

TUGAS BESAR 2
STRATEGI ALGORITMA IF2211
Pemanfaatan Algoritma BFS dan DFS dalam
Pencarian Recipe pada Permainan
Little Alchemy 2



Kelompok 39
H-1 Kelar

Felix Chandra	13523012
Kenneth Ricardo Chandra	13523022
Richard Christian	13523024

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

2025

DAFTAR ISI

BAB I	
DESKRIPSI TUGAS	3
BAB II	
LANDASAN TEORI	5
A. Graf dan Penjelajahan Graf	5
B. Algoritma Breadth First Search	5
C. Algoritma Depth First Search	6
D. Penjelasan Aplikasi Web	7
BAB III	
ANALISIS PEMECAHAN MASALAH	9
A. Langkah-langkah Pemecahan Masalah	9
B. Pemetaan Masalah ke Elemen-elemen Algoritma	9
C. Fitur Fungsional dan Arsitektur Aplikasi Web	10
D. Contoh ilustrasi kasus.	10
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	11
A. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).	11
B. Penjelasan tata cara penggunaan program	15
C. Hasil pengujian	16
D. Analisis hasil pengujian.	22
BAB V	
KESIMPULAN, SARAN, DAN REFLEKSI	23
A. Kesimpulan	23
B. Saran	23
C. Refleksi	23
LAMPIRAN	25
A. Checklist Pengerjaan	25
B. Tautan Repository Github	25
DAFTAR PUSTAKA	26

BAB I

DESKRIPSI TUGAS



Gambar 1. Little Alchemy 2
(sumber: <https://www.thegamer.com>)

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 2. Elemen dasar pada Little Alchemy 2

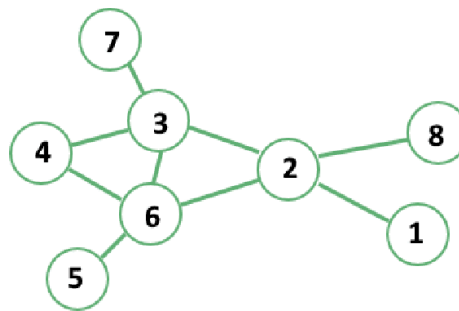
- Elemen turunan
Terdapat 720 elemen turunan yang dibagi menjadi beberapa *tier* tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki *recipe* yang terdiri atas elemen lainnya atau elemen itu sendiri.
- *Combine Mechanism*
Untuk mendapatkan elemen turunan 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.

BAB II

LANDASAN TEORI

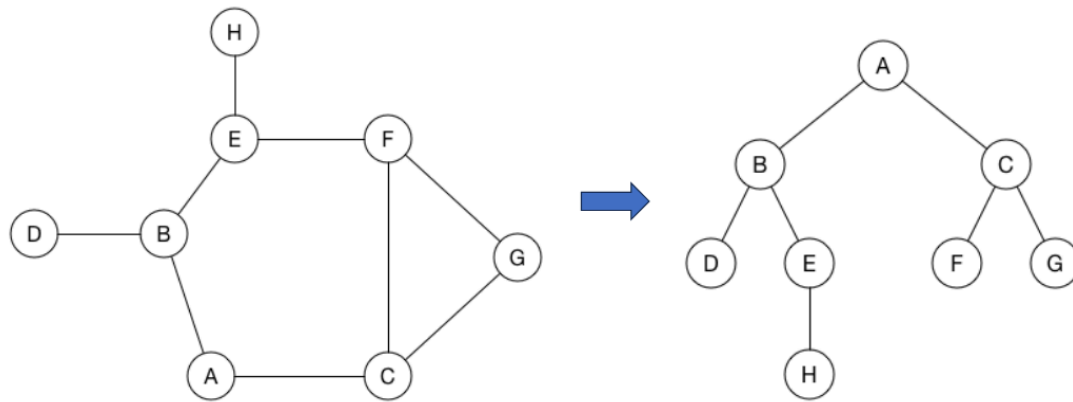
A. Graf dan Penjelajahan Graf

Graf adalah struktur data yang merepresentasikan hubungan antar objek. Objek-objek yang dihubungkan disebut dengan simpul, dan hubungan antara simpul-simpul disebut dengan sisi. Graf biasanya digunakan untuk dapat merepresentasikan berbagai masalah pada dunia nyata. Salah satu contoh masalah yang akan digunakan dalam tugas besar ini adalah pemetaan resep Little Alchemy. Graf yang digunakan untuk traversal biasanya berbentuk graf terhubung, dan dapat dilakukan baik dengan statis maupun dinamis.



