Laporan Tugas Besar 2 IF2211 Strategi Algoritma

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

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

Oleh:

Rafizan Muhammad Syawalazmi	13523034
Abdullah Farhan	13523042
Muhammad Zahran Ramadhan Ardiana	13523104

Kelompok 41 / Arachemy-chan

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Pada tugas besar ini, kami diminta untuk mengimplementasikan dua algoritma penjelajahan graf, yakni Breadth First Search (BFS) dan Depth First Search (DFS) untuk menyelesaikan permasalahan pencarian resep dalam game Little Alchemy. Game ini memungkinkan pemain untuk mengkombinasikan elemen-elemen dasar untuk membuat elemen baru. Tujuan dari implementasi algoritma ini adalah untuk menemukan jalur pembuatan elemen target dari elemen-elemen dasar.

Gambar 1.1 Little Alchemy 2

Spesifikasi tugas meliputi:

1. Mengimplementasikan algoritma BFS dan DFS untuk mencari resep pembuatan elemen
2. Membangun aplikasi web dengan antarmuka yang memungkinkan pengguna memilih algoritma dan melihat hasil pencarian
3. Menerapkan paralelisme dengan worker pool untuk meningkatkan efisiensi pencarian
4. Membandingkan performa kedua algoritma berdasarkan runtime dan jumlah node yang dikunjungi
5. Mengimplementasikan fitur pencarian multiple paths (beberapa jalur berbeda)

BAB II
LANDASAN TEORI

2.1 Penjelajahan Graf dan Algoritma BFS & DFS

2.1.1 Penjelajahan Graf

Graf merupakan struktur data yang terdiri dari simpul (nodes/vertices) dan sisi (edges) yang menghubungkan antar simpul. Dalam konteks game Little Alchemy, elemen-elemen dapat direpresentasikan sebagai simpul, sedangkan kombinasi antar elemen yang menghasilkan elemen baru direpresentasikan sebagai sisi.

Penjelajahan graf adalah proses mengunjungi setiap simpul dalam graf. Dua algoritma utama untuk penjelajahan graf adalah Breadth First Search (BFS) dan Depth First Search (DFS).

2.1.2 Breadth First Search (BFS)

BFS adalah algoritma penjelajahan graf yang mengunjungi semua simpul pada level yang sama sebelum melanjutkan ke level berikutnya. Algoritma ini menggunakan struktur data queue (antrian) untuk menyimpan simpul-simpul yang akan dikunjungi.

Karakteristik BFS:

- Menjelajah berdasarkan tingkat/level pohon
- Menggunakan antrian (FIFO - First In First Out)
- Menemukan jalur terpendek ke tujuan
- Kompleksitas waktu: O(V+E), di mana V adalah jumlah vertex dan E adalah jumlah edge
- Membutuhkan lebih banyak memori dibandingkan DFS

BFS sangat efektif ketika:

- Elemen target berada lebih dekat dengan elemen dasar
- Perlu menemukan jalur terpendek
- Pohon graf sangat dalam

Berikut pseudocode algoritma:

```
FUNGSI bfsSinglePath(target)
   Mulai waktu
   Ubah target menjadi huruf kecil
```

```
JIKA target adalah elemen dasar
    Kembalikan [], True, waktu yang dihabiskan, 1

Buat queue untuk menyimpan elemen yang akan diexplorasi
Buat map untuk menyimpan elemen yang sudah ditemukan dan resepnya
Tambahkan semua elemen dasar ke queue dan map

ULANGI sampai queue kosong
    Ambil ukuran level saat ini

    ULANGI untuk setiap elemen di level saat ini
        Ambil elemen saat ini dari queue

        ULANGI untuk setiap elemen yang sudah ditemukan
            Cek jika kombinasi elemen saat ini dan elemen lain dapat membentuk elemen
baru

            JIKA kombinasi valid
                Tambahkan elemen baru ke queue dan map

                JIKA elemen baru adalah target
                    Rekonstruksi jalur dari target ke elemen dasar
                    Kembalikan jalur, True, waktu yang dihabiskan, jumlah node yang
dikunjungi

Kembalikan [], False, waktu yang dihabiskan, jumlah node yang dikunjungi
```

2.1.3 Depth First Search (DFS)

DFS adalah algoritma penjelajahan graf yang menelusuri sebuah cabang sampai ke kedalaman maksimum sebelum melakukan backtracking dan menelusuri cabang lainnya. Algoritma ini menggunakan struktur data stack (tumpukan) untuk menyimpan simpul-simpul yang akan dikunjungi.

Karakteristik DFS:

- Menjelajah berdasarkan kedalaman pohon
- Menuju ke bagian bawah subpohon, lalu melakukan backtracking
- Kompleksitas waktu: O(V+E), di mana V adalah jumlah vertex dan E adalah jumlah edge
- Membutuhkan lebih sedikit memori dibandingkan BFS

DFS sangat efektif ketika:

- Perlu menelusuri semua kemungkinan jalur
- Memori terbatas
- Solusi berada jauh dBari elemen dasar

Berikut Pseudocode algoritma nya:

```
FUNGSI dfsSinglePath(element, visited, trace, nodesVisited)
  INKREMENT nodesVisited

  JIKA element adalah base element
    KEMBALIKAN kosong, BENAR

  JIKA element sudah dikunjungi
    KEMBALIKAN kosong, SALAH
  TANDAI element sebagai dikunjungi
  DAPATKAN resep untuk element

  JIKA tidak ada resep
    KEMBALIKAN kosong, SALAH

  UNTUK SETIAP bahan dalam resep
    JIKA bahan memiliki tier yang tidak kompatibel
      LOMPATI bahan ini
    LAIN
      BUAT salinan trace baru
        TAMBAHKAN element ke trace baru
      REKURSIF dfsSinglePath untuk bahan kiri
        JIKA tidak berhasil
          LOMPATI bahan ini
        LAIN
          REKURSIF dfsSinglePath untuk bahan kanan
            JIKA tidak berhasil
              LOMPATI bahan ini
            LAIN
              KOMBINASIKAN langkah-langkah dari bahan kiri dan kanan
                TAMBAHKAN langkah-langkah ke trace baru
              KEMBALIKAN trace baru, BENAR
  KEMBALIKAN kosong, SALAH
```

2.1.4 BFS Bidirectional

1. Pencarian Dua Arah: Pencarian forward dimulai dari elemen target menuju elemen dasar Pencarian backward dimulai dari elemen dasar menuju target

2. Titik Temu: Algoritma berhenti ketika kedua pencarian menemukan node yang sama Saat titik temu ditemukan, jalur lengkap bisa direkonstruksi Keuntungan Kompleksitas:
3. Jika jarak antara target dan dasar adalah d, BFS standar akan mengeksplorasi O(b^d) node BFS Bidirectional akan mengeksplorasi O(b^(d/2) + b^(d/2)) = O(b^(d/2)) node Ini bisa berarti pengurangan waktu dan memori yang sangat signifikan
4. Rekonstruksi Jalur: Jalur direkonstruksi dengan menggabungkan jalur dari titik temu ke target dan jalur dari titik temu ke elemen dasar

   Berikut pseudcocode nya:

```
FUNGSI bfsBidirectionalPath(target)

INISIALISASI dua antrian dan peta: (forward Queue, backwardQueue) dari (target) dan (elemen Dasar), serta forwardMap dan backwardMap untuk menyimpan informasi jalur

LAKUKAN BFS BIDIRECTIONAL: Perlebar forward Queue (target→dasar) dan backwardQueue (dasar→target) secara bergantian sambil melacak node yang dikunjungi

IDENTIFIKASI TITIK TEMU: Jika ditemukan node yang ada di kedua forward Map dan backwardMap, simpan sebagai calon titik temu dengan total kedalaman terpendek

REKONSTRUKSI JALUR: Dari titik temu terbaik, bangun jalur maju (titik temu→target) dan jalur mundur (titik temu→elemenDasar)

GABUNGKAN JALUR: Kombinasikan jalur maju dan mundur menjadi jalur lengkap dari target ke elemenDasar KEMBALIKAN [jalurLengkap, berhasil, waktu Eksekusi, jumlahNodeDikunjungi]
```

2.1.5 DFS Bidirectional

DFS Bidirectional adalah variasi dari algoritma Depth-First Search yang melakukan pencarian dari dua arah secara bersamaan untuk menemukan jalur antara dua titik. Berbeda dengan BFS Bidirectional yang mencari secara melebar, DFS Bidirectional mengutamakan pencarian mendalam terlebih dahulu.

Karakteristik Utama DFS Bidirectional:

1. Pendekatan Kedalaman: Menggunakan struktur data tumpukan (stack) dengan pengambilan elemen terakhir (LIFO - Last In First Out) Menjelajahi jalur sejauh mungkin sebelum mundur (backtracking)

2. Pencarian Dua Arah: Pencarian pertama dimulai dari target menuju elemen dasar Pencarian kedua dimulai dari elemen dasar menuju target Kedua pencarian berjalan secara bergantian, masing-masing mengambil satu langkah
3. Mekanisme Titik Temu: Algoritma berhenti ketika menemukan node yang telah dikunjungi oleh kedua arah pencarian Node ini menjadi titik pertemuan untuk merekonstruksi jalur lengkap
4. Perbedaan dengan BFS Bidirectional: DFS: Mengeksplorasi sejauh mungkin dalam satu cabang sebelum beralih ke cabang lain BFS: Mengeksplorasi semua node pada kedalaman yang sama sebelum pindah ke kedalaman berikutnya DFS bisa menemukan solusi lebih cepat untuk beberapa kasus, terutama jika jalur solusi cukup dalam
5. Kelebihan DFS Bidirectional: Penggunaan memori lebih efisien (proporsi dengan kedalaman maksimum, bukan lebar) Cocok untuk graf dengan percabangan lebar namun solusi berada di kedalaman tinggi Bisa menemukan solusi pertama lebih cepat dari BFS dalam beberapa kasus
6. Kelemahan DFS Bidirectional: Tidak menjamin menemukan jalur terpendek Bisa terjebak di jalur yang sangat panjang Mungkin kurang efisien untuk graf dengan banyak siklus

FUNGSI dfsBidirectionalPath(target)

INISIALISASI tumpukan dan set: (tumpukanAwal, tumpukanTujuan) dari (target) dan (elemenDasar), serta dikunjungiDariAwal dan dikunjungiDariTujuan untuk melacak node

LAKUKAN DFS BIDIRECTIONAL: Eksplorasi tumpukanAwal (target→dasar) dan tumpukanTujuan (dasar→target) secara bergantian dengan mengambil elemen terakhir dari setiap tumpukan

IDENTIFIKASI TITIK TEMU: Jika ditemukan node yang telah dikunjungi oleh kedua pencarian, simpan sebagai titikPertemuan dan rekam jalur dari kedua arah

BANGUN JALUR SETIAP ARAH: Rekonstruksi jalurTitikPertemuanAwal (dari target ke titikPertemuan) dan jalurTitikPertemuanTujuan (dari elemenDasar ke titikPertemuan)

GABUNGKAN JALUR: Kombinasikan jalurTitikPertemuanAwal dan jalurTitikPertemuanTujuan menjadi jalurLengkap dari target ke elemenDasar

KEMBALIKAN [jalurLengkap, berhasil, waktuEksekusi, nodeDikunjungi]

2.2 Penjelasan Singkat Mengenai Aplikasi Web

Aplikasi web yang dibangun merupakan sebuah sistem pencarian resep untuk game Little Alchemy. Aplikasi ini terdiri dari dua bagian utama:

1. **Backend**: Dibangun menggunakan bahasa pemrograman Go dengan framework Gin. Backend bertugas untuk mengimplementasikan algoritma BFS dan DFS, mengelola data resep, dan menyediakan API untuk frontend.
2. **Frontend**: Dibangun menggunakan React. Frontend menyediakan antarmuka pengguna untuk memasukkan elemen target, memilih algoritma, dan menampilkan hasil pencarian.

