

LAPORAN TUGAS BESAR II
IF2211 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Disusun oleh:

“WikiWiggy”

Erdianti Wiga Putri Andini	13522053
Shazya Audrea Taufik	13522063
Yudi Kurniawan	10023634

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I DESKRIPSI TUGAS.....	4
1.1 Deskripsi Tugas.....	4
1.2 Spesifikasi.....	4
BAB II LANDASAN TEORI.....	5
2.1 Traversal Graf.....	5
2.2 Breadth First Search (BFS).....	5
2.3 Iterative Deepening Search (IDS).....	7
2.4 Pengembangan Website.....	8
2.4.1. Pengembangan Web Front-End.....	9
2.4.2. Pengembangan Web Back-End.....	9
2.4.3. Pengembangan Web Full-Stack.....	10
2.5 Programming Language.....	11
2.5.1 Go Language.....	11
2.5.2 ReactJS dan JSX.....	11
2.5.3 ChakraUI.....	12
2.5.4 Gin API.....	12
2.5.5 Docker.....	13
BAB III ANALISIS PEMECAHAN MASALAH.....	14
3.1 Langkah-langkah Pemecahan Masalah.....	14
3.2 Mapping Persoalan sebagai Elemen BFS dan IDS.....	14
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web.....	15
3.4 Contoh Ilustrasi Kasus.....	16
3.4.1 BFS.....	16
3.4.2 IDS.....	18
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	20
4.1 Implementasi Algoritma BFS.....	20
4.1.1 Struktur Data.....	20
4.1.2 Fungsi dan Prosedur.....	21
4.2 Implementasi Algoritma IDS.....	28
4.2.1 Struktur Data.....	28
4.2.2 Fungsi dan Prosedur.....	29
4.3 Tata Cara Penggunaan Program.....	38
4.3.1 Laman Utama.....	38
4.3.2 Laman Profil.....	40
4.3.3 Laman Cara Penggunaan.....	41
4.3.4 Laman BFS.....	42
4.3.5 Laman IDS.....	44

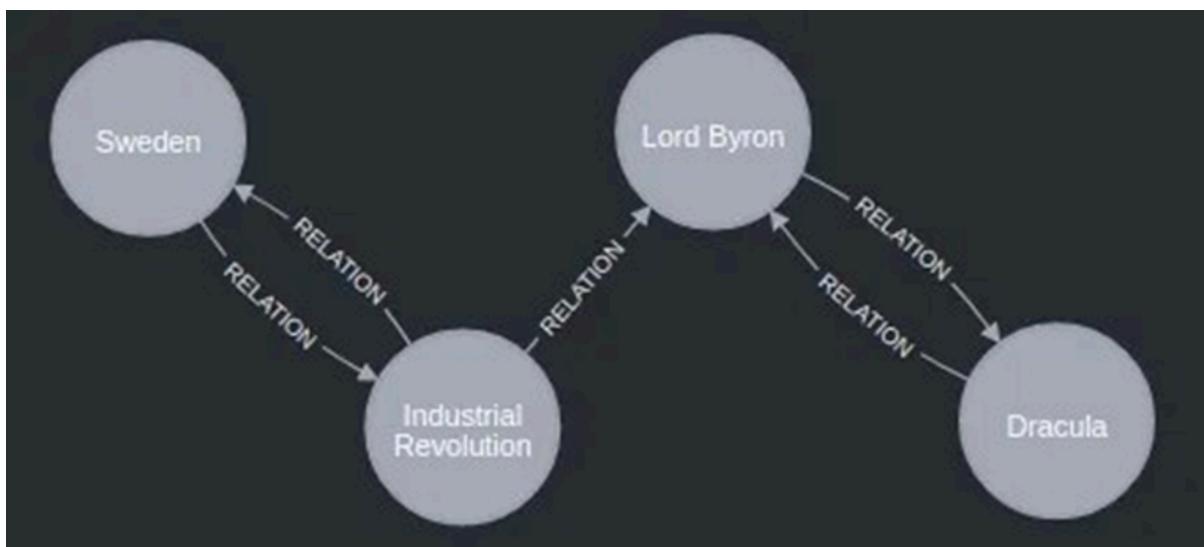
4.4 Hasil Pengujian.....	46
4.4.1 Test Case 1.....	46
4.4.2 Test Case 2.....	47
4.4.3 Test Case 3.....	48
4.4.4 Test Case 4.....	49
4.4.5 Test Case 5.....	50
4.4.6 Test Case 6.....	51
4.4.7 Test Case 7.....	52
4.4.8 Test Case 8.....	53
4.4.9 Test Case 9.....	53
4.4.10 Test Case 10.....	54
4.4.11 Test Case 11.....	54
4.5 Analisis Hasil Pengujian.....	55
BAB V KESIMPULAN DAN SARAN.....	56
5.1 Kesimpulan.....	56
5.2 Saran.....	56
LAMPIRAN.....	57
DAFTAR PUSTAKA.....	58

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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.



Gambar 1. Ilustrasi Graf WikiRace

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

1.2 Spesifikasi

Pada tugas ini kami diminta untuk membuat program sederhana dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace. Masukan program berupa jenis algoritma, judul artikel awal, dan judul artikel akhir. Keluaran dari program adalah jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (terpendek), dan waktu pencarian. Program berbasis web (perlu ada *frontend* dan *backend*). Pencarian rute terpendek perlu dilakukan dibawah 5 menit.

BAB II

LANDASAN TEORI

2.1 Traversal Graf

Algoritma traversal graf adalah algoritma yang mengunjungi simpul-simpul di dalam graf dengan cara yang sistematik. Traversal graf artinya melakukan pencarian solusi persoalan yang direpresentasikan dengan graf. Algoritma ini terbagi menjadi dua yaitu pencarian melebar (*breadth first search/BFS*) dan pencarian mendalam (*depth first search/DFS*).

Terdapat dua jenis algoritma pencarian solusi berbasis graf sebagai berikut:

- a. Tanpa informasi (*uninformed/blind search*)

Tidak ada informasi tambahan yang disediakan pada algoritma. Contohnya adalah *Depth First Search* (DFS), *Breadth First Search* (BFS), *Depth Limited Search* (DLS), *Iterative Deepening Search* (IDS), dan *Uniform Cost Search*.

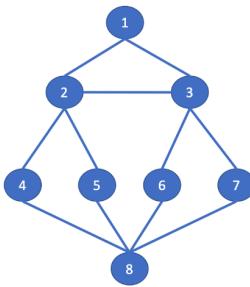
- b. Dengan informasi (*informed search*)

Pencarian yang berbasis heuristik dan mengetahui *non-goal state* yang ‘lebih menjanjikan’ daripada yang lain. Contohnya adalah Best First Search dan A*.

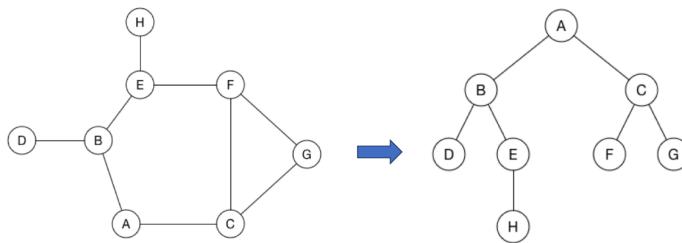
Dalam proses pencarian solusi, terdapat dua pendekatan yaitu menggunakan graf statis dan graf dinamis. Graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan. Graf disini direpresentasikan sebagai struktur data. Sedangkan graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan. Graf tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi.

2.2 Breadth First Search (BFS)

Algoritma Breadth First Search adalah algoritma yang melakukan pencarian secara melebar yang mengunjungi simpul secara preorder yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu. Algoritmanya dilakukan dengan mengunjungi simpul v , kemudian, mengunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu. Serta mengunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya. Pada gambar di bawa, urutan kunjungan menurut BFS adalah 1, 2, 3, 4, 5, 6, 7, 8.



Traversasi graf juga bisa digambarkan sebagai pohon. Contohnya adalah sebagai berikut:



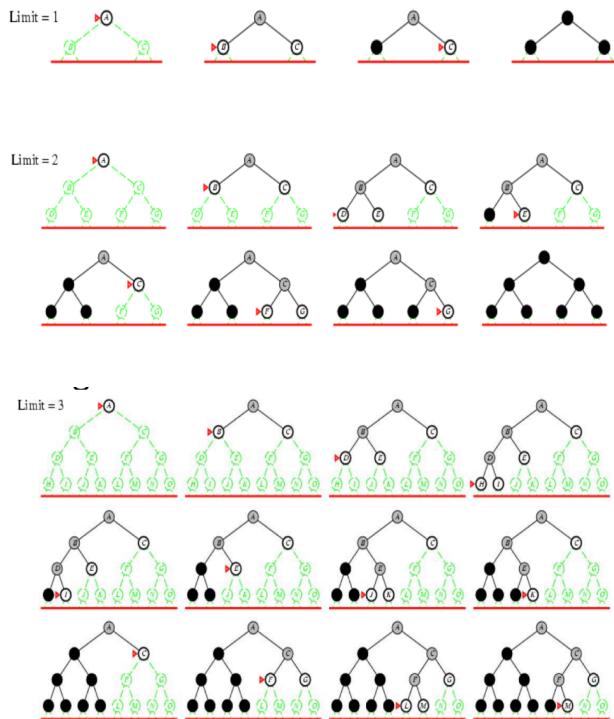
Algoritma ini memerlukan sebuah matriks ketetanggaan dan sebuah *queue* q untuk menyimpan simpul yang telah dikunjungi. Simpul-simpul ini diperlukan sebagai acuan untuk mengunjungi simpul-simpul yang bertetanggaan dengannya. Tiap simpul yang telah dikunjungi masuk ke dalam *queue* hanya satu kali. Algoritma ini juga membutuhkan table Boolean untuk menyimpan simpul yang telah dikunjungi sehingga tidak ada simpul yang dikunjungi lebih dari satu kali.

Langkah pada metode pencarian ini adalah:

- Masukkan simpul ujung (akar) ke dalam *queue*.
- Ambil simpul dari awal *queue*, lalu cek apakah simpul merupakan solusi.
- Jika simpul merupakan solusi, pencarian selesai dan hasil dikembalikan.
- Jika simpul bukan solusi, masukkan seluruh simpul yang bertetangga dengan simpul tersebut (simpul anak) ke dalam *queue*.
- Jika *queue* kosong dan setiap simpul sudah dicek, pencarian selesai dan mengembalikan hasil solusi tidak ditemukan.
- Ulangi pencarian dari langkah kedua.

2.3 Iterative Deepening Search (IDS)

Algoritma Iterative Deepening Search (IDS) adalah metode yang menggabungkan kelebihan BFS (optimal) dengan kelebihan DFS (*space complexity* rendah). IDS adalah pencarian ruang strategi di mana pencarian Depth Limited Search (DLS) dijalankan berulang kali, meningkatkan batas kedalaman dengan setiap iterasi sampai mencapai tujuan. Metode pencarian pada IDS menggunakan metode pencarian pada DFS, yaitu pencarian pada sebuah pohon dengan menelusuri satu cabang sebuah pohon sampai menemukan solusi. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking*. Akan tetapi, dalam metode pencarian IDS, pencarian mencoba menemukan batas kedalaman yang terbaik dengan terus menelusuri batas kedalaman satu per satu, pencarian dilakukan secara iteratif dimulai dari batasan level 0. Jika solusi belum ditemukan, maka dilakukan iterasi berikutnya dengan batasan level 1, demikian seterusnya sampai menemukan solusi. IDS melakukan pencarian berulang-ulang dengan menambah batas kedalaman setiap mengulang pencarian, dan akhirnya menemukan solusi pada penelusuran dengan batas kedalaman tertentu. Solusi ditemukan pada percabangan kiri atau kanan. Pencarian ini akan menemukan solusi lebih cepat daripada penelusuran pada DFS, jika solusi tidak berada pada node yang dalam.



Langkah pada metode pencarian ini adalah:

- Inisialisasi: Tetapkan kedalaman pencarian awal.
- Pencarian Bertingkat: Lakukan pencarian dalam kedalaman terbatas (DLS) dengan batasan kedalaman yang ditetapkan pada langkah sebelumnya. Jika solusi tidak ditemukan pada tingkat ini, tingkatkan batasan kedalaman dan ulangi pencarian. Proses ini diulangi hingga solusi ditemukan.
- Cek Solusi: Jika solusi telah ditemukan, kembalikan solusi tersebut.

IDS dijamin menemukan solusi optimal karena secara bertahap meningkatkan kedalaman pencarian hingga solusi ditemukan. Hal ini mirip dengan BFS yang menjamin solusi optimal, namun tanpa memerlukan penyimpanan yang sebesar BFS.

2.4 Pengembangan Website

Pengembangan situs web adalah istilah yang merujuk pada praktik membangun, membuat, dan merawat situs web. Berdasarkan definisi dari sumber Techterms, ini mencakup aspek seperti desain web, penerbitan web, pemrograman web, dan manajemen basis data. Terkait dengan fungsi utamanya, pengembangan web tidak hanya berfokus pada desain situs web, melainkan terutama berkaitan dengan bagian pemrograman dan pengkodean, yang merupakan alasan utama berfungsinya situs web.

Jenis pengembangan web terkait prosesnya, secara luas dapat didefinisikan menjadi tiga jenis, yaitu sebagai berikut:

2.4.1. Pengembangan Web *Front-End*

Tipe pengembangan web ini berkontribusi dalam mengubah informasi dan data yang tersedia menjadi antarmuka pengguna grafis menggunakan CSS, HTML, dan JavaScript. Hal ini bertujuan agar pengguna dapat berinteraksi dengan antarmuka yang disediakan dengan mudah. Tipe ini melibatkan semua elemen yang dapat secara langsung diamati dan dirasakan, termasuk warna, teks, gambar, tombol, dan unsur visual lainnya.

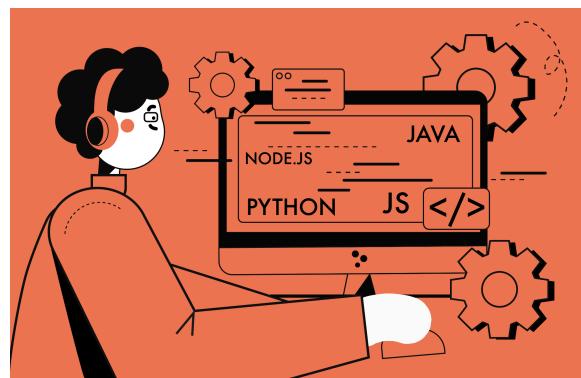


Dalam hal tujuan dan fungsi utamanya, front-end developer memiliki beberapa tanggung jawab. Ini mencakup memberikan prioritas kepada *User Experience* (UX), mewujudkan konsep menggunakan HTML, CSS, dan JavaScript, serta menghasilkan serta merawat *User Interface* (UI) dari situs web dan aplikasi web. Selain itu, mereka bertanggung jawab untuk menciptakan alat-alat yang meningkatkan interaksi pengguna dengan situs di berbagai browser, menerapkan desain yang responsif untuk situs seluler, menjaga manajemen alur kerja perangkat lunak, mengikuti praktik terbaik dalam *Search Engine Optimization* (SEO), dan melakukan pengujian fungsi situs serta memperbaiki bug yang muncul.

2.4.2. Pengembangan Web *Back-End*

Komponen back-end dari setiap halaman web tidak dapat terlihat atau diakses oleh pengguna. Meskipun tak terlihat, bagian ini merupakan tulang punggung atau inti dari suatu situs web. Meskipun tidak terlihat oleh pengguna, namun bagian ini memiliki peran krusial dalam memastikan fungsi keseluruhan situs web. Fungsi utamanya adalah menyimpan dan mengelola data serta menjamin kinerja dan fungsi

yang tepat dari semua elemen yang terlihat di bagian depan atau sisi klien (client side). Bagian back-end berinteraksi dengan bagian depan halaman web dengan mengirim dan menerima data, yang nantinya ditampilkan secara langsung di situs web. Sebagai contoh, ketika pengguna menginput data, mengisi formulir, atau melakukan pembelian, browser mengirimkan permintaan tersebut ke bagian back-end atau sisi server, yang kemudian merespons dengan mengirimkan data sebagai kode front-end untuk halaman web, sehingga dapat dipahami, ditafsirkan, dan ditampilkan kepada pengguna.