B. Algoritma Breadth First Search

Algoritma Breadth First Search (BFS) adalah algoritma penjelajahan graf yang melakukan penjelajahan tanpa informasi (*blind search*). Ini berarti tidak ada informasi tambahan yang disediakan pada awal pencarian. Algoritma BFS bekerja dengan memulai dari simpul awal, lalu menjelajahi seluruh simpul tetangga sebelum melanjutkan ke kedalaman simpul selanjutnya.



Urutan simpul-simpul yang dikunjungi secara BFS dari A \rightarrow A, B, C, D, E, F, G, H

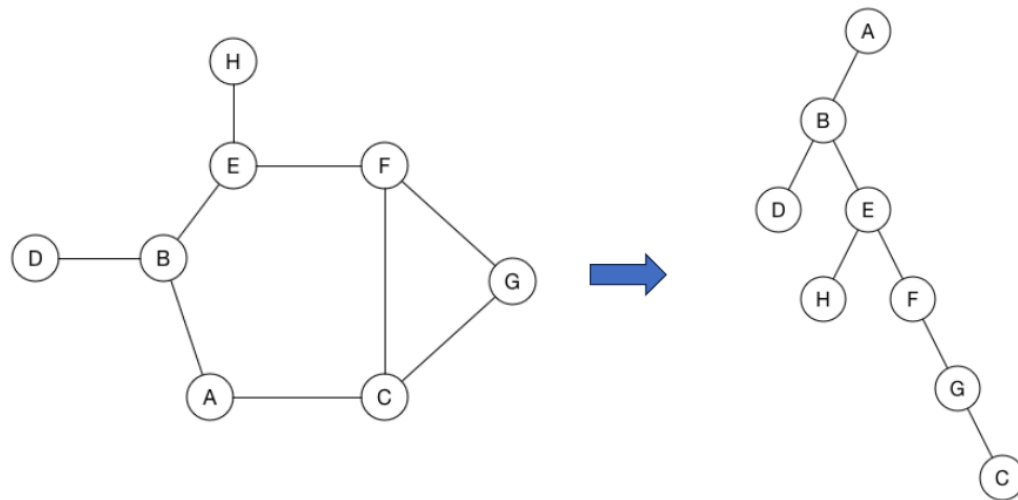
Algoritma BFS memiliki karakteristik:

- Completeness \rightarrow menjamin menemukan solusi jika solusi ada, selama branching factor terbatas.
- Optimality \rightarrow menjamin langkah optimal dalam pembentukan solusi (Tidak berlaku untuk kasus seperti Little Alchemy, karena BFS mencari resep dengan kedalaman terpendek, tetapi bukan berarti paling efisien (bisa membutuhkan lebih banyak kombinasi karena struktur resep dangkal tapi banyak penggabungan))
- Kompleksitas Waktu $\rightarrow O(b^d)$ dengan b sebagai branching factor dan d kedalaman solusi
- Kompleksitas Ruang $\rightarrow O(b^d)$, dengan setiap simpul pada depth sama disimpan pada memory

BFS memiliki kelebihan mencari solusi paling dangkal (untuk kasus tertentu), sehingga cocok untuk mencari solusi yang pendek dan memiliki solusi banyak, tetapi kurang ideal untuk penjelajahan besar karena konsumsi memori yang tinggi.

C. Algoritma Depth First Search

Algoritma Depth First Search (DFS) adalah algoritma penjelajahan graf yang melakukan penjelajahan tanpa informasi seperti BFS. Perbedaan pada algoritma DFS berada pada cara pencarian, dimana DFS sama-sama memulai dari simpul awal, tetapi menjelajahi simpul sedalam mungkin pada salah satu cabang, sebelum melakukan backtrack dan mencari lagi pada cabang lainnya.



Urutan simpul-simpul yang dikunjungi secara DFS dari A \rightarrow A, B, D, E, H, F, G, C

Algoritma DFS memiliki karakteristik:

- Completeness \rightarrow menjamin menemukan solusi jika solusi ada, selama ada penanganan redundant paths dan repeated states
- Optimality \rightarrow tidak selalu menemukan path paling optimal
- Kompleksitas waktu $\rightarrow O(b^m)$, dengan m kedalaman maksimum
- Kompleksitas ruang $\rightarrow O(bm)$, lebih baik dari BFS

DFS memiliki kelebihan kompleksitas ruang yang lebih kecil dibandingkan BFS, tetapi lebih terbatas dalam kecepatan dan tidak selalu dapat menemukan path yang paling optimal.

D. Penjelasan Aplikasi Web

Aplikasi web yang kami rancang dapat digunakan untuk menyelesaikan masalah pencarian resep pembentukan suatu elemen dalam game Little Alchemy secara otomatis. Aplikasi akan memiliki 3 bagian utama yaitu:

1. Scraper

Digunakan untuk melakukan scraping data dari website fandom Little Alchemy 2 ([https://little-alchemy.fandom.com/wiki/Elements_\(Little_Alchemy_2\)](https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2))), dengan menggunakan bahasa GO. Hasil scraping disimpan dalam bentuk JSON, kemudian diolah kembali oleh fungsi flatten untuk mempermudah pencarian menggunakan BFS/DFS.

2. Backend

Digunakan untuk melakukan pencarian satu atau banyak resep dari elemen tujuan. Program akan menggunakan multithreading saat banyak resep dicari. Pengguna juga dapat memilih antara pencarian menggunakan BFS maupun DFS untuk mencari resep. Output akan berupa rangkaian resep yang diperlukan, disertai waktu dan jumlah simpul yang dikunjungi oleh program. Bagian ini juga menggunakan bahasa GO.

3. Frontend

Interface utama program dengan user. User akan dapat melakukan pencarian resep elemen melalui website, sehingga pengguna dapat menemukan resep dengan mudah dan membandingkan hasil traversal antara algoritma BFS dan DFS. Interface juga menyediakan visualisasi hasil resep yang lebih jelas.

BAB III

ANALISIS PEMECAHAN MASALAH

A. Langkah-langkah Pemecahan Masalah

Gambaran langkah-langkah pemecahan masalah yang diambil secara umum:

1. Melakukan scraping dari website fandom Little Alchemy 2, dengan menggunakan program scraper.go, yang menggunakan bantuan modul dari colly dan goquery untuk melakukan web scraping.
2. Hasil scraping lalu diubah formatnya menjadi bentuk Element, Ingredient1, Ingredient2, dan Tier untuk setiap resep agar mempermudah logika traversal dari data menggunakan flatten.go. Filtering data juga dilakukan pada tahap ini, dimana elemen seperti Time dan hasil yang berkaitan dengan Time dihapus dari database. Tier ingredient yang tidak memenuhi juga sudah difilter dalam tahap ini.
3. Program lalu akan memuat data hasil flattening, dengan membentuk sebuah struct FlatRecipe.
4. Program kemudian menerima input pengguna yang berisi target elemen, metode pencarian (BFS/DFS), jumlah pencarian (single/multiple (multithreading)), dan jumlah maksimal resep bila melakukan multiple search.
5. Program lalu melakukan proses traversal BFS/DFS sesuai dengan fungsinya bfsSingleTree / buildSingleTreeDFS untuk single, dan buildNRecipesDFS / multiBFS untuk multiple. bfsSingleTree menggunakan elemen dasar sebagai titik awal pencarian, sedangkan multiBFS melakukan top down dari target elemen, dengan membagi pencarian ingredient awal ke thread masing-masing untuk multithreading. Kedua DFS melakukan top down search, dengan single DFS terbatas hanya mencari 1 resep saja.
6. Hasil pencarian lalu ditampilkan dalam website frontend, dengan jumlah node yang dikunjungi dan jalur pembuatan elemen.

B. Pemetaan Masalah ke Elemen-elemen Algoritma

Masalah	Algoritma
Elemen Dasar dan Turunan	Simpul-simpul dalam graf
Resep pembentuk elemen	Sisi dari graf
Validasi tier	Dilakukan pada proses filtering

Pencarian resep	BFS/DFS sesuai dengan input pengguna
-----------------	--------------------------------------

C. Fitur Fungsional dan Arsitektur Aplikasi Web

Fitur Fungsional :

- Pencarian jalur dengan BFS/DFS
- Metode Single/Multiple Recipe
- Perhitungan waktu pencarian dan simpul yang dikunjungi
- Validasi resep pembentuk berasal dari tier lebih tinggi
- Visualisasi graf pembentuk resep

Arsitektur Aplikasi Web :

- Frontend : Menggunakan [Next.js](#), React dan Tailwind CSS untuk styling, framer-motion untuk animation
- Backend : Menggunakan Go API

D. Contoh ilustrasi kasus.

Mencari resep Brick

Metode : multiBFS

Program akan mencari resep yang membuat Brick, dimana akan ditemukan Ingredient Mud dan Fire. Program lalu akan melanjutkan pencarian pada resep Mud, karena belum elemen dasar, sedangkan fire sudah selesai karena elemen dasar. Kedua ingredient di append pada tree, kemudian saat pengecekan selanjutnya ditemukan resep mud yaitu water dan earth. Kedua komponen ini juga sudah dasar sehingga dilakukan append terakhir dan proses selesai. Tree BFS lengkap dari Brick pun selesai dibuat.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).

Struktur data:

Struktur data	Kegunaan
<pre>type Recipe struct { Product string `json:"product"` Ingredients [][]string `json:"ingredients"` }</pre>	Menyimpan resep dari hasil scraping awal
<pre>type Block struct { Title string `json:"title"` Recipes []Recipe `json:"recipes"` }</pre>	Menyimpan blok resep per tier sesuai dengan title dari hasil scraping awal.
<pre>type FlatRecipe struct { Element string `json:"Element"` Ingredient1 string `json:"Ingredient1"` Ingredient2 string `json:"Ingredient2"` Tier int `json:"Tier"` }</pre>	Menyimpan resep yang sudah diproses lebih jauh untuk memudahkan parsing
<pre>type RecipeNode struct { Element string `json:"element"` Ingredients []*RecipeNode `json:"ingredients,omitempty"` }</pre>	Struktur data tree untuk menyimpan pohon hasil BFS dan DFS

}	
<pre> type SearchRequest struct { Target string `json:"target"` Method string `json:"method"` Mode string `json:"mode"` Limit int `json:"limit"` } </pre>	Menyimpan input dari user untuk keperluan frontend
<pre> type SearchResponse struct { Trees []*RecipeNode `json:"trees"` TimeMs float64 `json:"timeMs"` NodesVisited int `json:"nodesVisited"` } </pre>	Menyimpan hasil pencarian fungsi untuk diberikan kepada frontend
<pre> var baseElements = map[string]bool{ "air": true, "earth": true, "fire": true, "water": true, } </pre>	Menyimpan elemen dasar
<pre> var recipesIndex map[string][][2]string </pre>	Menyimpan resep dari setiap elemen dalam bentuk map untuk melakukan lookup yang lebih cepat dan mudah.

Fungsi / Prosedur:

Fungsi/ Prosedur	Kegunaan
<pre>func Scrape()</pre>	Melakukan proses scraping awal dari

	website wiki littlealchemy2
<pre>func splitIngredients(text string) []string</pre>	Fungsi bantuan untuk membagi-bagi ingredients dari website littlealchemy2 pada fungsi Scrape()
<pre>func newRecipe(element, ing1, ing2 string, tier int) FlatRecipe</pre>	Fungsi pembantu untuk FlattenRecipesFromFile(), mereturn bentuk FlatRecipe yang diinginkan.
<pre>func FlattenRecipesFromFile(inputPath string) ([]FlatRecipe, error)</pre>	Fungsi utama untuk melakukan proses FlattenRecipe dari JSON awal hasil scraping. Filtering dilakukan untuk menghilangkan elemen Time dan yang membutuhkan Time, juga menghilangkan resep dengan ingredient yang memiliki tier sama atau lebih tinggi.
<pre>func main()</pre>	! Main pada folder scraping Alur untuk melakukan scraping dan flattening jika diinginkan.
<pre>func main()</pre>	! Main pada folder Backend Alur fungsi untuk melakukan proses loading recipe, melakukan indexing dari recipe, dan menyiapkan HTTP untuk Frontend.
<pre>func loadRecipes(path string) ([]FlatRecipe, error)</pre>	Fungsi untuk melakukan proses loading recipes untuk keperluan BFS dan BFS dari hasil flattened.json
<pre>func indexRecipes(recipes []FlatRecipe) map[string][][2]string</pre>	Fungsi untuk memproses resep yang sudah di-load pada flatRecipes menjadi sebuah index map untuk memudahkan pengaksesan resep.
<pre>func cloneVisited(original map[string]bool) map[string]bool</pre>	Fungsi untuk melakukan copy map, digunakan untuk meng-clone map Visited yang diperlukan oleh DFS untuk mencegah pengunjungan resep yang sama dengan memastikan hasil copy map lengkap.

<pre>func buildSingleTreeDFS (element string, index map[string][][2]string, visited map[string]bool, counter *int) *RecipeNode</pre>	<p>Alur utama program DFS untuk single recipe. Melakukan pencarian resep secara top down.</p>
<pre>func buildNRecipesDFS (element string, index map[string][][2]string, visited map[string]bool, counter *int, maxRecipes int)</pre>	<p>Alur program sama seperti buildSingleTreeDFS, dengan tambahan logika untuk berhenti setelah N recipes tercapai, dan tambahan fungsi multithreading dimana pemanggilan DFS secara rekursif akan memanggil goroutine baru</p>
<pre>func bfsSingleTree (element string, index map[string][][2]string, counter *int) *RecipeNode</pre>	<p>Fungsi yang melakukan pemrosesan BFS secara bottom-up dari base element hingga ditemukan resep BFS yang memenuhi.</p>
<pre>func multiBFS (element string, index map[string][][2]string, counter *int) *RecipeNode</pre>	<p>Fungsi BFS yang handle proses pembuatan secara multithreading. Proses multithreading dilakukan hanya pada ingredient awal dari elemen tujuan, dimana masing-masing pasangan ingredient diberikan pada thread yang berbeda. Hasil ini kemudian akan digabungkan kembali dengan node awal pada tree sesuai dengan urutan pemanggilan.</p>
<pre>func buildRecipeTreeBFS (element string, index map[string][][2]string, counter *int) *RecipeNode</pre>	<p>Fungsi BFS utama, yang melakukan pencarian secara BFS hingga seluruh variasi resep ditemukan.</p>
<pre>func searchHandler (w http.ResponseWriter, r *http.Request)</pre>	<p>Fungsi yang menangani HTTP request dari FrontEnd, menjalankan alur pencarian resep.</p>

B. Penjelasan tata cara penggunaan program

Program menyediakan 4 fitur pencarian :

1. **Breadth First Search (BFS) Single Recipe**

BFS Single Recipe mencari satu jalur pembuatan elemen target dengan strategi melebar (breadth-first).

2. **Breadth First Search (BFS) Multiple Recipes**

BFS Multiple Recipe memperluas algoritma BFS untuk menemukan beberapa jalur pembuatan alternatif. Algoritma ini tidak berhenti setelah menemukan jalur pertama, melainkan terus mencari hingga semua kemungkinan jalur telah dieksplorasi.

3. **Depth First Search (DFS) Single Recipe**

DFS Single Recipe mencari satu jalur pembuatan dengan strategi mendalam (depth-first). Algoritma ini menelusuri satu cabang sejauh mungkin sebelum melakukan backtracking.

4. **Depth First Search (DFS) Multiple Recipes**

DFS Multiple Recipe memperluas algoritma DFS untuk menemukan beberapa jalur pembuatan alternatif. Algoritma ini menggunakan backtracking untuk mengeksplorasi berbagai kombinasi resep yang mungkin.

Setiap fitur pencarian menampilkan jumlah nodes yang dikunjungi dan waktu eksekusi untuk mencari elemen yang diinginkan. Pada mode Multiple Recipe, pengguna dapat memilih jumlah resep alternatif yang ingin ditampilkan. Selain itu, pencarian multiple recipe melibatkan multithreading.