Aplikasi ini memiliki fitur utama antara lain:

- Pencarian jalur pembuatan elemen menggunakan BFS atau DFS
- Pencarian single path (satu jalur) atau multiple paths (beberapa jalur)
- Visualisasi perbandingan performa algoritma (runtime dan nodes visited)
- Penggunaan worker pool untuk meningkatkan efisiensi dengan paralelisme

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

1. **Pengumpulan Data**: Mengumpulkan data resep dari Little Alchemy melalui web scraping (terlihat dari fungsi ScrapeHandler pada kode)
2. **Pemodelan Graf**: Merepresentasikan elemen sebagai simpul dan kombinasi sebagai sisi dalam graf
3. **Implementasi Algoritma**:
   - Mengimplementasikan BFS dan DFS untuk pencarian single path
   - Mengimplementasikan BFS dan DFS untuk pencarian multiple paths
4. **Optimasi dengan Worker Pool**: Menerapkan paralelisme untuk meningkatkan efisiensi pencarian
5. **Pengembangan Antarmuka Web**: Membuat UI yang interaktif dan informatif

3.2 Pemetaan Masalah ke Elemen Algoritma BFS dan DFS

Elemen Graf:

- **Simpul (Vertex)**: Setiap elemen dalam Little Alchemy (fire, water, earth, air, dll)
- **Sisi (Edge)**: Kombinasi dua elemen yang menghasilkan elemen baru
- **Elemen Dasar**: Fire, water, earth, air, time (berfungsi sebagai simpul awal)
- **Elemen Target**: Elemen yang ingin dibuat (simpul tujuan)

Pemetaan BFS:

- **Queue:** Menyimpan elemen-elemen yang akan dieksplorasi
- **Discovered Set:** Menyimpan elemen-elemen yang sudah ditemukan
- **Pencarian Level by Level:** Menelusuri semua kombinasi pada level yang sama sebelum melanjutkan

Pemetaan DFS:

- **List//Rekursi: Menyimpan elemen-elemen yang akan dieksplorasi**
- **Visited Set: Menyimpan elemen-elemen yang sudah dikunjungi**
- **Pencarian Depth by Depth: Menelusuri satu cabang sampai mencapai elemen dasar** atau dead end

3.3 Fitur Fungsional dan Arsitektur Aplikasi Web

Fitur Fungsional:

1. **Pencarian Single Path**: Menemukan satu jalur untuk membuat elemen target
2. **Pencarian Multiple Paths**: Menemukan beberapa jalur berbeda untuk membuat elemen target
3. **Pemilihan Algoritma**: Opsi untuk menggunakan BFS atau DFS
4. **Penyetelan Parameter**: Mengatur jumlah resep yang ingin ditemukan
5. **Statistik Performa**: Menampilkan runtime dan jumlah node yang dikunjungi

Arsitektur Aplikasi:

1. **Backend (Go/Gin)**:
    - **Package main**: Entry point aplikasi
    - **API Endpoints**: Menangani permintaan dari frontend
    - **Implementasi Algoritma**: BFS dan DFS untuk single dan multiple paths
    - **Worker Pool**: Paralelisme untuk meningkatkan efisiensi
    - **Data Management**: Pengelolaan data resep dan elemen
2. **Frontend (React)**:
    - **UI Components**: Form input, hasil pencarian, visualisasi performa
    - **State Management**: Mengelola state aplikasi
    - **API Client**: Berkomunikasi dengan backend