Perlu dicatat bahwa dalam pengembangan back-end, terdapat dua pendekatan utama, yaitu Object Oriented Programming (OOP) dan Functional Programming. OOP, sebagai jenis pertama, memusatkan pada penciptaan objek dan mengeksekusi pernyataan dalam urutan tertentu. Beberapa bahasa pemrograman OOP yang umum digunakan antara lain Java, .NET, Python, dan Ruby. Sementara itu, Functional Programming, sebagai jenis kedua, menggunakan pendekatan yang lebih berbasis "action atau aksi" dan memungkinkan eksekusi pernyataan dalam urutan apa pun. Biasanya digunakan dalam ilmu data atau data science, bahasa pemrograman fungsional melibatkan SQL, F#, dan R sebagai contoh yang populer.

2.4.3. Pengembangan Web *Full-Stack*

Full stack development merupakan pengembangan yang menggabungkan front-end dan back-end halaman web. Proses ini melibatkan pembuatan grafis dan desain halaman web, serta pengelolaan database untuk menyusun dan menyimpan data. Lebih spesifiknya, *full stack* developer bisa bekerja dengan javascript, PHP, java, database (backend) dan juga bisa mengkonversi desain ke dalam kode pemrograman seperti HTML, CSS, XML (front end).

2.5 Programming Language

2.5.1 Go Language



Bahasa Go atau Golang adalah bahasa pemrograman open-source yang dikembangkan oleh Google. Berguna untuk mengembangkan web, layanan cloud dan jaringan, serta jenis perangkat lunak lainnya. Golang dibuat statis dan eksplisit dari segi arsitektur serta proses. Golang sering digunakan sebagai *backend* dengan menggunakan Gin sebagai *framework*. Bahasa ini juga dapat bekerja dengan menggunakan sebuah proses ringan dan efisiensi lebih lanjut atau disebut juga dengan *goroutine*.

2.5.2 ReactJS dan JSX

React JS adalah framework JavaScript sumber terbuka yang dikembangkan oleh Facebook, dirancang untuk memudahkan pembuatan user interface yang interaktif untuk aplikasi web. Dengan menggunakan konsep komponen yang dapat digunakan kembali—mirip blok Lego—React memungkinkan pengembang untuk membangun aplikasi dengan lebih efisien, mengurangi jumlah kode yang diperlukan dibandingkan dengan JavaScript biasa. Fokus utama React adalah pada lapisan tampilan, mirip dengan komponen 'View' dalam model-view-controller (MVC), memungkinkan rendering yang efektif dan efisien. Ini memfasilitasi pembagian UI yang kompleks menjadi komponen individu yang lebih kecil, mempercepat rendering halaman web dan meningkatkan responsivitas aplikasi web.



JSX adalah ekstensi sintaks dari JavaScript yang memudahkan modifikasi Document Object Model (DOM) dengan menggunakan kode yang mirip dengan

HTML. DOM sendiri adalah Application Programming Interface (API) yang bertugas menyusun struktur halaman web. Dengan JSX, pengembang dapat lebih mudah menambahkan konten dinamis ke halaman karena memungkinkan penyisipan kode HTML-like langsung ke dalam DOM. Meski mirip HTML, JSX sejatinya berfungsi seperti JavaScript dan kompatibel dengan berbagai browser, seperti Chrome dan Mozilla Firefox, sehingga memperluas penggunaannya dalam pengembangan web.

2.5.3 ChakraUI



Chakra UI adalah pustaka komponen yang memungkinkan untuk membuat aplikasi React yang mudah diakses dan indah dengan lebih sedikit kode dan lebih cepat. Pustaka ini mengikuti standar WAI-ARIA, menawarkan komponen yang dapat diubah tema dan disusun ulang, serta mendukung mode tampilan terang dan gelap.

2.5.4 Gin API



Gin adalah sebuah HTTP web framework yang ditulis dalam bahasa Go (Golang). Gin merupakan sebuah *framework* yang didesain dengan mudah dan simpel. *Framework* ini sangat cocok untuk pemula dan menggunakan *router* HTTP untuk menangani lalu lintas Golang. Gin dapat memudahkan penggabungan atau *passing data frontend* dengan *backend* dengan menggunakan json.

2.5.5 Docker



Docker adalah platform perangkat lunak yang memungkinkan *programmer* untuk membuat, menguji, dan menerapkan aplikasi dengan cepat. Docker mengemas perangkat lunak ke dalam unit standar yang disebut *container* yang memiliki semua yang diperlukan perangkat lunak agar dapat berfungsi termasuk pustaka, alat sistem, kode, dan waktu proses. Dengan menggunakan Docker, Anda dapat dengan cepat menerapkan dan menskalakan aplikasi ke lingkungan apa pun.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Permasalahan yang akan diselesaikan pada Tugas Besar kali ini adalah pencarian rute terpendek dari suatu artikel asal ke suatu artikel lainnya. Pengguna dapat memasukkan masukan artikel asal dan artikel tujuan pada kolom masukan di website, baik itu halaman BFS maupun IDS. Untuk menyelesaikan permasalahan ini, kami menggunakan algoritma *Breadth Depth Search* (BFS) dan juga *Iterative Deepening Search* (IDS).

Dengan menggunakan input yang berupa judul artikel, kedua inputan akan dikonversi agar menjadi dalam bentuk *link*. Kemudian dengan menggunakan konsep web scraping, akan dibuat graf dimana setiap *link* yang ditemukan pada suatu artikel akan dijadikan simpul dan simpul tersebut akan memiliki simpul tetangga. Graf ini akan digunakan dalam pencarian dengan implementasi algoritma graf traversal seperti BFS dan IDS.

BFS menggunakan metode iteratif dan IDS menggunakan metode rekursif. Pada BFS, pencarian akan dilakukan per level. Sementara, pada IDS pencarian akan menggunakan konsep DLS dimana dilakukan pencarian untuk dari *maximum depth* dari 0 hingga level tertentu dimana simpul yang dicari telah ditemukan.

Program diimplementasikan menggunakan website dengan *frontend* menggunakan framework ReactJS dan chakraUI dengan bahasa JavaScript Syntax Extension, serta backend menggunakan bahasa Go.

3.2 Mapping Persoalan sebagai Elemen BFS dan IDS

Dari persoalan yang ingin dipecahkan dengan menggunakan algoritma BFS dan IDS, terdapat elemen yang bisa dipetakan, yaitu link artikel sebagai simpul. Simpul dikatakan bertetangga jika mereka berasal dari link yang sama. Ketetanggaan disimpan dalam *adjacency map*.

Mapping elemen-elemen dari algoritma yang akan dibuat berdasarkan pemecahan masalah tersebut adalah :

- a. Start article : Simpul graf
- b. Goal article : Simpul graf
- c. Article paths : Simpul graf

Dengan elemen-elemen tersebut, rancangan algoritma BFS dan DFS untuk menyelesaikan permasalahan ini yaitu sebagai berikut :

a. Algoritma BFS

Algoritma ini memanfaatkan struktur data *queue* untuk menyimpan node-node yang akan dieksplorasi selanjutnya dan menggunakan pendekatan iteratif untuk melalui node-node tetangga yang akan dieksplorasi. Node *start article* akan dimasukkan pertama kali ke dalam *queue* sebagai node yang akan dieksplorasi. Selanjutnya, node-node yang berhubungan langsung dengan node yang sedang dieksplorasi akan ditambahkan ke dalam *queue* nodes. Setelah iterasi pertama dari node *start article* selesai, eksplorasi berlanjut ke node tetangga dengan melakukan *dequeue* dari *queue* nodes. Node yang dikeluarkan dari *queue* kemudian diperiksa apakah itu merupakan *goal article* atau tidak. Jika node yang sedang dieksplorasi bukan *goal article*, maka langkah penambahan tetangga ke *queue* dan pengeluaran node dari *queue* akan diulang sampai *goal article* ditemukan.

b. Algoritma IDS

Algoritma ini memanfaatkan struktur data *queue* untuk menyimpan node-node yang akan dieksplorasi selanjutnya dan menggunakan pendekatan iteratif menggunakan algoritma DLS (Depth Limited Search) untuk melalui node-node tetangga yang akan dieksplorasi. Node *start article* akan dimasukkan pertama kali ke dalam *queue* sebagai node yang akan dieksplorasi. Selanjutnya, node-node yang berhubungan langsung dengan node yang sedang dieksplorasi akan ditambahkan ke dalam *queue* nodes dengan kedalaman maksimum bertambah satu. Jika pada kedalaman pertama tidak ditemukan *goal article*, maka pencarian akan diulang dari *start article* namun dengan kedalaman maksimum bertambah satu lagi. Proses ini diulang hingga ditemukan *goal article*.

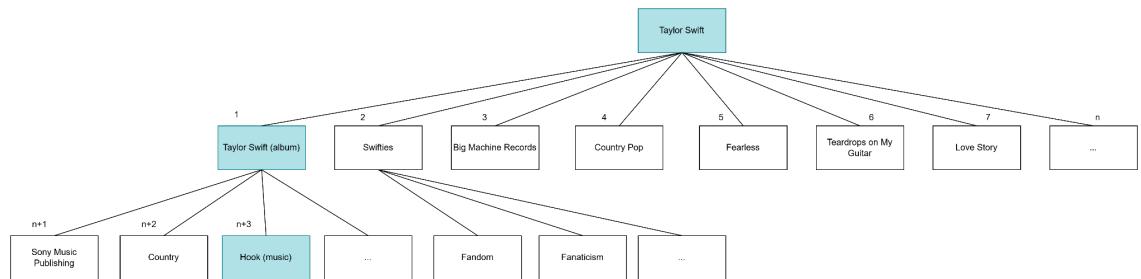
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web

Website dibangun dengan *frontend* menggunakan *framework* ReactJS dan chakraUI dengan bahasa JavaScript Syntax Extension. Fitur yang terdapat pada website ini adalah laman utama yang digunakan untuk memilih algoritma yang ingin digunakan untuk permainan WikiRace. Lalu ada pula laman profil, laman cara penggunaan, serta laman masing-masing algoritma. Pada masing-masing laman algoritma, pengguna harus memasukkan *start article* dan *goal article* yang akan dipassing ke *backend* yang

menggunakan bahasa Go melalui *framework* Gin yang digunakan untuk membuat server web. Pertama, server mendefinisikan endpoint HTTP GET di *path* akar ("") untuk menerima permintaan dengan parameter kueri *start article* dan *goal article*. Jika salah satu judul tidak disertakan, server langsung merespons dengan status HTTP 400 (*bad response*). Setelah input valid diterima, backend memulai beberapa operasi yang terdapat pada file *backend*. *Backend* menggunakan mekanisme *scraping* bersamaan di mana beberapa *goroutine* mengambil dan memproses tautan dari URL saat ini, menambahkan tautan yang belum dikunjungi ke *queue* dan menggunakan pencarian *path* menggunakan algoritma BFS atau IDS. Setelah *path* ditemukan atau *timeout* terjadi, *backend* mengubah *path* dari URL menjadi judul artikel, menghitung waktu yang diperlukan untuk menemukan *path*, total node (artikel) yang dikunjungi, dan panjang *path* hasil, serta mengirim data ini kembali ke *client* sebagai objek JSON menggunakan `c.JSON`.

3.4 Contoh Ilustrasi Kasus

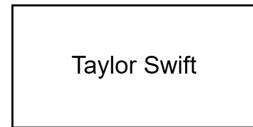
3.4.1 BFS



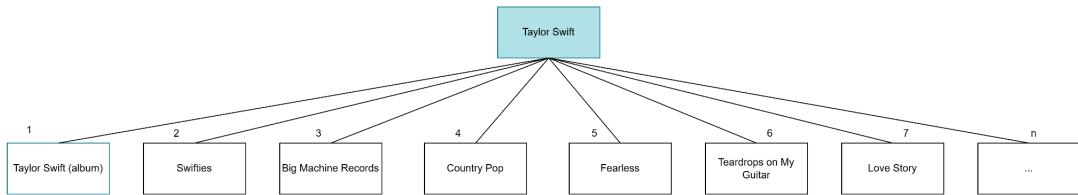
Itr n+2	n+2	n+3	T	T	T	T	T	T	T	T	T	T	T	T	T	T
Itr n+3	n+3	-	T	T	T	T	T	T	T	T	T	T	T	T	T	T

3.4.2 IDS

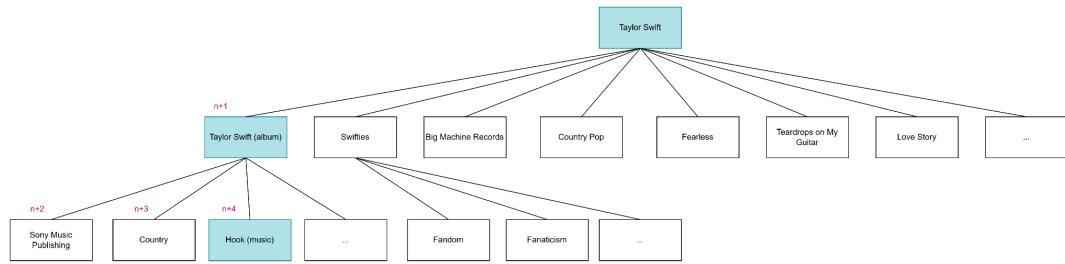
Kedalaman = 0: Taylor Swift: cutoff



Kedalaman = 1: Taylor Swift (album)_{Taylor Swift}, Swifties_{Taylor Swift}, Big Machine Records_{Taylor Swift}, Country Pop_{Taylor Swift}, Fearless_{Taylor Swift}, Teardrops on My Guitar_{Taylor Swift}, Love Story_{Taylor Swift}, ... → Taylor Swift (album)_{Taylor Swift : cutoff}, Swifties_{Taylor Swift : cutoff}, Big Machine Records_{Taylor Swift : cutoff}, Country Pop_{Taylor Swift : cutoff}, Fearless_{Taylor Swift : cutoff}, Teardrops on My Guitar_{Taylor Swift : cutoff}, Love Story_{Taylor Swift : cutoff}, ... : cutoff



Kedalaman = 2: Taylor Swift (album)_{Taylor Swift}, Swifties_{Taylor Swift}, Big Machine Records_{Taylor Swift}, Country Pop_{Taylor Swift}, Fearless_{Taylor Swift}, Teardrops on My Guitar_{Taylor Swift}, Love Story_{Taylor Swift}, ... → Sony Music Publishing_{Taylor Swift (album) | Taylor Swift}, Swifties_{Taylor Swift}, Big Machine Records_{Taylor Swift}, Country Pop_{Taylor Swift}, Fearless_{Taylor Swift}, Teardrops on My Guitar_{Taylor Swift}, Love Story_{Taylor Swift}, ... → Sony Music Publishing_{Taylor Swift (album) | Taylor Swift : cutoff} → Country_{Taylor Swift (album) | Taylor Swift}, Swifties_{Taylor Swift}, Big Machine Records_{Taylor Swift}, Country Pop_{Taylor Swift}, Fearless_{Taylor Swift}, Teardrops on My Guitar_{Taylor Swift}, Love Story_{Taylor Swift}, ... → Sony Music Publishing_{Taylor Swift (album) | Taylor Swift : cutoff} → **Hook (music)**_{Taylor Swift (album) | Taylor Swift}, Swifties_{Taylor Swift}, Big Machine Records_{Taylor Swift}, Country Pop_{Taylor Swift}, Fearless_{Taylor Swift}, Teardrops on My Guitar_{Taylor Swift}, Love Story_{Taylor Swift}, ... → Stop: Taylor Swift → Taylor Swift (album) → Hook (music)



BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma BFS

4.1.1 Struktur Data

Struktur Data	Penjelasan
Graph	Struktur data yang dibentuk dengan menggunakan struct dan terdiri dari beberapa elemen antara lain `nodes`, `adjList`, dan `visitedCount`. Elemen `nodes` adalah slice dari pointer ke objek Node, `adjList` yang merupakan map yang memetakan suatu link ke daftar link yang terhubung secara langsung dengan link tersebut. Dalam representasi ini, graf diimplementasikan menggunakan representasi adjacency list. Elemen `visitedCount` merupakan hitungan jumlah link yang telah dikunjungi.
Node	Struktur data yang dibentuk dengan menggunakan struct dan terdiri dari string url.
Queue	Struktur data list yang dapat diakses menggunakan paket ‘container/list’. List ini digunakan dengan menggunakan konsep <i>queue</i> , yaitu first in first out. Hal ini dapat diakses dengan melakukan pushback() dan remove().
LinkCache	Struktur data map di mana setiap key adalah string dan setiap value yang terkait adalah sebuah slice dari string, yang berisi sejumlah URL yang terhubung dengan key tersebut. Struktur data ini digunakan untuk menyimpan hasil cache untuk mencegah pengambilan ulang yang tidak perlu.

