

TUGAS BESAR 2
IF2211 STRATEGI ALGORITMA
Pemanfaatan Algoritma IDS dan BFS dalam Permainan
WikiRace



DISUSUN OLEH:

Aland Mulia Pratama	13522124
Rizqika Mulia Pratama	13522126
Ikhwan Al Hakim	13522147

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

DAFTAR ISI

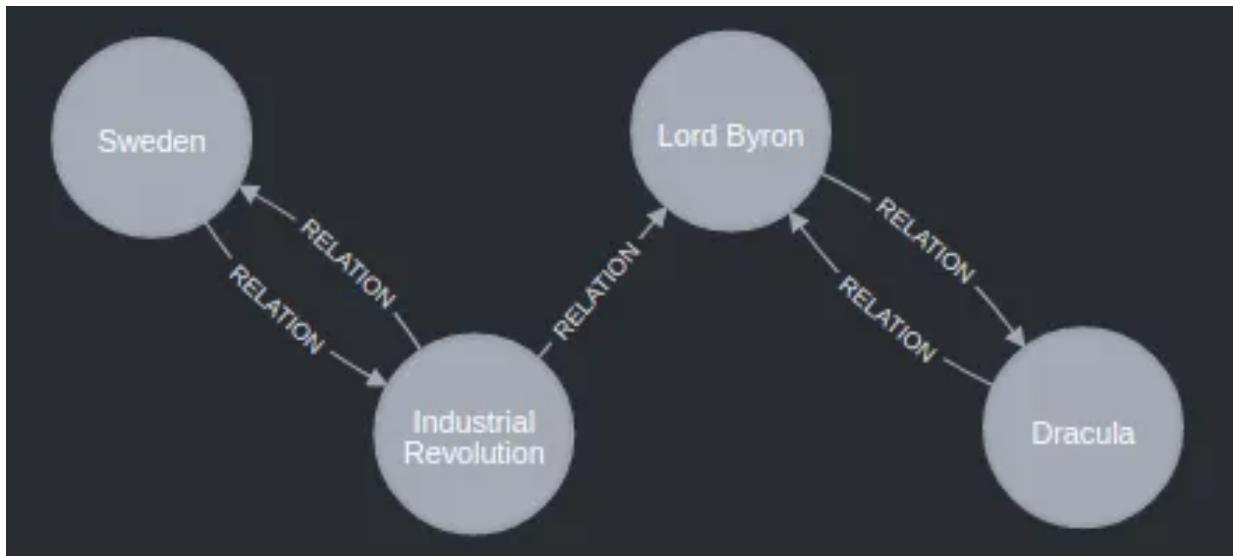
DESKRIPSI TUGAS.....	2
1.1 Latar Belakang.....	2
1.2 Deskripsi Tugas.....	3
1.3 Spesifikasi Program.....	5
LANDASAN TEORI.....	6
2.1 Algoritma Breadth-First Search (BFS).....	6
2.2 Algoritma Iterative Deepening Search (IDS).....	6
2.3 Website Development.....	6
2.3.1 Backend Development.....	6
2.3.2 Frontend Development.....	8
ANALISIS PEMECAHAN MASALAH.....	13
3.1 Langkah-Langkah Pemecahan Masalah.....	13
3.1.1 Algoritma BFS.....	13
3.1.2 Algoritma IDS.....	13
3.2 Mapping Persoalan Menjadi Elemen BFS dan IDS.....	13
3.3 Contoh Ilustrasi Kasus.....	14
IMPLEMENTASI DAN PENGUJIAN.....	14
4.1 Implementasi Program.....	14
4.1.1 BFS.....	14
4.1.2 IDS.....	18
4.2 Struktur Data dan Spesifikasi Program.....	22
4.1.1 BFS.....	22
4.1.2 IDS.....	23
4.3 Tata Cara Penggunaan Program.....	24
4.4 Hasil Pengujian.....	27
4.5 Analisis Algoritma.....	40
4.5.1 Algoritma BFS.....	41
4.5.1.1 Kelebihan.....	41
4.5.1.2 Kekurangan.....	41
4.5.2 Algoritma IFS.....	41
4.5.2.1 Kelebihan.....	41
4.5.2.2 Kekurangan.....	41
KESIMPULAN DAN SARAN.....	42
5.1 Kesimpulan.....	42
5.2 Saran.....	42
DAFTAR PUSTAKA.....	43

BAB I

DESKRIPSI TUGAS

1.1 Latar Belakang

WikiRace atau Wiki Game adalah sebuah permainan yang mengambil inspirasi dari struktur Wikipedia, sebuah ensiklopedia daring yang dikelola secara kolaboratif oleh ribuan relawan di seluruh dunia. Permainan ini menantang pemain untuk menavigasi melalui jaringan artikel-artikel Wikipedia dengan tujuan mencapai suatu artikel yang ditentukan sebelumnya dalam waktu sesingkat mungkin atau dengan menggunakan sejumlah klik yang sekecil mungkin.



Gambar 1. Ilustrasi Graf WikiRace

(Sumber: https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZsIicJCWO.png)

Dalam sebuah program WikiRace, program menerima masukkan sebuah artikel acuan dan kemudian harus mengikuti serangkaian tautan dalam artikel tersebut untuk mencapai artikel tujuan. Program WikiRace sangat mungkin untuk dirancang dan dikembangkan karena kedalaman maksimal dalam artikel acuan menuju artikel tujuan hanya hingga sekitar 6 tautan yang berdasarkan konsep yang dikenal sebagai "Six Degrees of Separation" atau "Enam Derajat Pemisahan".

Teori ini pertama kali diperkenalkan oleh penulis Hongaria, Frigyes Karinthy, dalam cerpen berbahasa Hungaria yang berjudul "Chain-Links" pada tahun 1929. Karinthy mengajukan gagasan bahwa setiap dua orang di dunia ini terhubung melalui serangkaian hubungan individu yang tidak lebih dari enam orang di antara mereka.

Selanjutnya, pada tahun 1967, sosiolog Stanley Milgram melakukan serangkaian eksperimen yang dikenal sebagai "Eksperimen Jalan Kota Kecil" untuk menguji gagasan ini. Dalam eksperimennya, Milgram memberikan amplop kepada sekelompok orang di Amerika Serikat dengan instruksi untuk mengirimnya kepada seseorang yang mereka kenal di Boston, Massachusetts, menggunakan jaringan kenalan mereka. Eksperimen ini memberikan hasil bahwa rata-rata, amplop tersebut hanya membutuhkan sekitar enam langkah atau "derajat pemisahan" sebelum sampai ke penerima yang dituju.

1.2 Deskripsi Tugas

Dalam Tugas Besar 2 Strategi Algoritma kami diminta untuk membuat sebuah website dengan antarmuka pengguna (GUI) yang dapat mengimplementasikan algoritma IDS dan BFS dalam permainan wikirace. Program dapat menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Pada spesifikasi program wajib, program diharuskan untuk mengeluarkan salah satu rute terpendek dari hasil-hasil yang ada. Program 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).

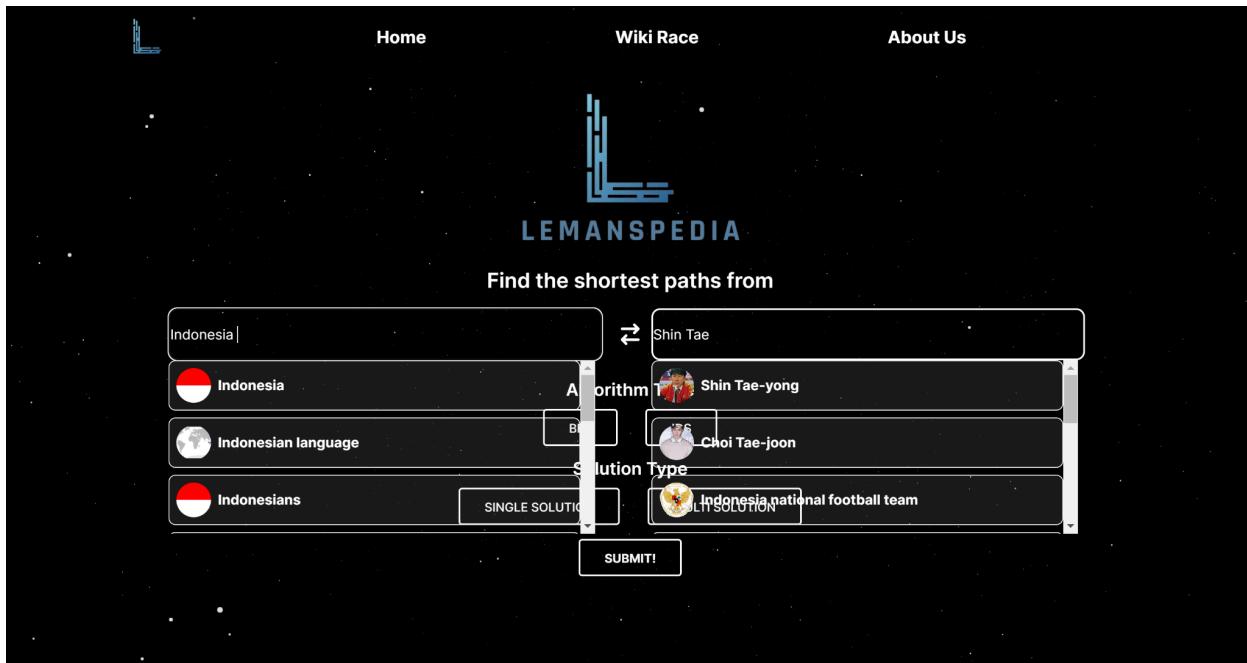
Kolom Input pada Website :

The screenshot shows a dark-themed web application interface. At the top, there is a navigation bar with links for 'Home', 'Wiki Race', and 'About Us'. Below the navigation is a logo consisting of a stylized 'L' shape above the word 'LEMANSPEDIA'. A sub-header reads 'Find the shortest paths from'. Below this, there are two input fields: 'Masukkan alamat awal' (Input starting address) and 'to' (separating word), followed by another input field 'Masukkan alamat akhir' (Input ending address). Underneath these fields is a section titled 'Algorithm Type' with two buttons: 'BFS' and 'IDS'. Below that is a section titled 'Solution Type' with two buttons: 'SINGLE SOLUTION' and 'MULTI SOLUTION'. At the bottom of the form is a single button labeled 'SUBMIT!'. The overall layout is clean and functional.

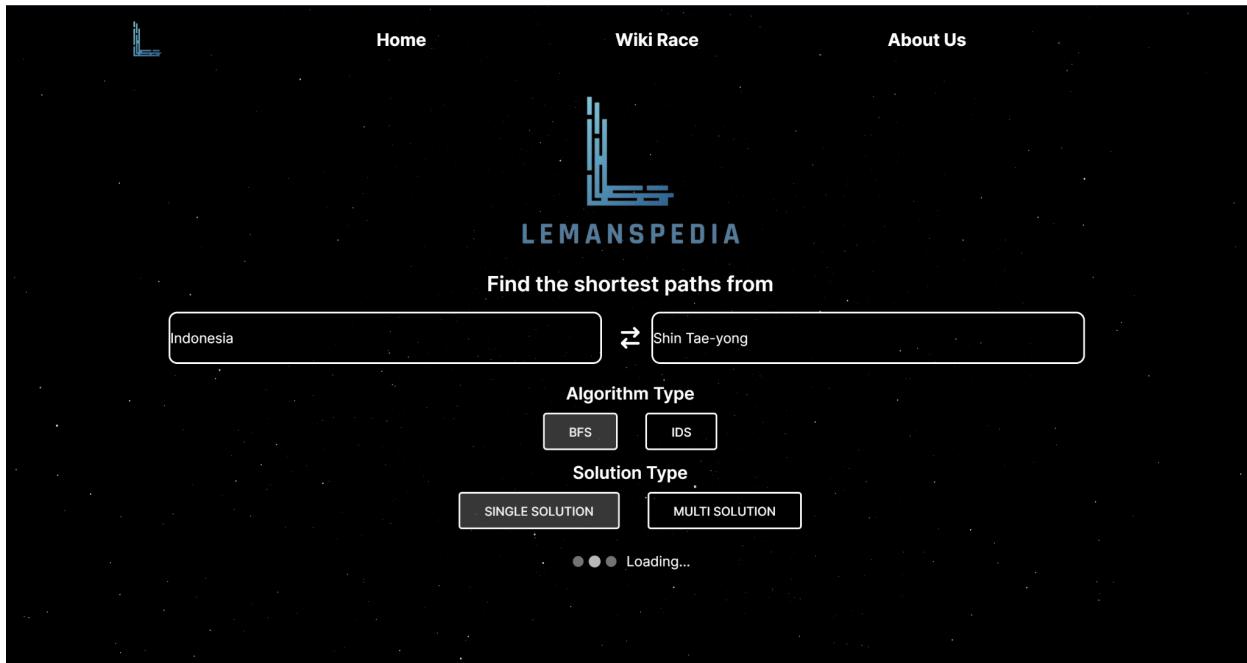
Gambar 1.2.1 Kolom Input pada Website

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Iterative Deepening Search (IDS) dapat ditelusuri rute solusi terpendek dari artikel awal hingga artikel tujuan. Rute solusi merupakan rute terpendek yang dapat ditempuh dari artikel awal hingga artikel tujuan. Rute terpendek yang diperoleh dengan algoritma BFS dan IDS dapat berbeda sehingga menyebabkan juga perbedaan terhadap banyak langkah yang dibutuhkan dalam menghasilkan solusi. Program dapat memberikan keluaran program yaitu seluruh rute terpendek serta visualisasi dari pembangkitan rute terpendek tersebut. Tampilan dari website dan visualisasi dari keluaran program dibebaskan kepada masing-masing kelompok.

Contoh input pada Website :



Gambar 1.2.2 Input pada Website dengan fitur autocomplete



Gambar 1.2.3 Website menerima input valid dari pengguna

Program disimpan dalam repository yang bernama **Tubes2_MarthenGantenk** dengan nama kelompok sesuai dengan pada sheets kelompok. Berikut merupakan struktur dari isi repository tersebut:

- Folder src berisi program yang dapat dijalankan
- Folder doc berisi laporan tugas besar dengan format **MarthenGantenk.pdf**
- README untuk tata cara penggunaan yang minimal berisi:

1. Penjelasan singkat algoritma IDS dan BFS yang diimplementasikan
2. Requirement program dan instalasi tertentu bila ada
3. Command atau langkah-langkah dalam meng-compile atau build program
4. Author (identitas pembuat)

Deskripsi tugas dan latar belakang dari Tugas Besar 2 IF2211 Strategi Algoritma ditulis berdasarkan spesifikasi tugas pada [pranala](#) berikut.

1.3 Spesifikasi Program

Kami kelompok **MarthenGantenk** membuat program Wiki Race berbasis website dengan algoritma Breadth First Search (BFS) dan Iterative Deepening Search (IDS). Program kami menerima input berupa alamat awal dan alamat tujuan artikel wikipedia. Program kami juga menerima input berupa jenis algoritma dan jenis solusi. Pengguna tidak dapat menekan tombol submit saat masukkan belum lengkap. Program WikiRace berbasis website ini kami namai “LemansPedia” yang terinspirasi dari balapan Le Mans selama 24 jam seperti kami senantiasa memikirkan keberjalanan program ini 24/7.



Gambar 1.3.1 Logo LemansPedia

BAB II

LANDASAN TEORI

2.1 Algoritma Breadth-First Search (BFS)

Breadth First Search (BFS) adalah salah satu algoritma pencarian yang digunakan dalam sebuah graf. Algoritma ini dimulai dari simpul awal, kemudian menelusuri semua simpul yang terhubung langsung dengan simpul awal, baru kemudian melanjutkan ke simpul-simpul yang terhubung dengan simpul-simpul tersebut. Dengan menggunakan antrian (queue) untuk menyimpan simpul yang akan diperiksa selanjutnya, BFS memastikan bahwa simpul-simpul yang lebih dekat dengan simpul awal akan diperiksa terlebih dahulu sebelum menjelajahi simpul-simpul yang lebih jauh. Algoritma ini umumnya digunakan untuk mencari jalur terpendek antara dua simpul dalam graf yang terhubung secara tidak terarah, serta dalam penyelesaian berbagai masalah pencarian lainnya. Salah satu kelemahan dari BFS adalah bahwa algoritma ini memerlukan ruang penyimpanan yang cukup besar, terutama jika graf yang akan dijelajahi memiliki banyak simpul.

2.2 Algoritma Iterative Deepening Search (IDS)

Iterative Deepening Search (IDS) atau Iterative Deepening Depth First Search (IDDS) adalah strategi umum yang sering digunakan dalam kombinasi dengan pencarian terbatas kedalaman (depth-limited search) yang menemukan batas kedalaman terbaik. IDS melakukan ini dengan meningkatkan batas secara bertahap, pertama 0, kemudian 1, kemudian 2, dan seterusnya sampai tujuan ditemukan.

IDS menggabungkan efisiensi ruang pencarian depth-first dan pencarian cepat breadth-first (untuk node lebih dekat ke root). IDS memanggil DFS untuk kedalaman yang berbeda mulai dari nilai awal. Dalam setiap panggilan, DFS dibatasi dari melampaui kedalaman yang diberikan. Jadi pada dasarnya kita melakukan DFS dengan cara BFS.

IDS adalah ide yang sangat sederhana, sangat unik, tetapi kontra-intuitif yang tidak ditemukan hingga pertengahan tahun 1970-an. Kemudian banyak orang menemukannya secara bersamaan. Ide ini melakukan DFS terbatas kedalaman secara berulang. Dengan batas kedalaman yang meningkat, tanpa solusi yang ditemukan.

2.3 Website Development

Luaran/output dari program ini adalah sebuah website, dalam pengembangan website ini kami menggunakan beberapa dependency dan juga framework. Pengembangan website dibagi menjadi dua yaitu frontend dan backend. Pada proses implementasi, kami juga memisahkan source code dari backend dan juga frontend yang dijalankan secara terpisah. Pengembangan backend menggunakan bahasa pemrograman GO sedangkan pengembangan frontend menggunakan bahasa pemrograman javascript. Berikut adalah framework atau dependency yang kami gunakan selama pengembangan website Lemanspedia:

2.3.1 Backend Development

- Go

Bahasa Go, juga dikenal sebagai Golang, adalah bahasa pemrograman yang relatif baru dengan sejumlah fitur menarik. Bahasa ini dikembangkan oleh Google dan dirilis pada tahun 2007. Go dirancang untuk menjadi sederhana dan efisien, baik dalam hal sintaks maupun eksekusi. Go memiliki sintaks yang jelas dan mudah dipahami, yang membuatnya menjadi pilihan yang baik untuk pengembangan baru dan berpengalaman.

Salah satu fitur utama Go adalah dukungannya untuk konkurensi. Go memiliki sejumlah fitur tingkat rendah yang sangat baik untuk menangani konkurensi, seperti goroutines dan channels. Ini membuat Go menjadi pilihan yang baik untuk menulis program yang memiliki banyak bagian yang berjalan secara independen, seperti server web.

Go juga memberikan banyak kontrol atas alokasi memori dan telah mengurangi latency secara signifikan. Selain itu, Go memiliki pustaka standar yang sangat baik, yang mencakup banyak fungsi dan struktur data yang umum digunakan. Namun, Go juga memiliki beberapa batasan. Misalnya, Go tidak mendukung OOP dalam arti tradisional. Go tidak memiliki subclassing, sehingga tidak ada inheritance atau metode virtual. Namun, Go mendukung beberapa konsep OOP dalam cara lain.

- **GoQuery**

GoQuery adalah pustaka yang dibuat oleh Martin Angers yang membawa sintaks dan serangkaian fitur yang mirip dengan jQuery ke bahasa Go. Dengan kata lain, GoQuery memungkinkan kita untuk melakukan manipulasi dan traversal dokumen HTML dengan cara yang mirip dengan jQuery.

GoQuery berbasis pada paket net/html Go dan pustaka CSS Selector Cascadia. Paket net/html Go digunakan untuk parsing HTML ke dalam struktur data yang dapat dimanipulasi, dan Cascadia digunakan untuk memilih elemen tertentu dalam struktur data tersebut. Namun, ada beberapa perbedaan penting antara GoQuery dan jQuery. Salah satunya bahwa GoQuery tidak mendukung fungsi manipulasi stateful jQuery seperti height(), css(), dan detach() karena parser net/html mengembalikan node, bukan pohon DOM yang lengkap.

Selain itu, GoQuery memerlukan dokumen sumber dalam pengkodean UTF-8, karena parser net/html juga memerlukannya. Oleh karena itu, tanggung jawab pemanggil untuk memastikan bahwa dokumen sumber menyediakan HTML yang dikodekan UTF-8.

- **Goroutines**

Goroutines adalah elemen penting dalam bahasa Go. Goroutine adalah thread ringan yang dikelola oleh runtime Go. Goroutine memungkinkan fungsi untuk dijalankan secara konkuren, yang memungkinkan pemanfaatan sumber daya sistem secara efisien.

Goroutine dimulai dengan kata kunci ‘go’ diikuti oleh panggilan fungsi. Evaluasi fungsi dan argumennya terjadi dalam goroutine saat ini, dan eksekusi fungsi terjadi dalam goroutine baru. Goroutine berjalan dalam ruang alamat yang sama, sehingga akses ke memori bersama harus disinkronkan.

- Channels

Channels dalam GO adalah medium yang memungkinkan goroutine berkomunikasi satu sama lain dan menyinkronkan eksekusi mereka. Channel dapat diilustrasikan sebagai pipa yang menghubungkan goroutine yang berjalan secara konkuren, memungkinkan mereka untuk mengirim dan menerima nilai.

Channel dibuat dengan fungsi bawaan ‘make’ bersama dengan kata kunci ‘chan’ dan menentukan jenis data yang akan mengalir melalui channel. Misalnya, ‘ch := make(chan int)’ akan membuat channel bernama ‘ch’ yang dapat mentransmisikan integer.

Setelah channel dibuat, kita dapat mengirim dan menerima nilai menggunakan operator ‘<-’. Panah menunjuk arah aliran data. Misalnya ‘ch <- 42’ mengirim nilai 42 ke channel ‘ch’, dan ‘value := <- ch’ menerima nilai dari channel ‘ch’ dan menetapkannya ke variabel ‘value’.

2.3.2 Frontend Development

- Tailwind CSS

Tailwind CSS adalah kerangka kerja CSS yang dirancang untuk mempermudah pembuatan antarmuka pengguna dengan memberikan sekumpulan kelas utilitas yang dapat digunakan untuk menggambarkan gaya dan tata letak elemen HTML. Berbeda dengan kerangka kerja CSS tradisional yang cenderung menyediakan komponen-komponen siap pakai, Tailwind lebih fokus pada penggunaan kelas-kelas kecil yang dapat diterapkan langsung pada elemen HTML. Tailwind menyediakan utilitas kelas yang mencakup berbagai properti CSS. Sebagai contoh, untuk memberikan warna teks merah, Anda dapat menggunakan kelas ‘*text-red-500*’. Tailwind dirancang untuk menjadi ringan dan cepat. Dengan menggunakan hanya kelas-kelas yang diperlukan, ukuran file CSS yang dihasilkan dapat diperkecil, mengoptimalkan waktu pemuatan halaman.

- Emotion

pustaka yang memungkinkan Anda menulis style CSS di dalam komponen React. Pustaka ini merupakan bagian dari keluarga Emotion, yang adalah sebuah library yang dirancang untuk menulis style CSS dengan JavaScript secara efisien dan powerful. Emotion memungkinkan pengembangan komponen dengan style yang terisolasi menggunakan pendekatan CSS-in-JS, di mana CSS ditulis sebagai bagian dari JavaScript, bukan dalam file terpisah. Ini membantu menghindari masalah spesifisitas dan global namespace yang sering terjadi dengan CSS tradisional.

- react-three

React Three Fiber adalah sebuah pustaka yang menggabungkan React, sebuah pustaka JavaScript untuk membangun antarmuka pengguna, dengan Three.js, sebuah pustaka JavaScript untuk rendering grafis 3D di web menggunakan WebGL. React Three Fiber bertujuan untuk membawa deklaratifitas, abstraksi, dan fitur organisasi React ke dalam dunia grafis 3D. Framework ini kami implementasikan saat membangun latar belakang website yang dinamis dengan partikel bintang yang terus bergerak.

- **d3-force-3d**

Pustaka ini menggunakan prinsip-prinsip fisika seperti gravitasi, tautan, tolakan, dan daya tarik untuk mengatur posisi node dalam ruang. Pustaka ini mampu menangani kolisi antar node dan memberikan gaya yang menentukan bagaimana elemen-elemen tersebut berinteraksi, sehingga menghasilkan layout yang optimal dan mudah dibaca. Pustaka ini juga dapat membuat simulasi 3D yang dapat membuat partikel dalam background bergerak halus dan terlihat nyata.

- **Axios**

Axios adalah sebuah library yang dapat digunakan untuk melakukan HTTP request dengan menggunakan promise berbasis JavaScript. Axios dapat digunakan baik di browser maupun di node.js, dan dapat berinteraksi dengan berbagai format data, seperti JSON, XML, URL encoded, dan lain-lain. Axios juga dapat melakukan konfigurasi, intercept, transform, dan cancel request, serta menghasilkan dokumentasi API secara otomatis.

- **Next.js**

Next.js memperkenalkan kemampuan untuk menghasilkan halaman web secara statis saat build time. Halaman-halaman ini kemudian dapat disajikan dari CDN, yang meningkatkan kecepatan dan mengurangi beban pada server. Perbedaan antara React.js dan Next.js adalah React memerlukan pustaka pihak ketiga untuk routing sedangkan Next.js menggunakan struktur folder dan file dalam proyek untuk menentukan rute suatu website. Dalam Next.js, file dalam folder /pages secara otomatis menjadi rute yang sesuai.

- **react**

Pustaka ini adalah inti dalam pengembangan website ini, react merupakan sebuah pustaka JavaScript yang populer untuk membangun antarmuka pengguna, terutama untuk aplikasi web single-page. Pustaka ini menyediakan fungsi-fungsi yang memungkinkan pengembang untuk mendefinisikan komponen sebagai blok bangunan dari antarmuka pengguna, mengelola state dengan {useState} dan siklus hidup komponen.

- **react-force-graph**

Pustaka mengimplementasikan algoritma force-directed yang digunakan untuk mengatur node dan link dalam grafik. Algoritma ini menggunakan simulasi fisik untuk posisi node sedemikian rupa sehingga semua tautan (link) seimbang, sering kali membuat visualisasi yang estetis dan mudah dipahami. Pustaka ini saya implementasikan untuk mengembangkan fitur connecting graph pada website. Connecting graph yang dihasilkan pada website berdasarkan setiap individual path yang dihasilkan.

- react-spinners

'react-spinners' merupakan pustaka yang disediakan dari react yang memiliki beberapa koleksi untuk loader. Pustaka ini sendiri kami terapkan saat pengguna telah menekan tombol submit. Tombol submit akan digantikan oleh loader yang berasal dari pustaka ini. Saat program berada di state loading, submit button akan di hide dan akan muncul kembali saat state loading di set ke false. Hal ini diterapkan agar backend tidak menerima input dari frontend saat sedang melakukan pencarian rute terbaik dari alamat awal hingga alamat tujuan artikel wikipedia.

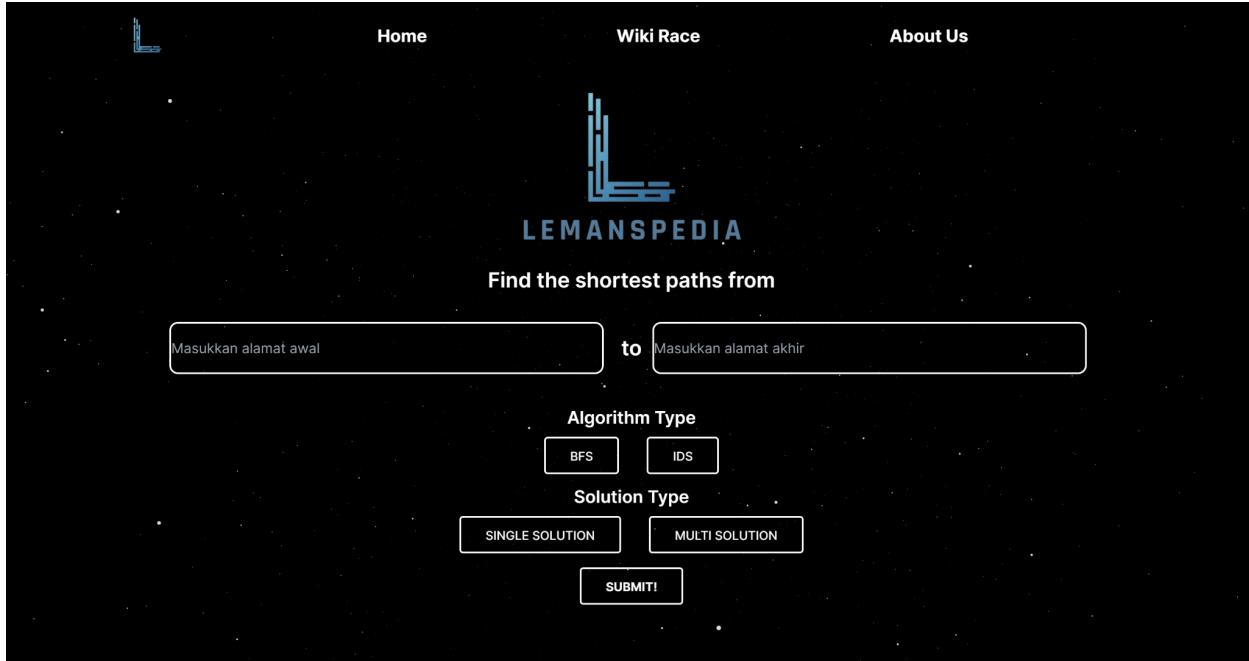
- Three JS

Three JS merupakan pustaka JavaScript yang membantu dalam proses pembuatan animasi 3D. Three JS ini menggunakan API WebGL yang dapat membantu website untuk menampilkan model atau animasi 3D secara langsung. Pustaka ini dapat berjalan dengan baik dalam program kami selama perangkat memiliki konektivitas yang baik dengan internet.

Dalam pembuatan dan pengembangan website, kami mengembangkan 3 halaman yaitu halaman utama pada lokasi '/', halaman Wiki Race pada lokasi '/wiki-race', dan juga halaman About Us pada lokasi '/about-us'. Halaman utama dari website Lemanspedia menjelaskan secara singkat apa itu Wiki Race serta penjelasan singkat mengenai algoritma BFS dan IDS. Program utama dari tugas ini berada pada halaman Wiki Race yang dapat diakses pada lokasi '/wiki-race'. Halaman Wiki Race terdiri dari kolom input alamat awal, kolom input alamat tujuan, pilihan algoritma (BFS/IDS) dan juga pilihan jenis solusi (Single Solution/Multi Solution).



Gambar 2.3.1 Halaman Utama Website LemansPedia



Gambar 2.3.2 Halaman Wiki Race Website LemansPedia

The screenshot shows the LemansPedia website's "About Us" page. At the top, there is a navigation bar with links for "Home", "Wiki Race", and "About Us". Below the navigation bar, the title "LEMANSPEDIA CONTRIBUTORS" is displayed in bold capital letters. A descriptive text follows, stating: "The main objective from the second major assignment of Algorithm Strategies courses is to make a WikiRace programs with Breadth First Search (BFS) and Iterative Deepening Search (IDS) Algorithms." Below this text are three circular profile pictures, each enclosed in a white-bordered box. The first box contains the profile picture of Aland Mulia Pratama, with the text "Aland Mulia Pratama" and "13522124" below it, followed by "Frontend & Backend Developer" and the email "(13522124@std.stei.itb.ac.id)". The second box contains the profile picture of Rizqika Mulia Pratama, with the text "Rizqika Mulia Pratama" and "13522126" below it, followed by "IDS Algorithm & Backend Developer" and the email "(13522126@std.stei.itb.ac.id)". The third box contains the profile picture of Ikhwan Al Hakim, with the text "Ikhwan Al Hakim" and "13522147" below it, followed by "BFS Algorithm & Backend Developer" and the email "(13522147@std.stei.itb.ac.id)".

Gambar 2.3.3 Halaman About Us Website LemansPedia

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

3.1.1 Algoritma BFS

Setelah menerima masukan judul mulai dan judul target dari front end, program akan memasukkan semua hyperlink dari laman Wikipedia target ke dalam sebuah antrian. Setelah itu, dilakukan dequeue dari antrian tersebut dan untuk masing-masing hasil dequeue, program akan mengakses halaman Wikipedia yang sesuai dengan hyperlink tersebut. Kemudian, program akan mengekstrak semua hyperlink dari halaman tersebut dan memasukkannya ke dalam antrian.

Proses ini akan terus berulang hingga entri-antri dalam antrian habis atau judul target ditemukan. Path dari setiap simpul dalam antrian akan disimpan. Ketika sudah ditemukan judul target dalam salah satu cabang pohon pencarian, maka pencarinya akan langsung berhenti dan program akan mengirimkan informasi path dari cabang dimana judul target ditemukan ke front end.

3.1.2 Algoritma IDS

Setelah menerima masukan judul mulai dan judul target dari front end, program ini akan memulai pencarian dengan kedalaman yang meningkat secara iteratif hingga kedalaman maksimum yang ditentukan. Pada setiap kedalaman, program ini akan menjelajahi semua node yang dapat dijangkau dalam kedalaman tersebut menggunakan struktur data stack.

Stack digunakan untuk menyimpan node yang perlu dijelajahi. Node awal dimasukkan ke dalam stack sebagai titik awal. Kemudian, dalam setiap iterasi, node teratas dari stack (node saat ini) dihapus dari stack dan semua tetangganya yang belum dikunjungi ditambahkan ke stack. Proses ini berlanjut sampai stack kosong atau node target ditemukan.

Untuk setiap node yang dijangkau, program ini akan memeriksa apakah node tersebut adalah node target. Jika ya, maka program akan menambahkan path yang ditemukan unik. Selama proses ini, program menggunakan beberapa goroutine pekerja untuk memproses URL dari setiap node. Setiap pekerja akan mengambil URL dari antrian URL, mengakses halaman Wikipedia yang sesuai, mengekstrak semua hyperlink dari halaman tersebut, dan memasukkannya ke dalam antrian hasil. Jika terjadi kesalahan selama proses ini, pekerja akan memasukkan kesalahan tersebut ke dalam antrian kesalahan.

Setelah semua node telah dijangkau atau node target ditemukan, program ini akan menutup semua antrian dan menunggu semua pekerja selesai. Kemudian, program akan mengembalikan hasil akhir, jumlah artikel yang diperiksa, jumlah artikel yang dijangkau, jumlah path yang ditemukan, dan waktu yang dibutuhkan untuk menyelesaikan pencarian.

3.2 Mapping Persoalan Menjadi Elemen BFS dan IDS

Pemetaan elemen-elemen graf ke dalam persoalan Wikirace adalah sebagai berikut:

Tabel 3.2.1 Pemetaan elemen-elemen graf ke dalam persoalan Wikirace

Elemen	Deskripsi
Vertex	Merepresentasikan judul dari laman Wikipedia
Edge	Memiliki arti bahwa dua Vertex saling berhubungan melalui sebuah hyperlink

Dengan elemen-elemen tersebut, berikut adalah rancangan singkat penggunaan algoritma BFS dan IDS untuk menyelesaikan persoalan Wikirace:

Breadth-First Search (BFS):

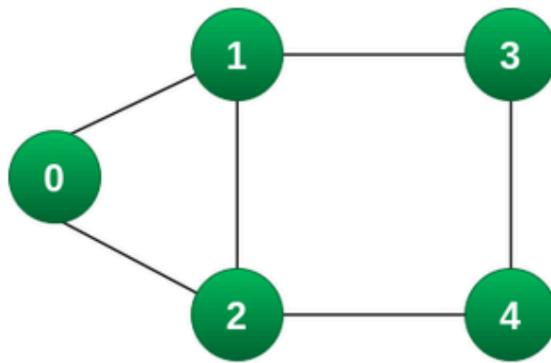
1. Mulai dari artikel awal dan tambahkan ke dalam antrian.
2. Selama antrian tidak kosong, ambil artikel dari depan antrian.
3. Jika artikel adalah artikel tujuan, selesaikan pencarian.
4. Jika bukan, tambahkan semua tautan dalam artikel ke antrian (asalkan mereka belum pernah dikunjungi).
5. Ulangi proses sampai menemukan artikel tujuan atau antrian kosong.

Iterative Deepening Search (IDS):

1. Mulai dengan batasan kedalaman pencarian 0.
2. Lakukan Depth-Limited Search (DLS) dengan batasan kedalaman saat ini.
3. Jika artikel tujuan belum ditemukan, tambahkan 1 pada batasan kedalaman dan ulangi langkah di atas.
4. Lakukan langkah ini hingga mencapai batasan kedalaman maksimum.

3.3 Contoh Ilustrasi Kasus

Diambil contoh yang hendak mencari jalur dari Bandung Institute of Technology ke Japan, maka program akan melakukan iterasi pada setiap page mulai dari Bandung Institute of Technology, bila masih belum ditemukan pada kedalaman tertentu, maka akan dicari di kedalaman berikutnya, hal ini terus berlanjut sampai target Japan ditemukan atau program telah mencapai kedalaman maksimum yang ditentukan, hasil pencarian yang berupa list of node lalu divisualisasikan dalam bentuk graf. Berikut adalah ilustrasi pohon perantangan algoritma BFS dan algoritma IDS:



Gambar 3.3.1 Pohon untuk ilustrasi kasus algoritma BFS dengan memanfaatkan queue

Tabel 3.3.1 Queue dan Visited mula-mula kosong

Visited					
Queue					

Tabel 3.3.2 Queue dan Visited pada iterasi pertama

Visited	0				
Queue	0				

Tabel 3.3.3 Queue dan Visited pada iterasi kedua

Visited	0	1	2		
Queue	1	2			

Tabel 3.3.4 Queue dan Visited pada iterasi ketiga

Visited	0	1	2	3	
Queue	2	3			

Tabel 3.3.5 Queue dan Visited pada iterasi keempat

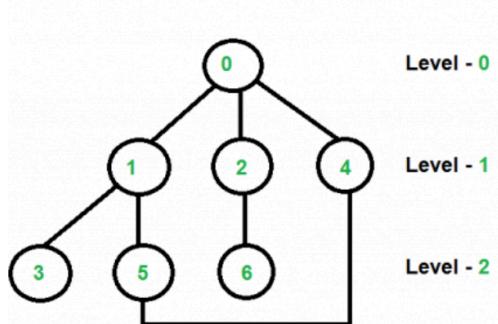
Visited	0	1	2	3	4
Queue	3	4			

Tabel 3.3.6 Queue dan Visited pada iterasi kelima

Visited	0	1	2	3	4
Queue	4				

Tabel 3.3.7 Queue dan Visited pada iterasi keenam

Visited	0	1	2	3	4
Queue					



Gambar 3.3.2 Pohon untuk ilustrasi kasus algoritma IDS

Tabel 3.3.8 Tabel perentangan pohon Algoritma IDS

Depth	Iterative Deepening Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1

Ilustrasi tersebut berlaku untuk kedua jenis algoritma yaitu Breadth First Search (BFS) dan Iterative Deepening Search (IDS). Kami juga melampirkan lima hasil uji coba pada program kami yang dapat dilihat lebih lanjut pada subbab 4.4 mengenai hasil pengujian. Perbedaan mendasar dari kedua algoritma tersebut adalah strategi pemilihan node berikutnya dan dalam kasus ini node adalah link halaman Wikipedia.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

4.1.1 BFS

```
function bfsMultiCall(startURL: String, targetTitle: String) -> (List of List of String, Integer, Integer, Integer, Float)
```

Kamus

```
results: List of List of String
articlesChecked, articlesTraversed, multiDepth: Integer
hrefs: List of String
mu: sync.Mutex
found: Boolean
```

Algoritma

```
found <- false
multiDepth <- 10
startTime <- time.Now()
bfsMulti(startURL, targetTitle, hrefs, &mu, &articlesChecked, &found, &results,
&multiDepth)
elapsedTime <- time.Since(startTime).Seconds()

i traversal [0..length(results)]
    results[i] = removeDuplicates(results[i])

numberPath <- len(results)

-> results, articlesChecked, articlesTraversed, numberPath, elapsedTime
```

```
procedure bfsMulti(startURL: String, targetTitle: String, hrefs: List of String,
mu: *sync.Mutex, passed: Integer, found: Boolean, results: String, multiDepth:
Integer)
```

Kamus

```
Page: {URL: String, Path: List of String, Depth: Integer}
depth: Integer
queue: Queue of Page
wg: sync.WaitGroup
```

Algoritma

```
queue <- []Page{{URL: startURL, Path: []string{}, Depth: depth}}

while len(queue) > 0 do
    mu.Lock()
    currentPages <- make([]Page, 0, 100)
    i traversal [0..len(queue)]
        currentPages <- append(currentPages, queue[0])
        queue <- queue[1:]
    mu.Unlock()

    i traversal [0..len(currentPages)]
        wg.Add(1)
```

```

go func(page Page) {
    defer wg.Done()

    if page.Depth > *multiDepth then
        ->

        res, err <- http.Get(page.URL)
        if err != nil then
            log.Fatal(err)

        defer res.Body.Close()
        if res.StatusCode != 200 then
            log.Fatalf("status code error: %d %s", res.StatusCode,
res.Status)

        doc, err <- goquery.NewDocumentFromReader(res.Body)
        if err != nil then
            log.Fatal(err)

        firstPath <- doc.Find("title").Text()
        firstPathSplit <- strings.Split(firstPath, " - ")

        if len(firstPathSplit) > 0 then
            page.Path = append(page.Path,
strings.TrimSpace(firstPathSplit[0]))

        content <- doc.Find("#mw-content-text")
        content.Find("a").Each(func(i int, s *goquery.Selection) {
            href, exists <- s.Attr("href")
            if exists && strings.HasPrefix(href, "/wiki/") then
                title <- firstPathSplit[0]
                fmt.Printf("checking %s\n", title)
                if title == targetTitle then
                    result <- append(page.Path, title)
                    *results <- append(*results, result)
                    *found <- true
                    *multiDepth <- page.Depth
                    ->
                else if href != "/wiki/Main_Page" &&
!isInList(href, hrefs) then
                    hrefs <- append(hrefs, href)
                    mu.Lock()
                    *passed++
                    queue <- append(queue, Page{URL:
"https://en.wikipedia.org" + href, Path: page.Path, Depth: page.Depth + 1})
                    mu.Unlock()
                }
            })
        })
    wg.Wait()
}

function bfsSingleCall(startURL: String, targetTitle: String) -> (List of List of
String, Integer, Integer, Integer, Float)

```

Kamus

```

results: List of List of String
articlesChecked, articlesTraversed: Integer
hrefs: List of String
mu: sync.Mutex
found: Boolean

Algoritma
found <- false
startTime <- time.Now()
bfsSingle(startURL, targetTitle, hrefs, &mu, &articlesChecked, &found, &results)
elapsedTime <- time.Since(startTime).Seconds()

i traversal [0..len(results)]
    results[i] = removeDuplicates(results[i])

numberPath <- len(results)

-> results, articlesChecked, articlesTraversed, numberPath, elapsedTime

```

```

procedure bfsSingle(startURL: String, targetTitle: String, hrefs: List of String,
mu: *sync.Mutex, passed: Integer, found: Boolean, results: String)

Kamus
Page: {URL: String, Path: List of String, Depth: Integer}
depth: Integer
queue: Queue of Page
wg: sync.WaitGroup

Algoritma
queue <- []Page{{URL: startURL, Path: []string{}, Depth: depth}}

while len(queue) > 0 do
    if *found then
        ->
    mu.Lock()
    currentPages <- make([]Page, 0, 100)
    i traversal [0..len(queue)]
        if *found then
            ->
            currentPages <- append(currentPages, queue[0])
            queue <- queue[1:]
    mu.Unlock()

    i traversal [0..len(currentPages)]
        wg.Add(1)
        go func(page Page) {
            defer wg.Done()

            if *found then
                ->

                res, err <- http.Get(page.URL)
                if err != nil then
                    log.Fatal(err)

```

```

        defer res.Body.Close()
        if res.StatusCode != 200 then
            log.Fatalf("status code error: %d %s", res.StatusCode,
res.Status)

        doc, err <- goquery.NewDocumentFromReader(res.Body)
        if err != nil then
            log.Fatal(err)

        firstPath <- doc.Find("title").Text()
        firstPathSplit <- strings.Split(firstPath, " - ")

        if len(firstPathSplit) > 0 then
            page.Path = append(page.Path,
strings.TrimSpace(firstPathSplit[0]))

            content <- doc.Find("#mw-content-text")
            content.Find("a").Each(func(i int, s *goquery.Selection) {
                href, exists <- s.Attr("href")
                if exists && strings.HasPrefix(href, "/wiki/") then
                    title <- firstPathSplit[0]
                    fmt.Printf("checking %s\n", title)
                    if title == targetTitle then
                        result <- append(page.Path, title)
                        *results <- append(*results, result)
                        *found <- true
                    ->
                else if href != "/wiki/Main_Page" &&
!isInList(href, hrefs) then
                    if *found then
                        ->
                    hrefs <- append(hrefs, href)
                    mu.Lock()
                    *passed++
                    queue <- append(queue, Page{URL:
"https://en.wikipedia.org" + href, Path: page.Path, Depth: page.Depth + 1})
                    mu.Unlock()
                }
            })
        currentPages[i])
    wg.Wait()

```

4.1.2 IDS

```

function iterativeDeepeningAll(start, target: Node) -> (List of List of String,
Integer, Integer, Integer, Float)

Kamus
results: List of List of String
articlesChecked, articlesTraversed, maxDepth, foundDepth, numberPath: Integer
pathSet: List of Boolean
urlQueue, visited: List of String
resultQueue, current: Node

```

```

errQueue: error
wg: sync.WaitGroup
stack: List of Node
pathKey: String
elapsedTime: Float

Algoritma
startTime <- time.Now()
i traversal [0..30]
    wg.Add(1)
    Go worker(urlQueue, resultQueue, errQueue, &wg)

foundDepth <- maxDepth + 1

depth iterate [0..maxDepth] and depth <= foundDepth
    start.Path <- []string{start.Title}
    stack <- append(stack, start)
    while len(stack) > 0
        current <- stack[len(stack)-1]
        stack <- stack[:len(stack)-1]
        pathKey <- strings.Join(current.Path, "->")
        output("Checking" + pathKey)

        If current.Title = target.Title and not pathSet[pathKey]
            results <- append(results, current.Path)
            pathSet[pathKey] = true
            if depth < foundDepth
                foundDepth <- depth

        If length(current.Path) > depth or visited[current.URL]
            continue

        visited[current.URL] <- true
        urlQueue <- current.URL

        select
        case neighbors <- <-resultQueue
            articlesChecked++
            _, neighbor iterate range neighbors
                If not visited[neighbor.URL]
                    neighbor.Path <- append([]string(nil), current.Path)
                    neighbor.Path <- append(neighbor.Path, neighbor.Title)
                    stack <- append(stack, neighbor)
                    articlesTraversed++

        case err <- <-errQueue
            output(err)

close(urlQueue)
wg.Wait()
close(resultQueue)
close(errQueue)

```

```

elapsedTime <- time.Since(startTime).Seconds()
numberPath <- len(results)

-> results, articlesChecked, articlesTraversed, numberPath, elapsedTime

```

```

function iterativeDeepeningShortest(start, target: Node) -> (List of List of
String, Integer, Integer, Integer, Float)

```

Kamus

```

results: List of List of String
articlesChecked, articlesTraversed, maxDepth, foundDepth, numberPath: Integer
pathSet: List of Boolean
urlQueue, visited: List of String
resultQueue, current: Node
errQueue: error
wg: sync.WaitGroup
stack: List of Node
pathKey: String
elapsedTime: Float
found: Boolean

```

Algoritma

```

startTime <- time.Now()
i traversal [0..30]
    wg.Add(1)
    Go worker(urlQueue, resultQueue, errQueue, &wg)

foundDepth <- maxDepth + 1
found <- false

depth iterate [0..maxDepth] and depth <= foundDepth
    start.Path <- []string{start.Title}
    stack <- append(stack, start)
    while len(stack) and not found > 0
        current <- stack[len(stack)-1]
        stack <- stack[:len(stack)-1]
        pathKey <- strings.Join(current.Path, "->")
        output("Checking" + pathKey)

        if current.Title = target.Title and not pathSet[pathKey]
            results <- append(results, current.Path)
            pathSet[pathKey] = true
            found <- true
            break

        if length(current.Path) > depth or visited[current.URL]
            continue

        visited[current.URL] <- true
        urlQueue <- current.URL

select

```

```

        case neighbors <- <-resultQueue
            articlesChecked++
            _, neighbor iterate range neighbors
                If not visited[neighbor.URL]
                    neighbor.Path <- append([]string(nil), current.Path)
                    neighbor.Path <- append(neighbor.Path, neighbor.Title)
                    stack <- append(stack, neighbor)
                    articlesTraversed++

        case err <- <-errQueue
            output(err)

close(urlQueue)
wg.Wait()
close(resultQueue)
close(errQueue)

elapsedTime <- time.Since(startTime).Seconds()
numberPath <- len(results)

-> results, articlesChecked, articlesTraversed, numberPath, elapsedTime

```

function fetchLinks(urlString: string, ch: chan<-[]Node, errCh: chan<-error)
I.S: -
F.S: URL hasil scraping dijadikan node

Kamus

- urlString, href, title, fullURL: String
- ch, nodes: List of Node
- errCh: chan<- error
- sem: chan struct{}
- urlCache: Map of String to Node
- req: http.request
- httpClient: http.Client
- res: http.Response
- doc: goquery.Document
- exists: Boolean

Algoritma

```

sem <- struct{}{}
defer func(){<-sem}()

if nodes, ok = urlCache[urlString] ok
    ch <- nodes
    ->

req, _ <- http.NewRequest("GET", urlString, nil)
req.Header.Set("Connection", "keep-alive")

res, err <- httpClient.Do(req)
if err != nil
    errCh <- err

```

```

return

defer res.Body.Close()
if res.StatusCode != 200
    errCh <- output("status code error: %d %s", res.StatusCode, res.Status)
    return

doc, err <- goquery.NewDocumentFromReader(res.Body)
if err != nil
    errCh <- err
    return

doc.Find("#mw-content-text a").Each(func(_ int, s *goquery.Selection)
    href, exists := s.Attr("href")
    if exists and strings.HasPrefix(href, "/wiki/") and not strings.Contains(href,
":") {
        title <- strings.TrimPrefix(href, "/wiki/")
        title, err <- url.QueryUnescape(title)
        if err != nil {
            errCh <- err
            return
        title <- strings.ReplaceAll(title, "_", " ") // Replace underscores with
spaces
        fullURL <- "https://en.wikipedia.org" + href
        nodes <- append(nodes, Node{Title: title, URL: fullURL, Path:
[]string{title}})
    urlCache[urlString] = nodes
    ch <- nodes
}

```

function worker(urlString: string, ch: chan<-[]Node, errCh: chan<-error)

I.S:

F.S: Worker selesai mengolah mengecek node

Kamus

urlQueue, ch: <-chan string
resultQueue: chan<- List of Node
errQueue, errCh: chan<- error
wg: sync.WaitGroup
url: String
result: List of Node
Err: error

Algoritma

```

url iterate [len(urlQueue)]
    ch <- make(chan []Node)
    errCh <- make(chan error)
    go fetchLinks(url, ch, errCh)
    select
        case result <- <-ch:
            resultQueue <- result
        case err <- <-errCh:

```

```
    errQueue <- err  
wg.Done()
```

4.2 Struktur Data dan Spesifikasi Program

4.1.1 BFS

```
type Page struct {  
    URL string  
    Path []string  
    Depth int  
}
```

- a. URL: menyimpan URL dari laman Wikipedia
- b. Path: menyimpan path dari judul mulai hingga judul target berupa array of title
- c. Depth: menyimpan informasi kedalaman judul target

4.1.2 IDS

```
type Node struct {  
    Title string  
    URL string  
    Path []string  
}
```

- a. Title: Judul dari halaman Wikipedia
- b. URL : Menyimpan link dari halaman Wikipedia
- c. Path : Menyimpan sekumpulan path dari alamat awal artikel hingga alamat tujuan artikel

```
type Result struct {  
    Results      []string `json:"results"  
    ArticlesChecked int     `json:"articlesChecked"  
    ArticlesTraversed int     `json:"articlesTraversed"  
    NumberPath    int     `json:"numberPath"  
    ElapsedTime   float64 `json:"elapsedTime"  
}
```

- a. Results: Sebuah slice dari string yang masing-masingnya menyimpan jalur (path) yang ditemukan dari node awal ke node tujuan. Setiap string dalam slice ini mewakili satu jalur unik dalam bentuk urutan judul halaman yang dikunjungi.
- b. ArticlesChecked: Sebuah integer yang menyimpan jumlah artikel yang telah diperiksa selama proses pencarian. Ini mencakup semua artikel yang diakses untuk mendapatkan tautan ke halaman lain, termasuk halaman yang mungkin tidak mengarah langsung ke tujuan tetapi diperiksa dalam proses mencari jalur yang mungkin.
- c. ArticlesTraversed: Sebuah integer yang menyimpan jumlah artikel yang benar-benar dilalui atau di-traverse selama pencarian. Nilai ini mungkin berbeda dari ArticlesChecked karena tidak semua artikel yang diperiksa akan dilalui; beberapa mungkin hanya diperiksa untuk melihat apakah mereka mengandung tautan yang berguna.

- d. NumberPath: Sebuah integer yang menyatakan jumlah jalur unik yang ditemukan dari node awal ke node tujuan. Ini mengukur berapa banyak jalur yang berhasil ditemukan yang memenuhi kriteria pencarian.
- e. ElapsedTime: Sebuah nilai float64 yang merepresentasikan total waktu yang dihabiskan untuk melakukan pencarian, diukur dalam detik. Ini membantu mengukur efisiensi algoritma pencarian dari segi waktu.

4.3 Tata Cara Penggunaan Program

Terdapat beberapa prasyarat/dependencies yang harus diunduh dan/atau diinstalasi sebelum menjalankan program LemansPedia yaitu program Wiki Race dengan algoritma BFS dan IDS yang berbasis website. Prasyarat/dependencies tersebut adalah:

1. Backend
 - GO Programming Language (<https://go.dev/doc/install>)
2. Frontend
 - Next JS (<https://nextjs.org/learn-pages-router/basics/create-nextjs-app>)
 - React (<https://react.dev/>)
 - JavaScript (<https://www.javascript.com/>)

Selain memenuhi dependencies dari program, perlu juga dilakukan beberapa konfigurasi untuk program Wiki Race berbasis website. Konfigurasi perlu dilakukan baik dari sisi Backend maupun Frontend. Berikut adalah konfigurasi yang diperlukan baik dari Backend maupun Frontend:

1. Clone Repository

Clone repository program LemansPedia dengan mengeksekusi perintah dibawah pada terminal. Repository program dapat diakses pada pranala [berikut](#).

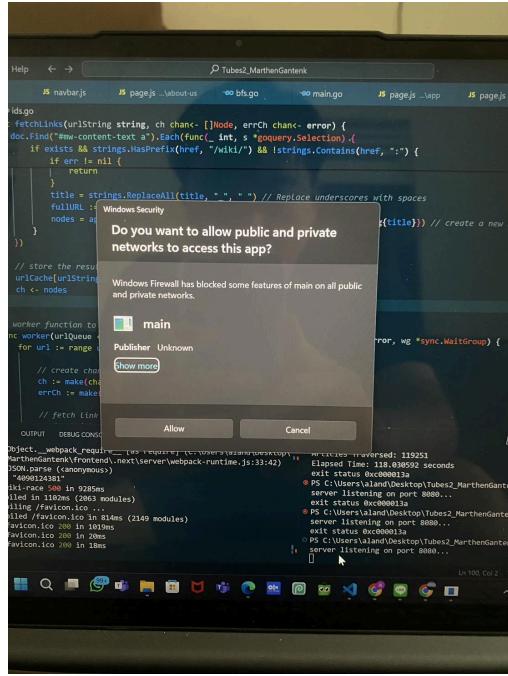
```
git clone https://github.com/alandmpertma/Tubes2_MarthenGantenk.git
```
2. Konfigurasi Backend
 - Lakukan instalasi GO Programming Language. Instalasi GO Programming Language dapat diakses pada <https://go.dev/doc/install>. Pastikan untuk menambahkan PATH saat instalasi pada perangkat anda.
 - Setelah GO Programming Language berhasil terinstall. Jalankan perintah dibawah untuk mengubah directory ke backend. Pastikan directory awal adalah project directory dari program ini.

```
cd backend
```

 - Jalankan backend dengan mengeksekusi perintah di bawah.

```
go run .
```

 - Apabila muncul peringatan dari Windows Security seperti dibawah anda bisa menekan tombol allow.



Gambar 4.3.1 Windows Security perangkat

3. Konfigurasi Frontend

- Pastikan directory awal berada pada project directory dan jalankan perintah dibawah pada terminal untuk pindah directory ke 'frontend'.

```
cd frontend
```

- Jalankan perintah dibawah pada terminal supaya framework React serta dependencies yang dibutuhkan dalam website dapat berjalan secara lokal.

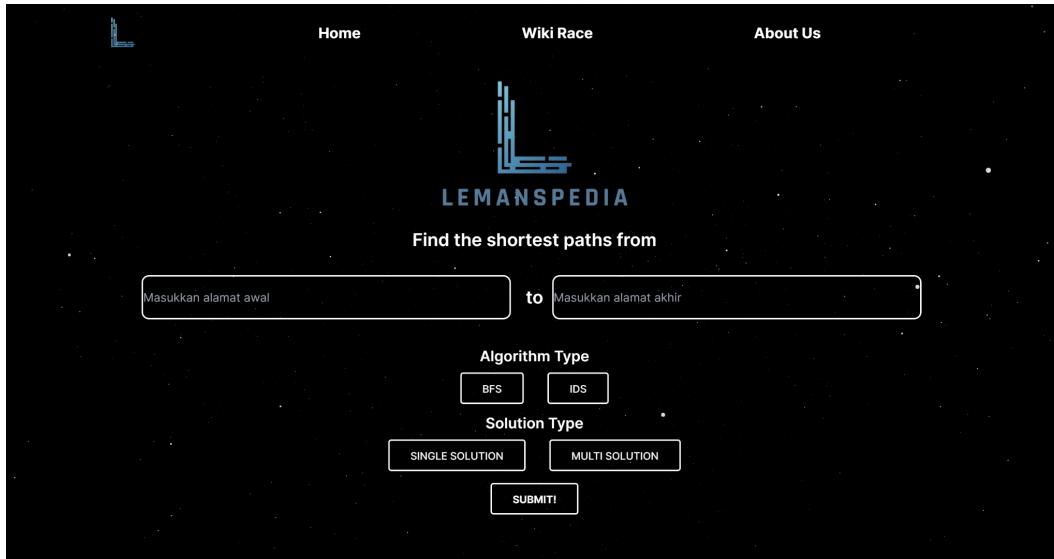
```
npm install
```

- Eksekusi perintah dibawah lalu akses halaman website secara lokal di <http://localhost:3000>.

```
npm run dev
```

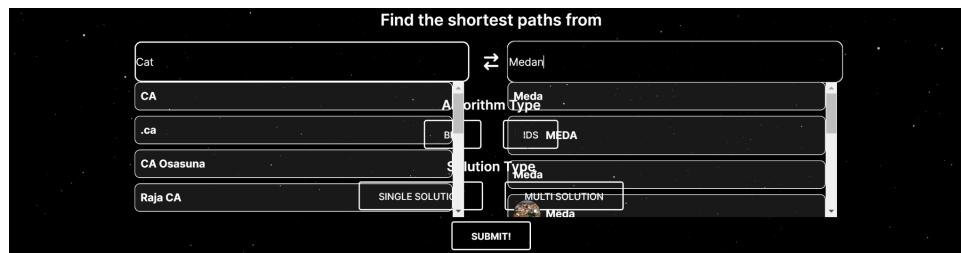
Setelah konfigurasi program selesai, program dapat digunakan dengan tata cara sebagai berikut:

- Akses halaman Wiki Race website pada laman <http://localhost:3000/wiki-race>.



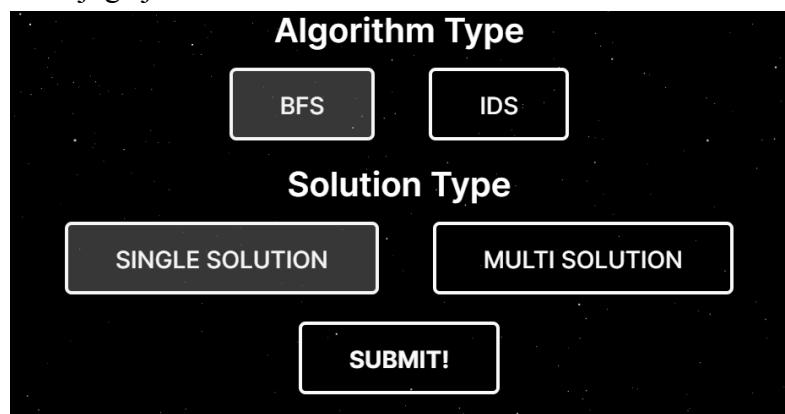
Gambar 4.3.2 Halaman Wiki Race

2. Masukkan input alamat awal dan alamat tujuan artikel website. Masukan dapat dilakukan secara manual atau menggunakan fitur *autocomplete*.



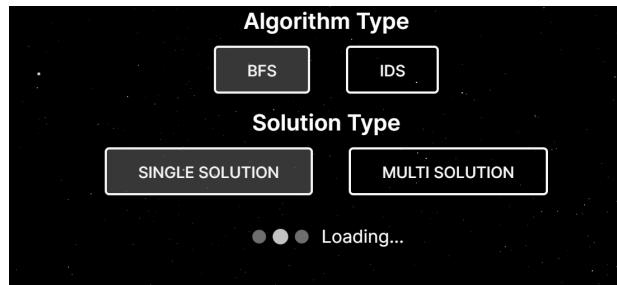
Gambar 4.3.3 Kolom Input Wiki Race

3. Setelah itu anda dapat memilih jenis algoritma (BFS/IDS) dan juga jenis solusi (Single Solution/Multi Solution). Masukan tidak dapat disubmit selama anda belum memilih jenis algoritma dan juga jenis solusi.



Gambar 4.3.4 Pilihan jenis algoritma dan solusi

4. Setelah semua telah dipastikan benar, anda dapat menekan tombol submit dan menunggu hasil rute terpendek dari alamat awal menuju alamat akhir artikel wikipedia.



Gambar 4.3.5 State loading website

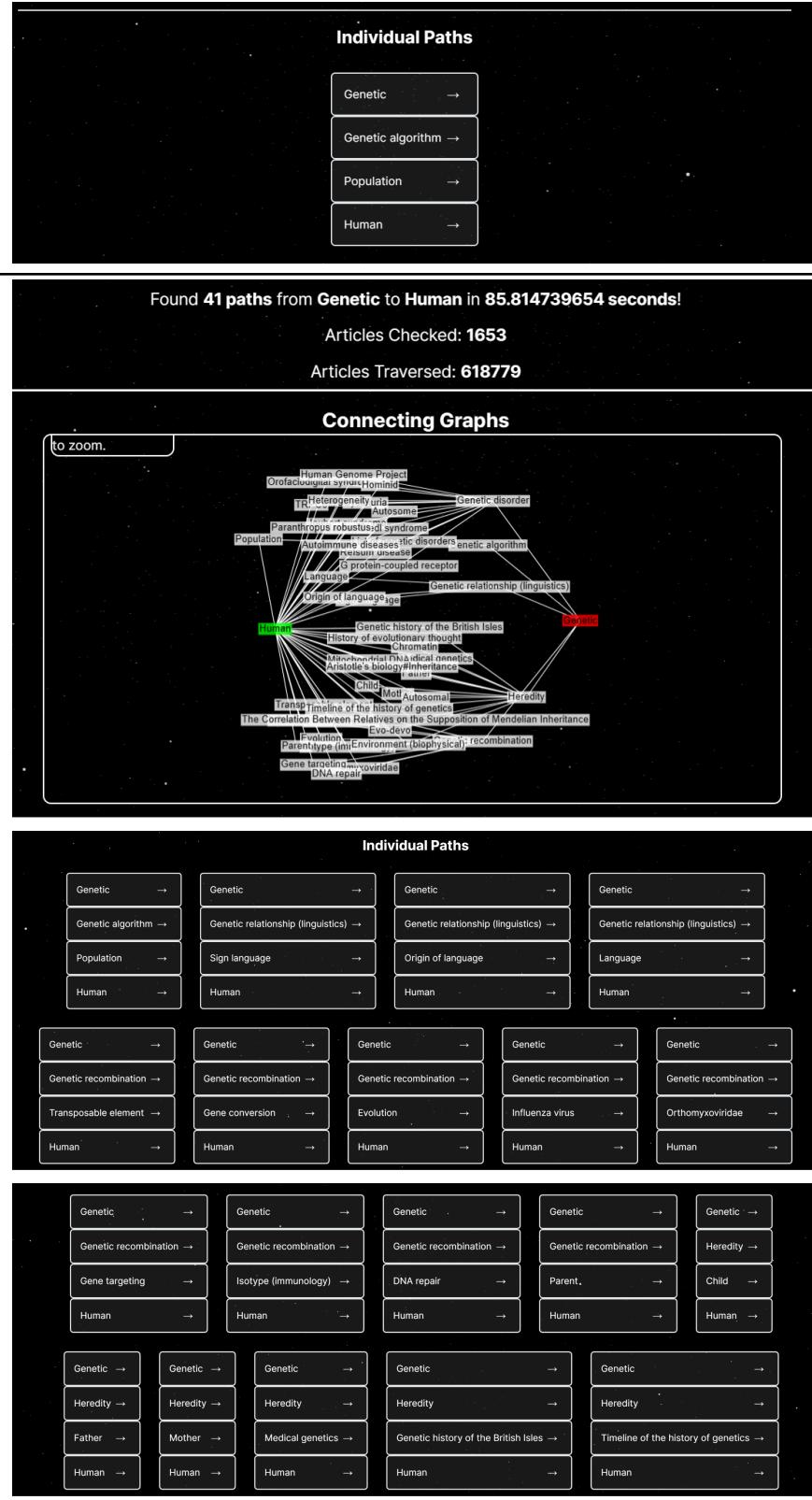
4.4 Hasil Pengujian

TEST 1		
ALGORITMA	SOLUSI	HASIL
BFS	Single Solution	<p>Find the shortest paths from</p> <div style="display: flex; justify-content: space-around;"> <div>Genetic</div> <div>\leftrightarrow</div> <div>Human</div> </div> <p>Algorithm Type</p> <div style="display: flex; justify-content: space-around;"> <div>BFS</div> <div>IDS</div> </div> <p>Solution Type</p> <div style="display: flex; justify-content: space-around;"> <div>SINGLE SOLUTION</div> <div>MULTI SOLUTION</div> </div> <p style="text-align: center;">SUBMIT!</p> <p>Found 1 paths from Genetic to Human in 3.246909356 seconds! Articles Checked: 429 Articles Traversed: 17663</p> <p>Connecting Graphs</p>

		<p>Individual Paths</p> <pre> graph TD Genetic --> Gene Gene --> GeneExpression[Gene expression] GeneExpression --> Human </pre>
BFS	Multi Solution	<p>Found 130 paths from Genetic to Human in 71.282980234 seconds!</p> <p>Articles Checked: 11673</p> <p>Articles Traversed: 187211</p> <p>Connecting Graphs</p>
	Single Solution	<p>Found 1 paths from Genetic to Human in 8.75373451 seconds!</p> <p>Articles Checked: 176</p> <p>Articles Traversed: 51923</p> <p>Connecting Graphs</p>

IDS

Multi
Solution





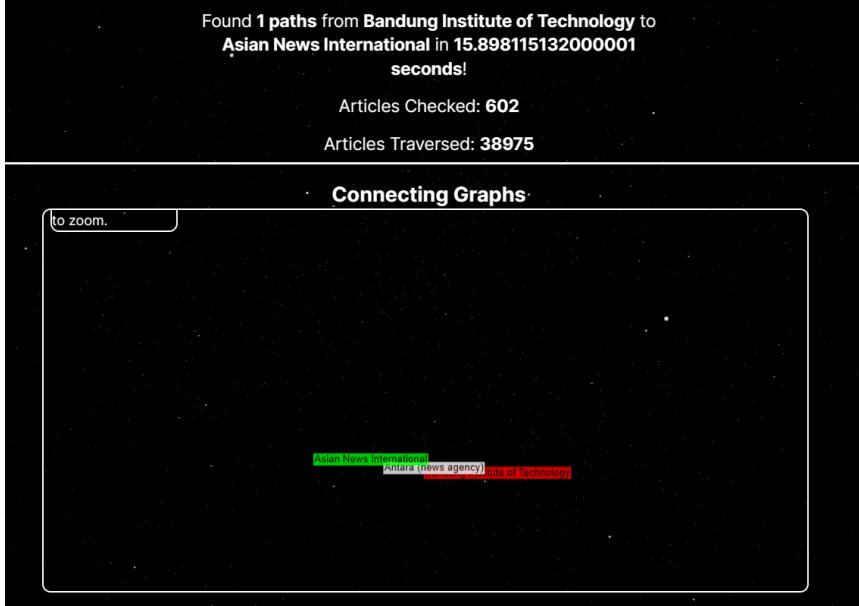
TEST 2

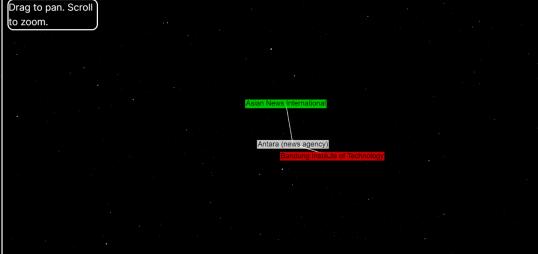
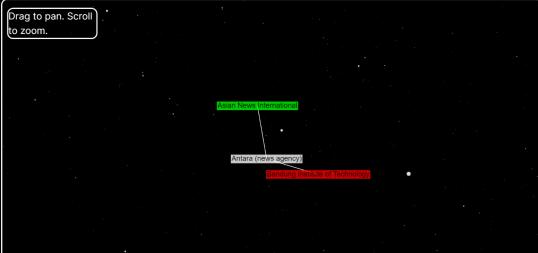
The page features a navigation bar with "Home", "Wiki Race", and "About Us". Below the navigation is the Lemanspedia logo, which is a stylized blue 'L' shape above the word "LEMANSPEDIA".

The main content area contains the text "Find the shortest paths from" followed by two input fields: "Bandung Institute of Technology" and "Asian News International".

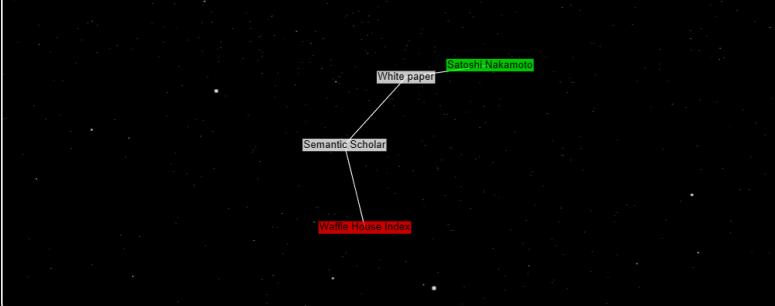
Below the input fields are sections for "Algorithm Type" (with "BFS" and "IDS" buttons), "Solution Type" (with "SINGLE SOLUTION" and "MULTI SOLUTION" buttons), and a "SUBMIT!" button.

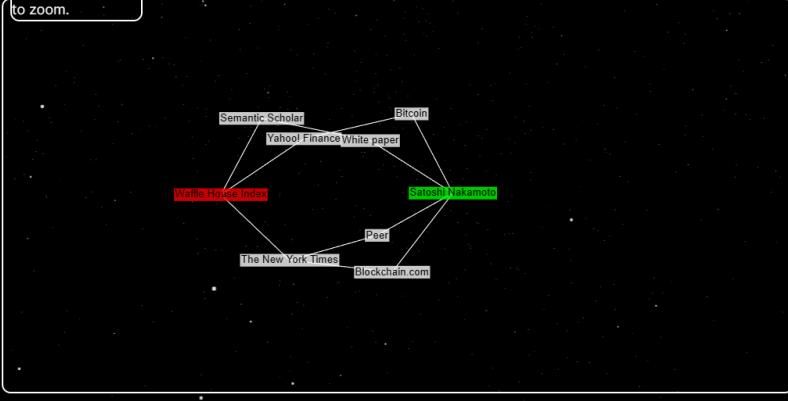
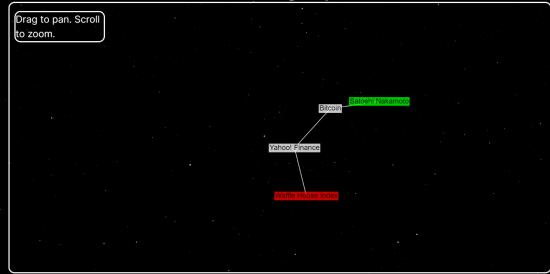
ALGORITMA	SOLUSI	HASIL
-----------	--------	-------

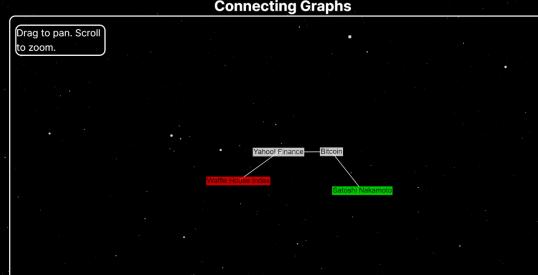
	BFS	<p>Single Solution</p>  <p>Found 1 paths from Bandung Institute of Technology to Asian News International in 15.898115132000001 seconds!</p> <p>Articles Checked: 602 Articles Traversed: 38975</p> <p>Connecting Graphs</p> <p>to zoom.</p> <p>Asian News International Antara (news agency) Bandung Institute of Technology</p>
	BFS	<p>Multi Solution</p>  <p>Found 1 paths from Bandung Institute of Technology to Asian News International in 9.46679294 seconds!</p> <p>Articles Checked: 1118 Articles Traversed: 46026</p> <p>Individual Paths</p> <ul style="list-style-type: none"> Bandung Institute of Technology → Antara (news agency) → Asian News International → <p>Connecting Graphs</p> <p>to zoom.</p> <p>Asian News International Antara (news agency) Bandung Institute of Technology</p>

		<p>Individual Paths</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Bandung Institute of Technology → Antara (news agency) → Asian News International → </div>
IDS	Single Solution	<p>Found 1 paths from Bandung Institute of Technology to Asian News International in 27.869977 seconds!</p> <p>Articles Checked: 204 Articles Traversed: 44424</p> <p>Connecting Graphs</p>  <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Bandung Institute of Technology → Antara (news agency) → Asian News International → </div>
IDS	Multi Solution	<p>Found 1 paths from Bandung Institute of Technology to Asian News International in 107.5780997 seconds!</p> <p>Articles Checked: 374 Articles Traversed: 119252</p> <p>Connecting Graphs</p>  <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Bandung Institute of Technology → Antara (news agency) → Asian News International → </div>

TEST 3

 <p>LEMANSPEDIA</p> <p>Find the shortest paths from</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <input type="text" value="Waffle House Index"/> </div> <div style="border: 1px solid black; padding: 5px; text-align: center;">  <input type="text" value="Satoshi Nakamoto"/> </div> </div> <p>Algorithm Type</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 2px 10px;">BFS</div> <div style="border: 1px solid black; padding: 2px 10px;">IDS</div> </div> <p>Solution Type</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 2px 10px;">SINGLE SOLUTION</div> <div style="border: 1px solid black; padding: 2px 10px;">MULTI SOLUTION</div> </div> <div style="text-align: center; margin-top: 10px;"> <input type="button" value="SUBMIT!"/> </div>		
ALGORITMA	SOLUSI	HASIL
BFS	Single Solution	<p>Found 1 paths from Waffle House Index to Satoshi Nakamoto in 142.238760959 seconds!</p> <p>Articles Checked: 15310</p> <p>Articles Traversed: 442462</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p style="text-align: center;">Connecting Graphs</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto;"> to zoom.  </div> </div> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p style="text-align: center;">Individual Paths</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Waffle House Index → </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Semantic Scholar → </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> White paper → </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> Satoshi Nakamoto → </div> </div> </div>

BFS	Multi Solution	<p>Found 4 paths from Waffle House Index to Satoshi Nakamoto in 480.682314157 seconds!</p> <p>Articles Checked: 42182 Articles Traversed: 791272</p> <p>Connecting Graphs</p>  <p>Individual Paths</p> <table border="1"> <tbody> <tr><td>Waffle House Index →</td><td>Waffle House Index →</td><td>Waffle House Index →</td><td>Waffle House Index →</td></tr> <tr><td>Semantic Scholar →</td><td>Yahoo! Finance →</td><td>The New York Times →</td><td>The New York Times →</td></tr> <tr><td>White paper →</td><td>Bitcoin →</td><td>Peer →</td><td>Blockchain.com →</td></tr> <tr><td>Satoshi Nakamoto ←</td><td>Satoshi Nakamoto →</td><td>Satoshi Nakamoto →</td><td>Satoshi Nakamoto →</td></tr> </tbody> </table>	Waffle House Index →	Semantic Scholar →	Yahoo! Finance →	The New York Times →	The New York Times →	White paper →	Bitcoin →	Peer →	Blockchain.com →	Satoshi Nakamoto ←	Satoshi Nakamoto →	Satoshi Nakamoto →	Satoshi Nakamoto →			
Waffle House Index →	Waffle House Index →	Waffle House Index →	Waffle House Index →															
Semantic Scholar →	Yahoo! Finance →	The New York Times →	The New York Times →															
White paper →	Bitcoin →	Peer →	Blockchain.com →															
Satoshi Nakamoto ←	Satoshi Nakamoto →	Satoshi Nakamoto →	Satoshi Nakamoto →															
IDS	Single Solution	<p>Found 1 paths from Waffle House Index to Satoshi Nakamoto in 170.7500789 seconds!</p> <p>Articles Checked: 266 Articles Traversed: 88472</p> <p>Connecting Graphs</p>  <p>Individual Paths</p> <table border="1"> <tbody> <tr><td>Waffle House Index →</td></tr> <tr><td>Yahoo! Finance →</td></tr> <tr><td>Bitcoin →</td></tr> <tr><td>Satoshi Nakamoto →</td></tr> </tbody> </table>	Waffle House Index →	Yahoo! Finance →	Bitcoin →	Satoshi Nakamoto →												
Waffle House Index →																		
Yahoo! Finance →																		
Bitcoin →																		
Satoshi Nakamoto →																		

		<p>Found 1 paths from Waffle House Index to Satoshi Nakamoto in 3060.6504342 seconds!</p> <p>Articles Checked: 4962 Articles Traversed: 2285702</p> <hr/> <p>Connecting Graphs</p>  <p>Drag to pan. Scroll to zoom.</p> <hr/> <p>Individual Paths</p> <ul style="list-style-type: none"> Waffle House Index → Yahoo Finance → Bitcoin → Satoshi Nakamoto →
IDS	Multi Solution	

TEST 4



LEMANS PEDIA

Find the shortest paths from

Chess

↔

Human

Algorithm Type

BFS

IDS

Solution Type

SINGLE SOLUTION

MULTI SOLUTION

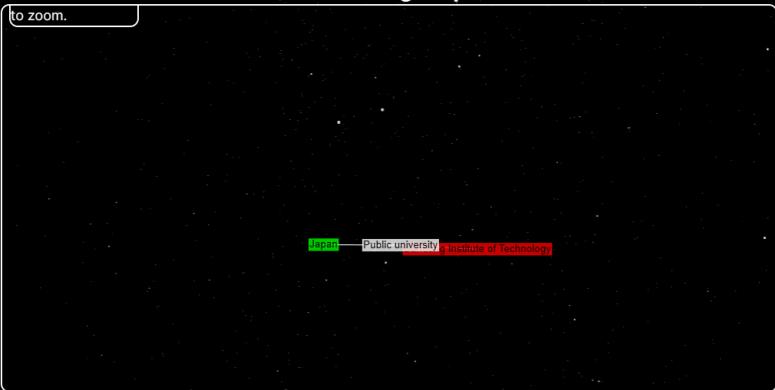
SUBMIT!

ALGORITMA	SOLUSI	HASIL
------------------	---------------	--------------

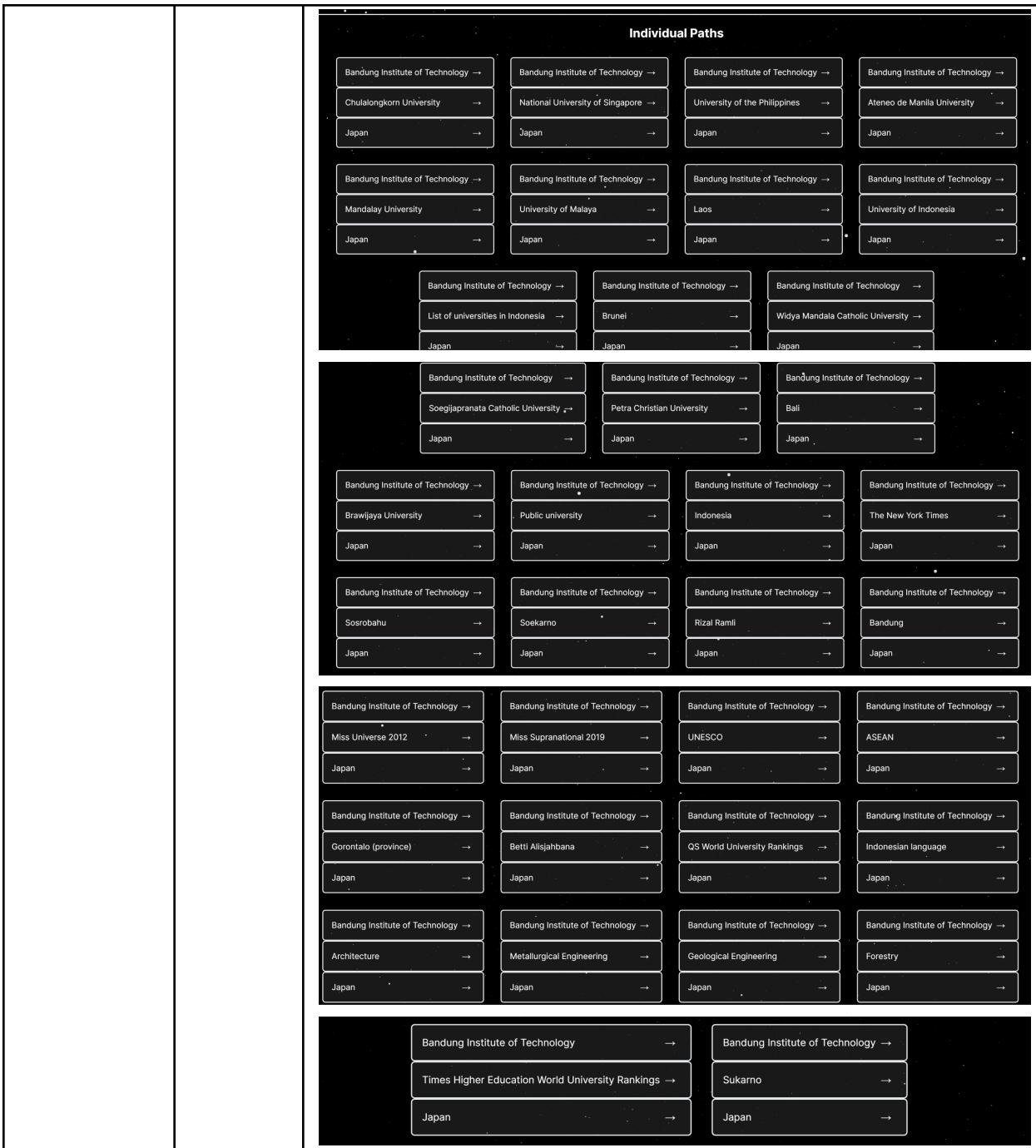
	BFS	Single Solution	<p>Found 1 paths from Chess to Human in 15.96374843 seconds!</p> <p>Articles Checked: 902 Articles Traversed: 46218</p> <p>Connecting Graphs</p> <pre> graph LR Human[Human] --- WE[War elephant] WE --- Chess[Chess] </pre> <p>Individual Paths</p> <ul style="list-style-type: none"> Chess → War elephant → Human →
	BFS	Multi Solution	<p>Found 3 paths from Chess to Human in 31.591537596 seconds!</p> <p>Articles Checked: 3062 Articles Traversed: 97322</p> <p>Connecting Graphs</p> <pre> graph LR Chess[Chess] --- WE[War elephant] Chess --- P[Perception] WE --- Human[Human] P --- Human WE --- A[Automation] </pre>

		<p style="text-align: center;">Individual Paths</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>Chess →</td><td>Chess →</td><td>Chess →</td></tr> <tr><td>War elephant →</td><td>Automaton →</td><td>Perception →</td></tr> <tr><td>Human →</td><td>Human →</td><td>Human →</td></tr> </table>	Chess →	Chess →	Chess →	War elephant →	Automaton →	Perception →	Human →	Human →	Human →
Chess →	Chess →	Chess →									
War elephant →	Automaton →	Perception →									
Human →	Human →	Human →									
IDS	Single Solution	<p>Found 1 paths from Chess to Human in 25.3938479 seconds!</p> <p>Articles Checked: 708 Articles Traversed: 189240</p> <p style="text-align: center;">Connecting Graphs</p> <p>Drag to pan. Scroll to zoom.</p>									
IDS	Multi Solution	<p>Found 3 paths from Chess to Human in 598.8050239 seconds!</p> <p>Articles Checked: 1022 Articles Traversed: 347085</p> <p style="text-align: center;">Connecting Graphs</p> <p>Drag to pan. Scroll to zoom.</p> <p style="text-align: center;">Individual Paths</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>Chess →</td><td>Chess →</td><td>Chess →</td></tr> <tr><td>Automaton →</td><td>Perception →</td><td>War elephant →</td></tr> <tr><td>Human →</td><td>Human →</td><td>Human →</td></tr> </table>	Chess →	Chess →	Chess →	Automaton →	Perception →	War elephant →	Human →	Human →	Human →
Chess →	Chess →	Chess →									
Automaton →	Perception →	War elephant →									
Human →	Human →	Human →									

TEST 5

 Home Wiki Race About Us		
ALGORITMA	SOLUSI	HASIL
BFS	Single Solution	<p>Found 1 paths from Bandung Institute of Technology to Japan in 3.049715627 seconds!</p> <p>Articles Checked: 302 Articles Traversed: 29768</p> <p>Connecting Graphs</p>  <p>to zoom.</p> <p>Japan → Public university → Institute of Technology</p> <p>Individual Paths</p> <pre> Bandung Institute of Technology → . Public university → . Japan → . </pre>

	BFS	<p>Multi Solution</p>
	IDS	<p>Single Solution</p>
	IDS	<p>Multi Solution</p>



4.5 Analisis Algoritma

Dalam permainan WikiRace, tujuannya adalah untuk mencapai suatu halaman Wikipedia tertentu dari halaman awal dalam waktu sesingkat mungkin. Di sini, kita bisa menganalisis dua metode pencarian graf yang populer, yaitu Breadth-First Search (BFS) dan Iterative Deepening Search (IDS), untuk melihat mana yang lebih efektif dalam konteks permainan ini.

4.5.1 Algoritma BFS

BFS bekerja dengan mencari semua node pada kedalaman tertentu sebelum berpindah ke kedalaman berikutnya. Ini dilakukan dengan menggunakan struktur data antrian untuk menyimpan semua simpul yang harus dieksplorasi.

4.5.1.1 Kelebihan

1. Menjamin Temuan Jalur Terpendek: BFS akan menemukan jalur terpendek ke tujuan jika ada karena selalu memperluas node pada kedalaman terendah terlebih dahulu.
2. Lebih Cepat Untuk Graf yang Dangkal: Jika halaman tujuan tidak terlalu jauh dari titik awal

4.5.1.2 Kekurangan

1. Penggunaan Memori yang Tinggi: Karena menyimpan semua simpul di sebuah level sebelum beralih ke level berikutnya, penggunaan memorinya dapat menjadi sangat besar, terutama pada graf dengan faktor percabangan yang tinggi.

4.5.2 Algoritma IDS

IDS menggabungkan kedalaman pencarian pertama (DFS) dengan metode yang bertingkat. Ini dimulai dengan kedalaman terbatas dan meningkatkan batas ini dengan satu setiap kali siklus lengkap dari pencarian telah selesai.

4.5.2.1 Kelebihan

1. Hemat Memori: IDS menggunakan memori yang lebih sedikit daripada BFS karena hanya menyimpan satu jalur dari akar ke simpul daun pada satu waktu, bukan seluruh lapisan dari pohon.
2. Fleksibel: Karena IDS mengulang dengan kedalaman yang bertambah, ia memiliki fleksibilitas dalam mencari jalur yang lebih panjang tanpa membutuhkan penyimpanan ekstra besar di awal.

4.5.2.2 Kekurangan

1. Lambat untuk Graf Luas: IDS mungkin menjadi sangat lambat terutama jika solusi berada pada kedalaman yang lebih dalam karena harus mengulang semua pencarian di kedalaman sebelumnya, membuatnya redundant dan memakan waktu lebih banyak.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar 2 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2023/2024 ini, kami diminta untuk membuat program permainan Wiki Race berbasis website. Persoalan yang harus diselesaikan dalam program permainan Wiki Race adalah mencari rute terpendek dari alamat awal hingga alamat tujuan artikel. Kami diminta untuk menyelesaikan persoalan tersebut menggunakan algoritma *Breadth First Search* (BFS) dan *Iterative Deepening Search* (IDS). Algoritma program permainan dibuat dengan menggunakan bahasa pemrograman GO baik untuk DFS dan IDS. Bahasa pemrograman GO kami gunakan juga sebagai backend dari website untuk menerima response dari frontend dan passing hasil program ke dalam JSON (Javascript Object Notation). Dalam pengembangan website, kami menggunakan framework next JS yang berbasis bahasa pemrograman JavaScript. Program kami juga dapat mencari multi solution rute terpendek dari alamat awal hingga alamat tujuan. Sebagai ilustrasi, semakin dalam depth maka akan semakin banyak juga kemungkinan solusi rute terpendek yang ada.

5.2 Saran

Selama proses penggerjaan tugas besar 2 IF2211 Strategi Algoritma, kami sempat mengalami kendala saat mengakses jumlah situs wikipedia terlalu banyak dalam rentang waktu tertentu. Ketika kita melebihi limit akses situs wikipedia maka kita akan dihentikan aksesnya untuk sementara waktu. Hal ini dapat kami atasi dengan menerapkan batasan atau *constraint* saat melakukan scraping link wikipedia. Karena program dijalankan dengan menggunakan link scraping maka koneksi dengan internet sangat berpengaruh terhadap kecepatan program. Semakin baik koneksi internet maka akan semakin cepat juga program dapat memberikan hasil rute terpendek. Saran untuk pengembangan selanjutnya, program ini dapat dijalankan tanpa memerlukan koneksi dengan internet. Pengembangan ini sangat layak untuk dilakukan karena kita dapat melihat kinerja algoritma secara murni tanpa batasan koneksi internet.

DAFTAR PUSTAKA

- [1] “Go Programming Language (Introduction) - GeeksforGeeks,” *GeeksforGeeks*, Apr. 25, 2018. <https://www.geeksforgeeks.org/go-programming-language-introduction/>
- [2] K. Kelche, “A Guide to Closure Functions in Go,” *www.kelche.co*, Mar. 10, 2023. <https://www.kelche.co/blog/go/closure/> (accessed Apr. 27, 2024).
- [3] W. C. Example, “Concurrency and Goroutines - Learn Parallelism in Golang,” *golang.withcodeexample.com*.
<https://golang.withcodeexample.com/blog/concurrency-goroutines-mastering-parallelism-go/> (accessed Apr. 26, 2024).
- [4] “Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS) - GeeksforGeeks,” *GeeksforGeeks*, May 19, 2016. <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-search-ddfs/>
- [5] M. Simic, “Iterative Deepening vs. Depth-First Search | Baeldung on Computer Science,” *www.baeldung.com*, Sep. 27, 2021. <https://www.baeldung.com/cs/iterative-deepening-vs-depth-first-search>
- [6] “Getting Started – React,” *reactjs.org*. <https://reactjs.org/docs/getting-started.html>
- [7] “Documentation - Tailwind CSS,” *tailwindcss.com*. <https://tailwindcss.com/docs>

Repository GitHub

Link Repository : https://github.com/alandmprtma/Tubes2_MarthenGantenk

Penjelasan Algoritma dan Bonus

Link Youtube : <https://youtu.be/rjMq1VCH3G8>