Interface program terdiri dari 3 kolom yang dapat diisi. Kolom pertama untuk memasukkan keyword atau recipe yang ingin dicari, lalu untuk kolom kedua terdapat pilihan untuk memilih BFS atau DFS, tergantung mana jenis pencarian yang ingin digunakan. Kolom ketiga digunakan untuk memilih apakah pengguna ingin mencari satu recipe saja atau multiple recipe. Bila memilih multiple recipe, untuk DFS akan dapat memilih berapa banyak recipe yang diinginkan, dan untuk BFS akan dimunculkan semua.

Cara penggunaan program sangat simpel. Masukkan keyword atau recipe yang diinginkan, lalu pilih metode pencarian dan jumlah pencariannya lalu tekan search. Hasil pencarian akan muncul dalam bentuk tree dan akan muncul juga jumlah node yang ditelusuri serta lama waktu pencarian.

C. Hasil pengujian

Test Case 1: Pencarian Elemen Brick (1 solusi)

BFS 1

Brick

BFS

Multiple Recipes

Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 2

Visual Recipe Tree

brick

mud

fire

water

earth

BFS Multiple

Brick

BFS

Single Recipe

Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 162

Visual Recipe Tree

brick


mud

fire

water

earth

DFS 1

 **Little Alchemy 2**
Element Recipe Search

Brick

DFS

Single Recipe

Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 5

Visual Recipe Tree

brick


mud

fire

water

earth

DFS Multiple (hanya return resep mungkin)

 **Little Alchemy 2**
Element Recipe Search

Brick

DFS

Multiple Recipes

3

Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 5

Visual Recipe Tree

brick

mud

fire

water

earth

Test Case 2: Pencarian Elemen Blade

BFS 1

BFS Multiple

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 219

Visual Recipe Tree

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 13

Visual Recipe Tree

DFS 1


Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 13

Visual Recipe Tree

DFS Multiple


Little Alchemy 2
 Element Recipe Search

DFS
▼

Multiple Recipes
▼

3
Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 35

Recipe #1

Visual Recipe Tree

```

graph TD
    blade[blade] --- stone1[stone]
    blade --- metal1[metal]
    stone1 --- earth1[earth]
    stone1 --- pressure1[pressure]
    pressure1 --- air1[air]
    pressure1 --- air2[air]
    metal1 --- stone2[stone]
    metal1 --- fire1[fire]
    stone2 --- earth2[earth]
    stone2 --- pressure2[pressure]
    pressure2 --- air3[air]
    pressure2 --- air4[air]
                    
```

Recipe #2

Visual Recipe Tree

```

graph TD
    blade[blade] --- stone1[stone]
    blade --- metal1[metal]
    stone1 --- earth1[earth]
    stone1 --- pressure1[pressure]
    pressure1 --- air1[air]
    pressure1 --- air2[air]
    metal1 --- stone2[stone]
    metal1 --- fire1[fire]
    stone2 --- air3[air]
    stone2 --- lava[lava]
    lava --- earth2[earth]
    lava --- fire2[fire]
                    
```

Recipe #3

Visual Recipe Tree

```

graph TD
    blade[blade] --- stone1[stone]
    blade --- metal1[metal]
    stone1 --- earth1[earth]
    stone1 --- pressure1[pressure]
    pressure1 --- air1[air]
    pressure1 --- air2[air]
    metal1 --- stone2[stone]
    metal1 --- heat[heat]
    stone2 --- earth2[earth]
    stone2 --- pressure2[pressure]
    pressure2 --- air3[air]
    pressure2 --- air4[air]
    heat --- air5[air]
    heat --- energy[energy]
    energy --- fire1[fire]
    energy --- fire2[fire]
                    
```

Test Case 3: Pencarian Elemen Cheese

BFS 1

Search Statistics

Execution Time: 0.525 ms

Nodes Visited: 304

cheese

BFS

Single Recipe

Search

```

graph TD
    cheese[cheese] --> barn[barn]
    cheese --> field[field]
    barn --> house[house]
    barn --> field2[field]
    house --> wall[wall]
    wall --> brick1[brick]
    wall --> brick2[brick]
    brick1 --> mud1[mud]
    brick1 --> fire1[fire]
    brick2 --> mud2[mud]
    brick2 --> fire2[fire]
    mud1 --> water1[water]
    mud1 --> earth1[earth]
    mud2 --> water2[water]
    mud2 --> earth2[earth]
    field2 --> plow[plow]
    field2 --> earth3[earth]
    plow --> earth4[earth]
    plow --> metal[metal]
    metal --> stone[stone]
    metal --> fire3[fire]
    stone --> earth5[earth]
    stone --> pressure[pressure]
    pressure --> air1[air]
    pressure --> air2[air]
  