4.1.2 Fungsi dan Prosedur

1. Function NewGraph() → *Graph

Fungsi untuk membuat suatu graf baru dengan inisialisasi `nodes`, inisialisasi `adjList`, dan `visitedCount` diinisialisasi 0.

```
func NewGraph() *Graph {
    return &Graph{
        nodes:      []*Node{ },
        adjList:    make(map[string][]string),
        visitedCount: 0,
    }
}
```

2. Function AddNode(value string) → *Node

Fungsi untuk menambahkan suatu link dengan tipe Node pada graf dan mereturn referensi ke link tersebut untuk digunakan dalam operasi graf selanjutnya. Fungsi ini merupakan metode dari struktur data ‘Graph’, sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.AddNode(val).

```
func (g *Graph) AddNode(value string) *Node {
    node := &Node{val: value}
    g.nodes = append(g.nodes, node)
    return node
}
```

3. Procedure AddEdge(node1, node2 string)

Fungsi untuk menambahkan suatu edge di antara dua link pada graf.

```
g.adjList[node1] = append(g.adjList[node1], node2)
```

Baris ini menambahkan node2 ke dalam daftar tetangga dari node1 dalam adjList. Jika node1 belum memiliki tetangga, adjList[node1] akan menjadi slice kosong dan node2 akan ditambahkan ke dalamnya.

```
g.adjList[node2] = append(g.adjList[node2], node1)
```

Baris ini melakukan operasi yang sama seperti baris sebelumnya, tetapi kali ini menambahkan node1 ke dalam daftar tetangga dari node2. Struktur data adjList digunakan untuk menyimpan informasi mengenai tetangga-tetangga dari setiap node dalam graf. Fungsi ini merupakan metode dari struktur data ‘Graph’,

sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.AddEdge(n1, n2).

```
func (g *Graph) AddEdge(node1, node2 string) {
    g.adjList[node1] = append(g.adjList[node1], node2)
    g.adjList[node2] = append(g.adjList[node2], node1)
}
```

4. Function BFS(start, end string) → slice of string

Fungsi yang digunakan untuk mencari link akhir yang dicari dengan menggunakan implementasi konsep breadth first search dimana pencarian dilakukan per level. Fungsi ini mengembalikan path atau rute dari link awal ke link akhir dalam bentuk slice of string. Fungsi ini merupakan metode dari struktur data ‘Graph’, sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.BFS(startURL, goalURL).

```
func (g *Graph) BFS(start, end string) []string {
    visited := make(map[string]bool)
    parent := make(map[string]string)
    q := list.New()
    visited[start] = true
    q.PushBack(start)
    for q.Len() != 0 {
        currentNode := q.Front().Value.(string)
        q.Remove(q.Front())
        if currentNode == end {
            path := []string{}
            current := end
            for current != "" {
                path = append([]string{current}, path...)
                current = parent[current]
            }
            return path
        }
        for _, neighbor := range g.adjList[currentNode] {
            g.visitedCount++
            if !visited[neighbor] {
                visited[neighbor] = true
                parent[neighbor] = currentNode
                q.PushBack(neighbor)
            }
        }
    }
    return nil
}
```

5. Function convertToURL(title string) → string

Fungsi ini digunakan untuk mengubah input yang berupa judul artikel menjadi dalam bentuk link sehingga bisa langsung diakses. Fungsi ini mengembalikan hasil link.

```
func convertToURL(title string) string {
    return fmt.Sprintf("https://en.wikipedia.org/wiki/%s",
        strings.ReplaceAll(title, " ", "_"))
}
```

6. Function isValidArticleLink(link string) → boolean

Fungsi ini digunakan untuk mengecek apakah suatu link mengarah ke suatu artikel dengan menuliskan daftar awalan link yang tidak mengarah ke artikel tertentu. Hal ini dapat mengurangi jumlah artikel yang perlu dicek, sehingga mempercepat proses. Fungsi ini mengembalikan boolean dari apakah link memiliki prefix tersebut apa tidak.

```
func isValidArticleLink(link string) bool {
    prefixes := []string{
        "/wiki/Special:",
        "/wiki/Talk:",
        "/wiki/User:",
        "/wiki/Portal:",
        "/wiki/Wikipedia:",
        "/wiki/File:",
        "/wiki/Category:",
        "/wiki/Help:",
        "/wiki/Template:",
        "/wiki/Main_Page",
        "/wiki/Main_Page:",
        "/wiki/Draft:",
        "/wiki/Module:",
        "/wiki/MediaWiki:",
        "/wiki/Index:",
        "/wiki/Education_Program:",
        "/wiki/TimedText:",
        "/wiki/Gadget:",
        "/wiki/Gadget_Definition:",
        "/wiki/Book:",
        "/wiki/AFD:",
        "/wiki/Namespace:",
        "/wiki/Transwiki:",
        "/wiki/Course:",
        "/wiki/Thread:",
```

```

        "/wiki/Summary:",
    }
    for _, prefix := range prefixes {
        if strings.HasPrefix(link, prefix) {
            return false
        }
    }
    return strings.HasPrefix(link, "/wiki/") &&
strings.Contains(link, ":")
}

```

7. Procedure initLinkCache()

Fungsi ini digunakan untuk menginisialisasi proses *caching* dimana memuat cache hasil BFS dari file CSV dengan nama file cached-bfs.csv (jika ada) saat program dimulai.

```

func initLinkCache() {
    linkCache = make(map[string][]string)
    file, err := os.Open("cached-bfs.csv")
    if err != nil {
        log.Println("No existing cache file.")
        return
    }
    defer file.Close()
    reader := csv.NewReader(file)
    records, err := reader.ReadAll()
    if err != nil {
        log.Fatal("Failed to read cache file: ", err)
    }
    for _, record := range records {
        if len(record) >= 2 {
            url := record[0]
            links := record[1:]
            linkCache[url] = links
        }
    }
}

```

8. Procedure saveLinkCache()

Prosedur ini digunakan untuk menyimpan cache hasil BFS ke dalam file cached-bfs.csv untuk digunakan lagi sehingga pencarian link tidak perlu diulang setiap kali program dijalankan.

```

func saveLinkCache() {

```

```

file, err := os.Create("cached-bfs.csv")
if err != nil {
    log.Fatal("Failed to create cache file: ", err)
}
defer file.Close()
writer := csv.NewWriter(file)
defer writer.Flush()
for url, links := range linkCache {
    record := append([]string{url}, links...)
    if err := writer.Write(record); err != nil {
        log.Fatal("Failed to write to cache file: ", err)
    }
}
}

```

9. Function linkScraper(url string, visited map[string]bool) → slice of string

Fungsi ini digunakan untuk mengambil semua link pada suatu laman artikel.
Fungsi ini mengembalikan link-link tersebut dalam bentuk slice of string.

```

func linkScraper(url string, visited map[string]bool) []string {
    if links, ok := linkCache[url]; ok {
        return links
    }
    doc, err := goquery.NewDocument(url)
    if err != nil {
        log.Fatal(err)
    }
    var uniqueLinks []string
    doc.Find("body a").Each(func(index int, item
*goquery.Selection) {
        style, exists := item.Attr("style")
        if exists && (strings.Contains(style, "display: none") || strings.Contains(style, "visibility: hidden")) {
            return
        }
        if _, hiddenExists := item.Attr("hidden"); hiddenExists {
            return
        }
        link, exists := item.Attr("href")
        if !exists || !isValidArticleLink(link) || visited[link] {
            return
        }
        visited[link] = true
        uniqueLinks = append(uniqueLinks,
"https://en.wikipedia.org"+link)
    })
}

```

```
    linkCache[url] = uniqueLinks
    return uniqueLinks
}
```

10. Function getTitle(urlString string) → string

Fungsi ini digunakan untuk mengembalikan judul dari artikel dengan parameter berupa linknya. Fungsi ini akan mengembalikan judul dalam bentuk string.

```
func getTitle(urlString string) string {
    parsedURL, err := url.Parse(urlString)
    if err != nil {
        return ""
    }
    pathParts := strings.Split(parsedURL.Path, "/")
    lastPart := pathParts[len(pathParts)-1]
    title, err := url.PathUnescape(lastPart)
    if err != nil {
        return ""
    }
    return title
}
```

11. Function CORSMiddleware() → gin.HandlerFunc

Fungsi ini digunakan untuk menangani masalah akses lintas domain pada aplikasi web dengan mengatur header HTTP yang sesuai untuk memperbolehkan permintaan dari berbagai domain.

```
func CORSMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        c.Writer.Header().Set("Access-Control-Allow-Origin",
        "*")

        c.Writer.Header().Set("Access-Control-Allow-Credentials", "true")
        c.Writer.Header().Set("Access-Control-Allow-Headers",
        "Origin, Content-Type")
        c.Writer.Header().Set("Access-Control-Allow-Methods",
        "GET, POST, PUT, DELETE, OPTIONS")
        c.Next()
    }
}
```

12. Procedure main()

Prosedur ini memanggil fungsi-fungsi yang telah dibuat dalam menunjang keberjalanannya alur program. Prosedur ini juga menerima input dari *frontend* dan juga mengirimkan hasil ke *frontend*.

```
func main() {
    initLinkCache()
    r := gin.Default()
    r.Use(CORSMiddleware())
    r.GET("/", func(c *gin.Context) {
        startTitle := c.Query("startTitle")
        goalTitle := c.Query("goalTitle")
        if startTitle == "" || goalTitle == "" {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Start title and Goal title are required"})
            return
        }
        start := time.Now()
        g := NewGraph()
        visited := make(map[string]bool)
        startURL := convertToURL(startTitle)
        goalURL := convertToURL(goalTitle)
        visited[startURL] = true
        q := list.New()
        q.PushBack(startURL)
        var mutex sync.Mutex
        pathFound := make(chan []string)
        go func() {
            defer close(pathFound)
            var wg sync.WaitGroup
            for q.Len() != 0 {
                currentURL := q.Front().Value.(string)
                q.Remove(q.Front())
                links := linkScraper(currentURL, visited)
                for _, link := range links {
                    wg.Add(1)
                    go func(link string) {
                        defer wg.Done()
                        mutex.Lock()
                        defer mutex.Unlock()
                        g.AddEdge(currentURL, link)
                        if link == goalURL {
                            path := g.BFS(startURL, goalURL)
                            if path != nil {
                                pathFound <- path
                                return
                            }
                        }
                    }
                }
            }
        }()
        wg.Wait()
        close(pathFound)
    })
}
```

```

        }
        q.PushBack(link)
    } (link)
}
wg.Wait()
}
}()
select {
case path := <-pathFound:
    var pathTitle []string
    for _, node := range path {
        title := getTitle(node)
        pathTitle = append(pathTitle,
strings.ReplaceAll(title, "_", " "))
    }
    endTime := time.Since(start).Milliseconds()
    c.JSON(http.StatusOK, gin.H{"paths": pathTitle,
"timeTaken": endTime, "visited": g.visitedCount, "length":
len(pathTitle) - 1})
    case <-time.After(1000000 * time.Second):
        c.JSON(http.StatusRequestTimeout, gin.H{"error":
"Request timed out"})
    }
    saveLinkCache()
})
r.Run(":8080")
}

```

4.2 Implementasi Algoritma IDS

4.2.1 Struktur Data

Struktur Data	Penjelasan
Graph	Struktur data yang dibentuk dengan menggunakan struct dan terdiri dari beberapa elemen antara lain `nodes`, `adjList`, dan `visitedCount`. Elemen `nodes` adalah slice dari pointer ke objek Node, `adjList` yang merupakan map yang memetakan suatu link ke daftar link yang terhubung secara langsung dengan link tersebut. Dalam representasi ini, graf diimplementasikan menggunakan representasi adjacency list. Elemen `visitedCount` merupakan hitungan jumlah link yang telah dikunjungi.
Node	Struktur data yang dibentuk dengan menggunakan

	struct dan terdiri dari string url.
Queue	Struktur data list yang dapat diakses menggunakan paket ‘container/list’. List ini digunakan dengan menggunakan konsep <i>queue</i> , yaitu first in first out. Hal ini dapat diakses dengan melakukan pushback() dan remove().
LinkCache	Struktur data map di mana setiap key adalah string dan setiap value yang terkait adalah sebuah slice dari string, yang berisi sejumlah URL yang terhubung dengan key tersebut. Struktur data ini digunakan untuk menyimpan hasil cache untuk mencegah pengambilan ulang yang tidak perlu.

4.2.2 Fungsi dan Prosedur

1. Function NewGraph() → *Graph

Fungsi untuk membuat suatu graf baru dengan inisialisasi ‘nodes’, inisialisasi ‘adjList’, dan ‘visitedCount’ diinisialisasi 0.

```
func NewGraph() *Graph {
    return &Graph{
        nodes:      []*Node{},
        adjList:    make(map[string][]string),
        visitedCount: 0,
    }
}
```

2. Function AddNode(value string) → *Node

Fungsi untuk menambahkan suatu link dengan tipe Node pada graf dan mereturn referensi ke link tersebut untuk digunakan dalam operasi graf selanjutnya. Fungsi ini merupakan metode dari struktur data ‘Graph’, sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.AddNode(val).

```
func (g *Graph) AddNode(value string) *Node {
    node := &Node{val: value}
    g.nodes = append(g.nodes, node)
    return node
}
```

3. Procedure AddEdge(node1, node2 string)

Fungsi untuk menambahkan suatu edge di antara dua link pada graf.

```
g.adjList[node1] = append(g.adjList[node1], node2)
```

Baris ini menambahkan node2 ke dalam daftar tetangga dari node1 dalam adjList. Jika node1 belum memiliki tetangga, adjList[node1] akan menjadi slice kosong dan node2 akan ditambahkan ke dalamnya.

```
g.adjList[node2] = append(g.adjList[node2], node1)
```

Baris ini melakukan operasi yang sama seperti baris sebelumnya, tetapi kali ini menambahkan node1 ke dalam daftar tetangga dari node2. Struktur data adjList digunakan untuk menyimpan informasi mengenai tetangga-tetangga dari setiap node dalam graf. Fungsi ini merupakan metode dari struktur data ‘Graph’, sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.AddEdge(n1, n2).

```
func (g *Graph) AddEdge(node1, node2 string) {
    g.adjList[node1] = append(g.adjList[node1], node2)
    g.adjList[node2] = append(g.adjList[node2], node1)
}
```

4. Function IDS(startNode string, goalNode string, maxDepth int) → slice of string

Fungsi yang digunakan untuk mencari link akhir yang dicari dengan menggunakan implementasi konsep iterative deepening search dimana pencarian dilakukan per hingga kedalaman tertentu dan kedalaman ini ditambah terus-menerus hingga ditemukan. Fungsi ini menggunakan fungsi DLS. Fungsi ini mengembalikan path atau rute dari link awal ke link akhir dalam bentuk slice of string. Fungsi ini merupakan metode dari struktur data ‘Graph’, sehingga dalam pemanggilannya, misal sudah di inisiasi graph g, maka fungsi ini dipanggil dengan g.IDS(startURL, goalURL, maxDepth).

```
func (g *Graph) IDS(startNode string, goalNode string, maxDepth int) []string {
    g.visitedCount = 0
    var wg sync.WaitGroup
    var mutex sync.Mutex
    run := func(current string, goal string, depth int, visited map[string]bool) []string {
        defer wg.Done()
        mutex.Lock()
```

```

        defer mutex.Unlock()
        return g.DLS(current, goal, depth, visited)
    }
    for depth := 0; depth <= maxDepth; depth++ {
        visited := make(map[string]bool)
        wg.Add(1)
        result := run(startNode, goalNode, depth, visited)
        wg.Wait()
        if len(result) > 0 {
            return result
        }
    }
    return nil
}

