

LAPORAN TUGAS BESAR II

IF 2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Kelompok DebDob

Anggota:

Immanuel Sebastian Girsang (13522058)

Ahmad Mudabbir Arif (13522072)

Muhammad Neo Cicero Koda (13522108)

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

DAFTAR ISI

DAFTAR ISI.....	1
BAB I	
DESKRIPSI TUGAS.....	3
BAB II	
LANDASAN TEORI.....	5
2.1. Penjelajahan Graf.....	5
2.2. BFS (Breadth First Search).....	6
2.3. IDS (Iterative Deepening Search).....	8
2.4. Pembuatan Aplikasi Website.....	9
2.4.1. Frontend Development.....	9
2.4.2. Backend.....	10
2.4.3. Dockerization.....	12
BAB III	
ANALISIS PEMECAHAN MASALAH.....	13
3.1. Langkah-Langkah Pemecahan Masalah.....	13
3.1.1. Menentukan strategi untuk menemukan jarak terpendek antara dua pranala.....	13
3.1.2. Menemukan cara mendapatkan kumpulan pranala yang terdapat pada suatu halaman wikipedia.....	13
3.1.3. Menentukan strategi tukar menukar data antara pengguna dan program.....	14
3.1.4. Mendapatkan hasil serta menampilkannya kepada pengguna.....	15
3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma IDS dan BFS.....	16
3.2.1. Identifikasi Jenis Permasalahan.....	16
3.2.2. Identifikasi Simpul.....	16
3.2.3. Identifikasi Sisi.....	17
3.2.4. Identifikasi optimasi yang ingin dilakukan.....	17
3.3. Fitur fungsional dan arsitektur aplikasi web yang dibangun.....	17
3.3.1. Arsitektur Aplikasi Web.....	17
3.3.2. Fitur Fungsional Aplikasi Web.....	21
3.4. Contoh ilustrasi kasus.....	25
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	28
4.1. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).....	28
4.1.1. Struktur data.....	28
4.1.2. Fungsi.....	29
4.1.3. Prosedur.....	39
4.2. Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).....	40
4.3. Hasil pengujian (screenshot antarmuka dan skenario yang memperlihatkan berbagai kasus yang	

mencakup seluruh fitur pada aplikasi Anda).....	42
4.4. Analisis hasil pengujian.....	50
BAB V	
SIMPULAN DAN SARAN.....	52
5.1. Simpulan.....	52
5.2. Saran dan Refleksi.....	52
LAMPIRAN.....	54
Repositori Github.....	54
Link Video Tugas Besar II Strategi Algoritma.....	54
DAFTAR PUSTAKA.....	54

BAB I

DESKRIPSI TUGAS



Gambar 1.1. Contoh gambar permainan WikiRace

WikiRace atau Wiki Game adalah permainan yang melibatkan Wikipedia, sebuah ensiklopedia daring gratis yang dikelola oleh berbagai relawan di dunia, dimana pemain mulai pada suatu artikel Wikipedia dan harus menelusuri artikel-artikel lain pada Wikipedia (dengan mengeklik tautan di dalam setiap artikel) untuk menuju suatu artikel lain yang telah ditentukan sebelumnya dalam waktu paling singkat atau klik (artikel) paling sedikit. Berikut adalah beberapa aturan umum yang sering digunakan dalam Wikirace:

- Awal dan tujuan: Pemain mulai dari halaman tertentu dan harus mencapai halaman lain yang ditentukan.
- Penggunaan tautan: Pemain hanya boleh menggunakan tautan internal dalam Wikipedia untuk navigasi. Tautan eksternal atau alat pencarian tidak diperbolehkan.
- Batasan waktu atau langkah: Beberapa versi permainan menetapkan batas waktu atau jumlah langkah maksimal.

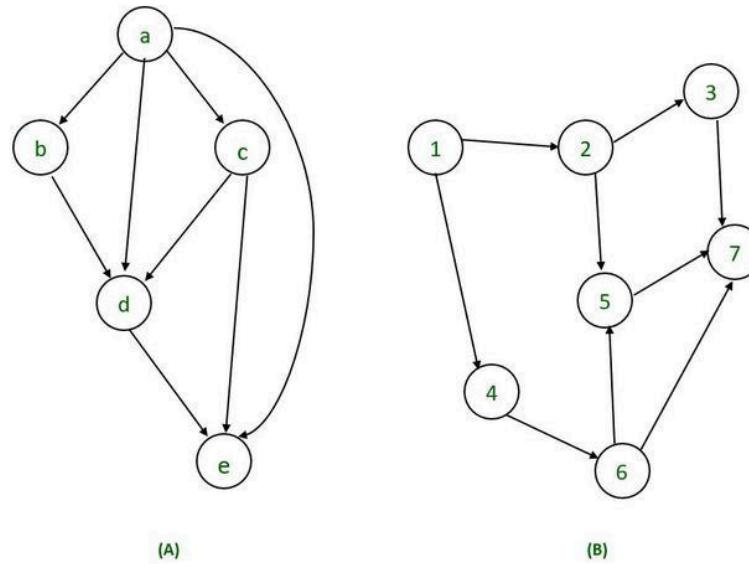
- Variasi permainan: Ada beberapa varian Wikirace, seperti "Shortest Path" (mencari jalur terpendek antara dua halaman) dan "Speed Race" (mencapai tujuan secepat mungkin).

Wikirace bisa dimainkan sendiri atau dalam kelompok. Ini sering dianggap sebagai cara yang menyenangkan untuk menjelajahi pengetahuan dalam Wikipedia dan menguji kreativitas serta kemampuan navigasi pemain. Selain sebagai hiburan, Wikirace juga dapat mengilustrasikan konsep "enam derajat pemisahan" (*six degrees of separation*), di mana kebanyakan halaman di Wikipedia terhubung melalui sejumlah kecil langkah.

BAB II

LANDASAN TEORI

2.1. Penjelajahan Graf



Gambar 2.1.1. Contoh gambar graf

Penjelajahan graf adalah proses mengunjungi semua simpul atau *verteks* dalam sebuah graf, yaitu struktur data yang terdiri dari simpul-simpul yang terhubung dengan sisi (*edges*). Penjelajahan graf sering digunakan dalam algoritma komputer untuk berbagai tujuan, seperti pencarian jalur, penemuan komponen terhubung, dan analisis jaringan. Ada dua metode umum dalam penjelajahan graf:

1. Penjelajahan secara lebar (*Breadth-First Search/BFS*):

BFS menjelajahi graf dengan mengunjungi semua simpul pada tingkat kedalaman yang sama sebelum melanjutkan ke tingkat berikutnya. BFS menggunakan struktur data antrian (*queue*) untuk melacak simpul yang akan dikunjungi berikutnya. BFS sangat berguna untuk mencari jalur terpendek dalam graf yang tidak berbobot.

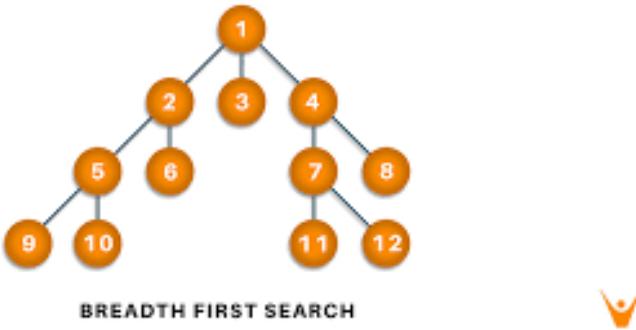
2. Penjelajahan secara dalam (*Depth-First Search/DFS*):

DFS menjelajahi graf dengan bergerak sedalam mungkin sebelum kembali untuk menjelajahi jalur lainnya. DFS menggunakan struktur data tumpukan (*stack*) atau rekursi untuk

melacak perjalanan melalui graf. DFS sering digunakan untuk menemukan komponen terhubung, mendeteksi siklus, dan melakukan topological sorting.

Penjelajahan graf dapat diterapkan pada graf berarah (*directed graph*) maupun graf tidak berarah (*undirected graph*). Setiap metode penjelajahan memiliki aplikasi dan kegunaannya sendiri, tergantung pada masalah yang ingin diselesaikan. Dalam implementasi praktis, penjelajahan graf juga harus memperhitungkan kondisi seperti siklus (*cycles*), komponen terputus, dan node yang telah dikunjungi untuk menghindari *loop* tak berujung.

2.2. BFS (Breadth First Search)



Gambar 2.2.1. Contoh Alur Penelusuran BFS

Algoritma Breadth First Search (BFS) adalah teknik eksplorasi dan pencarian yang menggunakan pendekatan "melebar" untuk menjelajahi semua simpul pada tingkat tertentu sebelum melanjutkan ke tingkat berikutnya. BFS adalah salah satu algoritma dasar dalam ilmu komputer dan digunakan secara luas untuk berbagai aplikasi, terutama dalam konteks graf dan pohon. BFS memulai eksplorasi dari simpul awal (sering disebut sebagai simpul sumber atau *root*), kemudian mengunjungi simpul-simpul tetangga terlebih dahulu sebelum bergerak ke simpul pada tingkat berikutnya. Algoritma ini menggunakan antrian (*queue*) untuk menyimpan urutan simpul yang akan dikunjungi. Langkah-langkah untuk melakukan BFS adalah:

- Inisialisasi: Simpul awal dimasukkan ke dalam antrian dan ditandai sebagai sudah dikunjungi.

- Proses Antrian: Ambil simpul dari awal antrian dan periksa apakah simpul tersebut merupakan solusi.
- Pemeriksaan Simpul Tetangga: Jika bukan solusi, tambahkan semua simpul tetangga yang belum dikunjungi ke dalam antrian dan tandai sebagai sudah dikunjungi.
- Pengulangan: Lanjutkan proses ini sampai antrian kosong atau solusi ditemukan.

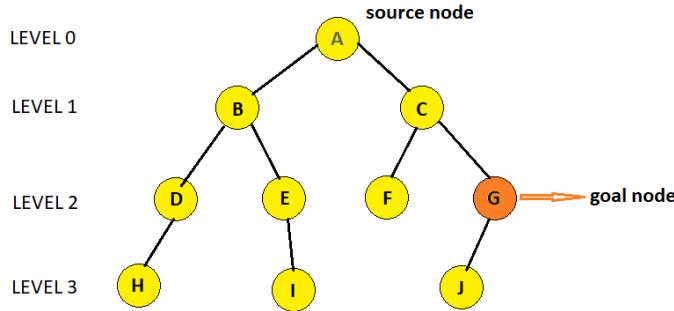
Beberapa contoh kelebihan dari BFS adalah :

- Tidak Akan Menemui Jalan Buntu: BFS selalu menjelajahi semua simpul pada tingkat tertentu sebelum bergerak ke tingkat berikutnya, sehingga tidak pernah terjebak pada jalur buntu, kecuali memang tidak ada solusi yang ditemukan.
- Menemukan Solusi Terpendek: Jika graf tidak berbobot, BFS selalu menemukan jalur terpendek ke tujuan. Ini karena BFS mengunjungi simpul berdasarkan urutan tingkat. Maka tidak mungkin BFS menemukan solusi di kedalaman yang lebih dalam dari solusi yang sedang dicari.
- Dapat Menemukan Lebih dari Satu Solusi: Jika ada beberapa solusi, BFS akan menemukan semua solusi dan memberikan solusi minimum (dengan tingkat terendah).

Beberapa contoh kekurangan dari BFS adalah :

- Membutuhkan Memori Besar: Karena BFS harus menyimpan semua simpul pada tingkat tertentu, ia bisa memakan banyak memori, terutama jika graf sangat lebar atau memiliki banyak simpul.
- Lemah untuk pencarian solusi yang relatif ada dalam kedalaman yang dalam: BFS menguji semua simpul pada tingkat tertentu sebelum melanjutkan ke tingkat berikutnya. Jika solusi berada pada tingkat yang lebih dalam, BFS akan menghabiskan waktu yang cukup lama untuk mencapainya.

2.3. IDS (Iterative Deepening Search)



IDDFS with max depth-limit = 3
 Note that iteration terminates at depth-limit=2
Iteration 0: A
Iteration 1: A->B->C
Iteration 2: A->B->D->E->C->F->G

Gambar 2.3.1. Contoh alur penelusuran IDS

Iterative Deepening Search (IDS) adalah algoritma pencarian yang menggabungkan manfaat dari *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS). IDS digunakan untuk menjelajahi graf atau pohon dengan pendekatan "bertahap dalam", yang secara bertahap meningkatkan kedalaman pencarian sampai solusi ditemukan atau semua simpul telah dieksplorasi. IDS dimulai dengan kedalaman nol dan meningkatkannya satu tingkat setiap iterasi, mirip dengan DFS dalam hal pendekatan kedalaman, tetapi dengan keuntungan tambahan seperti BFS. Algoritma ini menggunakan teknik rekursi untuk menyelesaikan masalah yang kompleks. Langkah-langkah untuk melakukan IDS adalah:

- Inisialisasi: Tentukan kedalaman maksimum untuk setiap iterasi, biasanya dimulai dari nol.
- Pencarian Rekursif: Mulai dari simpul awal, lakukan pencarian DFS sampai batas kedalaman saat ini (tidak ada lagi anak yang bisa ditelusuri).
- Periksa Solusi: Jika simpul tersebut adalah solusi, hentikan pencarian dan kembalikan hasilnya.
- Lanjutkan ke Simpul Tetangga: Jika bukan solusi, lanjutkan ke simpul tetangga, tetapi hanya sampai batas kedalaman saat ini.

- Iterasi: Jika semua simpul pada kedalaman saat ini telah dijelajahi tanpa menemukan solusi, tingkatkan batas kedalaman dan ulangi pencarian.

Beberapa contoh kelebihan IDS adalah :

- Kebutuhan Memori Lebih Rendah: IDS membutuhkan lebih sedikit memori dibandingkan BFS karena hanya menyimpan simpul dalam jalur saat ini, bukan semua simpul pada tingkat tertentu.
- Menemukan Solusi Terpendek (Jika Ada): Seperti BFS, IDS selalu menemukan solusi dengan kedalaman terendah karena pencarian dilakukan bertahap.
- Efisiensi dalam Pencarian di Graf Luas dan Dalam: IDS lebih cocok untuk graf yang sangat dalam karena tidak perlu menyimpan semua simpul di setiap tingkat.

Beberapa contoh kekurangan IDS adalah :

- Pengulangan Pencarian: Karena IDS memulai ulang pencarian dari awal setiap kali kedalaman ditingkatkan, ada banyak pengulangan dalam eksplorasi simpul yang sama.
- Kurang Efisien untuk Graf Dangkal: Untuk graf yang relatif dangkal, IDS bisa menjadi kurang efisien karena pengulangan pencarian.