3.4 Contoh Ilustrasi Kasus

Misalkan kita ingin mencari resep untuk membuat elemen "bread":

1. **Input**: Target = "bread", Method = "bfs", Number of Recipes = 1
2. **Proses**:
    - BFS akan mulai dari elemen-elemen dasar (fire, water, earth, air, time)
    - BFS akan menelusuri level demi level, menemukan kombinasi elemen-elemen yang menghasilkan elemen baru
    - Ketika menemukan "bread", BFS akan berhenti dan merekonstruksi jalur
3. **Output**:

- Jalur pembuatan "bread" (misal: "fire + earth = clay", "clay + water = mud", "mud + fire = brick", "brick + fire = bread")
- Runtime: x ms
- Nodes Visited: y nodes

BAB IV
IMPLEMENTRASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

Struktur Data:

1. **Recipe**: Struct untuk merepresentasikan resep

```
type Recipe struct {
    Element     string `json:"Element"`
    Ingredient1 string `json:"Ingredient1"`
    Ingredient2 string `json:"Ingredient2"`
    Type        int    `json:"Type"`
}
```

2. **Result**: Struct untuk merepresentasikan hasil pencarian

```
type Result struct {
    Found        bool          `json:"found"`
    Steps        []string      `json:"steps"`
    Runtime      time.Duration `json:"runtime"`
    NodesVisited int           `json:"nodesVisited"`
}
```

3. **BFSMultipleJob/DFSMultipleJob**: Struct untuk worker pool

```
type BFSMultipleJob struct {
    Target   string
    MaxPaths int
    JobID    int
}
```

Fungsi Utama:

1. **bfsSinglePath**: Implementasi BFS untuk single path
2. **dfsSinglePath**: Implementasi DFS untuk single path
3. **bfsMultiplePaths**: Implementasi BFS untuk multiple paths
4. **dfsMulPath**: Implementasi DFS untuk multiple paths
5. **StartBFSMultipleWorkerPool/StartDFSMultipleWorkerPool**: Implementasi worker pool

6. **reconstructPath**: Merekonstruksi jalur dari elemen target ke elemen dasar pada algoritma BFS
7. **dfsBidirectionalPath:** Implementasi DFS bidirectional untuk single path
8. **bfsBidirectionalPath:** Implementasi BFS bidirectional untuk single path

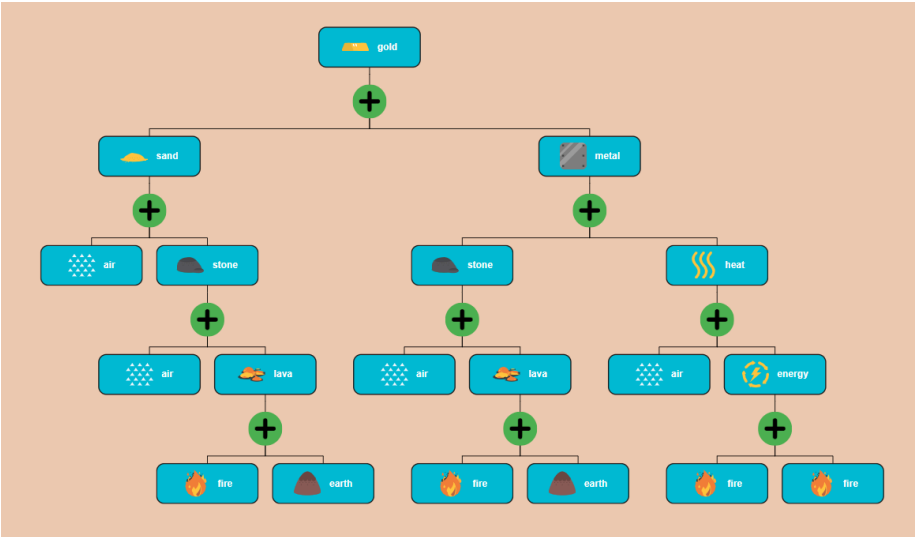
4.2 Tata Cara Penggunaan Program

1. **Memulai Aplikasi**:
   - Jalankan backend: `go build -o main && go run main`
   - Jalankan frontend: `npm run dev`
   - Akses frontend melalui browser: `http://localhost:5173`
2. **Pencarian Resep**:
   - Masukkan elemen target pada kolom input
   - Pilih algoritma pencarian (BFS atau DFS)
   - Tentukan jumlah resep yang ingin dicari
   - Klik tombol "Cari" untuk memulai pencarian
3. **Melihat Hasil**:
   - Hasil pencarian akan ditampilkan berupa langkah-langkah pembuatan elemen
   - Statistik performa (runtime dan nodes visited) juga ditampilkan
   - Jika tidak ditemukan, akan muncul pesan "Tidak Ditemukan"

4.3 Hasil Pengujian

Pengujian 1: Pencarian "gold" menggunakan BFS (Single Path)

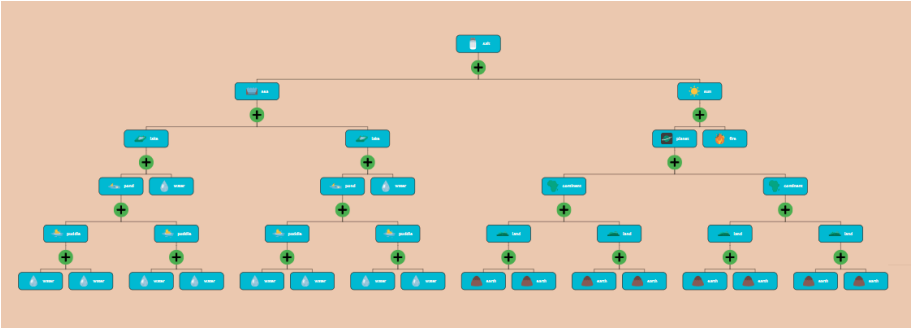
- **Input**: Target = "gold", Method = "bfs", Number of Recipes = 1
- **Hasil**:
  - Hasil Tree:



- Runtime: 75.957ms
- Nodes Visited: 52

Pengujian 2: Pencarian "gold" menggunakan DFS (Single Path)

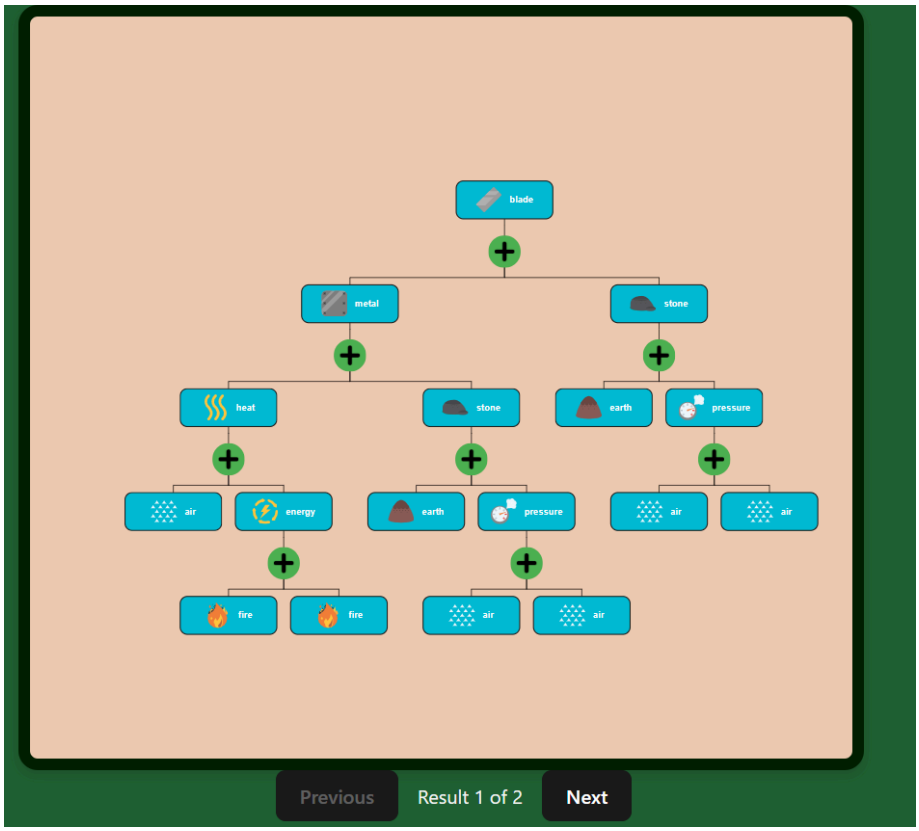
- **Input**: Target = "salt", Method = "dfs", Number of Recipes = 1
- **Hasil**:
  - Hasil Tree:



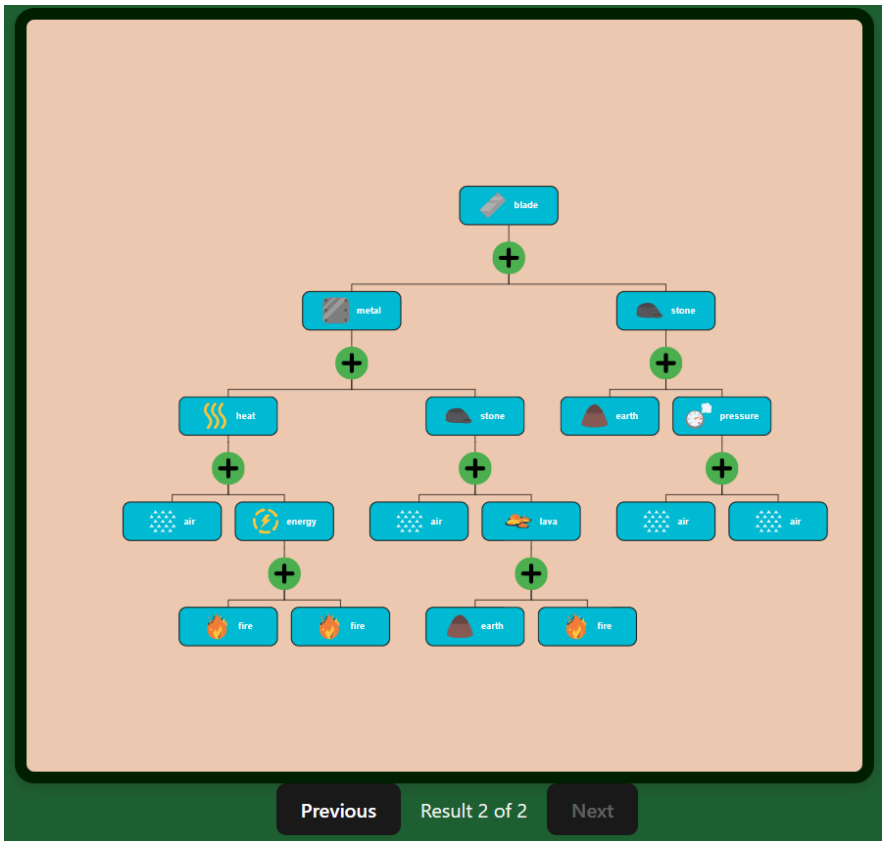
  - Runtime: 3.9929ms
  - Nodes Visited: 37

Pengujian 3: Pencarian "bread" menggunakan BFS (Multiple Paths)

- **Input**: Target = "blade", Method = "bfs", Number of Recipes = 2
- **Hasil**:
  - Path 1:


blade
metal
stone
heat
stone
earth
pressure
air
energy
earth
pressure
air
air
fire
fire
air
air
Previous
Result 1 of 2
Next

- Path 2:



- Runtime: 1.1963575s
- Nodes Visited: 619

4.4 Analisis Hasil Pengujian

1. **Perbandingan BFS vs DFS**:
   - BFS cenderung mengunjungi lebih banyak node daripada DFS untuk single path
   - DFS memiliki runtime yang lebih cepat untuk single path pada beberapa kasus
   - BFS menemukan jalur yang lebih pendek (optimal) untuk beberapa elemen
2. **Analisis Efisiensi Worker Pool**:
   - Worker pool meningkatkan performa terutama untuk pencarian multiple paths
   - Paralelisme efektif ketika mencari banyak jalur sekaligus
3. **Faktor yang Mempengaruhi Performa**:
   - Kompleksitas elemen target (tingkat/tier elemen)
   - Jumlah resep yang tersedia
   - Algoritma yang digunakan

- Jumlah worker dalam worker pool

BAB V
KESIMPULAN, SARAN, REFLEKSI

5.1 Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa:

1. Algoritma BFS dan DFS sama-sama efektif untuk mencari resep dalam game Little Alchemy, namun memiliki karakteristik yang berbeda.
2. BFS cenderung menemukan jalur yang lebih pendek (optimal), tetapi mengunjungi lebih banyak node.
3. DFS cenderung lebih cepat untuk single path pada beberapa kasus, tetapi tidak menjamin jalur terpendek.
4. Implementasi worker pool berhasil meningkatkan efisiensi pencarian terutama untuk multiple paths.
5. Aplikasi web berhasil menyediakan antarmuka yang interaktif dan informatif untuk pengguna.

5.2 Saran

Untuk pengembangan lebih lanjut, beberapa saran yang dapat dipertimbangkan:

1. Menambahkan algoritma pencarian lain seperti A* untuk perbandingan
2. Mengimplementasikan caching untuk meningkatkan performa
3. Menambahkan visualisasi grafik untuk memperjelas perbandingan algoritma
4. Meningkatkan UI/UX dengan animasi dan tampilan yang lebih menarik
5. Menambahkan fitur favorit atau history pencarian

5.3 Refleksi

Melalui pengerjaan tugas besar ini, kami telah belajar banyak hal, antara lain:

1. Pemahaman mendalam tentang algoritma BFS dan DFS serta implementasinya
2. Pengembangan aplikasi web dengan teknologi modern (Go/Gin dan React)
3. Penerapan konsep paralelisme dengan worker pool
4. Analisis performa dan optimasi algoritma
5. Kerja sama tim dan manajemen proyek

LAMPIRAN

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

Tautan Repository GitHub

https://github.com/Farhanabd05/Tubes2_Arachemy-chan

Tautan Web (Possible in 30 days From May 13, 2025)

https://frontend-production-c72f.up.railway.app/

DAFTAR PUSTAKA

1. "BFS vs DFS – Perbedaan Antara Keduanya." Guru99. https://www.guru99.com/id/difference-between-bfs-and-dfs.html
2. "Breadth-first search." Wikipedia. https://en.wikipedia.org/wiki/Breadth-first_search
3. "When to Use Depth First Search vs Breadth First Search." Dgraph Blog. https://dgraph.io/blog/post/depth-first-search-vs-breadth-first-search/
4. "Tutorial: Developing a RESTful API with Go and Gin." Go Documentation. https://go.dev/doc/tutorial/web-service-gin
5. "Understanding Concurrency Patterns in Go." HackerNoon. https://hackernoon.com/understanding-concurrency-patterns-in-go
6. "Exploring Time Complexity of the Breadth First Search Algorithm." Programiz Pro. https://programiz.pro/resources/dsa-bfs-complexity/