```

5. Function DLS(current string, goal string, depth int, visited map[string]bool) → slice of string

Fungsi ini menggunakan konsep depth limited search sehingga fungsi ini mencari link akhir dengan melakukan pencarian hingga kedalaman tertentu. Fungsi ini melakukan *backtrack* secara rekursif. Fungsi ini mengembalikan path atau rute dari link awal ke link akhir dalam bentuk slice of string.

```

func (g *Graph) DLS(current string, goal string, depth int,
visited map[string]bool) []string {
    if depth == 0 && current == goal {
        return []string{current}
    }
    if depth <= 0 || visited[current] {
        return nil
    }
    visited[current] = true
    for _, neighbor := range g.adjList[current] {
        g.visitedCount++
        if result := g.DLS(neighbor, goal, depth-1, visited);
result != nil {
            return append([]string{current}, result...)
        }
    }
    return nil
}

```

6. Function convertToURL(title string) → string

Fungsi ini digunakan untuk mengubah input yang berupa judul artikel menjadi dalam bentuk link sehingga bisa langsung diakses. Fungsi ini mengembalikan hasil link.

```
func convertToURL(title string) string {
    return fmt.Sprintf("https://en.wikipedia.org/wiki/%s",
        strings.ReplaceAll(title, " ", "_"))
}
```

7. Function isValidArticleLink(link string) → boolean

Fungsi ini digunakan untuk mengecek apakah suatu link mengarah ke suatu artikel dengan menuliskan daftar awalan link yang tidak mengarah ke artikel tertentu. Hal ini dapat mengurangi jumlah artikel yang perlu dicek, sehingga mempercepat proses. Fungsi ini mengembalikan boolean dari apakah link memiliki prefix tersebut apa tidak.

```
func isValidArticleLink(link string) bool {
    prefixes := []string{
        "/wiki/Special:",
        "/wiki/Talk:",
        "/wiki/User:",
        "/wiki/Portal:",
        "/wiki/Wikipedia:",
        "/wiki/File:",
        "/wiki/Category:",
        "/wiki/Help:",
        "/wiki/Template:",
        "/wiki/Main_Page",
        "/wiki/Main_Page:",
        "/wiki/Draft:",
        "/wiki/Module:",
        "/wiki/MediaWiki:",
        "/wiki/Index:",
        "/wiki/Education_Program:",
        "/wiki/TimedText:",
        "/wiki/Gadget:",
        "/wiki/Gadget_Definition:",
        "/wiki/Book:",
        "/wiki/AFD:",
        "/wiki/Namespace:",
        "/wiki/Transwiki:",
        "/wiki/Course:",
        "/wiki/Thread:",
        "/wiki/Summary:",
```

```

    }
    for _, prefix := range prefixes {
        if strings.HasPrefix(link, prefix) {
            return false
        }
    }
    return strings.HasPrefix(link, "/wiki/") &&
!strings.Contains(link, ":")
}

```

8. Procedure initLinkCache()

Fungsi ini digunakan untuk menginisialisasi proses *caching* dimana memuat cache hasil BFS dari file CSV dengan nama file cached-bfs.csv (jika ada) saat program dimulai.

```

func initLinkCache() {
    linkCache = make(map[string][]string)
    file, err := os.Open("cached-bfs.csv")
    if err != nil {
        log.Println("No existing cache file.")
        return
    }
    defer file.Close()
    reader := csv.NewReader(file)
    records, err := reader.ReadAll()
    if err != nil {
        log.Fatal("Failed to read cache file: ", err)
    }
    for _, record := range records {
        if len(record) >= 2 {
            url := record[0]
            links := record[1:]
            linkCache[url] = links
        }
    }
}

```

9. Procedure saveLinkCache()

Prosedur ini digunakan untuk menyimpan cache hasil BFS ke dalam file cached-bfs.csv untuk digunakan lagi sehingga pencarian link tidak perlu diulang setiap kali program dijalankan.

```

func saveLinkCache() {
    file, err := os.Create("cached-bfs.csv")

```

```

        if err != nil {
            log.Fatal("Failed to create cache file: ", err)
        }
        defer file.Close()
        writer := csv.NewWriter(file)
        defer writer.Flush()
        for url, links := range linkCache {
            record := append([]string{url}, links...)
            if err := writer.Write(record); err != nil {
                log.Fatal("Failed to write to cache file: ", err)
            }
        }
    }
}

```

10. Function linkScraper(url string, visited map[string]bool) → slice of string

Fungsi ini digunakan untuk mengambil semua link pada suatu laman artikel.

Fungsi ini mengembalikan link-link tersebut dalam bentuk slice of string.

```

func linkScraper(url string, visited map[string]bool) []string {
    if links, ok := linkCache[url]; ok {
        return links
    }
    doc, err := goquery.NewDocument(url)
    if err != nil {
        log.Fatal(err)
    }
    var uniqueLinks []string
    doc.Find("body a").Each(func(index int, item
*goquery.Selection) {
        style, exists := item.Attr("style")
        if exists && (strings.Contains(style, "display: none") || strings.Contains(style, "visibility: hidden")) {
            return
        }
        if _, hiddenExists := item.Attr("hidden"); hiddenExists {
            return
        }
        link, exists := item.Attr("href")
        if !exists || !isValidArticleLink(link) || visited[link] {
            return
        }
        visited[link] = true
        uniqueLinks = append(uniqueLinks,
"https://en.wikipedia.org"+link)
    })
    linkCache[url] = uniqueLinks
}

```

```
    return uniqueLinks  
}
```

11. Function getTitle(urlString string) → string

Fungsi ini digunakan untuk mengembalikan judul dari artikel dengan parameter berupa linknya. Fungsi ini akan mengembalikan judul dalam bentuk string.

```
func getTitle(urlString string) string {
    parsedURL, err := url.Parse(urlString)
    if err != nil {
        return ""
    }
    pathParts := strings.Split(parsedURL.Path, "/")
    lastPart := pathParts[len(pathParts)-1]
    title, err := url.PathUnescape(lastPart)
    if err != nil {
        return ""
    }
    return title
}
```

12. Function CORSMiddleware() → gin.HandlerFunc

Fungsi ini digunakan untuk menangani masalah akses lintas domain pada aplikasi web dengan mengatur header HTTP yang sesuai untuk memperbolehkan permintaan dari berbagai domain.

```
func CORSMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        c.Writer.Header().Set("Access-Control-Allow-Origin",
        "*")

        c.Writer.Header().Set("Access-Control-Allow-Credentials", "true")
            c.Writer.Header().Set("Access-Control-Allow-Headers",
        "Origin, Content-Type")
                c.Writer.Header().Set("Access-Control-Allow-Methods",
        "GET, POST, PUT, DELETE, OPTIONS")
                    c.Next()
    }
}
```

13. Procedure main()

Prosedur ini memanggil fungsi-fungsi yang telah dibuat dalam menunjang keberjalan alur program. Prosedur ini juga menerima input dari *frontend* dan juga mengirimkan hasil ke *frontend*.

```
func main () {
    initLinkCache()
```

```

r := gin.Default()
r.Use(CORSMiddleware())
r.GET("/", func(c *gin.Context) {
    startTitle := c.Query("startTitle")
    goalTitle := c.Query("goalTitle")
    if startTitle == "" || goalTitle == "" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Start
title and Goal title is required"})
        return
    }
    start := time.Now()
    g := NewGraph()
    if startTitle == goalTitle {
        var paths []string
        paths = append(paths, startTitle)
        c.JSON(http.StatusOK, gin.H{"paths": paths,
"timeTaken": time.Since(start).Milliseconds(), "visited": 0,
"length": 0})
        return
    }
    startURL := convertToURL(startTitle)
    goalURL := convertToURL(goalTitle)
    visited := make(map[string]bool)
    visited[startURL] = true
    q := list.New()
    q.PushBack(startURL)
    maxDepth := 0
    var mutex sync.Mutex
    pathFound := make(chan []string)
    go func() {
        defer close(pathFound)
        var wg sync.WaitGroup
        for q.Len() != 0 {
            currentURL := q.Front().Value.(string)
            q.Remove(q.Front())
            links := linkScraper(currentURL, visited)
            for _, link := range links {
                wg.Add(1)
                go func(link string) {
                    defer wg.Done()
                    mutex.Lock()
                    defer mutex.Unlock()
                    g.AddEdge(currentURL, link)
                    if link == goalURL {
                        path := g.IDS(startURL, goalURL,
maxDepth)
                        for path == nil {
                            maxDepth++

```

```

path = g.IDS(startURL, goalURL,
maxDepth)
    if path != nil {
        pathFound <- path
        return
    }
}
q.PushBack(link)
} (link)
}
wg.Wait()
}
}
} ()
select {
case path := <-pathFound:
    var pathTitle []string
    for _, node := range path {
        title := getTitle(node)
        pathTitle = append(pathTitle,
strings.ReplaceAll(title, "_", " "))
    }
    endTime := time.Since(start).Milliseconds()
    c.JSON(http.StatusOK, gin.H{"paths": pathTitle,
"timeTaken": endTime, "visited": g.visitedCount, "length":
len(pathTitle) - 1})
    case <-time.After(1000000 * time.Second):
        c.JSON(http.StatusRequestTimeout, gin.H{"error":
"Request timed out"})
    }
    saveLinkCache()
}
r.Run(":8081")
}

```

4.3 Tata Cara Penggunaan Program

4.3.1 Laman Utama

Laman ini berisi informasi utama mengenai program WikiRace. Pada laman ini juga tersedia pilihan bagi pengguna untuk memainkan WikiRace dengan algoritma BFS atau IDS. Pengguna dapat menekan pilihannya, bubbles untuk BFS, sedangkan buttercup untuk IDS.

Vite + React × +
localhost:5173

WikiRace Home About Us How To Use

WIKIRACE

By WikiWiggy



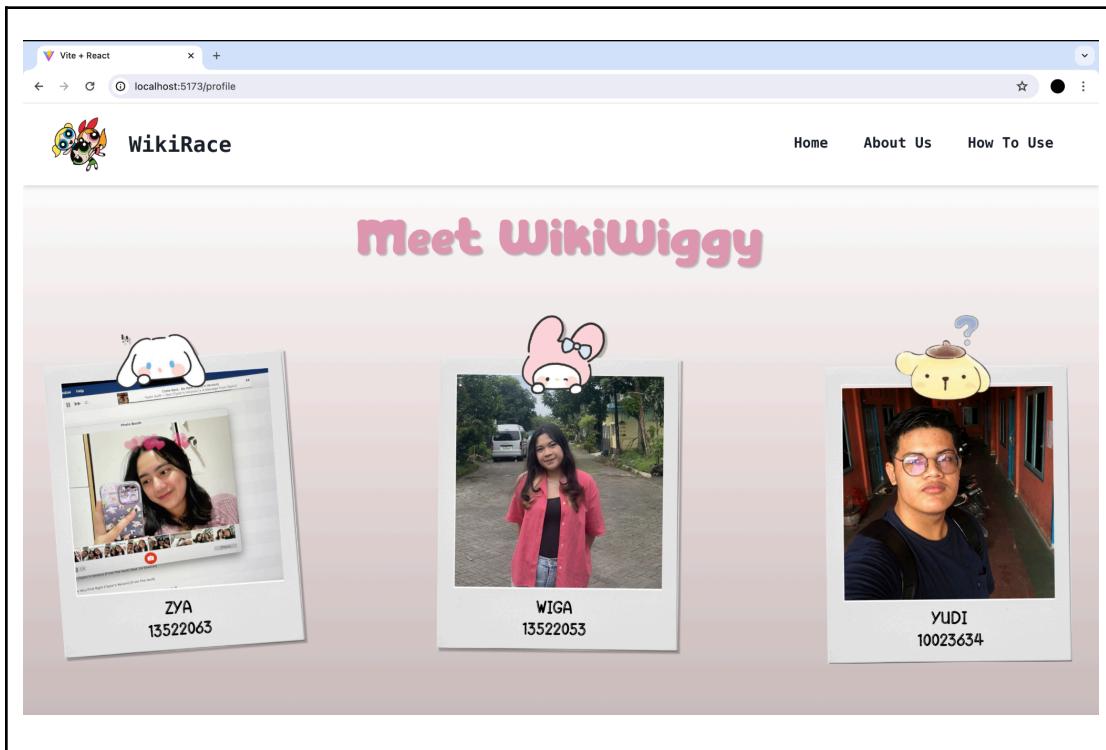
WikiRace or Wiki Game is a game involving Wikipedia, a free online encyclopedia managed by various volunteers worldwide, where players start at a Wikipedia article and must navigate through other articles on Wikipedia (by clicking on links within each) to reach another pre-determined article within the shortest time or with the fewest clicks (articles).

How does it work??



4.3.2 Laman Profil

Laman ini berisi informasi pembuat program yaitu kelompok WikiWiggy.



4.3.3 Laman Cara Penggunaan

Laman ini berisi informasi mengenai cara penggunaan web permainan WikiRace, baik dengan menggunakan algoritma BFS maupun IDS.

The screenshot shows the 'How To Use' section of the WikiRace website. It features two main boxes: one for BFS (Breadth-first search) and one for IDS (Iterative deepening search). Both boxes contain text descriptions and small cartoon character icons above the algorithm names.

BFS:

Breadth-first search (BFS) is a graph traversal algorithm to systematically explore and visit all the vertices of a graph. It starts at a specified vertex and explores all its neighbors before moving on to the next level of neighbors. The algorithm traverses the graph in a level-by-level fashion, ensuring that all vertices at a given level are visited before moving deeper into the graph.

IDS:

Iterative deepening search (IDS) is a variant of depth-first search in which it will have increasing depth until achieving the goal node. This algorithm implements depth-limited search but with an increasing maximum depth. IDS is used to search without exhaustively exploring the entire search space at once. IDS is like climbing a building floor by floor, searching more deeply each time until found, minimizing unnecessary exploration.

HOW TO USE

1. Open the Home Page

HOW TO USE

1. Open the Home Page
2. Scroll until you find the algorithm choice
3. Choose the BFS algorithm
4. Input the starting article
5. Input the ending article
6. Click the start button
7. Done!

HOW TO USE

1. Open the Home Page
2. Scroll until you find the algorithm choice
3. Choose the IDS algorithm
4. Input the starting article
5. Input the ending article
6. Click the start button
7. Done!

4.3.4 Laman BFS

Laman ini berisi program WikiRace dengan algoritma BFS. Pengguna menginput *start article* dan *goal article* lalu menekan tombol *start* untuk mendapatkan *shortest path* antara kedua artikel tersebut berdasarkan algoritma BFS.

The image displays two screenshots of a web application titled "WikiRace" built with Vite + React. The application is designed for performing Breadth First Search (BFS) on Wikipedia articles.

Initial State: The top screenshot shows the application's main interface. It features a blue gradient background with a cartoon character of a girl with yellow hair and large blue eyes on the left. In the center, there is a dark blue rounded rectangle containing the text "Breadth First Search". Below this are two input fields: "Enter the start article" and "Enter the goal article", separated by a small gap. A central "Start" button is positioned between the input fields. At the top of the page, there is a navigation bar with links for "Home", "About Us", and "How To Use". The browser header indicates the page is at "localhost:5173/bfs-page".

After Input: The bottom screenshot shows the same interface after user input. The "start article" field now contains the text "Taylor Swift". The "goal article" field is empty. The "Start" button remains visible below the input fields. The rest of the interface, including the character icon and the central search title, remains the same as in the initial state.

Vite + React

localhost:5173/bfs-page

WikiRace

Home About Us How To Use



Breadth First Search

Taylor Swift

Hook (music)

Start

Vite + React

localhost:5173/bfs-page

WikiRace

Home About Us How To Use



Breadth First Search

Taylor Swift

Hook (music)

Start

Found path with length 2 from Taylor Swift to Hook (music) using BFS :

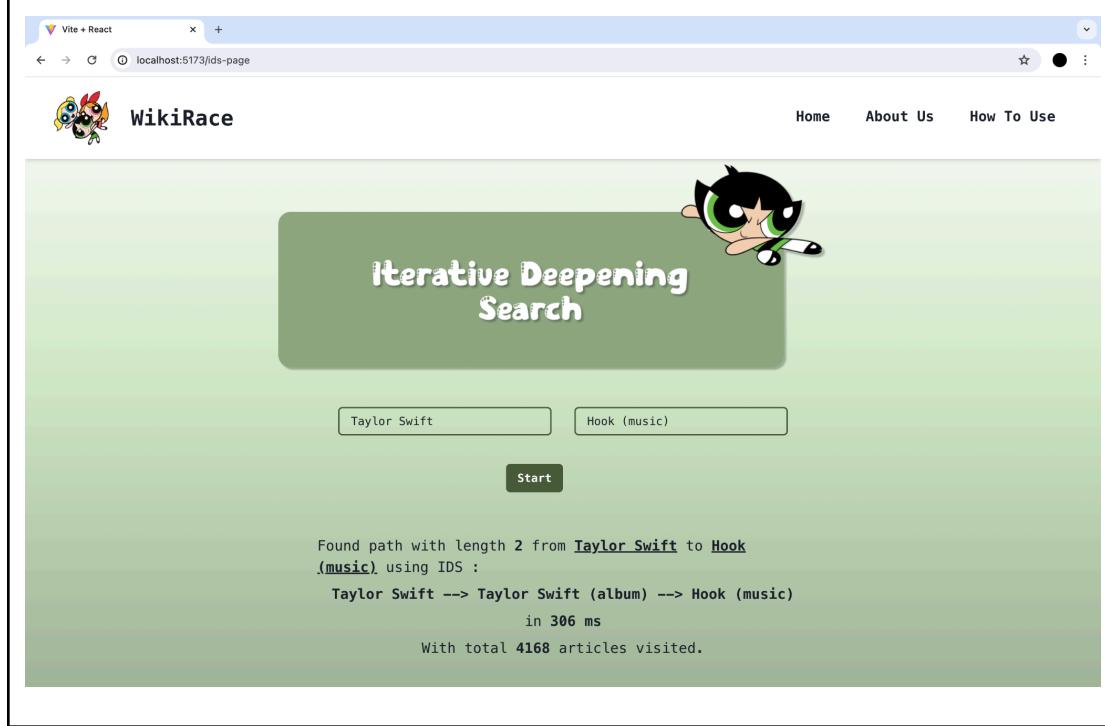
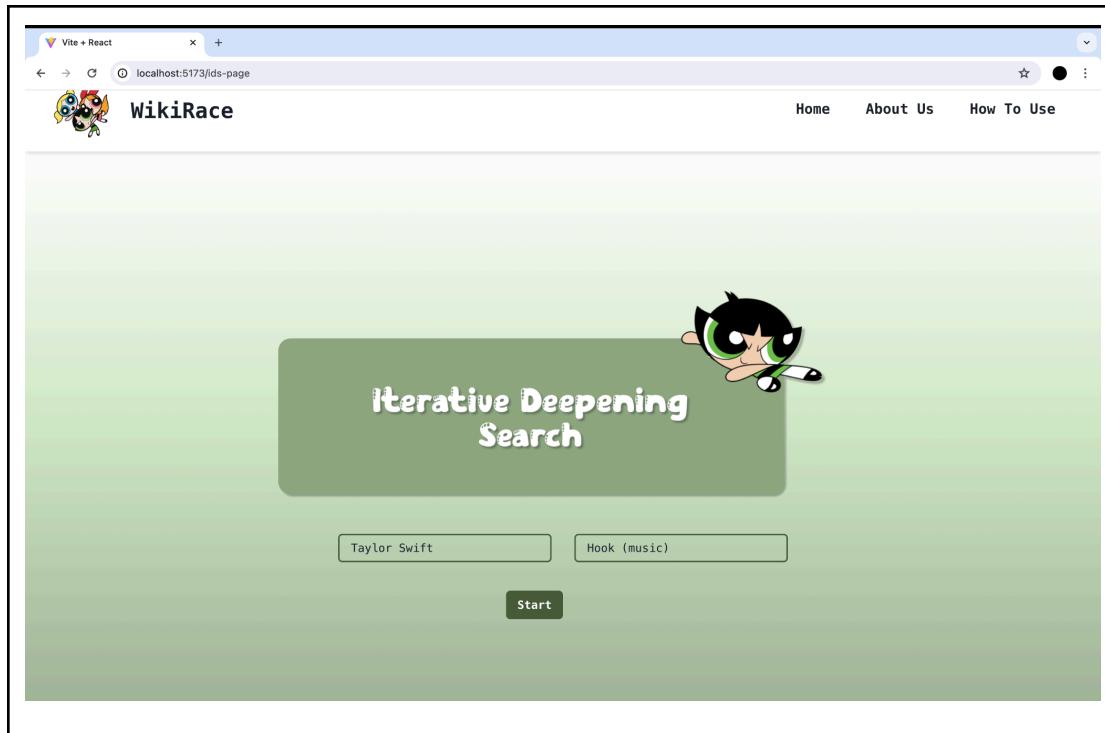
Taylor Swift --> Taylor Swift (album) --> Hook (music)
in 1034 ms

With total 8745 articles visited.

4.3.5 Laman IDS

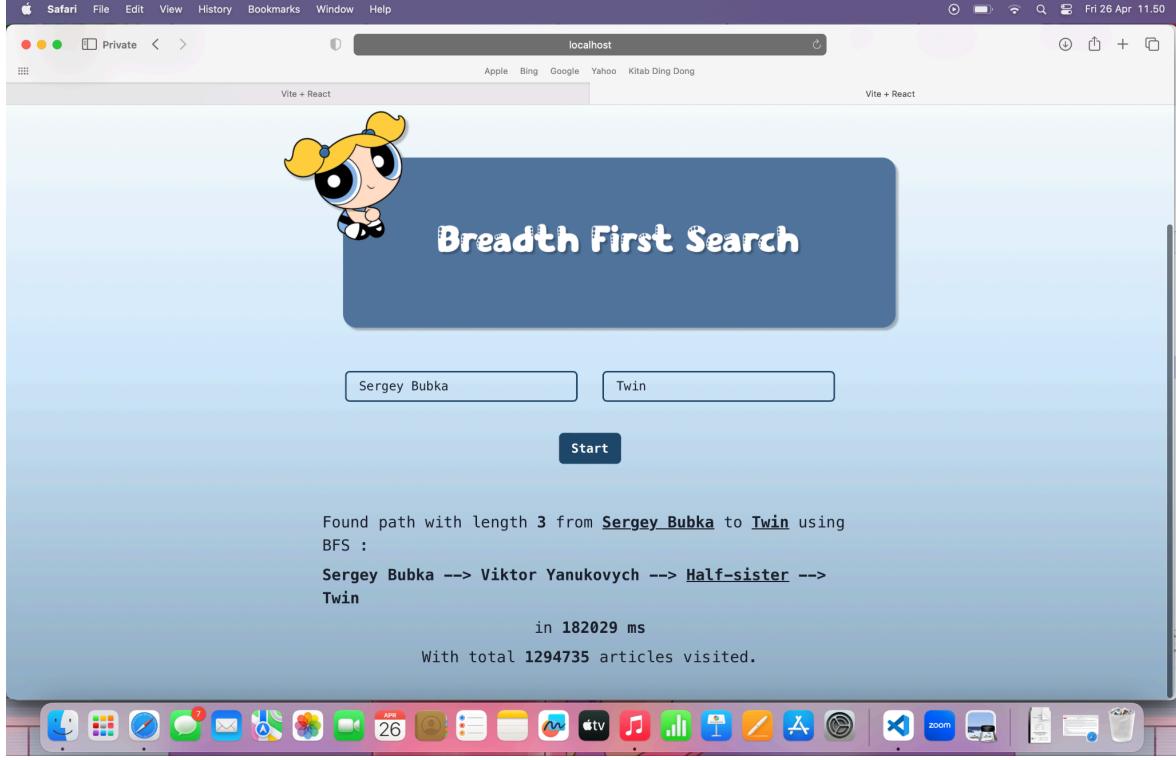
Laman ini berisi program WikiRace dengan algoritma IDS. Pengguna menginput *start article* dan *goal article* lalu menekan tombol *start* untuk mendapatkan *shortest path* antara kedua artikel tersebut berdasarkan algoritma IDS.

The screenshot shows two identical browser windows side-by-side, both displaying the 'Iterative Deepening Search' page of the WikiRace application. The URL in the address bar is 'localhost:5173/ids-page'. The page features a green gradient background with a central dark green rounded rectangle containing the text 'Iterative Deepening Search' and a cartoon character of a black-haired girl with green eyes. Below this are two input fields: 'Enter the start article' containing 'Taylor Swift' and 'Enter the goal article' which is empty. A dark green 'Start' button is positioned between the input fields. At the top of the page, there is a navigation bar with links for 'Home', 'About Us', and 'How To Use'. The top left corner of the browser window shows 'Vite + React'.



4.4 Hasil Pengujian

4.4.1 Test Case 1



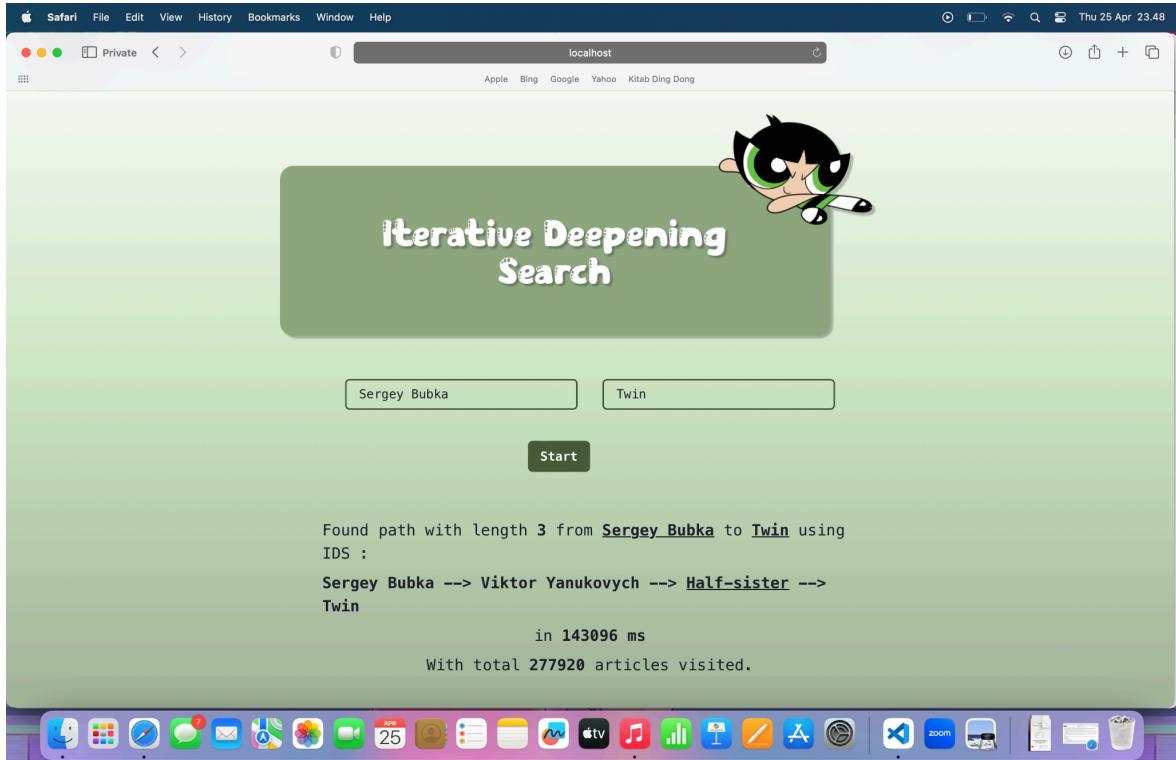
Screenshot of a web browser window titled "localhost" showing the results of a Breadth First Search (BFS) between "Sergey Bubka" and "Twin". The search was completed in 182029 ms with a total of 1294735 articles visited. The path found is Sergey Bubka --> Viktor Yanukovych --> Half-sister --> Twin.

Breadth First Search

Sergey Bubka Twin

Start

Found path with length 3 from Sergey_Bubka to Twin using
BFS :
Sergey Bubka --> Viktor Yanukovych --> Half-sister -->
Twin
in 182029 ms
With total 1294735 articles visited.



Screenshot of a web browser window titled "localhost" showing the results of an Iterative Deepening Search (IDS) between "Sergey Bubka" and "Twin". The search was completed in 143096 ms with a total of 277920 articles visited. The path found is Sergey Bubka --> Viktor Yanukovych --> Half-sister --> Twin.

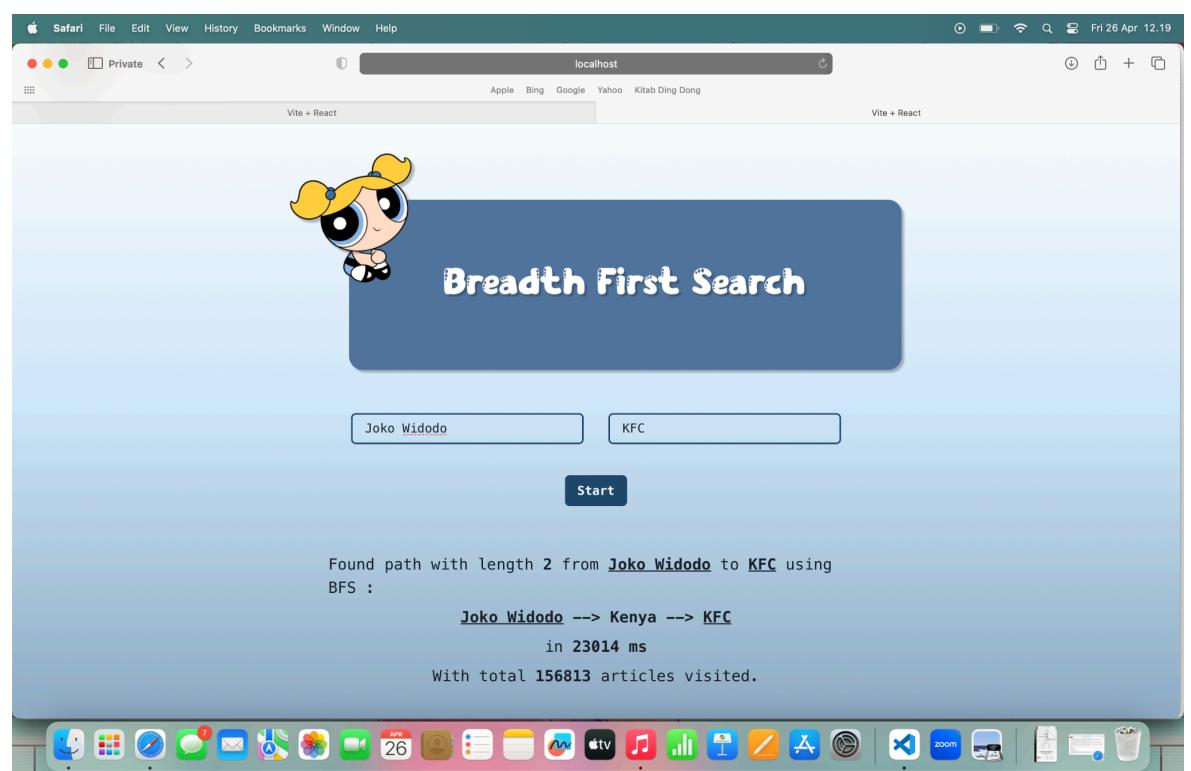
Iterative Deepening
Search

Sergey Bubka Twin

Start

Found path with length 3 from Sergey_Bubka to Twin using
IDS :
Sergey Bubka --> Viktor Yanukovych --> Half-sister -->
Twin
in 143096 ms
With total 277920 articles visited.

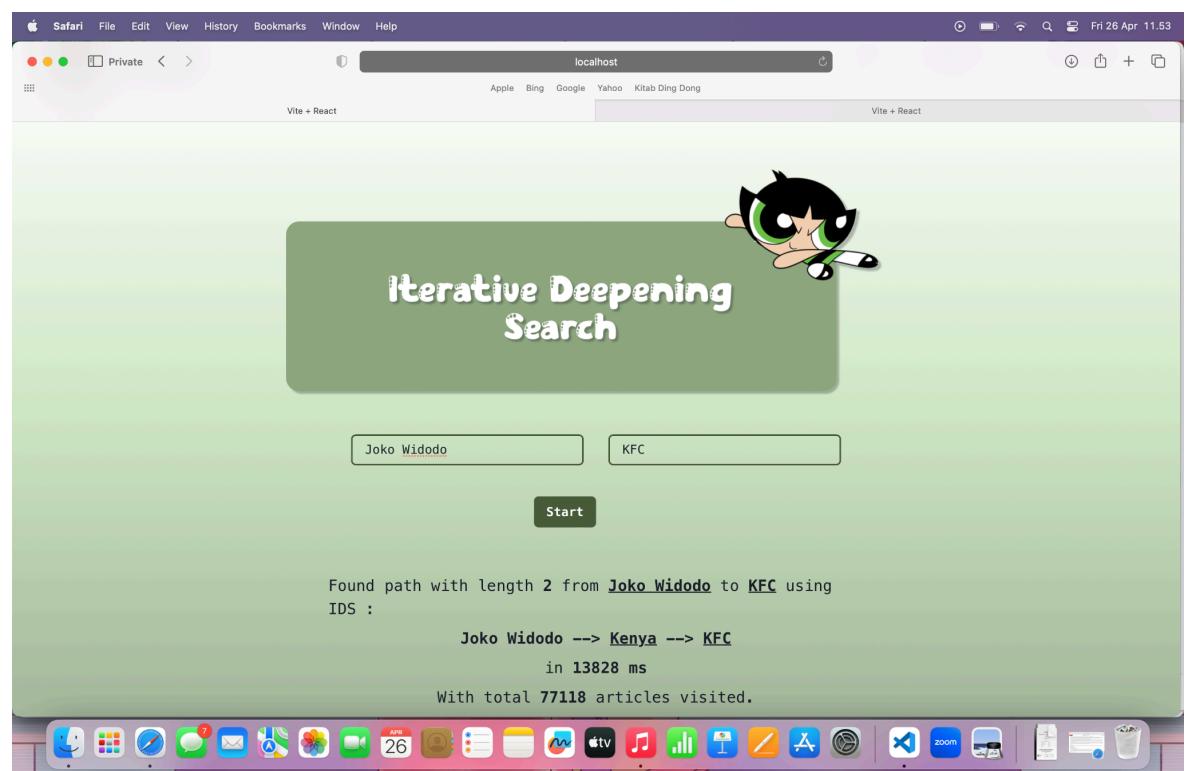
4.4.2 Test Case 2



Screenshot of a web browser window titled "localhost" showing the results of a Breadth First Search (BFS) algorithm. The search path found from "Joko Widodo" to "KFC" is: Joko Widodo --> Kenya --> KFC, completed in 23014 ms with a total of 156813 articles visited. The interface includes input fields for "Joko Widodo" and "KFC" and a "Start" button. A cartoon character is visible above the search results.

Found path with length 2 from Joko Widodo to KFC using BFS :

Joko Widodo --> Kenya --> KFC
in 23014 ms
With total 156813 articles visited.

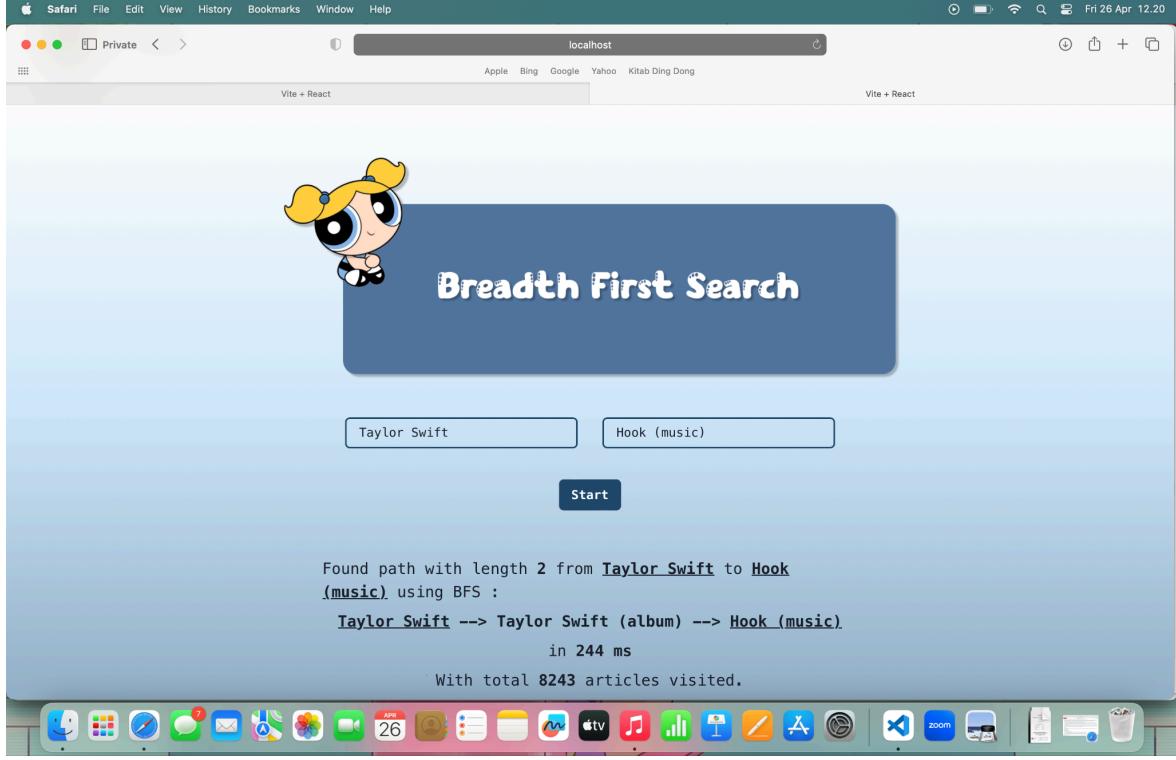


Screenshot of a web browser window titled "localhost" showing the results of an Iterative Deepening Search (IDS) algorithm. The search path found from "Joko Widodo" to "KFC" is: Joko Widodo --> Kenya --> KFC, completed in 13828 ms with a total of 77118 articles visited. The interface includes input fields for "Joko Widodo" and "KFC" and a "Start" button. A cartoon character is visible above the search results.

Found path with length 2 from Joko Widodo to KFC using IDS :

Joko Widodo --> Kenya --> KFC
in 13828 ms
With total 77118 articles visited.

4.4.3 Test Case 3



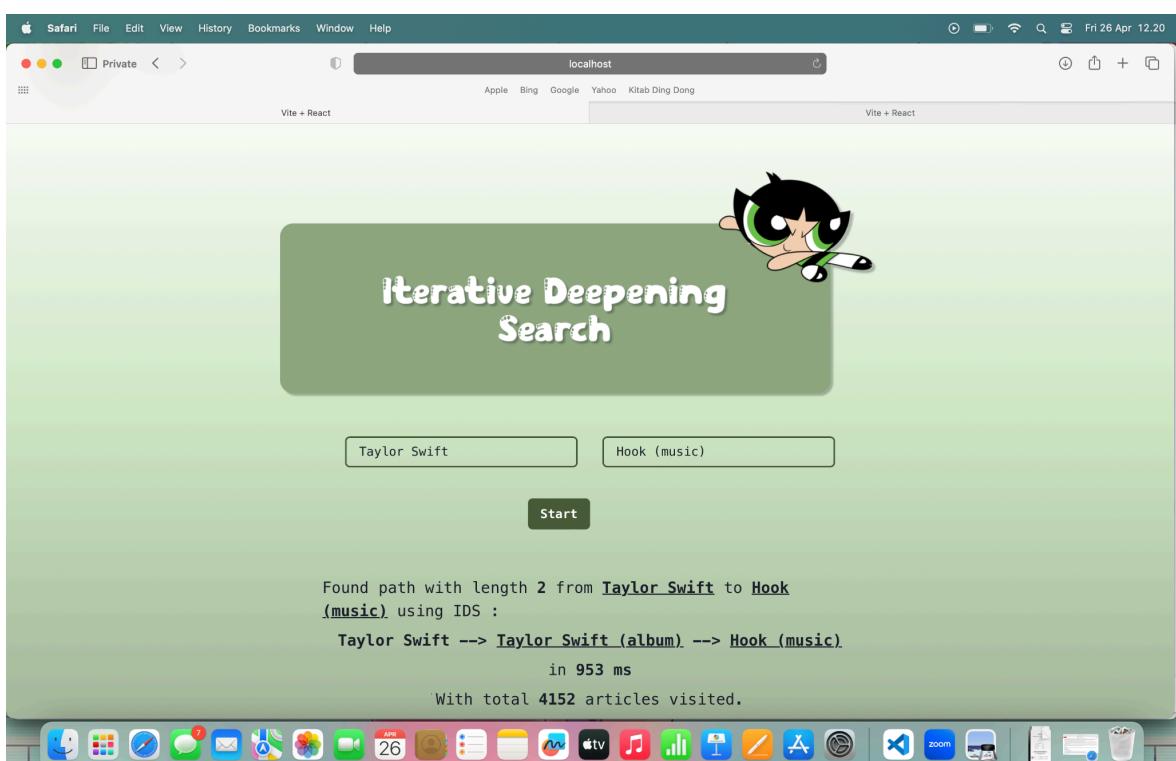
Screenshot of a web browser window titled "localhost" showing the results of a Breadth First Search (BFS) for paths between "Taylor Swift" and "Hook (music)".

The search interface features a cartoon character from "The Powerpuff Girls" at the top left. A central blue box displays the title "Breadth First Search". Below the search bar, there are two input fields: "Taylor Swift" and "Hook (music)". A "Start" button is centered below the inputs.

The search results output is as follows:

```
Found path with length 2 from Taylor Swift to Hook (music) :  
Taylor Swift --> Taylor Swift (album) --> Hook (music).  
in 244 ms  
With total 8243 articles visited.
```

The browser's address bar shows "Vite + React". The status bar at the bottom indicates "Fri 26 Apr 12.20". The Mac OS X Dock is visible at the bottom of the screen.



Screenshot of a web browser window titled "localhost" showing the results of an Iterative Deepening Search (IDS) for paths between "Taylor Swift" and "Hook (music)".

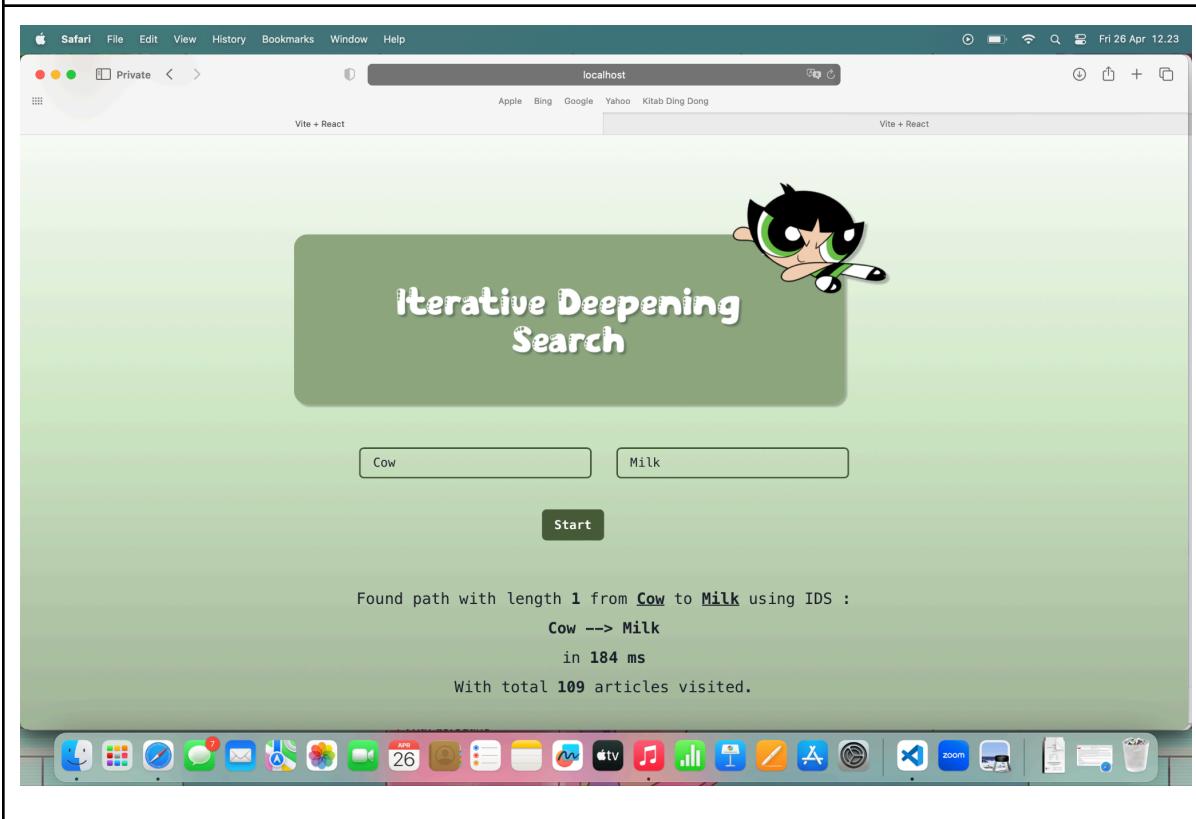
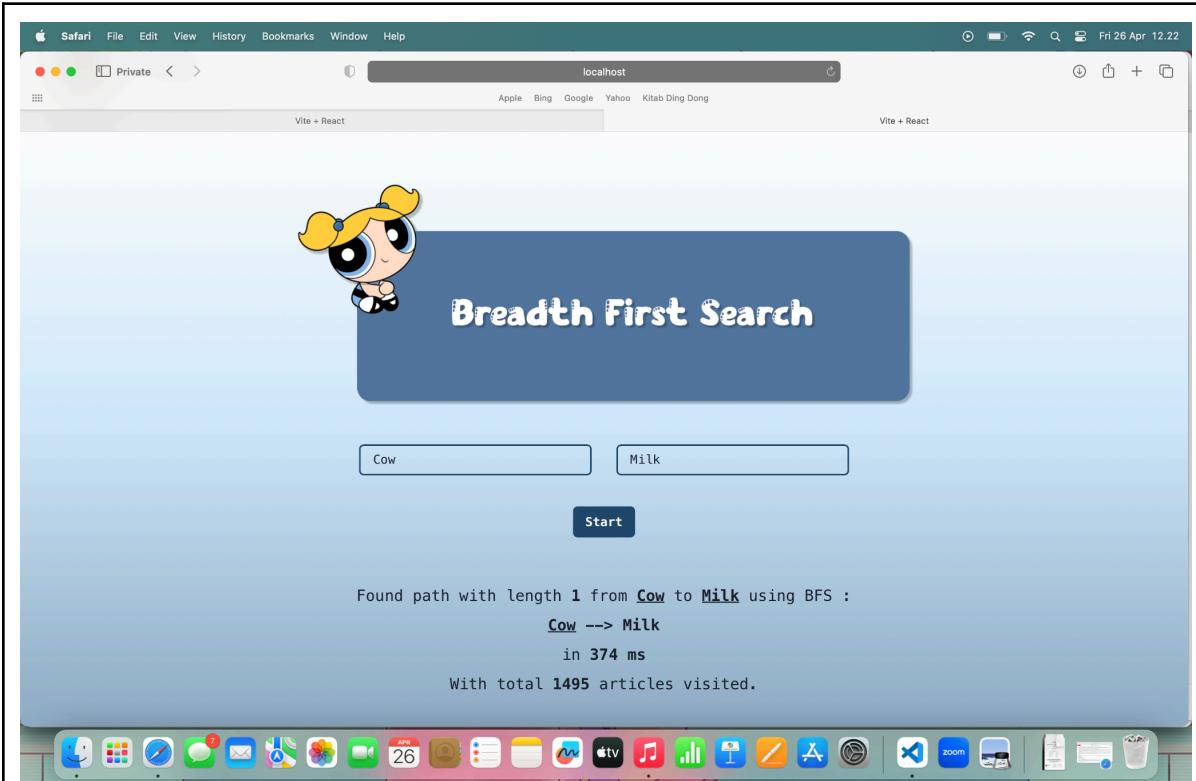
The search interface features a cartoon character from "The Powerpuff Girls" at the top right. A central green box displays the title "Iterative Deepening Search". Below the search bar, there are two input fields: "Taylor Swift" and "Hook (music)". A "Start" button is centered below the inputs.

The search results output is as follows:

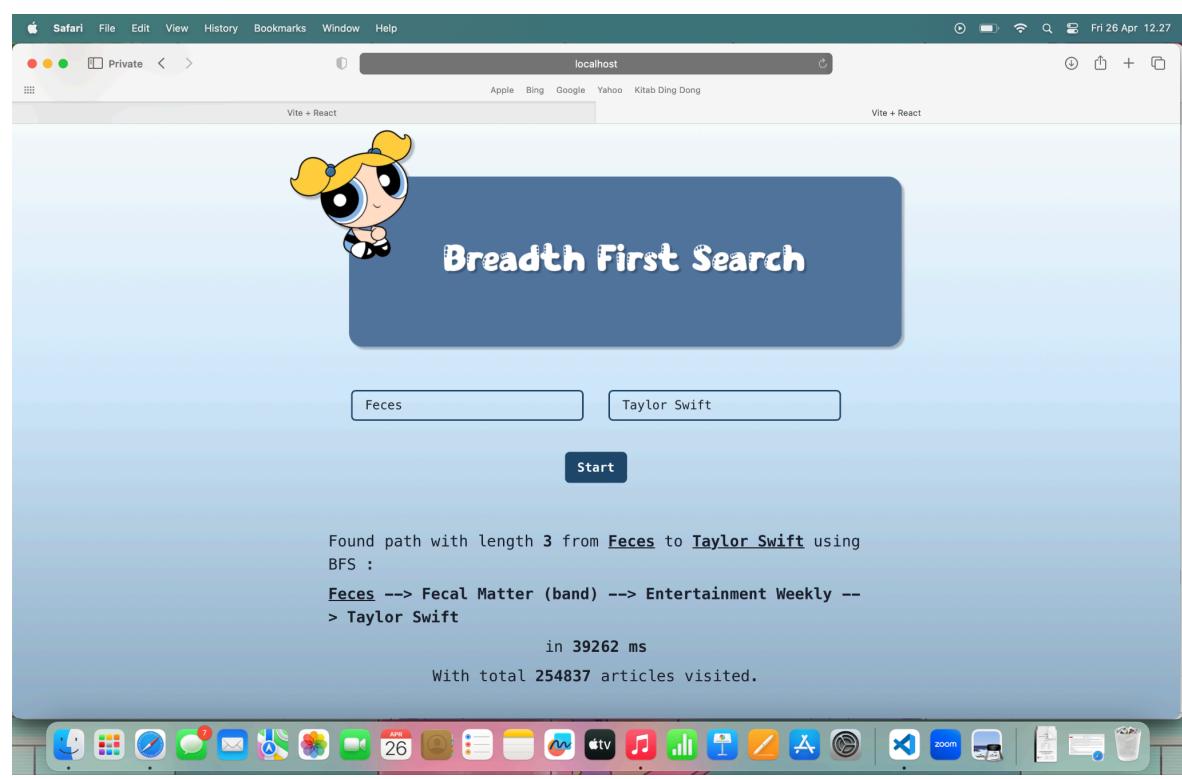
```
Found path with length 2 from Taylor Swift to Hook (music) :  
Taylor Swift --> Taylor Swift (album) --> Hook (music).  
in 953 ms  
With total 4152 articles visited.
```

The browser's address bar shows "Vite + React". The status bar at the bottom indicates "Fri 26 Apr 12.20". The Mac OS X Dock is visible at the bottom of the screen.

4.4.4 Test Case 4

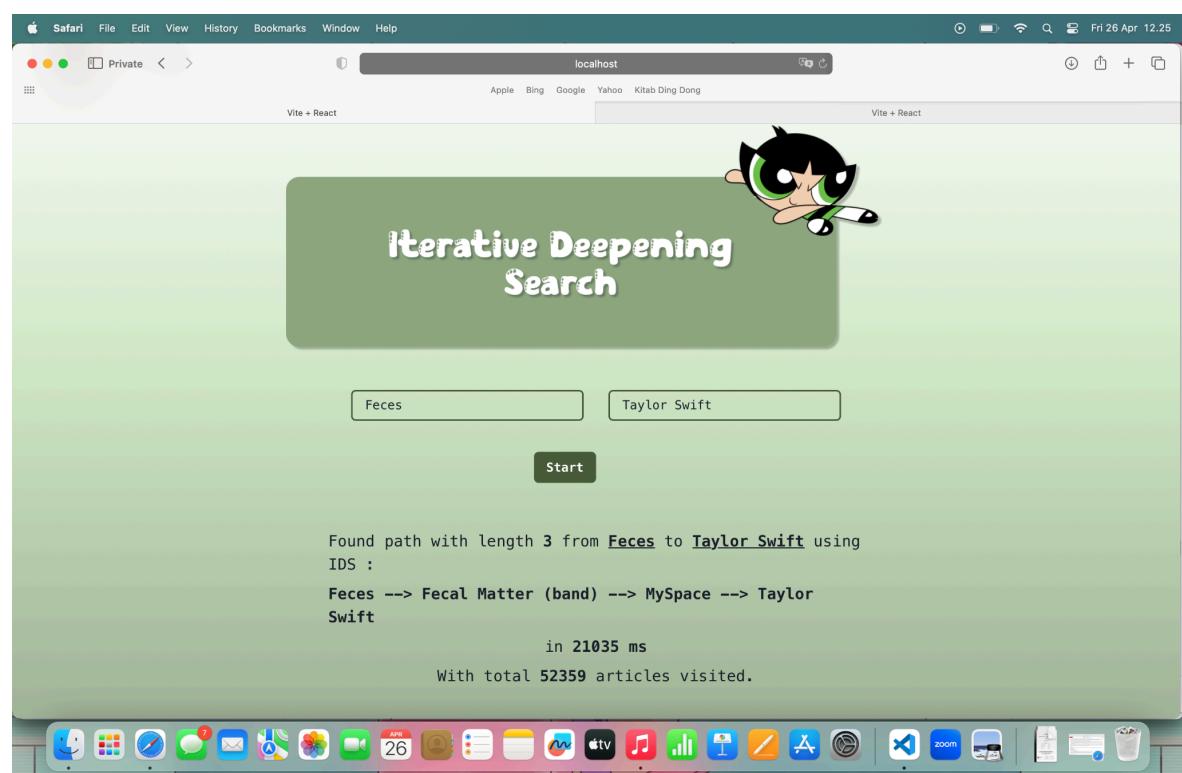


4.4.5 Test Case 5



Screenshot of a web browser window titled "localhost" showing the results of a Breadth First Search (BFS) from "Feces" to "Taylor Swift". The search interface features a cartoon character at the top, a central title "Breadth First Search", and two input fields containing "Feces" and "Taylor Swift". A "Start" button is positioned below the fields. The search results are displayed below the button, showing a path length of 3, a total of 254837 articles visited, and a completion time of 39262 ms.

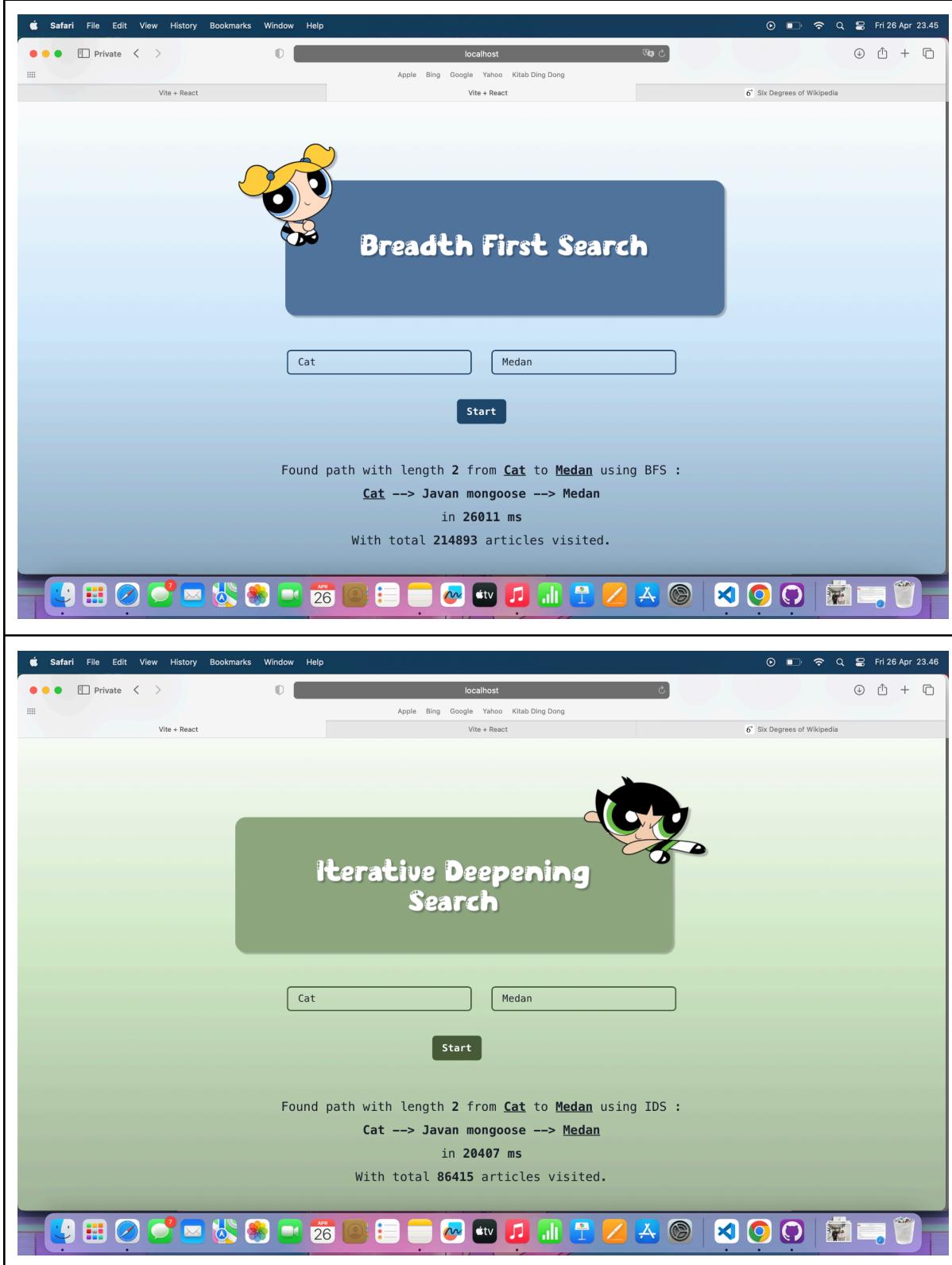
```
Found path with length 3 from Feces to Taylor Swift using
BFS :
Feces --> Fecal Matter (band) --> Entertainment Weekly -->
Taylor Swift
in 39262 ms
With total 254837 articles visited.
```



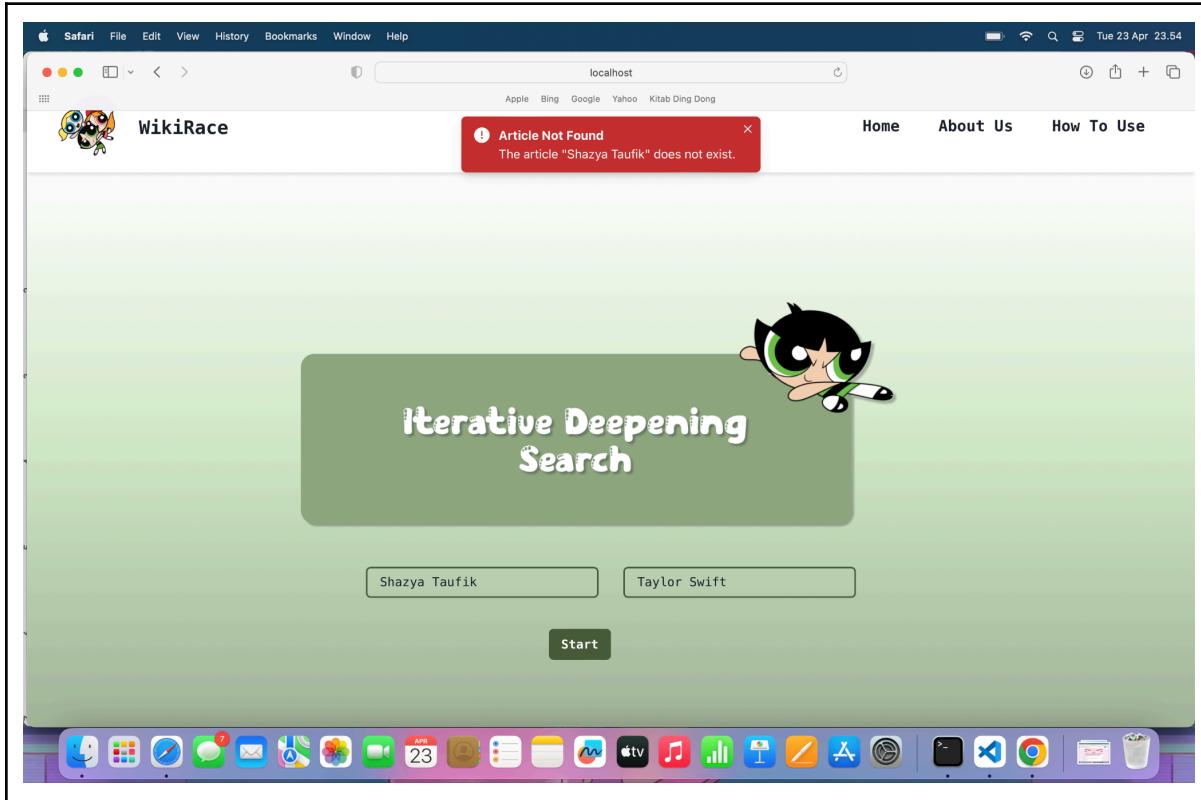
Screenshot of a web browser window titled "localhost" showing the results of an Iterative Deepening Search (IDS) from "Feces" to "Taylor Swift". The search interface is identical to the BFS screenshot, featuring a cartoon character, a central title "Iterative Deepening Search", and two input fields for "Feces" and "Taylor Swift". The search results show a path length of 3, a total of 52359 articles visited, and a completion time of 21035 ms.

```
Found path with length 3 from Feces to Taylor Swift using
IDS :
Feces --> Fecal Matter (band) --> MySpace --> Taylor
Swift
in 21035 ms
With total 52359 articles visited.
```

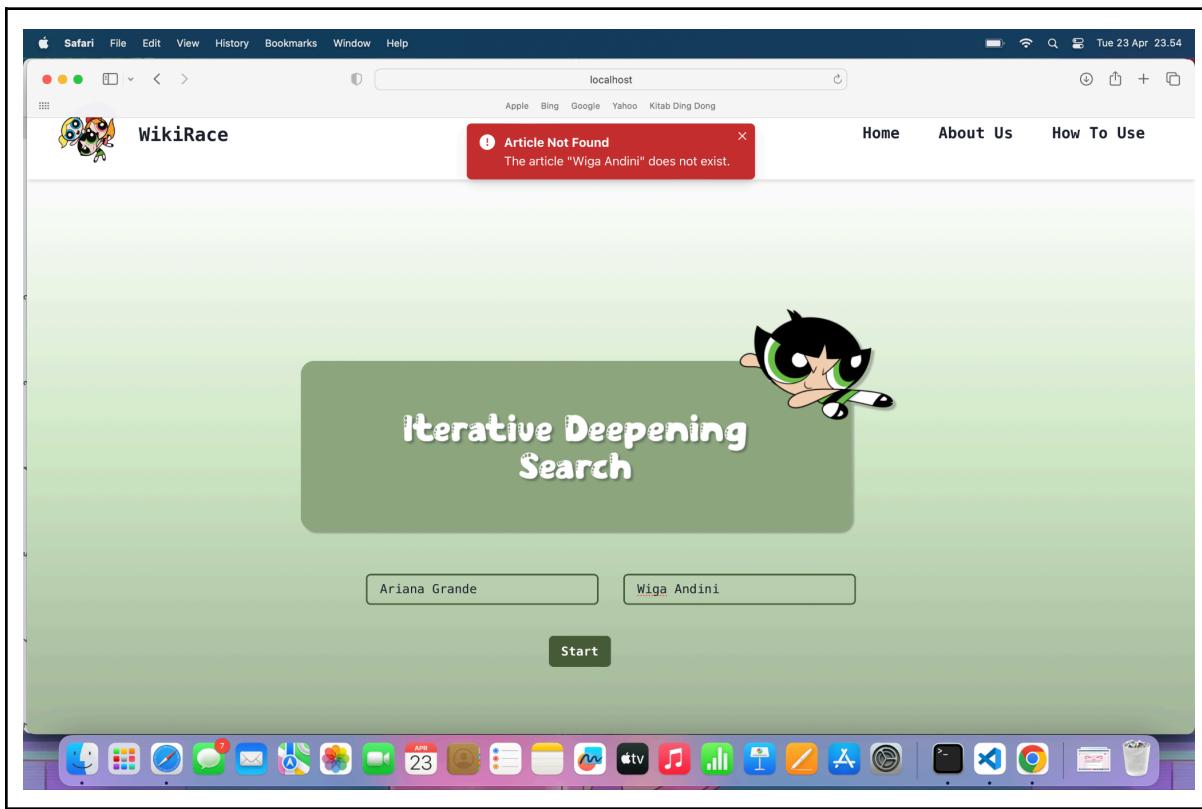
4.4.6 Test Case 6



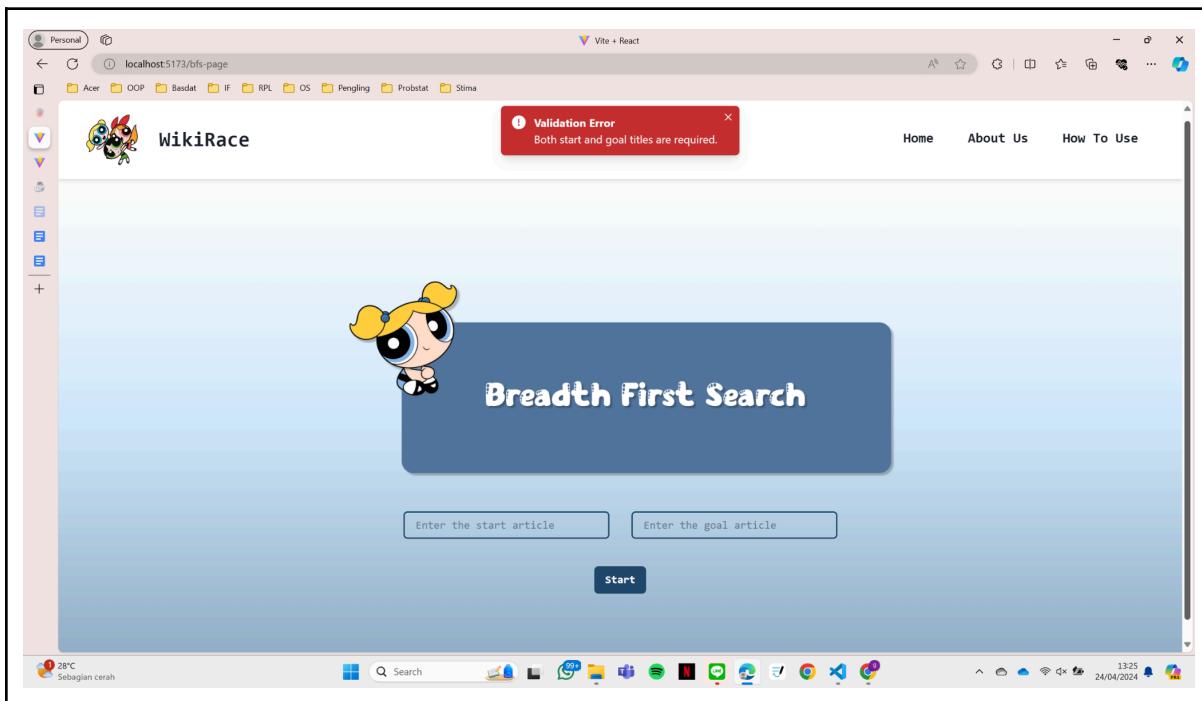
4.4.7 Test Case 7



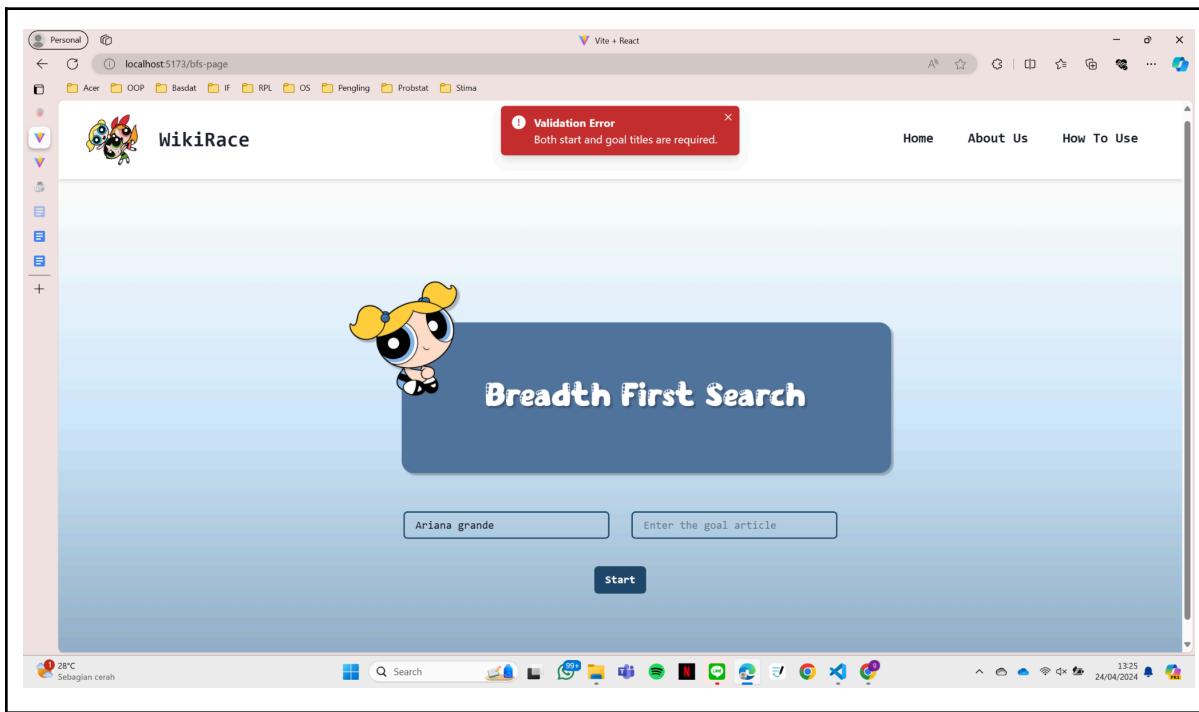
4.4.8 Test Case 8



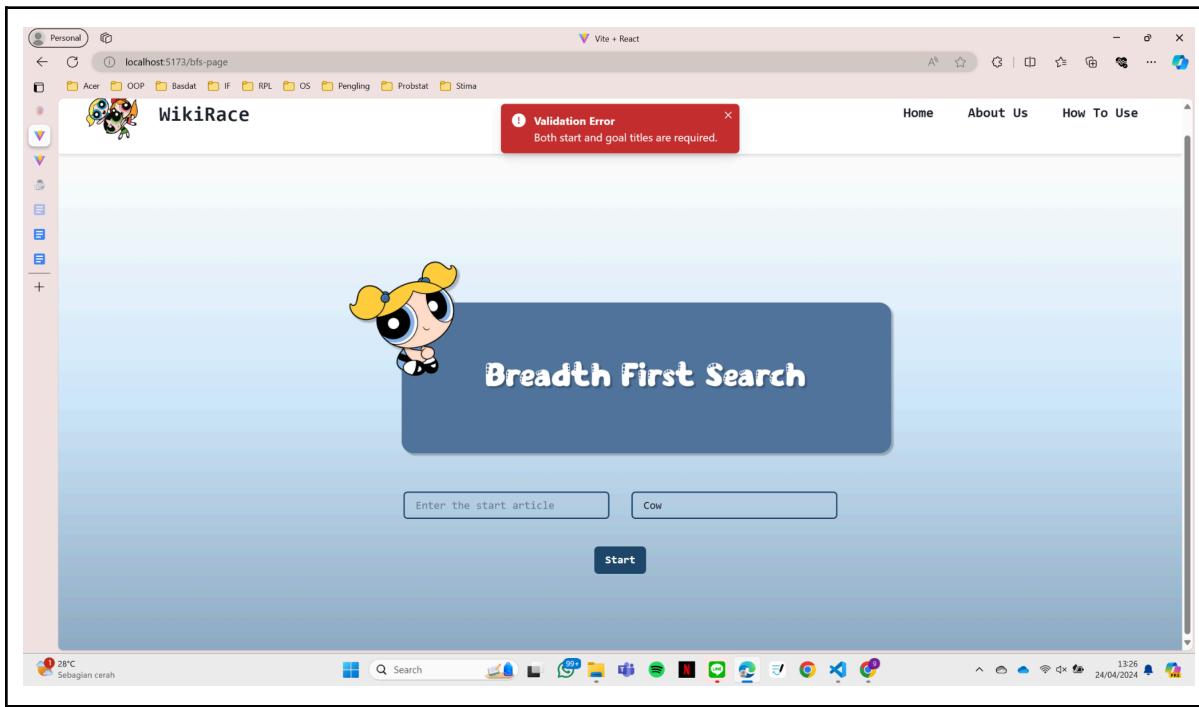
4.4.9 Test Case 9



4.4.10 Test Case 10



4.4.11 Test Case 11



4.5 Analisis Hasil Pengujian

Terdapat beberapa pengujian yang dilakukan terhadap artikel awal dan artikel akhir yang berbeda. Algoritma BFS yang kami gunakan menggunakan metode iteratif. Algoritma akan mengecek apakah tetangga dari node yang sedang dikunjungi apakah memiliki artikel akhir dan belum dikunjungi. Algoritma IDS yang kami gunakan menggunakan metode rekursif. Algoritma akan mengecek semua tetangga yang belum pernah dikunjungi dan memasukkannya ke dalam queue. Berdasarkan pengujian yang kami lakukan, dapat dilihat bahwa metode IDS kami cenderung lebih cepat dan lebih hemat memori. Namun pada kasus tertentu, algoritma BFS kami lebih cepat. Hal ini dikarenakan lokasi artikel akhir yang jauh di kanan graf. Dalam beberapa kasus, kami juga menemukan bahwa terdapat *link* yang tidak langsung melakukan *redirect* ke artikel terkait.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Melalui Tugas Besar II Strategi Algoritma ini, kami diminta untuk membuat website yang dapat mencari rute terpendek antara dua buah artikel pada laman Wikipedia. Kami membuat *frontend website* dengan menggunakan *framework* ReactJS dan chakraUI dengan menggunakan bahasa JSX (JavaScript Syntax Extension) serta bahasa Go untuk *backend*. Melalui pengujian yang kami lakukan, algoritma IDS cenderung lebih optimal karena dapat menemukan artikel akhir dengan waktu yang lebih singkat dan juga lebih hemat memori (laman yang dikunjungi berjumlah lebih sedikit). Namun, pada beberapa kasus, BFS lebih cepat karena lokasi artikel akhir yang terletak pada bagian kanan pada graf.

5.2 Saran

Saran untuk kelompok WikiWiggy untuk kedepannya adalah sebagai berikut,

1. Mengoptimalkan struktur pembagian tugas serta meningkatkan kerjasama tim untuk mengoptimalkan hasil program.
2. Perluasan pengembangan algoritma untuk meningkatkan efisiensi dalam mencapai goal program.
3. Peningkatan dalam penulisan komentar dan dokumentasi untuk mempermudah kerja sama tim dan *maintenance* kode program

LAMPIRAN

- Tautan *repository* Github:

https://github.com/wigaandini/Tubes2_WikiWiggy

- Tautan video penjelasan:

https://youtu.be/p_GyjQVEhv0

- Tabel pembagian tugas

Nama	NIM	Tugas
Erdianti Wiga Putri Andini	13522053	Frontend
		Backend
		Algoritma BFS
		Algoritma IDS
		Caching
		Docker
		Laporan bab 1, 2, 3, 4, 5
Shazya Audrea Taufik	13522063	Video
		Backend
		Algoritma BFS
		Algoritma IDS
		Link scraping
		Docker
		Laporan bab 1, 2, 3, 4, 5
Yudi Kurniawan	10023634	Video
		Laporan bab 1 dan 2

DAFTAR PUSTAKA

- [1] Rinaldi, M. (n.d.). *BFS DFS 2021 Bag1* 2024. Diakses 26 April 2024 melalui
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>
- [2] Rinaldi, M. (n.d.). *BFS DFS 2021 Bag2*. Diakses 26 April 2024 melalui
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>
- [3] Rifqi Mulyawan (27 April 2024) *Penjelasan Pengertian Back-End Development: Apa itu? Tujuan, Jenis dan Bagian, Macam-Macam Bahasa Pemrograman serta Apa Saja Keterampilannya!*. Diambil dari
<https://rifqimulyawan.com/blog/pengertian-back-end-development/>
- [4] Rifqi Mulyawan (27 April 2024) *Memahami Pengertian Front-End Development: Apa itu? Tujuan dan Fungsi, Jenis Pekerjaan, Macam-Macam Bahasa serta Perbedaannya dengan Back-End Development!*. Diambil dari
<https://rifqimulyawan.com/blog/pengertian-front-end-development/>.
- [5] Rifqi Mulyawan (27 April 2024) *Penjelasan Pengertian Back-End Development: Apa itu? Tujuan, Jenis dan Bagian, Macam-Macam Bahasa Pemrograman serta Apa Saja Keterampilannya!*. Diambil dari
<https://rifqimulyawan.com/blog/pengertian-back-end-development/>.
- [6] Intern, D. (2020, June 16). *Apa itu Full Stack Developer? Keahlian-keahlian yang harus dikuasai*. Diakses 27 April 2024 melalui
<https://www.dicoding.com/blog/apa-itu-full-stack-developer-keahlian-keahlian-yang-harus-dikuasai/>
- [7] Maulana, M. (13 Maret 2023) *React JS Adalah : Definisi , Pengertian & Cara Installnya*. Diakses 27 April 2024 melalui <https://itbox.id/blog/react-js-adalah/>
- [8] Gin Web Framework. *Gin Documentation*. Diakses 27 April 2024 melalui
<https://gin-gonic.com/docs/>

[9] GO. *Go Documentation*. Diakses 27 April 2024 melalui <https://go.dev/doc/>

[10] ChakraUI. *ChakraUI Documentation*. Diakses 27 April 2024 melalui
<https://v2.chakra-ui.com>

[11] Docker. *Docker: Accelerated Container Application Development*. Diakses 27 April 2024
melalui <https://www.docker.com/#build>