```

BFS Multiple (berhasil, load fe terlalu berat)

BFS

Multiple Recipes

Search

Search Statistics

Execution Time: 0.000 ms

Nodes Visited: 259

Visual Recipe Tree

DFS 1

DFS

Single Recipe

Search

Search Statistics
Execution Time: 0.000 ms
Nodes Visited: 101

DFS Multiple

DFS

Multiple Recipes

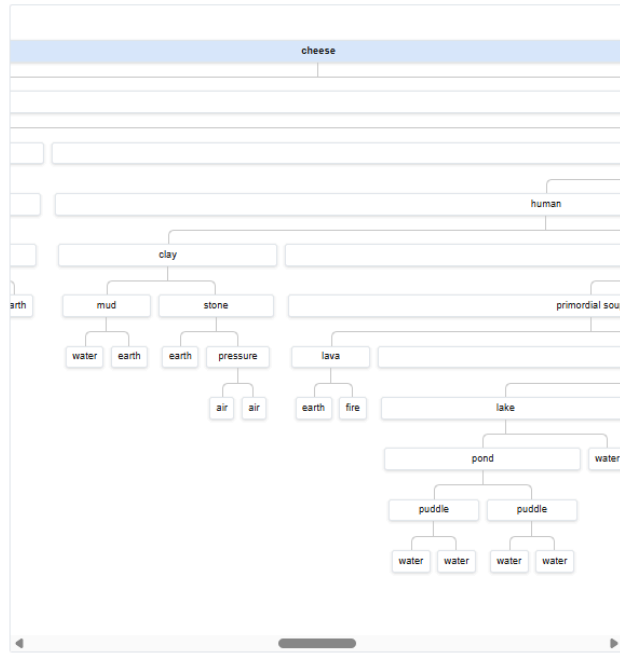
100

Search

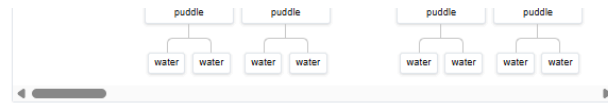
Search Statistics
Execution Time: 5.600 ms
Nodes Visited: 1349

Recipe #1

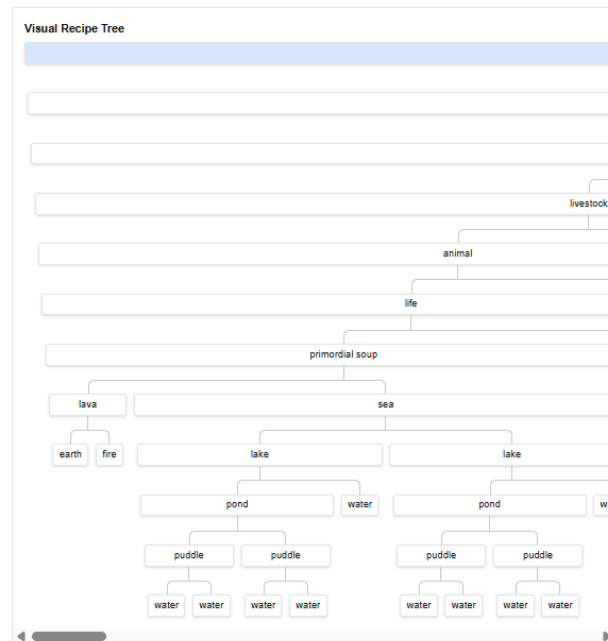
Recipe #53



Recipe #54



Recipe #100



D. Analisis hasil pengujian.

Algoritma berhasil mencari resep baik untuk Brick, Blade, maupun Cheese. Algoritma yang mendapatkan hasil paling lama adalah algoritma BFS Single, karena struktur algoritma yang mencari dengan cara bottom-up. Ini berarti lebih banyak node yang harus dilacak dan dicari untuk mencapai target elemen.

BFS Multiple menjadi algoritma yang paling cepat untuk melakukan pemetaan, karena approach top down dan hanya mengembalikan 1 tree yang tergabung. DFS Single juga algoritma yang sangat cepat karena approachnya yang sama-sama top down. DFS Multiple lebih lambat dari BFS Multiple, karena perbedaan dalam bentuk output dimana DFS melakukan output dengan batasan N resep, dan output resep dibagi menjadi tree masing-masing. Ini berarti aplikasi DFS butuh waktu yang lebih lama untuk pemrosesan.

Untuk resep yang sangat besar, seperti pada pencarian Cheese dengan BFS, algoritma dapat mencari resep dan mengembalikan tree. Masalah muncul saat mencoba melakukan display pada front end, karena perlu komputasi yang berat untuk menunjukkan tree yang sangat besar. Bottleneck ini muncul dari UI, bukan dari Backend maupun Frontend. Aplikasi pembatasan resep dapat membantu untuk mengurangi masalah ini, seperti dengan pencarian DFS Multiple dengan N recipe 100.

Dari pengujian, program pencarian resep Little Alchemy 2 sudah berhasil, dan output dari berbagai elemen yang dicari sudah sesuai dengan resep aslinya dan bisa diaplikasikan oleh pengguna pada permainan aslinya.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Implementasi algoritma pencarian pada game elemen berhasil dilakukan dengan empat metode berbeda: BFS Single Recipe, BFS Multiple Recipe, DFS Single Recipe, dan DFS Multiple Recipe. Setiap algoritma memiliki karakteristik dan keunggulan tersendiri dalam menemukan jalur pembuatan elemen. Untuk mode Multiple Recipe, program berhasil memberikan fleksibilitas kepada pengguna untuk menentukan jumlah resep alternatif yang diinginkan. Pengukuran performa berupa jumlah node yang dikunjungi dan waktu eksekusi memberikan insight berharga tentang efisiensi masing-masing algoritma pada dataset yang digunakan.

B. Saran

Berdasarkan pengalaman dalam mengembangkan program pencarian resep elemen dalam tugas ini, berikut merupakan beberapa saran untuk pengembangan lebih lanjut:

- 1) Implementasi Algoritma Hybrid: Mengkombinasikan kelebihan BFS dan DFS untuk menciptakan algoritma yang lebih efisien, misalnya dengan Iterative Deepening DFS atau Bidirectional Search.
- 2) Optimasi Memori: Menerapkan teknik kompresi data atau struktur data yang lebih efisien untuk menangani dataset yang sangat besar.
- 3) Algoritma BFS yang lebih fleksibel: Membatasi N recipe agar program dapat lebih fleksibel

C. Refleksi

Pengembangan program pencarian resep elemen ini memberikan pembelajaran mendalam tentang algoritma graf dan strategi pencarian. Kami menyadari bahwa tidak ada algoritma terbaik secara universal dan setiap algoritma memiliki tradeoff antara kecepatan, penggunaan memori, dan kelengkapan hasil.

BFS dan DFS mempunyai cara kerja algoritma yang berbeda, dan keduanya memiliki kelebihan dan kekurangannya masing-masing sesuai dengan situasi. Implementasi Multiple Recipe mengajari kami kompleksitas dalam menyajikan beberapa solusi alternatif dan tantangan dalam mengimplementasikan sistem multithreading. Pengukuran performa membantu kami memahami dampak struktur data dan algoritma terhadap efisiensi sistem secara keseluruhan.

Tugas ini memperkuat pemahaman kami bahwa desain algoritma yang baik harus mempertimbangkan karakteristik data, kebutuhan pengguna, dan batasan sistem. Keseimbangan antara kompleksitas waktu, penggunaan memori, dan kualitas hasil menjadi kunci dalam mengembangkan sistem pencarian yang efektif dan efisien.

LAMPIRAN

A. Checklist Pengerjaan

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-containerize dengan Docker.	✓	
11	Aplikasi di-deploy dan dapat diakses melalui internet.	✓	

B. Tautan Repository Github

https://github.com/doober22/Tubes2_H-1-Kelar

C. Tautan Video

<https://youtu.be/FGG4Mu5rtAE>

D. Tautan Deployment

<https://littlealchemy2-v4g4.vercel.app/>

DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm>