

IF2211 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Laporan Tugas Besar 2

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2023/2024



Oleh

Shabrina Maharani	13522134
Rayhan Ridhar Rahman	13522160
Valentino Chryslie Triadi	13522134

Kelompok GasAja

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

DAFTAR ISI	2
BAB 1 DESKRIPSI TUGAS	3
BAB 2 LANDASAN TEORI	4
2.1 Algoritma Breadth First Search	4
2.2 Algoritma Iterative Deepening Search	5
2.3 Graf	5
2.3.1 Terminologi Graf	7
2.3.2 Traversal Graf	9
2.4 Pembangunan Aplikasi Web	10
BAB 3 ANALISIS PEMECAHAN MASALAH	12
3.1 Langkah-Langkah Pemecahan Masalah	12
3.2 Mapping Persoalan menjadi Elemen Algoritma IDS dan BFS	13
3.2.1 BFS	13
3.2.2 IDS	14
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web GasAja WikiRace Solver	16
3.4 Contoh Ilustrasi Kasus	17
BAB 4 IMPLEMENTASI DAN PENGUJIAN	21
4.1 Spesifikasi Teknis Program	21
4.2 Source Code	33
4.3 Penjelasan Tata Cara Penggunaan Program	48
4.4 Hasil Pengujian	49
4.4.1 Hasil Pengujian Breadth First Search	49
4.4.2 Hasil Pengujian Iterative Deepening Search	51
4.5 Analisis Hasil Pengujian	52
BAB 5 KESIMPULAN DAN SARAN	53
5.1 Kesimpulan	53
5.2 Saran	53
5.3 Refleksi	53
DAFTAR PUSTAKA	55

BAB 1 DESKRIPSI TUGAS

WikiRace atau Wiki Game merupakan sebuah permainan yang melibatkan Wikipedia, sebuah ensiklopedia daring gratis yang dikelola oleh berbagai relawan di dunia. Alur permainannya dimulai ketika pemain dilandaskan pada suatu artikel Wikipedia. Kemudian pemain tersebut memiliki tujuan untuk mencapai artikel tertentu lainnya dengan cara menelusuri artikel-artikel lain yang tertera secara internal pada Wikipedia dalam waktu paling singkat atau jumlah klik (artikel) paling sedikit.



Gambar 1.1 Ilustrasi Graf Contoh WikiRace

(Sumber: https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZsIicJCWQ.png)

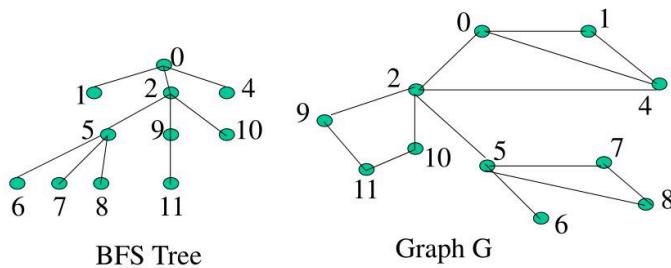
Tugas Besar 2 Strategi Algoritma kali ini adalah membuat sebuah WikiRace Solver berbasis web dengan mengimplementasikan algoritma Breadth First Search dan Iterative Deepening Search. Program ini menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan kemudian program akan memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms). Program diharapkan dapat memproses rute dalam waktu 5 menit untuk setiap permainan.

BAB 2 LANDASAN TEORI

2.1 Algoritma Breadth First Search

Algoritma *breadth first search* adalah salah satu algoritma yang dapat digunakan untuk melakukan traversal struktur data graf sehingga dapat mencari solusi dari suatu persoalan. Algoritma ini dapat dicirikan dengan mengambil semua jalur yang dapat ditempuh pada kedalaman tertentu, kemudian menelusuri bagian graf yang lebih dalam.

Illustration of BFS



Gambar 2.1.1 Ilustrasi Pencarian BFS dengan Konsep Graf dalam Representasi Pohon

(Sumber : <https://image.slideserve.com/146378/illustration-of-bfs-1.jpg>)

Breadth first search dimulai dengan menentukan simpul awal, kemudian melakukan pencarian terhadap simpul-simpul yang bertetangga dengan simpul tersebut. Dilanjutkan oleh pencarian simpul lainnya melalui simpul-simpul yang terhubung tersebut. Dalam beberapa kasus, BFS dapat menelusuri ulang suatu simpul tetapi dalam kasus seperti mencari rute terpendek, simpul yang telah ditelusuri tidak perlu diperiksa lagi. Algoritma ini akan menggunakan memori tambahan juga untuk menyimpan simpul-simpul yang telah ditelusuri tetapi belum dijelajahi. Hasil yang didapat akan menjadi solusi paling optimal tetapi memakan lebih banyak memori.

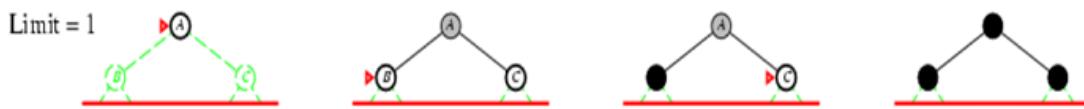
Breadth First Search (BFS) dan Depth First Search (DFS) tidak hanya efektif dalam menelusuri struktur data graf tetapi juga sangat bermanfaat dalam pencarian solusi untuk masalah yang kompleks melalui pembentukan pohon dinamis. Dalam konteks ini, kedua metode tersebut memungkinkan eksplorasi dari ruang status melalui pohon dinamis, di mana setiap simpul di dalam pohon mewakili keadaan masalah yang potensial. Pohon ini diorganisir sedemikian rupa sehingga akar merepresentasikan keadaan awal, cabang-cabangnya merepresentasikan langkah-langkah atau operator yang mungkin dalam persoalan, dan daun-daunnya adalah keadaan solusi atau goal.

Dalam pencarian solusi, setiap simpul dalam pohon dinamis diperiksa untuk menentukan apakah itu merupakan solusi dari masalah. Jika sebuah simpul ditemukan sebagai solusi, pencarian bisa dihentikan jika hanya diperlukan satu solusi, atau bisa dilanjutkan untuk menemukan solusi lain jika semua solusi yang mungkin diinginkan. Ruang status dalam konteks ini adalah himpunan semua simpul yang mungkin, sedangkan ruang solusi adalah himpunan dari semua keadaan yang merupakan solusi dari masalah. Pembangkitan status baru dilakukan dengan menerapkan operator atau langkah-langkah legal pada simpul-simpul dari suatu jalur, sehingga memperluas pohon status dan mengarah pada pencapaian solusi masalah. Ini menunjukkan bagaimana BFS tidak hanya membantu dalam navigasi graf tetapi juga dalam mengembangkan dan mengeksplorasi kemungkinan solusi secara dinamis.

2.2 Algoritma Iterative Deepening Search

Algoritma *iterative deepening search* merupakan salah satu upaya untuk melakukan optimasi terhadap algoritma *breadth first search* yang memakan banyak memori dan *depth first search* yang mungkin menghasilkan langkah yang salah sehingga solusi bisa jadi sangat panjang bahkan tak hingga.

IDS dengan $d=1$



Gambar 2.2.1 Ilustrasi Pencarian IDS dengan Depth = 1

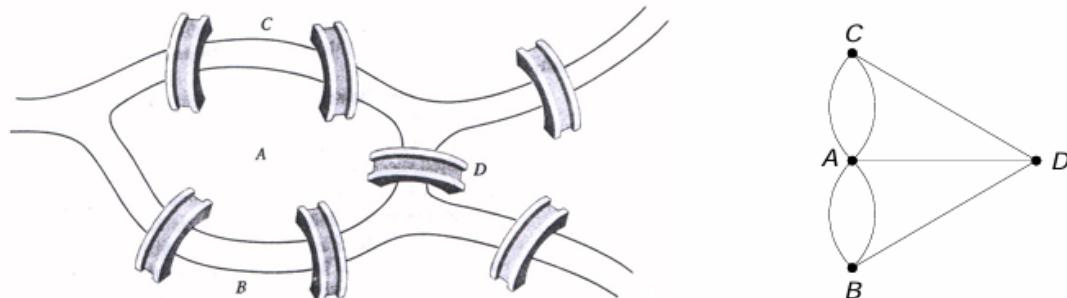
(Sumber : [BFS-DFS-2021-Bag2.pdf \(itb.ac.id\)](#))

Iterative deepening search mengimplementasikan algoritma depth-limited search yang menyerupai algoritma DFS namun solusi yang didapat memiliki batas tertentu. IDS akan melakukan iterasi dari kedalaman 0 yang akan terus bertambah. Setiap kali kedalaman bertambah, akan dilakukan DLS terhadap simpul tersebut sampai kedalaman saat itu, kemudian kedalaman bertambah untuk iterasi selanjutnya jika tidak ditemukan solusi. Dengan algoritma ini, solusi yang didapatkan memiliki kompleksitas ruang yang lebih kecil walau kecepatan yang lebih rendah dibanding BFS.

2.3 Graf

Graf adalah kumpulan titik yang mungkin terhubung maupun tidak terhubung dengan titik lainnya dengan garis. Graf biasanya digunakan untuk menunjukkan objek-objek diskrit dan hubungan-hubungan dari objek-objek tersebut. Dalam graf

terdapat simpul yang merupakan representasi suatu entitas dan juga sisi yang merepresentasi hubungannya.

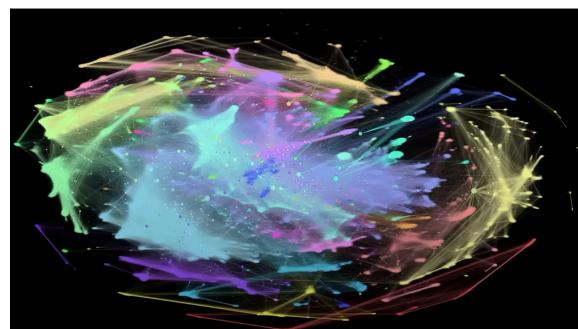


Gambar 2.3.1 Masalah Jembatan Königsberg dan Ilustrasinya dalam Graf
(Sumber : [19-Graf-Bagian1-2023.pdf](#) (itb.ac.id))

Teori graf ini pertama kali digagas oleh Swiss Leonhard Euler pada abad ke-18, untuk memecahkan “Masalah Tujuh Jembatan Königsberg.” Dengan teori ini, dapat dibuktikan bahwa permasalahan tersebut tidak dapat dilakukan. Kemudian untuk beberapa dekade ke depan, teori graf berkembang pesat berkat kontribusi dari banyak matematikawan seperti Gustav Kirchhoff, William Hamilton, dan Arthur Cayley.

Graf dapat dituliskan dalam bentuk $G = (V, E)$ yang dalam hal ini V adalah himpunan tidak-kosong dari simpul-simpul (vertices), $V = \{ v_1, v_2, \dots, v_n \}$ serta E adalah himpunan sisi (edges) yang menghubungkan sepasang simpul, $E = \{ e_1, e_2, \dots, e_n \}$.

Berdasarkan ada tidaknya gelang atau sisi ganda pada graf digolongkan menjadi dua jenis yaitu graf sederhana dan graf tak-sederhana. Kemudian graf tak-sederhana dapat digolongkan lagi menjadi graf ganda dan graf semu. Lalu berdasarkan orientasi arah traversal, graf dapat dibagi menjadi graf tak-berarah dan graf berarah.



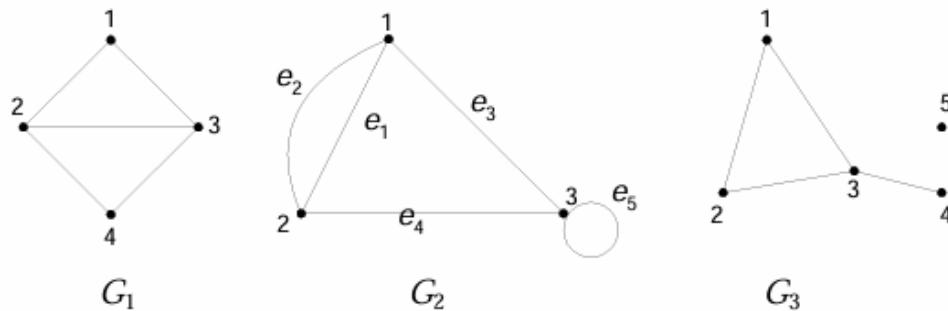
Gambar 2.3.2 Ilustrasi Graf Wikipedia
(Sumber: <https://www.youtube.com/watch?v=JheGL6uSF-4>)

Terdapat beberapa aplikasi dalam teori graf yang dapat digunakan untuk melakukan permainan Wikipedia. Ketetanggaan memiliki arti dua objek yang terhubung

secara langsung, bisa dihubungkan dengan setiap artikel memiliki *link* internal yang merujuk pada artikel lain. Sedangkan derajat dapat merujuk pada jarak dari suatu artikel ke artikel lainnya. Selain itu ada juga artikel yang tidak memiliki sisi yang mengarah kepadanya dan ada juga artikel yang tidak memiliki link internal lainnya, bahkan ada yang memenuhi kedua kondisi tersebut.

2.3.1 Terminologi Graf

1. Ketetanggaan (*Adjacent*)

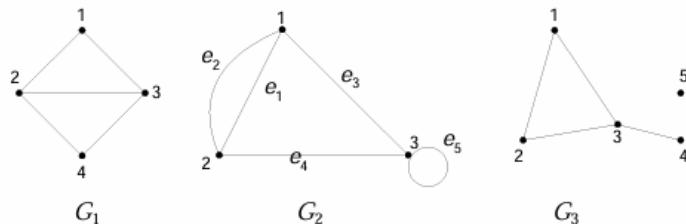


Gambar 2.3.3 Konsep Ketetanggan
(Sumber : [19-Graf-Bagian1-2023.pdf \(itb.ac.id\)](#))

Ketetanggaan menggambarkan hubungan antara dua verteks dalam graf. Dua verteks dikatakan bertetangga atau bersebelahan jika terdapat sisi yang menghubungkan keduanya secara langsung. Dalam konteks graf, memahami ketetanggaan penting untuk analisis hubungan antar elemen dalam graf. Pada Graf G1 di atas, simpul 1 bertetangga dengan simpul 2 dan 3.

2. Bersisian (*Incidency*)

Tinjau graf G_1 : sisi $(2, 3)$ bersisian dengan simpul 2 dan simpul 3, sisi $(2, 4)$ bersisian dengan simpul 2 dan simpul 4, tetapi sisi $(1, 2)$ tidak bersisian dengan simpul 4.

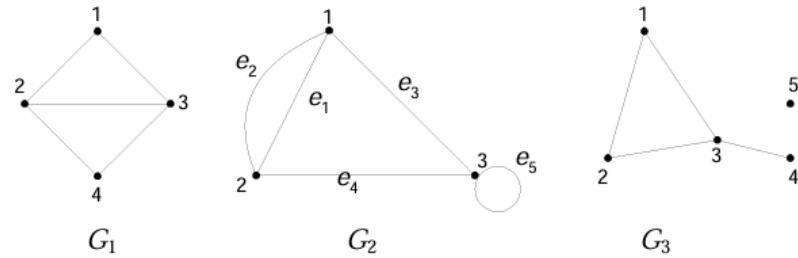


Gambar 2.3.4 Konsep Bersisian
(Sumber : [19-Graf-Bagian1-2023.pdf \(itb.ac.id\)](#))

Incidence adalah istilah yang digunakan untuk menggambarkan hubungan antara verteks dan sisi dalam graf. Sebuah sisi dikatakan insiden atau bersisian dengan verteks jika sisi tersebut berakhir atau dimulai dari verteks tersebut. Dalam graf berarah, arah sisi terhadap verteks juga penting dalam menentukan *incidence*.

3. Simpul Terpencil (*Isolated Vertex*)

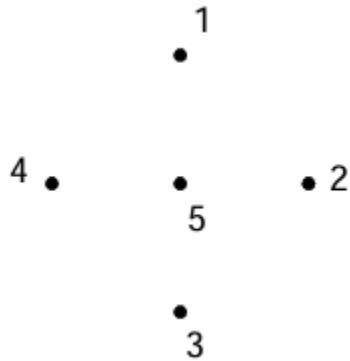
Tinjau graf G_3 : simpul 5 adalah simpul terpencil.



Gambar 2.3.5 Konsep Bersisian
(Sumber : [19-Graf-Bagian1-2023.pdf \(itb.ac.id\)](#))

Simpul terpencil adalah verteks yang tidak memiliki sambungan atau sisi yang insiden dengannya. Dalam kata lain, simpul ini tidak terhubung dengan simpul lain dalam graf. Simpul terpencil sering menunjukkan elemen yang tidak terlibat dalam interaksi dalam konteks model yang direpresentasikan oleh graf.

4. Graf Kosong (*null graph* atau *empty graph*)



Gambar 2.3.5 Konsep Graf Kosong
(Sumber : [19-Graf-Bagian1-2023.pdf \(itb.ac.id\)](#))

Graf kosong adalah graf yang tidak memiliki sisi sama sekali. Graf ini dapat memiliki satu atau lebih verteks, tetapi tidak ada sisi yang menghubungkan verteks-verteks tersebut. Graf kosong sering digunakan sebagai kasus dasar dalam banyak teori dan analisis graf.

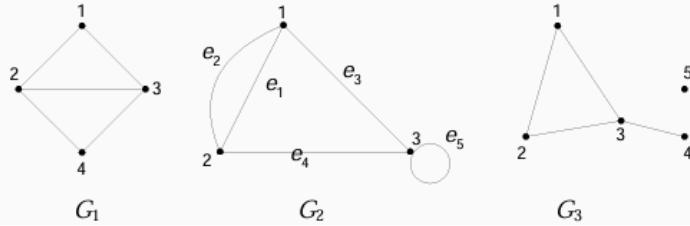
5. Derajat (*degree*)

Notasi: $d(v)$

Tinjau graf G_1 : $d(1) = d(4) = 2$
 $d(2) = d(3) = 3$

Tinjau graf G_3 : $d(5) = 0 \rightarrow$ simpul terpencil
 $d(4) = 1 \rightarrow$ simpul anting-anting (*pendant vertex*)

Tinjau graf G_2 : $d(1) = 3 \rightarrow$ bersisian dengan sisi ganda
 $d(3) = 4 \rightarrow$ bersisian dengan sisi gelang (*loop*)



Gambar 2.3.5 Konsep Derajat pada Graf
(Sumber : [19-Graf-Bagian1-2023.pdf \(itb.ac.id\)](#))

Derajat sebuah verteks didefinisikan sebagai jumlah sisi yang insiden dengan verteks tersebut. Dalam graf berarah, derajat ini dapat dibagi menjadi dua, yaitu derajat masuk (jumlah sisi yang masuk ke verteks) dan derajat keluar (jumlah sisi yang keluar dari verteks). Mengetahui derajat verteks penting untuk memahami seberapa terhubung verteks tersebut dalam graf.

2.3.2 Traversal Graf

Traversal graf adalah proses melakukan pencarian solusi yang direpresentasikan dengan konsep graf. Algoritma traversal graf merupakan algoritma pengunjungan simpul-simpul di dalam graf dengan cara yang sistematis. Algoritma traversal graf terbagi menjadi pencarian melebar (*breadth first search/BFS*) dan pencarian mendalam (*depth first search/DFS*) dengan asumsi bahwa graf terhubung semua simpulnya.

Dalam algoritma pencarian solusi berbasis graf, terdapat dua jenis pendekatan yaitu pencarian tanpa informasi dan pencarian dengan informasi. Pencarian tanpa informasi, atau sering juga disebut pencarian buta, tidak menyertakan informasi tambahan selama proses pencarian. Metode yang termasuk dalam kategori ini antara lain Depth-First Search (DFS), Breadth-First Search (BFS), Depth Limited Search, Iterative Deepening Search, dan Uniform Cost Search. Sementara itu, pencarian dengan informasi menggunakan heuristik untuk mengidentifikasi keadaan yang tidak merupakan tujuan namun "lebih

"menjanjikan" daripada keadaan lainnya, contoh metode ini adalah Best First Search dan A*.

Dalam konteks representasi graf dalam proses pencarian solusi, ada dua pendekatan yaitu graf statis dan graf dinamis. Graf statis adalah graf yang sudah ada dan terbentuk sebelum proses pencarian dimulai dan direpresentasikan sebagai struktur data. Di sisi lain, graf dinamis dibangun dan berkembang selama proses pencarian berlangsung, dimana struktur graf tidak tersedia sebelum pencarian dimulai dan hanya terbentuk selama pencarian solusi dilakukan.

2.4 Pembangunan Aplikasi Web



Gambar 2.3.5 Dokumentasi GasAja WikiRace Solver

(Sumber : Dokumentasi Pribadi)

Aplikasi web "GasAja WikiRace Solver" dirancang untuk membantu menyelesaikan tantangan WikiRace, sebuah permainan yang mengharuskan pemain berpindah dari satu artikel Wikipedia ke artikel lain hanya melalui link yang ada. Aplikasi ini menggunakan dua metode pencarian yaitu Breadth First Search (BFS) dan Iterative Deepening Search (IDS). Pengguna dapat memilih metode yang diinginkan dan memasukkan judul artikel awal serta tujuan. Setelah itu, sistem akan melakukan scraping data dari Wikipedia, memprosesnya, dan menampilkan hasil berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute yang ditempuh, serta durasi pencarian.

Aplikasi web ini dibangun dengan menggunakan Node.js untuk antarmuka pengguna (*frontend*) dan Golang untuk bagian *backend*, aplikasi ini menawarkan kinerja yang efisien dan antarmuka yang cukup responsif. Golang khususnya dikenal akan kemampuannya dalam mengelola memori dan proses secara concurrent, sehingga dapat menangani beban kerja yang tinggi tanpa mengurangi kecepatan respons aplikasi. Proses scraping yang dinamis memastikan bahwa data yang diperoleh selalu terbaru,

memungkinkan aplikasi menawarkan rute yang optimal, terpendek, dan akurat kepada pengguna.

BAB 3 ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan utama yang harus diselesaikan dalam Tugas Besar 2 ini adalah bagaimana caranya aplikasi web yang dibangun dapat menemukan jalur/rute terpendek dari artikel Wikipedia awal menuju artikel Wikipedia akhir (tujuan) yang diinginkan oleh pengguna dan menampilkannya pada antarmuka aplikasi web. Berikut adalah langkah-langkah yang digunakan untuk menyelesaikan permasalahan tersebut.

1. Setelah user memasukkan judul artikel awal, judul awal tersebut akan diubah bentuknya dari sebuah string menjadi struct web yang berisi URL dan Title dengan cara melakukan replace terhadap judul awal tersebut. Judul artikel awal ini akan menjadi sebuah *start node (root)* yang akan memicu pembentukan graf dinamis (bisa juga direpresentasikan sebagai sebuah pohon dinamis). Sedangkan, judul artikel akhir (tujuan) yang akan menjadi *end node* atau dalam arti lain judul artikel akhir inilah yang akan dicari keberadaannya untuk didapatkan tujuannya.
2. Kemudian, user juga akan memasukkan algoritma yang dipilih untuk menyelesaikan pencarian ini. Pilihan tersebut akan dikirimkan ke bagian penyelesaian solusi (*backend*) bersamaan dengan pengiriman judul artikel dan pilihan tersebut akan menjadi titik awal dari program untuk menyelesaikan solusi dengan algoritma yang dipilih oleh user. Misalnya user memasukkan sebuah judul artikel awal yaitu “A” dan judul artikel tujuannya yaitu “B”. Pencarian solusi untuk permasalahan tersebut sesuai algoritma yang dipilih akan dijelaskan lebih lanjut pada bagian mapping persoalan. Utamanya, pada tahap ini, program menggunakan pustaka gocolly untuk melakukan scraping terhadap web yang dikunjungi dikarenakan dengan penggunaan pustaka tersebut program dapat mela



path?



Gambar 3.1.1 Ilustrasi awal pencarian jalur terpendek
Sumber : Dokumentasi Pribadi

- Setelah jalur terpendek berhasil ditemukan, data jalur tersebut akan dikirim ke frontend. Untuk Iterative Deepening Search (IDS), jalur dikirim dalam bentuk list sederhana, sementara untuk Breadth First Search (BFS), data dikirim dalam bentuk list of lists. Data ini kemudian diolah dan ditampilkan pada antarmuka web. Tampilan ini memungkinkan pengguna untuk secara visual melacak dan memahami jalur yang telah ditempuh dari artikel Wikipedia awal hingga artikel tujuan. Dengan cara ini, pengguna dapat melihat setiap langkah dari proses pencarian, termasuk setiap artikel yang dilintasi, yang memberikan wawasan mendalam tentang bagaimana solusi dihasilkan oleh algoritma yang dipilih.

3.2 Mapping Persoalan menjadi Elemen Algoritma IDS dan BFS

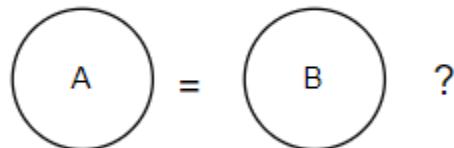
Mapping elemen-elemen dari algoritma yang akan dibuat berdasarkan pemecahan masalah dalam Tugas Besar ini adalah :

- Artikel Awal : Root atau Start Node dari sebuah pohon dinamis yang akan dibangun
- Artikel Akhir : Node yang dicari keberadaannya untuk nantinya disimpan jalur terpendek yang ditemukan
- Child dari node : Node dari sebuah tree yang akan dilalui oleh proses IDS maupun BFS untuk dilakukan pengecekan

Dari informasi tersebut program dapat menjalankan prosesnya sesuai dengan konsep algoritma yang diinginkan user. Untuk mempermudah penjelasan, diasumsikan artikel awal nya adalah web A yang akan menjadi node A dan artikel akhir atau artikel tujuannya adalah web B yang akan menjadi node B.

3.2.1 BFS

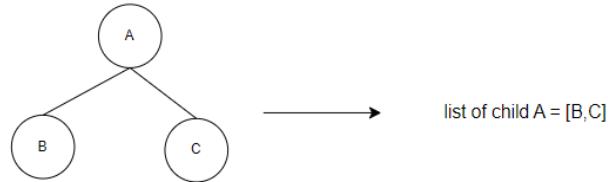
- Pengecekan akan dilakukan terlebih dahulu terhadap root atau *start node* itu sendiri. Jika root atau start node adalah web yang kita cari, maka proses BFS akan langsung dihentikan.



Gambar 3.1.2 Pengecekan
Sumber : Dokumentasi Pribadi

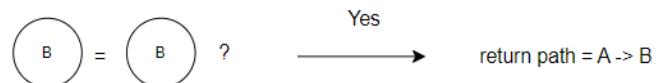
- Oleh karena kita ingin mencari web dengan judul B, sedangkan root nya adalah web dengan judul A, maka proses BFS akan dilanjutkan dengan cara men-generate child dari web A yaitu dengan cara melakukan scraping terhadap

web A menggunakan pustaka go colly yang akan membuat proses scraping menjadi lebih cepat. Setelah scraping dilakukan pada web A, program akan menyimpan data child dari A dalam bentuk list. Kemudian, akan dilakukan pengecekan terhadap child dari A. Pengecekan dilakukan dengan cara mencocokkan judul dari masing masing child web A dengan judul dari web B. Pengecekan child dilakukan dari child yang berada pada indeks awal dalam list of child.



Gambar 3.1.3 Penyimpanan
Sumber : Dokumentasi Pribadi

3. Terdapat dua perlakuan sesuai kondisi dari child A. Perlakuan pertama, jika ternyata dari *list of child* A ditemukan kecocokan terhadap web B, maka proses BFS akan dihentikan dan program akan mengirimkan jalur terpendek dari web A ke web B. Jika dari *list of child* A, tidak ditemukan kecocokan tersebut, maka proses akan berlanjut dengan menyimpan terlebih dahulu *list of child* A ke sebuah penyimpanan lalu mengulangi langkah dari langkah ke-2 sampai ke-3 secara rekursif sampai ditemukannya web B.



Gambar 3.1.4 Pengecekan Child
Sumber : Dokumentasi Pribadi

3.2.2 IDS

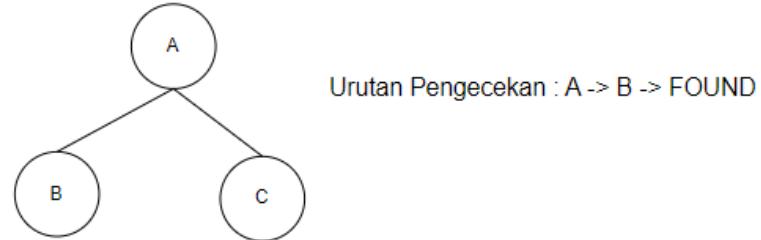
1. Langkah pertama IDS sama dengan langkah pertama pada BFS yaitu pengecekan akan dilakukan terlebih dahulu terhadap root atau *start node* itu sendiri. Dalam Algoritma IDS, pengecekan ini dianggap sebagai iterasi = 0 atau pengecekan pada depth = 0.



Gambar 3.1.5 Pengecekan d=0 IDS

Sumber : Dokumentasi Pribadi

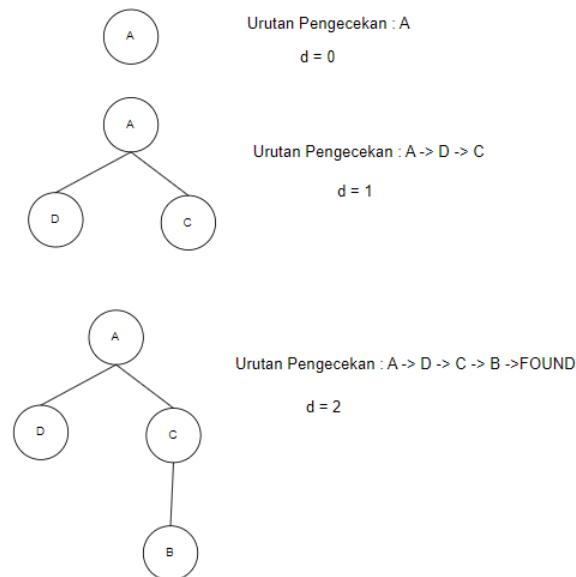
2. Jika pada depth 0, solusi belum ditemukan maka, program akan menerapkan konsep DLS dengan depth bertambah 1. Konsep DLS merupakan penerapan algoritma DFS dengan maksimal kedalaman tertentu. Setelah itu, pengecekan akan diulang kembali dari root nya seperti gambar di bawah ini.



Gambar 3.1.6 Pengecekan d=1 IDS

Sumber : Dokumentasi Pribadi

3. Namun, jika ternyata tidak ditemukan solusi pada child dari A, maka proses akan mengulangi penerapan DLS dengan maksimal kedalaman bertambah 1 (kalau tadinya kedalaman = 1 maka di tahap ini kedalaman menjadi 2). Proses akan melakukan pengecekan kembali dari root sampai node yang ada pada kedalaman saat ini.



Gambar 3.1.7 Pengecekan d=2 IDS

Sumber : Dokumentasi Pribadi

3.3 Fitur Fungsional dan Arsitektur Aplikasi Web GasAja WikiRace Solver

Aplikasi web "GasAja WikiRace Solver" dirancang untuk menemukan rute terpendek dari satu laman Wikipedia ke laman Wikipedia lain dengan memanfaatkan Go Colly, sebuah pustaka yang memungkinkan scraping secara paralel. Keunggulan Go Colly terletak pada kemampuannya mengelola banyak halaman secara simultan, yang meningkatkan efisiensi dan kecepatan proses pencarian, faktor kritis dalam pencarian rute terpendek. Proses scraping dan pencocokan menggunakan algoritma Iterative Deepening Search (IDS) dan Breadth First Search (BFS) mengoptimalkan pencarian sehingga hasilnya tidak hanya cepat tetapi juga akurat. Aplikasi web ini juga dibangun dengan mengintegrasikan frontend yang menampilkan antarmuka kepada user serta backend yang melakukan pencarian solusi.

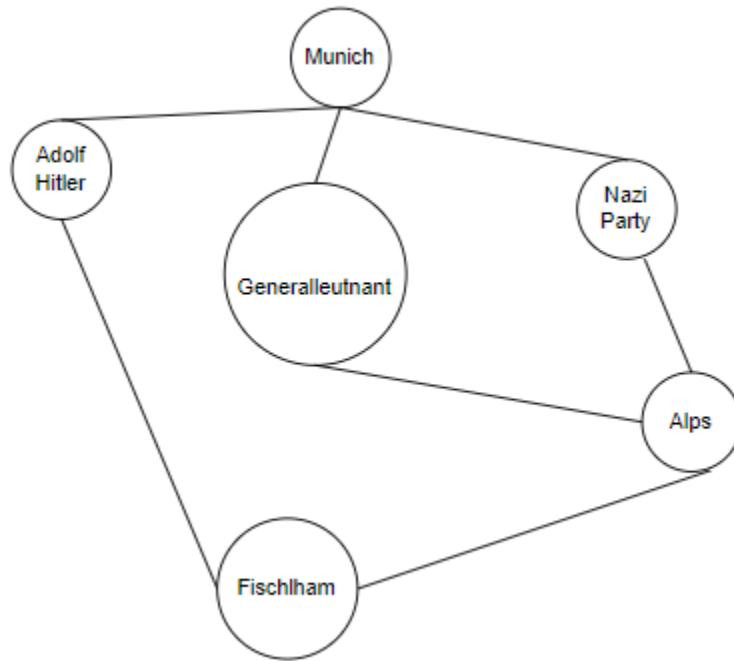
Pada sisi frontend, aplikasi ini menggunakan Node.js untuk menyediakan antarmuka pengguna yang responsif dan interaktif. Fitur-fitur seperti light mode dan dark mode memungkinkan pengguna untuk menyesuaikan tampilan sesuai dengan preferensi user, meningkatkan kenyamanan visual selama penggunaan aplikasi. Selain itu, aplikasi juga dilengkapi dengan animasi yang halus untuk memperkaya pengalaman pengguna. Halaman 'About Us' juga tersedia untuk memberikan informasi tentang tujuan dari aplikasi web.

Backend aplikasi dibangun menggunakan Golang, yang mendukung kinerja tinggi khususnya dalam pengelolaan konkurensi yang dibutuhkan oleh algoritma BFS dan IDS. Kode backend yang efisien mengatur rute API, mengelola permintaan HTTP, dan menyampaikan hasil pencarian ke frontend dalam format JSON. Komunikasi antara client dan server dirancang untuk lancar, mendukung pertukaran data yang cepat dan handal.

Aplikasi web ini juga menggunakan mekanisme caching dari Go Colly yang menyimpan hasil scraping sementara. Hal ini sangat mengurangi waktu akses kembali ke halaman yang sama, mempercepat proses pencarian secara signifikan. Seluruh sistem telah dirancang untuk beroperasi secara efisien di bawah beban tinggi, dengan kemampuan untuk membatasi jumlah thread yang berjalan secara simultan dan mengimplementasikan kontrol akses asal lintas sumber (CORS) untuk menjaga keamanan dan kompatibilitas dalam aplikasi web.

3.4 Contoh Ilustrasi Kasus

Untuk memperjelas analisis pemecahan masalah, kami akan membuat sebuah ilustrasi kasus dengan penyelesaiannya. Misal pada kasus dimana sebuah artikel wikipedia berjudul “Munich” memiliki bentuk graph sebagai berikut (ilustrasi graf di bawah ini tidak sama dengan aslinya (hanya contoh ilustrasi) :



Gambar 3.4.1 Graf Ilustrasi Kasus
Sumber : Dokumentasi Pribadi

Kemudian, user memasukkan judul artikel awal yaitu Munich dan judul artikel akhir yaitu Fischlham.

3.4.1 BFS

Program tersebut adalah sebuah web scraper yang menggunakan algoritma BFS (Breadth-First Search) untuk melakukan crawling pada situs Wikipedia dalam bahasa yang ditentukan. Ketika pengguna mengirim permintaan dengan judul awal (startTitle) "Munich" dan kata kunci (keyword) "Fischlham", fungsi `bfsScrapeHandler` akan dipanggil.

Pertama-tama, permintaan diproses, dan input dari permintaan HTTP di-decode menjadi struktur data 'Input'. Selanjutnya, URL dasar (BASEURL) ditentukan berdasarkan bahasa yang diberikan oleh pengguna. Proses scraping dimulai dengan memanggil fungsi `bfsScrape2` dengan parameter titik awal, yaitu halaman Wikipedia tentang Munich, dan kata kunci Fischlham. Selama proses

scraping, total jumlah entitas web yang telah di-scrape dan hasil scraping disimpan.

Setelah proses scraping selesai, waktu yang dibutuhkan dan hasil scraping dikodekan ke dalam sebuah struktur data, kemudian dikirim sebagai respons HTTP.

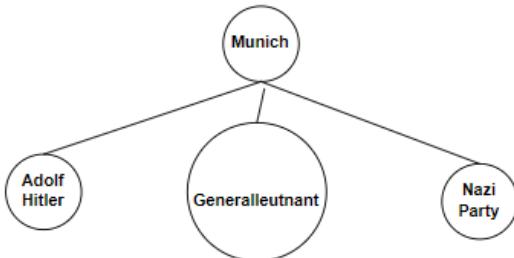
Dalam fungsi `bfsScrape2`, proses scraping dilakukan dengan menggunakan algoritma BFS. Halaman Wikipedia awal tentang Munich dijadikan titik awal, dan dilakukan pencarian melalui halaman-halaman terhubung untuk menemukan halaman yang berisi kata kunci "Fischlham". Proses scraping dilakukan secara paralel menggunakan goroutine untuk meningkatkan efisiensi, dengan batasan waktu maksimal 5 menit. Jika kata kunci ditemukan atau waktu maksimal tercapai, proses scraping dihentikan dan hasilnya dikirim sebagai respons. Berikut adalah penyelesaian dari ilustrasi kasus berikut oleh program GasAja WikiRace Solver jika diilustrasikan dalam proses pembangkitan pohon dinamis.

1. Iterasi 1



Urutan Pengecekan : Munich

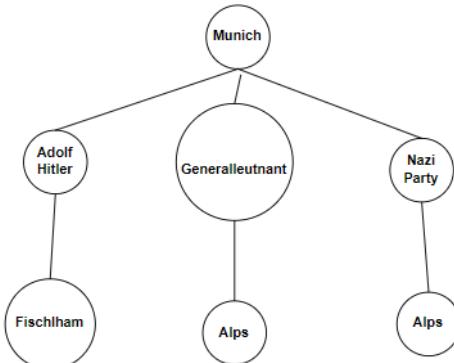
2. Iterasi 2



Urutan Pengecekan :
1. Adolf Hitler
2. Generalleutnant
3. Nazi Party

Penyimpanan : [Munich, Adolf Hitler, Generalleutnant, Nazi Party]

3. Iterasi 3



Urutan Pengecekan :
1. Fischlham
Found
path = Munich -> Adolf Hitler -> Fischlham
Penyimpanan : [Munich, Adolf Hitler, Generalleutnant, Nazi Party, Fischlham]

3.4.2 IDS

Program akan membentuk URL dasar Wikipedia dan membuat sebuah struktur pohon ('Tree') dengan akar ('Root') berupa halaman Wikipedia awal. Seluruh halaman yang akan dijelajahi akan disimpan dalam bentuk pohon, di mana setiap node dalam pohon mewakili sebuah halaman Wikipedia.

Kemudian, program memulai proses pencarian dengan memanggil fungsi 'findSolution'. Fungsi ini bertujuan untuk mengeksplorasi halaman-halaman Wikipedia terkait dengan halaman awal yang telah ditentukan, hingga ditemukan halaman yang mengandung kata kunci yang dicari oleh pengguna. Selama proses pencarian, setiap halaman yang dikunjungi akan disimpan dalam variabel 'allWebs', yang berisi semua halaman yang telah dijelajahi.

Selanjutnya, pada setiap langkah eksplorasi, program memanggil fungsi 'expandTree' untuk mengeksplorasi halaman-halaman terkait dengan halaman saat ini. Fungsi ini menggunakan library Colly untuk melakukan scraping halaman-halaman Wikipedia terkait, dengan batasan kedalaman yang ditentukan oleh pengguna. Selama proses eksplorasi, program juga melakukan pengecekan apakah halaman yang telah dikunjungi telah tercatat sebelumnya dalam variabel 'allWebs'. Jika belum, halaman tersebut akan ditambahkan ke 'allWebs'.

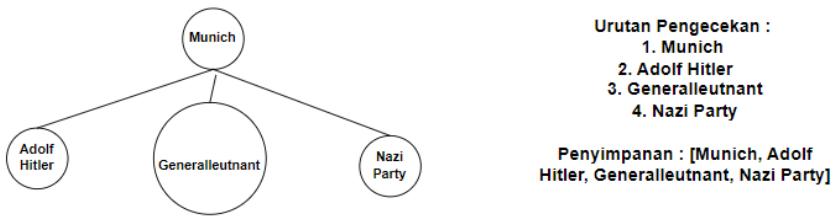
Setelah ditemukan halaman yang mengandung kata kunci yang dicari, program akan menghentikan pencarian dan menyusun ulang jalur dari halaman awal hingga halaman tujuan yang mengandung kata kunci. Jalur yang telah ditemukan akan disimpan dalam variabel 'webs'. Berikut adalah penyelesaian dari ilustrasi kasus berikut oleh program GasAja WikiRace Solver jika diilustrasikan dalam proses pembangkitan pohon dinamis.

1. Iterasi 0 (depth = 0)

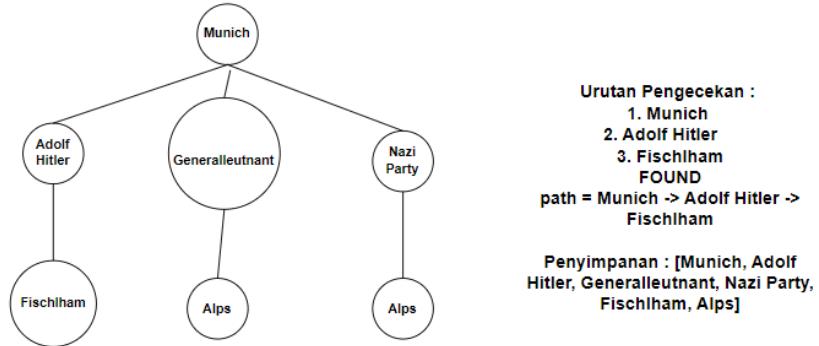


Urutan Pengecekan : Munich

2. Iterasi 1



3. Iterasi 2



BAB 4 IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

1. Struktur data

Terdapat beberapa struktur data yang digunakan dalam program ini. Struktur data tersebut memiliki peran penting dalam proses pencarian solusi dengan menggunakan algoritma BFS (Breadth First Search) dan IDS (Iterative Deepening Search) pada aplikasi web "GasAja WikiRace Solver".

Pertama, terdapat dua jenis struktur data yang digunakan, yaitu 'web' dan 'Input'. Struktur data 'web' digunakan untuk merepresentasikan informasi mengenai sebuah halaman Wikipedia, yang terdiri dari URL dan judul artikel. Sedangkan, struktur data 'Input' digunakan untuk menampung input dari pengguna, seperti kata kunci pencarian, judul artikel awal, dan bahasa Wikipedia yang akan digunakan.

```
type web struct {
    Url    string `json:"url"`
    Title  string `json:"title"`
}

type Input struct {
    Keyword string `json:"keyword"`
    Start   string `json:"start"`
    Lang    string `json:"lang"`
}
```

Selanjutnya, untuk pencarian solusi menggunakan algoritma IDS, terdapat struktur data 'Node' dan 'Tree'. Struktur data 'Node' merepresentasikan sebuah node dalam pohon pencarian, yang berisi informasi mengenai halaman Wikipedia dan kedalaman dari node tersebut. Sedangkan, struktur data 'Tree' merepresentasikan keseluruhan pohon pencarian, yang memiliki sebuah node akar sebagai titik awal pencarian.

```
type Node struct {
    Value    web
    Depth    int
    Children []*Node
}

type Tree struct {
    Root Node
}
```

Implementasi pencarian menggunakan algoritma BFS menggunakan struktur data 'ResultEntity', yang menyimpan informasi tentang hasil pencarian, seperti indeks parent dari node, dan informasi mengenai halaman Wikipedia yang ditemukan.

```
type ResultEntity struct {
    index      int      // index of parent
    webEntity  web
}
```

Selain itu, dalam kode scraper.go dan BFScolly.go, terdapat fungsi-fungsi yang menggunakan struktur data tersebut untuk melakukan scraping halaman Wikipedia dan pencarian solusi dengan algoritma BFS dan IDS. Proses scraping dilakukan secara paralel menggunakan pustaka Go Colly, yang memungkinkan untuk mengumpulkan data dari beberapa halaman secara efisien.

```
type TokenBucket struct {
    tokens int
    mutex  sync.Mutex
}
```

Terakhir, terdapat struktur data Token Bucket. Ini digunakan untuk mengimplementasikan algoritma pembatasan laju (rate limiting). Token Bucket mengontrol laju permintaan atau akses ke suatu sumber daya dengan memberikan token ke setiap permintaan.

2. Fungsi dan Prosedur

Fungsi dan prosedur yang dijelaskan dalam tabel ini hanya fungsi dan prosedur utama dari proses pencarian solusi sehingga fungsi maupun prosedur seperti frontend dan beberapa fungsi integrasi antara frontend dan backend tidak ada dalam tabel berikut.

Fungsi / Prosedur	Tujuan
scraper.go	
<pre>func bfsScrapeHandler(w http.ResponseWriter, r *http.Request)</pre>	<p>Fungsi `idsScrapeHandler` bertujuan untuk menangani permintaan pencarian solusi menggunakan algoritma Iterative Deepening Search (IDS) pada aplikasi web "GasAja WikiRace Solver". Saat menerima permintaan pencarian dari pengguna melalui antarmuka web, fungsi ini akan mengurai input yang diterima untuk mendapatkan kata kunci pencarian, judul artikel awal, dan bahasa Wikipedia yang ditentukan. Selanjutnya, fungsi ini akan membentuk sebuah pohon pencarian dengan menggunakan judul artikel awal sebagai node akar, dan mulai mencari solusi dengan algoritma IDS. Proses pencarian ini dilakukan dengan memeriksa halaman-halaman Wikipedia secara iteratif dengan kedalaman yang bertambah pada setiap iterasi. Setelah pencarian selesai, hasilnya akan dikemas ke dalam format JSON dan dikirimkan kembali ke antarmuka web untuk ditampilkan kepada</p>

	<p>pengguna, termasuk informasi mengenai jalur pencarian, waktu yang dibutuhkan, dan total jumlah halaman Wikipedia yang diperiksa.</p>
	<pre>func idsScrapeHandler(w http.ResponseWriter, r *http.Request)</pre> <p>Fungsi bfsScrapeHandler bertujuan untuk menangani permintaan pencarian solusi menggunakan algoritma Breadth First Search (BFS) pada aplikasi web "GasAja WikiRace Solver". Ketika menerima permintaan pencarian dari pengguna melalui antarmuka web, fungsi ini akan mengurai input yang diterima untuk mendapatkan kata kunci pencarian, judul artikel awal, dan bahasa Wikipedia yang ditentukan. Setelah itu, fungsi ini akan memulai proses pencarian jalur terpendek menggunakan algoritma BFS dengan memeriksa halaman-halaman Wikipedia secara bertahap sesuai dengan tingkat kedalaman. Saat proses pencarian selesai, hasilnya akan disusun dalam format yang sesuai dan dikirimkan kembali ke antarmuka web untuk ditampilkan kepada pengguna, termasuk informasi mengenai jalur pencarian, waktu yang dibutuhkan,</p>

	dan total jumlah halaman Wikipedia yang diperiksa.
BFSColly.go	<pre>func gocollyScrape(w web, keyword string, BASEURL string, index int, found *bool, level int, saveRes *[][]web, storage *[][]ResultEntity, ch chan<- ResultEntity, wg *sync.WaitGroup, allWebs *[]web)</pre> <p>Fungsi `gocollyScrape` bertujuan untuk melakukan scraping halaman Wikipedia menggunakan library Colly dalam upaya mencari jalur terpendek antara dua halaman berdasarkan algoritma BFS. Fungsi ini menerima input berupa halaman web awal, kata kunci pencarian, URL dasar Wikipedia, indeks level pencarian, status penemuan jalur, serta variabel-variabel lain yang diperlukan untuk menyimpan dan memproses hasil scraping. Saat dieksekusi, fungsi ini membuat instance dari collector Colly dengan konfigurasi yang sesuai, yang kemudian digunakan untuk menelusuri dan mengumpulkan tautan-tautan dari halaman web yang telah diindeks oleh algoritma BFS. Setiap tautan yang relevan dengan Wikipedia akan disaring dan dimasukkan ke dalam antrian untuk dianalisis lebih lanjut. Jika tautan yang dijelajahi sesuai dengan kata kunci</p>

	<p>pencarian, status penemuan jalur akan diubah, dan hasil pencarian akan disimpan. Melalui fungsi ini, proses pencarian jalur terpendek diimplementasikan dengan memanfaatkan fitur scraping dari Colly secara efektif, memastikan bahwa pencarian dilakukan secara efisien dan akurat.</p>
	<pre>func gocollyScrapeBase(w web, keyword string, BASEURL string, saveRes *[][]web, allWebs *[]web)</pre> <p>Fungsi `gocollyScrapeBase` bertujuan untuk melakukan pencarian jalur terpendek antara dua halaman Wikipedia menggunakan pendekatan algoritma Breadth First Search (BFS). Fungsi ini menerima input berupa halaman awal, kata kunci pencarian, URL dasar Wikipedia, serta variabel untuk menyimpan hasil pencarian dan semua halaman Wikipedia yang telah diperiksa. Proses pencarian dimulai dengan menginisialisasi variabel lokal, kemudian memulai iterasi BFS untuk menjelajahi halaman-halaman Wikipedia secara bertahap hingga jalur terpendek ditemukan. Selama iterasi, setiap halaman yang diperiksa akan dimasukkan ke dalam antrian untuk diperiksa lebih lanjut, dan</p>

	<p>pencarian akan berlanjut hingga jalur terpendek ditemukan atau batas waktu pencarian telah tercapai. Hal ini memastikan bahwa pencarian jalur terpendek dilakukan dengan efisien dan akurat dalam kerangka waktu yang ditentukan.</p>
IDSColly.go	<pre>func expandTree(node *Node, baseURL string, limit int, count *int, allWebs *[]web)</pre> <p>Fungsi `expandTree` bertujuan untuk memperluas pohon pencarian dengan mengeksplorasi halaman-halaman terkait dalam Wikipedia dan menambahkan node-node baru ke dalam struktur pohon. Fungsi ini menerima input berupa node yang akan diperluas, URL dasar Wikipedia, batasan kedalaman pohon yang diizinkan, jumlah total langkah yang diambil, dan daftar semua halaman yang telah diakses. Selama ekspansi, fungsi menggunakan pustaka Go Colly untuk melakukan scraping halaman web dan menemukan tautan-tautan ke halaman Wikipedia terkait. Setiap tautan yang valid akan dibuat menjadi node baru dalam pohon, dan jika kedalaman node masih</p>

	<p>memenuhi batasan, fungsi akan memanggil dirinya sendiri secara rekursif untuk mengeksplorasi lebih lanjut. Dengan memperluas pohon secara dinamis, fungsi ini memungkinkan pencarian jalur terpendek dengan menggali lebih dalam ke dalam jaringan artikel Wikipedia, sehingga memperluas cakupan pencarian dan meningkatkan kesempatan menemukan solusi yang optimal.</p>
	<pre>func findSolution(node *Node, keyword string, count *int, baseURL string, webs *[]web, allWebs *[]web)</pre> <p>Fungsi `findSolution` bertujuan untuk menemukan solusi atau jalur terpendek antara dua halaman Wikipedia menggunakan algoritma pencarian yang berbasis pada ekspansi pohon dan pencarian rekursif. Fungsi ini menerima input berupa node yang merupakan bagian dari struktur data pohon yang merepresentasikan jalur pencarian, kata kunci pencarian, jumlah langkah maksimum untuk membatasi ekspansi pohon, variabel untuk menghitung jumlah total langkah yang diambil, serta variabel yang menyimpan jalur hasil dan semua halaman yang telah diakses. Selama proses pencarian, fungsi akan terus</p>

	<p>memperluas pohon dengan memperhitungkan batasan jumlah langkah maksimum dan memeriksa apakah solusi telah ditemukan setiap kali ekspansi dilakukan. Begitu solusi ditemukan, fungsi akan menghentikan proses pencarian dan mengembalikan jalur terpendek yang telah ditemukan. Dengan menggunakan kombinasi ekspansi pohon dan pencarian rekursif, fungsi ini memungkinkan penemuan jalur terpendek dengan efisien dalam struktur data yang terstruktur secara hierarkis.</p>
	<pre>func searchSolution(node *Node, keyword string, currentPath *[]web, webs *[]web) bool</pre> <p>Fungsi `searchSolution` bertujuan untuk mencari solusi atau jalur terpendek antara dua halaman Wikipedia menggunakan algoritma pencarian rekursif. Fungsi ini menerima input berupa node yang merupakan bagian dari struktur data pohon yang merepresentasikan jalur pencarian, kata kunci pencarian, serta variabel yang menyimpan jalur sementara dan jalur hasil. Saat dieksekusi, fungsi ini menambahkan node saat ini ke jalur sementara. Jika node saat ini memiliki judul yang sesuai dengan kata kunci</p>

	<p>pencarian, jalur sementara akan disalin menjadi jalur hasil dan fungsi mengembalikan nilai true, menandakan bahwa jalur telah ditemukan. Jika tidak, fungsi akan melanjutkan pencarian ke anak-anak node saat ini secara rekursif. Jika tidak ada jalur yang ditemukan dari node saat ini, fungsi akan menghapus node terakhir dari jalur sementara dan mengembalikan nilai false. Melalui fungsi ini, proses pencarian jalur terpendek diimplementasikan secara rekursif, memungkinkan penemuan jalur yang optimal dalam struktur data pohon yang terorganisir dengan baik.</p>
utils.go	
<pre>func containsWebEntity (webEntity web, Res []web) bool</pre>	<p>Fungsi containsWebEntity bertujuan untuk memeriksa apakah suatu entitas web sudah ada dalam sebuah slice dari entitas web. Dengan menggunakan perulangan, fungsi ini membandingkan URL dari entitas web yang diberikan dengan setiap entitas web dalam slice, dan jika ditemukan entitas web dengan URL yang sama, fungsi mengembalikan nilai true, menunjukkan</p>

	bahwa entitas web tersebut sudah ada dalam slice.
<pre>func isStorageContainsWebEntity(webEnti ty web, storage *[][]ResultEntity) bool</pre>	Fungsi isStorageContainsWebEntity memiliki tujuan untuk memeriksa apakah suatu entitas web sudah ada dalam struktur penyimpanan yang lebih kompleks. Fungsi ini menerima input berupa entitas web dan struktur penyimpanan yang merupakan slice dari slice dari entitas ResultEntity. Dengan menggunakan dua perulangan bersarang, fungsi ini memeriksa setiap entitas dalam setiap slice dalam struktur penyimpanan, dan jika ditemukan entitas web dengan URL yang sama, fungsi mengembalikan nilai true.
<pre>func appendToResult(storage *[][]ResultEntity, level int, index int, webEntity web, saveRes *[][]web)</pre>	Fungsi appendToResult bertujuan untuk menambahkan hasil pencarian ke dalam daftar hasil yang disimpan. Fungsi ini menerima input berupa struktur penyimpanan hasil, tingkat kedalaman pencarian, indeks dari hasil parent, entitas web yang akan ditambahkan, dan daftar hasil yang disimpan. Fungsi ini bekerja dengan cara membangun jalur dari hasil pencarian, dimulai dari hasil terakhir dan

	<p>bergerak mundur ke parent hingga mencapai akar pencarian. Setelah jalur selesai dibangun, hasilnya dibalik dan kemudian dimasukkan ke dalam daftar hasil yang disimpan.</p>
<pre>func isResultNotInSaveRes(result []web, saveRes *[][]web) bool</pre>	<p>Fungsi isResultNotInSaveRes bertujuan untuk memeriksa apakah hasil pencarian sudah ada dalam daftar hasil yang disimpan. Fungsi ini membandingkan setiap hasil yang baru ditemukan dengan setiap hasil yang sudah ada dalam daftar hasil yang disimpan. Jika hasil baru tidak ada di dalam daftar hasil yang disimpan, fungsi mengembalikan nilai true, menunjukkan bahwa hasil tersebut belum pernah ditemukan sebelumnya.</p>
<pre>func isSameResult(res1 []web, res2 []web) bool</pre>	<p>Fungsi isSameResult memiliki tujuan untuk memeriksa apakah dua hasil pencarian identik. Fungsi ini membandingkan setiap entitas web dari dua hasil pencarian, dan jika semua entitas web memiliki URL yang sama dalam urutan yang sama, fungsi mengembalikan</p>

	nilai true, menunjukkan bahwa kedua hasil tersebut identik.
<pre>func reverse(res *[]web)</pre>	Fungsi reverse bertujuan untuk membalik urutan elemen-elemen dalam sebuah slice entitas web. Dengan menggunakan teknik swap, fungsi ini membalikkan urutan elemen-elemen dalam slice entitas web yang diberikan.

4.2 Source Code

Source Code pada bagian ini hanya akan menampilkan implementasi pada backend karena pada backend terdapat program utama dari proses pencarian solusi sehingga implementasi frontend tidak ada dalam bagian ini.

1. scraper.go

```
package main

import (
    "encoding/json"
    "fmt"
    "runtime/debug"
    "log"
    "net/http"
    "strings"
    "time"

    // "github.com/PuerkitoBio/goquery"
    "github.com/gorilla/mux"
)

type web struct {
    Url  string `json:"url"`
    Title string `json:"title"`
}
```

```

type Input struct {
    Keyword string `json:"keyword"`
    Start   string `json:"start"`
    Lang    string `json:"lang"`
}

type ResultEntity struct {
    index    int   // index of parent
    webEntity web
}

var GlobalLimit int

func main() {
    // Set Runtime Limit
    debug.SetMaxThreads(500)
    // Create Router
    router := mux.NewRouter()

    // Handle route
    router.HandleFunc("/", helloWorld).Methods("GET")
    router.HandleFunc("/api/scrape/bfs2", bfsScrapeHandler).Methods("POST")
    router.HandleFunc("/api/scrape/bfs", gocollytest).Methods("POST")
    router.HandleFunc("/api/scrape/ids", idsScrapeHandler).Methods("POST")

    enhancedRouter := enableCORS(jsonContentTypeMiddleware(router))

    log.Fatal(http.ListenAndServe(":8000", enhancedRouter))
}

func enableCORS(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Access-Control-Allow-Origin", "*")
        w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")
        w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

        if r.Method == "OPTIONS" {

```

```

        return
    }

    next.ServeHTTP(w, r)
}
}

func jsonContentTypeMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Content-Type", "application/json")
        next.ServeHTTP(w, r)
    })
}

func helloWorld(w http.ResponseWriter, r *http.Request) {
    json.NewEncoder(w).Encode("Hello World")
}

/*
=====
=====
===== BFS
Scrape=====
=====
===== */
func bfsScrapeHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Println("Start BFS scraping...")

    // Var lokal
    var count int // Total number of web entities scraped
    var res [][]web // Slice to keep the result of the scraping

    // Decode the request body into an Input struct
    var i Input
    err := json.NewDecoder(r.Body).Decode(&i)

```

```

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

// Base URL
BASEURL := "https://" + i.Lang + ".wikipedia.org"

// Start scraping
timeStart := time.Now()
bfsScrape2(web{/wiki/" + strings.ReplaceAll(i.Start, " ", "_"), i.Start}, i.Keyword, &count,
&BASEURL, &res)
timeEnd := time.Now()

// Encode the result into a struct and send it as a response
result := struct {
    Webs   [][]web
    Time   string
    Total  int
} {
    Webs:  res,
    Time:  timeEnd.Sub(timeStart).String(),
    Total: count,
}

fmt.Println("End scraping...")

json.NewEncoder(w).Encode(result)
}

/*
=====
===== */
/* ===== */
/* ===== BFS
Scrape(USED) ===== */

```

```

/*
=====
===== */

func gocollytest(w http.ResponseWriter, r *http.Request) {
    // Decode the request body into an Input struct
    var i Input
    err := json.NewDecoder(r.Body).Decode(&i)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    var res [][]web
    var allWebs = []web{}

    // Base URL
    BASEURL := "https://" + i.Lang + ".wikipedia.org"

    // Start scraping
    timeStart := time.Now()
    // gocollyScrape(web{"wiki/" + strings.ReplaceAll(i.Start, " ", "_"), i.Start}, i.Keyword, &count,
    &BASEURL, &res)
    gocollyScrapeBase(web{"wiki/" + strings.ReplaceAll(i.Start, " ", "_"), i.Start}, i.Keyword,
    BASEURL, &res, &allWebs)
    timeEnd := time.Now()

    // Encode the result into a struct and send it as a response
    result := struct {
        Webs  [][]web
        Time  string
        Total int
    }{
        Webs:  res,
        Time:  timeEnd.Sub(timeStart).String(),
        Total: len(allWebs),
    }

    fmt.Println("End scraping...")
}

```

```

        json.NewEncoder(w).Encode(result)
    }

/*
=====
===== */
/* =====IDS
Scrape===== */
/*
=====
===== */
===== */

func idsScrapeHandler(w http.ResponseWriter, r *http.Request) {
    var input Input
    if err := json.NewDecoder(r.Body).Decode(&input); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    baseURL := "https://" + input.Lang + ".wikipedia.org"
    startWeb := web{Url: "/wiki/" + strings.ReplaceAll(input.Start, " ", "_"), Title: input.Start}
    tree := Tree{Root: Node{Value: startWeb, Depth: 0}}
    var webs []web
    var count int
    var allWebs []web

    timeStart := time.Now()
    findSolution(&tree.Root, input.Keyword, &count, baseURL, &webs, &allWebs)
    timeEnd := time.Now()

    result := struct {
        Webs [][]web
        Time string
        Total int
    }{
        Webs: [][]web{webs},

```

```

    Time: timeEnd.Sub(timeStart).String(),
    Total: len(allWebs),
}

fmt.Println("End scraping...")
json.NewEncoder(w).Encode(result)
}

```

2. BFSColly.go

```

package main

import (
    "fmt"
    "regexp"
    "strings"
    "time"
    "sync"

    "github.com/gocolly/colly/v2"
)

func gocollyScrapeBase(w web, keyword string, BASEURL string, saveRes *[][]web, allWebs
*[]web) {
    // Initialize local variable
    timeStart := time.Now()
    var storage = [][]ResultEntity{}
    var nextLevel = []ResultEntity{}
    var found = false
    var level = 0

    // Initialize storage
    temp := ResultEntity{0, w}
    storage = append(storage, []ResultEntity{temp})

    var wg sync.WaitGroup

    go tb.AddTokens(MaxToken, 1*time.Second) // Tambahkan Max token setiap detik

```

```

// Start BFS scraping until found
for !found {

    ch := make(chan ResultEntity)

    // Loop through current level
    for index, res := range storage[level] {
        wg.Add(1)
        go gocollyScrape(res.webEntity, keyword, BASEURL, index, &found, level, saveRes,
&storage, ch, &wg, allWebs)

        // Limit time to 5 minutes
        if time.Since(timeStart) > 5*time.Minute {
            fmt.Println("Time limit reached")
            break
        }
    }

    go func() {
        wg.Wait()
        close(ch)
    }()
}

// Add webs to next level
for w := range ch {
    nextLevel = append(nextLevel, w)
}

storage = append(storage, nextLevel) // Add next level to storage
level++ // Increment level
nextLevel = []ResultEntity{} // empty next level

// Limit time to 5 minutes
if time.Since(timeStart) > 5*time.Minute {
    fmt.Println("Time limit reached")
    return
}
}
}
}

```

```

func gocollyScrape(w web, keyword string, BASEURL string, index int, found *bool, level int,
saveRes *[][]web, storage *[][]ResultEntity, ch chan<- ResultEntity, wg *sync.WaitGroup, allWebs
*[]web) {

    defer wg.Done()

    // Tunggu sampai token tersedia
    for !tb.Consume() {
        time.Sleep(100 * time.Millisecond) // Tidur selama durasi singkat jika token tidak tersedia
    }

    var webs []web

    // Instantiate default collector
    c := colly.NewCollector(
        // colly.AllowedDomains(BASEURL),
        // colly.MaxDepth(3),
        colly.Async(true),
        colly.URLFilters(
            regexp.MustCompile(BASEURL + `/wiki/[^\n]*$`),
        ),
    )

    c.CacheDir = "./cache"

    c.Limit(&colly.LimitRule{
        DomainGlob: "*",
        Parallelism: 10,
    })

    c.SetRequestTimeout(100 * time.Second)

    // On every a element which has href attribute call callback
    c.OnHTML("a[href]", func(e *colly.HTMLElement) {
        link := e.Attr("href")
        title := e.Attr("title")
        if strings.HasPrefix(link, "/wiki/") && !strings.Contains(link, ":") && !strings.Contains(link,
        "%3A")){

```

```

        if !containsWebEntity(web{link, title}, webs) {
            webs = append(webs, web{link, title})
            ch <- ResultEntity{index, web{link, title}}
        }
        if (title == keyword) {
            *found = true
            appendToResult(storage, level, index, web{link, title}, saveRes)
        }
    }
}

c.OnResponse(func(r *colly.Response) {
    if !containsWebEntity(web{r.Request.URL.Path, r.Request.URL.Path}, *allWebs) {
        *allWebs = append(*allWebs, web{r.Request.URL.Path, r.Request.URL.Path})
    }
})

c.OnScraped(func(r *colly.Response) {
    fmt.Println("Scraped", r.Request.URL)
})

c.OnError(func(r *colly.Response, err error) {
    fmt.Println("Error:", r.Request.URL, err)
})

// Start scraping
c.Visit(BASEURL + w.Url)

c.Wait()
}

```

3. IDSColly.go

```

package main

import (
    //"encoding/json"
    "fmt"
    //"log"
    //"net/http"
)

```

```

"strings"
//time"
"regexp"
//sync"

"github.com/gocolly/colly/v2"
//github.com/gorilla/mux"
)

type Node struct {
    Value  web
    Depth  int
    Children []*Node
}

type Tree struct {
    Root Node
}

func expandTree(node *Node, baseURL string, limit int, count *int, allWebs *[]web) {
    if node.Depth >= limit {
        return
    }

    c := colly.NewCollector(
        colly.Async(true),
        colly.URLFilters(
            regexp.MustCompile(baseURL + `/wiki/[^\n]*$`),
        ),
    )

    c.Limit(&colly.LimitRule{
        DomainGlob: "*",
        Parallelism: 10,
    })

    c.CacheDir = "./cache"
    c.OnHTML("a[href]", func(e *colly.HTMLElement) {

```

```

    href := e.Attr("href")
    title := e.Attr("title")
    if strings.HasPrefix(href, "/wiki/") && !strings.Contains(href, ":") {
        childWeb := web{Url: href, Title: title}
        childNode := Node{Value: childWeb, Depth: node.Depth + 1}
        node.Children = append(node.Children, &childNode)
        if node.Depth+1 < limit {
            expandTree(&childNode, baseURL, limit, count, allWebs)
        }
    }
}

c.OnResponse(func(r *colly.Response) {
    if !containsWebEntity(web{r.Request.URL.Path, r.Request.URL.Path}, *allWebs) {
        *allWebs = append(*allWebs, web{r.Request.URL.Path, r.Request.URL.Path})
    }
})

c.OnScraped(func(r *colly.Response) {
    fmt.Println("Scraped", r.Request.URL)
})

c.OnError(func(r *colly.Response, err error) {
    fmt.Println("Error:", r.Request.URL, err)
})

c.Visit(baseURL + node.Value.Url)
c.Wait()
}

func findSolution(node *Node, keyword string, count *int, baseURL string, webs *[]web, allWebs *[]web) {
    limit := 1
    found := false
    path := []web{}
    for !found {
        expandTree(node, baseURL, limit, count, allWebs)
        found = searchSolution(node, keyword, &path, webs)
        limit++
    }
}

```

```

        }

func searchSolution(node *Node, keyword string, currentPath *[]web, webs *[]web) bool {
    *currentPath = append(*currentPath, node.Value)
    if node.Value.Title == keyword {
        reversePath := make([]web, len(*currentPath))
        copy(reversePath, *currentPath)
        *webs = append(*webs, reversePath...)
        return true
    }
    for _, child := range node.Children {
        if searchSolution(child, keyword, currentPath, webs) {
            return true
        }
    }
    *currentPath = (*currentPath)[:len(*currentPath)-1]
    return false
}

```

4. utils.go

```

package main

// Utils function to check if a web entity is already in the slice
func containsWebEntity (webEntity web, Res []web) bool {
    for _, w := range Res {
        if w.Url == webEntity.Url {
            return true
        }
    }
    return false
}

func isStorageContainsWebEntity(webEntity web, storage *[][]ResultEntity) bool {
    for _, res := range *storage {
        for _, r := range res {
            if r.webEntity.Url == webEntity.Url {
                return true
            }
        }
    }
}

```

```

        }

        return false
    }

func appendToResult(storage *[][]ResultEntity, level int, index int, webEntity web, saveRes *[][]web)
{
    // Get the result
    var result = []web{ }

    // Append the web entity to the result
    result = append(result, webEntity)

    // Get the parent index
    var parentIndex = index

    // Loop through the storage to get the result
    for i := level; i >= 0; i-- {
        // Append the web entity to the result
        result = append(result, (*storage)[i][parentIndex].webEntity)
        // Get the parent index
        parentIndex = (*storage)[i][parentIndex].index
    }

    // Reverse the result
    reverse(&result)

    // Append the result to the saveRes
    if isResultNotInSaveRes(result, saveRes) {
        *saveRes = append(*saveRes, result)
    }
}

func isResultNotInSaveRes(result []web, saveRes *[][]web) bool {
    for _, res := range *saveRes {
        if isSameResult(res, result) {
            return false
        }
    }
    return true
}

```

```
}
```

```
func isSameResult(res1 []web, res2 []web) bool {
    if len(res1) != len(res2) {
        return false
    }
    for i := 0; i < len(res1); i++ {
        if res1[i].Url != res2[i].Url {
            return false
        }
    }
    return true
}
```

```
func reverse(res *[]web) {
    for i, j := 0, len(*res)-1; i < j; i, j = i+1, j-1 {
        (*res)[i], (*res)[j] = (*res)[j], (*res)[i]
    }
}
```

5. TokenBucket.go

```
● ○ ●
1 package main
2
3 import (
4     "sync"
5     "time"
6 )
7
8 // Buat struktur untuk token bucket
9 type TokenBucket struct {
10     tokens int
11     mutex sync.Mutex
12 }
13
14 // Buat fungsi untuk mengkonsumsi token
15 func (tb *TokenBucket) Consume() bool {
16     tb.mutex.Lock()
17     defer tb.mutex.Unlock()
18
19     if tb.tokens > 0 {
20         tb.tokens--
21         return true
22     } else {
23         return false
24     }
25 }
26
27 // Buat fungsi untuk menambahkan token secara berkala
28 func (tb *TokenBucket) AddTokens(rate int, period time.Duration) {
29     for range time.Tick(period) {
30         tb.mutex.Lock()
31         tb.tokens = rate
32         tb.mutex.Unlock()
33     }
34 }
35
36 var MaxToken = 25
37 // Inisialisasi token bucket
38 var tb = TokenBucket{tokens: MaxToken} // Atur jumlah token awal dan tingkat maksimum
```

4.3 Penjelasan Tata Cara Penggunaan Program

Berikut adalah interface dari aplikasi web yang telah dibangun.

The screenshot displays three pages of a web application:

- About Us:** A dark-themed page with blue circular patterns. It features the title "About Us" and a welcome message: "Welcome to GasAja WikiRace (Solver)! We're the platform dedicated to solving WikiRace challenges accurately and efficiently. GasAja WikiRace (Solver) utilizes search algorithms like Iterative Deepening Search and BFS (Breadth-First Search) to help you navigate through Wikipedia article networks swiftly."
- Welcome to GasAja WikiRace Solver!**: A search interface with a dark background and blue highlights. It includes fields for "Algorithm" (set to "Breadth First Search"), "Start Keyword" (set to "Munich"), "Search Keyword" (set to "Munich"), and "Language" (set to "English"). A "Start" button is at the bottom.
- Result**: A results page showing the search parameters and the first result. The parameters are:
 - Algorithm: Breadth First Search
 - Start Keyword: Munich
 - Search Keyword: MunichThe result section shows "Result 1" with a link to "Munich (<https://en.wikipedia.org/wiki/Munich>)". To the right, there are three boxes: "Time: 0s", "Total artikel yang dilalui: 0", and "Total artikel yang dicari: 0". The footer of this page includes the text "© GasAja 2024".

Program dapat dijalankan dengan mengikuti tata cara penggunaan sebagai berikut:

1. Mengisi Start Keyword

User diminta untuk memasukkan judul awal (Start Keyword) yang merupakan halaman Wikipedia yang ingin dijadikan titik awal pencarian. Misalnya, pengguna memasukkan "Munich".

2. Mengisi Search Keyword

User diminta untuk memasukkan judul artikel tujuan (keyword) yang akan dicari dalam proses scraping. Misalnya, pengguna memasukkan "Fischlham".

3. Memilih Algoritma

Pengguna diminta untuk memilih algoritma yang ingin digunakan untuk proses scraping. Dalam kasus ini, user dapat memilih algoritma BFS (Breadth-First Search) atau IDS (Iterative Deepening Search).

4. Menekan tombol Start

Setelah mengisi kedua input dan memilih algoritma, pengguna dapat mengklik tombol "Start" untuk memulai proses scraping.

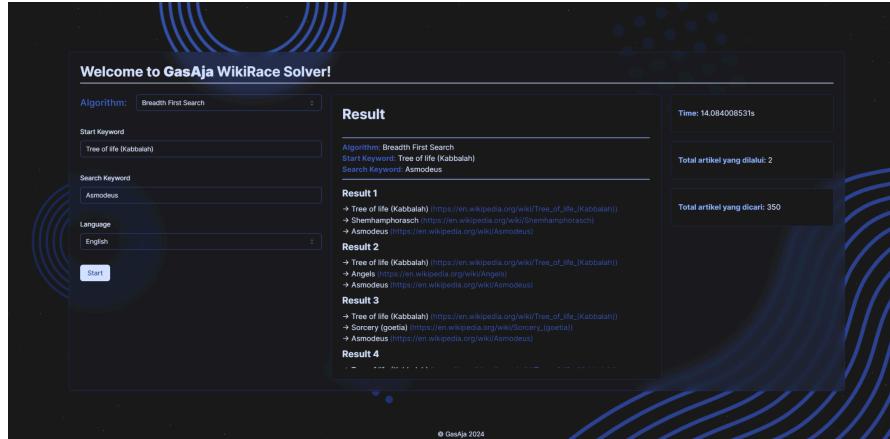
5. Menunggu Hasil

Setelah mengklik tombol "Start", pengguna diminta untuk menunggu hasilnya. Program akan melakukan scraping pada halaman Wikipedia dengan judul awal yang diberikan, menggunakan algoritma BFS untuk mencari halaman yang mengandung kata kunci yang dimasukkan. Setelah proses selesai, hasil scraping akan ditampilkan.

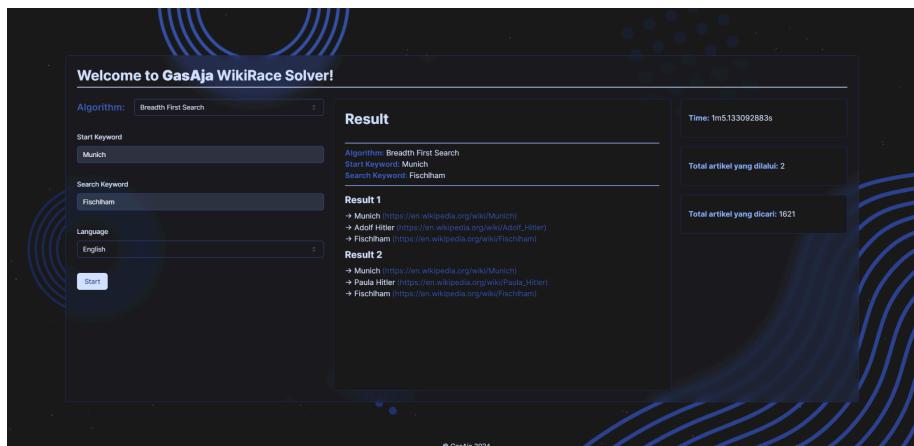
Aplikasi web yang kami bangun mempunyai fitur dark mode serta light mode sehingga apabila user ingin mengganti mode dalam light ataupun dark, user dapat mengklik tombol berbentuk bulan dan bintang yang terdapat di pojok kanan atas halaman web tersebut.

4.4 Hasil Pengujian

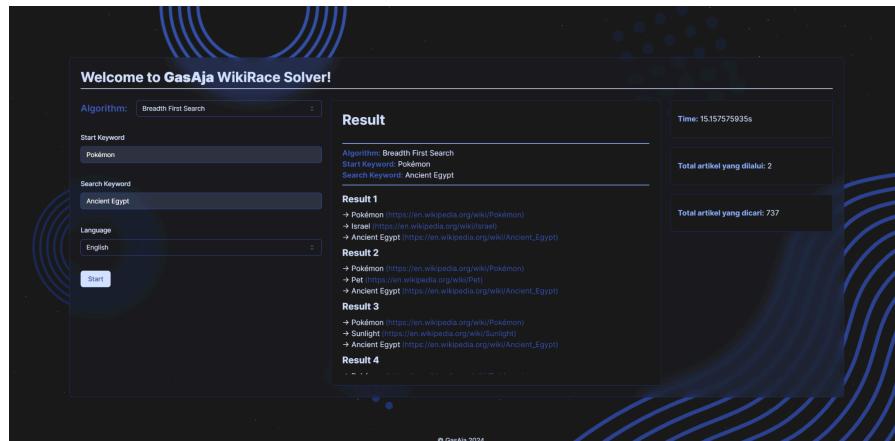
4.4.1 Hasil Pengujian Breadth First Search



Gambar 4.4.1.1 Screenshot BFS Tree of Life (Kabbalah) to Asmodeus
(Sumber: personal localhost)

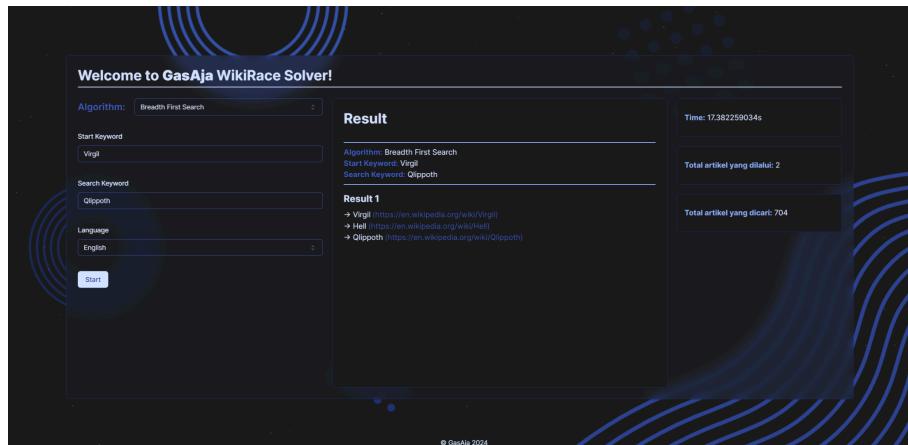


Gambar 4.4.1.2 Screenshot BFS Munich to Fischlham
(Sumber: personal localhost)



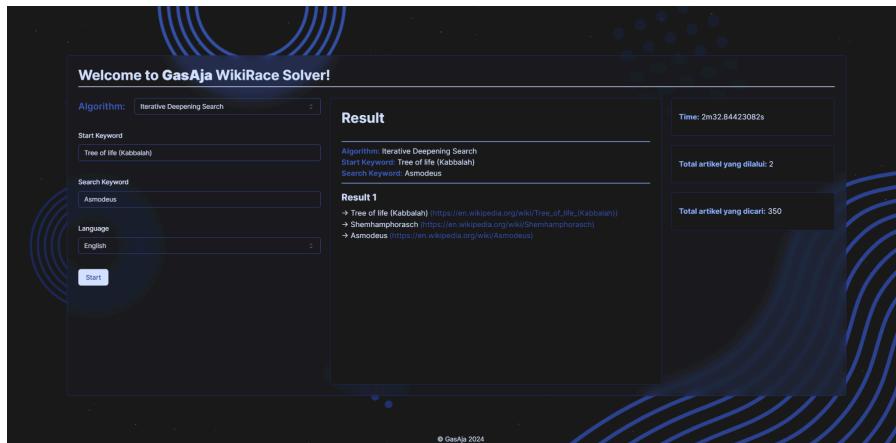
Gambar 4.4.1.3 Screenshot BFS Pokémon) to Ancient Egypt

(Sumber: personal localhost)

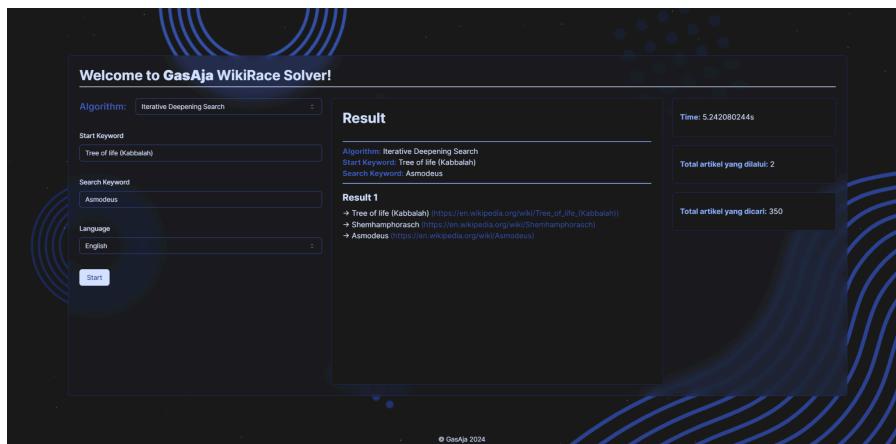


Gambar 4.4.1.4 Screenshot BFS Pokémon ke Ancient Egypt
(Sumber: personal localhost)

4.4.2 Hasil Pengujian Iterative Deepening Search

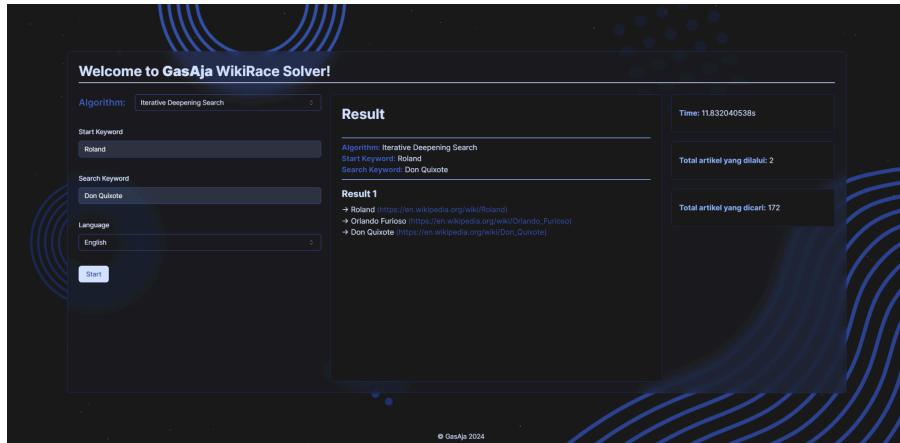


Gambar 4.4.2.1 Screenshot IDS Tree of Life (Kabbalah) ke Asmodeus tanpa Cache
(Sumber: personal localhost)



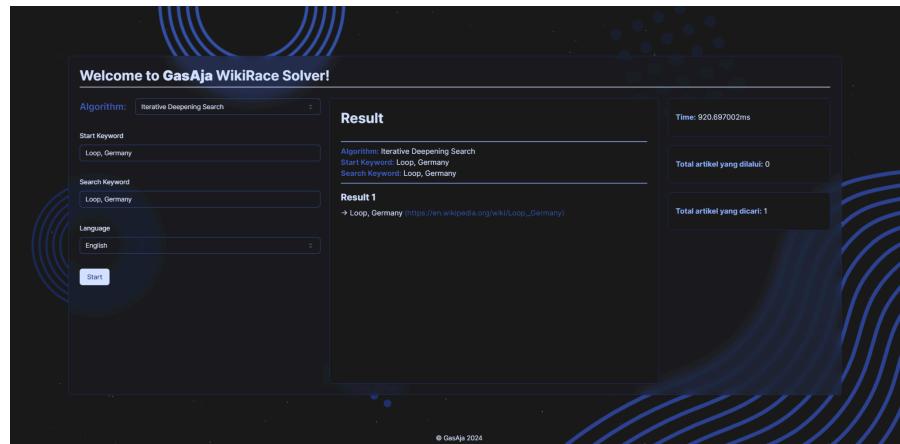
Gambar 4.4.2.2 Screenshot IDS Tree of Life (Kabbalah) ke Asmodeus dengan Cache

(Sumber: personal localhost)



Gambar 4.4.2.3 Screenshot IDS Roland ke Don Quixote

(Sumber: personal localhost)



Gambar 4.4.2.4 Screenshot IDS Loop, Germany ke dirinya sendiri

(Sumber: personal localhost)

4.5 Analisis Hasil Pengujian

Kedua algoritma berhasil untuk mengembalikan hasil yang paling optimal (Dengan derajat terendah) jika memang terdapat jalan, walaupun algoritma IDS yang telah dirancang masih belum bisa mengembalikan seluruh solusi yang ada, melainkan hanya yang paling awal ditemukan. Setelah melakukan *scraping* kedua algoritma dapat menyimpan dan membaca cache yang memungkinkan kecepatan waktu eksekusi bertambah. Tetapi kedua algoritma akan membaca seluruh cabang sebelum melanjutkan ke kedalaman berikutnya

Jika algoritma yang telah dirancang dianalisis, kita bisa melakukan estimasi kompleksitas ruang dan waktunya. Untuk kompleksitas dalam notasi big O, keduanya memiliki kompleksitas $O(b^d)$ dengan b adalah nilai maksimum percabangan yang mungkin dan d adalah kedalaman dari solusi terbaik, tetapi perlu diingat keduanya memiliki kompleksitas yang berbeda mengingat BFS menyimpan simpul terdalam. IDS

memiliki keunggulan dalam kompleksitas ruang. IDS memiliki kompleksitas ruang $O(bd)$ karena IDS hanya memerhatikan jalur saat ini, sedangkan BFS memiliki kompleksitas ruang $O(|V|)$ atau $O(b^d)$ dengan $|V|$ adalah jumlah simpul, karena menyimpan seluruh simpul pada memori ekstra.

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam pembangunan WikiRace Solver berbasis web ini, telah berhasil diimplementasikan algoritma Breadth First Search (BFS) dan Iterative Deepening Search (IDS) untuk menemukan rute terpendek dari satu artikel Wikipedia ke artikel lainnya. Dengan menggunakan teknik scraping dan analisis graf, aplikasi ini mampu memberikan solusi dalam waktu yang relatif cepat. Selain itu, antarmuka pengguna yang responsif dan fitur-fitur seperti mode gelap dan animasi meningkatkan pengalaman pengguna secara keseluruhan.

5.2 Saran

Saran untuk pengembangan aplikasi adalah melakukan optimasi secara terus-menerus guna menjaga kinerja dan responsivitas aplikasi, baik dari segi algoritma pencarian maupun penggunaan penyimpanan. Selain itu, disarankan untuk terus memperdalam pengetahuan tentang teknik scraping dan konkurensi karena keduanya memiliki peran penting dalam pengembangan aplikasi web semacam ini. Selalu meningkatkan fitur aplikasi juga diperlukan untuk meningkatkan kualitas dan kepuasan user secara berkelanjutan.

Perlu diketahui pula, untuk mempelajari *scraping* sebaiknya menggunakan situs-situs yang memang didesain untuk hal tersebut, tidak seperti Wikipedia yang banyak orang publik menggunakan. Scraping dapat mengganggu ketersediaan dari situs untuk orang lain. Sehingga seseorang atau sesuatu yang melakukannya terlalu cepat dapat terkena blokir dan melanggar robots.txt dari Wikipedia di mana robot yang diterima adalah robot yang pelan. Menyebabkan algoritma yang dibuat harus memiliki kecepatan terbatas walaupun harus bisa melakukan prosesnya dalam waktu tertentu.

5.3 Refleksi

Proses pembangunan WikiRace Solver telah memberikan pengalaman berharga dalam pemahaman tentang scraping data, algoritma pencarian, dan pengelolaan konkurensi dalam lingkungan pemrograman. Melalui proyek ini, telah dipelajari bagaimana mengintegrasikan berbagai teknologi dan algoritma untuk mencapai solusi yang efektif. Proyek ini juga menjadi pengingat bahwa pembelajaran kontinu dan kerja keras adalah kunci untuk meraih kesuksesan dalam pengembangan perangkat lunak.

LAMPIRAN

- 1. Link Repository Github :** https://github.com/ValentinoTriadi/Tubes2_GasAja/
- 2. Link Video :**
https://drive.google.com/file/d/1imGFmR-FANnPFe0votl_K6YCK7yLLP-w/view?usp=drive_link

DAFTAR PUSTAKA

- Rinaldi Munir. 2024. "Breadth/Depth First Search (BFS/DFS) (Bagian 1)"
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>
- Rinaldi Munir. 2021. "Breadth/Depth First Search (BFS/DFS) (Bagian 2)"
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
- Rinaldi Munir. 2023. "Graf (Bagian 1)"
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf>
- Rinaldi Munir. 2018. "Route/Path Planning"
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/A-Star-Best-FS-dan-UCS-\(2018\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/A-Star-Best-FS-dan-UCS-(2018).pdf)
- Chanda, Subha. 2021. "Web Scraping In Golang With Colly".
<https://www.scrapingbee.com/blog/web-scraping-go/>
- Bodnar, Jan. 2024. "Go colly".
https://zetcode.com/golang/colly/#google_vignette