IDS sering digunakan dalam konteks masalah dengan batasan kedalaman yang tidak diketahui dan memerlukan pengendalian memori yang baik. Kombinasi sifat BFS dan DFS membuat IDS menjadi alat yang fleksibel untuk pencarian dalam graf dan pohon.

2.4. Pembuatan Aplikasi Website

2.4.1. Frontend Development

Front-end development adalah bagian penting dari pengembangan website dan aplikasi yang berfokus pada elemen yang dilihat dan berinteraksi dengan pengguna akhir. Dalam perannya, *Front-End Developer* bertanggung jawab untuk mengubah kode dari *Back-End Developer* menjadi antarmuka grafis yang mudah diakses, dipahami, dan digunakan oleh pengguna. *Front-End Developer* memiliki beberapa tugas penting dalam proses pengembangan, termasuk:

- Pembuatan Antarmuka Pengguna (*UI*): Merancang elemen visual seperti tombol, form, menu, dan tata letak halaman. Elemen-elemen ini harus menarik secara estetika dan mudah dinavigasi.

- Pengalaman Pengguna (*UX*): Mengoptimalkan pengalaman pengguna dengan memastikan antarmuka intuitif, responsif, dan ramah pengguna. Ini juga mencakup memastikan navigasi yang mudah dan interaksi yang lancar.
- Interaksi dengan *Back-End*: Menghubungkan antarmuka dengan data yang disediakan oleh *back-end*. *Front-End Developer* sering bekerja dengan API (Application Programming Interface) untuk mengakses data dan fungsionalitas dari server.
- Responsivitas dan Keterhubungan dengan Perangkat Seluler: Memastikan situs web atau aplikasi dapat digunakan dengan baik di berbagai perangkat, termasuk komputer desktop, tablet, dan smartphone. Ini membutuhkan pendekatan desain responsif yang menyesuaikan antarmuka dengan ukuran layar yang berbeda.

Front-End Developer menggunakan berbagai alat dan teknologi untuk melakukan tugas-tugas mereka. Beberapa teknologi utama yang biasanya digunakan dalam front-end development adalah:

- HTML (HyperText Markup Language): Bahasa markup dasar untuk membuat struktur dan konten halaman web.
- CSS (*Cascading Style Sheets*): Digunakan untuk menata tampilan dan tata letak elemen HTML, seperti warna, font, dan margin.
- JavaScript: Bahasa pemrograman yang memungkinkan interaksi dinamis dan fungsionalitas tambahan di halaman web, seperti animasi, validasi form, dan perubahan konten secara dinamis.
- Framework dan Library Front-End: Framework seperti React, Angular, dan Vue.js memberikan struktur dan alat untuk mengembangkan aplikasi front-end yang kompleks. Library seperti jQuery menyediakan fungsi tambahan untuk interaksi dan animasi.
- Alat Pengembangan dan Build: *Front-End Developer* sering menggunakan alat seperti Webpack, Gulp, dan npm untuk mengelola dan mengotomatiskan proses build, serta alat seperti Git untuk versi kontrol.

2.4.2. Backend

Back-end development adalah aspek krusial dalam pengembangan situs web dan aplikasi yang berfokus pada apa yang terjadi di sisi server. Meskipun tidak terlihat oleh pengguna akhir, komponen back-end mengelola data, logika bisnis, dan komunikasi dengan

front-end untuk memastikan situs web atau aplikasi bekerja dengan lancar dan aman. Mari kita bahas berbagai aspek back-end development, termasuk tugas utama Back-End Developer, alat dan teknologi yang digunakan, serta tantangan yang mereka hadapi. Back-End Developer bertanggung jawab atas berbagai aspek di balik layar dalam pengembangan situs web atau aplikasi. Berikut adalah beberapa tugas utama mereka:

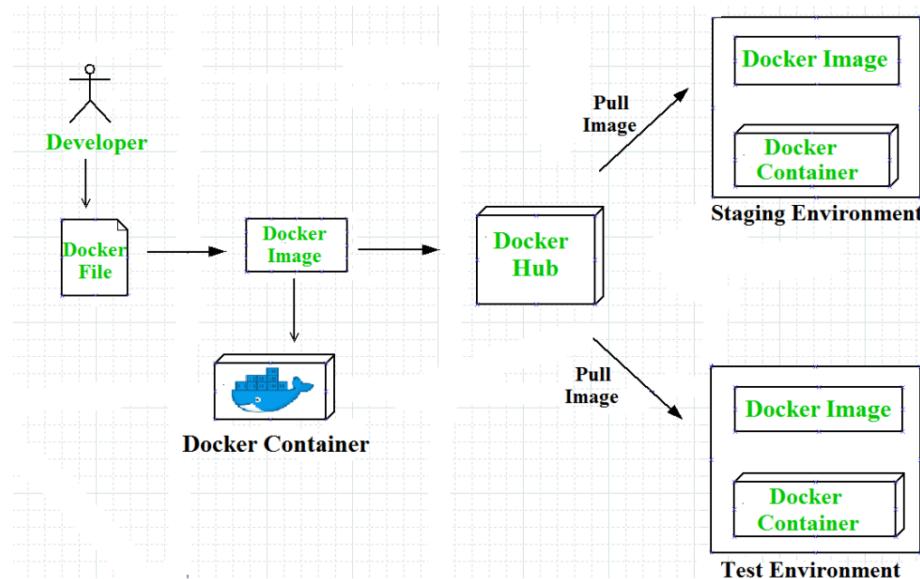
- Merancang dan Mengelola Database: Back-End Developer harus merancang struktur database yang efisien dan mengelola data yang disimpan. Database yang populer mencakup MySQL, PostgreSQL, MongoDB, dan Redis.
- Mengembangkan Logika Bisnis: Logika bisnis mengatur bagaimana data diproses dan bagaimana aplikasi berfungsi. Ini bisa mencakup aturan bisnis, validasi data, dan prosedur operasi standar.
- Mengimplementasikan API: Back-End Developer sering membuat API (Application Programming Interface) yang memungkinkan komunikasi antara front-end dan back-end. API ini biasanya berbasis REST atau GraphQL.
- Mengelola Keamanan: Keamanan adalah aspek penting dalam back-end development. Back-End Developer harus melindungi data sensitif, mencegah akses tidak sah, dan memastikan situs web atau aplikasi tahan terhadap serangan siber.
- Optimasi Kinerja: Back-End Developer harus memastikan aplikasi dapat menangani beban kerja yang tinggi dan tetap responsif. Ini mencakup pengoptimalan kueri database, caching, dan pemanfaatan sumber daya server secara efisien.

Back-End Developer menggunakan berbagai alat dan teknologi untuk menjalankan tugas-tugas mereka. Berikut adalah beberapa teknologi utama yang umum digunakan dalam back-end development:

- Bahasa Pemrograman Server-Side: Bahasa seperti Java, Python, Ruby, PHP, dan Node.js sering digunakan dalam back-end development. Bahasa ini memungkinkan pengembangan logika bisnis dan komunikasi dengan database.
- Framework dan Platform Server-Side: Framework seperti Django (Python), Ruby on Rails (Ruby), Spring Boot (Java), dan Express (Node.js) menyediakan kerangka kerja yang memudahkan pengembangan back-end.

- Database: Back-End Developer bekerja dengan berbagai jenis database, termasuk database relasional seperti MySQL dan PostgreSQL, serta database NoSQL seperti MongoDB dan Redis.
- API dan Integrasi: Teknologi seperti REST, GraphQL, dan gRPC digunakan untuk mengembangkan API yang menghubungkan front-end dengan back-end serta mengintegrasikan layanan pihak ketiga.
- Alat Pengembangan dan Deployment: Alat seperti Docker, Kubernetes, dan Git memungkinkan pengembangan, pengujian, dan penerapan aplikasi yang efisien.

2.4.3. Dockerization



Gambar 2.4.3.1. Ilustrasi proses *Dockerization*

Docker adalah platform sumber terbuka yang memungkinkan pembuatan, pengemasan, dan penerapan aplikasi dalam bentuk container. Container Docker mengisolasi aplikasi dan dependensinya dari sistem host, memungkinkan aplikasi untuk berjalan di berbagai lingkungan tanpa masalah kompatibilitas. Platform Docker menyediakan alat dan layanan untuk membangun, mengelola, dan menjalankan container, serta orkestrasi container untuk lingkungan produksi yang kompleks.

Dockerization adalah proses mengemas aplikasi dan semua dependensinya ke dalam sebuah wadah yang disebut container. Dockerization memungkinkan pengembang untuk membuat aplikasi yang dapat dijalankan di mana saja, tanpa mempedulikan lingkungan

sistem operasi atau infrastruktur yang digunakan. Konsep ini telah merevolusi cara aplikasi dikembangkan, diuji, dan diterapkan, memungkinkan portabilitas, konsistensi, dan skalabilitas yang tinggi.

\

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

3.1.1. Menentukan strategi untuk menemukan jarak terpendek antara dua pranala

Masalah penentuan jarak terpendek pada tugas besar ini akan diselesaikan dengan algoritma IDS dan BFS. Alasan penggunaan IDS dan BFS dibandingkan dengan algoritma lainnya misalkan DFS adalah karena tujuan utama yang ingin dicapai adalah jarak terpendek dari suatu pranala halaman wikipedia ke pranala lainnya, hanya dengan menggunakan semua pranala yang ada di halaman tersebut. Apabila menggunakan pencarian mendalam (DFS) maka ada kemungkinan bahwa jarak yang didapatkan tidak akan yang terpendek dan apabila tidak dibatasi, ada kemungkinan bahwa DFS tidak akan pernah selesai. Maka dari itu, pencarian melebar yang digunakan dalam BFS dan IDS tentu menjadi opsi paling utama untuk menyelesaikan permasalahan ini. Mengenai implementasi algoritma IDS dan BFS secara lebih rinci akan dijelaskan di bab 4.

3.1.2. Menemukan cara mendapatkan kumpulan pranala yang terdapat pada suatu halaman wikipedia

Masalah lain yang perlu dipecahkan untuk menyelesaikan permasalahan utama ini adalah mengetahui pranala apa saja yang terdapat dalam suatu halaman wikipedia. Hal ini dapat dilakukan dengan cara melakukan *scraping* terdapat suatu halaman wikipedia. *Scraper* yang digunakan dalam tugas besar ini adalah Colly, sebuah pustaka *scraping* berbasis Go yang mendukung *scraping* asinkron. Scraper ini diatur untuk mengumpulkan informasi tentang pranala di halaman Wikipedia. *Scraper* mengambil pranala dari bagian konten utama di Wikipedia, yang ditentukan oleh selector "div#mw-content-text a[href]". *Constraint* yang membatasi pranala yang tidak akan diambil antara lain:

1. /wiki/Special: - Merujuk ke halaman khusus yang biasanya memiliki fungsi administratif atau sistem.

2. /wiki/Talk: - Merujuk ke halaman diskusi yang digunakan untuk membahas artikel.
3. /wiki/User: - Merujuk ke halaman profil pengguna Wikipedia.
4. /wiki/Portal: - Merujuk ke halaman portal yang biasanya mengarahkan ke berbagai topik terkait.
5. /wiki/Wikipedia: - Merujuk ke halaman yang terkait dengan informasi internal Wikipedia atau komunitas.
6. /wiki/File: - Merujuk ke halaman yang mengarahkan ke file multimedia seperti gambar atau video.
7. /wiki/Category: - Merujuk ke halaman kategori yang mengelompokkan artikel.
8. /wiki/Help: - Merujuk ke halaman bantuan dan panduan.
9. /wiki/Template: - Merujuk ke halaman template yang digunakan untuk menyusun artikel atau informasi lainnya.
10. /wiki/Template_talk: - Merujuk ke halaman diskusi terkait template.

Pranala yang diawali dengan salah satu dari awalan ini dianggap tidak relevan untuk scraping konten utama Wikipedia, dan oleh karena itu, scraper tidak akan mengambilnya. Selain itu juga akan diperiksa bahwa pranala yang dipilih haruslah berawalan wiki (merujuk ke halaman wikipedia).

3.1.3. Menentukan strategi tukar menukar data antara pengguna dan program

Agar pengguna dapat berinteraksi dengan program secara lebih menarik dan interaktif, diperlukan antarmuka yang dapat menerima masukan dari pengguna dan memprosesnya. Salah satu solusi untuk masalah ini adalah dengan membuat situs web yang memungkinkan pengguna memasukkan judul artikel Wikipedia sebagai titik awal dan tujuan pencarian, memilih metode pencarian, serta menentukan jenis solusi yang diinginkan.

Untuk memungkinkan situs web berkomunikasi dengan program yang dibuat menggunakan bahasa Go, diperlukan perantara yang memungkinkan interaksi antara keduanya. Hal ini dapat dilakukan dengan membangun REST API yang menerima parameter query seperti jenis algoritma yang ingin digunakan, judul

artikel awal, judul artikel tujuan, serta tipe solusi yang diharapkan (satu solusi atau banyak solusi).

Dengan REST API ini, frontend situs web dapat mengirimkan GET request ke backend dengan data yang dimasukkan oleh pengguna. Backend kemudian memproses permintaan tersebut menggunakan algoritma yang sesuai, menemukan rute yang diinginkan, dan mengirimkan hasilnya kembali ke frontend. Hasil ini kemudian dapat ditampilkan kepada pengguna dalam bentuk yang mudah dimengerti, seperti visualisasi graf atau daftar langkah.

3.1.4. Mendapatkan hasil serta menampilkannya kepada pengguna

Setelah hasil didapatkan dari program, langkah selanjutnya adalah menampilkan dan memvisualisasikannya kepada pengguna. Beberapa informasi yang perlu disajikan mencakup jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam milidetik).

Jika pengguna memilih tipe hasil "multiple" atau banyak solusi, diperlukan visualisasi yang dapat menggambarkan semua rute yang mungkin. Salah satu cara untuk melakukan ini adalah dengan menggunakan pustaka seperti React-D3 untuk membuat graf berarah yang menampilkan rute dari titik awal ke tujuan. Dengan pendekatan ini, dapat dibedakan warna simpul untuk setiap level, sehingga pengguna dapat dengan mudah melihat struktur dan urutan penjelajahan.

Visualisasi ini memberikan cara yang intuitif bagi pengguna untuk memahami hasil pencarian. Penggunaan graf berarah dengan warna yang berbeda untuk setiap level memungkinkan pengguna untuk dengan cepat melihat jalur mana yang paling optimal, serta memahami bagaimana algoritma melakukan penjelajahan. Ini juga membantu pengguna untuk membandingkan berbagai solusi yang mungkin, terutama ketika ada lebih dari satu rute yang dapat menghubungkan artikel awal dan tujuan.

3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma IDS dan BFS

Untuk mengonversi masalah wikiRace menjadi masalah yang dapat diselesaikan menggunakan IDS atau BFS, kita perlu memahami komponen-komponen utama dalam

struktur masalah ini. Pertama, kita harus menentukan apa yang akan menjadi simpul (nodes) yang dikunjungi dalam pencarian, serta apa yang menjadi sisi penghubung (links/edges) antara simpul-simpul tersebut. Selain itu, penting juga untuk mempertimbangkan apakah ada biaya (cost) yang mempengaruhi pemilihan simpul, serta apa kriteria yang menentukan hasil yang "optimal" dalam konteks masalah ini.

3.2.1. Identifikasi Jenis Permasalahan

Jenis permasalahan yang dihadapi dalam permainan ini adalah masalah yang berfokus pada urutan atau keterurutan (*order-oriented*). Artinya, solusi yang dihasilkan harus memperhatikan urutan tertentu dalam hal bagaimana rute ditemukan. Dalam konteks permainan wikiRace, hal ini mengarah pada penggunaan graf berarah (*directed graph*), di mana arah dari satu simpul ke simpul lainnya ditentukan oleh tautan antar artikel di Wikipedia.

Graf berarah yang digunakan dalam permainan ini tidak memiliki bobot atau cost tertentu. Dengan kata lain, tidak ada nilai numerik yang membedakan jarak antara satu simpul dengan simpul lainnya. Semua tautan dianggap memiliki bobot yang sama, sehingga solusi yang dicari adalah rute terpendek berdasarkan jumlah simpul atau langkah yang diambil.

3.2.2. Identifikasi Simpul

Dalam permainan ini, simpul yang akan dikunjungi adalah pranala yang ada di suatu halaman Wikipedia (artikel wikipedia terkait). Pendekatan ini masuk akal karena setiap kali mengunjungi sebuah halaman Wikipedia, perlu diperiksa apakah tautan di halaman tersebut mengarah ke tujuan yang diinginkan.

Jadi, setiap simpul dalam permainan mewakili pranala yang bisa diikuti, dan tujuan pencarian adalah menemukan jalur melalui pranala-pranala ini yang menghubungkan halaman awal dengan halaman tujuan. Setiap tautan diperiksa untuk melihat apakah itu adalah jalur yang benar untuk mencapai tujuan yang ditetapkan dalam permainan.

3.2.3. Identifikasi Sisi

Dalam permainan ini, sisi menunjukkan bahwa satu artikel di graf berarah bisa dicapai dengan satu loncatan (satu klik tautan) dari halaman tertentu untuk

mencapai simpul lainnya yang terhubung. Sisi ini mewakili hubungan langsung antara dua simpul, yang pada dasarnya adalah tautan antara dua artikel Wikipedia.

Dengan kata lain, jika kita memiliki dua simpul yang terhubung oleh sebuah sisi, ini berarti dapat berpindah dari satu artikel ke artikel lainnya hanya dengan satu klik pada tautan yang ada di halaman Wikipedia terkait. Sisi ini penting dalam konteks permainan, karena menunjukkan jalur langsung dan menentukan rute yang bisa diambil untuk mencapai tujuan.

3.2.4. Identifikasi optimasi yang ingin dilakukan

Solusi yang diinginkan dalam permainan wikiRace adalah rute yang menghubungkan serangkaian artikel di Wikipedia, dari artikel awal hingga artikel tujuan, dengan jumlah lompatan pranala (klik tautan) paling sedikit. Oleh karena itu, algoritma BFS (Breadth-First Search) dan IDS (Iterative Deepening Search) merupakan pilihan yang paling sesuai karena mereka dirancang untuk menemukan jalur terpendek dalam graf tanpa bobot.

3.3. Fitur fungsional dan arsitektur aplikasi web yang dibangun

3.3.1. Arsitektur Aplikasi Web

Aplikasi web yang dibangun pada tugas besar kali ini menggunakan *framework* NextJS¹⁴ sebagai *tools* untuk mempercepat proses pembuatan Web. Next.js adalah framework populer berbasis React yang memungkinkan pengembangan aplikasi web modern dengan fitur-fitur canggih seperti rendering sisi server (server-side rendering), rendering statis (static site generation), dan dukungan untuk aplikasi serverless. Selain menggunakan NextJS, terdapat beberapa pustaka yang digunakan untuk membantu pembuatan website ini diantaranya:

1. D3.js

D3.js adalah pustaka JavaScript yang kuat untuk visualisasi data. Dalam arsitektur ini, D3.js digunakan untuk menampilkan visualisasi grafis hasil tertentu. Penggunaannya biasanya melibatkan manipulasi DOM untuk membuat grafik interaktif.

2. React-Hook-Form

React-Hook-Form adalah pustaka untuk mengelola formulir input pengguna. Dalam arsitektur ini, pustaka ini memungkinkan Anda membuat formulir yang mudah dikelola, dengan dukungan untuk validasi dan error handling.

3. ZodResolver

ZodResolver adalah alat validasi berbasis pustaka Zod. Ia berperan sebagai penghubung antara React-Hook-Form dan Zod, memungkinkan Anda melakukan validasi input pengguna dengan mudah. Dalam arsitektur ini, ZodResolver membantu menjaga integritas data dengan memastikan input yang valid.

4. ShadCN

ShadCN adalah kumpulan komponen React yang dapat digunakan kembali dan mengikuti prinsip-prinsip desain modern. Komponen ini membantu mempercepat pengembangan antarmuka pengguna (UI) dengan menyediakan elemen yang umum digunakan seperti tombol, formulir, dan lainnya.

5. TailwindCSS

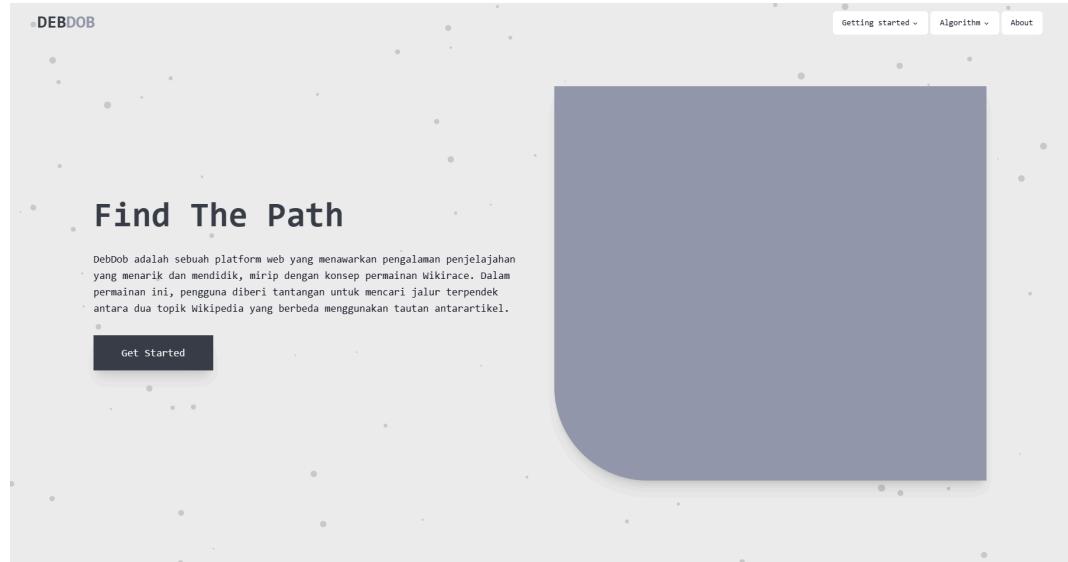
TailwindCSS adalah pustaka CSS utility-first yang memungkinkan Anda mendesain antarmuka dengan cepat. Dalam arsitektur ini, TailwindCSS digunakan untuk styling aplikasi, memberikan fleksibilitas dan kecepatan dalam mendesain.

6. ParticleJS

ParticleJS adalah pustaka JavaScript yang dapat membuat efek partikel pada antarmuka web. Dalam arsitektur ini, ParticleJS bisa digunakan untuk memberikan elemen interaktif atau visual yang menarik, seperti efek latar belakang yang dinamis.

Terdapat beberapa halaman yang kami buat pada aplikasi web ini, diantaranya:

1. Halaman Utama

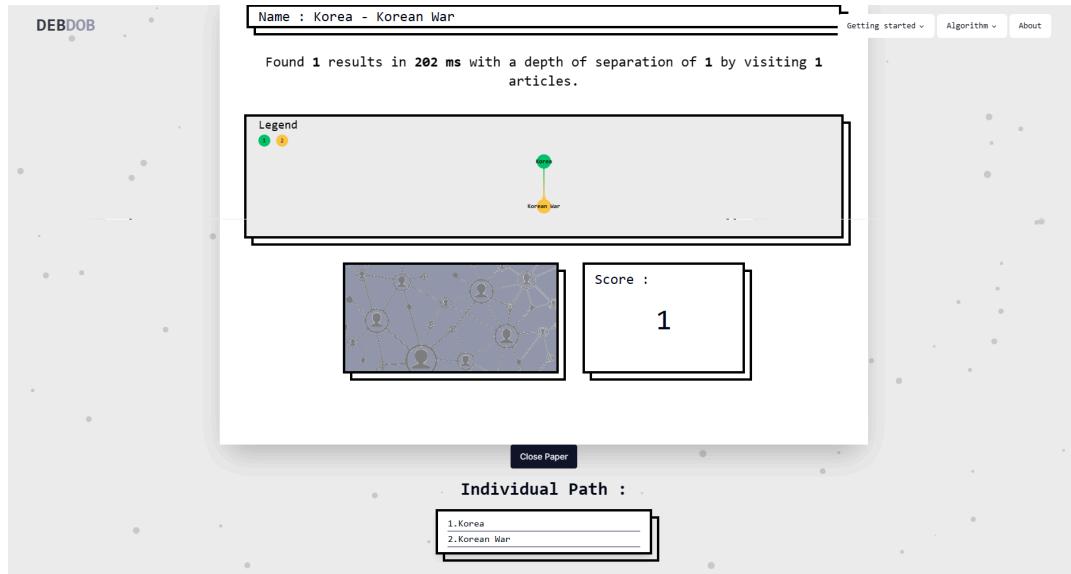
Gambar 3.3.1.1. Gambar *Landing Page*

Halaman ini berfungsi sebagai halaman sambutan bagi pengguna. Di dalamnya, terdapat foto dan deskripsi singkat yang menjelaskan tujuan dan isi situs ini. Pengguna dapat melanjutkan ke halaman formulir dengan mengklik tombol "Get Started."

2. Halaman Form dan Hasil

The screenshot shows the DebDob search interface. At the top right are links for 'Getting started', 'Algorithm', and 'About'. The main title 'Find the shortest paths from' is followed by input fields for 'Source' (Korea) and 'To' (Korean War). Below these are options for 'Method' (BFS selected), 'Result type' (Single Solution selected), and a 'Submit' button. The results section displays a single path entry: 'Name : Korea - Korean War'. Below this, it says 'Found 1 results in 202 ms with a depth of separation of 1 by visiting 1 articles.' A legend at the bottom left shows green and yellow dots.

Gambar 3.3.1.2. Gambar *Halaman Form*



Gambar 3.3.1.3. Gambar Hasil

Halaman ini berfungsi sebagai tempat utama interaksi pengguna dengan program. Melalui halaman ini, pengguna dapat memasukkan artikel asal serta artikel tujuan, metode yang digunakan, serta jenis solusi. Ketika hasil sudah didapatkan, maka akan ditampilkan dalam bentuk graf berserta dengan data-data lain yang relevan untuk ditampilkan.

3. Halaman Penjelasan Singkat Algoritma

Halaman ini berisi penjelasan singkat mengenai algoritma IDS dan BFS, beserta dengan keunggulan serta kelemahan dari setiap algoritma. Halaman ini digunakan sebagai pengetahuan umum kepada pengguna mengenai cara kerja dari program yang kami buat.

4. Halaman *About Us*

Halaman ini didedikasikan untuk memberikan pengakuan dan apresiasi kepada anggota kelompok yang telah berperan penting dalam pembuatan dan penyelesaian website ini. Setiap anggota memiliki peran dan tanggung jawab tertentu yang telah membantu pembuatan website ini dari awal hingga akhir.

5. Halaman *Tech Stacks*

Halaman ini berisi tentang penjelasan mengenai *tech stack* yang digunakan. Halaman ini dibuat sebagai penjelasan singkat mengenai arsitektur

dari web yang telah kami buat beserta inspirasi untuk penggunaan *tech stack* lain kedepannya.

3.3.2. Fitur Fungsional Aplikasi Web

3.3.2.1. Navigation Bar



Gambar 3.3.2.1.1. Gambar *Navigation Bar*

Navigation Bar merupakan sarana yang dapat digunakan oleh para pengguna untuk melakukan navigasi antar halaman-halaman yang terdapat pada website ini. *Navigation bar* berada di bagian atas website dan berisi halaman-halaman yang bisa dikunjungi oleh pengguna beserta dengan logo dari kelompok kami. *Navigation bar* ini dibuat dengan bantuan dari Pustaka UI ShadCN dengan menggunakan berbagai modifikasi yang disesuaikan dengan kebutuhan kelompok kami.

3.3.2.2. Form Input Pengguna

Gambar 3.3.2.2.1. Gambar Form Input Pengguna

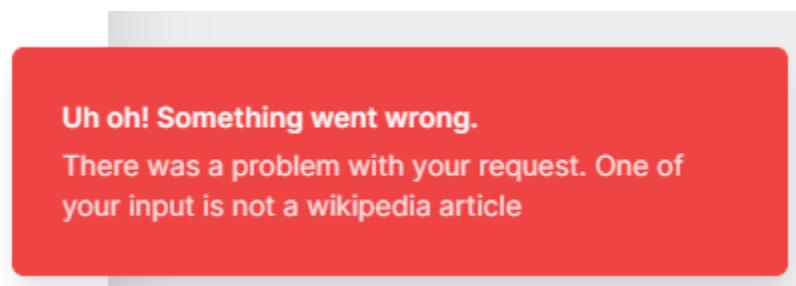
Form input adalah fitur fungsional penting dalam situs web ini. Formulir ini berisi beberapa *field* yang dapat diisi pengguna, seperti artikel asal, artikel tujuan, dan dua pilihan yaitu algoritma yang ingin digunakan (BFS atau IDS) serta tipe solusi yang diinginkan (Satu atau banyak). Untuk memudahkan pengguna dalam memilih artikel, disediakan juga fitur saran input yang diperoleh melalui permintaan

GET ke API Wikipedia. Ada juga tombol untuk menukar sumber dengan tujuan dengan menekan ikon dua panah yang berlawanan arah. Dengan demikian, input sumber dan tujuan akan tertukar.

