IF2211 Strategi Algoritma

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

Laporan Tugas Besar

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
pada Semester IV Tahun Akademik 2024/2025

Dibuat Oleh NamaKelompok

Nicholas Andhika Lucas	13523014
Samantha Laqueenna Ginting	13523138
Naufarrel Zhafif Abhista	13523149

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR ISI 2
BAB 1 DESKRIPSI MASALAH 3
BAB 2 LANDASAN TEORI 7
A. Traversal Graf 7
B. Breadth First Search (BFS) 7
C. Depth First Search (DFS) 7
D. Bidirectional Search 8
E. Aplikasi Web 8
F. Docker 9
G. Multithreading 9
BAB 3 ANALISIS PEMECAHAN MASALAH 11
BAB 4 IMPLEMENTASI DAN PENGUJIAN 23
BAB 5 KESIMPULAN, SARAN, DAN REFLEKSI 29
LAMPIRAN 31
A. Github 31
B. Video 31
C. Tabel Pemeriksaan 31
REFERENSI 32

BAB 1 DESKRIPSI MASALAH

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.

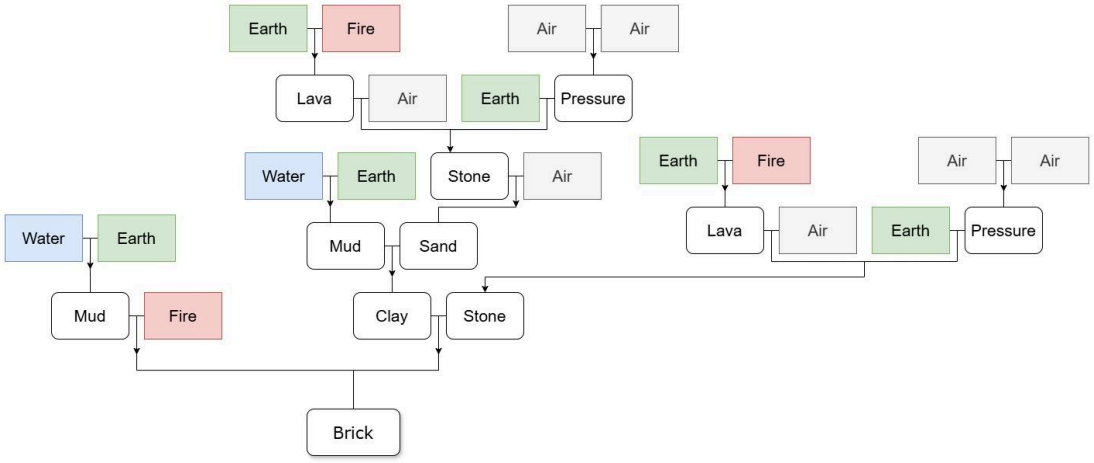
3. *Combine Mechanism*

   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.

Spesifikasi Bonus

- (maks 5) Membuat video tentang aplikasi BFS dan DFS pada permainan Little Alchemy 2 di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll. Semakin menarik video, maka semakin banyak poin yang diberikan.
- (maks 3) Aplikasi dijalankan menggunakan Docker baik untuk *frontend* maupun *backend.*
- (maks 3) Aplikasi di-*deploy* ke aplikasi *deployment* (aplikasi *deployment* bebas) agar bisa diakses secara daring
- (maks 3) Menambahkan algoritma bukan hanya DFS dan BFS, tetapi juga strategi bidirectional
- (maks 6) Aplikasi memiliki fitur *Live Update* visualisasi *recipe* selama proses pencarian. *Tree* visualisasi akan dibangun bertahap secara *real time* sesuai dengan progress pencarian. Tambahkan *delay* pada Live Update karena pencarian dapat berjalan dengan sangat cepat.

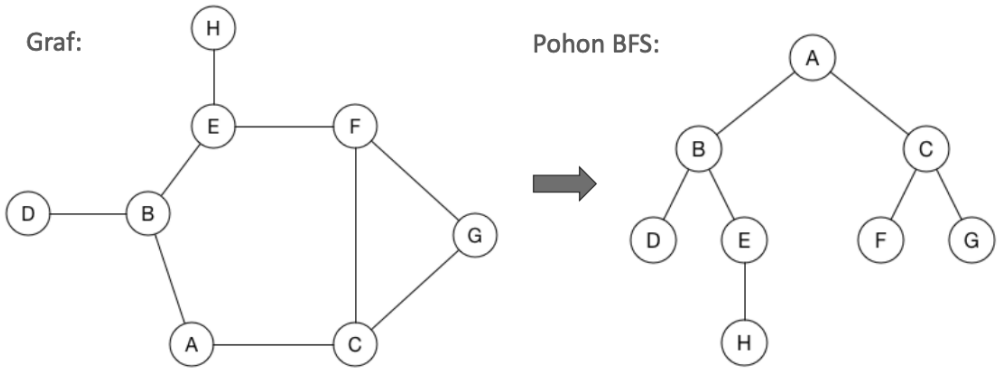
BAB 2 LANDASAN TEORI

A. Traversal Graf

Algoritma traversal graf adalah algoritma yang mengunjungi simpul-simpul di dalam graf terhubung dengan cara yang sistematik untuk mencari solusi persoalan. Metode ini dapat melakukan pencarian solusi dengan atau tanpa informasi tambahan. Salah satu tipe traversal graf tanpa informasi tambahan adalah Breadth First Search (BFS) dan Depth First Search (DFS).

B. Breadth First Search (BFS)

Misal ditentukan traversal graf dimulai dari simpul v. Algoritma pencarian melebar atau BFS dimulai dengan mengunjungi simpul v, kemudian mengunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu. Lalu, simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang telah dilewati dikunjungi. Proses ini terus dilakukan hingga tidak ada lagi simpul yang belum dikunjungi.



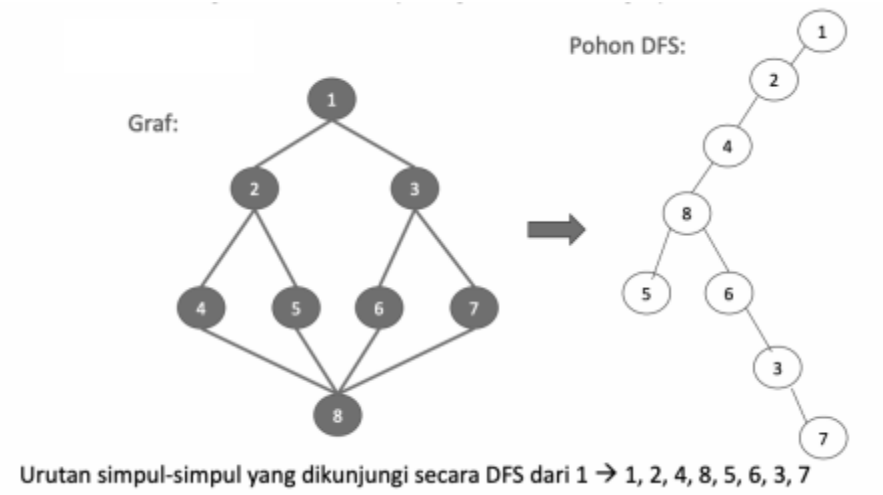
Urutan simpul-simpul yang dikunjungi secara BFS dari A → A, B, C, D, E, F, G, H

Gambar 4. Contoh Graf dan Pohon BFS

C. Depth First Search (DFS)

Misal ditentukan traversal graf dimulai dari simpul v. Algoritma pencarian mendalam atau DFS dimulai dengan mengunjungi simpul v, kemudian mengunjungi simpul w yang bertetangga dengan simpul v. Ulangi DFS mulai dari simpul w. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, lakukan pencarian backtrack ke simpul terakhir yang dikunjungi dan mempunyai simpul w yang belum dikunjungi. Pencarian akan berakhir

bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.



Gambar 5. Contoh Graf dan Pohon DFS

D. Bidirectional Search

Algoritma pencarian dua arah atau Bidirectional Search dimulai dengan mengunjungi dua simpul secara bersamaan: simpul awal v dan simpul tujuan t. Dari simpul v, algoritma melakukan ekspansi ke semua simpul yang bertetangga dengan v, sedangkan dari simpul t, algoritma melakukan ekspansi ke semua simpul yang bertetangga dengan t. Proses ini kemudian diulang untuk setiap simpul baru yang dikunjungi dari kedua arah. Dari arah maju, untuk setiap simpul yang dikunjungi, algoritma mengunjungi semua tetangganya yang belum dikunjungi oleh pencarian maju. Dari arah mundur, untuk setiap simpul yang dikunjungi, algoritma mengunjungi semua tetangganya yang belum dikunjungi oleh pencarian mundur. Setelah setiap ekspansi, algoritma memeriksa apakah ada simpul yang telah dikunjungi oleh kedua pencarian. Jika ditemukan simpul u yang telah dikunjungi oleh kedua pencarian, maka algoritma berhenti dan mengkonstruksi jalur dari v ke t melalui simpul u tersebut. Jalur ini dibentuk dengan menggabungkan jalur dari v ke u (dari pencarian maju) dengan jalur dari u ke t (dari pencarian mundur yang dibalik). Pencarian akan berakhir ketika kedua pencarian bertemu di tengah, atau ketika semua simpul yang mungkin telah dieksplorasi tanpa menemukan jalur yang menghubungkan kedua simpul.

E. Aplikasi Web

Website dirancang dengan arsitektur *client-server*, dengan pemisahan antara antarmuka pengguna (*frontend*) dan logika dan pengolahan data (*backend*). *Backend* dari *website* dibangun menggunakan bahasa Go, yang dikenal modern dan efisien untuk pengembangan web dan layanan

API (Application Programming Interface). Salah satu keunggulan bahasa Go adalah fitur Goroutine. Goroutine adalah sebuah unit eksekusi ringan yang memungkinkan multithreading. Dengan proses multithreading, *backend* mampu menangani banyak permintaan secara bersamaan sehingga kinerja *server* meningkat.

Bagian frontend dibangun menggunakan React JS, sebuah pustaka JavaScript yang digunakan untuk membangun antarmuka pengguna. Keunggulan dari React JS adalah setiap bagian dari antarmukanya dapat dibagi menjadi komponen yang modular. Karena fitur tersebut, React sangat cocok untuk aplikasi web yang dinamis. Form input untuk pencarian disediakan dalam pustaka React-hook-form, yang dapat mengelola form dengan *handling* konvensional (misalnya dengan useState). Tidak hanya itu, pustaka React-d3-tree memberikan tampilan struktur data berbentuk hirarki pohon yang interaktif dan dinamis.

Frontend dan *backend* dihubungkan menggunakan REST API berbasis HTTP. API atau Application Programming Interface berfungsi sebagai jembatan komunikasi yang memungkinkan *frontend* mengirimkan data untuk diproses ke *backend*, dan menerima hasil olahan data yang akan ditampilkan di antarmuka.

F. Docker

Docker adalah platform virtualisasi berbasis kontainer yang memungkinkan pengembang untuk mengemas aplikasi beserta seluruh dependensinya (kode, runtime, pustaka sistem, dan pengaturan) ke dalam unit standar yang disebut kontainer. Kontainer ini bersifat ringan, portabel, dan konsisten, sehingga aplikasi dapat berjalan dengan cara yang sama di lingkungan yang berbeda—dari laptop pengembang hingga server produksi—tanpa khawatir tentang perbedaan sistem operasi atau konfigurasi. Docker menggunakan arsitektur klien-server dengan daemon Docker yang mengelola kontainer dan Docker CLI untuk interaksi pengguna, sementara Docker Hub menyediakan repositori publik untuk berbagi dan mendistribusikan gambar kontainer.

G. Multithreading

Go (Golang) mengimplementasikan multithreading melalui fitur bernama goroutine, yaitu thread ringan yang dikelola oleh runtime Go, bukan oleh sistem operasi. Goroutine memiliki keunggulan berupa overhead memori yang sangat kecil (sekitar 2KB per goroutine), memungkinkan pembuatan ribuan goroutine secara bersamaan tanpa menimbulkan masalah kinerja. Untuk menjalankan goroutine, cukup menambahkan kata kunci go sebelum pemanggilan fungsi. Komunikasi antar goroutine tidak menggunakan memori bersama dan mutex seperti pada bahasa lain, melainkan menggunakan channels yang mengimplementasikan filosofi "jangan berbagi memori untuk

berkomunikasi; komunikasilah untuk berbagi memori". Go juga menyediakan package sync untuk primitive sinkronisasi seperti mutex dan waitgroup ketika diperlukan. Runtime Go menggunakan scheduler yang canggih untuk mendistribusikan goroutine ke thread OS yang tersedia, menerapkan multiplexing M:N di mana M goroutine dijalankan pada N thread OS. Model ini, yang disebut "concurrency with parallelism", membuat pemrograman konkuren di Go menjadi lebih mudah, aman, dan efisien dibandingkan pendekatan multithreading tradisional.

BAB 3 ANALISIS PEMECAHAN MASALAH

Algoritma traversal graf yaitu BFS dan DFS, digunakan untuk menyelesaikan persoalan pencarian resep pada permainan Little Alchemy 2. Dalam analisis penyelesaian masalah, kami menemukan beberapa *key insight*:

1. Karena BFS lebih bersifat menyeluruh (level-per-level), dapat digunakan *Forward Search* dalam implementasinya.
2. Karena DFS lebih fokus pada kedalaman, sulit jika pencarian dimulai dari target (Karena jadinya kita harus set kedalaman maksimal). Jadi, lebih aman dengan *Backward Search*.
3. Ada aturan revisi dari spek, yakni item hanya bisa dibuat dari item dengan tier yang lebih rendah. Ini *gamechanger* untuk DFS, karena kita bisa menghindari siklus dengan ini.
4. Poin satu dan dua hanya menyinggung DFS-BFS untuk *single recipe*. Dalam beberapa waktu, belum terbayang cara kerja algoritma dalam *multiple recipe*. Hingga kemudian mendapat *insight* tambahan: DFS itu bisa *backtracking*! Apa artinya? Kita bisa *kick-off* DFS yang satu resep, baru kemudian menjelajahi kemungkinan lain dari leaf, dengan cara *backtrack* ke kemungkinan lain, ketika suatu node sudah dipakai dalam satu resep. Ini bisa *backtrack* terus, sampai kita sampai ke node target. Hasilnya jadi berbagai resep yang mungkin.
5. Dari ide ini, kebayang juga, ide untuk BFS. Masih sama awalannya, kita *kick-off* BFS yang satu resep, kemudian dari target, satu persatu kita masukkan ke queue. Baru nanti kita telaah *alternative path*-nya, berdasarkan elemen yang di-*pop*.
6. Tentang bonus Bidirectional, idenya tidak berbeda jauh dengan BFS. Hanya saja, diberikan tambahan tambahan dari target ke base + base ke target, dengan pencarian level-per-level.

Sebelum melangkah ke perancangan algoritma, perlu diketahui bahwa karena terdapat aturan nomor 3 dan tidak adanya elemen Time, terdapat beberapa elemen yang memiliki 0 resep:

1. Clock
2. Death
3. Dinosaur
4. Family tree
5. Peat
6. Skeleton
7. Sloth
8. Tree

Supaya algoritma dapat berjalan lancar untuk elemen yang lain, maka elemen dengan 0 resep ini akan dianggap sebagai **BaseElement.** Dengan ini, kita bisa melangkah ke ide implementasi mendalam untuk setiap spesifikasi:

A. Single Recipe Breadth First Search (BFS)

1. Algoritma menjalankan loop BFS, dimana setiap elemen dalam antrian diproses secara berurutan mulai dari indeks pertama. Untuk setiap elemen yang diproses (disebut current), algoritma pertama memeriksa apakah elemen tersebut adalah target yang dicari. Jika ya, pencarian dihentikan karena jalur terpendek ke target telah ditemukan. Jika bukan, algoritma melanjutkan untuk mencoba membuat elemen baru dari elemen saat ini.
2. Untuk setiap elemen saat ini, algoritma mencoba mengkombinasikannya dengan semua elemen yang telah dilihat sebelumnya (dalam map seen). Ini dilakukan dengan loop melalui semua elemen yang telah dilihat dan memeriksa apakah pasangan ini dapat membentuk produk baru. Algoritma menggunakan map combinations untuk memeriksa apakah kombinasi pasangan elemen menghasilkan produk yang valid.
3. Ketika menemukan kombinasi valid, algoritma menerapkan aturan penting: produk yang dihasilkan harus berada pada tier yang lebih tinggi daripada kedua elemen bahan (source dan partner). Jika tidak, kombinasi tersebut diabaikan. Pemeriksaan ini memastikan alur pencarian logis, dimana elemen kompleks dibuat dari elemen yang lebih sederhana, bukan sebaliknya. Tier setiap elemen diambil dari tierMap yang disediakan sebagai parameter.
4. Jika kombinasi menghasilkan produk yang valid dan memenuhi aturan tier, algoritma memeriksa apakah produk tersebut belum pernah dilihat sebelumnya. Jika belum, produk ini merupakan penemuan baru dan akan ditambahkan ke tiga tempat: ditandai dalam map seen sebagai telah dilihat, dicatat dalam map prev bagaimana cara membuatnya (elemen sumber dan partner), dan ditambahkan ke ujung antrian untuk dieksplorasi lebih lanjut pada iterasi berikutnya.
5. Algoritma berhenti ketika target ditemukan atau ketika seluruh antrian telah diproses tanpa menemukan target. Hasil akhirnya adalah map prev yang berisi informasi tentang bagaimana setiap elemen (termasuk target jika ditemukan) dibuat dari pasangan elemen lain. Dari struktur ini, dapat direkonstruksi resep lengkap untuk membuat elemen target dari elemen-elemen dasar melalui pohon resep.

Untuk *single recipe* BFS,

- Setiap elemen dalam antrian diproses sekali
- Untuk setiap elemen yang diproses, algoritma mencoba mengkombinasikannya dengan semua elemen yang telah dilihat
- Jumlah elemen yang dilihat meningkat selama proses pencarian

- Dalam kasus terburuk, semua kemungkinan elemen (E) akan ditemukan

Kompleksitas Waktu Terburuknya adalah $O(E^2)$, dimana E adalah jumlah total elemen yang mungkin dalam sistem. Ini karena untuk setiap elemen yang diproses (maksimal E elemen), kita mungkin perlu memeriksa kombinasi dengan semua elemen yang telah dilihat (yang juga bisa mencapai E elemen)

Sedangkan untuk ruang,

1. Queue: Menyimpan elemen yang akan diproses - O(E) dalam kasus terburuk
2. Seen Map: Menyimpan semua elemen yang telah ditemui - O(E)
3. Prev Map: Menyimpan informasi resep untuk setiap elemen - O(E)

Total kompleksitas ruangnya adalah O(E) untuk penyimpanan keseluruhan.

B. Multiple Recipe Breadth First Search (BFS)

1. Algoritma dimulai dengan menentukan jumlah goroutine worker yang akan digunakan, dengan nilai default jumlah *core* CPU yang tersedia jika tidak ditentukan. Algoritma menginisialisasi struktur pelacakan, berupa:
    a. map untuk memantau elemen-elemen yang telah dikunjungi
    b. mutex untuk melindungi map ini selama akses bersamaan, dan menandai semua elemen dasar sebagai telah dikunjungi.
2. Algoritma mendapatkan resep pertama dengan memanggil ShortestBfs() untuk menemukan jalur valid untuk membuat elemen target. Resep awal ini berfungsi sebagai titik awal untuk variasi. Jika tidak ditemukan resep awal, algoritma langsung mengembalikan hasil kosong. Ketika resep awal ditemukan, semua elemen dalam resep ini ditandai sebagai dikunjungi dan resep itu sendiri disimpan dalam koleksi resep dan ditandai sebagai "telah dilihat" untuk menghindari duplikasi di masa mendatang.
3. Queue BFS diinisialisasi dengan elemen target sebagai elemen fokus pertama, diikuti oleh semua komponen non-dasar yang ditemukan dalam resep awal. Setiap item queue berisi baik resep (peta elemen ke pasangan sumber/pasangan mereka) dan elemen fokus yang akan menjadi subjek upaya variasi. Pengaturan queue ini memastikan bahwa algoritma akan terlebih dahulu menjelajahi variasi cara membuat elemen target itu sendiri, diikuti oleh variasi dalam cara membuat komponennya.
4. Beberapa goroutine worker dibentuk berdasarkan jumlah worker yang ditentukan. Worker-worker ini beroperasi secara independen tetapi berkoordinasi melalui struct

yang dilindungi oleh mutex. WaitGroup melacak worker aktif, saluran "done" memungkinkan sinyal semua worker untuk berhenti, dan fungsi pembantu mengelola operasi queue dengan aman di seluruh goroutine.

5. Worker memproses queue dalam batch, mengambil beberapa item sekaligus untuk mengurangi pertentangan pada queue bersama. Setiap worker terus mengambil batch hingga diberi sinyal untuk berhenti atau hingga queue kosong. Untuk setiap batch, worker memanggil processBatch(), yang memeriksa cara alternatif untuk membuat setiap elemen fokus.
6. Untuk setiap elemen fokus dalam batch, algoritma mengidentifikasi semua cara alternatif yang valid untuk membuatnya menggunakan map revCombinations. Untuk setiap kombinasi alternatif, algoritma membuat variasi resep baru dengan mengganti cara pembuatan elemen fokus sambil mempertahankan bagian lain dari resep. Jika alternatif memerlukan bahan yang belum ada dalam resep, algoritma secara rekursif mencari resep untuk bahan-bahan tersebut. Proses pembuatan variasi ini adalah inti dari algoritma, karena menjelajahi ruang kombinatorial dari resep yang mungkin.
7. Setiap variasi yang dihasilkan divalidasi untuk memastikan semua komponen memiliki resep yang valid. Variasi yang berhasil diperiksa terhadap resep yang sebelumnya terlihat untuk memastikan keunikan. Resep unik baru ditambahkan ke koleksi hasil, dan untuk setiap resep baru, semua komponen non-dasarnya ditambahkan ke queue untuk eksplorasi lebih lanjut. Pendekatan ini memastikan sifat breadth-first dari pencarian, menjelajahi semua variasi yang mungkin pada level saat ini sebelum bergerak lebih dalam.
8. Goroutine terpisah secara berkala memeriksa apakah proses harus dihentikan berdasarkan dua kondisi: jumlah maksimum resep telah ditemukan, atau queue kosong tanpa worker yang aktif memproses item. Ketika salah satu kondisi terpenuhi, goroutine memberi sinyal semua worker untuk berhenti melalui saluran "done". worker keluar dengan anggun dengan menggabungkan elemen yang dilacak secara lokal ke dalam set yang dikunjungi secara global.
9. Setelah semua worker selesai dan penghitung WaitGroup mencapai nol, algoritma merakit hasil akhir yang berisi semua variasi resep yang ditemukan dan jumlah total elemen yang dikunjungi. Hitungan ini berfungsi sebagai metrik seberapa luas algoritma mencari ruang kombinatorial. Resep yang dikumpulkan mewakili berbagai cara untuk membuat elemen target, dengan setiap resep menawarkan kombinasi bahan dan sub-resep yang berbeda.

Kompleksitas Waktu dari Multiple BFS:

- Inisialisasi dan Pencarian Awal: $O(E^2)$ dari panggilan ShortestBfs
- Eksplorasi Paralel:
    - Setiap worker memproses batch dari queue
    - Untuk setiap elemen fokus, algoritma mencari semua cara alternatif untuk membuatnya
    - Setiap alternatif menghasilkan variasi resep baru

Kompleksitas Terburuk: $O(E^2 + R \times A \times E)$, dimana:

- E = jumlah total elemen yang mungkin
- R = jumlah resep yang dicari (dibatasi oleh maxRecipes)
- A = rata-rata jumlah alternatif cara untuk membuat sebuah elemen

Kompleksitas Ruang dari Multiple BFS:

- Kumpulan Resep: $O(R \times E)$ - menyimpan hingga R resep dengan maksimal E elemen per resep
- Queue: $O(R \times E)$ - dalam kasus terburuk, bisa berisi banyak variasi resep yang menunggu diproses
- Tracking Maps:
    - seenRecipes: O(R) - menyimpan semua resep yang telah ditemukan
    - visited: O(E) - melacak semua elemen yang telah dijelajahi
- Overhead Paralelisasi
    - Worker Batches: $O(W \times B)$ dimana W adalah jumlah worker dan B adalah ukuran batch
    - Sinkronisasi Struktur: Mutex, WaitGroup, dan channel - overhead minimal

Kompleksitas Total: $O(R \times E + W \times B)$ untuk penyimpanan keseluruhan

C. Single Recipe Depth First Search (DFS)

1. Algoritma dimulai dengan membuat struktur data utama: map result untuk menyimpan resep yang ditemukan, map nodeStates untuk melacak status eksplorasi setiap elemen, dan map inProgress untuk deteksi siklus. Algoritma juga mengisi nodeStates dengan seluruh elemen dasar yang ditandai sebagai sudah dikunjungi, dan menambahkan elemen dasar ke map hasil dengan nilai resep kosong untuk menandakan bahwa elemen tersebut tidak perlu diuraikan lebih lanjut.
2. Sebelum memulai eksplorasi, algoritma memeriksa apakah elemen target dapat

dibuat dengan melihat apakah target ada dalam map revCombinations. Jika target tidak ada dalam daftar kombinasi, berarti tidak ada cara untuk membuatnya, dan algoritma langsung mengembalikan map kosong tanpa perlu melakukan pencarian lebih lanjut.

3. Algoritma mendefinisikan fungsi rekursif explore yang merupakan inti dari pencarian DFS. Fungsi ini menerima elemen yang sedang dieksplorasi dan mengembalikan boolean yang menunjukkan apakah elemen tersebut berhasil dibuat atau tidak. Fungsi ini akan dipanggil secara rekursif untuk setiap bahan dalam resep, membangun pohon eksplorasi secara mendalam.
4. Saat explore dipanggil untuk suatu elemen, algoritma pertama memeriksa kondisi dasar: apakah elemen tersebut adalah elemen dasar atau sudah pernah dikunjungi sebelumnya. Jika ya, eksplorasi untuk elemen ini selesai dan mengembalikan true. Algoritma juga memeriksa deteksi siklus dengan melihat apakah elemen sedang dalam proses eksplorasi (dalam map inProgress). Jika ya, mengembalikan false untuk mencegah rekursi tak terbatas.
5. Jika elemen belum pernah dieksplorasi, algoritma menyiapkan status node untuk elemen tersebut. Ini melibatkan pengambilan semua pasangan bahan yang dapat menghasilkan elemen dari revCombinations, penyaringan pasangan yang valid berdasarkan aturan tier (hanya menggunakan bahan dari tier lebih rendah), dan inisialisasi status eksplorasi.
6. Algoritma mencoba setiap pasangan bahan yang valid secara berurutan. Untuk setiap pasangan, algoritma:
    - Menyimpan resep sementara dalam map result
    - Secara rekursif mencoba menyelesaikan bahan pertama
    - Jika berhasil, secara rekursif mencoba menyelesaikan bahan kedua
    - Jika kedua bahan berhasil diselesaikan, menandai elemen sebagai telah dikunjungi dan mengembalikan true
7. Jika suatu pasangan bahan gagal diselesaikan (salah satu atau kedua bahan tidak dapat dibuat), algoritma melanjutkan ke pasangan berikutnya. Jika semua pasangan telah dicoba dan tidak ada yang berhasil, algoritma menghapus resep sementara dari map result dan mengembalikan false, menandakan bahwa elemen tidak dapat dibuat.
8. Melalui panggilan rekursif explore, algoritma membangun pohon resep dari bawah ke atas. Elemen dasar menjadi daun pohon, dan resep kompleks dibangun dengan menyelesaikan sub-resep terlebih dahulu. Pendekatan depth-first ini memastikan

algoritma menjelajahi satu jalur sepenuhnya sebelum mencoba jalur lain.

9. Setelah eksplorasi selesai, algoritma membersihkan map result dengan menghapus elemen yang bukan elemen dasar tetapi tidak memiliki resep yang valid (ditandai dengan sumber atau partner kosong). Akhirnya, algoritma mengembalikan map result yang berisi resep lengkap untuk membuat elemen target.

Kompleksitas Waktu Single DFS:

Faktor Utama:

- Rekursi dan Backtracking: Algoritma menggunakan pendekatan rekursif untuk menjelajahi pohon resep
- Percabangan: Setiap elemen dapat memiliki beberapa pasangan bahan yang menghasilkannya
- Kedalaman Pencarian: Ditentukan oleh panjang rantai resep dari elemen dasar ke target

Kompleksitas terburuknya adalah O(b^d) dimana:

- b = faktor percabangan (jumlah rata-rata pasangan valid per elemen)
- d = kedalaman maksimum pohon resep

Kompleksitas Ruang Single DFS:

Beberapa rancangan struktur data yang digunakan, meliputi:

1. result map: O(E) - menyimpan resep untuk setiap elemen
2. nodeStates map: O(E) - menyimpan status untuk setiap elemen
3. inProgress map: O(E) - dalam kasus terburuk, semua elemen dalam jalur saat ini
4. Stack rekursi: O(d) - dimana d adalah kedalaman maksimum pohon resep

Untuk nodeStates map, terdiri dari struct nodeState. Setiap entri NodeState menyimpan:

- Pairs: Semua pasangan yang dapat menghasilkan elemen - O(b)
- ValidPairs: Pasangan yang valid berdasarkan aturan tier - O(b)
- Metadata lainnya (indeks, status) - O(1)

Sehingga kompleksitas ruang totalnya adalah O(E*b + d), di mana:

- E = jumlah elemen total
- b = rata-rata jumlah pasangan per elemen
- d = kedalaman maksimum pohon resep

D. Multiple Recipe Depth First Search (DFS)

1. MultipleDfs dimulai dengan menentukan jumlah goroutine worker berdasarkan jumlah inti CPU yang tersedia jika tidak ditentukan secara eksplisit. Algoritma menyiapkan struktur data pelacakan dan sinkronisasi, termasuk map visited untuk melacak elemen yang telah dijelajahi, mutex untuk mengamankan akses konkurensi, dan atomic counter untuk menghitung jumlah resep yang ditemukan.
2. Algoritma memanggil ShortestDfs untuk menemukan resep awal yang valid untuk target. Jika tidak ditemukan resep awal, algoritma langsung mengembalikan hasil kosong. Resep awal ini berfungsi sebagai titik awal untuk mencari variasi. Semua elemen dalam resep awal ditandai sebagai telah dikunjungi dan resep ini ditambahkan ke kumpulan resep hasil serta ditandai sebagai "telah dilihat" untuk menghindari duplikasi.
3. Tidak seperti BFS yang menggunakan queue, MultipleDfs menggunakan stack untuk menyimpan elemen pekerjaan, mencerminkan karakteristik DFS yang mengeksplorasi secara mendalam sebelum ke arah lebar. Algoritma mengidentifikasi elemen-elemen dalam resep awal yang memiliki cara alternatif untuk dibuat, dan menambahkannya ke stack awal sebagai DFSWorkItem. Setiap item kerja berisi elemen fokus, resep dasar, dan map untuk melacak pasangan bahan yang telah dieksplorasi.
4. Algoritma membuat beberapa goroutine worker untuk memproses stack pekerjaan secara paralel. Struktur sinkronisasi termasuk WaitGroup untuk menunggu semua worker selesai, channel "done" untuk menghentikan worker, dan berbagai mutex untuk mengamankan akses ke struktur data bersama. Fungsi helper getWorkBatch dan addToWorkStack dirancang untuk mengakses stack pekerjaan dengan aman di lingkungan yang bersamaan.
5. Worker mengambil batch item dari bagian atas stack (LIFO - Last In First Out), yang merupakan karakteristik DFS. Pendekatan ini memastikan bahwa algoritma memprioritaskan eksplorasi mendalam dalam pohon pencarian. Setiap worker memproses batch menggunakan processWorkBatchAtomic, yang mengeksplorasi alternatif untuk elemen-elemen yang menjadi fokus.
6. Untuk setiap elemen fokus, algoritma memeriksa semua pasangan bahan alternatif yang valid. Untuk setiap pasangan yang berbeda dari resep asli dan belum dieksplorasi, algoritma membuat variasi resep baru. Pendekatan DFS ini memprioritaskan eksplorasi mendalam untuk satu elemen fokus dan semua alternatifnya sebelum berpindah ke elemen fokus lainnya, berbeda dengan BFS yang

akan memperluas semua elemen fokus di level yang sama terlebih dahulu.

7. Setelah membuat variasi, algoritma memastikan bahwa semua bahan yang diperlukan memiliki resep valid. Jika bahan tidak memiliki resep, algoritma mencoba mencari resep untuk bahan tersebut. Algoritma juga melakukan validasi dan perbaikan untuk memastikan bahwa perubahan pada satu elemen tidak menyebabkan inkonsistensi dalam resep secara keseluruhan. Proses validasi ini penting untuk menjamin bahwa setiap variasi menghasilkan resep yang lengkap dan valid.
8. Setiap variasi resep valid yang unik ditambahkan ke hasil akhir dan atomic counter diperbarui. Algoritma kemudian menambahkan item kerja baru ke stack untuk elemen-elemen non-dasar dalam resep baru, memperluas pencarian lebih dalam. Pendekatan DFS ini terus menggali alternatif untuk resep yang baru ditemukan sebelum menyelesaikan eksplorasi resep yang ditemukan sebelumnya, memungkinkan algoritma cepat menemukan variasi yang sangat berbeda.
9. Sebuah goroutine terpisah secara berkala memeriksa kondisi terminasi: apakah jumlah maksimum resep telah tercapai atau stack pekerjaan kosong. Ketika salah satu kondisi terpenuhi, semua worker diberi sinyal untuk berhenti. Setelah semua worker selesai, algoritma mengembalikan hasil berupa kumpulan resep yang ditemukan dan jumlah node yang dikunjungi, memberikan informasi tentang variasi resep dan ruang pencarian yang dieksplorasi.

Kompleksitas Waktu Multiple DFS:

1. Pencarian DFS Paralel: O(b^d / W) dimana:
    a. b = faktor percabangan (jumlah alternatif per elemen)
    b. d = kedalaman maksimum pohon resep
    c. W = jumlah worker
2. Validasi dan Perbaikan Resep: O(E) per variasi, dimana E adalah jumlah elemen dalam resep

Sehingga kompleksitas keseluruhannya O((b^d × E) / W) dalam kasus terburuk.

Kompleksitas RuangMultiple DFS:

1. Kumpulan Resep: O(R × E) - menyimpan hingga R resep dengan maksimal E elemen per resep
2. Work Stack: O(E × R) - dalam kasus terburuk, stack dapat berisi banyak elemen fokus untuk banyak variasi resep
3. Struktur Pelacakan:

- seenRecipes (sync.Map): O(R) - melacak resep unik yang telah ditemukan
- visited: O(E) - melacak elemen yang telah dijelajahi
- ExploredPairs: O(b × E) - melacak pasangan yang telah dieksplorasi

4. Struktur Sinkronisasi: Mutex, WaitGroup, atomic counter - overhead minimal
5. Memori Worker Lokal: O(W × (E + b)) per worker untuk penyimpanan lokal

Kompleksitas Totalnya adalah O(R × E + W × (E + b)) untuk penyimpanan keseluruhan.

E. Single Recipe Bidirectional Search

1. Algoritma menyiapkan dua antrian dan struktur pelacakan terpisah: satu untuk pencarian maju dari elemen dasar, satu untuk pencarian mundur dari target.
2. Dari elemen dasar, algoritma menggabungkan elemen-elemen yang sudah ditemukan untuk membuat elemen baru. Setiap produk valid dicatat dalam forwardRecipes dan ditambahkan ke antrian maju.
3. Dari target, algoritma mengidentifikasi pasangan bahan yang dapat menghasilkan elemen saat ini. Bahan-bahan ini ditambahkan ke antrian mundur dan hubungannya dicatat dalam backwardRecipes.
4. Algoritma terus melakukan pencarian dua arah sampai menemukan elemen yang muncul di kedua arah. Elemen ini menjadi titik temu yang menghubungkan kedua jalur.
5. Jika titik temu ditemukan, algoritma menggabungkan resep dari kedua arah. Jalur mundur direkonstruksi dari titik temu ke target menggunakan completeBackwardPath.
6. Jika ada "celah" dalam rekonstruksi jalur mundur, algoritma memanggil ShortestDfs untuk menyelesaikannya. Ini memastikan jalur lengkap dari elemen dasar ke target.

Kompleksitas Waktu:

- O(b^(d/2)) di mana b adalah faktor percabangan dan d adalah kedalaman pencarian
- Pencarian dua arah secara signifikan lebih efisien daripada pencarian satu arah yang memiliki kompleksitas O(b^d)
- Pencarian berakhir ketika dua arah bertemu di tengah, biasanya setelah sekitar d/2 langkah dari masing-masing arah

Kompleksitas Ruang:

- O(b^(d/2)) untuk menyimpan state pencarian kedua arah

- Membutuhkan lebih banyak memori daripada DFS tetapi memberikan hasil yang lebih cepat
- Struktur utama: dua queue, dua set visited elements, dan dua map resep

F. Multiple Recipe Bidirectional Search

1. Algoritma menentukan jumlah goroutine pekerja dan menyiapkan struktur sinkronisasi seperti mutex dan atomic counter.
2. Algoritma memanggil ShortestBidirectional untuk mendapatkan resep pertama sebagai titik awal. Jika tidak ditemukan resep awal, algoritma langsung mengembalikan hasil kosong.
3. Antrian awal dibuat dengan elemen target sebagai fokus pertama, diikuti komponen non-dasar lainnya. Setiap item berisi resep lengkap dan elemen yang menjadi fokus untuk variasi.
4. Beberapa goroutine pekerja dibuat untuk memproses antrian secara bersamaan dengan koordinasi melalui mutex. Fungsi-fungsi helper seperti getBatch dan addToQueue memastikan akses aman ke antrian bersama.
5. Pekerja mengambil batch item dari antrian dan memprosesan menggunakan bidirectional search. Pendekatan batch mengurangi overhead sinkronisasi dan meningkatkan efisiensi.
6. Untuk setiap elemen fokus, algoritma mencari cara alternatif untuk membuatnya menggunakan pencarian dua arah. Ini menggabungkan kelebihan pencarian maju dan mundur untuk menemukan variasi lebih efisien.
7. Untuk bahan yang belum memiliki resep, algoritma memanggil findIngredientRecipeBidir yang menggunakan pencarian bidirectional terfokus. Ini memastikan semua komponen memiliki resep valid dengan pendekatan divide and conquer.
8. Algoritma mengumpulkan resep unik dan memantau kondisi terminasi: maksimum resep tercapai atau antrian kosong. Pendekatan ini memastikan algoritma berhenti tepat waktu dengan sumber daya yang efisien.

Kompleksitas Waktu:

O(b^(d/2) × R / W) dimana b adalah faktor percabangan, d adalah kedalaman maksimum, R adalah jumlah resep yang dicari, dan W adalah jumlah worker. Pendekatan bidirectional mengurangi kompleksitas eksponensial menjadi akar kuadrat dari pencarian satu arah,

sementara paralelisasi lebih lanjut meningkatkan throughput

Kompleksitas Ruang:
O(R × E + b^(d/2))** untuk menyimpan R resep dengan E elemen dan state pencarian bidirectional. Struktur utama termasuk: recipes array, queue, visited map, dan seenRecipes concurrent map

BAB 4 IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

a. Struktur Data

Nama	Deskripsi
`Element`	Struct berisi bahan penyusun elemen.
`map[string]Element`	Representasi satu resep lengkap. Key = elemen, Value = komposisinya.
`MultipleRecipesResult`	Struct hasil akhir pencarian.
`TreeNode`	Struktur data untuk menampung hasil tree sehingga dapat dikirim ke frontend.
`SearchRequest`	Struct untuk menampung input pencarian dari frontend: namaResep, maksimalResep, algoritma, dan modePencarian.

b. Fungsi Utama

Nama	Deskripsi
`MultipleBfs()`	Pencarian dengan algoritma BFS paralel, menggunakan mutex dan Goroutine.
`MultipleDfs()`	Pencarian DFS multithreaded, memakai stack dan atomic counter untuk pembatasan.
`MultipleBidirectional()`	Pencarian dua arah (bidirectional) untuk efisiensi menemukan resep, menggunakan channel done dan atomic.
`MultipleParallelDfs()`	Versi ringan dari DFS paralel, tiap elemen diproses oleh worker berbeda.
`BuildTree()`, `BuildMultipleTrees()`	Mengubah hasil resep menjadi struktur pohon untuk divisualisasikan di frontend.

c. Fungsi Pendukung

Nama	Deskripsi
`ShortestBfs()`, `ShortestDfs()`, `ShortestBidirectional()`	Fungsi awal untuk mencari jalur/resep termudah.

`RecipeToString()`	Mengubah resep jadi string unik untuk mendeteksi duplikasi.
`isUniqueRecipe()`	Mengecek apakah kombinasi resep sudah pernah ditemukan.
`findIngredientRecipe()`	Menemukan resep untuk bahan yang belum memiliki komposisi valid.
`repairRecipeAfterChange()`	Memastikan resep tetap valid setelah dilakukan modifikasi.
`searchHandler()`	Fungsi endpoint. Menerima permintaan POST dari frontend, menjalankan algoritma yang dipilih dan mengembalikan data.
`main()`	Menjalankan HTTP server pada port 8080 dan mendaftarkan endpoint pencarian.
`sync.WaitGroup`	Sinkronisasi semua Goroutine worker agar menunggu hingga selesai.
`sync.Mutex, sync.Map`	Mencegah race condition saat mengakses variabel bersama seperti queue, visited, dan recipes.
`channel done`	Mengirim sinyal berhenti ke seluruh worker ketika pencarian sudah mencukupi.
`atomic counter`	Menjaga agar jumlah resep yang dihasilkan tidak melebihi maxRecipes.

B. Tata Cara Penggunaan Program

Desain *website* ini mengusung tema *pixel art* yang terinspirasi dari permainan Growtopia dan Minecraft. Saat pertama kali mengakses *website*, pengguna akan diarahkan ke halaman beranda (*homepage*). Di halaman ini, pengguna dapat langsung menekan tombol Call to Action (CTA) untuk melakukan pencarian resep, atau menjelajahi halaman-halaman lainnya.

Halaman utama dari *website* ini adalah halaman "Cari Resep", di mana pengguna akan menemukan sebuah form yang terdiri dari input Nama Resep, Jumlah Resep, Algoritma Pencarian, Mode Pencarian, serta Mode DFS (yang muncul jika pengguna memilih algoritma DFS). Setelah menekan tombol “Submit”, sistem akan memproses *query* dari pengguna dan menampilkan hasil pencarian berupa visualisasi pohon resep.

Selain itu, terdapat halaman "Lihat Bahan" yang memungkinkan pengguna untuk mengeksplorasi berbagai bahan yang tersedia dalam permainan Little Alchemy 2. Terakhir, terdapat

halaman "Tentang Kami" yang berisi informasi mengenai *website* yang dikembangkan serta deskripsi singkat dari masing-masing anggota kelompok yang terlibat dalam proyek ini.

C. Pengujian dan Analisis

a. Pengujian 1

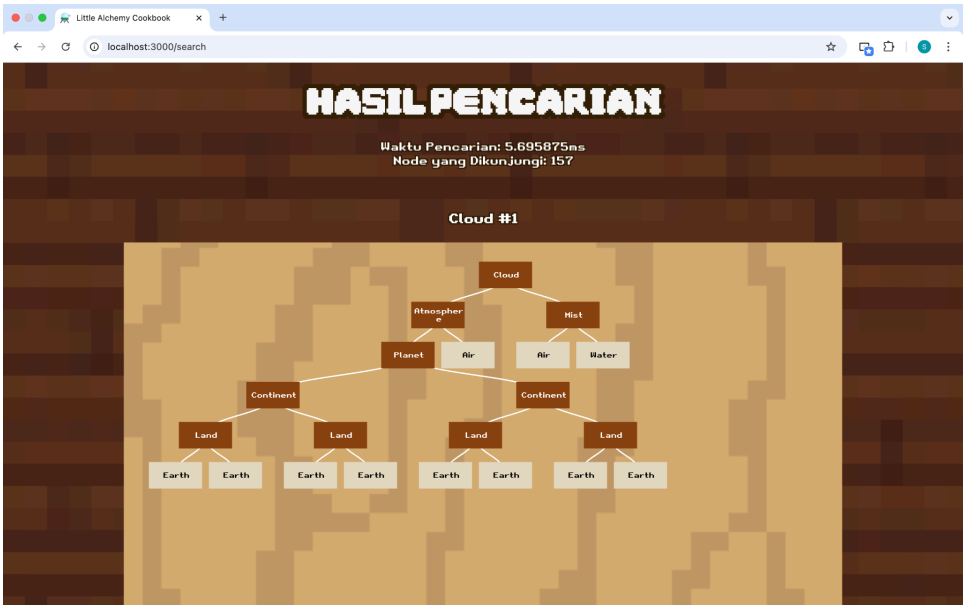
i. Nama resep : Cloud

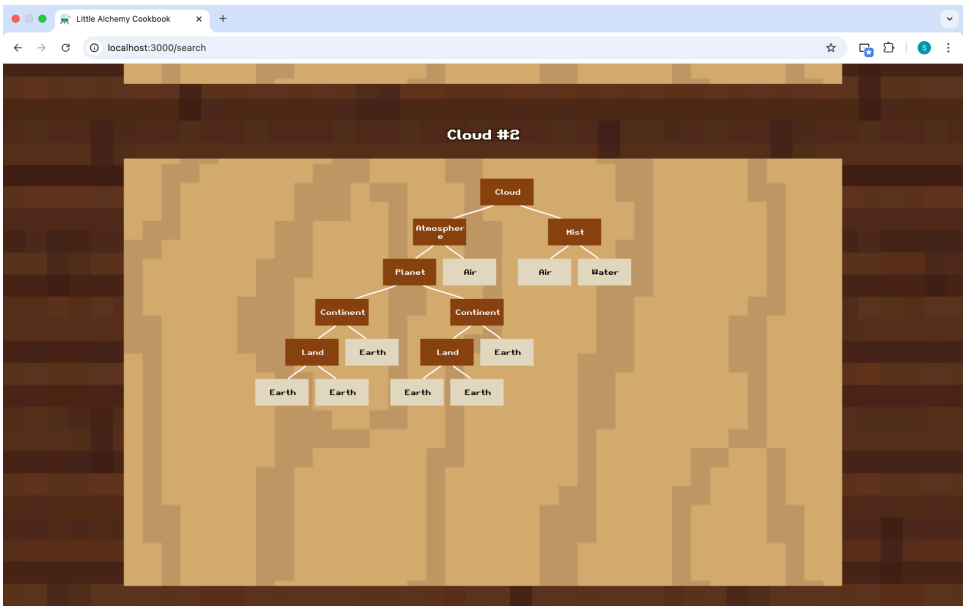
ii. Jumlah : 4

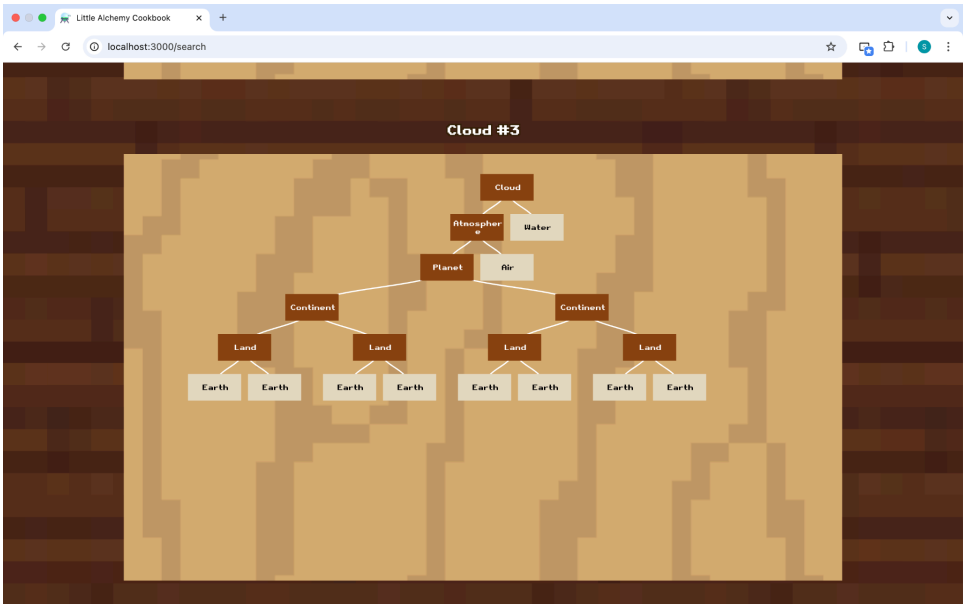
iii. Algoritma : BFS

iv. Mode pencarian : Multiple

v. Lampiran hasil :







Gambar 6. Pengujian 1

vi. Analisa : Pencarian elemen Cloud berjumlah 4 resep menggunakan algoritma BFS dilakukan. Cloud merupakan elemen Tier 5 dan keempat hasil menampilkan resep-resep yang berbeda.

b. Pengujian 2

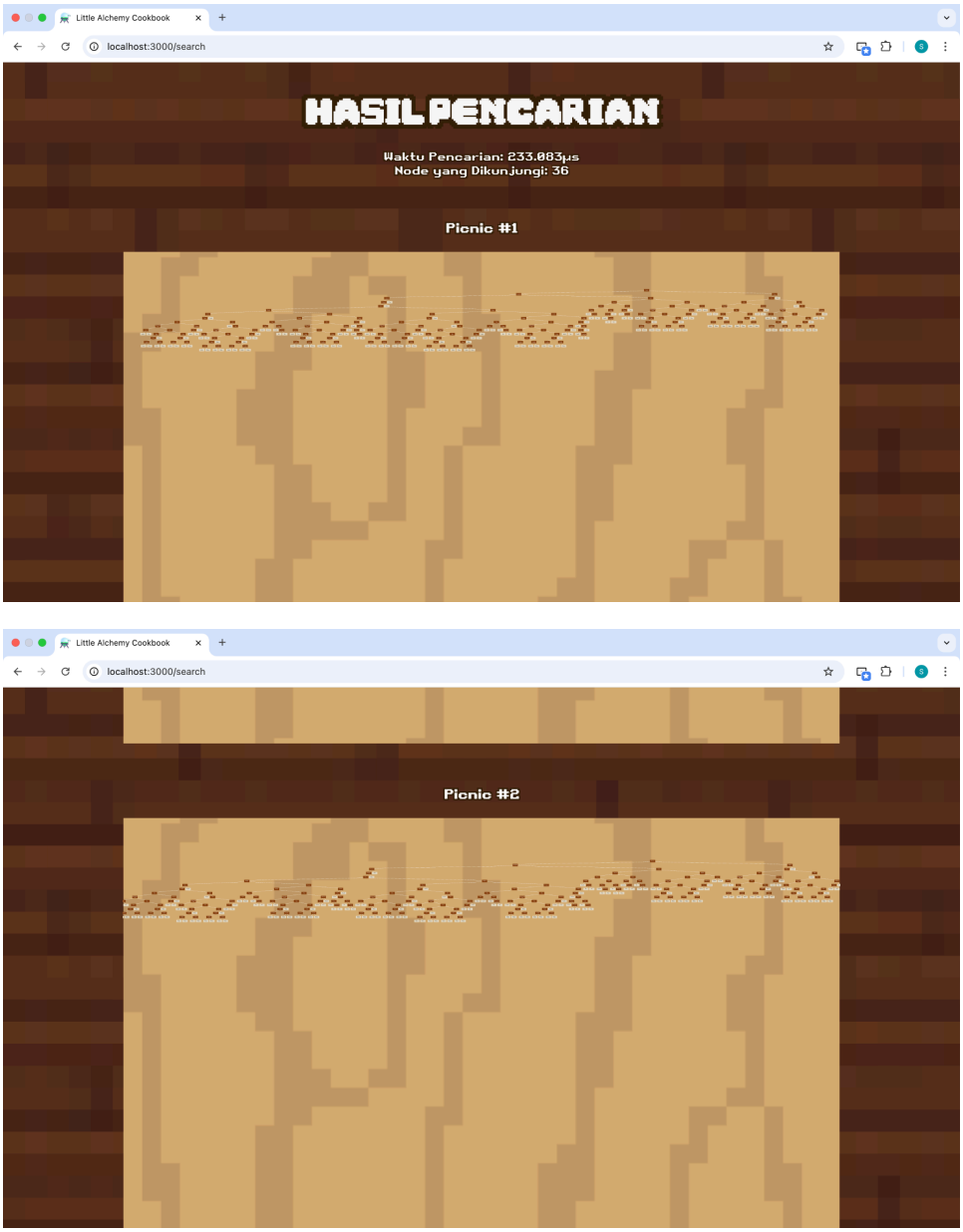
i. Nama resep : Picnic

ii. Jumlah : 2

iii. Algoritma : DFS

iv. Mode pencarian : Multiple

v. Lampiran hasil :



Gambar 7. Pengujian 2

vi. Analisis : Pencarian elemen Picnic berjumlah dua resep menggunakan algoritma DFS dilakukan. Picnic merupakan elemen Tier 15, sehingga kedua *tree* yang unik cukup besar. *Website* menyediakan fitur zoom manual, sehingga untuk melihat tree dengan lebih jelas, pengguna dapat membuatnya lebih besar.

c. Pengujian 3

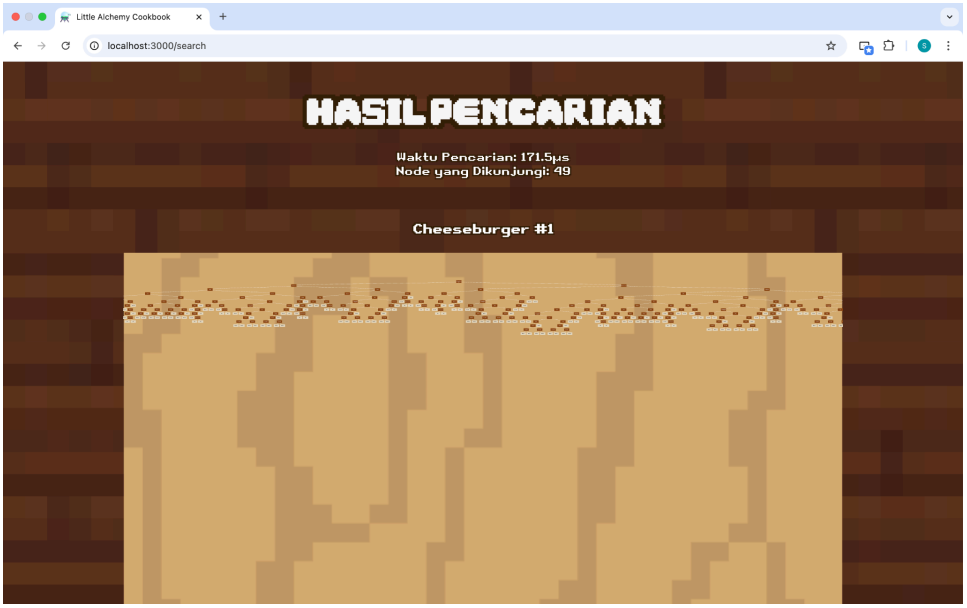
i. Nama resep : Cheeseburger

ii. Jumlah : 1

iii. Algoritma : Bidirectional BFS

iv. Mode pencarian : Single

v. Lampiran hasil :



Gambar 8. Pengujian 3

vi. Analisis : Pencarian elemen Cheeseburger menggunakan algoritma Bidirectional BFS dilakukan. Mode Single yang dipilih berarti, jumlah resep yang ditampilkan hanya satu. Cheeseburger merupakan elemen Tier 13, sehingga *tree* yang terbentuk cukup besar. *Website* menyediakan fitur zoom manual, sehingga untuk melihat tree dengan lebih jelas, pengguna dapat membuatnya lebih besar.

BAB 5 KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Dalam proyek Tugas Besar IF2211 Strategi Algoritma ini, kami telah mengimplementasikan pencarian melebar (BFS) dan mendalam (DFS) pada penyelesaian permainan Little Alchemy. Implementasi dilakukan dalam bahasa Go (*backend*), dan dibangun pada sebuah website menggunakan *framework* React (*frontend*). Multithreading digunakan agar proses eksplorasi berbagai kemungkinan alternatif resep untuk setiap elemen dalam suatu resep dapat dilakukan secara paralel, bukan satu per satu. Ini sangat meningkatkan efisiensi, terutama saat jumlah kombinasi sangat besar.

Dengan bantuan *framework* React dan REST API, pengguna dapat mencari seluruh resep yang dapat dibangun dari permainan Little Alchemy 2, memilih jumlah resep yang ingin dilihat, dan memilih algoritma serta mode pencarian. Hasil pencarian divisualisasikan dalam *tree* yang menunjukkan kombinasi elemen.

B. Saran

Bagi kelompok kami, Tugas Besar ini merupakan pengalaman yang cukup menarik. Pada Tugas Besar II mata kuliah IF2123 Aljabar Linier dan Geometri di Semester I tahun lalu, kami pernah mengerjakan proyek berbasis web dengan kebebasan dalam memilih framework dan bahasa pemrograman. Pengalaman tersebut membekali kami dengan pemahaman mengenai hal-hal yang perlu dipersiapkan untuk membangun sebuah website, termasuk bagaimana menghubungkan antara *backend* dan *frontend*.

Meskipun demikian, karena kami menemukan berbagai tantangan selama pengerjaan tugas besar ini, kami ingin memberikan saran pada perancang tugas besar sekaligus pembaca laporan ini.

1. Penjelasan Multithreading

   Dalam dokumen spesifikasi tugas ini, perancang tugas tidak menyertakan penjelasan, contoh, tutorial, maupun dokumentasi terkait multithreading yang dijadikan sebagai fitur optimasi proses pencarian. Hal ini membuat kelompok kami perlu melakukan eksplorasi mandiri untuk memahami konsep multithreading serta bagaimana mengimplementasikannya dalam bahasa Go. Oleh karena itu, kami menyarankan agar perancang tugas di masa mendatang dapat menyertakan penjelasan singkat, contoh implementasi, atau referensi yang relevan untuk membantu proses pengembangan. Mengingat tenggat waktu tugas ini yang cukup singkat, keberadaan dokumentasi tersebut akan sangat membantu dalam mempercepat pemahaman dan pengerjaan tugas.

2. Membaca Dokumentasi

   Untuk membangun web menggunakan *framework* dan API tertentu, penting bagi pengembang untuk terlebih dahulu membaca dokumentasi resmi sebelum mempelajari contoh-contoh kode program. Dengan memahami dokumentasi secara menyeluruh, pengembang dapat menggunakan fitur yang tersedia secara optimal, menyesuaikan dengan kebutuhan proyek, dan menghindari praktik-praktik yang tidak direkomendasikan. Setelah pemahaman dasar terbentuk, barulah contoh kode menjadi lebih bermakna dan efektif sebagai acuan implementasi.

3. Kerja Sama Kelompok

   Pengerjaan tugas akan lebih mudah dilalui jika kelompok telah membuat strategi terlebih dahulu. Strategi yang dimaksud dapat berupa pembagian tugas yang adil. Masing-masing anggota kelompok juga harus berkontribusi aktif selama pengerjaan tugas. Manajemen waktu juga merupakan aspek yang sangat penting dalam pengerjaan suatu tugas mandiri maupun berkelompok. Setiap anggota kelompok harus dapat membagi waktu dengan baik dan mempunyai kedisiplinan. Jika anggota kelompok sedang terkendala waktu dikarenakan kesibukan lainnya, anggota tersebut wajib mengabari kelompoknya agar kelompok dapat saling membantu.

C. Refleksi

Melalui tugas ini, kami mempelajari materi BFS dan DFS lebih dalam dari apa yang telah diajarkan di kelas. Selain itu, kami juga kembali melatih diri kami dalam membangun sebuah aplikasi berbasis web yang responsif dan ramah untuk pengguna.

LAMPIRAN

A. Github

https://github.com/reletz/Tubes2_NamaKelompok

B. Video

https://youtu.be/DIwRzelwOIE

C. Tabel Pemeriksaan

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

REFERENSI

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/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf