LAPORAN TUGAS BESAR IF2211
STRATEGI ALGORITMA

Tugas Besar 2 Strategi Algoritma

Kelompok : Help Stima

Brian Albar Hadian	13523048
M. Izzat Jundy	13523092
Andrew Isra Putra DB	13523110

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

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:

1. Elemen dasar

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

Gambar 2. Elemen dasar pada Little Alchemy 2

2. Elemen turunan

   Terdapat 720 elemen turunan yang dibagi menjadi beberapa tier tergantung tingkat kesulitan dan banyak langkah yang harus dilakukan. Setiap elemen turunan memiliki recipe yang terdiri atas elemen lainnya atau elemen itu sendiri.

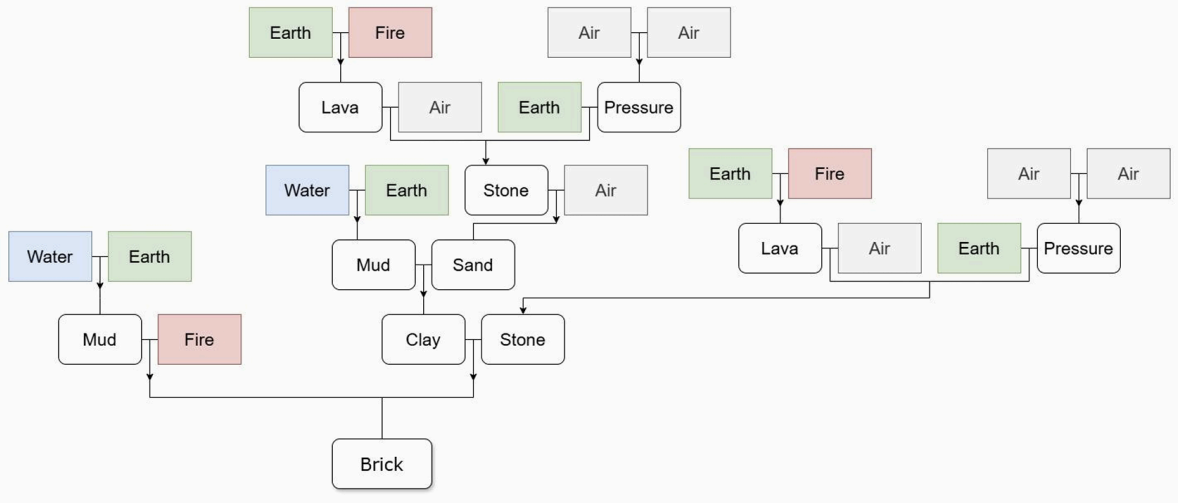
3. Combine Mechanism

   Untuk mendapatkan elemen turunan pemain dapat melakukan combine antara 2 elemen untuk menghasilkan elemen baru. Elemen turunan yang telah didapatkan dapat digunakan kembali oleh pemain untuk membentuk elemen lainnya.

Spesifikasi Wajib

- Buatlah aplikasi pencarian recipe elemen dalam permainan Little Alchemy 2 dengan menggunakan strategi BFS dan DFS.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
- Aplikasi berbasis web, untuk frontend dibangun menggunakan bahasa Javascript dengan framework Next.js atau React.js, dan untuk backend menggunakan bahasa Golang.
- Untuk repository frontend dan backend diperbolehkan digabung maupun dipisah.
- Untuk data elemen beserta resep dapat diperoleh dari scraping website Fandom Little Alchemy 2.

- Terdapat opsi pada aplikasi untuk memilih algoritma BFS atau DFS (juga bidirectional jika membuat bonus)
- Terdapat toggle button untuk memilih untuk menemukan sebuah recipe terpendek (output dengan rute terpendek) atau mencari banyak recipe (multiple recipe) menuju suatu elemen tertentu. Apabila pengguna ingin mencari banyak recipe maka terdapat cara bagi pengguna untuk memasukkan parameter banyak recipe maksimal yang ingin dicari. Aplikasi boleh mengeluarkan recipe apapun asalkan berbeda dan memenuhi banyak yang diinginkan pengguna (apabila mungkin).
- Mode pencarian multiple recipe wajib dioptimasi menggunakan multithreading.
- Aplikasi akan memvisualisasikan recipe yang ditemukan sebagai sebuah tree yang menunjukkan kombinasi elemen yang diperlukan dari elemen dasar. Agar lebih jelas perhatikan contoh berikut



Gambar 3. Contoh visualisasi *recipe* elemen

Gambar diatas menunjukkan contoh visualisasi recipe dari elemen Brick. Setiap elemen bersebelahan menunjukkan elemen yang perlu dikombinasikan. Amati bahwa leaf dari tree selalu berupa elemen dasar. Apabila dihitung, gambar diatas menunjukkan 5 buah recipe untuk Brick (karena Brick dapat dibentuk dengan kombinasi Mud+Fire atau Clay+Stone, begitu pula Stone yang dapat dibentuk oleh kombinasi Lava+Air atau Earth+Pressure). Visualisasi pada aplikasi tidak perlu persis seperti contoh diatas, tetapi pastikan bahwa recipe ditampilkan dengan jelas.

- Aplikasi juga menampilkan waktu pencarian serta banyak node yang dikunjungi.

BAB II

LANDASAN TEORI

2.1. Dasar Teori Algoritma BFS dan DFS

Graf adalah struktur data nonlinier yang digunakan untuk merepresentasikan hubungan antar-objek. Graf terdiri dari simpul dan sisi. Simpul adalah titik yang merepresentasikan objek. Sisi adalah garis yang menghubungkan antar dua simpul. Graf memiliki beberapa jenis dengan dua faktor pembeda, yaitu graf berarah atau tak berarah, graf berbobot atau tidak berbobot. Jadi, terdapat empat jenis graf. Dalam tugas besar ini, graf yang digunakan adalah graf berarah dan tak berbobot.

Breadth First Search (BFS) dan *Depth First Search* (DFS) adalah algoritma pencarian graf yang menjelajahi simpul-simpul dalam graf dengan urutan tertentu untuk menemukan sebuah simpul yang dicari. BFS menjelajahi simpul pada level yang sama terlebih dahulu, sementara DFS menjelajahi graf hingga simpul terdalam sebelum mundur dan menjelajahi jalur lain. BFS diimplementasikan dengan menggunakan sistem *queue*, yaitu *First In First Out* (FIFO). DFS diimplementasikan dengan menggunakan sistem *stack*, yaitu *Last In First Out* (LIFO). BFS, jika diterapkan pada graf tidak berbobot, dapat menjamin bahwa jalur dari penemuan pertama simpul merupakan jalur terpendek. Sementara DFS tidak selalu menemukan jalur terpendek.

Elemen-elemen pada algoritma BFS dan DFS:

1. Queue pada BFS, Stack pada DFS
2. Visited Set/Array, untuk menandai simpul-simpul yang sudah dikunjungi agar tidak terjadi infinite loop
3. Graf, untuk merepresentasikan hubungan antarsimpul yang dijelajahi
4. Prosedur
    a. BFS
        i. Memasukkan simpul ke queue
        ii. Selagi queue tidak kosong
            1. Mengambil simpul terdepan dari queue

2. Memeriksa kesesuaian simpul dengan yang dicari. Jika sesuai, berhenti
3. Menandai simpul tersebut sebagai telah dikunjungi
4. Memasukkan semua tetangga yang belum dikunjungi ke queue

b. DFS
   i. Memasukkan simpul ke stack
   ii. Selagi stack tidak kosong
      1. Mengambil simpul teratas dari stack
      2. Memeriksa kesesuaian simpul dengan ayng dicari. Jika sesuai, berhenti
      3. Menandai simpul tersebut sebagai telah dikunjungi
      4. Memasukkan semua tetangga yang belum dikunjungi ke stack

2.2. Penjelasan Web yang Dibangun

Website ini dirancang dengan menggunakan menggunakan *framework* NextJs dengan bahasa pemrograman yang digunakan adalah Javascript. Alasan menggunakan *framework* ini yaitu kemudahan dalam menampilkan desain dengan penyatuan antara script dan style. Adapun alasan lain karena pada React/Next Js ini sudah tersedia fitur untuk menampilkan tree sehingga memudahkan *developer* untuk memvisualisasikan hasilnya. Adapun untuk memperindah tampilan website, digunakan kakas tailwind. Terakhir, Axios digunnakan untuk melakukan fetch API ke backend.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Penyelesaian Masalah

Langkah penyelesaian masalah dimulai dengan identifikasi masalah, pemilihan pendekatan algoritma penyelesaian, dan kemudian implementasi penerapan algoritma penyelesaian diterapkan dengan memanfaatkan pendekatan dan struktur data yang telah diperoleh.

Langkah identifikasi masalah dilakukan dengan meninjau tujuan akhir (output) dan melakukan riset terlebih dahulu mengenai permainan *Little Alchemy 2*. Riset tersebut mencakup riset untuk mencari tahu hubungan antar objek yang akan dibuat dan melakukan desain awal terhadap struktur data yang hendak dibuat. Hubungan antar objek kemudian dilanjutkan dengan pemetaan dengan objek lainnya sedemikian sehingga mendapatkan garis besar dari hubungan antar objek tersebut. Didasarkan dari garis besar tersebut, desain struktur data awal dapat dilakukan dan pendefinisian fungsi-fungsi yang mengolah data tersebut dapat dilakukan.

Tahap selanjutnya merupakan pemilihan pendekatan algoritma penyelesaian. Pemilihan ini dilakukan dengan menggunakan hubungan antar objek dan struktur data yang telah didefinisikan sebelumnya. Pertimbangan yang digunakan adalah atribut-atribut yang dimiliki oleh struktur data tersebut dan hubungan antar objek sedemikian sehingga desain sistem dirancang untuk mempermudah pengolahan hubungan antar objek tersebut.

Langkah terakhir dilakukan dengan implementasi algoritma dan struktur data yang telah dipilih. Pada tahap ini, dilakukan dekomposisi komponen fungsional dan perancangan struktur folder yang sesuai dengan *best practice* yang ada.

Berdasarkan identifikasi masalah yang dilakukan, diperoleh bahwa hubungan antar objek merupakan suatu graf berarah yang setiap simpulnya memiliki jumlah panah masuk sejumlah kelipatan dari dua. Hal ini didasarkan dari perilaku objek pada Little Alchemy 2 yang membutuhkan dua objek lainnya untuk menghasilkan objek tertentu atau bahkan objek itu sendiri. Dengan suatu simpul memiliki panah keluar dan panah masuk, maka didefinisikan

struktur data yang dinamakan AlchemyTree yang memiliki atribut Name bertipe *string*, Parent bertipe *array* berisi *tuple* yang terdiri dari penunjuk memori ke suatu pasangan struktur data *AlchemyTree* lainnya, Companion bertipe *array* yang berisi penunjuk memori ke struktur data *AlchemyTree* lainnya, serta Children bertipe *array* yang berisi penunjuk memori ke struktur data *AlchemyTree* lainnya.

Selanjutnya, pendekatan yang dipilih untuk menyelesaikan persoalan, yakni mencari suatu resep dari suatu objek hingga ke objek dasar, adalah algoritma BFS dan DFS. Hal ini dipengaruhi dari keluaran yang diharapkan berupa resep dari proses pencarian yang telah diperoleh. Selain itu, pemilihan algoritma tersebut juga dipengaruhi oleh struktur data yang digunakan, yakni struktur data graf. Proses pencarian resep kemudian dilakukan dengan menggunakan algoritma BFS dan DFS untuk mencari dari suatu simpul *target* ke simpul lainnya, yaitu simpul *parent* yang membuat simpul *target*. Dengan memanfaatkan algoritma BFS, proses pencarian resep akan dilakukan dengan menelusuri terlebih dahulu semua simpul *parent* yang langsung terhubung dengan simpul *target* yang sedang ditelusuri. Hal ini kemudian dilanjutkan dengan pencarian simpul *target* melalui simpul *parent* yang telah diperoleh apabila belum ditemukan simpul *target* tersebut atau simpul *target* merupakan suatu simpul yang merepresentasikan elemen dasar.

Tahap terakhir dilakukan dengan membagi keperluan fungsional program secara modular, pembagian kerja sesuai dengan beban masing-masing, serta perancangan struktur *folder* dirancang dengan struktur dasar sebagai berikut :

1. *src*

Struktur *folder* ini berfungsi untuk menyimpan seluruh bagian dari program yang utamanya terdiri atas *frontend,* yang berfungsi untuk menampilkan hasil dalam format laman situs, dan *backend*, yang bertanggung jawab untuk melakukan kalkulasi hasil

2. *doc*
3.

3.2. Proses Pemetaan Masalah ke BFS dan DFS

<pembagian kerja sesuai elemen masing-masing, cara kerja BFS secara dasar, pendekatan dan motivasi yang dimiliki, serta struktur data apa yang digunakan dan letak peneraan multithreading>

<hasil data yang diperoleh dan struktur data yang didefinisikan sebelumnya + alasannya, proses pemetaan struktur data ke BFS dan DFS, manfaat yang diperoleh>

Persoalan Little Alchemy 2 pada Tugas Besar 2 ini kami uraikan menjadi elemen-elemen algoritma *DFS* dan *BFS* seperti berikut:

Pencarian solusi pembentukan pohon dinamis • Setiap simpul diperiksa apakah solusi (goal) telah dicapai atau tidak. Jika simpul merupakan solusi , pencarian dapat selesai (satu solusi) atau dilanjutkan mencari solusi lain (semua solusi). • Representasi pohon dinamis: • Pohon ruang status (state space tree) • Simpul: problem state (layak membentuk solusi) • Akar: initial state • Daun: solution/goal state • Cabang: operator/langkah dalam persoalan • Ruang status (state space): himpunan semua simpul • Ruang solusi: himpunan status solusi • Solusi: path ke status solusi

1. Struktur Graf
    a. Node : simpul yang melambangkan suatu objek dalam *Little Alchemy 2*
    b. Edge : hubungan antar objek secara terarah
    c. Parent : pasangan objek lainnya yang membentuk objek terkini
2. Karakteristik Graf
    a. Terhubung

    Bentuk graf yang diperoleh adalah suatu graf yang setiap simpul (Node) saling terhubung secara terarah dengan minimal suatu panah masuk yang diperoleh dari suatu pasangan simpul lainnya.

    b. Cyclic

    Bentuk graf yang diperoleh adalah suatu graf yang memiliki simpul yang dapat memiliki arah panah masuk ke dirinya sendiri membentuk suatu hubungan siklik. Panah tersebut bermakna bahwa simpul tersebut berperan sebagai *parent* dari simpul itu sendiri.
3. Penerapan algoritma DFS dan BFS
    a. DFS

Penerapan algoritma DFS menerapkan pola umum sebagai berikut,

1) Kunjungi Node v pada Graf g yang telah terdefinisi
2) Kunjungi Node w yang bertetangga dengan Node v
3) Ulangi prosedur 1) mulai dari Node w
4) Ketika mencapai suatu Node u sedemikian sehingga seluruh simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai Node w yang belum dikunjungi.
5) Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

b. BFS

Penerapan algoritma BFS menerapkan pola umum sebagai berikut,

1) Kunjungi Node v
2) Kunjungi seluruh simpul yang bertetangga dengan Node v terlebih dahulu
3) Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

3.3. Fitur Fungsional dan Arsitektural Web

1. Landing Page

Landing page berisi penjelasan singkat mengenai game Little Alchemy 2. Pada page ini, disediakan tombol yang bisa digunakan untuk menuju game. Jika ingin menggunakan fitur *finder recipe*, diberikan toggle on/off untuk mengaktifkan menu *finder.*

2. Recipe Page

Pada page ini, diberikan fitur utama yang dibuat berupa pencarian recipe dengan berbagai configurasi yaitu, element tujuan, algoritma pencarian, mode pencarian, dan jumlah maksimal recipe. Website akan menampilkan visualisasi berupa representasi resep element yang ingin dicari dengan cara menggabungkan dua node child (element) di atasnya.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

Program ini menggunakan Golang sebagai *backend* dan Next JS berbahasa JavaScript sebagai *frontend*.

4.1.1. Struktur Data

1. AlchemyTree
    a. Name : string
    b. Parent : *array of tuple(AlchemyTree, AlchemyTree)*
    c. Children : *array of AlchemyTree*
    d. Companion : *array of AlchemyTree*
2. Pair
    a. Ingridient1 : AlchemyTree
    b. Ingridient2 : AlchemyTree
3. Tree
    a. Name : string
    b. Children : array of Tree
4. Response
    a. NumOfRecipe : integer
    b. TotalVisitedNode : integer
    c. ExecutionTime : integer
    d. Data : integer

4.1.2. Fungsi

1. DFSAlchemyTree
    a. Fungsionalitas
        i. Melakukan proses pencarian resep dengan menggunakan teknik DFS

b. Input
   i. target : string
   ii. listOfCreatedNodes : array of AlchemyTree
   iii. Mode : short
   iv. askedNumOfRecipes : long
   v. childNode : Tree
   vi. mapOfElementsTier : map of string into int
   vii. currentFoundRecipe : int64
   viii. totalVisitedNode : int64
c. Output : childNode terdefinisi dan tidak kosong
d. Pseudocode

```
procedure DFSAlchemyTree(target : string,
listOfCreatedNodes : array of AlchemyTree, Mode : short,
askedNumOfRecipes : long,  childNode : Tree,
mapOfElementsTier : map of string into int,
currentFoundRecipe : int64, totalVisitedNode : int64)
KAMUS

found : bool

ALGORITMA

if target = "Fire" or target = "Water" or target = "Air" or
target = "Earth" or  target = "Time"

            childNode.Name = target

      found = false

      i traversal listOfCreatedNodes

            if listOfCreatedNodes[i] = nil or
listOfCreatedNodes[i].Name != target

                  { stop }

            j traversal n.Parent

                  if (mapOfElementsTier[p.Ingridient1.Name]
<= mapOfElementsTier[n.Name])  OR
(mapOfElementsTier[p.Ingridient2.Name] <=
mapOfElementsTier[n.Name]) {

                        (totalVisitedNode)++
```

```
                        ing1 = Tree{Name:
p.Ingridient1.Name}

                        ing2 =Tree{Name:
p.Ingridient2.Name}

                        childNode.Name = target

                        childNode.Children =
[]*model.Tree{ing1, ing2}

                        DFSAlchemyTree(ing1.Name,
listOfCreatedNodes, mode, askedNumOfRecipes, ing1,
mapOfElementsTier, currentFoundRecipe, totalVisitedNode)

                        DFSAlchemyTree(ing2.Name,
listOfCreatedNodes, mode, askedNumOfRecipes, ing2,
mapOfElementsTier, currentFoundRecipe, totalVisitedNode)

                        found = true

                        if mode == 1 then

                              ->

      if !found then

            childNode.Name = target
```

2. BFSAlchemyTree (target string, listOfCreatedNodes []*model.AlchemyTree, mode int8, askedNumOfRecipes *int64, response *model.Response, mapOfElementsTier map[string]int, currentFoundRecipe *int64, totalVisitedNode *int64)
    a. Fungsionalitas
        i. Melakukan proses pencarian resep dengan menggunakan teknik DFS
    b. Input
        i. target : string
        ii. listOfCreatedNodes : array of AlchemyTree
        iii. Mode : short
        iv. askedNumOfRecipes : long
        v. response : Response

vi. mapOfElementsTier : map of string into int

vii. currentFoundRecipe : int64

viii. totalVisitedNode : int64

c. Output : childNode terdefinisi dan tidak kosong

d. Pseudocode

```
procedure BFSAlchemyTree(
    target : string,
    listOfCreatedNodes : array of AlchemyTree,
    mode : short,
    askedNumOfRecipes : long,
    response : Response,
    mapOfElementsTier : map of string into int,
    totalVisitedNode : long)


KAMUS
    type QueueItem : (Name : string, Tree : Tree)
    BFSQueue : queue of QueueItem
    nextLevelChan : temporary list of QueueItem
    stopFlag : bool
    item, nextItem : QueueItem
    node : AlchemyTree
    p : Parent
    ing1, ing2 : Tree


ALGORITMA
    BFSQueue ← enqueue (target, &response.Data)
```

```
    while BFSQueue is not empty do

        nextLevelChan ← kosong

        stopFlag ← false

        totalVisitedNode ← totalVisitedNode + 1


        for setiap item dalam BFSQueue lakukan secara
paralel

            if item.Name = "Fire" or "Water" or "Air" or
"Earth" or "Time" then

                continue


            untuk setiap node dalam listOfCreatedNodes do

                if node = nil or node.Name ≠ item.Name
then

                    continue


                untuk setiap p dalam node.Parent do

                    if
mapOfElementsTier[p.Ingridient1.Name] ≤
mapOfElementsTier[item.Name] or


mapOfElementsTier[p.Ingridient2.Name] ≤
mapOfElementsTier[item.Name] then



                        if mode = 1 and stopFlag = true
then

                            return



                        ing1 ← Tree { Name :
p.Ingridient1.Name, Children : [] }

                        ing2 ← Tree { Name :
```

```
p.Ingridient2.Name, Children : [] }



                        tambahkan ing1 dan ing2 ke
item.Tree.Children

                        tambahkan (ing1.Name, &ing1) ke
nextLevelChan

                        tambahkan (ing2.Name, &ing2) ke
nextLevelChan



                        response.NumOfRecipe ←
response.NumOfRecipe + 1



                        if mode = 1 and
response.NumOfRecipe ≥ askedNumOfRecipes then

                            stopFlag ← true

                            return



        BFSQueue ← nextLevelChan



        if mode = 1 and stopFlag = true then

            break
```

4.1.3. Prosedur

1. normalizeTreeData(node)

   Fungsi ini digunakan untuk memformat data tree yang diterima dari server agar dapat digunakan dalam komponen visualisasi pohon.

   Input: Node objek dengan properti Name dan Children.

   Output: Objek dengan properti name dan children, di mana children adalah hasil rekursif dari pemanggilan normalizeTreeData pada anak-anak node (jika ada).

2. handleBack()

   Prosedur ini digunakan untuk mengarahkan pengguna kembali ke halaman sebelumnya menggunakan fungsi window.history.back().

3. handleRefresh()

   Fungsi ini digunakan untuk me-refresh halaman saat ini menggunakan window.location.reload().

4. handleSubmit(e)

   Fungsi ini menangani pengiriman form. Ini mengumpulkan data dari form, mengirimkan request ke backend menggunakan axios, dan memperbarui state result dengan hasil yang diterima.

5. useState()

   Hook React yang digunakan untuk mendeklarasikan state dalam komponen.

6. axios.post()

   Digunakan untuk mengirim data ke server. Pada kode ini, digunakan untuk mengirim payload ke server dengan endpoint http://localhost:8080/api/post-recipe.

7. RecipeTree()

   Komponen ini digunakan untuk merender visualisasi tree berdasarkan data yang diterima dari server.

4.2. Tata Cara Penggunaan Program

Sebelum menjalankan aplikasi, pastikan Anda telah menginstal perangkat lunak berikut:

1. **Go**: Digunakan untuk menjalankan backend.
2. **Node.js**: Digunakan untuk menjalankan frontend.

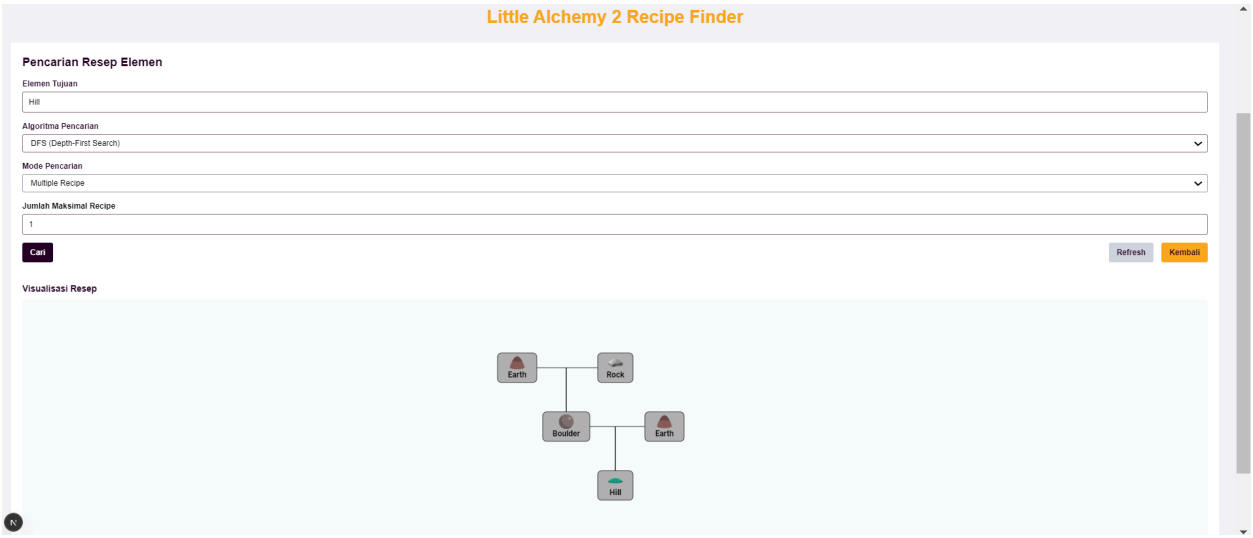
Masuk ke folder src/frontend dan jalankan perintah berikut untuk mengunduh dependencies yang diperlukan oleh frontend: `npm install .` Selanjutnya pindah ke folder src/backend di terminal, kemudian jalankan perintah berikut untuk menjalankan backend: `go run main.go`.

Terakhir masuk ke folder src/frontend di terminal, kemudian jalankan perintah berikut untuk menjalankan frontend: `npm run dev`

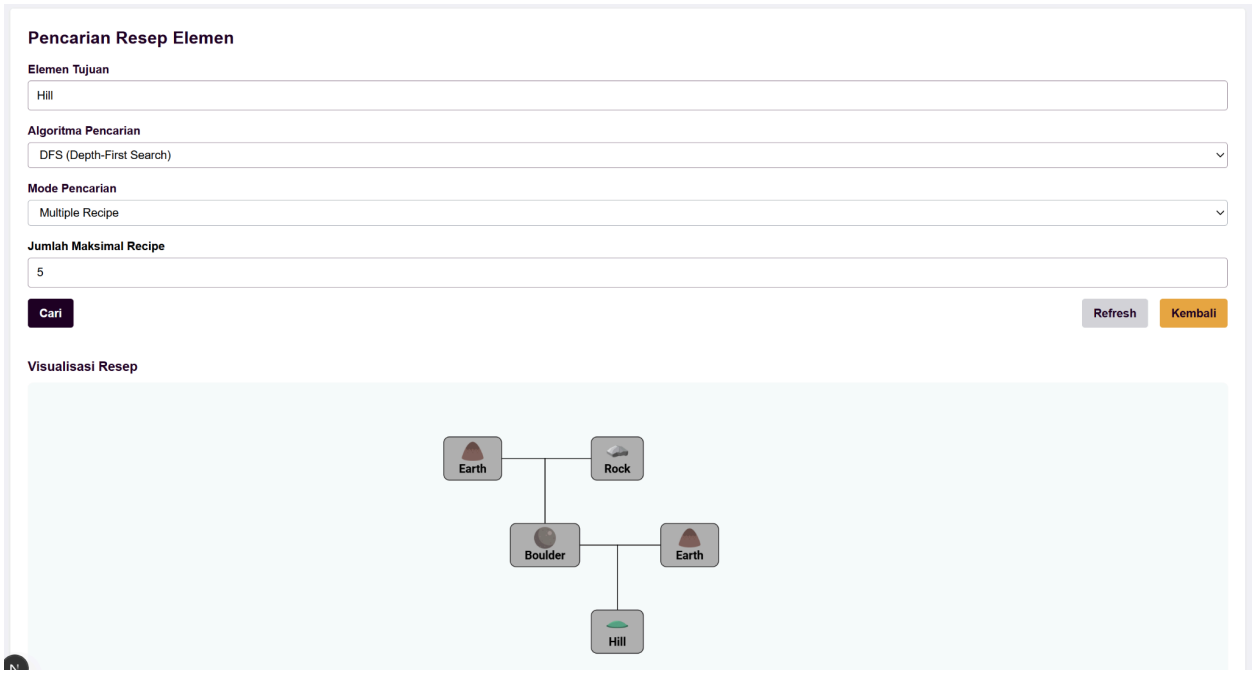
4.3. Hasil Pengujian

Catatan : screenshot menggunakan zoom out 50% agar bisa memuat semua informasi halaman dalam satu screenshot

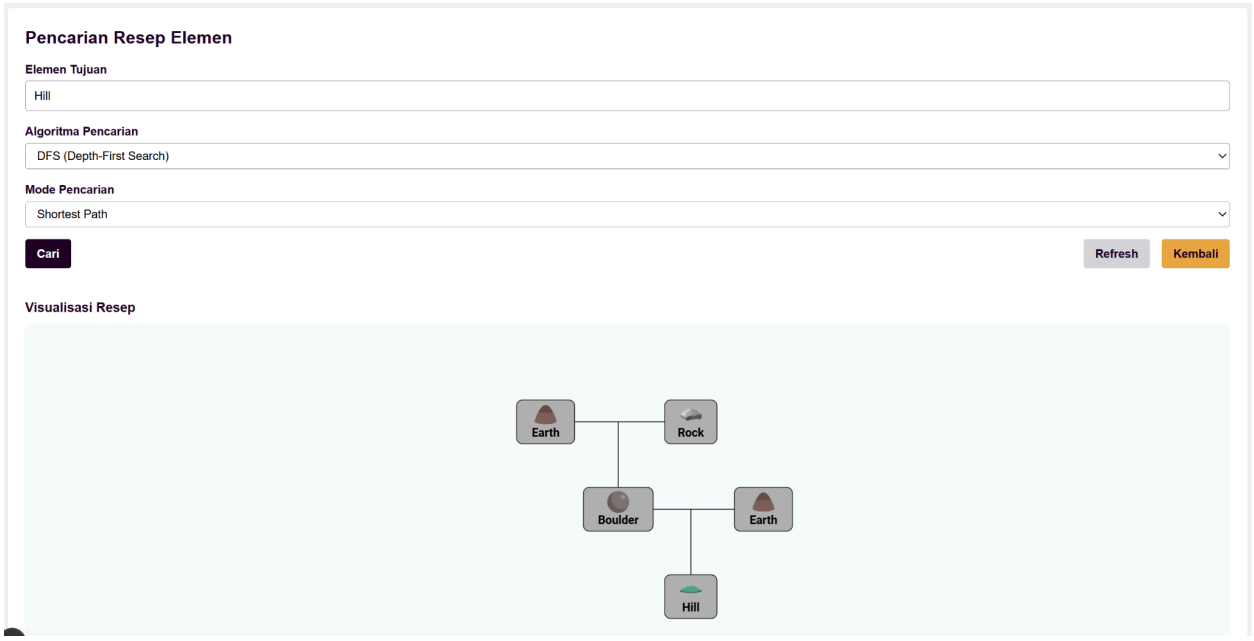
1. Mencari element Hill dengan DFS dan 1 resep.



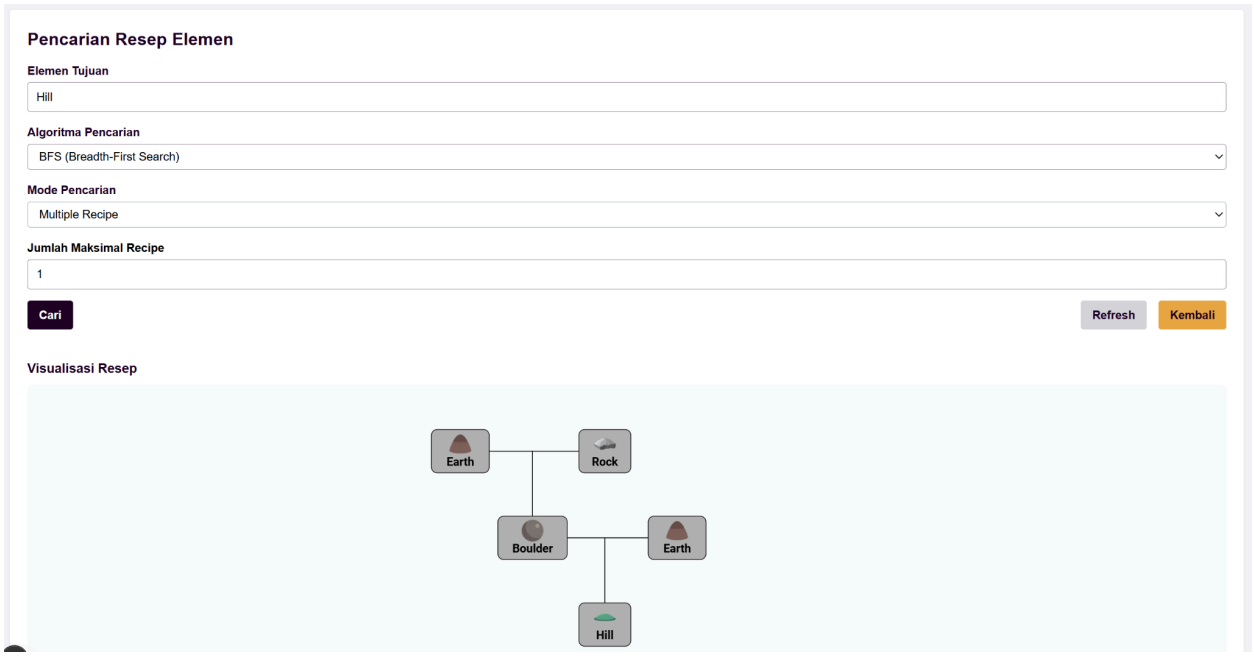
2. Mencari elemen Hill dengan DFS multiple recipe.



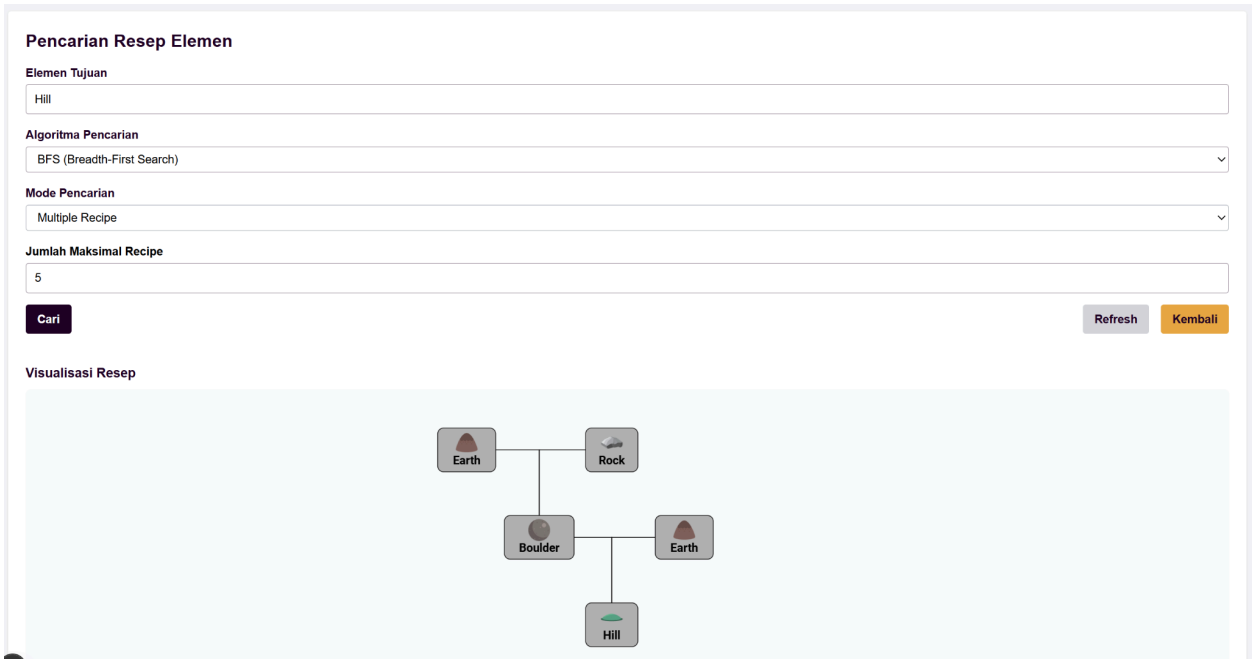
3. Mencari elemen Hill dengan DFS shortest path.



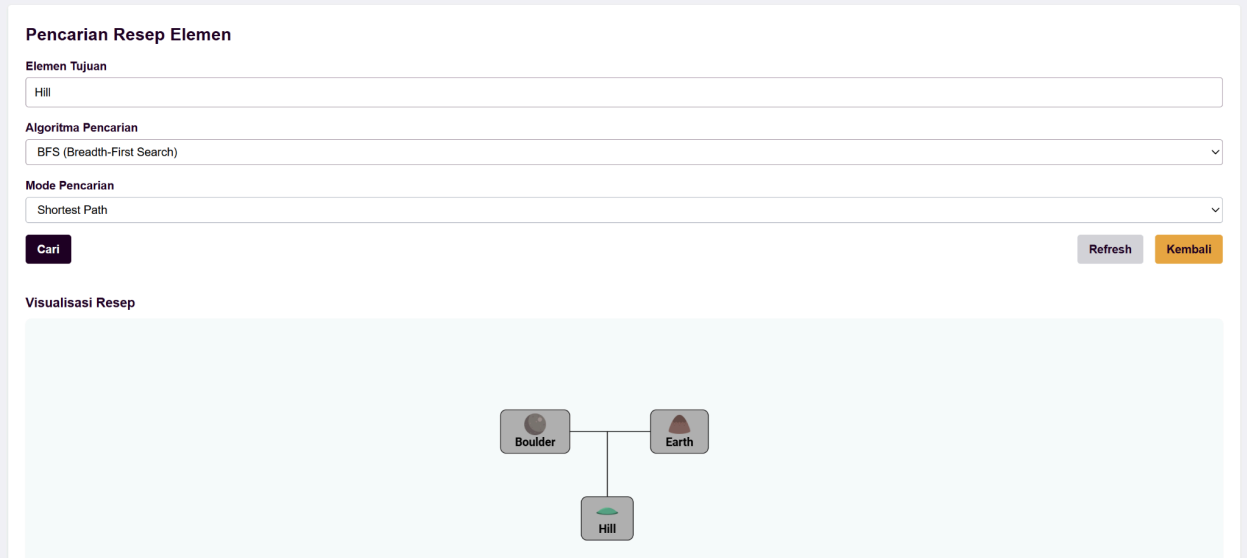
4. Mencari elemen Hill dengan BFS 1 resep.



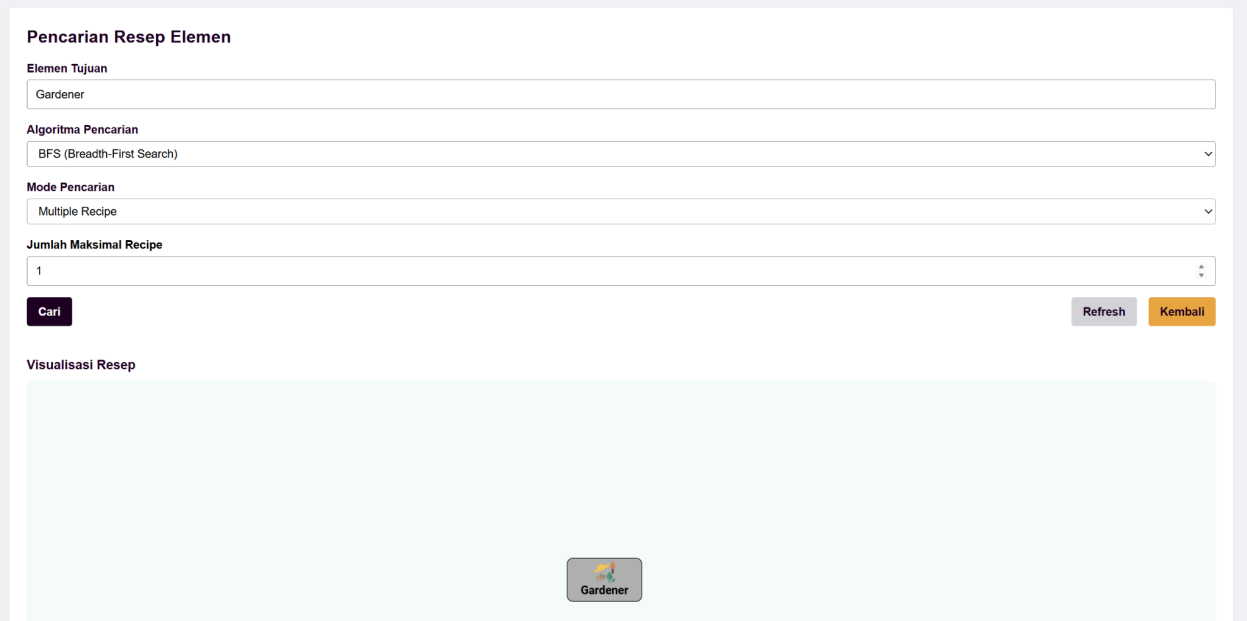
5. Mencari elemen Hill dengan BFS multiple recipe.



6. Mencari elemen Hill dengan BFS shortest path.



7. Mencari elemen Gardener dengan BFS 1 resep (semua resep terdiri dari tier yang lebih tinggi)



4.4. Analisis Hasil Pengujian

Pada pengujian pertama hingga ketujuh, dilakukan pengujian untuk setiap fitur yang tersedia. Dapat dilihat bahwa semuanya berhasil menunjukkan sesuai fungsi yang seharusnya. Terlihat bahwa pada BFS shortest path, simpul yang dijelajahi tidak sebanyak multiple recipe. Hal ini dikarenakan pencarian berakhir pada pertemuan pertama terhadap simpul yang dicari.

Pengujian ketujuh ada untuk menunjukkan bahwa semua yang ditampilkan hanyalah resep yang memenuhi “berasal dari tier yang lebih rendah atau sama dengan” sesuai dengan spek yang turun beberapa hari mendekati tenggat pengumpulan. Garden memiliki resep yang terdiri dari elemen-elemen dengan tier yang lebih besar.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pengerjaan dan proses pengembangan website ini, kami semakin memahami bagaimana algoritma BFS dan DFS bekerja, penerapannya dalam website, serta pembuatan website itu sendiri. Selain itu, kami memperoleh wawasan trivial tentang bagaimana caranya scraping website. Segala backend yang dikerjakan dalam Golang menambah wawasan tentang bahasa tersebut. Proses ini juga memberikan pemahaman lebih lanjut tentang implementasi strategi algoritma untuk mencari rumus elemen yang tepat menggunakan BFS dan DFS.

5.2. Saran

Program yang kami buat masih jauh dari sempurna. Untuk kedepannya, bisa lebih memperhatikan struktur program, flow program yang lebih jelas, dan juga menyisihkan waktu lebih banyak untuk bug fixing secara total.

5.3. Refleksi

Tugas besar ini telah memberikan kami pengalaman yang sangat mengesankan dari senang hingga tekanan. Kami menghadapi berbagai macam kendala seperti alokasi waktu yang kurang efektif akibat kesibukannya masing-masing, belum terbiasa dengan bahasa Golang, dan kinerja dari kami sendiri yang kurang baik. Untuk kedepannya, diharapkan tugas besar ini bisa menjadi motivasi untuk kami agar kami lebih berusaha kedepannya baik dalam pengerjaan tugas maupun menghadapi hidup.

LAMPIRAN

Pranala Repositori GitHub:

https://github.com/BrianHadianSTEI23/Tubes2_Help-Stima

Pranala Video:

https://youtu.be/jP2SG1xt7wk

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

DAFTAR PUSTAKA

"Little Alchemy 2"
https://littlealchemy2.com
Web permainan Little Alchemy 2.

"All elements in Little Alchemy 2"
https://little-alchemy.fandom.com/wiki/Elements_(Little_Alchemy_2)
Halaman wiki Little Alchemy 2 yang menampilkan seluruh kombinasi elemen (juga digunakan untuk scraping).

"Client-Server Architecture"
https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview
Penjelasan arsitektur Client-Server yang umum digunakan pada pengembangan aplikasi web.

"What is an API?"
https://aws.amazon.com/what-is/api/
Penjelasan apa itu API bagi yang memerlukan.

"Next Documentation"
https://nextjs.org/docs
Dokumentasi framework Next.js untuk Front end.

"React Documentation"
https://go.dev/doc/
Dokumentasi framework React.js untuk Front end.

"Golang Documentation"
https://go.dev/doc/
Dokumentasi bahasa Go untuk Back end.

"goquery"
https://github.com/PuerkitoBio/goquery
Kakas web scraping menggunakan bahasa Go.

“Effective Go Concurrency”

https://go.dev/doc/effective_go#concurrency

Dokumentasi Golang untuk mempelajari multithreading secara efektif.