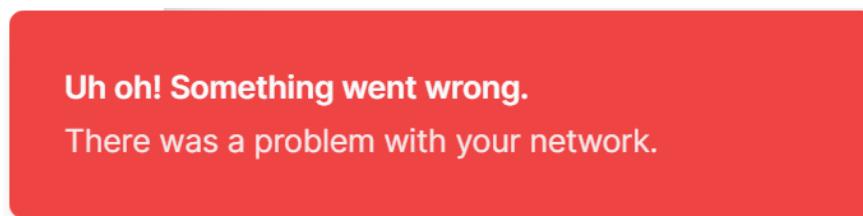
Ketika pengguna menekan tombol submit, sistem akan melakukan validasi terhadap data yang diinput. Beberapa validasi yang dilakukan meliputi memastikan tidak ada field yang kosong, serta semua input adalah judul artikel di Wikipedia bahasa Inggris. Jika semua validasi berhasil, request akan dikirim ke backend, dan animasi *loading* akan ditampilkan untuk menandakan bahwa *request* sedang diproses.

3.3.2.3. Toast Notification

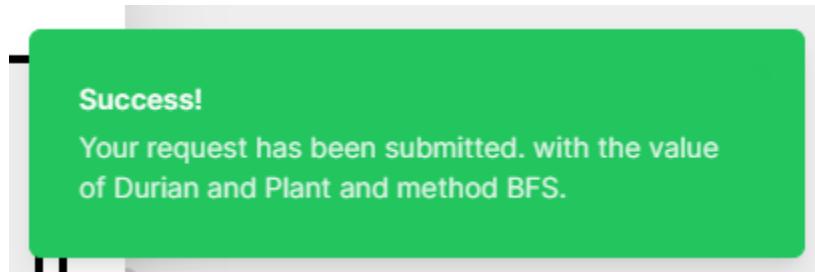
Toast merupakan salah satu sarana untuk memberikan notifikasi kepada pengguna terkait apapun yang terjadi di browser. Pada aplikasi web kami, terdapat 3 kondisi yang memungkinkan untuk munculnya *Toast*.



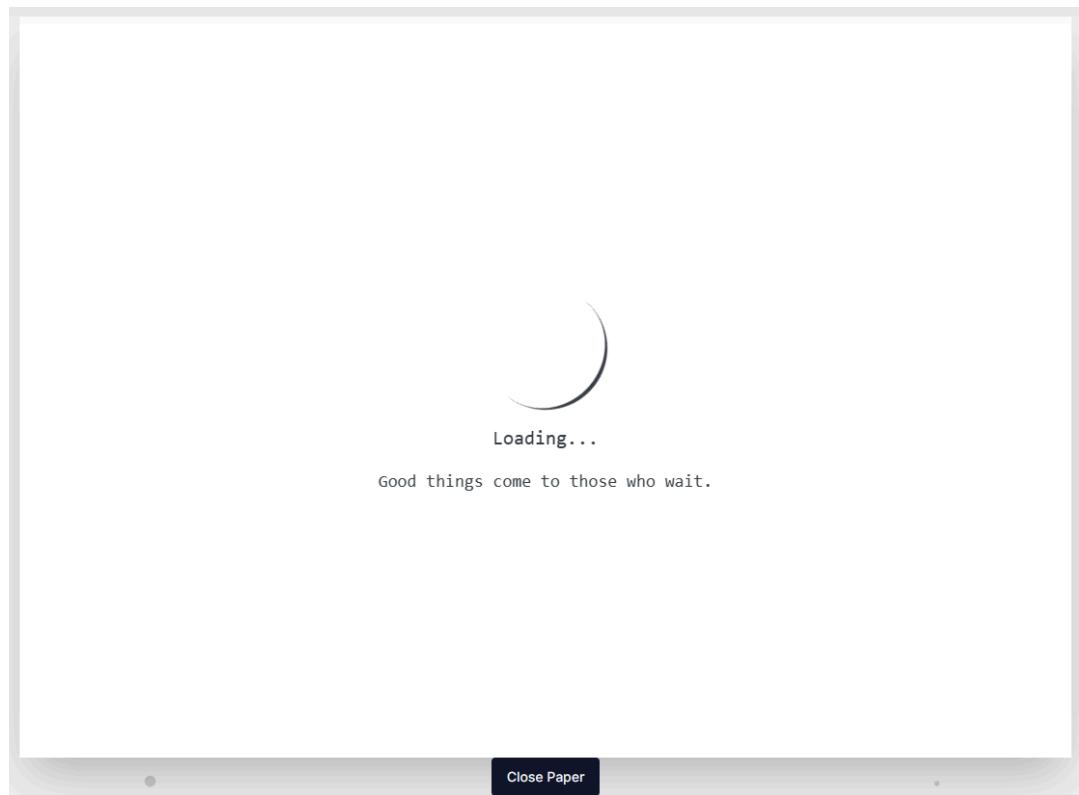
Gambar 3.3.2.3.1. Gambar *Toast* Judul tidak Ditemukan



Gambar 3.3.2.3.2. Gambar *Toast* *request* gagal (Kesalahan di backend atau internet)

Gambar 3.3.2.3.3. Gambar Toast *request* berhasil

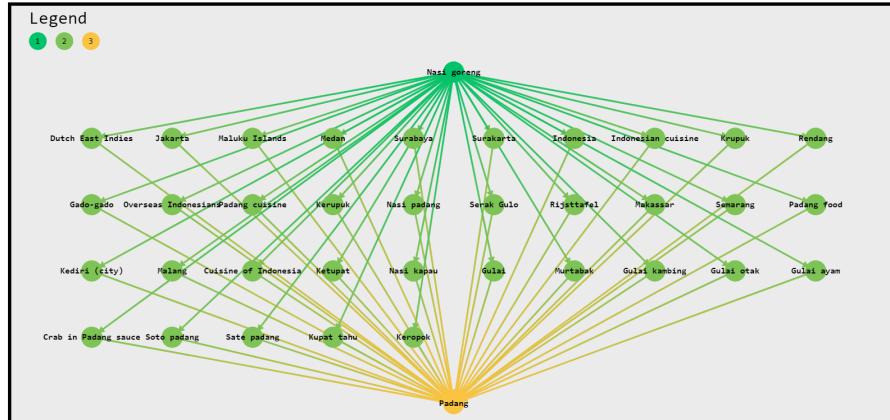
3.3.2.4. Loading



Gambar 3.3.2.4.1. Gambar Loading Animation

Fitur ini digunakan sebagai “pengisi” ketika pengguna menunggu hasil dari *backend*. Ditampilkan sebuah *spinner* yang akan terus berputar hingga *request* selesai di proses, serta tulisan berupa random quotes dari orang-orang terkenal yang akan berganti setiap detik. Fitur ini dibuat untuk membuat pengguna tidak cepat merasa bosan ketika menunggu hasil pengolahan data dari *backend*.

3.3.2.5. Visualisasi Hasil



Gambar 3.3.2.5.1. Gambar Contoh Visualisasi Hasil

Ketika hasil dari *backend* berhasil diterima, maka perlu ditampilkan kepada pengguna. Hal ini dilakukan dengan membuat sebuah graf yang dibuat dengan pustaka D3.js dan juga menampilkan data-data lain yang relevan dalam bentuk teks. Untuk setiap kedalaman pada graf, warna akan dibedakan untuk mempermudah pengguna dalam melihat arah pergerakan. Ketika simpul diklik, maka pengguna akan ter *redirect* ke halaman wikipedia terkait.

3.4. Contoh ilustrasi kasus

Ilustrasi kasus beserta penyelesaiannya akan dilakukan untuk memperjelas analisis pemecahan masalah. Misalkan seseorang ingin mengetahui cara untuk berpindah dari artikel Bandung Institute of Technology ke artikel Joko Widodo. Hal pertama yang dia harus lakukan adalah mengisi judul artikel *source* dan *destination*, memilih tipe *searching* (BFS atau IDS), dan memilih jumlah hasil yang akan ditampilkan (satu solusi atau berbagai solusi). Setelah itu, dia akan menekan tombol *submit* untuk mengirim data tersebut ke *backend*. Masukan dari pengguna akan diterima oleh API dalam bentuk *query parameter*.

The screenshot shows a user interface for finding shortest paths. At the top, it says "Find the shortest paths from". Below that, there are three input fields: "Source" containing "Bandung Institute of Technology", "To" containing "Joko Widodo", and "Destination" which is empty. Between the "Source" and "To" fields is a double-headed arrow icon. Below these fields are two radio buttons under "Method": "BFS" (selected) and "IDS". Under "Result type", there are two radio buttons: "Single Solution" (unchecked) and "Multiple Solution" (selected). At the bottom is a dark blue "Submit" button.

Gambar 3.4.1. Tampilan form entri untuk pengguna

Setelah menerima masukan dari *frontend*, sisi *backend* akan memvalidasi masukan tersebut dengan memastikan bahwa semua parameter terisi dan terisi dengan valid. Selanjutnya, *backend* akan memproses masukan dari pengguna berdasarkan data yang dimasukkan oleh pengguna.

Untuk kasus BFS, pada awal pemanggilan fungsi BFS, satu instansiasi *collector* colly akan dibuat. Konfigurasi terkait *colly* akan diatur terlebih dahulu, seperti *domain* yang diperbolehkan, tingkat paralelisme, kemampuan asinkron, dan perilakunya ketika membuat permintaan, menemukan tag *anchor*, dan selesai melakukan *scraping* terhadap suatu artikel..

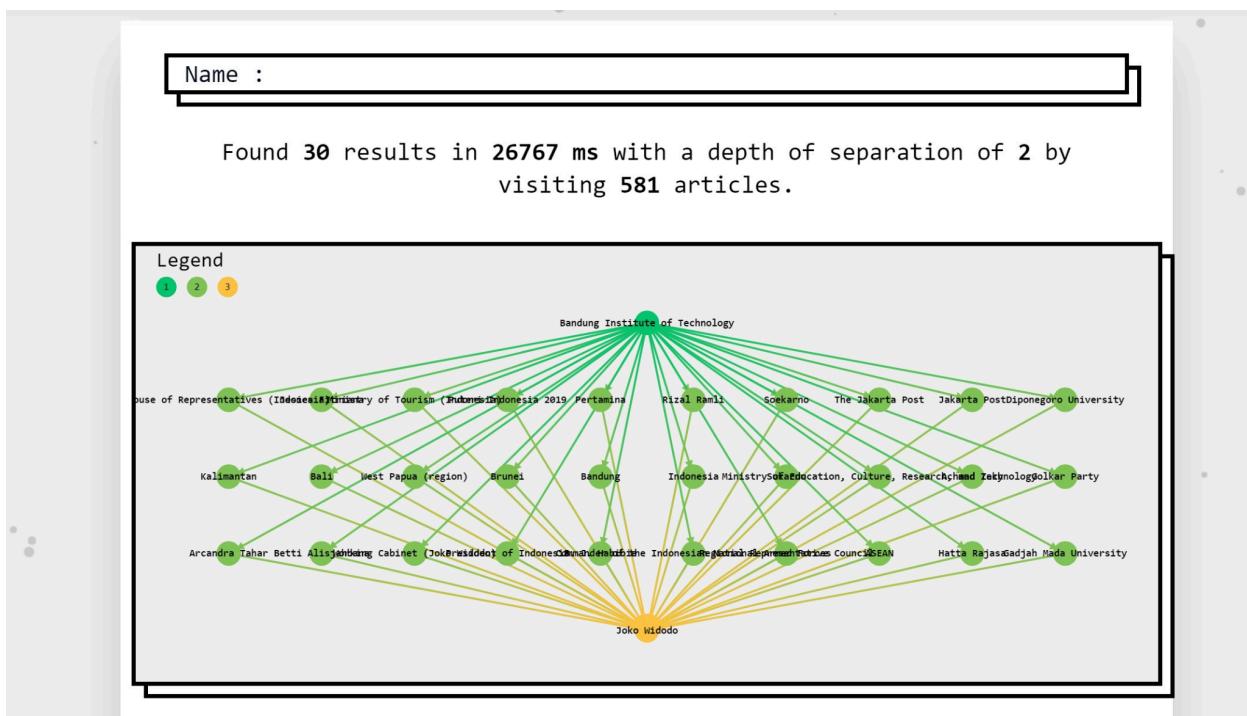
Pada BFS, colly akan mengunjungi artikel awal terlebih dahulu. Artikel tersebut akan dicatat sebagai telah dikunjungi pada *map* bernama *visited*. Setelah itu, colly akan melakukan *scraping* terhadap artikel tersebut. Setiap kali colly menemukan suatu *anchor*, jika tautan pada *anchor* tersebut memenuhi syarat, tautan tersebut beserta induknya akan dicatat pada map bernama *predecessors*. Selanjutnya, tautan tersebut juga akan dimasukkan ke daftar tautan tetangga (*neighborLinks*).

Untuk setiap *neighbor link* yang terdapat pada *neighborLinks*, jika *neighbor link* tersebut belum dikunjungi, *neighbor link* tersebut akan dikunjungi, dicatat sebagai *visited*., dan dilakukan *scraping* untuk mencatat setiap *neighbor link* dari *neighbor link* tersebut. Hal tersebut akan terus terjadi hingga bertemu dengan artikel tujuan. Setelah semua *neighbor link* pada suatu iterasi ditelusuri, *neighbor link* pada iterasi/kedalamannya selanjutnya akan ditelusuri. Pada kasus Bandung

Institute of Technology dan Joko Widodo, Artikel tersebut tidak ditemukan pada iterasi pertama *neighborLinks*, sehingga *neighbor link* dari setiap *link* di *neighborLinks* akan ditelusuri dan disimpan nilainya ke *neighborLinks* iterasi selanjutnya. Pada iterasi selanjutnya, ditemukan artikel Joko Widodo sehingga proses pencarian berhenti.

Untuk kasus IDS, pada awal pemanggilan fungsi IDS, satu instansiasi *collector colly* akan dibuat juga dan konfigurasi terkait *colly* juga akan diatur terlebih dahulu. Fungsi IDS akan melakukan DLS (*depth limited search*) dengan *depth* maksimum sebesar 1 pada awal. Selama artikel tujuan tidak ditemukan, IDS akan melakukan *increment* terhadap *depth* maksimum hingga mencapai artikel tujuan. Selama penelusuran tersebut, *collector* akan menelusuri setiap artikel hingga mencapai kedalaman sebesar *maxDepth*. Setelah menelusuri artikel dengan kedalaman tersebut, *collector* akan melakukan runut balik dan menelusuri artikel selanjutnya.

Setelah bertemu dengan artikel tujuan, jalur awal dari sumber ke tujuan akan ditentukan melalui *map predecessors/predecessorsMulti*. *Predecessors/predecessorsMulti* akan diberikan sebagai parameter ke fungsi *getPath/getPaths* (tergantung solusi tunggal atau ganda) dan fungsi tersebut akan mengembalikan rute lengkap.



Gambar 3.4.2. Hasil penelusuran dari artikel Bandung Institute of Technology ke artikel Joko Widodo (solusi banyak)

Terakhir, *backend* akan mengembalikan hasil (jumlah artikel yang diperiksa, jalur dari artikel sumber ke destinasi, waktu eksekusi) ke *front end* untuk ditampilkan kembali kepada pengguna.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

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

4.1.1. Struktur data



```
1 type URLStore struct {
2     predecessors      map[string]string
3     predecessorsMulti map[string][]string
4     visited           sync.Map
5     numVisited        int
6     neighborLinks    []string
7     resultPath        []string
8     resultPaths       [][]string
9 }
```

Gambar 4.1.1.1. Struktur data URLStore

URLStore merupakan Struktur data buatan yang digunakan sebagai penampung dari hasil-hasil yang didapatkan selama proses mencari jalur terdekat. Predecessors merupakan sebuah map yang menggunakan key berupa string yang adalah link artikel wikipedia terkait, serta bernilai link wikipedia lain yang merupakan *predecessor* dari link terkait. PredecessorsMulti merupakan hal yang mirip, tetapi kali ini digunakan *array of string* sebagai value hasilnya sebab kali ini solusi yang dicari bernilai banyak. Visited merupakan map yang menggunakan key berupa string yaitu link artikel wikipedia, dan bernilai boolean yang dimana akan bernilai false ketika belum dikunjungi, dan true ketika sudah. Numvisited menyimpan berapa banyak artikel yang dikunjungi sebagai sarana menampilkan hasil, sedangkan resultPath dan resultPaths merupakan solusi akhir yang didapatkan program, apabila bernilai banyak, maka digunakan resultPaths.



Gambar 4.1.1.2. Struktur data URLCache

URLCache merupakan Struktur data buatan yang digunakan sebagai penampung sementara dari hasil *caching* yang kami lakukan. Struktur data ini terdiri atas Links yang merupakan map dengan key berupa string yaitu artikel dan bernila *array of string* yang adalah kumpulan link artikel yang terdapat dalam link tersebut. Contoh dari URLCache ketika sudah dalam bentuk JSON adalah sebagai berikut.

```
{  
    "links": [  
        "https://en.wikipedia.org/wiki/$50SAT",  
        "https://en.wikipedia.org/wiki/Amateur_radio",  
        "https://en.wikipedia.org/wiki/Communications_satellite",  
        "https://en.wikipedia.org/wiki/Morehead_State_University",  
        "https://en.wikipedia.org/wiki/International_Designator",  
        "https://en.wikipedia.org/wiki/Satellite_Catalog_Number",  
        "https://en.wikipedia.org/wiki/CubeSat",  
        "https://en.wikipedia.org/wiki/Satellite_bus",  
        "https://en.wikipedia.org/wiki/Coordinated_Universal_Time",  
        "https://en.wikipedia.org/wiki/Dnepr_rocket",  
        "https://en.wikipedia.org/wiki/Dombarovsky_Air_Base",  
        "https://en.wikipedia.org/wiki/Yuzhmash",  
        "https://en.wikipedia.org/wiki/Geocentric_orbit",  
    ]  
}
```

Gambar 4.1.1.3. Contoh JSON cache

4.1.2. Fungsi



```
● ● ●

1 // Check if link is wikipedia article
2 func validLink(link string) bool {
3     invalidPrefixes := []string{"/wiki/Special:", "/wiki/Talk:", "/wiki/User:",
4     "/wiki/Portal:", "/wiki/Wikipedia:", "/wiki/File:", "/wiki/Category:", "/wiki/Help:",
5     "/wiki/Template:", "/wiki/Template_talk:"}
6     for _, prefix := range invalidPrefixes {
7         if strings.HasPrefix(link, prefix) {
8             return false
9         }
10    }
11    return strings.HasPrefix(link, "/wiki/")
12 }
13
14 // Reverses a slice
15 func reverseSlice(slice []string) {
16     for i := 0; i < len(slice)/2; i++ {
17         j := len(slice) - i - 1
18         slice[i], slice[j] = slice[j], slice[i]
19     }
20 }
21
22 // Checks whether or not a string is in a slice
23 func stringInSlice(str string, list []string) bool {
24     for _, item := range list {
25         if item == str {
26             return true
27         }
28     }
29     return false
30 }
```

Gambar 4.1.2.1. Kode berbagai fungsi manipulasi string

Untuk melakukan operasi terhadap *link-link* yang ada, dibuat beberapa fungsi. Fungsi valid link berguna untuk mengecek apakah link yang sedang di-*scrape* oleh *scraper* valid sesuai dengan ketentuan yang ditentukan. Fungsi reverseSlice berguna sebagai fungsi untuk membalik urutan dari suatu *array of string*. Sedangkan fungsi stringInSlice berfungsi mengecek apakah dalam suatu array terdapat string yang kita cari.

```
// Gets full path from map of predecessors
func getPath(predecessors map[string]string, dest string) []string {
    path := make([]string, 0)
    node := dest

    for node != "" { // source has no predecessor
        path = append(path, node)
        node = predecessors[node]
    }

    reverseSlice(path)
    return path
}
```

Gambar 4.1.2.2. Kode getPath

Kode getPaths berfungsi untuk mendapat solusi tunggal dari suatu *map* yang berisi *predecessor* dari suatu artikel.

```
// Gets all paths from map of predecessors
func getPaths(predecessors map[string][]string, src string, dest string) [][]string {
    var paths [][]string
    var resultPaths [][]string

    found := false
    paths = append(paths, []string{dest})
    // builds paths from destination node, every iteration adds the length of the path by one
    for !found {
        currentPaths := paths
        paths = nil
        for _, path := range currentPaths {
            for _, pred := range predecessors[path[len(path)-1]] {
                if pred == src {
                    found = true
                }
                newPath := append(path, pred)
                paths = append(paths, newPath)
            }
        }
    }

    // reverses and adds paths that have src as the final element
    for _, path := range paths {
        if path[len(path)-1] == src {
            reverseSlice(path)
            resultPaths = append(resultPaths, path)
        }
    }
}

return resultPaths
}
```

Gambar 4.1.2.3 Kode getPaths

Kode `getPaths` berfungsi untuk menghasilkan semua solusi dari suatu *map* yang berisi *predecessor* dari suatu artikel.

```
// Multi solution BFS
func BFSMulti(src string, dest string, cache *URLCache) *URLStore {
    // initialize url store and mutex
    urlQueue := NewURLStore()

    var mutex sync.Mutex

    // set up colly config and On... functions
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
        colly.Async(true),
    )

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

    c.OnRequest(Func(r *colly.Request) {
        currentLink := r.URL.String()
        urlQueue.visited.Store(currentLink, true)
        urlQueue.numVisited++
    })

    c.OnHTML("div#mw-content-text a[href]", func(e *colly.HTMLElement) {
        currentLink := e.Request.URL.String()
        neighborLink, _ := url.QueryUnescape(e.Attr("href"))
        if validLink(neighborLink) && urlQueue.HasVisited(neighborLink) {
            mutex.Lock()
            // append to existing predecessor map if already exists, creates new one if not
            if _, ok := urlQueue.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)]; ok && !stringInSlice(currentLink, urlQueue.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)]) {
                urlQueue.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)] = append(urlQueue.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)], currentLink)
            } else if !ok {
                urlQueue.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)] = []string{currentLink}
            }
            urlQueue.neighborLinks = append(urlQueue.neighborLinks, e.Request.AbsoluteURL(neighborLink))
            mutex.Unlock()
        }
    })
}

func validLink(link string) bool {
    return len(link) > 0 && strings.HasPrefix(link, "/")
}
```

Gambar 4.1.2.4 Kode BFSMulti

```
found := false
for !found { // one iteration is one addition to depth
    currentNeighborLinks := urlQueue.neighborLinks
    urlQueue.neighborLinks = nil // copy neighborLinks and clear

    for _, neighborLink := range currentNeighborLinks {
        if !urlQueue.HasVisited(neighborLink) {
            _, ok := cache.Links[neighborLink]
            if ok { // present in cache
                go func(neighborLink string) {
                    urlQueue.visited.Store(neighborLink, true)

                    mutex.Lock()
                    for _, neighborLink2 := range cache.Links[neighborLink] {
                        if _, ok2 := urlQueue.predecessorsMulti[neighborLink2]; ok && !stringInSlice(neighborLink, urlQueue.predecessorsMulti[neighborLink2]) {
                            urlQueue.predecessorsMulti[neighborLink2] = append(urlQueue.predecessorsMulti[neighborLink2], neighborLink)
                        } else if !ok2 {
                            urlQueue.predecessorsMulti[neighborLink2] = []string{neighborLink}
                        }
                    }
                    urlQueue.neighborLinks = append(urlQueue.neighborLinks, cache.Links[neighborLink]...)
                    mutex.Unlock()
                }
                urlQueue.numVisited++
            } (neighborLink)
        } else { // manually scrape
            c.Visit(neighborLink)
        }
    }
    if neighborLink == dest {
        found = true
        break
    }
}

c.Wait()

// get paths from predecessorsMulti
urlQueue.resultPaths = getPaths(urlQueue.predecessorsMulti, src, dest)

return urlQueue
}
```

Gambar 4.1.2.5. Kode BFSMulti (2)

Kode BFSMulti akan melakukan *scraping* secara *breadth first search* dan mengembalikan hasil *scraping* berupa semua *path* yang dapat dilalui dan jumlah artikel yang diperiksa.

```
func BFS(src string, dest string, cache *URLCache) *URLStore {
    // initialize url store and mutex
    urlQueue := NewURLStore()

    // set up colly config and On<...> functions
    var mutex sync.Mutex

    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
        colly.Async(true),
    )

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

    c.OnRequest(func(r *colly.Request) {
        currentLink := r.URL.String()
        urlQueue.visited.Store(currentLink, true)
        urlQueue.numVisited++
    })

    c.OnHTML("div#mw-content-text +a[href]", func(e *colly.HTMLElement) {
        currentLink := e.Request.URL.String()
        neighborLink, _ := url.QueryUnescape(e.Attr("href"))
        if validLink(neighborLink) {
            mutex.Lock()
            if urlQueue.predecessors[e.Request.AbsoluteURL(neighborLink)] == "" && e.Request.AbsoluteURL(neighborLink) != src {
                urlQueue.predecessors[e.Request.AbsoluteURL(neighborLink)] = currentLink
            }
            urlQueue.neighborLinks = append(urlQueue.neighborLinks, e.Request.AbsoluteURL(neighborLink))
            mutex.Unlock()
        }
    })
}

urlQueue.predecessors[src] = ""
c.Visit(src)
```

Gambar 4.1.2.6. Kode BFS

```

urlQueue.predecessors[src] = ""
c.Visit(src)

found := false
for !found { // one iteration is one addition to depth
    currentNeighborLinks := urlQueue.neighborLinks
    urlQueue.neighborLinks = nil // copy neighborlinks and clear

    for _, neighborLink := range currentNeighborLinks {
        if !urlQueue.HasVisited(neighborLink) {
            _, ok := cache.Links[neighborLink]
            if ok { // present in cache
                go func(neighborLink string) {
                    urlQueue.visited.Store(neighborLink, true)

                    mutex.Lock()
                    for _, neighborLink2 := range cache.Links[neighborLink] {
                        neighborLink2, _ := url.QueryUnescape(neighborLink2)
                        if urlQueue.predecessors[neighborLink2] == "" && neighborLink2 != src {
                            urlQueue.predecessors[neighborLink2] = neighborLink
                        }
                    }
                    urlQueue.neighborLinks = append(urlQueue.neighborLinks, cache.Links[neighborLink]...)
                    mutex.Unlock()

                    urlQueue.numVisited++
                }(neighborLink)
            } else { // manually scrape
                c.Visit(neighborLink)
            }
        }
        if neighborLink == dest {
            found = true
            break
        }
    }
    if !found {
        c.Wait()
    }
}

// get paths from predecessors
mutex.Lock()
urlQueue.resultPath = getPath(urlQueue.predecessors, dest)
mutex.Unlock()

```

Gambar 4.1.2.7 Kode BFS (2)

Fungsi BFS akan mengembalikan solusi tunggal dari hasil *scraping* serta jumlah artikel yang dilalui.

```

func DLS(src string, dest string, maxDepth int) *URLStore {
    // initialize new URLStore
    urlStore := NewURLStore()

    // concurrency tools
    var mutex sync.Mutex
    timer := time.NewTimer(2 * time.Second) // Stop DLS if no visits are being made after two seconds
    noVisits := make(chan struct{})

    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    // initialize colly config
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
        colly.MaxDepth(maxDepth),
        colly.Async(true),
    )

    c.Limit(&colly.LimitRule{DomainGlob: "", Parallelism: 20})

    // DLS function to immediately stop execution on first finding of destination
    scraper := func() {
        defer c.Wait()

        c.OnRequest(func(r *colly.Request) {
            timer.Reset(2 * time.Second)
            currentLink := r.URL.String()
            urlStore.visited.Store(currentLink, true)
            urlStore.numVisited++
            if currentLink == dest {
                urlStore.resultPath = getPath(urlStore.predecessors, dest)
                cancel()
            }
        })

        c.OnHTML("div#mw-content-text "+"a[href]", func(e *colly.HTMLElement) {
            currentLink := e.Request.URL.String()
            neighborLink, _ := url.QueryUnescape(e.Attr("href"))
            if validLink(neighborLink) {
                mutex.Lock()
                if urlStore.predecessors[e.Request.AbsoluteURL(neighborLink)] == "" && e.Request.AbsoluteURL(neighborLink) != src {
                    urlStore.predecessors[e.Request.AbsoluteURL(neighborLink)] = currentLink
                }
                mutex.Unlock()
                e.Request.Visit(e.Request.AbsoluteURL(neighborLink))
            }
        })
    }
}

```

Gambar 4.1.2.8 Kode DLS

```

        c.Visit(src)
    }

    // execute scraper
    go scraper()

    // timer watcher function
    go func() {
        <-timer.C
        noVisits <- struct{}(){}
    }()

    select {
    case <-ctx.Done():
        return urlStore
    case <-noVisits:
        return urlStore
    }
}

```

Gambar 4.1.2.9 Kode DLS (2)

```

func DLSMulti(src string, dest string, maxDepth int) *URLStore {
    // initialize new URLStore
    urlStore := NewURLStore()
    found := false

    // concurrency tools
    var mutex sync.Mutex

    // initialize colly config
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
        colly.MaxDepth(maxDepth),
        colly.Async(true),
    )

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

    c.OnRequest(func(r *colly.Request) {
        currentLink := r.URL.String()
        urlStore.visited.Store(currentLink, true)
        urlStore.numVisited++
        if currentLink == dest {
            found = true
        }
    })

    c.OnHTML("table.infobox "+"a[href]", func(e *colly.HTMLElement) {
        currentLink := e.Request.URL.String()
        neighborLink, _ := url.QueryUnescape(e.Attr("href"))
        if validLink(neighborLink) {
            mutex.Lock()
            if _, ok := urlStore.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)]; ok && !stringInSlice(currentLink, urlStore.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)]) {
                urlStore.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)] = append(urlStore.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)], currentLink)
            } else if ok {
                urlStore.predecessorsMulti[e.Request.AbsoluteURL(neighborLink)] = []string(currentLink)
            }
            mutex.Unlock()
        }
        e.Request.Visit(e.Request.AbsoluteURL(neighborLink))
    })
}

```

Gambar 4.1.2.10 Kode DLSMulti

```

    c.Visit(src)

    c.Wait() // wait until all nodes are done being visited

    if found {
        urlStore.resultPaths = getPaths(urlStore.predecessorsMulti, src, dest)
    }

    return urlStore
}

```

Gambar 4.1.2.11 Kode DLSMulti (2)

Kode DLS dan DLSMulti keduanya akan melakukan *scraping* secara *depth limited search* terhadap artikel *src*. Kode tersebut akan mengembalikan *path* dari *src* ke *dest* jika *dest* dilalui dan mengembalikan kosong jika tidak. Perbedaan kedua kode tersebut adalah DLS mengembalikan solusi tunggal dan DLSMulti mengembalikan semua solusi.

```

func IDS(src string, dest string) *URLStore {
    depth := 1
    urlStore := NewURLStore()
    for {
        urlStore = DLS(src, dest, depth)
        if len(urlStore.resultPath) > 0 { // solution not found, increase depth
            break
        }
        depth++
    }
    return urlStore
}

func IDSMulti(src string, dest string) *URLStore {
    depth := 1
    urlStore := NewURLStore()
    for {
        urlStore = DLSMulti(src, dest, depth)
        if len(urlStore.resultPaths) > 0 { // solution not found, increase depth
            break
        }
        depth++
    }
    return urlStore
}

```

Gambar 4.1.2.12 Kode IDS dan IDSMulti

Kedua kode tersebut akan melakukan DLS secara iteratif dengan menambah nilai *depth* hingga *dest* ditemukan. Kode tersebut akan mengembalikan jalur/jalur-jalur dari *src* ke *dest* serta jumlah artikel yang dilalui.

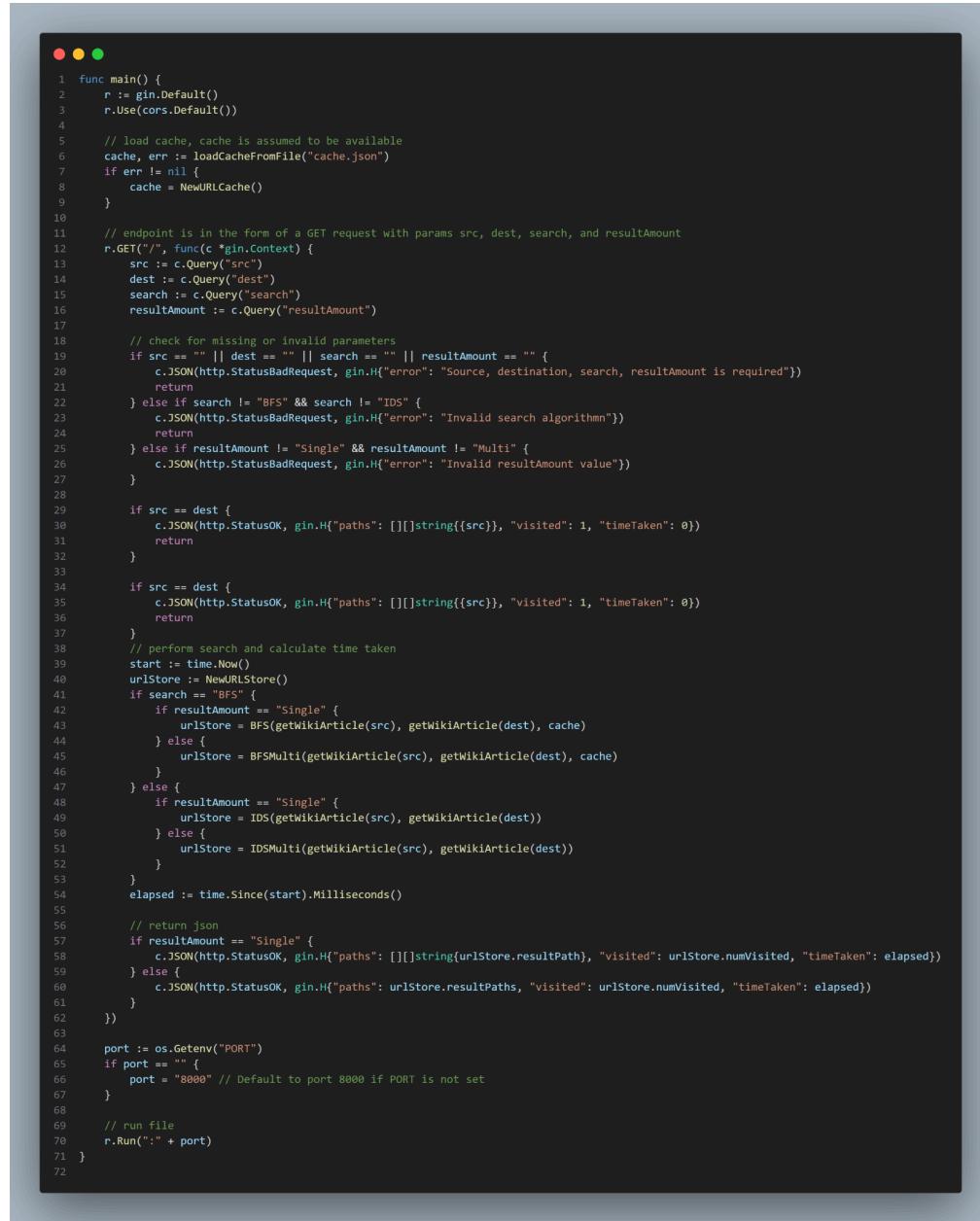
```
func saveCacheToFile(cache *URLCache, filename string) error {
    data, err := json.MarshalIndent(cache, "", " ")
    if err != nil {
        return err
    }
    return os.WriteFile(filename, data, 0644)
}

func loadCacheFromFile(filename string) (*URLCache, error) {
    data, err := os.ReadFile(filename)
    if err != nil {
        return nil, err
    }
    var cache URLCache
    if err := json.Unmarshal(data, &cache); err != nil {
        return nil, err
    }
    return &cache, nil
}
```

Gambar 4.1.2.1.13 Kode File I/O untuk Cache

Kode `loadCacheFromFile` dan `saveCacheToFile` masing-masing akan membaca *cache* dari dan menyimpan *cache* ke suatu file.

4.1.3. Prosedur



```

1 func main() {
2     r := gin.Default()
3     r.Use(cors.Default())
4
5     // load cache, cache is assumed to be available
6     cache, err := loadCacheFromFile("cache.json")
7     if err != nil {
8         cache = NewURLCache()
9     }
10
11    // endpoint is in the form of a GET request with params src, dest, search, and resultAmount
12    r.GET("/", func(c *gin.Context) {
13        src := c.Query("src")
14        dest := c.Query("dest")
15        search := c.Query("search")
16        resultAmount := c.Query("resultAmount")
17
18        // check for missing or invalid parameters
19        if src == "" || dest == "" || search == "" || resultAmount == "" {
20            c.JSON(http.StatusBadRequest, gin.H{"error": "Source, destination, search, resultAmount is required"})
21            return
22        } else if search != "BFS" && search != "IDS" {
23            c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid search algorithm"})
24            return
25        } else if resultAmount != "Single" && resultAmount != "Multi" {
26            c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid resultAmount value"})
27        }
28
29        if src == dest {
30            c.JSON(http.StatusOK, gin.H{"paths": [][]string{{src}}, "visited": 1, "timeTaken": 0})
31            return
32        }
33
34        if src == dest {
35            c.JSON(http.StatusOK, gin.H{"paths": [][]string{{src}}, "visited": 1, "timeTaken": 0})
36            return
37        }
38        // perform search and calculate time taken
39        start := time.Now()
40        urlStore := NewURLStore()
41        if search == "BFS" {
42            if resultAmount == "Single" {
43                urlStore = BFS(getWikiArticle(src), getWikiArticle(dest), cache)
44            } else {
45                urlStore = BFSMulti(getWikiArticle(src), getWikiArticle(dest), cache)
46            }
47        } else {
48            if resultAmount == "Single" {
49                urlStore = IDS(getWikiArticle(src), getWikiArticle(dest))
50            } else {
51                urlStore = IDSMulti(getWikiArticle(src), getWikiArticle(dest))
52            }
53        }
54        elapsed := time.Since(start).Milliseconds()
55
56        // return json
57        if resultAmount == "Single" {
58            c.JSON(http.StatusOK, gin.H{"paths": [][]string{urlStore.resultPath}, "visited": urlStore.numVisited, "timeTaken": elapsed})
59        } else {
60            c.JSON(http.StatusOK, gin.H{"paths": urlStore.resultPaths, "visited": urlStore.numVisited, "timeTaken": elapsed})
61        }
62    })
63
64    port := os.Getenv("PORT")
65    if port == "" {
66        port = "8000" // Default to port 8000 if PORT is not set
67    }
68
69    // run file
70    r.Run(": " + port)
71 }
72

```

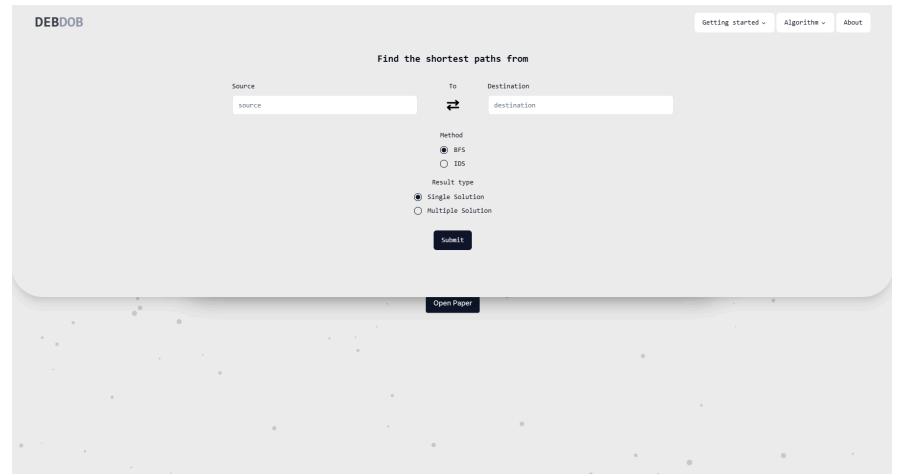
Gambar 4.1.3.1. Kode Fungsi Main

Kode main akan menyalakan suatu *endpoint* untuk menjalankan program. Kode tersebut akan menerima GET *request* pada *endpoint* “/”. Kode tersebut akan memvalidasi keberadaan dan kebenaran atribut *src*, *dest*, *search*, dan *resultAmount* pada parameter *query*. Selanjutnya, kode tersebut akan melakukan pencarian dari *src* ke *dest* menggunakan metode yang ditentukan oleh *search* dan jumlah hasil yang

ditentukan oleh *resultAmount*. Hasil *response* dari API tersebut meliputi jalur dari *src* ke *dest*, jumlah artikel yang diperiksa, dan waktu yang diperlukan.

4.2. Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).

Fitur-Fitur yang disediakan program

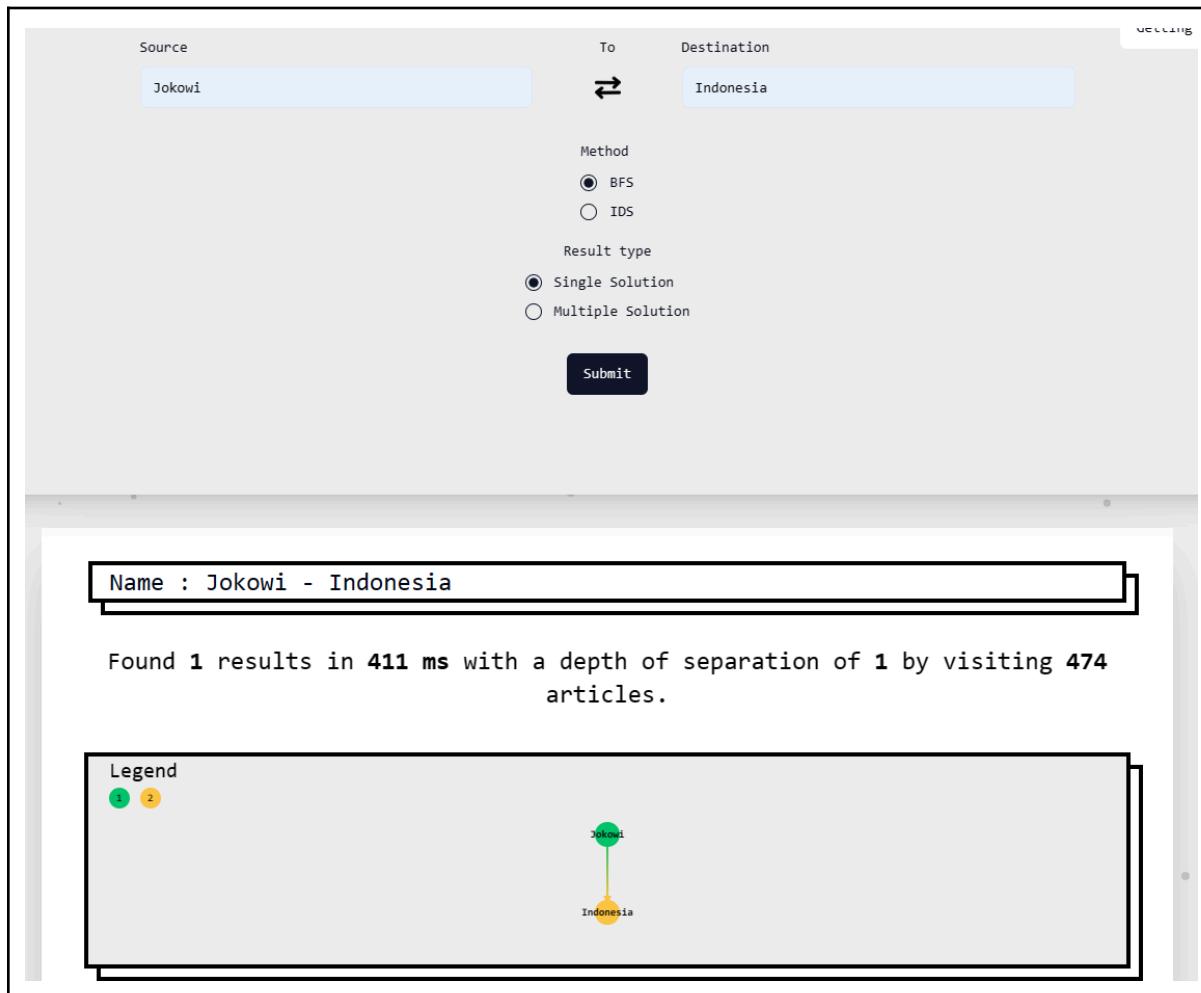
Fitur	Screenshot
<p>Landing Page</p> <p>Sebagai halaman pertama saat aplikasi dijalankan</p>	
<p>Start Page</p> <p>Sebagai halaman yang memuat fitur utama yaitu pencarian paths dengan algoritma BFS dan IDS.</p>	

<h3>Docs Page</h3> <p>Sebagai halaman dokumentasi terkait aplikasi web yang dibuat</p>	<p>Ringkasan Pengembangan Aplikasi Website</p> <p>Bagian 1: Front-end Development</p> <p>Deskripsi: Berfokus pada elemen yang dilihat dan berinteraksi dengan pengguna akhir.</p> <p>Tugas Utama:</p> <ul style="list-style-type: none"> Membuat UI (User Interface) yang menarik dan mudah dinavigasi. Mengoptimalkan UX (User Experience) untuk interaksi yang intuitif. Menghubungkan antarükma dengan data back-end. Menastasikan responsivitas di berbagai perangkat. <p>Teknologi Utama:</p> <ul style="list-style-type: none"> HTML (HyperText Markup Language) CSS (Cascading Style Sheets) JavaScript Framework dan Library Front-end (React, Angular, Vue.js, jQuery) Alat Pengembangan dan Build (Webpack, Gulp, npm, Git) <p>Bagian 2: Back-end Development</p> <p>Deskripsi: Berfokus pada apa yang terjadi di sisi server.</p> <p>Tugas Utama:</p> <ul style="list-style-type: none"> Merancang dan mengelola database. Mengembangkan logika bisnis. Mengimplementasikan API. Mengelola keamanan. Mengoptimalkan kinerja. <p>Teknologi Utama:</p> <ul style="list-style-type: none"> Bahasa Pengembangan Server-Side (Java, Python, Ruby, PHP, Node.js) Framework dan Platform Server-Side (Django, Ruby on Rails, Spring Boot, Express) Database (MySQL, PostgreSQL, MongoDB, Redis) API dan Integrasi (REST, GraphQL, gRPC) Alat Pengembangan dan Deployment (Docker, Kubernetes, Git) <p>Bagian 3: Dockerization</p> <p>Deskripsi: Proses mengemas aplikasi dan dependensinya ke dalam container.</p> <p>Manifest:</p> <ul style="list-style-type: none"> Portabilitas: Aplikasi dapat dijalankan di mana saja tanpa masalah kompatibilitas. Konsistensi: Aplikasi berjalan dengan cara yang sama di semua lingkungan. Skalabilitas: Container dapat dengan mudah di-scale up atau down untuk memenuhi kebutuhan. <p>Technologies Used</p> <ul style="list-style-type: none"> Golang - version 1.22.2 Colly - version 1.2.0 Newt.js - version 14.1.4 D3.js - version 7.9.0 Dockerfile - version 3.23.0 Particleground - version 2.0.0 TailwindCSS Shadcn UI Library
<h3>BFS Page</h3> <p>Sebagai halaman dokumentasi terkait algoritma BFS</p>	<p>Breadth First Search (BFS) Algorithm</p> <p>Algoritma Breadth First Search (BFS) adalah teknik eksplorasi dan pencarian yang menggunakan pendekatan "lebar". Untuk menjelajahi semua simpul pada tingkat tertentu sebelum melanjutkan ke tingkat berikutnya. BFS adalah salah satu algoritme dasar dalam ilmu komputer dan digunakan secara luas untuk berbagai aplikasi, terutama dalam konteks graf dan pohon. BFS memulai eksplorasi dari simpul awal (sering disebut sebagai simpul sumber atau root), kemudian mengunjungi simpul-simpul tetangga terlebih dahulu sebelum bergeser ke simpul pada tingkat berikutnya. Algoritma ini menggunakan antrian (queue) untuk menyimpan urutan simpul yang akan dikunjungi.</p> <p>Steps of BFS:</p> <ol style="list-style-type: none"> Inisialisasi (Initialization): Simpul awal dimasukkan ke dalam antrian dan ditandai sebagai simpul sudah dikunjungi. Proses iteratif (Queue Processing): Sementara antrian tidak kosong, lakukan langkah-langkah berulang: <ol style="list-style-type: none"> Eksplorasi Simpul Tetangga (Checking Neighbor Nodes): Jika bukan solusi, tambahkan semua simpul tetangga yang belum dikunjungi ke dalam antrian dan tandai sebagai sudah dikunjungi. Pengulangan (Iteration): Lanjutkan proses ini sampai antrian kosong atau solusi ditemukan. <p>Kelebihan dan Kekurangan Breadth First Search (BFS)</p> <p>Kelebihan BFS:</p> <ul style="list-style-type: none"> Tidak Akan Memenuhi Jalan Buntu: BFS selalu menjelajahi semua simpul pada tingkat tertentu sebelum bergeser ke tingkat berikutnya, sehingga tidak akan pernah terjebak pada jalur buntu, meskipun itu tidak ada solusi yang ditentukan. Menemukan Solusi Terpendek (Jika Ada): Jika solusi tidak berbobot, BFS selalu menemukan jalur terpendek ke tujuan karena mengunjungi simpul berdasarkan urutan tingkat. Dapat Menemukan Lebih dari Satu Solusi: Jika ada beberapa solusi, BFS akan menemukan semua solusi dan memberikan solusi minimum (dengan tingkat terendah). <p>Kekurangan BFS:</p> <ul style="list-style-type: none"> Membutuhkan Memori Besar: Karena BFS harus menyimpan semua simpul pada tingkat tertentu, ia bisa memakan banyak memori, terutama jika graf sangat lebar atau memiliki banyak simpul. Lama untuk Pencarian Solusi yang Relatif Ada dalam Kedalaman yang Dalam: BFS menguji semua simpul pada tingkat tertentu sebelum melanjutkan ke tingkat berikutnya. Jika solusi berada pada tingkat yang lebih dalam, BFS akan menghabiskan waktu yang cukup lama untuk mencapainya.
<h3>IDS Page</h3> <p>Sebagai halaman dokumentasi terkait algoritma IDS</p>	<p>Iterative Deepening Search (IDS) Algorithm</p> <p>Iterative Deepening Search (IDS) adalah algoritma pencarian yang menggunakan manfaat dari Breadth-First Search (BFS) dan Depth-First Search (DFS). IDS digunakan untuk menjelajahi graf atau pohon dengan pendekatan "ber tahap dalam", yang secara bertahap meningkatkan kedalaman pencarian sampai solusi ditemukan atau semua simpul telah dieksplorasi.</p> <p>IDS dilakukan dengan kedalaman nol dan meningkatannya satu tingkat setiap iterasi, mirip dengan DFS dalam hal pendekatan kedalaman, tetapi dengan keuntungan tambahan seperti BFS. Algoritma ini menggunakan teknik rekursi untuk menyelesaikan masalah yang kompleks.</p> <p>Langkah-langkah IDS:</p> <ol style="list-style-type: none"> Inisialisasi: Tentukan kedalaman maksimum untuk setiap iterasi, biasanya dimulai dari nol. Pencarian Recursif: Mulai dari simpul awal, lakukan pencarian dalam lingkup batas kedalaman saat ini (tidak ada lagi anak bila dituliskan). Periksa Solusi: Jika simpul tersebut adalah solusi, hentikan pencarian dan kembalikan hasilnya. Lanjutkan ke Simpul Tetangga: Jika bukan solusi, lanjutkan ke simpul tetangga, tetapi hanya sampai batas kedalaman saat ini. Tersegi: Jika semua simpul pada kedalaman saat ini telah dijalankan tanpa menemukan solusi, tingkatkan batas kedalaman dan ulangi pencarian. <p>Kelebihan dan Kekurangan Iterative Deepening Search (IDS)</p> <p>Kelebihan IDS:</p> <ul style="list-style-type: none"> Rebutuhkan Memori Lebih Kecil: IDS membutuhkan lebih sedikit memori dibandingkan BFS karena hanya menyimpan simpul dalam jalur saat ini, bukan semua simpul pada tingkat tertentu. Menemukan Solusi Terpendek (Jika Ada): Seperti BFS, IDS selalu menemukan solusi dengan kedalaman terendah karena pencarian dilakukan bertahap. Effisien dalam Pencarian di Graf Lurus dan Dalam: IDS lebih cocok untuk graf yang sangat dalam karena tidak perlu menyimpan semua simpul di setiap tingkat. <p>Kekurangan IDS:</p> <ul style="list-style-type: none"> Pengulangan Pencarian: Karena IDS memulai ulang pencarian dari awal setiap kali kedalaman ditingkatkan, ada banyak pengulangan dalam eksplorasi simpul yang sama. Kurang Efisien untuk Graf Dangkal: Untuk graf yang relatif dangkal, IDS bisa menjadi kurang efisien karena pengulangan pencarian.

<p>About Page</p> <p>Sebagai halaman informasi tim pengembang aplikasi</p>	<p>DEBDOD</p> <p>About Us</p> <table border="1"><thead><tr><th>Name</th><th>ID</th></tr></thead><tbody><tr><td>Nue1</td><td>Name : Imanuel Sebastian Girsang NDM : 135220958</td></tr><tr><td>Dab</td><td>Name : Ahmad Mudabbir Arif NDM : 13522072</td></tr><tr><td>Neo</td><td>Name : Muhammad Neo Cicero Koda NDM : 13522108</td></tr></tbody></table>	Name	ID	Nue1	Name : Imanuel Sebastian Girsang NDM : 135220958	Dab	Name : Ahmad Mudabbir Arif NDM : 13522072	Neo	Name : Muhammad Neo Cicero Koda NDM : 13522108
Name	ID								
Nue1	Name : Imanuel Sebastian Girsang NDM : 135220958								
Dab	Name : Ahmad Mudabbir Arif NDM : 13522072								
Neo	Name : Muhammad Neo Cicero Koda NDM : 13522108								

4.3. Hasil pengujian (screenshot antarmuka dan skenario yang memperlihatkan berbagai kasus yang mencakup seluruh fitur pada aplikasi Anda)

Hasil Pengujian Kedalaman 1 Single Solution



Source: Jokowi, To: Indonesia

Method: IDS (selected)

Result type: Single Solution (selected)

Submit

Name : Jokowi - Indonesia

Found 1 results in 2696 ms with a depth of separation of 1 by visiting 473 articles.

Legend: 1 (green), 2 (yellow)

Diagram: A graph showing a single edge connecting a green node labeled "Jokowi" to a yellow node labeled "Indonesia".

Hasil	BFS	IDS
Waktu Eksekusi	411 ms	2696 ms
Artikel Dikunjungi	474 artikel	473 artikel

Hasil Pengujian Kedalaman 2 Single Solution

Method
 BFS
 IDS

Result type
 Single Solution
 Multiple Solution

Submit

Name : Chicken - Duck

Found 1 results in 4367 ms with a depth of separation of 2 by visiting 511 articles.

Legend
1 2 3

```
graph TD; Chicken((Chicken)) --- Bird((Bird)); Bird --- Duck((Duck))
```

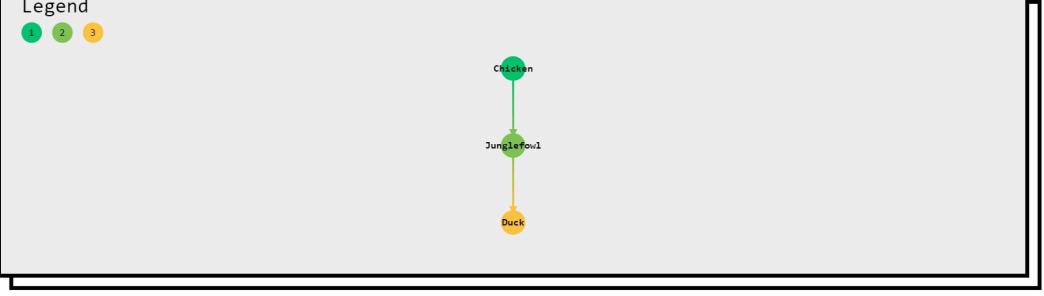
Method
 BFS
 IDS

Result type
 Single Solution
 Multiple Solution

Name : Chicken - Duck

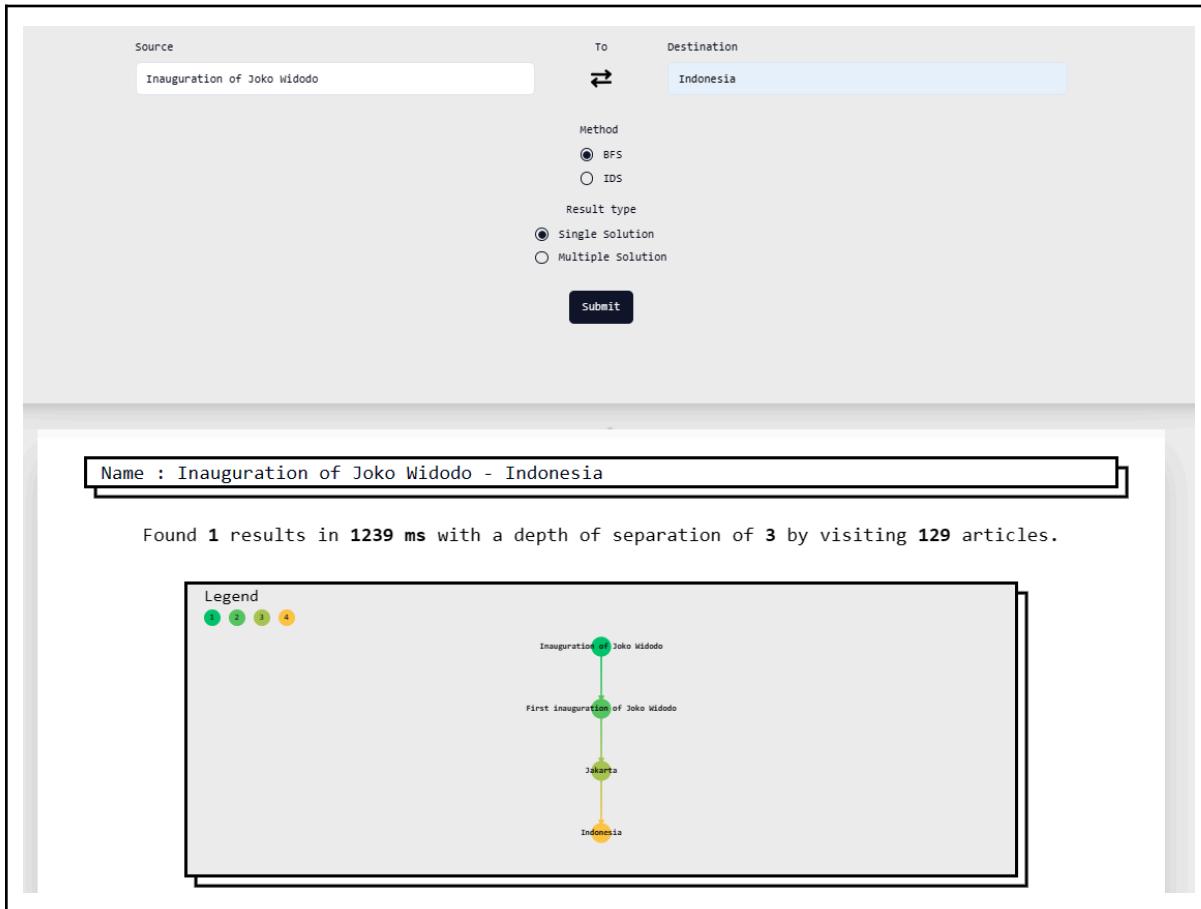
Found 1 results in **6581 ms** with a depth of separation of **2** by visiting **886** articles.

Legend
● 1
● 2
● 3



Hasil	BFS	IDS
Waktu Eksekusi	4367 ms	6581 ms
Artikel Dikunjungi	511 artikel	886 artikel

Hasil Pengujian Kedalaman 3 Single Solution



Method
 BFS
 IDS

Result type
 Single Solution
 Multiple Solution

Submit

Name : Vario - Indonesia

Found 1 results in **13857 ms** with a depth of separation of **3** by visiting **8476** articles.

Legend
● 1
● 2
● 3
● 4

```
graph TD; Vario((Vario)) --- Varionometer((Varionometer)); Varionometer --- Paragliding((Paragliding)); Paragliding --- Indonesia((Indonesia))
```


Hasil	BFS	IDS
Waktu Eksekusi	1239 ms	13587 ms
Artikel Dikunjungi	129 artikel	8476 artikel

Hasil Pengujian Kedalaman 2 Multiple Solution

Method
 BFS
 IDS

Result type
 Single Solution
 Multiple Solution

Name : Chicken - Duck

Found **21** results in **9166 ms** with a depth of separation of **2** by visiting **512** articles.

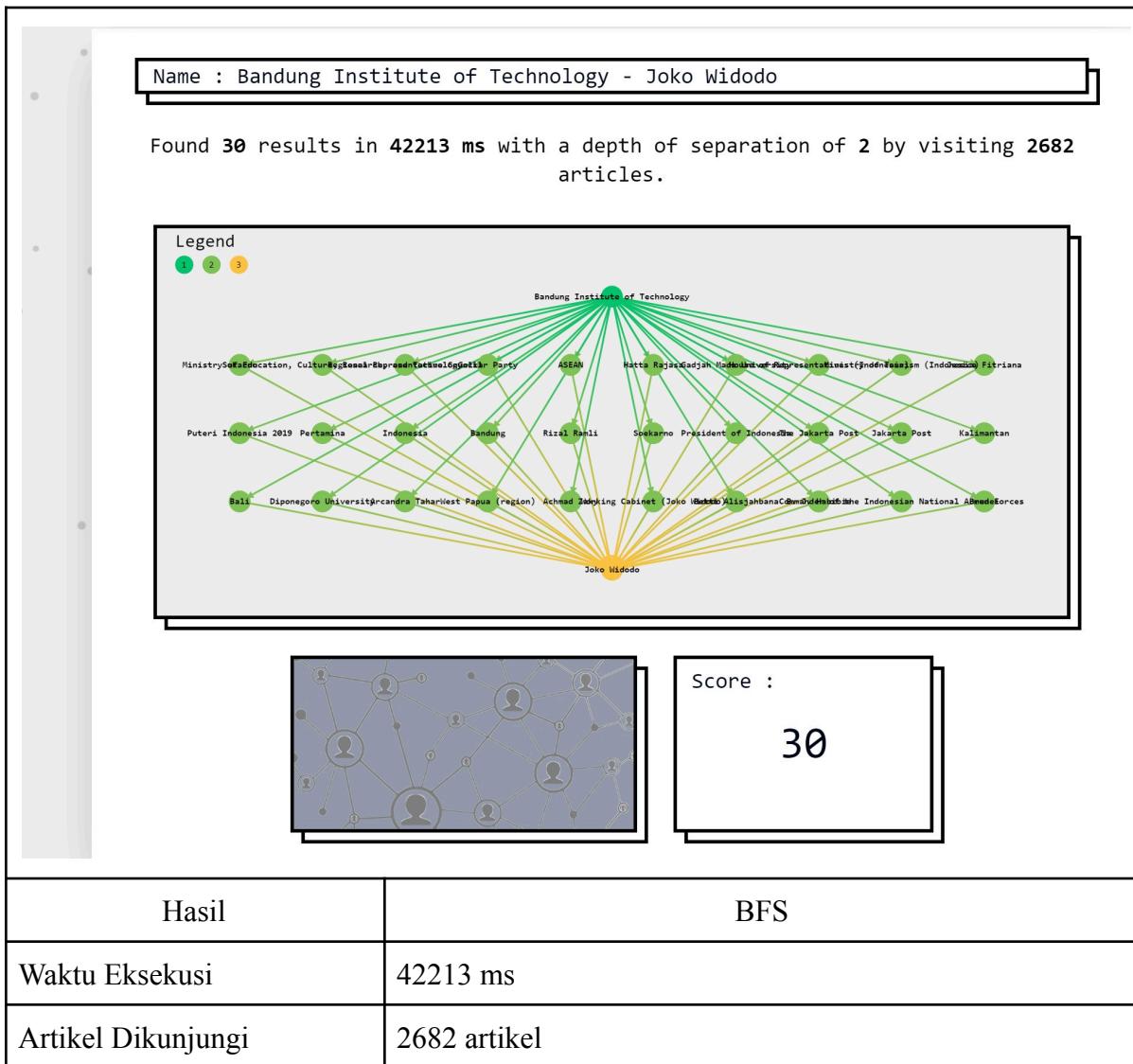
Legend

```

graph TD
    Chicken --- Bird
    Chicken --- Domestication
    Chicken --- EggAsFood
    Chicken --- Galliformes
    Chicken --- EditionOfSystemaNaturae
    Chicken --- Incubation
    Chicken --- BirdFlight
    Chicken --- Chick
    Chicken --- Phasianidae
    Chicken --- Closca
    Chicken --- PoultryDisease
    Chicken --- Parasitism
    Chicken --- Junglefowl
    Chicken --- CrueltyToAnimals
    Chicken --- Dinosaur
    Chicken --- Poultry
    Chicken --- Debeaking
    Chicken --- ChickenDisease
    Chicken --- Psittacosis
    Chicken --- DomesticTurkey
    Duck --- Bird
    Duck --- Domestication
    Duck --- EggAsFood
    Duck --- Galliformes
    Duck --- EditionOfSystemaNaturae
    Duck --- Incubation
    Duck --- BirdFlight
    Duck --- Chick
    Duck --- Phasianidae
    Duck --- Closca
    Duck --- PoultryDisease
    Duck --- Parasitism
    Duck --- Junglefowl
    Duck --- CrueltyToAnimals
    Duck --- Dinosaur
    Duck --- Poultry
    Duck --- Debeaking
    Duck --- ChickenDisease
    Duck --- Psittacosis
    Duck --- DomesticTurkey
  
```

Hasil	BFS
Waktu Eksekusi	9166 ms
Artikel Dikunjungi	512 artikel

Hasil Pengujian Kedalaman 2 Multiple Solution



4.4. Analisis hasil pengujian

Berdasarkan hasil uji coba pada bab sebelumnya, dapat dilihat bahwa algoritma BFS cenderung memiliki waktu eksekusi yang lebih cepat dibandingkan dengan IDS pada sebagian besar kasus. Hal ini dapat dipahami karena BFS melakukan eksplorasi secara luas (breadth-first) dan mencari solusi dengan cara mengunjungi simpul-simpul secara bertahap berdasarkan tingkat kedalaman, sedangkan IDS menerapkan pendekatan depth-first dengan batas kedalaman tertentu. Walaupun secara notasi Big O keduanya memiliki kompleksitas waktu $O(b^d)$, namun jelas $T(N)$

keduanya jauh berbeda dengan BFS yang maksimal hanya akan mengunjungi seluruh simpul yang tersedia, sedangkan IDS akan menambah beberapa langkah pada setiap iterasi karena harus dimulai dari awal kembali.

Tujuan utama dari IDS adalah untuk menghemat penggunaan memori dengan menjalankan pencarian secara bertahap hingga batas kedalaman yang ditentukan. Namun, ini berarti IDS akan mengulang pencarian dari simpul awal setiap kali mencapai batas kedalaman, yang secara tidak langsung memperpanjang waktu eksekusi. Oleh karena itu, jika dibandingkan dengan BFS yang secara konsisten menjelajahi setiap tingkat dengan sekali jalan, IDS sering kali memerlukan waktu lebih lama karena proses pengulangan ini.

Namun, ada beberapa kasus di mana IDS bisa menjadi lebih cepat dibandingkan dengan BFS. Hal ini bisa terjadi karena penggunaan multithreading yang memungkinkan IDS melakukan eksplorasi secara paralel, sehingga simpul hasil mungkin ditemukan lebih cepat dibandingkan dengan eksplorasi BFS yang berurutan. Selain itu, faktor eksternal seperti kecepatan internet juga dapat memengaruhi waktu scraping saat mengambil data dari situs web, dan ini dapat membuat waktu eksekusi bervariasi.

Kesimpulannya, meskipun BFS biasanya memiliki keunggulan dalam hal kecepatan eksekusi karena eksplorasi secara menyeluruh, IDS mungkin unggul dalam situasi tertentu yang memanfaatkan multithreading dan kondisi khusus lainnya yang dapat memengaruhi hasil, serta adanya penghematan memori.

BAB V

SIMPULAN DAN SARAN

5.1. Simpulan

Dalam pengerjaan Tugas Besar II IF2211 Strategi Algoritma, kami telah mengeksplorasi secara mendalam implementasi algoritma BFS dan IDS untuk permainan wikiRace, yang menjadi topik tugas besar ini. Selain itu, kami juga mengeksplorasi berbagai aspek terkait, seperti teknik *scraping*, *multithreading*, *caching*, Docker, serta pengembangan situs web dan API menggunakan bahasa Go.

Dalam proses eksplorasi, kami menemukan bahwa banyak tautan di Wikipedia yang bisa dianggap "tidak penting" dan beberapa di antaranya tidak terlihat dengan jelas. Hal ini perlu menjadi perhatian khusus saat melakukan *scraping* Wikipedia agar tautan-tautan yang didapat dapat disaring dengan tepat.

Dari hasil percobaan menggunakan jaringan internet yang stabil, kami menemukan bahwa algoritma BFS relatif lebih cepat dalam menemukan rute terpendek dibandingkan dengan IDS. Ini masuk akal karena IDS selalu memulai pencarian dari simpul awal untuk menghemat ruang memori. Namun, karena penggunaan *multithreading* dan faktor koneksi internet yang menjadi variabel penting dalam proses pencarian, hasil ini bisa bervariasi.

Meskipun demikian, kami yakin bahwa BFS adalah strategi pencarian terbaik untuk permainan wikiRace karena pohon pencarian relatif tidak terlalu besar untuk sebagian besar halaman Wikipedia. Oleh karena itu, keuntungan dalam penggunaan memori yang dimiliki oleh IDS kurang signifikan dibandingkan dengan kerugian karena harus mengulang pencarian dari simpul awal.

5.2. Saran dan Refleksi

1. Kami menemukan bahwa *multithreading* dapat menyebabkan terlalu banyak permintaan yang dilakukan dalam waktu bersamaan dan terblokirnya program. Oleh karena itu, sebaiknya kami lebih berhati-hati dalam menggunakan multithreading agar hal tersebut tidak terjadi. Pertimbangkan batasan pada jumlah permintaan dan sesuaikan dengan kebijakan situs yang Anda tuju untuk menghindari pemblokiran.

2. Kami dapat melakukan eksplorasi lebih jauh fitur-fitur lain yang dapat diimplementasikan di situs web, serta strategi visualisasi graf menggunakan pustaka yang lebih canggih untuk mempermudah dan memperindah visualisasi data.
3. Kami merasa bahwa strategi *caching* yang sekarang diimplementasikan masih dapat diperbaiki. Hal tersebut dapat dilakukan dengan merancang strategi *caching* yang lebih efisien, atau buat database yang lebih maju untuk mengoptimalkan performa program. Ini penting karena saat ini program sangat bergantung pada kualitas dan stabilitas koneksi internet. Dengan strategi *caching* atau *database* yang baik, program dapat bekerja lebih cepat dan lebih andal, meskipun koneksi internet tidak selalu stabil.

LAMPIRAN

Repositori Github

https://github.com/ImmanuelSG/Tubes2_FE_debdob

https://github.com/neokoda/Tubes2_BE_DebDob

Link Video Tugas Besar II Strategi Algoritma

<https://youtu.be/ff997QOMAhY?si=ZP8DnQ9aoV9bGdDF>

DAFTAR PUSTAKA

- GeeksforGeeks, "Breadth First Search or BFS for a Graph," [Online]. Available: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>. [Accessed: Apr. 27, 2024].
- GeeksforGeeks, "Containerization using Docker," [Online]. Available: <https://www.geeksforgeeks.org/containerization-using-docker/>. [Accessed: Apr. 27, 2024].
- R. Munir, "Breadth/Depth First Search (BFS/DFS) Bagian 2," IF2211 Strategi Algoritma, 2024. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>. [Accessed: Mar. 9, 2024].
- R. Munir, "Breadth/Depth First Search (BFS/DFS) Bagian 2," IF2211 Strategi Algoritma, 2024. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>. [Accessed: Apr. 27, 2024].
- OpenGenus, "Iterative Deepening Search," [Online]. Available: <https://iq.opengenus.org/iterative-deepening-search/>. [Accessed: Apr. 27, 2024].

