

IF2211 – Strategi Algoritma
Laporan Tugas Besar 2: Algoritma Greedy



Dipersiapkan oleh:

gitulah

Frederiko Eldad Mugiyono (13523147)

Arlow Emmanuel Hergara (13523161)

Filbert Engyo (13523163)

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

2025

Daftar Isi

Daftar Isi	2
Daftar Gambar	3
1. Deskripsi Tugas	4
2. Landasan Teori	6
2.1. Algoritma Breadth First Search	6
2.2. Algoritma Depth First Search	6
2.3. Algoritma Bidirectional Search	6
3. Analisis Pemecahan Masalah	8
3.1. Scraping Web	8
3.2. Metode Pencarian	8
3.3. Fungsionalitas dan Arsitektur	8
3.4. Contoh Ilustrasi Kasus	8
4. Implementasi dan Pengujian	9
4.1. Spesifikasi Teknis	9
4.1.1. Frontend	9
4.1.2. Backend	12
4.1.3. Database	35
4.2. Tata Cara Penggunaan	35
4.3. Hasil Pengujian	35
4.4. Analisis Hasil	37
5. Kesimpulan dan Saran	39
5.1. Kesimpulan	39
5.2. Saran	39
Lampiran	40
Daftar Pustaka	40

Daftar Gambar

Gambar 1.1. Elemen-Elemen dalam Little Alchemy 2	4
Gambar 1.2. Elemen dasar pada Little Alchemy 2	5

1. Deskripsi Tugas



Gambar 1.1. Elemen-Elemen dalam Little Alchemy 2

Little Alchemy 2 merupakan permainan berbasis web / aplikasi yang dikembangkan oleh Recloak yang dirilis pada tahun 2017, permainan ini bertujuan untuk membuat 720 elemen dari 4 elemen dasar yang tersedia yaitu air, earth, fire, dan water. Permainan ini merupakan sekuel dari permainan sebelumnya yakni Little Alchemy 1 yang dirilis tahun 2010.

Mekanisme dari permainan ini adalah pemain dapat menggabungkan kedua elemen dengan melakukan drag and drop, jika kombinasi kedua elemen valid, akan memunculkan elemen baru, jika kombinasi tidak valid maka tidak akan terjadi apa-apa. Permainan ini tersedia di web browser, Android atau iOS

Pada Tugas Besar pertama Strategi Algoritma ini, mahasiswa diminta untuk menyelesaikan permainan Little Alchemy 2 ini dengan menggunakan strategi Depth First Search dan Breadth First Search.

Komponen-komponen dari permainan ini antara lain:

- Elemen dasar

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



Gambar 1.2. Elemen dasar pada Little Alchemy 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.

- *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.

2. Landasan Teori

2.1. Algoritma Breadth First Search

Breadth First Search (BFS) adalah algoritma pencarian atau penelusuran pada struktur data graf atau pohon yang bekerja dengan cara menjelajahi semua simpul (node) pada level yang sama terlebih dahulu sebelum berpindah ke level berikutnya. BFS sering digunakan untuk menemukan jalur terpendek dalam graf tak berbobot dan sangat populer dalam berbagai aplikasi seperti pencarian rute, penelusuran jaringan, dan pemecahan masalah teka-teki.

Untuk cara kerjanya, dibagi menjadi 3 langkah utama yaitu:

1. Inisialisasi: Tentukan simpul awal (root/start node) yang akan dijadikan titik awal penelusuran, lalu masukkan simpul awal ke dalam struktur data antrian (*queue*).
2. Pencarian: diambil simpul dari depan antrian, lalu tandai simpul tersebut sebagai simpul yang telah dikunjungi. Lalu, kunjungi semua simpul tetangga (*neighbor*) yang belum dikunjungi, kemudian masukkan mereka ke dalam antrian. Proses diulangi hingga antrian kosong atau tujuan ditemukan.
3. Penandaan: Setiap simpul yang sudah dikunjungi dicatat agar tidak dikunjungi berulang kali, biasanya menggunakan tabel boolean atau set.

2.2. Algoritma Depth First Search

Depth First Search (DFS) adalah algoritma pencarian atau penelusuran pada struktur data graf atau pohon yang bekerja dengan cara menelusuri satu cabang secara mendalam sebelum kembali (backtracking) dan melanjutkan ke cabang berikutnya. DFS sangat populer dalam pemecahan masalah yang membutuhkan eksplorasi mendalam, seperti pemecahan labirin, pencarian jalur, dan traversal struktur pohon.

Untuk cara kerjanya, pencarian dimulai dari simpul (node) awal, lalu algoritma akan memilih satu cabang dan menelusurnya hingga ke node terdalam atau hingga menemukan jalan buntu (tidak ada tetangga yang belum dikunjungi). Jika sudah tidak ada node yang bisa dikunjungi pada cabang tersebut, algoritma melakukan *backtracking* ke node sebelumnya dan melanjutkan ke cabang lain yang belum dieksplorasi. Proses ini diulang secara rekursif hingga semua node telah dikunjungi atau solusi ditemukan

2.3. Algoritma Bidirectional Search

Bidirectional Search adalah algoritma pencarian pada graf yang dirancang untuk menemukan jalur terpendek dari simpul awal ke simpul tujuan dengan melakukan dua proses pencarian secara bersamaan yaitu satu dari titik awal menuju tujuan (*forward search*) dan satu lagi dari titik tujuan menuju awal (*backward search*). Algoritma ini akan berhenti ketika kedua pencarian bertemu di satu simpul yang sama, kemudian jalur dari kedua arah tersebut digabungkan untuk membentuk solusi akhir.

Untuk cara kerjanya, terbagi menjadi 3 yaitu:

- Dua Proses Pencarian: Algoritma ini menjalankan dua pencarian secara simultan; satu mulai dari simpul awal, satu lagi dari simpul tujuan.
- Pertemuan di Tengah: Pencarian dihentikan ketika kedua proses bertemu pada satu simpul yang sama. Jalur dari simpul awal ke titik pertemuan dan dari simpul tujuan ke titik pertemuan digabungkan menjadi satu jalur lengkap.
- Pengurangan Kompleksitas: Dengan memulai pencarian dari dua arah, jumlah simpul yang perlu dieksplorasi menjadi jauh lebih sedikit dibandingkan pencarian satu arah (*unidirectional*), sehingga waktu komputasi dapat berkurang secara signifikan, terutama pada graf yang besar.

Secara garis besar, bidirectional search adalah BFS berjumlah 2 dari root.

3. Analisis Pemecahan Masalah

3.1. *Scraping Web*

Proses diawali dengan dilakukan scraping terhadap web wiki Little Alchemy 2, lalu hasil dari scraping tersebut disimpan dalam database dengan 2 tabel / entitas utama yaitu elements dan recipes. Table elements memiliki atribut id sebagai Primary Key (PK), name (VARCHAR), tier (INTEGER), dan image_url (VARCHAR). Sedangkan untuk tabel recipes memiliki atribut result_id (FK ke elements(id)), dependency1_id (FK ke elements(id)), dependency2_id (FK ke elements(id)), dan gabungan dari ketiganya berperan sebagai PK.

3.2. *Metode Pencarian*

Setelah didapatkan segala data yang diperlukan, maka langkah selanjutnya ada menentukan cara penelusuran resep. Setelah melakukan diskusi, ditentukan bahwa metode paling mudah untuk dilakukan adalah mencari dari elemen target yang terus dipecah menjadi 2 elemen pembentuknya, proses terus dilakukan secara rekursif hingga elemen pembentuk tergolong sebagai elemen dasar.

Untuk penjelasan teknikalnya, elemen yang menjadi target akan menjadi root / base node dari suatu pohon, 2 simpul daun yang bersambungan dengan simpul utama adalah elemen pembentuk dari elemen target. Setiap simpul lain adalah resep yang digunakan atau elemen pembentuk yang digabung untuk mendapat suatu elemen tertentu. Jika terdapat suatu percabangan menandakan adanya resep lain dengan 2 elemen pembentuk yang berbeda untuk membentuk elemen dituju yang sama.

3.3. *Fungsionalitas dan Arsitektur*

Berdasarkan metode yang telah ditetapkan, hal yang diperlukan adalah mengolah proses menggunakan algoritma sesuai spesifikasi yaitu algoritma BFS, DFS, dan Bidirectional Search (bonus). untuk setiap fungsi algoritma pencarian, akan terbagi menjadi 2 fungsi utama yaitu menerapkan single threading dan multi threading untuk meningkatkan efisiensi proses.

3.4. *Contoh Ilustrasi Kasus*

Contohnya elemen yang ingin dicari adalah brick, brick akan dipecah menjadi Mud dan Fire atau Clay dan Fire atau berbagai resep lain. Setelah itu Mud atau Clay atau yang lain akan dipecah menjadi elemen-elemen dengan tier yang lebih rendah sampai akhirnya pemecahan berhasil pada hanya elemen dasar sebagai pembentuk yaitu Fire, Earth, Water, dan Air sebagai elemen dengan tier paling rendah. Tergantung pada proses algoritmanya, proses dapat berhenti saat penemuan dari resep pertama saja atau sampai resep-resep lainnya terpenuhi baru dibandingkan.

4. Implementasi dan Pengujian

4.1. Spesifikasi Teknis

Program dibuat dengan *frontend*, *backend*, dan *database* yang terpisah. Ketiga bagian ini dihubungkan melalui docker compose yang mempermudah pengaturan ketiga bagian pada *environment* manapun.

4.1.1. Frontend

Frontend dibuat menggunakan Framework [Next.Js](#), dan ReactFlow untuk visualisasi. Beserta shadcn ui untuk komponen yang digunakan. Dengan itu, dibentuk website dengan tampilan sederhana yang bagian krusial disimpan dalam folder components segala hal yang menyangkut dengan tampilan, komunikasi dengan backend, dan menampilkan pohon pencarian.

page.tsx

```
const handleSearch = async (requestBody: unknown) => {
    try {
        const response = await
fetch(`#${process.env.NEXT_PUBLIC_BACKEND_PUBLIC_API_URL}/fullrecipe/`, {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(requestBody)
        })

        if (!response.ok) throw new Error(`HTTP error! Status:
${response.status}`)

        const data = await response.json()
        const searchId = data.search_id

        pollRecipeData(searchId)
    } catch (error) {
        console.error("Fetch error:", error)
    }
}
```

Fungsi handleSearch adalah fungsi yang mengurus *request* ke backend, sebenarnya banyak fungsi yang juga mengurus request tetapi perbedaan utama hanya saja perbedaan terletak pada route yang terbentuk pada backend. Secara garis besar, fungsi ini melakukan fetching secara asinkron ke PUBLIC_API_URL pada backend yang sudah diatur dalam .env. Lalu metode POST sudah jelas untuk mengirim request dengan body seperti pada contoh body dibawah ini:

page.tsx

```
{
    "element":717,
    "method":"bfs",
```

```
    "count":100,  
    "delay":0,  
    "threading":"multi"  
}
```

parser.ts

```
export function parseRecipeJson(recipeJson: RecipeJson): { flowNodes: FlowNode[]; flowEdges: FlowEdge[] } {  
  
    const { nodes, dependencies } = recipeJson;  
    const nodeMap: { [key: number]: string } = {};  
    const flowNodes: FlowNode[] = [];  
    const flowEdges: FlowEdge[] = [];  
    const nameMap = recipeJson.nameMap || {};  
  
    const g = new dagre.graphlib.Graph();  
    g.setGraph({});  
    g.setDefaultEdgeLabel(() => ({}));  
  
    nodes.forEach((name, index) => {  
        const id = index.toString();  
        const element = nameMap[name];  
        const label = element?.name || `Element ${name}`;  
        const image_url = element?.image_url.split("svg")[0] + "svg";  
  
        nodeMap[index] = id;  
        flowNodes.push({  
            id,  
            type: 'image',  
            data: {  
                label,  
                image_url  
            },  
            position: { x: 0, y: 0 },  
            width: 25,  
            height: 25,  
        });  
        g.setNode(id, { width: 25, height: 25 });  
    });  
  
    let sauce = 'Yuuka'  
    let color = generateRandomColor();  
    dependencies.forEach(dep => {  
        const source = nodeMap[dep.result];  
        if (source === sauce) color = generateRandomColor();  
        sauce = source;  
        const target1 = nodeMap[dep.dependency1];  
        const target2 = nodeMap[dep.dependency2];  
  
        flowEdges.push({ id: `${target1}-${source}`, type: 'step', source: source, target: target1, style: { stroke: color } });  
        flowEdges.push({ id: `${target2}-${source}`, type: 'step', source: source, target: target2, style: { stroke: color } });  
  
        g.setEdge(source, target1);  
        g.setEdge(source, target2);  
    });  
  
    dagre.layout(g);  
}
```

```

// Get the Dagre-computed position of the first node
const anchorId = flowNodes[0].id;
const anchorNode = g.node(anchorId);

// Calculate offset to move anchor node to (0, 0)
const offsetX = anchorNode.x;
const offsetY = anchorNode.y;

flowNodes.forEach(node => {
  const nodeWithPosition = g.node(node.id);
  node.position = {
    x: nodeWithPosition.x - offsetX - node.width / 2,
    y: nodeWithPosition.y - offsetY - node.height / 2
  };
});

return { flowNodes, flowEdges };
}

```

Fungsi `parseRecipeJson` menjadi fungsi yang mengurus tampilan pohon yang terbentuk, proses berawal dari dilakukan parsing berdasarkan format yang dikembalikan dari fungsi pencarian pada backend dan data dari setiap elemen, lalu dilakukan mapping yang berisi index dan id elemen yang berada pada nodes. Setelah itu, nodes dimasukkan kedalam `flowNodes` dalam tipe bentukan tentunya sampai mencakup seluruhnya. Lalu, pembuatan edge baru dimulai dengan elemen target berperan sebagai *source* atau asal, kemudian berdasarkan struktur yang telah ditentukan akan terbentuk id unik untuk setiap edge berdasarkan `targetX-source` yang bertipe string. Akhirnya, dilakukan auto-layout memanfaatkan library dagre, auto-layout tidak akan mengubah elemen target karena dijadikan patokan dengan penghitungan offset 0,0 untuk pergeseran seluruh node yang berhasil dibuat. Proses diakhiri dengan mengembalikan `flowNodes` dan `flowEdges` untuk diproses lebih lanjut.

tree.tsx

```

export default function RecipeTree({ recipeJson } : { recipeJson :
RecipeJson}) {
  const { flowNodes, flowEdges } = parseRecipeJson(recipeJson);
  return (
    <div style={{ width: '90%', height: '62.5vh' }}>
      <ReactFlow nodes={flowNodes} edges={flowEdges}
      edgeTypes={edgeTypes} nodeTypes={nodeTypes} fitView
        nodesDraggable={false}
        nodesConnectable={false}
        elementsSelectable={false}
        nodesFocusable={false}
        panOnScroll={true}
        zoomOnScroll={true}
        zoomOnPinch={true}
        panOnDrag={true}
        selectionOnDrag={false}
        preventScrolling={false}
      >
        <Controls />
      </ReactFlow>
    </div>
  );
}

```

```
};
```

Kode diatas adalah react component yang diperlukan melakukan rendering hasil parsing, proses rendering memanfaatkan sarana Reactflow dan nodes sebagai flowNodes, edges sebagai flowEdges, semua dengan type edge/node yang sudah ditetapkan sebelumnya. Lalu, komponen diatur sedemikian rupa agar tidak bisa dirusak oleh pengguna dan juga disiapkan navigator untuk melakukan zoom in atau out, *handle* ini diperlukan karena pengguna hanya dapat melakukan scroll dan drag. Selain itu, diatur juga tampilan fitView dimana penyesuaian tampilan 62.5% dari ketinggian browser dengan lebar 90%.

4.1.2. Backend

Backend dibuat menggunakan golang dan terdiri atas 5 package selain package utama, yaitu database, schema, findfullrecipe, middleware, dan route. Database merupakan wrapper untuk penggunaan database dan berisi beberapa fungsi seperti berikut.

```
connection.go
```

```
// membuka hubungan antara database dan backend
func Initialize() error {
    ...
}

// menutup hubungan antara database dan backend
func Close() {
    ...
}

// mengirim query pada database dan membaca kembali hasilnya
func Exec(sql string, args ...any) (pgconn.CommandTag, error) {
    ...
}

// mengirim query pada database
func Query(sql string, args ...any) (pgx.Rows, error) {
    ...
}

// mengirim query pada database dan membaca kembali 1 baris
func QueryRow(sql string, args ...any) pgx.Row {
    ...
}
```

```
definition.go
```

```
// mendefinisikan tabel-tabel dalam database
func Define() {
    ...
}

// mengecek apakah tabel-tabel dalam database sudah terdefinisi
```

```
func IsDefined() bool {  
    ...  
}
```

seeder.go

```
// melakukan scraping dan mengisi database  
func Seed() {  
    ...  
}
```

basicquery.go

```
// mencari elemen dengan id tertentu  
func FindElementById(id int) (schema.Element, error) {  
    ...  
}  
  
// mencari semua elemen dengan nama serupa  
func FindElementByName(name string) ([]schema.Element, error) {  
    ...  
}  
  
// mencari semua elemen dalam tier tertentu  
func FindElementInTier(start int, end int, tier int) ([]schema.Element, error) {  
    ...  
}  
  
// mencari elemen dengan nomor urut start hingga end berdasarkan nama  
func Elements(start int, end int) ([]schema.Element, error) {  
    ...  
}  
  
// mencari semua resep untuk elemen dengan id  
func FindRecipeFor(elementID int) ([]schema.Recipe, error) {  
    ...  
}  
  
// mencari semua resep yang menggunakan elemen dengan id  
func FindRecipesUsingElement(elementID int) ([]schema.Recipe, error) {  
    ...  
}
```

Fungsi-fungsi ini digunakan untuk menghubungkan backend dan database dengan mudah.

Schema berisi kebanyakan dari struktur data yang digunakan dalam backend, khususnya struktur data yang dipertukarkan antara frontend, backend, dan database. Struktur data yang terkandung dalam schema adalah sebagai berikut.

element.go

```

package schema

import (
    "encoding/json"
    "strings"
)

type Element struct {
    ID      int32 `json:"id"`
    Name    string `json:"name"`
    Tier    int32 `json:"tier"`
    ImageUrl string `json:"image_url"`
}

func (e Element) Serialize() string {
    var w strings.Builder
    json.NewEncoder(&w).Encode(e)
    return w.String()
}

```

recipe.go

```

package schema

import (
    "encoding/json"
    "strings"
)

type Recipe struct {
    ResultID      int32 `json:"result_id"`
    Dependency1ID int32 `json:"dependency1_id"`
    Dependency2ID int32 `json:"dependency2_id"`
}

func (r Recipe) Serialize() string {
    var w strings.Builder
    json.NewEncoder(&w).Encode(r)
    return w.String()
}

```

searchrequest.go

```

package schema

type SearchRequest struct {
    Element    int     `json:"element"`
    Method     string  `json:"method"`
    Count      int     `json:"count"`
    Delay      int     `json:"delay"`
    Threading  string  `json:"threading"`
}

func (sr *SearchRequest) Valid() bool {
    return ((sr.Method == "dfs" || sr.Method == "bfs") &&
        (sr.Count > 0) &&
        (sr.Delay >= 0) &&
        (sr.Threading == "single" || sr.Threading == "multi"))
}

```

```
}
```

searchresponse.go

```
package schema

import (
    "encoding/json"
    "strings"
)

type SearchResponse struct {
    SearchID int `json:"search_id"`
}

func (sr SearchResponse) Serialize() string {
    var w strings.Builder
    json.NewEncoder(&w).Encode(sr)
    return w.String()
}
```

searchresult.go

```
package schema

import (
    "encoding/json"
    "strings"
    "sync"
)

type Combination struct {
    Result      *SearchNode
    Ingredient1 *SearchNode
    Ingredient2 *SearchNode
}

type SearchNode struct {
    Element      Element
    Parent       *Combination
    Dependencies []*Combination
    RecipesFound int
    lock         sync.RWMutex
}

type SearchResult struct {
    Root      *SearchNode
    TimeTaken int
    NodesSearched int
    Finished   bool
    lock       sync.RWMutex
}

type SerializedCombination struct {
    Result      int `json:"result"`
    Dependency1 int `json:"dependency1"`
    Dependency2 int `json:"dependency2"`
}
```

```

type SerializedSearchResult struct {
    Nodes          []int           `json:"nodes"`
    Dependencies  []SerializedCombination `json:"dependencies"`
    TimeTaken     int             `json:"time_taken"`
    NodesSearched int            `json:"nodes_searched"`
    RecipesFound int            `json:"recipes_found"`
    Finished      bool           `json:"finsihed"`
}
...

```

Package ini berisi hasil akhir pohon pencarian yang dibuat oleh program. Selain dari struktur data yang diperlihatkan, terdapat juga method Serialize pada SearchResult yang mengubah SearchResult menjadi SerializedSearchResult dan mengubahnya menjadi bentuk json.

serializable.go

```

package schema

type Serializable interface {
    Serialize() string
}

```

Banyak struktur data dalam schema memiliki method serialize yang digunakan untuk mengubah struktur data ke dalam bentuk json sehingga dapat dikirim menuju frontend.

Findfullrecipe merupakan package yang berisi fungsi-fungsi untuk melakukan pencarian resep. Terdapat viewer yang menyimpan pohon-pohon pencarian yang sedang dikerjakan oleh program serta fungsi-fungsi pencarian menggunakan dfs serta bfs.

viewer.go

```

var searches map[int]*schema.SearchResult =
make(map[int]*schema.SearchResult)

// menyiapkan search baru dalam map searches dengan elemen tertentu
func prepareSearch(element schema.Element) (int, *schema.SearchResult) {
    ...
}

// mengembalikan search yang ada dalam searches
func FindSearch(searchID int) *schema.SearchResult {
    ...
}

// memulai pembersih searches setiap 30 detik
func InitializeSearchCleaner() {
    ...
}

// mematikan pembersih searches
func DeinitializeSearchCleaner() {
    ...
}

```

dfs.go

```
package findfullrecipe

import (
    "math"
    "time"

    "github.com/filbertengyo/Tubes2_gitulah/database"
    "github.com/filbertengyo/Tubes2_gitulah/schema"
)

func WithSinglethreadedDFS(element schema.Element, count int, delay int) int {
    searchID, search := prepareSearch(element)

    go func() {
        start := time.Now()

        singlethreadedDFS(search, search.Root, count, delay)

        search.Lock()
        search.Finished = true
        search.Timetaken = int(time.Since(start).Milliseconds())
        search.Unlock()
    }()

    return searchID
}

func WithMultithreadedDFS(element schema.Element, count int, delay int) int {
    searchID, search := prepareSearch(element)

    go func() {
        start := time.Now()

        channels := multiDFSChannels{
            ready:      make(chan bool),
            finish:     make(chan int),
            redistribute: make(chan int),
            close:      make(chan bool),
        }

        go multithreadedDFS(search, search.Root, channels, delay)

        <-channels.ready
        channels.redistribute <- count
        <-channels.finish
        channels.close <- true

        search.Lock()
        search.Finished = true
        search.Timetaken = int(time.Since(start).Milliseconds())
        search.Unlock()
    }()

    return searchID
}

func singlethreadedDFS(result *schema.SearchResult, node
```

```

*schema.SearchNode, count int, delay int) {
    time.Sleep(time.Duration(delay) * time.Millisecond)

    node.RLock()
    recipes, _ := database.FindRecipeFor(int(node.Element.ID))
    node.RUnlock()

    if len(recipes) == 0 {
        node.RecipesFound = 1
        updateRecipeCounts(node)
        return
    }

    node.RLock()
    for i := 0; i < len(recipes) && node.RecipesFound < count; i++ {
        node.RUnlock()

        result.Lock()
        result.NodesSearched++
        result.Unlock()

        ingredient1, _ :=
database.FindElementById(int(recipes[i].Dependency1ID))
        ingredient2, _ :=
database.FindElementById(int(recipes[i].Dependency2ID))
        combination := schema.Combination{
            Result:      node,
            Ingredient1: &schema.SearchNode{Element:
ingredient1},
            Ingredient2: &schema.SearchNode{Element:
ingredient2},
        }
        combination.Ingredient1.Parent = &combination
        combination.Ingredient2.Parent = &combination

        node.Lock()
        node.Dependencies = append(node.Dependencies,
&combination)
        node.Unlock()

        singlethreadedDFS(result, combination.Ingredient1, count,
delay)

        combination.Ingredient1.RLock()
        adjacentCount :=
max(count/combination.Ingredient1.RecipesFound, 1)
        if count%combination.Ingredient1.RecipesFound > 0 &&
count/combination.Ingredient1.RecipesFound > 0 {
            adjacentCount++
        }
        combination.Ingredient1.RUnlock()

        singlethreadedDFS(result, combination.Ingredient2,
adjacentCount, delay)

        node.RLock()
    }
    node.RUnlock()
}

type multiDFSChannels struct {
    ready      chan bool
    finish     chan int
    redistribute chan int
}

```

```

        close      chan bool
    }

func multithreadedDFS(result *schema.SearchResult, node
*schema.SearchNode, channels multiDFSChannels, delay int) {
    time.Sleep(time.Duration(delay) * time.Millisecond)

    node.RLock()
    recipes, _ := database.FindRecipeFor(int(node.Element.ID))
    node.RUnlock()

    channels.ready <- true
    quota := <-channels.redistribute

    if len(recipes) == 0 {
        node.RecipesFound = 1
        channels.finish <- node.RecipesFound

        ok := true
        for ok {
            select {
            case <-channels.redistribute:
                channels.finish <- node.RecipesFound
            case <-channels.close:
                ok = false
            }
        }
        return
    }

    for node.RecipesFound >= quota {
        channels.finish <- node.RecipesFound
        select {
        case quota = <-channels.redistribute:
        case <-channels.close:
            return
        }
    }

    node.RLock()
    for i := 0; i < len(recipes); i++ {
        node.RUnlock()

        result.Lock()
        result.NodesSearched++
        result.Unlock()

        ingredient1, _ :=
database.FindElementById(int(recipes[i].Dependency1ID))
        ingredient2, _ :=
database.FindElementById(int(recipes[i].Dependency2ID))
        combination := schema.Combination{
            Result:      node,
            Ingredient1: &schema.SearchNode{Element:
ingredient1},
            Ingredient2: &schema.SearchNode{Element:
ingredient2},
        }
        combination.Ingredient1.Parent = &combination
        combination.Ingredient2.Parent = &combination

        node.Lock()
        node.Dependencies = append(node.Dependencies,
&combination)
    }
}

```

```

        node.Unlock()

        leftChannels := multiDFSChannels{
            ready:      make(chan bool),
            finish:     make(chan int),
            redistribute: make(chan int),
            close:      make(chan bool),
        }

        rightChannels := multiDFSChannels{
            ready:      make(chan bool),
            finish:     make(chan int),
            redistribute: make(chan int),
            close:      make(chan bool),
        }

        go multithreadedDFS(result, combination.Ingredient1,
leftChannels, delay)
        go multithreadedDFS(result, combination.Ingredient2,
rightChannels, delay)

        <-leftChannels.ready
        <-rightChannels.ready

        for {
            node.RLock()
            leftDistribution :=
int(math.Ceil(math.Sqrt(float64(quota - node.RecipesFound))))
            node.RUnlock()

            rightDistribution := quota / leftDistribution
            if quota%leftDistribution > 0 {
                rightDistribution++
            }

            ldr := leftDistribution
            rdr := rightDistribution

            leftChannels.redistribute <- ldr
            rightChannels.redistribute <- rdr

            leftFound := <-leftChannels.finish
            rightFound := <-rightChannels.finish

            if leftFound < ldr && rightFound >= rdr {
                rdr = quota / leftFound
                if quota%leftFound > 0 {
                    rdr++
                }
                rightChannels.redistribute <- rdr
                rightFound = <-rightChannels.finish
            } else if rightFound < rdr && leftFound >= ldr {
                ldr = quota / rightFound
                if quota%rightFound > 0 {
                    ldr++
                }
                leftChannels.redistribute <- ldr
                leftFound = <-leftChannels.finish
            }

            node.Lock()
            node.RecipesFound += leftFound * rightFound
            node.Unlock()
        }
    }
}

```

```

        if leftFound < ldr && rightFound < rdr {
            break
        }

        channels.finish <- node.RecipesFound

        select {
        case quota = <-channels.redistribute:
            node.Lock()
            node.RecipesFound -= leftFound *
rightFound
            node.Unlock()
        case <-channels.close:
            leftChannels.close <- true
            rightChannels.close <- true
            return
        }
    }

    leftChannels.close <- true
    rightChannels.close <- true

    node.RLock()
}
node.RUnlock()

channels.finish <- node.RecipesFound

ok := true
for ok {
    select {
    case <-channels.redistribute:
        channels.finish <- node.RecipesFound
    case <-channels.close:
        ok = false
    }
}
}

```

Fungsi dfs singlethreaded melakukan pencarian secara rekursif pada thread yang sama. Untuk setiap kombinasi yang diproses fungsi DFS singlethreaded, resep untuk cabang kiri ditentukan terlebih dahulu kemudian dihitung jumlah resepnya untuk menentukan target resep pada cabang kanan. Fungsi DFS multithreaded juga melakukan hal yang serupa dengan awalnya memberi kuota yang sama pada kedua cabang, tetapi dan mendistribusi kembali kuota resep yang dicari untuk menyesuaikan. Kedua fungsi rekursif berakhir ketika bertemu dengan elemen yang tidak memiliki resep yang membuat jumlah resep yang ditemukan pada elemen tersebut menjadi 1. Jumlah resep pada elemen lain untuk setiap kombinasi dapat ditentukan dengan jumlah resep pada cabang kiri dikali dengan jumlah resep pada cabang kanan. Total resep yang dimiliki satu elemen dihitung sebagai jumlah total resep pada semua kombinasi.

bfs.go

```

package findfullrecipe

import (
    "sync"

```

```

        "time"

        "github.com/filbertengyo/Tubes2_gitulah/database"
        "github.com/filbertengyo/Tubes2_gitulah/schema"
    )

func WithSinglethreadedBFS(element schema.Element, count int, delay int)
int {
    searchID, search := prepareSearch(element)

    go func() {
        start := time.Now()

        singlethreadedBFS(search, count, delay)
        cleanupInvalidCombinations(search.Root)

        search.Lock()
        search.Timetaken = int(time.Since(start).Milliseconds())
        search.Finished = true
        search.Unlock()
    }()

    return searchID
}

func WithMultithreadedBFS(element schema.Element, count int, delay int)
int {
    searchID, search := prepareSearch(element)

    go func() {
        start := time.Now()

        var wg sync.WaitGroup
        setup := make(chan bool)

        go multithreadedBFS(search, search.Root, count, count,
delay, &wg, setup)
        <-setup
        wg.Wait()
        cleanupInvalidCombinations(search.Root)

        search.Lock()
        search.Timetaken = int(time.Since(start).Milliseconds())
        search.Finished = true
        search.Unlock()
    }()

    return searchID
}

func multithreadedBFS(result *schema.SearchResult, node
*schema.SearchNode, count int, topCount int, delay int, wg
*sync.WaitGroup, setup chan bool) {
    wg.Add(1)
    defer wg.Done()
    setup <- true

    time.Sleep(time.Duration(delay) * time.Millisecond)

    node.RLock()
    recipes, _ := database.FindRecipeFor(int(node.Element.ID))
    node.RUnlock()

    if len(recipes) == 0 {

```

```

        node.RecipesFound = 1
        updateRecipeCounts(node)
        return
    }

    result.Root.RLock()
    rootContinue := result.Root.RecipesFound < topCount
    result.Root.RUnlock()

    node.RLock()
    for i := 0; i < len(recipes) && i < count && node.RecipesFound <
count && rootContinue; i++ {
        node.RUnlock()

        result.Lock()
        result.NodesSearched++
        result.Unlock()

        ingredient1, _ :=
database.FindElementById(int(recipes[i].Dependency1ID))
        ingredient2, _ :=
database.FindElementById(int(recipes[i].Dependency2ID))
        combination := schema.Combination{
            Result:      node,
            Ingredient1: &schema.SearchNode{Element:
ingredient1},
            Ingredient2: &schema.SearchNode{Element:
ingredient2},
        }
        combination.Ingredient1.Parent = &combination
        combination.Ingredient2.Parent = &combination

        node.Lock()
        node.Dependencies = append(node.Dependencies,
&combination)
        node.Unlock()

        setup1 := make(chan bool)
        setup2 := make(chan bool)

        go multithreadedBFS(result, combination.Ingredient1,
max(count/len(recipes)/2, 1), topCount, delay, wg, setup1)
        go multithreadedBFS(result, combination.Ingredient2,
max(count/len(recipes)/2, 1), topCount, delay, wg, setup2)

        <-setup1
        <-setup2

        result.Root.RLock()
        rootContinue = result.Root.RecipesFound < topCount
        result.Root.RUnlock()

        node.RLock()
    }
    node.RUnlock()
}

func singlethreadedBFS(result *schema.SearchResult, count int, delay
int) {
    result.RLock()
    nodes := []*schema.SearchNode{result.Root}
    result.RUnlock()

    iterations := 0
}

```

```

        result.Root.RLock()
        for len(nodes) > 0 && result.Root.RecipesFound < count {
            result.Root.RUnlock()

            nextNodes := []*schema.SearchNode{}

            result.Root.RLock()
            for i := 0; i < len(nodes) && result.Root.RecipesFound <
count; i++ {
                result.Root.RUnlock()

                nodes[i].RLock()
                recipes, _ :=
database.FindRecipeFor(int(nodes[i].Element.ID))
                nodes[i].RUnlock()

                if len(recipes) == 0 {
                    nodes[i].Lock()
                    nodes[i].RecipesFound = 1
                    nodes[i].Unlock()

                    updateRecipeCounts(nodes[i])
                }

                result.Root.RLock()
                rootContinue := result.Root.RecipesFound < count
                result.Root.RUnlock()

                nodes[i].RLock()
                nodeContinue := nodes[i].RecipesFound <
max((count>>iterations)/len(recipes), 1)
                nodes[i].RUnlock()

                for j := 0; j < len(recipes) && j <
max((count>>iterations)/len(recipes), 1) && rootContinue &&
nodeContinue; j++ {
                    time.Sleep(time.Duration(delay) *
time.Millisecond)

                    result.Lock()
                    result.NodesSearched++
                    result.Unlock()

                    ingredient1, _ :=
database.FindElementById(int(recipes[j].Dependency1ID))
                    ingredient2, _ :=
database.FindElementById(int(recipes[j].Dependency2ID))
                    combination := schema.Combination{
                        Result:      nodes[i],
                        Ingredient1:
&schema.SearchNode{Element: ingredient1},
                        Ingredient2:
&schema.SearchNode{Element: ingredient2},
                    }
                    combination.Ingredient1.Parent =
&combination
                    combination.Ingredient2.Parent =
&combination

                    nodes[i].Lock()
                    nodes[i].Dependencies =
append(nodes[i].Dependencies, &combination)
                    nodes[i].Unlock()
                }
            }
        }
    }
}

```

```

                nextNodes = append(nextNodes,
combination.Ingredient1, combination.Ingredient2)

                result.Root.RLock()
rootContinue = result.Root.RecipesFound <
count
                result.Root.RUnlock()

                nodes[i].RLock()
nodeContinue = nodes[i].RecipesFound <
max((count>>iterations)/len(recipes), 1)
                nodes[i].RUnlock()
            }

            result.Root.RLock()
        }
        result.Root.RUnlock()

        nodes = nextNodes
iterations++

        result.Root.RLock()
    }
    result.Root.RUnlock()
}

func cleanupInvalidCombinations(node *schema.SearchNode) {
    node.RLock()
    if node.RecipesFound == 0 {
        if node.Parent != nil {
            result := node.Parent.Result
            combination := node.Parent

            result.Lock()
newDeps := []*schema.Combination{}
for _, dep := range result.Dependencies {
            if dep != combination {
                newDeps = append(newDeps, dep)
            }
}
result.Dependencies = newDeps
            result.Unlock()
        }
    } else {
        for _, combination := range node.Dependencies {

cleanupInvalidCombinations(combination.Ingredient1)

cleanupInvalidCombinations(combination.Ingredient2)
        }
    }
    node.RUnlock()
}

```

Fungsi BFS singlethreaded menyimpan daftar semua node yang akan dikunjungi berikutnya. Fungsi ini melakukan iterasi pada node yang dikunjungi berikutnya dan menggantinya dengan daftar node baru hingga tidak ada node lagi yang ditambahkan. Untuk setiap node yang dikunjungi, jika node tersebut memiliki resep, fungsi akan menambahkan elemen pada resep tersebut dalam daftar node yang perlu dikunjungi, jika

tidak maka node tersebut dianggap telah memiliki total resep 1 yang dipropagasi ke node-node di atasnya. Sementara itu, BFS multithreaded mirip dengan DFS singlethreaded tetapi setiap cabang menjalankan thread sendiri dan semua cabang pada semua resep hingga batas tertentu pada suatu elemen dijalankan sekaligus. Setelah menjalankan BFS baik singlethreaded maupun multithreaded, dipanggil fungsi untuk membersihkan node-node yang tidak menghasilkan resep.

```
recipecounter.go
```

```
package findfullrecipe

import "github.com/filbertengyo/Tubes2_gitulah/schema"

func updateRecipeCounts(node *schema.SearchNode) {
    node.RLock()
    combination := node.Parent

    if combination == nil {
        node.RUnlock()
        return
    }

    result := combination.Result
    node.RUnlock()

    result.Lock()
    recipeCount := 0

    for i := 0; i < len(result.Dependencies); i++ {
        result.Dependencies[i].Ingredient1.RLock()
        recipeCount1 :=
    result.Dependencies[i].Ingredient1.RecipesFound
        result.Dependencies[i].Ingredient1.RUnlock()

        result.Dependencies[i].Ingredient2.RLock()
        recipeCount2 :=
    result.Dependencies[i].Ingredient2.RecipesFound
        result.Dependencies[i].Ingredient2.RUnlock()

        recipeCount += recipeCount1 * recipeCount2
    }

    result.RecipesFound = recipeCount
    result.Unlock()

    updateRecipeCounts(result)
}
```

Fungsi ini digunakan untuk melakukan propagasi jumlah resep yang ditemukan dari *child node* ke *parent node*-nya.

Middleware merupakan package yang berisi middleware agar backend dapat diakses oleh frontend tanpa menyalahi CORS. Package middleware hanya berisi satu fungsi sebagai berikut.

```
corsmiddleware.go
```

```
package middleware
```

```

import (
    "net/http"
    "os"
)

// CORSMiddleware adds the necessary headers for CORS
func CORSMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r
    *http.Request) {
        // Set CORS headers
        w.Header().Set("Access-Control-Allow-Origin",
os.Getenv("FRONTEND_PUBLIC_URL")) // Allow any origin
        w.Header().Set("Access-Control-Allow-Methods", "GET,
POST, OPTIONS")
        w.Header().Set("Access-Control-Allow-Headers",
"Content-Type, Authorization")

        // Handle OPTIONS request for preflight checks
        if r.Method == "OPTIONS" {
            w.WriteHeader(http.StatusOK)
            return
        }

        // Call the next handler
        next.ServeHTTP(w, r)
    })
}

```

Route merupakan package yang berisi method untuk mengelola semua API route yang disediakan oleh backend. Berikut adalah fungsi-fungsi yang terdapat dalam package route.

element.go
<pre> package route import ("bytes" "encoding/json" "net/http" "strconv" "github.com/filbertengyo/Tubes2_gitulah/database") func Element(w http.ResponseWriter, r *http.Request) { queries := r.URL.Query() if val, ok := queries["type"]; ok { if len(val) == 0 { w.WriteHeader(http.StatusBadRequest) } else if val[0] == "name" { elementByName(w, r) } else if val[0] == "id" { elementById(w, r) } else { w.WriteHeader(http.StatusBadRequest) } } else { elementById(w, r) } } </pre>

```

        }

func elementById(w http.ResponseWriter, r *http.Request) {
    identifier, err := strconv.ParseInt(r.PathValue("identifier"),
10, 32)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    val, err := database.FindElementById(int(identifier))

    if err != nil {
        w.WriteHeader(http.StatusNotFound)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write([]byte(val.Serialize()))
}

func elementByName(w http.ResponseWriter, r *http.Request) {
    name := r.PathValue("identifier")

    val, err := database.FindElementByName(name)

    if err != nil {
        w.WriteHeader(http.StatusNotFound)
        return
    }

    var body bytes.Buffer
    json.NewEncoder(&body).Encode(val)

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write(body.Bytes())
}

```

elements.go

```

package route

import (
    "bytes"
    "encoding/json"
    "net/http"
    "strconv"

    "github.com/filbertengyo/Tubes2_gitulah/database"
    "github.com/filbertengyo/Tubes2_gitulah/schema"
)

func Elements(w http.ResponseWriter, r *http.Request) {
    queries := r.URL.Query()

    var err error

```

```

var start uint64 = 0
var end uint64 = 20
tiers := []int{}

if str, ok := queries["start"]; ok {
    if len(str) > 0 {
        start, err = strconv.ParseUint(str[0], 10, 32)
    }
}

if str, ok := queries["end"]; ok {
    if len(str) > 0 {
        end, err = strconv.ParseUint(str[0], 10, 32)
    }
}

if strs, ok := queries["tiers"]; ok {
    for i := range strs {
        if tier, _err := strconv.ParseUint(strs[i], 10,
32); err != nil {
            err = _err
            break
        } else {
            tiers = append(tiers, int(tier))
        }
    }
}

if err != nil {
    w.WriteHeader(http.StatusBadRequest)
    return
}

if end < start {
    w.WriteHeader(http.StatusBadRequest)
    return
}

query := []schema.Element{}

if len(tiers) > 0 {
    for i := range tiers {
        q, _err := database.FindElementInTier(int(start),
int(end), tiers[i])

        if _err != nil {
            err = _err
            break
        }

        query = append(query, q...)
    }
} else {
    query, err = database.Elements(int(start), int(end))
}

if err != nil {
    w.WriteHeader(http.StatusInternalServerError)
}
}

var body bytes.Buffer
json.NewEncoder(&body).Encode(query)

w.WriteHeader(http.StatusOK)

```

```
w.Header().Add("Content-Type", "application/json")
w.Write(body.Bytes())
}
```

getfullrecipe.go

```
package route

import (
    "net/http"
    "strconv"

    "github.com/filbertengyo/Tubes2_gitulah/service/findfullrecipe"
)

func GetFullRecipe(w http.ResponseWriter, r *http.Request) {
    identifier, err := strconv.ParseInt(r.PathValue("identifier"),
10, 32)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    searchResult := findfullrecipe.FindSearch(int(identifier))

    if searchResult == nil {
        w.WriteHeader(http.StatusNotFound)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write([]byte(searchResult.Serialize()))
}
```

immediatefullrecipe.go

```
package route

import (
    "encoding/json"
    "io"
    "net/http"
    "time"

    "fmt"

    "github.com/filbertengyo/Tubes2_gitulah/schema"
    "github.com/filbertengyo/Tubes2_gitulah/service/findfullrecipe"
)

func ImmediateFullRecipe(w http.ResponseWriter, r *http.Request) {
    resp, err := http.Post("http://localhost:5761/fullrecipe/",
"application/json", r.Body)

    if err != nil {
```

```

        w.WriteHeader(resp.StatusCode)
        return
    }

    var searchResponse schema.SearchResponse
    err = json.NewDecoder(resp.Body).Decode(&searchResponse)
    resp.Body.Close()

    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    for !findfullrecipe.FindSearch(searchResponse.SearchID).Finished
    {
        time.Sleep(1 * time.Millisecond)
    }

    resp, err = http.Get("http://localhost:5761/fullrecipe/" +
fmt.Sprint(searchResponse.SearchID))

    if err != nil {
        w.WriteHeader(resp.StatusCode)
        return
    }

    body, err := io.ReadAll(resp.Body)

    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write(body)
}

```

postfullrecipe.go

```

package route

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

    "github.com/filbertengyo/Tubes2_gitulah/database"
    "github.com/filbertengyo/Tubes2_gitulah/schema"
    "github.com/filbertengyo/Tubes2_gitulah/service/findfullrecipe"
)

func PostFullRecipe(w http.ResponseWriter, r *http.Request) {
    var request schema.SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&request); err != nil {
        w.WriteHeader(http.StatusBadGateway)
        return
    }

    if !request.Valid() {
        w.WriteHeader(http.StatusBadRequest)
    }
}

```

```

        return
    }

    element, err := database.FindElementById(request.Element)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    var response schema.SearchResponse

    if request.Method == "dfs" && request.Threading == "single" {
        response.SearchID =
findfullrecipe.WithSinglethreadedDFS(element, request.Count,
request.Delay)
    } else if request.Method == "dfs" && request.Threading == "multi"
{
        response.SearchID =
findfullrecipe.WithMultithreadedDFS(element, request.Count,
request.Delay)
    } else if request.Method == "bfs" && request.Threading ==
"single" {
        response.SearchID =
findfullrecipe.WithSinglethreadedBFS(element, request.Count,
request.Delay)
    } else if request.Method == "bfs" && request.Threading == "multi"
{
        response.SearchID =
findfullrecipe.WithMultithreadedBFS(element, request.Count,
request.Delay)
        //} else if request.Method == "bidirectional" &&
request.Threading == "single" {
        //      response.SearchID =
findfullrecipe.WithSinglethreadedBidirectional(element, request.Count,
request.Delay)
        //} else if request.Method == "bidirectional" &&
request.Threading == "multi" {

    } else {
        w.WriteHeader(http.StatusNotImplemented)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write([]byte(response.Serialize()))
}

```

recipe.go

```

package route

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

    "github.com/filbertengyo/Tubes2_gitulah/database"
    "github.com/filbertengyo/Tubes2_gitulah/schema"
    "github.com/filbertengyo/Tubes2_gitulah/service/findfullrecipe"

```

```

    )

func PostFullRecipe(w http.ResponseWriter, r *http.Request) {
    var request schema.SearchRequest
    if err := json.NewDecoder(r.Body).Decode(&request); err != nil {
        w.WriteHeader(http.StatusBadGateway)
        return
    }

    if !request.Valid() {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    element, err := database.FindElementById(request.Element)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    var response schema.SearchResponse

    if request.Method == "dfs" && request.Threading == "single" {
        response.SearchID =
findfullrecipe.WithSinglethreadedDFS(element, request.Count,
request.Delay)
    } else if request.Method == "dfs" && request.Threading == "multi"
{
        response.SearchID =
findfullrecipe.WithMultithreadedDFS(element, request.Count,
request.Delay)
    } else if request.Method == "bfs" && request.Threading ==
"single" {
        response.SearchID =
findfullrecipe.WithSinglethreadedBFS(element, request.Count,
request.Delay)
    } else if request.Method == "bfs" && request.Threading == "multi"
{
        response.SearchID =
findfullrecipe.WithMultithreadedBFS(element, request.Count,
request.Delay)
    //} else if request.Method == "bidirectional" &&
request.Threading == "single" {
        //      response.SearchID =
findfullrecipe.WithSinglethreadedBidirectional(element, request.Count,
request.Delay)
    //} else if request.Method == "bidirectional" &&
request.Threading == "multi" {

    } else {
        w.WriteHeader(http.StatusNotImplemented)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Header().Add("Content-Type", "application/json")
    w.Write([]byte(response.Serialize()))
}

```

Semua package disatukan dalam file [main.go](#) yang melakukan inisialisasi backend serta semua API routes. Berikut adalah isi dari file [main.go](#).

main.go

```
package main

import (
    "fmt"
    "net/http"
    "os"
    "time"

    "github.com/filbertengyo/Tubes2_gitulah/database"
    "github.com/filbertengyo/Tubes2_gitulah/service/findfullrecipe"
    "github.com/filbertengyo/Tubes2_gitulah/service/middleware"
    "github.com/filbertengyo/Tubes2_gitulah/service/route"
)

func main() {
    time.Sleep(5 * time.Second)
    fmt.Println("Hello World!")

    if err := database.Initialize(); err != nil {
        fmt.Println("An error occurred during initialization!")
        fmt.Println(err.Error())
        os.Exit(1)
    }
    defer database.Close()

    if !database.isDefined() {
        database.Define()

        if err := database.Seed(); err != nil {
            fmt.Println("An error occurred during seeding!")
            fmt.Println(err.Error())
            os.Exit(1)
        }
    }

    findfullrecipe.InitializeSearchCleaner()
    defer findfullrecipe.DeinitializeSearchCleaner()

    http.HandleFunc("/",
        func(w http.ResponseWriter, r
        *http.Request) {
            q := r.URL.Query()
            if val, ok := q["msg"]; ok {
                fmt.Fprintf(w, "Echo: %v", val[0])
            } else {
                fmt.Fprintf(w, "Hello World!")
            }
        })
}

http.HandleFunc("/elements", route.Elements)

http.HandleFunc("/elements/{identifier}", route.Element)

http.HandleFunc("/elements/{identifier}/recipe",
    route.Recipe)

http.HandleFunc("POST /fullrecipe/", route.PostFullRecipe)

http.HandleFunc("GET /fullrecipe/immediate",
    route.ImmediateFullRecipe)

http.HandleFunc("GET /fullrecipe/{identifier}",
    route.GetFullRecipe)
```

```

    handler := middleware.CORSMiddleware(http.DefaultServeMux)

    port := os.Getenv("BACKEND_HOST_PORT")

    if err := http.ListenAndServe(": "+port, handler); err != nil {
        fmt.Println("An error occurred!")
        fmt.Println(err.Error())
    }

    fmt.Println("Goodbye World!")
}

```

4.1.3. Database

Program ini menggunakan database berupa postgres untuk menyimpan informasi tentang elemen-elemen serta resep-resep dalam permainan Little Alchemy 2. Untuk itu, database ini berisi dua tabel, yaitu *elements* dan *recipes*. *Elements* mengandung data elemen dengan kolom *id*, *name*, *tier*, dan *image_url*. Sementara itu, *recipes* mengandung data resep yang berisi kolom *result_id*, *dependency1_id*, dan *dependency2_id*. Semua informasi pada database didapatkan dari hasil scraping pada backend dan disimpan dalam docker volume agar tidak perlu diulangi setiap kali database dinyalakan.

4.2. Tata Cara Penggunaan

Oleh karena website yang dikembangkan di deploy ke server, sehingga proses instalasi requirement tidak perlu dilakukan dan website bisa langsung digunakan. Website dapat digunakan dengan:

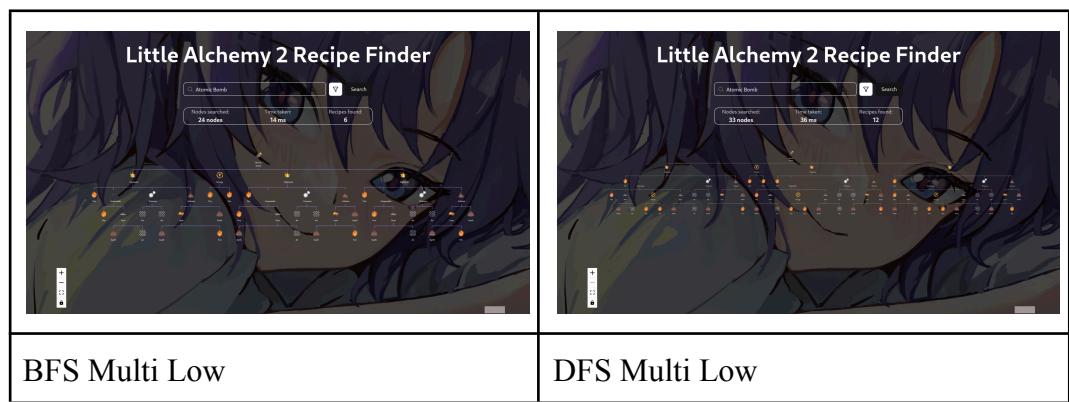
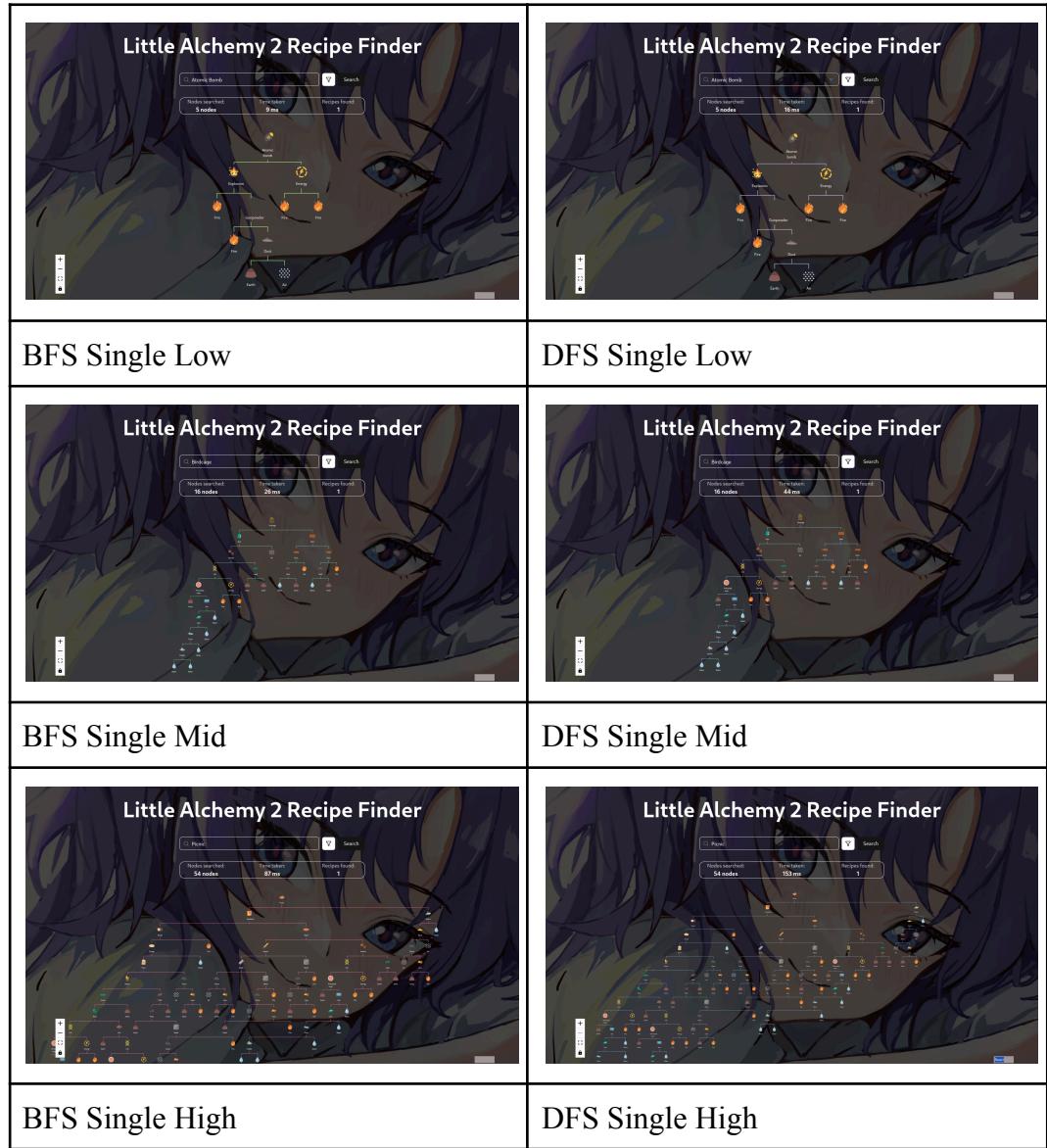
1. Mengunjungi laman website
2. Memasukan nama elemen / item yang diinginkan pada kotak yang telah disediakan.
3. Lalu memilih metode yang diinginkan untuk melakukan pencarian.
4. Apabila memilih opsi multithreading dengan multirecipe, diperlukan masukan tambahan berupa delay time dalam ms dan jumlah recipe.
5. Lalu tekan tombol “search”.
6. Tunggu sampai pohon berhasil ditampilkan.
7. Pengguna dapat menggeser atau memperbesar tampilan hasil berdasarkan navigator yang ditentukan.

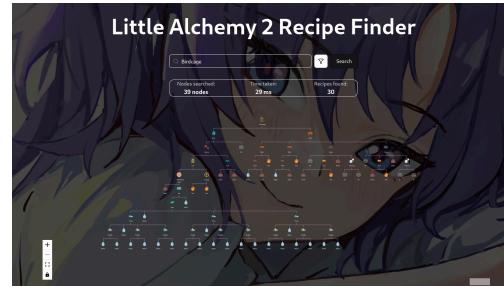
4.3. Hasil Pengujian

Pengujian dilakukan untuk memverifikasi kebenaran program dalam menentukan resep serta kecepatan jalan program. Berikut adalah hasil pengujian untuk beberapa elemen yang dilakukan pencarian.

Untuk multithreading seluruhnya menggunakan count sebesar 10. Hasil yang didapatkan dapat bervariasi tergantung berapa banyak yang didapatkan oleh algoritma, dengan 10 sebagai pendekatan.

Dengan tingkatan Low yaitu Atomic Bomb (Tier 4), tingkatan Mid yaitu Birdcage (Tier 9), dan tingkatan High yaitu Picnic (Tier 15).



	
BFS Multi Mid	DFS Multi Mid
	
BFS Multi High	DFS Multi High

4.4. Analysis Hasil

Berdasarkan pengujian yang telah dilakukan, program telah diimplementasikan dengan benar sehingga baik DFS maupun BFS dapat menghasilkan resep yang tepat. Walaupun keduanya menghasilkan resep yang valid, resep yang dihasilkan oleh kedua algoritma tetap berbeda. Hal tersebut dikarenakan cara penelusuran kedua algoritma berbeda. BFS akan mencoba semua resep alternatif sekaligus, sementara DFS fokus pada satu resep hingga selesai sebelum lanjut ke resep lain. Perbedaan proses pencarian tersebut menyebabkan resep hasil DFS terlihat lebih dalam, sementara resep hasil BFS terlihat lebih lebar.

Terdapat juga hal yang menarik dengan penerapan multithreading baik pada DFS maupun BFS. Untuk resep elemen dengan tier yang rendah, *multithreading* mengurangi kecepatan dari algoritma. Hal tersebut dikarenakan elemen dengan tier rendah cenderung memiliki resep yang lebih kecil dan sederhana, sehingga keuntungan dari multithreading tidak dapat melebihi *overhead* yang ditambahnya. Namun, *multithreading* mempercepat algoritma pada elemen dengan tier yang lebih tinggi. Hal tersebut dikarenakan elemen-elemen dengan tier tinggi memiliki resep yang besar dan mengandung banyak *node* sehingga kemampuan memproses lebih dari satu *node* sekaligus membuat algoritma berjalan lebih cepat.

Selain itu, perbedaan waktu yang diperlukan antara DFS dan BFS juga menarik. Untuk elemen tier rendah, BFS cenderung lebih cepat daripada DFS. Namun, untuk elemen tier tinggi, BFS menjadi jauh lebih pelan. Alasan hal ini terjadi adalah BFS kesulitan untuk mencari resep ketika elemen yang dicari memiliki banyak resep alternatif. BFS perlu memperhitungkan semua resep alternatif dan tidak bisa fokus pada satu resep saja. Sementara itu, DFS memilih untuk fokus

pada satu resep tertentu hingga selesai menelusuri resep itu sebelum pindah pada resep lain. Hal tersebut membuat DFS lebih efektif pada resep yang lebih kompleks.

5. Kesimpulan dan Saran

5.1. Kesimpulan

Kesimpulan dari pengembangan algoritma *BFS* dan *DFS* untuk pencarian elemen pada permainan Little Alchemy 2:

1. Algoritma *BFS* dan *DFS* cukup efektif untuk diimplementasikan dalam pencarian permainan Little Alchemy 2. Hal ini dikarenakan dengan *BFS* dan *DFS* kita dapat mencari kombinasi suatu elemen secara efisien, tetapi tetap perlu memerhatikan kondisi *use case*.
2. Algoritma *DFS* menunjukkan performa yang optimal dalam mencari suatu resep kombinasi elemen dengan tier yang tinggi karena prosesnya mencari sampai dasar baru melakukan *backtracking*.
3. Algoritma *BFS* menunjukkan performa yang optimal dalam mencari banyak resep pada suatu elemen yang memang memiliki kombinasi resep yang banyak, tetapi kurang optimal untuk tier yang tinggi.
4. Penggunaan *multithreading* dalam algoritma pencarian *BFS* maupun *DFS* dapat mempercepat jalannya program apabila ukuran resep cukup besar karena proses dialokasikan secara lebih baik.

5.2. Saran

Terdapat banyak hal mengenai implementasi algoritma *BFS* dan *DFS* yang masih dapat dikembangkan dalam tugas besar ini. Khususnya, implementasi *multithreading* pada tugas besar ini masih dapat dikembangkan sehingga lebih cepat dan juga lebih tepat dalam menghasilkan jumlah resep. Terdapat juga beberapa tambahan seperti algoritma *bidirectional* yang seharusnya bisa ditambahkan sebagai metode pencarian tetapi tidak dapat diimplementasikan tepat waktu. Implementasi fitur *Live Update* juga dapat dikembangkan sehingga tidak memerlukan *polling* dari *frontend*, melainan memampukan *backend* untuk menotifikasi *frontend* melalui SSE atau *Websocket*.

Lampiran

- Tabel penyelesaian tugas besar

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

- Tautan repository

https://github.com/filbertengyo/Tubes2_gitulah

- Tautan hosting website

<http://103.125.181.41:5761>

Daftar Pustaka

1. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf)
2. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf)
3. <https://fikti.umsu.ac.id/algoritma-bfs-breadth-first-search-pengertian-fungsi-dan-cara-kerja/>
4. <https://dikakaryatech.com/software/development/2024/05/01/breadth-first-search-BFS-penjelajahan-struktur-data-yang-efisien-dan-mendalam.html>
5. <https://www.trivusi.web.id/2022/05/apa-itu-algoritma-depth-first-search.html>
6. https://en.wikipedia.org/wiki/Bidirectional_search
7. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Makalah/stima2020k3-008.pdf>