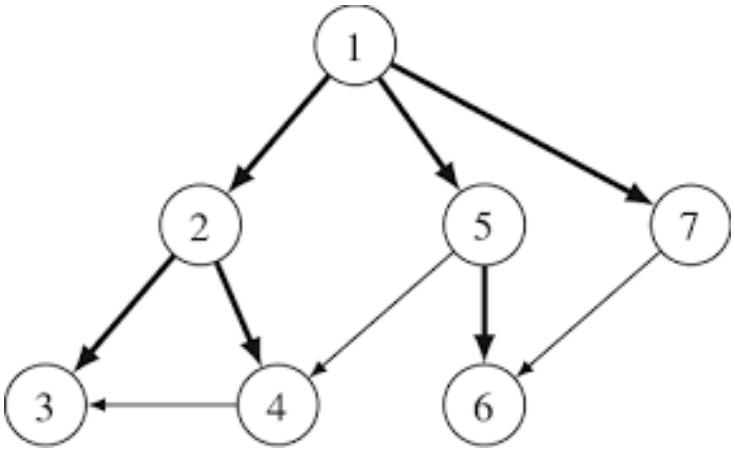
Breadth First Search (BFS) adalah algoritma penulusuran graf tanpa informasi (*blind search/uninformed*) yang bekerja dengan menelusuri atau mengunjungi semua simpul yang terhubung pada suatu kedalaman tertentu sebelum bergerak ke kedalaman selanjutnya. Algoritma BFS menggunakan struktur data *queue* untuk mengimplementasi urutan kunjungan simpul.



Gambar 5: Contoh Urutan pada BFS

2.3. Depth First Search

Depth First Search (DFS) adalah algoritma penulusuran graf tanpa informasi (*blind search/uninformed*) yang bekerja dengan menelusuri atau mengunjungi sedalam mungkin pada cabang pertama sebelum kembali dan mecoba cabang lainnya. Algoritma DFS menggunakan struktur data stack untuk mengimplementasi urutan kunjungan simpul.



Gambar 6: Contoh Urutan pada DFS

2.4. Penjelasan Singkat Aplikasi Web

2.4.1 Frontend

Frontend aplikasi web ini dikembangkan menggunakan Next.js dengan TypeScript untuk memastikan keandalan kode melalui pengecekan tipe statis. Node.js berperan sebagai runtime environment yang mendukung proses pembangunan (*build*) dan *server development*. Antarmuka pengguna dirancang interaktif, memungkinkan pengguna memilih elemen target, algoritma pencarian (BFS atau DFS), serta mode pencarian resep (*single* untuk jalur terpendek atau *multiple* untuk variasi resep). Visualisasi hierarki kombinasi elemen diimplementasikan dengan library React Flow, yang menampilkan struktur resep dalam bentuk pohon grafis. Statistik pencarian, seperti waktu eksekusi dan jumlah node yang dikunjungi, ditampilkan secara jelas untuk analisis pengguna.

2.4.2 Backend

Backend dibangun menggunakan Golang (Go) dengan *framework* Gin untuk mengoptimalkan performa dengan memanfaatkan kemampuan konkurensi yang kuat, terutama dalam menangani pencarian multiple recipe secara paralel menggunakan *multithreading*. Proses *web scraping data* elemen dan resep dari Fandom Little Alchemy 2 Wiki dilakukan dengan GoQuery, data tersebut disimpan ke dalam format JSON. Selain itu, backend menjalankan logika algoritma BFS dan DFS serta, sekaligus menjaga responsivitas aplikasi secara keseluruhan.

Bab III
Analisis Pemecahan Masalah

3.1. Langkah-langkah Pemecahan Masalah

3.1.1 Identifikasi Masalah

Dalam penyelesaian Recipe Finder pada Little Alchemy 2, ada beberapa masalah yang bisa kita identifikasi.

- Mengumpulkan data-data resep elemen dari Little Alchemy 2.
- Mencari resep elemen yang dinginkan dengan menggunakan algoritma DFS dan BFS.
- Memvisualisasikan hasil pencarian sebelumnya dengan menggunakan representasi *tree*.

3.1.2 Pengumpulan Data

Pengumpulan data berupa resep elemen dari Little Alchemy 2 dapat dilakukan dengan memanfaatkan *library* GoQuery yang dieksekusi di *backend*. Data hasil *scrapping* kemudian disimpan secara lokal di *backend* menggunakan format JSON.

3.1.3 Pembuatan Algoritma

Pembuatan dimulai dengan pembuatan *parsing* data hasil *scrapping* menjadi data yang bisa digunakan pada algoritma penelusuran graf yaitu sebuah *map* dimana *key* adalah elemen dan *value* berupa larik dari resep elemen tersebut.

Algoritma BFS dan DFS dibuat dengan pendekatan *bottom-up* untuk memastikan semua elemen dibuat dari empat elemen dasar menggunakan *queue* untuk BFS dan *stack* untuk DFS. Keduanya dilakukan secara *iterative* untuk menghindari terjadinya *stack overflow* yang dapat diakibatkan oleh pemanggilan *recursive* secara berlebihan.

Hasil dari penelusuran BFS dan DFS akan disimpan dalam *memory* dengan suatu tipe data *map*. Kemudian, *map* tersebut akan direkonstruksi ke dalam bentuk *tree* dengan *function helper* secara *recursive* dari elemen target sebagai *root* dari *tree* tersebut. *Tree* hasil dari rekonstruksi tersebutlah yang menjadi jawaban langkah-langkah resep pembentuk elemen target.

Selain itu, dilakukan optimisasi secara *heuristic* yaitu penerapan aturan bahwa elemen pembentuk dari suatu elemen harus memiliki *tier* atau tingkat yang lebih rendah dari elemen hasilnya dan penggunaan *multithreading* untuk meningkatkan performa terutama ketika melakukan penelusuran dan rekonstruksi untuk *multiple recipe*

3.2. Pemetaan Masalah

Permasalahan pencarian resep dalam Little Alchemy 2 dipetakan ke dalam struktur graf, di mana setiap elemen direpresentasikan sebagai *node*, sementara kombinasi dua elemen yang menghasilkan elemen baru membentuk *edge*. Sebagai contoh, kombinasi Air (udara) dan Earth (tanah) menghasilkan Mud (lumpur), sehingga Air dan Earth menjadi node yang terhubung ke Mud melalui edge. Representasi ini memungkinkan pencarian resep dilakukan dengan menjelajahi hubungan hierarkis antar elemen.

Breadth First Search (BFS) bekerja dengan mengeksplorasi graf level per level menggunakan struktur data *queue*, memastikan elemen yang lebih dekat ke elemen dasar diprioritaskan. Dengan menjamin jalur terpendek, BFS ideal untuk kasus di mana pengguna ingin memperoleh kombinasi minimal langkah, seperti menemukan jalur terpendek untuk elemen yang memiliki resep yang valid.

Di sisi lain, Depth First Search (DFS) bekerja dengan mendalami sedalam mungkin cabang pertama yang dikunjungi dengan memanfaatkan struktur data *stack*, menjelajahi satu cabang graf hingga mencapai elemen target sebelum beralih ke cabang lain. DFS kurang ideal untuk memperoleh kombinasi minimal langkah, akan tetapi ideal untuk diterapkan *multiple recipe*

3.3. Fitur dan Arsitektur Aplikasi Web

3.3.1 Fitur

Aplikasi web pencari resep Little Alchemy 2 ini memiliki beberapa fitur fungsional yaitu:

- Pemilihan Algoritma (BFS/DFS)

  Pengguna dapat memilih algoritma apa yang akan digunakan dalam pencarian resepnya.

- Toggle Mode Resep

  Pengguna dapat memilih untuk mencari *multiple recipe* atau *shortest recipe*, jika pengguna memilih *multiple recipe* pengguna bisa mengisi form Maximum Recipe untuk menentukan jumlah maksimum resep yang akan didapatkan. Secara *default shortest path* hanya akan mengembalikan satu resep.

- Visualisasi Pohon Resep

  Resep divisualisasikan sebagai *tree* hierarkis dengan elemen dasar sebagai *leaf* dan elemen target sebagai *root*. Setiap kombinasi ditampilkan sebagai *node* yang terhubung. *Library* React Flow untuk visualisasi *tree* yang interaktif.

- Statistik Pencarian

  Waktu eksekusi pencarian (dalam milidetik) dan jumlah *node* yang dikunjungi ditampilkan untuk analisis performa algoritma.

- Multithreading

  Pencarian resep dalam mode *multiple* dioptimasi dengan Goroutine di *backend* untuk menjalankan beberapa *thread* pencarian secara paralel.

3.3.2 Arsitektur Aplikasi Web

Aplikasi web ini menggunakan arsitektur microservices dengan pendekatan Decoupled (Loosely Coupled) Architecture, di mana Frontend dan Backend dipisahkan secara independen. Pendekatan ini mengimplementasikan pola Headless Architecture, memungkinkan *frontend* dan *backend* berkomunikasi melalui API Call.

- Frontend

  Frontend menggunakan *framework* NextJS dengan bahasa *Typescript* dan *library* React sebagai fondasi utama pengembangan web aplikasi dengan dukungan Node.js untuk manajemen *enviroment* dan *dependency* web.

  Modul React Flow digunakan untuk komponen graf dalam *tree* untuk menvisualisasikan langkah-langkah penyusunan resep elemen. Modul React Flow dipilih sebagai komponen graf untuk memudahkan pengembangan aplikasi web dengan fungsionalitas *layouting* dan utilitas yang ditawarkan.

- Backend

  Backend mengunakan bahasa Golang dengan *framework* Gin untuk membangun API yang ringan dan cepat serta memanfaatkan *multithreading* untuk meningkatkan performa proses.

3.4. Ilustrasi Kasus

Pada bagian ini, akan dijelaskan ilustrasi kasus pencarian resep untuk tiga elemen berbeda dalam game Little Alchemy 2 menggunakan algoritma BFS dan DFS dengan pendekatan bottom-up. Pendekatan bottom-up berarti pencarian dimulai dari elemen dasar menuju elemen target.

3.4.1 Pencarian Resep Brick dengan BFS

Untuk mencari resep elemen Brick menggunakan algoritma BFS dengan pendekatan bottom-up:

1. **Elemen Target:** Brick.
2. **Resep yang mungkin dari Brick**.
   (a) Mud + Fire.
   (b) Mud + Sun.
   (c) Clay + Fire.
   (d) Clay + Sun.
   (e) Clay + Stone.
3. **Eksplorasi BFS**
   Mulai dari empat elemen dasar: Earth, Fire,Water, Air. Kemudian akan mencari secara per level semua resep dimana salah satu elemen pembentuknya adalah *head queue* dan elemen lainnya bisa kita bentuk.
   - Level 0: Earth, Fire, Water, Air.
   - Level 1: Mud, Mist, Steam, Energy, etc.
   - Level 2: Stone, Clay, Brick(Found).

4. **Hasil:** BFS akan menemukan recipe Mud + Fire terlebih dahulu karena berada pada level yang lebih rendah. Dikarenakan BFS menelusuri berdasarkan level satu per satu, hasil dari BFS dapat dijamin merupakan *shortest path* karena *tree* ini adalah *unweighted graph*.

3.4.2 Pencarian Resep Brick dengan DFS

Untuk mencari resep elemen Brick menggunakan algoritma DFS dengan pendekatan bottom-up:

1. **Elemen Target:** Brick.
2. **Resep yang mungkin dari Brick**.
   (a) Mud + Fire.
   (b) Mud + Sun.
   (c) Clay + Fire.
   (d) Clay + Sun.
   (e) Clay + Stone.
3. **Eksplorasi DFS**
   Mulai dari empat elemen dasar: Earth, Fire,Water, Air. Kemudian akan mencari ke suatu cabang hingga menemukan elemen target atau sudah buntu, kemudian melakukan *backtracking* untuk mencoba cabang lainnya.
   - Mulai dari Earth, kemudian menambahkan elemen yang bisa dibentuk ke dalam *stack*.
   - Mengunjungi Mud, kemudian menambahkan elemen yang bisa dibentuk ke dalam *stack*.
   - Mengunjugi Brick, karena sudah pada elemen target, eksplorasi selesai.
4. **Hasil:** Dalam kasus Brick ini DFS berhasil menemukan resep Brick lebih cepat dibandingkan BFS, akan tetapi untuk kasus dimana tier elemen tinggi, DFS tidak bisa menjamin bahwa resep yang ditemukannya adalah resep dengan langkah terpendek.

Bab IV
Implementasi dan Pengujian

4.1. Struktur Data

4.1.1 TreeNode

```
type TreeNode struct {
  Element    string
  Children   []*TreeNode
  RecipeStep *RecipeStep
}
```

4.1.2 RecipeStep

```
type RecipeStep struct {
  Ingredient1 string
  Ingredient2 string
  Result      string
}
```

4.1.3 JSONResponse

```
type JSONResponse struct {
  Data         *JSONRecipeNode `json:"data"`
  Errors       []string    `json:"errors"`
  Time         int64       `json:"time"`          // milliseconds
  NodeCount    int         `json:"nodeCount"`     // nodes visited
  RecipeFound  int         `json:"recipeFound"`   // recipes found
}
```

4.1.4 JSONResponseNode

```
type JSONRecipeNode struct {
  Name     string              `json:"name"`
  Recipes  [][2]*JSONRecipeNode   `json:"recipes,omitempty"`
}
```

4.2. Implementasi Program

4.2.1 Fungsi Web Scrapper

```
package utils
import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
  "os"

  "github.com/PuerkitoBio/goquery"
)

type Element struct {
  Name string `json:"name"`
  Tier int    `json:"tier"`
}

type ElementRecipe struct {
  Tier   int      `json:"tier"`
  Result string   `json:"result"`
  Recipe []string `json:"recipe"`
}

func InitializeData() {
  url := "https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)"
  url2 := "https://little-alchemy.fandom.com/wiki/Elements_(Myths_and_Monsters)"

  // Get filters (myth & monsters recipes)
  filters := getFilters(url2)
  filters = append(filters, "Time")

  // Get recipes minus filters
  recipes, elements := getRecipesAndElements(url, filters)

  // Write recipes to file
  writeToFile("recipes.json", recipes)

  // Write elements to file
  writeToFile("elements.json", elements)

  fmt.Printf("Amount of recipes found: %d\n", len(recipes))
  fmt.Printf("Amount of elements found: %d\n", len(elements))
}

func getFilters(url string) []string {
  res, err := http.Get(url)
  if err != nil {
    log.Fatal(err)
  }
  defer res.Body.Close()

  if res.StatusCode != 200 {
    log.Fatalf("status code error: %d %s", res.StatusCode, res.Status)
  }

  doc, err := goquery.NewDocumentFromReader(res.Body)
  if err != nil {
    log.Fatal(err)
  }

  var filters []string

  doc.Find("table.list-table").Each(func(i int, table *goquery.Selection) {
    table.Find("tr").Each(func(j int, tr *goquery.Selection) {
      if j == 0 {
        return // skip header row
      }

      tds := tr.Find("td")
      if tds.Length() == 0 {
        return
      }

      result := tds.Eq(0).Find("a[href^='/wiki/']").First().Text()
      filters = append(filters, result)
    })
  })

  return filters
}

func getRecipesAndElements(url string, filters []string) ([]ElementRecipe, []Element) {
  res, err := http.Get(url)
  if err != nil {
    log.Fatal(err)
  }
  defer res.Body.Close()

  if res.StatusCode != 200 {
    log.Fatalf("status code error: %d %s", res.StatusCode, res.Status)
  }

  doc, err := goquery.NewDocumentFromReader(res.Body)
  if err != nil {
    log.Fatal(err)
  }

  var recipes []ElementRecipe
  var elements []Element
  tier := 0
```

```
  doc.Find("table.list-table").Each(func(i int, table *goquery.Selection) {
    table.Find("tr").Each(func(j int, tr *goquery.Selection) {
      if j == 0 {
        return // skip header row
      }

      tds := tr.Find("td")
      if tds.Length() == 0 {
        return
      }

      result := tds.Eq(0).Find("a[href^='/wiki/']").First().Text()

      // Get element name and tier only
      element := Element{
        Name: result,
        Tier: adjustTier(tier),
      }

      if !contains(filters, result) {
        elements = append(elements, element)
      }

      // Get recipes
      if tds.Eq(1).Find("ul").Length() > 0 {
        var currentRecipe ElementRecipe
        var recipeItems []string

        tds.Eq(1).Find("a[href^='/wiki/']").Each(func(k int, a *goquery.Selection) {
          item := a.Text()
          recipeItems = append(recipeItems, item)

          if k%2 == 1 {
            currentRecipe = ElementRecipe{
              Tier:   adjustTier(tier),
              Result: result,
              Recipe: recipeItems,
            }

            valid := true
            for _, filter := range filters {
              if filter == currentRecipe.Result || contains(currentRecipe.Recipe, filter) {
                valid = false
                break
              }
            }

            if valid {
              recipes = append(recipes, currentRecipe)
            }

            recipeItems = []string{}
          }
        })
      }
    })
    tier++
  })

  return recipes, elements
}

func adjustTier(tier int) int {
  if tier > 1 {
    return tier - 1
  }
  return tier
}

func contains(slice []string, item string) bool {
  for _, s := range slice {
    if s == item {
      return true
    }
  }
  return false
}

func writeToFile(filename string, data interface{}) {
  file, err := os.Create(filename)
  if err != nil {
    log.Fatal(err)
  }
  defer file.Close()

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

4.2.2 Fungsi Algoritma BFS

```
func BFS(target string, graph map[string][][2]string, tiers map[string]int, maxRecipes int) ([]RecipePath, int) {
  craftable := make(map[string]bool)
  visited := make(map[string]bool)
  recipeVariants := make(map[string][]RecipeStep)
  visitCount := 0
  // Initialize base elements
  for base := range baseElements {
    craftable[base] = true
    visited[base] = true
  }
  queue := make([]string, 0, len(baseElements))
  for base := range baseElements {
    queue = append(queue, base)
  }
  for len(queue) > 0 && len(recipeVariants[target]) < maxRecipes {
    current := queue[0]
    queue = queue[1:]
    visitCount++
    for ingredient := range craftable {
      possibleResults := findRecipes(current, ingredient, graph)
      for _, result := range possibleResults {
        resultTier := tiers[result]
        currentTier := tiers[current]
        ingredientTier := tiers[ingredient]
        if resultTier > currentTier && resultTier > ingredientTier {
          newRecipe := RecipeStep{
            Ingredient1: current,
            Ingredient2: ingredient,
            Result:      result,
          }
          if len(recipeVariants[result]) < maxRecipes {
            isDuplicate := false
            for _, existing := range recipeVariants[result] {
              if (existing.Ingredient1 == current && existing.Ingredient2 == ingredient) ||
                (existing.Ingredient1 == ingredient && existing.Ingredient2 == current) {
                isDuplicate = true
                break
              }
            }
            if !isDuplicate {
              recipeVariants[result] = append(recipeVariants[result], newRecipe)
            }
          }
          if !craftable[result] {
            craftable[result] = true
            visited[result] = true
            queue = append(queue, result)
          }
        }
      }
    }
  }
  if !craftable[target] {
    fmt.Printf("Cannot craft %s from base elements\n", target)
    return nil, visitCount
  }
  var allPaths []RecipePath
  processedCount := 0
  if maxRecipes > 1 {
    resultChan := make(chan RecipePath, maxRecipes)
    var wg sync.WaitGroup
    maxWorkers := 3
    sem := make(chan struct{}, maxWorkers)
    recipesToProcess := 0
    for range recipeVariants[target] {
      if processedCount >= maxRecipes {
        break
      }
      processedCount++
      recipesToProcess++
    }
    processedCount = 0
    for _, recipeVariant := range recipeVariants[target] {
      if processedCount >= maxRecipes {
        break
      }
      processedCount++
      wg.Add(1)
      sem <- struct{}{}
      go func(recipe RecipeStep) {
        defer wg.Done()
        defer func() { <-sem }()
        recipeMap := buildIterativeRecipeMap(recipe, recipeVariants)
        craftingPath := make([]RecipeStep, 0, len(recipeMap))
        for _, step := range recipeMap {
          craftingPath = append(craftingPath, step)
        }
        treeRoot := buildCraftingTreeFromMap(target, recipeMap, make(map[string]bool))
        resultChan <- RecipePath{craftingPath, treeRoot}
      }(recipeVariant)
    }
    go func() {
      wg.Wait()
      close(resultChan)
    }()
    collectedCount := 0
```

```
    for path := range resultChan {
      allPaths = append(allPaths, path)
      collectedCount++
      if collectedCount >= recipesToProcess {
        break
      }
    }
  } else {
    for _, recipeVariant := range recipeVariants[target] {
      if processedCount >= maxRecipes {
        break
      }
      processedCount++
      recipeMap := buildIterativeRecipeMap(recipeVariant, recipeVariants)
      craftingPath := make([]RecipeStep, 0, len(recipeMap))
      for _, step := range recipeMap {
        craftingPath = append(craftingPath, step)
      }
      treeRoot := buildCraftingTreeFromMap(target, recipeMap, make(map[string]bool))
      allPaths = append(allPaths, RecipePath{craftingPath, treeRoot})
      recipeMap = nil
    }
  }
  sort.Slice(allPaths, func(i, j int) bool {
    return len(allPaths[i].Steps) < len(allPaths[j].Steps)
  })
  if len(allPaths) > maxRecipes {
    allPaths = allPaths[:maxRecipes]
  }
  return allPaths, visitCount
}
```

4.2.3 Fungsi Algoritma DFS

```
func DFS(target string, graph map[string][][2]string, tiers map[string]int, maxRecipes int) ([]RecipePath, int) {
  if maxRecipes <= 0 {
    maxRecipes = 1
  }

  craftable := make(map[string]bool)
  recipeVariants := make(map[string][]RecipeStep)
  visitCount := 0
  visited := make(map[string]bool)
  stack := []string{}

  for base := range baseElements {
    craftable[base] = true
    visited[base] = true
    stack = append(stack, base)
  }

  for len(stack) > 0 && len(recipeVariants[target]) < maxRecipes {
    lastIdx := len(stack) - 1
    current := stack[lastIdx]
    stack = stack[:lastIdx]
    visitCount++

    for ingredient := range craftable {
      possibleResults := findRecipes(current, ingredient, graph)

      for _, result := range possibleResults {
        resultTier := tiers[result]
        currentTier := tiers[current]
        ingredientTier := tiers[ingredient]

        if resultTier > currentTier && resultTier > ingredientTier {
          newRecipe := RecipeStep{
            Ingredient1: current,
            Ingredient2: ingredient,
            Result:      result,
          }

          if len(recipeVariants[result]) < maxRecipes {
            isDuplicate := false
            for _, existingRecipe := range recipeVariants[result] {
              if (existingRecipe.Ingredient1 == current && existingRecipe.Ingredient2 == ingredient) ||
                (existingRecipe.Ingredient1 == ingredient && existingRecipe.Ingredient2 == current) {
                isDuplicate = true
                break
              }
            }

            if !isDuplicate {
              recipeVariants[result] = append(recipeVariants[result], newRecipe)
            }
          }

          if !craftable[result] && !visited[result] {
            craftable[result] = true
            visited[result] = true
            stack = append(stack, result)
          }
        }
```

```
      }
    }
  }

  if !craftable[target] {
    fmt.Printf("Cannot craft %s from base elements\n", target)
    return nil, visitCount
  }

  var allPaths []RecipePath
  processedCount := 0

  if maxRecipes > 1 {
    resultChan := make(chan RecipePath, maxRecipes)
    var wg sync.WaitGroup
    maxWorkers := 3
    sem := make(chan struct{}, maxWorkers)

    for _, recipeVariant := range recipeVariants[target] {
      if processedCount >= maxRecipes {
        break
      }
      processedCount++

      wg.Add(1)
      sem <- struct{}{}
      go func(recipe RecipeStep) {
        defer wg.Done()
        defer func() { <-sem }()

        recipeMap := buildIterativeRecipeMap(recipe, recipeVariants)
        craftingPath := make([]RecipeStep, 0, len(recipeMap))
        for _, step := range recipeMap {
          craftingPath = append(craftingPath, step)
        }

        treeRoot := buildCraftingTreeFromMap(target, recipeMap, make(map[string]bool))
        resultChan <- RecipePath{craftingPath, treeRoot}
        recipeMap = nil
      }(recipeVariant)
    }

    go func() {
      wg.Wait()
      close(resultChan)
    }()

    for path := range resultChan {
      allPaths = append(allPaths, path)
    }
  } else {
    for _, recipeVariant := range recipeVariants[target] {
      if processedCount >= maxRecipes {
        break
      }
      processedCount++

      recipeMap := buildIterativeRecipeMap(recipeVariant, recipeVariants)
      craftingPath := make([]RecipeStep, 0, len(recipeMap))
      for _, step := range recipeMap {
        craftingPath = append(craftingPath, step)
      }

      treeRoot := buildCraftingTreeFromMap(target, recipeMap, make(map[string]bool))
      allPaths = append(allPaths, RecipePath{craftingPath, treeRoot})
      recipeMap = nil
    }
  }

  sort.Slice(allPaths, func(i, j int) bool {
    return len(allPaths[i].Steps) < len(allPaths[j].Steps)
  })

  if len(allPaths) > maxRecipes {
    allPaths = allPaths[:maxRecipes]
  }

  return allPaths, visitCount
}
```

4.2.4 Fungsi Rekonstruksi Tree

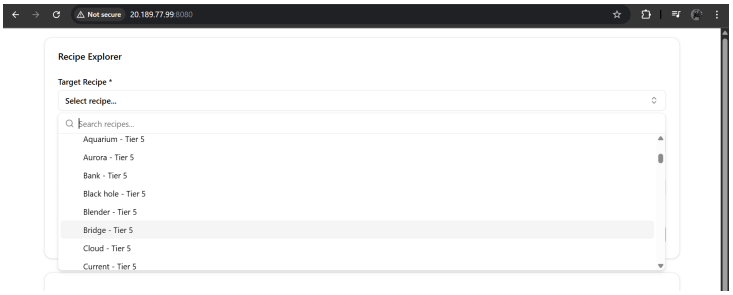
```
func buildCraftingTreeFromMap(element string, recipeMap map[string]RecipeStep, path map[string]bool) *TreeNode {
  if path[element] {
    return &TreeNode{
      Element:  element,
      Children: nil,
    }
  }

  if baseElements[element] {
    return &TreeNode{
      Element:  element,
```

```
      Children: nil,
    }
  }

  recipe, exists := recipeMap[element]
  if !exists {
    return &TreeNode{
      Element:  element,
      Children: nil,
    }
  }

  newPath := make(map[string]bool, len(path)+1)
  for k, v := range path {
    newPath[k] = v
  }
  newPath[element] = true

  node := &TreeNode{
    Element:    element,
    RecipeStep: &recipe,
    Children:   make([]*TreeNode, 0),
  }

  ing1 := recipe.Ingredient1
  ing2 := recipe.Ingredient2

  node.Children = append(node.Children, buildCraftingTreeFromMap(ing1, recipeMap, newPath))
  node.Children = append(node.Children, buildCraftingTreeFromMap(ing2, recipeMap, newPath))

  return node
}
```

4.3. Cara Penggunaan Aplikasi Web

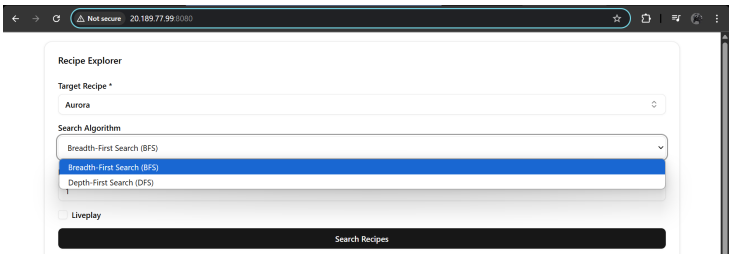
Penggunaan website dapat dilakukan dengan langkah-langkah berikut

1. Pilih target elemen



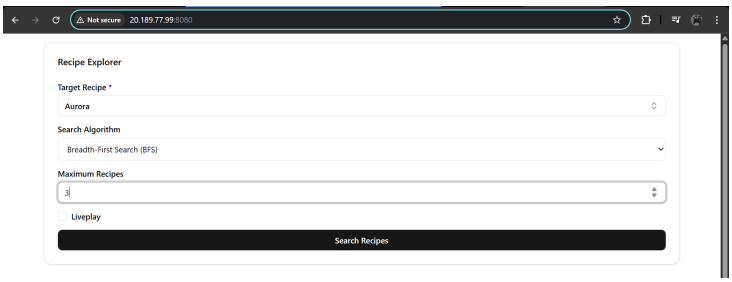
Gambar 7: Memilih target elemen

2. Pilih algoritma pencarian elemen

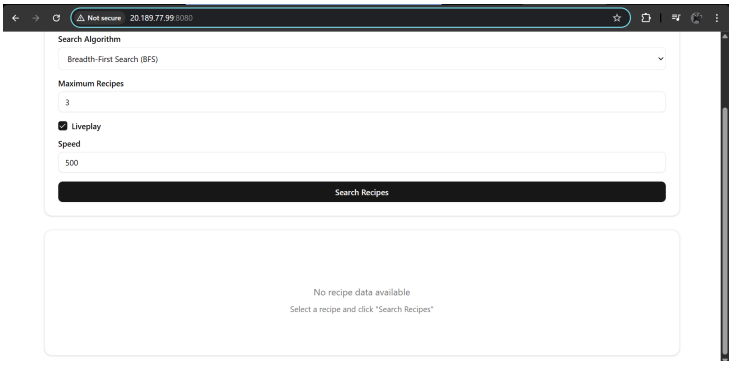


Gambar 8: Memilih algoritma pencarian elemen

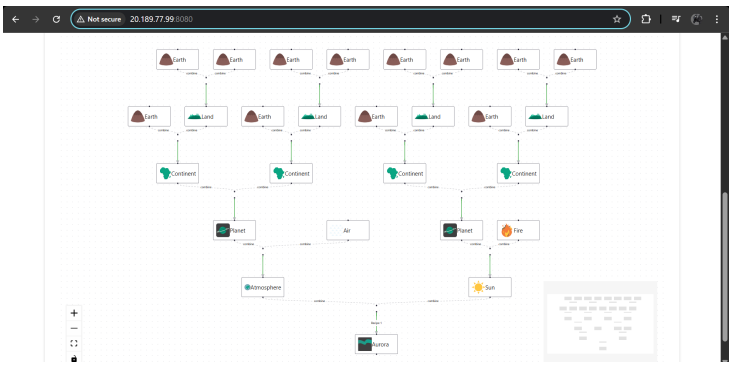
3. Tentukan jumlah maksimal recep yang dicari
4. Gunakan liveplay **(optional)**
5. Klik tombol "Search Recipes"



Gambar 9: Memilih maksimal jumlah resep



Gambar 10: Memilih liveplay



Gambar 11: Hasil pencarian resep elemen

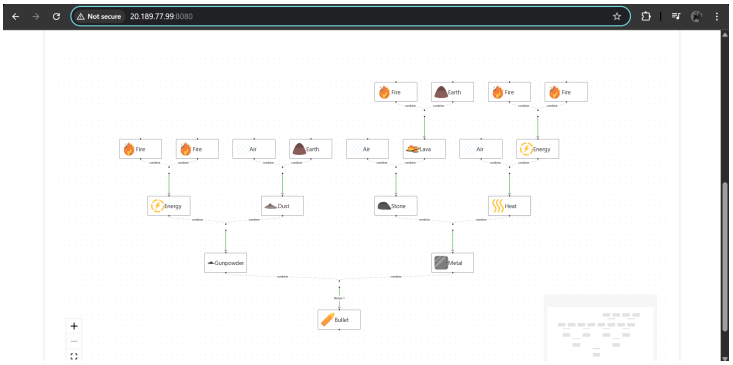
4.4. Pengujian

4.4.1 Test Case 1

Input:

- Target elemen: Bullet
- Algoritma: BFS
- Maks resep: 1

Output:



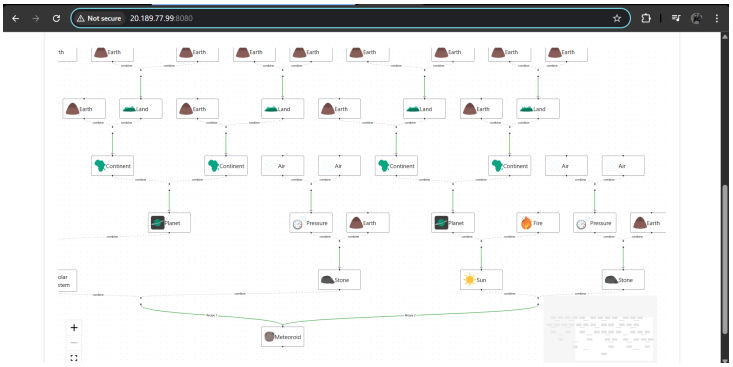
Gambar 12: Test Case 1

4.4.2 Test Case 2

Input:

- Target elemen: Meteorid
- Algoritma: DFS
- Maks resep: 3

Output:



Gambar 13: Test Case 2

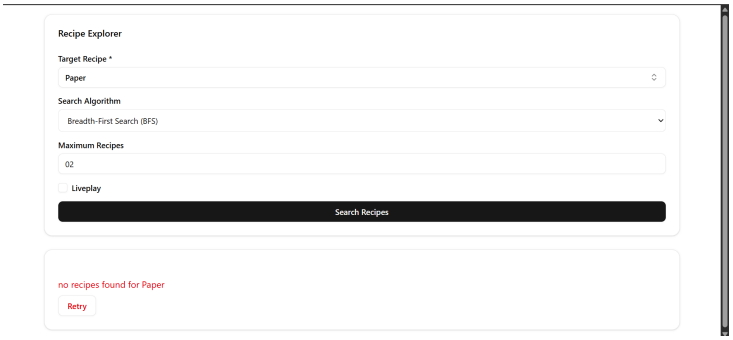
4.4.3 Test Case 3

Input:

- Target elemen: Paper

- Algoritma: BFS
- Maks resep: 2

Output:



Gambar 14: Test Case 3

Kegagalan terjadi karena semua resep dari Paper menggunakan elemen yang memiliki tier lebih besar dibandingkan dirinya sendiri.

4.5. Analisis dan Pembahasan

a. Algoritma BFS

Berdasarkan hasil pengujian kita dapatkan bahwa algoritma BFS yang dibuat telah berhasil menemukan resep untuk suatu target elemen. Algoritma BFS lebih mengutamakan eksplorasi tiap resep yang dimiliki oleh suatu elemen. Akibatnya, algoritma ini mengunjungi lebih banyak simpul dibandingkan dengan DFS. Alhasil, secara umum algoritma ini lebih lambat dibandingkan algoritma DFS dalam mencari resep dari suatu elemen.

b. Algoritma DFS

Berdasarkan hasil pengujian kita dapatkan bahwa algoritma DFS yang dibuat telah berhasil menemukan resep untuk suatu target elemen. Algoritma DFS lebih mengutamakan eksplorasi secara mendalam satu per satu resep milik suatu elemen. Akibatnya, algoritma ini mengunjungi lebih sedikit simpul dibandingkan BFS. Alhasil, Secara umum algoritma ini akan lebih cepat dibandingkan dengan algoritma BFS dalam mencari resep dari suatu elemen.

Bab V
Penutup

5.1. Kesimpulan

Dalam proyek Tugas Besar 2 IF2211 Strategi Algoritma ini, kami berhasil mengimplementasikan algoritma BFS dan DFS pada aplikasi web *recipe finder* Little Alchemy 2.

Kami berhasil membangun aplikasi web *recipe finder* dengan fungsionalitas seperti pemilihan algoritma, masukan banyak resep, masukan elemen target dan lainnya. Selain itu, kami juga berhasil menvisualisasikan resep elemen ke dalam bentuk *tree* dengan target elemen sebagai *root* dan elemen dasar sebagai *leaf*.

5.2. Saran

Pelaksanaan Tugas Besar 2 IF2211 Strategi Algoritma di Semester II (Genap) Tahun 2024/2025 merupakan pengalaman yang sangat berharga bagi kami. Dari pengalaman ini, kami ingin berbagi beberapa saran kepada pembaca yang mungkin akan menghadapi tugas serupa di masa depan:

Dicky

gw sigma lu ginger

Jibril

Read Shadow Slave its peak and not that long, just about 2300 chapters.

Adha

Crazy? I was crazy once. They locked me in a room. A rubber room. A rubber room full of tubes. And tubes make me crazy.

Semoga saran-saran ini membantu pembaca dalam menyiapkan diri untuk menangani tugas serupa di masa depan.

5.3. Refleksi

Ruang perbaikan dan pengembangan dalam mengerjakan tugas dapat difokuskan pada beberapa aspek yang dapat ditingkatkan. Pertama-tama, pengaturan waktu menjadi kunci utama. Kami menyadari bahwa perencanaan waktu yang lebih baik dapat meningkatkan efisiensi pengerjaan. Menerapkan strategi manajemen waktu, seperti membuat jadwal yang terstruktur dan menetapkan tenggat waktu internal untuk setiap tahap pekerjaan, dapat membantu menghindari tekanan waktu yang tidak perlu.

Selain itu, ketelitian membaca spesifikasi dari awal menjadi aspek yang perlu diperhatikan. Memahami secara menyeluruh tentang apa yang diminta dalam tugas, termasuk detail-detail kecil, dapat mengurangi risiko kesalahan dan memastikan pekerjaan berjalan sesuai dengan harapan. Menempatkan perhatian ekstra pada spesifikasi tugas dapat meminimalkan revisi dan penyesuaian yang mungkin diperlukan.

Komunikasi tim yang baik juga dapat dioptimalkan. Membuat saluran komunikasi yang jelas dan terbuka di antara anggota tim dapat menghindari kebingungan dan memastikan bahwa semua anggota memiliki pemahaman yang sama tentang tujuan dan tanggung jawab masing-masing. Diskusi reguler dan pembahasan mengenai kemajuan proyek dapat meningkatkan keterlibatan semua anggota tim.

Terakhir, evaluasi diri secara berkala dapat menjadi langkah penting. Menerima umpan balik, baik dari anggota tim maupun asisten, dan memanfaatkannya sebagai dasar untuk perbaikan lebih lanjut adalah cara efektif untuk terus berkembang. Selalu terbuka terhadap saran dan berkomitmen untuk belajar dari setiap pengalaman dapat membantu memperbaiki kinerja secara berkelanjutan.

Lampiran

6.1. Tautan

Repository program dapat diakses melalui tautan berikut:
`https://github.com/DaDecky/Tubes2_AshtonHallMorningRoutine`

Video penjelasan program dapat diakses melalui tautan berikut:
`https://linktr.ee/adharid1`

Website yang telah di-*deploy* dapat dikases melalui tautan berikut
`http://20.189.77.99:8080/`

6.2. Tabel Spesifikasi

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

Referensi

Rinaldi Munir. 2025. "Breadth First Search dan Depth First Search - Bagian 1." Diakses pada 12 Mei 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf

Rinaldi Munir. 2025. "Breadth First Search dan Depth First Search - Bagian 2." Diakses pada 12 Mei 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf

USACO Guide. -. "Graph Traversal" Diakses pada 11 Mei 2025. https://usaco.guide/silver/graph-traversal?lang=cpp

Ikatan Alumni Tim Olimpiade Komputer Indonesia. -. "Pemrograman Kompetitif Dasar" Diakses pada 10 Mei 2025. https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf