LAPORAN TUGAS BESAR 2

IF2211 Strategi Algoritma

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Disusun oleh:

Rafiki Prawhira Harianto	13522065
Bagas Sambega Rosyada	13522071
Abdullah Mubarak	13522101

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

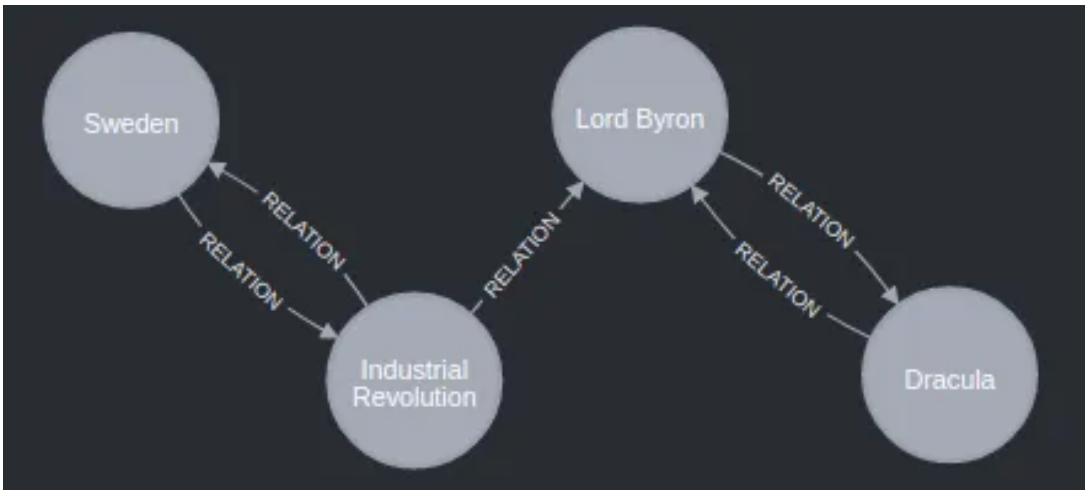
Daftar Isi

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 menelusuri artikel-artikel lain pada Wikipedia (dengan mengunjungi tautan di dalam setiap artikel) untuk menuju suatu artikel lain yang telah ditentukan sebelumnya dalam waktu paling singkat atau jumlah kunjungan ke artikel paling sedikit. Permainan WikiRace pada tugas ini melibatkan proses *web scraping*, yaitu proses untuk melakukan ekstraksi data pada suatu *website* dengan mengakses kode HTML yang ada pada laman tersebut. Tugas permainan WikiRace ini menggunakan proses *web scraping* untuk melakukan pencarian terhadap tautan yang menuju ke artikel Wikipedia lain, dan menyimpan setiap artikel yang perlu dikunjungi untuk mencapai artikel tujuan yang di-*scraping* dari artikel awal.

Batasan yang ada pada tugas ini adalah laman Wikipedia yang digunakan merupakan halaman Wikipedia berbahasa Inggris (*domain* en.wikipedia.org). Tautan yang ada pada laman Wikipedia yang bukan menuju artikel Wikipedia lainnya tidak akan dikunjungi. Pengecualian laman Wikipedia yang tidak dikunjungi pada implementasi Tugas Besar ini adalah halaman spesial pada Wikipedia yang bukan merupakan suatu artikel, seperti *file*, *template, category*, meliputi halaman Wikipedia dengan tipe laman sebagai berikut: "/File:", "/Special:", "/Template:", "/Template_page:", "/Help:", "/Category:", "Special:", "/Wikipedia:", "/Portal:", "/Talk:".



Gambar 1.1.1. Ilustrasi Graf WikiRace

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

1.2 Spesifikasi

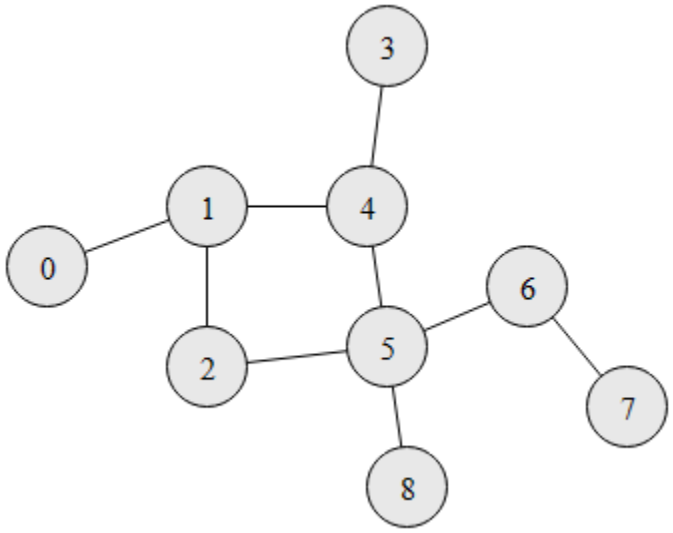
1) Membuat program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace.
2) Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan.
3) Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel, dan waktu pencarian.
4) Program mengeluarkan salah satu rute terpendek.
5) Program berbasis web, dengan front-end React dan back-end Golang.
6) Program mampu menampilkan rute terpendek yang perlu dituju dari artikel awal ke artikel akhir.
7) Program melakukan *web scraping* atau pencarian elemen pada *website* menggunakan bahasa Go untuk menemukan rute terpendek dari artikel awal ke artikel akhir.
8) Program menyimpan dan menampilkan data banyak artikel yang dilalui, waktu pemrosesan, dan rute artikel yang dilalui.

BAB II
LANDASAN TEORI

2.1 Penjelajahan Graf

Graf merupakan representasi objek-objek diskrit dan relasi antarobjek tersebut. Graf merepresentasikan objek diskrit sebagai sebuah simpul (*node*) dan hubungan antarsimpul sebagai garis [1]. Penjelajahan graf atau traversal graf merupakan metode penyelesaian graf untuk mencari relasi antara suatu simpul dengan simpul lainnya, dengan melakukan pencarian menelusuri simpul dan garis-garis relasi antarsimpul sampai ditemukan simpul tujuan.

Penjelajahan graf mengunjungi simpul-simpul pada graf secara sistematik. Penjelajahan graf terbagi menjadi dua metode umum, yaitu pencarian berdasarkan kedalaman (*Depth-first Search, DFS*) dan pencarian yang melebar (*Breadth First Search*). Algoritma BFS mengunjungi seluruh simpul pada level yang sama sebelum melanjutkan ke kedalaman yang baru, sementara algoritma DFS mengunjungi seluruh simpul pada satu jalur sebelum melanjutkan ke jalur yang lain.



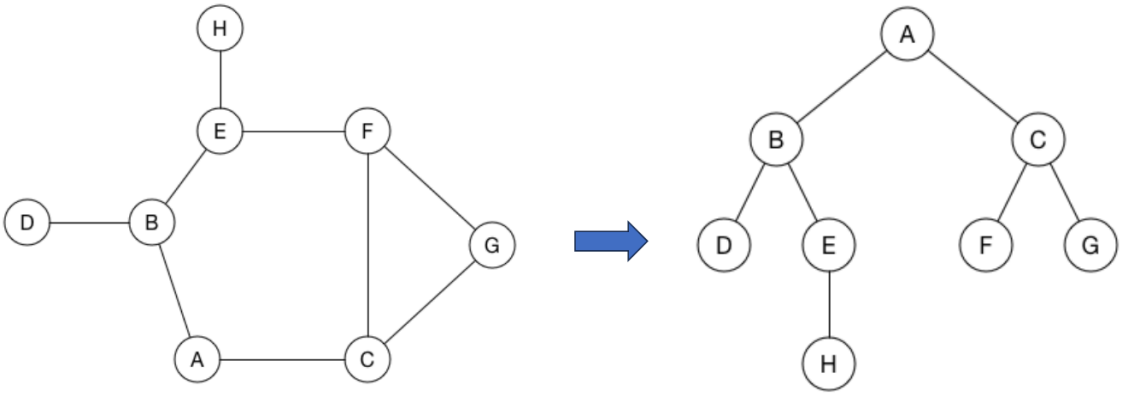
Gambar 2.1.1 Ilustrasi graf, diambil dari https://datastructures.maximal.io/graph-algorithms/graph-traversal/

Kedua algoritma tersebut dapat dikembangkan lebih lanjut dan dikombinasikan. Salah satu algoritma yang menerapkan kombinasi BFS dan DFS adalah pencarian beralgoritma IDS (*Iterative Deepening Search*), yaitu pencarian berdasarkan kedalaman hingga suatu level

tertentu, lalu pencarian dilakukan secara melebar hingga level tersebut. Jika pada suatu level kedalaman solusi masih belum ditemukan, level kedalaman akan ditingkatkan.

2.2 Algoritma BFS

Algoritma *Breadth First Search* (BFS) adalah algoritma pencarian graf secara traversal yang mencari di semua simpul di kedalaman graf saat ini sebelum mencari ke kedalaman berikutnya [2]. Algoritma ini dimulai dari sebuah titik dalam suatu graf lalu mencari di semua simpul tetangganya terlebih dahulu sebagai kedalaman pertama. BFS biasanya digunakan pada algoritma *pathfinding*, simpul terkait, dan jarak terpendek pada graf. Algoritma BFS biasanya menggunakan satu *queue* dan satu larik penyimpan titik yang sudah dikunjungi.



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

Gambar 2.2.1 diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/

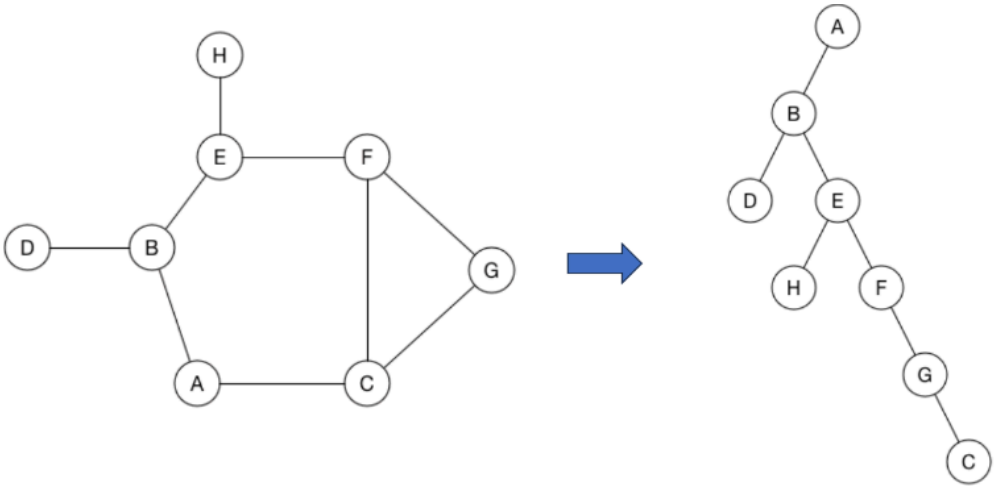
Berikut adalah langkah-langkah umum algoritma BFS:

1. Inisialisasi sebuah antrian (queue) yang akan digunakan untuk menyimpan simpul-simpul yang akan dieksplorasi. Masukkan simpul awal ke dalam antrian, dan tandai simpul tersebut sebagai sudah dikunjungi.
2. Selama antrian tidak kosong lakukan langkah-langkah:
   - Ambil simpul pertama antrian.
   - Periksa semua tetangga dari simpul tersebut yang belum dikunjungi.
   - Tandai setiap tetangga yang belum dikunjungi, masukkan mereka ke dalam antrian, dan tandai mereka sebagai sudah dikunjungi.

Pencarian akan berhenti ketika tidak ada lagi simpul yang belum dikunjungi atau simpul tujuan sudah ditemukan

2.3 Algoritma DFS

Algoritma *Depth First Search* (DFS) adalah algoritma pencarian yang bersifat rekursif, dengan mencari titik target dengan mengunjungi semua simpul yang bertetangga terlebih dahulu. Algoritma DFS akan menelusuri simpul-simpul yang bertetangga terlebih dahulu hingga kedalaman maksimal dari pohon/graf, lalu kembali ke simpul sebelumnya jika memiliki simpul tetangga yang belum dikunjungi simpul solusi, dan melanjutkan ke simpul tetangga. Algoritma DFS biasa disebut juga sebagai pencarian mendalam karena akan mencari hingga kedalaman maksimal terlebih dahulu sebelum mencari melebar. [2]



Urutan pencarian dari simpul A : A, B, D, E, H, F, G, C

Gambar 2.3.1 diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/

Algoritma DFS memiliki satu kekurangan, yaitu jika graf atau pohon bersifat dinamis, dalam artian kedalaman dapat terus meningkat, maka pencarian akan berlangsung terus menerus tak berujung hanya pada satu jalur saja. Oleh karenanya, terdapat salah satu modifikasi pada algoritma DFS, yaitu dengan membatasi pencarian secara mendalam hanya sampai mencapai level kedalaman tertentu saja. Algoritma ini disebut sebagai algoritma *Depth Limited Search* (DLS).

2.4 Algoritma IDS

Algoritma IDS (*Iterative Deepening Search*) merupakan algoritma pemecahan masalah yang melakukan serangkaian pencarian berdasarkan kedalaman (DFS), dengan pembatasan level

dan peningkatan level kedalaman pencarian hingga solusi ditemukan atau disebut juga sebagai algoritma *Depth Limited Search* (DLS) yang merupakan modifikasi dari algoritma DFS [3]. Algoritma IDS merupakan implementasi algoritma DLS dengan tingkat kedalaman tertentu, dan jika pada level tersebut solusi tidak ditemukan, tingkat kedalaman pencarian akan ditingkatkan dan pencarian akan dimulai kembali secara mendalam hingga kedalaman yang baru.

Algoritma IDS menggabungkan strategi pencarian yang menggunakan lebih sedikit memori seperti pada DFS, dengan *completeness* dari pencarian yang menyeluruh pada algoritma BFS [2]. Proses pencarian berdasarkan algoritma IDS akan menginisialisasi suatu level kedalaman yang akan membatasi pencarian secara kedalaman (DLS), lalu pencarian akan dilakukan pada seluruh jalur yang ada hingga kedalaman tersebut (BFS). Jika belum ditemukan solusi, maka tingkat kedalaman akan diperbesar dan pencarian akan diulang kembali dari awal simpul dengan kedalaman yang baru.

2.5 Aplikasi Web Berbasis React JS dan Golang

React (JS) merupakan *library* JavaScript yang digunakan untuk membangun UI pada aplikasi web. React sering digunakan dalam pengembangan aplikasi web modern, karena kemampuannya dalam memudahkan programmer untuk membuat komponen UI yang dinamis dan dapat diubah tanpa memuat ulang halaman secara keseluruhan.

Golang, atau Go, adalah bahasa pemrograman *open-source* yang dikembangkan oleh Google. Salah satu kelebihan dari Golang adalah kemampuannya untuk menjalankan kode secara konkuren. Golang memiliki Goroutine, yang pada dasarnya berupa kumpulan fungsi yang dapat berjalan secara sekaligus dan independen.

Bahasa Go memiliki beberapa *open-source library* yang dapat digunakan untuk menjalankan fungsi-fungsi tertentu sehingga *developer* tidak perlu untuk membangun suatu fungsionalitas dari awal. Kelebihan bahasa Go dalam kebebasan menggunakan *framework* yang dapat berasal dari mana saja ini membuat Go menjadi bahasa pemrograman yang fleksibel dan mudah untuk diakses di segala perangkat. Hal ini karena Go akan menyimpan seluruh *dependency* yang digunakan dalam sebuah *file* bernama go.mod yang berisi pengaturan *package*, *dependency* yang digunakan pada program tersebut.

Kerja sama antara keduanya terjadi melalui penggunaan fungsi *fetch* pada React dan API Golang. API pada Golang dapat dibuat dengan menggunakan *framework* Gin, yang memungkinkan suatu program untuk menggunakan HTTP *method* GET untuk mendapatkan data dan POST untuk mengirim data. Ketika pengguna berinteraksi dengan aplikasi web, data dikirimkan ke server melalui React menggunakan fungsi *fetch* dengan parameter yang sesuai dengan API Golang. Selanjutnya, Golang akan memproses parameter tersebut, dan mengirimkan hasil fetch dalam bentuk data JSON. Data ini kemudian ditampilkan kepada pengguna melalui React, melengkapi siklus interaksi antara pengguna dan aplikasi web.

Kombinasi React dan Golang menghasilkan aplikasi web yang cepat dan efisien. Menggunakan pustaka React yang luas, digabung dengan kinerja tinggi Golang, kombinasi tersebut menjadi salah satu opsi yang mangkus dalam pengembangan aplikasi web modern.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Pemecahan Masalah

Permasalahan yang akan diselesaikan dalam Tugas Besar ini adalah penyelesaian permainan Wikirace, yaitu mencari artikel-artikel apa saja yang perlu dilalui untuk mencapai artikel tujuan dari artikel awal pada laman Wikipedia. Batasan permasalahan ini adalah laman Wikipedia yang digunakan adalah laman dan artikel yang menggunakan bahasa Inggris (en.wikipedia.org).

Penyelesaian permainan Wikirace dimulai dengan memilih artikel awal yang akan menjadi simpul awal dan artikel akhir yang akan dijadikan tujuan atau simpul akhir untuk dituju pada tampilan *website*. Kemudian *frontend* akan mengirim *request* ke *backend* berupa data judul artikel awal dan artikel akhir. Setelah artikel awal dan artikel akhir divalidasi dan ditentukan, pengguna dapat memilih metode penyelesaian permainan, baik dengan algoritma BFS atau IDS. Pemilihan algoritma akan menentukan langkah-langkah apa saja yang ditempuh program untuk menghasilkan solusi dari permainan Wikirace tersebut. Setiap algoritma akan mencari solusi dengan menyimpan artikel-artikel yang dilalui untuk mencapai artikel tujuan, banyak artikel yang dilalui, dan waktu yang dibutuhkan untuk menghasilkan solusi permainan.

Langkah-langkah pemecahan masalah menggunakan algoritma BFS:

1. Queue berisi URL artikel mulai dibuat.
2. URL di Queue yang belum ada di map “visited” dikunjungi (*scraping*).
3. Semua URL yang ada di web dengan URL saat ini dicek apakah berjudul sama dengan artikel tujuan.
4. URL yang telah dicek dimasukkan ke map “visited” dengan value bernilai false. URL tersebut juga akan dicatat di map “history” dengan key adalah judul artikel yang bersesuaian dengan URL hasil scraping dan value adalah judul artikel yang di-scraping.
5. URL yang telah dikunjungi dihapus dari Queue dan dimasukkan ke map “visited” dengan value bernilai true.
6. Langkah 2 sampai 3 diulangi sampai URL berjudul sama dengan artikel tujuan ditemukan.

Langkah-langkah pemecahan masalah dan pencarian solusi menggunakan algoritma IDS adalah [2],

1. Ambil artikel awal dan artikel akhir dan konversi judul artikel menjadi URL Wikipedia. Judul artikel sudah dipastikan valid saat dikirim dari *frontend*. Jadikan artikel awal sebagai simpul utama.
2. Inisialisasi level kedalaman pencarian $n = 0$ terlebih dahulu, artinya program akan mengecek pohon artikel dengan kedalaman 0 atau artikel awal itu sendiri. Buat sebuah larik *s* untuk mencatat artikel apa saja yang dikunjungi oleh program dan inisialisasi dengan artikel awal.
3. Jika ternyata ditemukan bahwa artikel awal adalah sama dengan artikel akhir, program akan berhenti mencari, dan jika tidak program tidak menemukan artikel akhir, maka level kedalaman akan ditingkatkan menjadi n = n + 1.
4. Setelah level kedalaman pencarian bertambah, program akan mencari ulang dari awal atau level kedalaman $i = 0$ hingga $i = n$. Saat $i < n$, pohon akan menjadikan artikel yang ditemukan dari artikel awal menjadi sebuah daun, lalu pada artikel tersebut akan dicari lagi seluruh tautan ke artikel Wikipedia lain dan dijadikan daun dari artikel tersebut dan ditambahkan ke larik *s*.
5. Saat $i = n$, jika artikel di level kedalaman *n* tersebut bukan merupakan artikel akhir, maka akan iterasi akan kembali ke level atas dan mencari ke daun lain yang tersedia dari cabang artikel tersebut. Larik *s* akan menghapus artikel yang sudah dicek sebelumnya.
6. Jika saat level pencarian level $i = n$ artikel akhir belum ditemukan, nilai kedalaman maksimal *n* akan ditambahkan 1 lagi, dan ulangi langkah 4.
7. Setelah artikel akhir ditemukan pada salah satu daun, program akan berhenti dan keluar dari fungsi IDS dengan mengembalikan larik *s* sebagai penyimpanan rute yang dibutuhkan dari artikel awal ke artikel akhir.

Setelah menghasilkan solusi permainan, *backend* akan mengirimkan data melalui API (*Application Programming Interface*) yang berbentuk JSON (*Javascript Object Notation*) ke *frontend* untuk ditampilkan solusinya. *Frontend* akan menangkap data yang dikirimkan melalui API tersebut dan menampilkan solusi berupa artikel yang dilalui, jumlah artikel yang dicek, dan lama pemrosesan untuk mendapatkan solusi.

3.2 Mapping Permasalahan

Permainan Wikirace memerlukan suatu metode untuk melakukan pencarian tautan pada suatu artikel menggunakan suatu kakas yang bernama *web scraper*. Setiap artikel yang ditemukan dan hubungannya dengan artikel lain dapat direpresentasikan dalam bentuk graf. Permasalahan permainan Wikirace dapat dipetakan menjadi elemen-elemen pada graf sebagai berikut,

a. Artikel, direpresentasikan sebagai simpul (*node*).
b. Tautan artikel ke halaman lain akan membentuk sebuah garis (*edge*) sebagai penghubung antar artikel.

Permainan Wikirace yang dapat direpresentasikan dengan graf dapat diselesaikan pula dengan menggunakan algoritma yang mengutilisasi traversal graf, seperti algoritma BFS, DFS, maupun IDS. Penggunaan algoritma yang berbeda akan menggunakan tahapan-tahapan penyelesaian yang berbeda dan penggunaan *resource* perangkat dan waktu pemrosesan yang berbeda pula.

a. BFS

Pada permainan Wikirace, titik-titik graf pada algoritma BFS diwakili dengan artikel dan garis antartitik merepresentasikan tautan ke artikel. Graf yang digunakan adalah graf berarah. Ketetanggan suatu titik adalah artikel apa saja yang bisa dikunjungi dari artikel (titik) tersebut.

Pada algoritma BFS ini, tipe data map digunakan untuk memudahkan dan mengefisiensi pencatatan dan pembacaan data-data penting. Terdapat dua map, yaitu map “visited” sebagai penanda artikel yang sudah dikunjungi dan map “history” yang mencatat *parent* dari masing-masing artikel.

b. IDS

Algoritma IDS merupakan salah satu algoritma yang menjamin *completeness* dan penemuan solusi, dengan melakukan pencarian pada simpul sesedikit mungkin (tidak menghitung kembali simpul yang sudah dikunjungi saat peningkatan level kedalaman) [3]. Implementasi algoritma IDS dalam permainan Wikirace melibatkan pengulangan pencarian dari artikel awal ke artikel tujuan untuk level yang meningkat seiring prosesnya.

Mapping persoalan pada algoritma IDS adalah

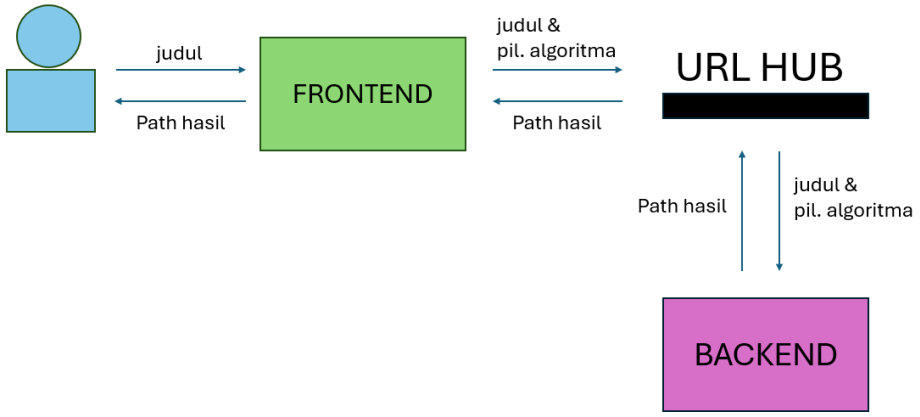
1. Artikel : simpul-simpul pada pohon.

2. Pohon dinamis : pohon yang berisi artikel awal sebagai akar pohon dan artikel yang dapat dikunjungi dari akar sebagai cabang, dan artikel dengan level kedalaman maksimal saat itu sebagai daun.

3.3 Fitur Fungsionalitas dan Arsitektur Website

Arsitektur website GoPedia meliputi *frontend* dan *backend*. *Frontend* bertugas untuk menerima judul artikel mulai dan tujuan dari pengguna dan memanggil fungsi dari *backend*. *Frontend* akan membuat URL yang berisi algoritma yang digunakan, artikel awal, dan artikel akhir, sehingga *backend* dapat mengambil nilai-nilai yang sudah dikirim dari *frontend* tersebut untuk diproses. Bagian *backend* yang sudah dijalankan akan mengambil data dari URL yang sudah dijalankan oleh *frontend* untuk menemukan jarak terpendek antara judul mulai dan judul tujuan dengan algoritma IDS atau BFS sesuai pilihan. Setelah URL dibuat, *backend* akan membaca URL yang disediakan *frontend* untuk menerima judul artikel mulai dan judul artikel akhir serta pilihan algoritma. *Backend* hanya melakukan scraping ke website dalam domain "en.wikipedia.org" untuk mencari jarak terpendek. *Backend* akan mengirimkan hasil jarak terpendek ke URL penghubung.

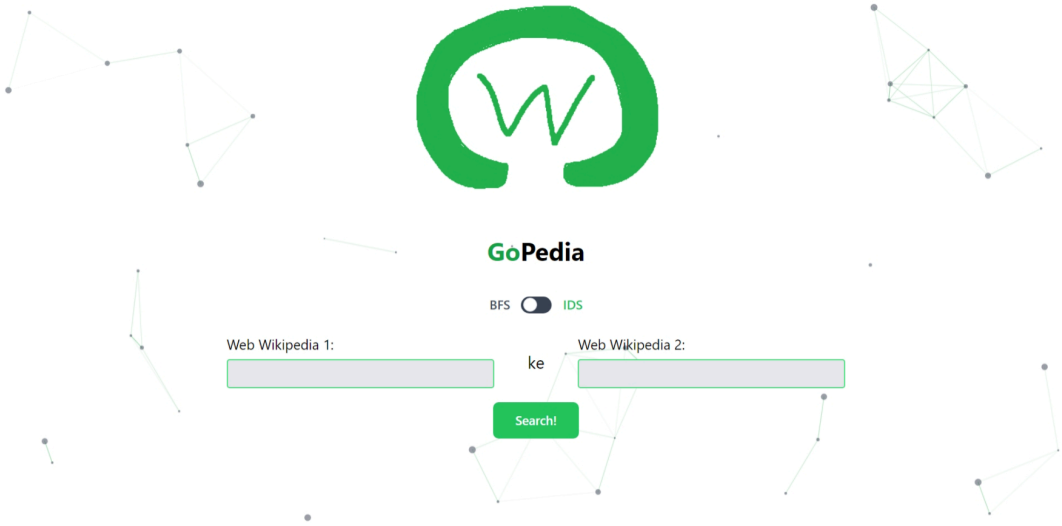
Untuk lebih memperjelas gambaran arsitektur website GoPedia, terdapat diagram berikut:



Gambar 3.3.1 Arsitektur *website* GoPedia

Website GoPedia memiliki tampilan utama yang berisi *field* masukan untuk pengguna melakukan *input* judul artikel dan *button* untuk memilih algoritma yang digunakan, BFS atau IDS. Setelah pengguna memilih artikel yang dituju dan artikel awal dan menekan *button search*, setelah

proses pencarian rute selesai, maka *backend* akan mengirim API yang akan di-*fetch* oleh *frontend*, dan menampilkan data hasil ke *frontend*.



Gambar 3.3.2 Tampilan antarmuka GoPedia

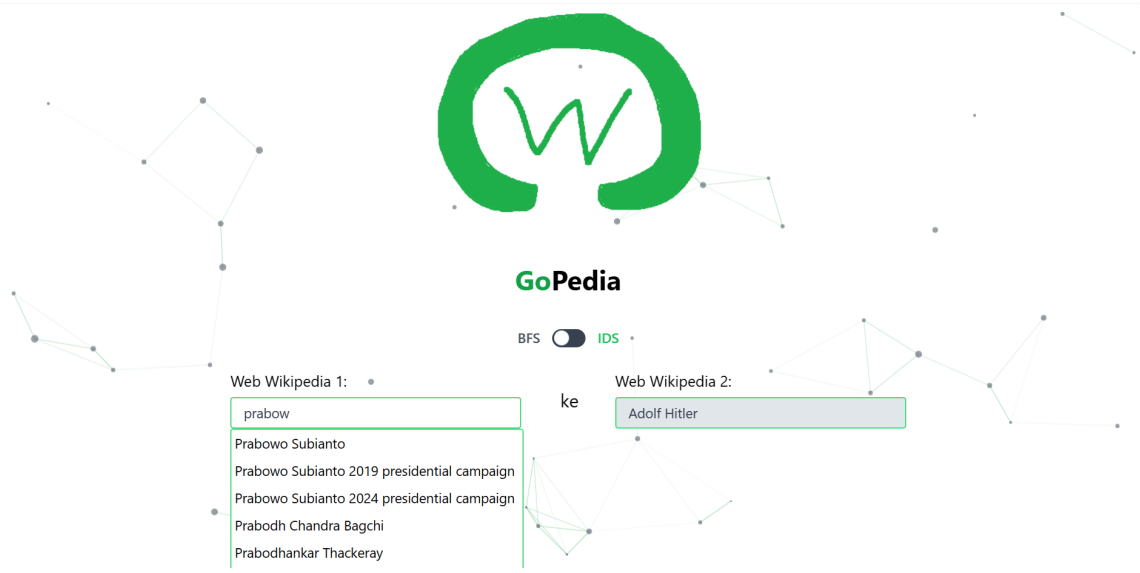
Fitur-fitur utama di *website* GoPedia meliputi:

1. Pengguna dapat memasukkan judul artikel dan menerima rekomendasi judul artikel pada *dropdown* yang muncul.
2. Pengguna dapat memilih metode algoritma yang digunakan melalui *switch button* pada halaman utama
3. Pengguna dapat menampilkan hasil rute dari artikel awal ke artikel tujuan setelah menekan tombol *search* dan menunggu proses pencarian rute selesai.
4. Pengguna dapat melihat data berupa artikel apa saja yang perlu dikunjungi untuk mencapai artikel tujuan dari artikel awal menggunakan metode *scraping*, melihat data waktu proses, dan jumlah artikel yang dikunjungi untuk dapat menemukan hasil.
5. Pengguna dapat mengunjungi/klik artikel-artikel yang menjadi rute dari artikel awal menuju artikel akhir.

Gambar 3.3.3 Tampilan hasil pencarian dari Marvel Cinematic Universe ke Jakarta

3.4 Ilustrasi Kasus

Misal pengguna ingin mencari jarak terpendek antara kata kunci/judul artikel Joko Widodo dan Adolf Hitler pada Wikipedia dengan menggunakan website GoPedia. Pengguna harus mengisi data: judul artikel mulai dan judul artikel tujuan serta pilihan algoritma. Kemudian, *frontend* akan membuat URL yang tersusun dari data-data tersebut. *Backend* akan terinvokasi setelah URL dibuat. Setelah itu, *backend* akan mengolah data untuk mencari jarak terpendek antara judul mulai ke judul tujuan sesuai algoritma yang dipilih.



Gambar 3.4.1 Tampilan antarmuka saat pengguna melakukan *input* judul artikel dan metode algoritma

Pada pemecahan masalah dengan BFS, program pertama-tama akan menginisialisasi queue dengan judul artikel mulai. Kemudian, program akan melakukan *scraping* ke website "en.wikipedia.org/wiki/Joko_Widodo". Program akan mengecek apakah terdapat URL yang bersesuaian dengan judul artikel tujuan (Adolf Hitler) dalam hasil scraping. Jika ditemukan, judul artikel saat ini akan disimpan di map "history" dengan key adalah judul artikel tujuan dan value adalah judul artikel yang sedang di-scraping. Lalu, proses scraping dan pencarian akan berhenti. Jika belum ditemukan, program akan memasukkan semua judul artikel yang didapat selama proses scraping ke queue dan map "visited" dengan key merupakan judul dan nilai adalah false. Program juga memasukkan key bernilai judul artikel hasil scraping dan value adalah judul artikel yang baru saja di-scraping. Elemen pertama (judul yang baru saja dikunjungi) akan dihapus dari queue dan dimasukkan ke map "visited" sebagai key dan nilainya adalah true. Selanjutnya, program akan terus membaca elemen pertama queue dan melakukan scraping ke website wikipedia yang berkesesuaian sampai URL dengan judul artikel tujuan ditemukan. Rute hasil dibuat dengan menelusuri map "history", dimulai dari key berisi "Adolf_Hitler" hingga menemukan value yang berisi "Joko_Widodo".

Pada pemecahan menggunakan algoritma IDS, program akan melakukan konversi judul artikel menjadi *link* artikel Wikipedia, yaitu artikel dengan judul "Joko Widodo" akan menjadi "en.wikipedia.org/wiki/Joko_Widodo", dan melakukan *scraping* ke *link* tersebut untuk mendapatkan kumpulan tautan ke artikel lain dan mulai mencari dengan algoritma IDS. Pada algoritma IDS, dibuat sebuah tipe data *map* untuk menyimpan *cache* untuk menyimpan hasil *scraping* dari suatu *website.* Jika suatu *link* sudah pernah di-*scrape*, maka saat *link* tersebut dikunjungi lagi, program akan mengambil hasil *scraping* dari *cache* daripada *scraping* ulang. Setelah proses pencarian menemukan artikel tujuan, maka program akan menyimpan seluruh artikel yang perlu dilalui untuk menuju artikel tujuan tersebut dalam sebuah larik yang menyimpan *string* judul artikel.

Setelah program *backend* berhasil menyelesaikan proses pencarian rute, maka *backend* akan menghasilkan suatu JSON yang berisi data banyak artikel yang dikunjungi, data waktu pemrosesan, dan rute dari artikel awal ke artikel akhir, lalu *frontend* akan mengambil data dari *backend* dan menampilkannya di tampilan antarmuka *website*.

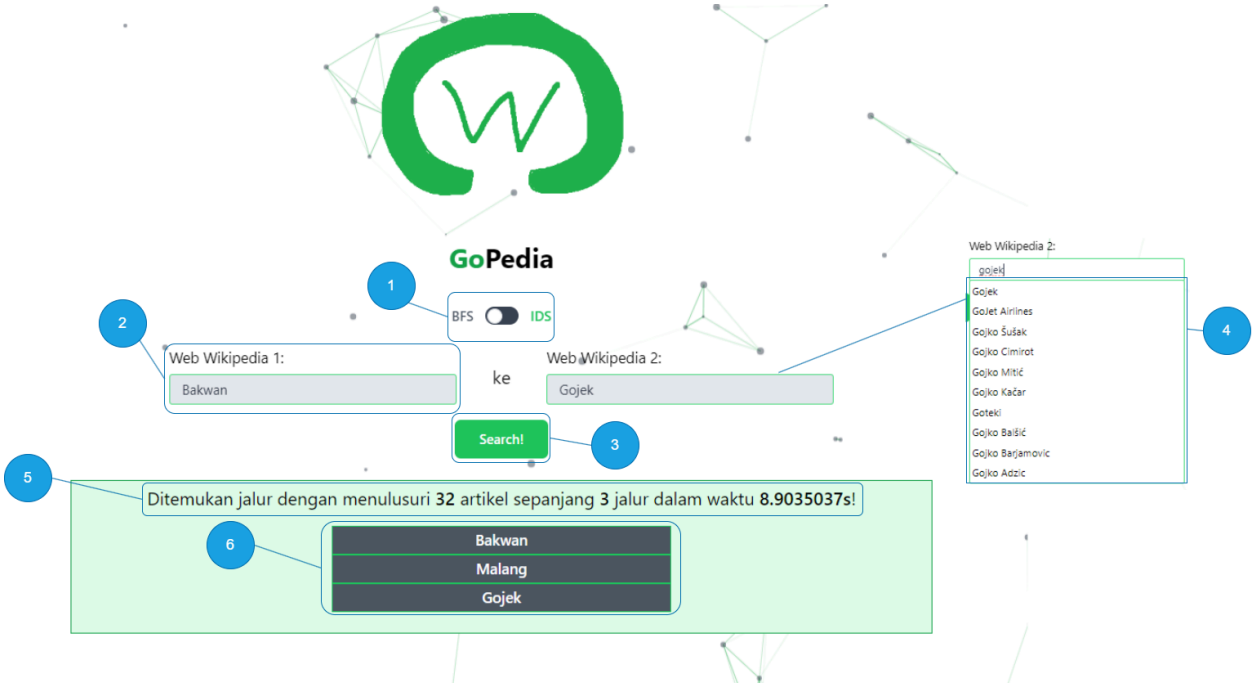
BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi dan Struktur Program

4.1.1 Struktur Program

Secara struktural, program terbagi menjadi *frontend* dan *backend*. *Frontend* merupakan kode yang jika dijalankan akan menjadi tampilan utama pada *website*, dan *backend* akan menjalankan fungsi-fungsi untuk mengolah data untuk mencari rute dari artikel awal ke artikel tujuan.

4.1.2 Spesifikasi Website



1) Toggle jenis algoritma IDS/BFS
2) Input Judul artikel
3) Tombol Search
4) AutoSuggest untuk membantu input judul artikel
5) Keluaran: Jumlah artikel yang diperiksa, panjang jalur artikel yang dilalui, serta waktu eksekusi
6) Keluaran: Jalur penjelajahan artikel

4.1.3 Implementasi Algoritma BFS

Pencarian solusi permainan Wikirace menggunakan algoritma BFS membutuhkan fungsi untuk melakukan scraping ke website dalam domain “en.wikipedia.org” sekaligus menambah elemen

pada queue dan mengecek keberadaan artikel tujuan pada hasil scraping, fungsi untuk melakukan algoritma BFS. Terdapat juga fungsi untuk menelusuri path artikel tujuan ke artikel mulai.
Selain itu, kami menggunakan *struct* map beserta fungsi untuk membaca dan memasukkan nilai yang aman untuk proses *concurrency*:

```go
type SafeBoolMap struct {
    sync.RWMutex
    SafeMap map[string]bool
}

type SafeStringMap struct {
    sync.RWMutex
    SafeMap map[string]string
}

func (rm *SafeBoolMap) Store(key string, value bool) {
    rm.Lock()
    rm.SafeMap[key] = value
    rm.Unlock()
}

func (rm *SafeBoolMap) Load(key string) (bool, bool) {
    rm.RLock()
    result, ok := rm.SafeMap[key]
    rm.RUnlock()
    return result, ok
}

func (rm *SafeStringMap) Store(key string, value string) {
    rm.Lock()
    rm.SafeMap[key] = value
    rm.Unlock()
}

func (rm *SafeStringMap) Load(key string) (string, bool) {
    rm.RLock()
    result,ok  := rm.SafeMap[key]
    rm.RUnlock()
    return result, ok
}
```

Fungsi-fungsi yang terdapat pada pemecahan masalah Wikirace menggunakan algoritma BFS adalah sebagai berikut:

1. Fungsi Scrape

```go
func scrape (currLink string, visited *SafeBoolMap, history *SafeStringMap, urlVisited *int, goal string, found *bool) ([]string){
    tempQueue := []string{}
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
    )

    c.OnRequest(func(r *colly.Request) {
        *urlVisited++
    })

    c.OnHTML("div#mw-content-text a[href]", func(e *colly.HTMLElement) {
        href := e.Attr("href")
        if strings.HasPrefix(href, "/wiki/") && !checkIgnoredLink(href) {
            kode := href[6:]
            if href == "/wiki/" + goal {
                *found = true
                history.Store(kode, currLink)
                return
            } else {
                if _, exists := history.Load(kode); !exists {
                    history.Store(kode, currLink)
                }
                tempQueue = append(tempQueue, kode)
                visited.Store(kode, false)
            }
        }
    })

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

    c.Visit("https://en.wikipedia.org/wiki/" + currLink)

    return tempQueue
}
```

Fungsi ini berfungsi untuk melakukan scraping kepada URL “currLink”, mengecek judul artikel yang didapat dan menambahkan semua judul artikel hasil ke “tempQueue”. Fungsi ini juga berfungsi memetakan sudah atau belumnya sebuah URL dikunjungi ke map “visited” dan memetakan anak (URL hasil scraping) ke orangtuanya(URL yang sedang discraping).

2. BFS

```go
func BFS(start string, goal string, urlVisited *int) ([]string, bool) {
    var shortestPath []string
    var tempQueue []string
    var queue []string
    var parent string
    found := false
    visited := SafeBoolMap{SafeMap : make(map[string]bool)}
    history := SafeStringMap{SafeMap : make(map[string]string)}

    startTime := time.Now()

    tempQueueChan := make(chan []string)

    go func() {
        tempQueue := scrape(start, &visited, &history, urlVisited, goal, &found)
        tempQueueChan <- tempQueue
    }()
    tempQueue = <-tempQueueChan
    visited.Store(parent, true)

    limiter := make(chan int, 100)
    var wg sync.WaitGroup
    for !found {
        queue = []string{}
        queue = append(queue, tempQueue...)
        tempQueue = []string{}
        for _, element := range queue{
            wg.Add(1)
            limiter <- 1
            go func(link string) {
                defer wg.Done()
                if isVisited, _ := visited.Load(link); !isVisited {
                    tempQueue = append(tempQueue, scrape(link, &visited, &history, urlVisited, goal, &found)...)
                    visited.Store(parent, true)
                }
                <-limiter
            }(element)
            if (found) {
                break
            }
        }
        wg.Wait()
    }

    end := time.Now()
    fmt.Println("Waktu eksekusi:", end.Sub(startTime))
    fmt.Println("Url visited:", urlVisited)
    if (found) {
        fmt.Println(goal)
        shortestPath = getResult(history.SafeMap, goal, start)
        fmt.Println(shortestPath)
    } else {
        fmt.Println("Goal not found")
    }
    return shortestPath, found
}
```

Fungsi ini berfungsi untuk mencari jarak terpendek dari artikel mulai dan artikel akhir dengan algoritma BFS.

3. getResult

```go
func getResult(history map[string]string, start string, goal string) []string {
    var result []string
    key := start
    for key != goal {
        result = append(result, key)
        key = history[key]
    }
    result = append(result, goal)
    return result
}
```

Fungsi ini digunakan untuk membuat path dari “start” ke “goal” dengan menggunakan data dari map history.

4.1.4 Implementasi Algoritma IDS

Pencarian solusi permainan Wikirace menggunakan algoritma IDS membutuhkan fungsi untuk meningkatkan level kedalaman pencarian jika saat suatu level kedalaman solusi masih belum ditemukan, dan fungsi untuk melakukan pencarian secara DLS atau DFS sampai level tertentu. Pada implementasi menggunakan bahasa Golang, algoritma IDS memerlukan fungsi IDS() yang melakukan iterasi hingga suatu level kedalaman tertentu dan meningkatkan level jika pada suatu kedalaman hasil masih belum ditemukan, dan fungsi DLS() untuk melakukan pencarian secara DFS.

Fungsi-fungsi yang dibutuhkan untuk mencari rute dari artikel awal ke artikel akhir menggunakan algoritma IDS adalah sebagai berikut,

1. Fungsi untuk mendapatkan semua tautan pada suatu artikel dipecah menjadi fungsi getAllLinks(), yaitu fungsi *scraping* menggunakan *library* GoColly, dan cacheLinks(), yaitu fungsi untuk membuat *cache* sementara hanya selama keberjalanan program saja. Jika program digunakan untuk mencari artikel awal dan artikel tujuan yang baru, maka *cache* akan dihapus.

```go
// Global variables
var linkCache = make(map[string][]string)
var visited = make(map[string]bool)

func getAllLinks(url string) []string {
    //c := colly.NewCollector(colly.CacheDir("./cache"))
    c := colly.NewCollector()

    // Inisialisasi array
    var links []string

    // Cari semua link dan kalau berawalan /wiki/ ditambahkan, dan jika ada yang mengandung ignoredLinks diabaikan
    c.OnHTML("div#mw-content-text a[href]", func(e *colly.HTMLElement) {
        link := e.Attr("href")
        if strings.HasPrefix(link, "/wiki/") && !checkIgnoredLink(link) {
            links = append(links, "https://en.wikipedia.org"+link)
        }
    })

    err := c.Visit(url)
    if err != nil {
        return nil
    }
    return links
}

func cacheLinks(url string) ([]string, bool) {
    links, exists := linkCache[url]

    if exists {
        return links, true
    }

    links = getAllLinks(url)

    linkCache[url] = links
    return links, false
}
```

2. Fungsi untuk menjalankan algoritma IDS. Fungsi ini akan memanggil fungsi DLS sampai ditemukan artikel tujuan pada pencariannya. Algoritma IDS akan menginisialisasi nilai banyak artikel yang dilalui dan akan mengembalikan larik yang berisi rute dari artikel awal ke artikel tujuan dan banyak artikel yang dilalui. Fungsi IDS akan dipanggil oleh fungsi main() pada file main.go.

```go
func IDS(startURL, targetURL string, maxDepth int, numOfArticles *int) ([]string, int, bool) {
    i := 1
    var result []string

    ch := make(chan []string, maxDepth)
    go func(ch chan []string) {
        for {
            result, success := DLS(startURL, targetURL, i, result, numOfArticles)
            if success {
                ch <- result
                close(ch)
                return
            }
            fmt.Println(i)
            i++
            if i > maxDepth { // Safe condition only
                ch <- nil
                close(ch)
                return
            }
        }
    }(ch)
    result = <-ch
    linkCache = make(map[string][]string)
    visited = make(map[string]bool)
    return result, *numOfArticles, result != nil
}
```

3. Fungsi untuk menjalankan algoritma DLS dengan maksimum kedalaman yang didapatkan dari algoritma IDS. Fungsi DLS adalah fungsi yang bersifat rekursif karena akan memanggil dirinya sendiri untuk mengecek dan mengunjungi artikel baru yang didapat dari artikel sebelumnya, artinya menuju level yang lebih dalam lagi. Fungsi ini pertama akan memanggil fungsi cacheLinks untuk mendapatkan kumpulan *link* yang terdapat pada artikel tersebut. Setelah didapatkan kumpulan *link*, maka fungsi ini akan mengunjungi satu per satu *link* artikel yang didapat dan akan mengembalikan nilai True jika ditemukan bahwa artikel yang dikunjungi adalah sama dengan artikel tujuan. Fungsi ini akan menambahkan artikel yang dikunjungi ke sebuah larik *result* jika menemukan artikel tujuan dalam pencariannya.

```go
func DLS(currentURL string, targetURL string, limit int, result []string, numOfArticles *int) ([]string, bool) {
    if currentURL == targetURL {
        return result, true
    }

    if limit <= 1 || visited[currentURL] {
        return nil, false
    }

    visited[currentURL] = true
    defer delete(visited, currentURL)
    links, cached := cacheLinks(currentURL)
    if !cached {
        *numOfArticles++
    }

    for _, link := range links {
        //fmt.Println("Cek link", link)
        newPath, found := DLS(link, targetURL, limit-1, append(result, link), numOfArticles)
        if found {
            return newPath, true
        }
    }
    return nil, false
}
```

4.2 Penggunaan Program

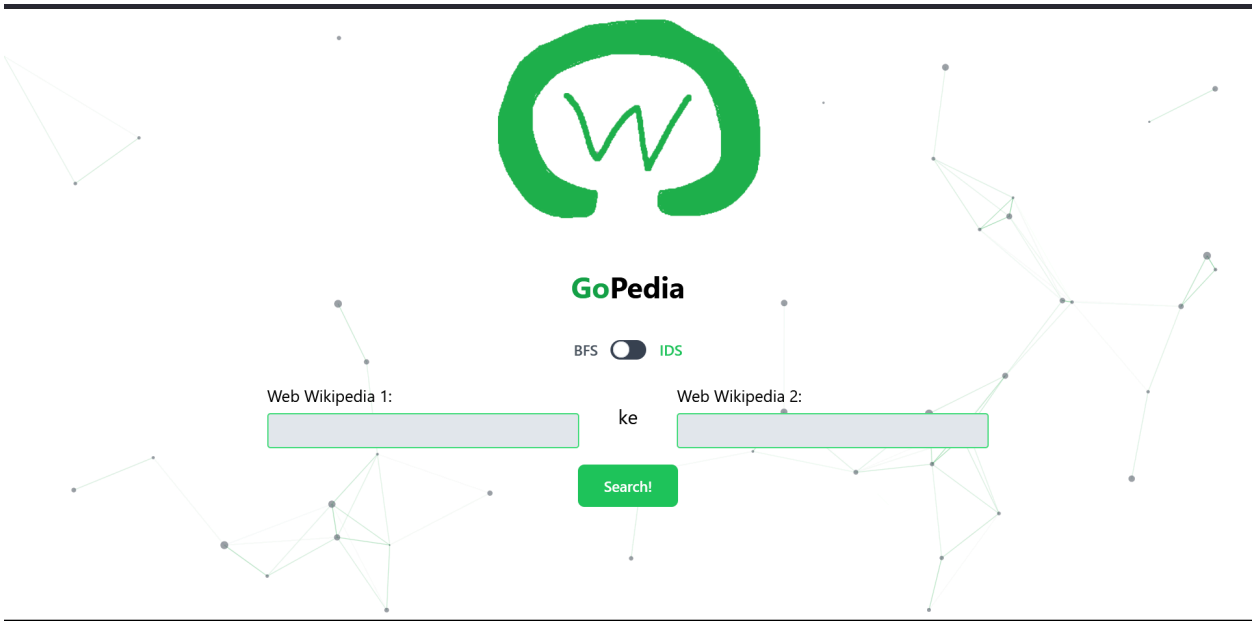
Program terbagi menjadi dua bagian, yaitu *frontend* untuk menyediakan tampilan antarmuka dan *backend* yang memproses pencarian rute dari artikel awal ke artikel akhir. Cara instalasi program dapat dilihat melalui laman Github *backend* dan *frontend*. Setelah program sudah dipasang, program dapat dijalankan dengan melakukan *run* pada folder *frontend* Tubes2_FE_GoPedia dengan mengetikkan *command* berikut pada *terminal*:

```
docker run gopedia-frontend
```

Untuk menjalankan bagian *backend*, masuk ke *folder backend* Tubes2_BE_GoPedia dan ke folder src, lalu jalankan *command* berikut melalui terminal:

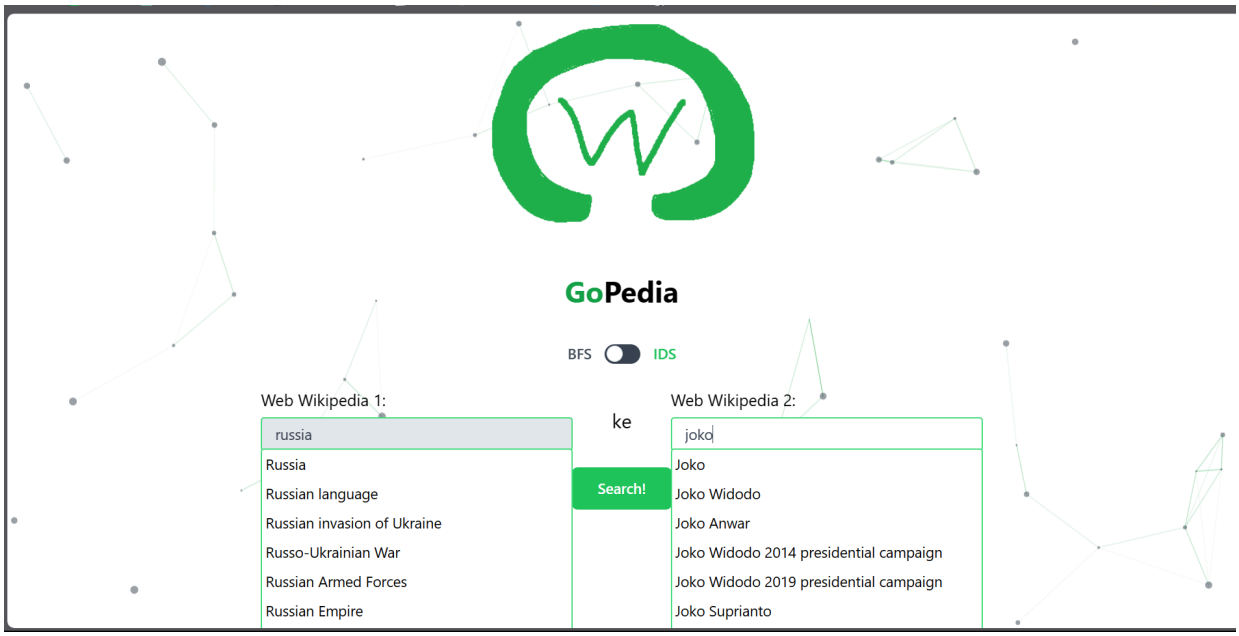
```
docker run gopedia-backend
```

Setelah program berjalan, program dapat dibuka melalui `localhost:5173/` di peramban pada perangkat. Setelah *website* berjalan, akan muncul tampilan sebagai berikut,

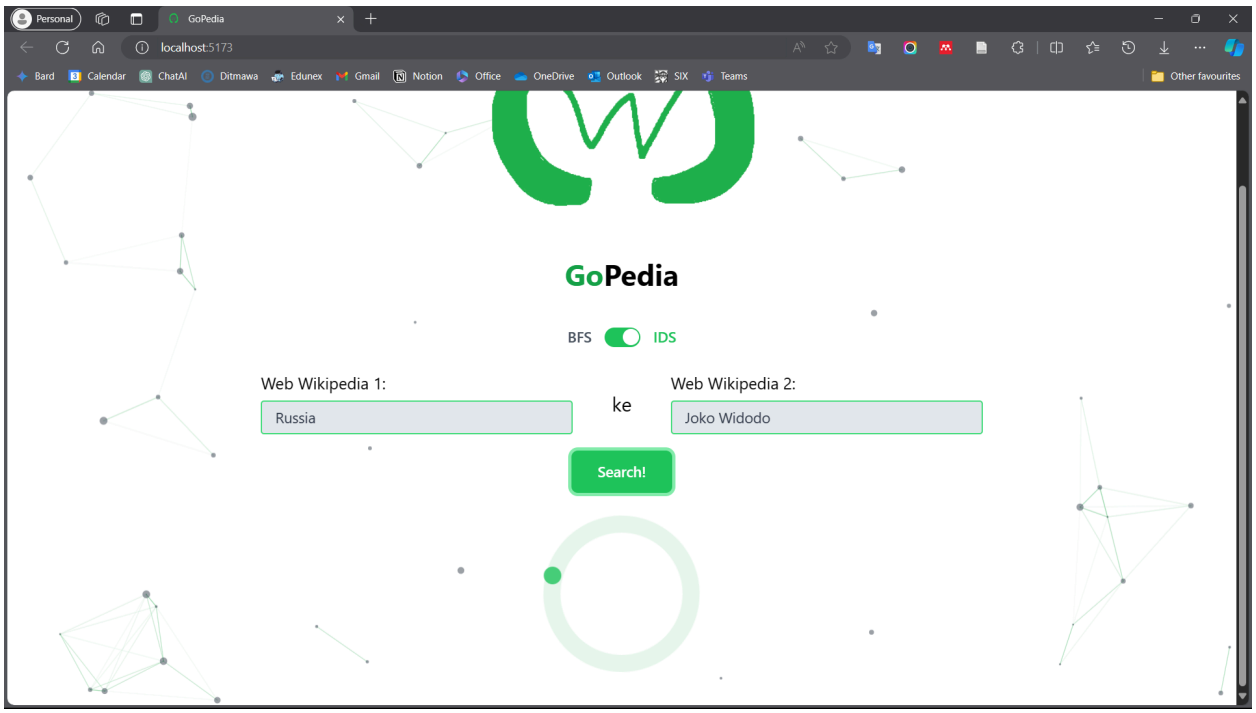


Setelah muncul tampilan tersebut, pengguna dapat memilih algoritma melalui *switch button* di bagian bawah logo, saat *switch* berada di kiri dan berwarna abu, berarti metode BFS yang terpilih, dan saat *switch* berada di kanan dan berwarna hijau, metode IDS yang terpilih. Setelah memilih algoritma yang digunakan, pengguna dapat memasukkan judul artikel awal dan artikel akhir, dan program akan memberikan validasi judul artikel Wikipedia dalam bentuk *dropdown*

yang dapat diklik oleh pengguna. Sebagai contoh akan digunakan artikel Indonesia dan Pitbull (*rapper*) sebagai artikel awal dan artikel akhir.



Setelah pengguna memilih judul artikel yang akan digunakan dalam permainan Wikirace, tekan tombol *search* untuk memulai pencarian. Sebuah animasi *loading* akan muncul yang menandakan program sedang menunggu hasil pencarian selesai diolah dari *backend*.



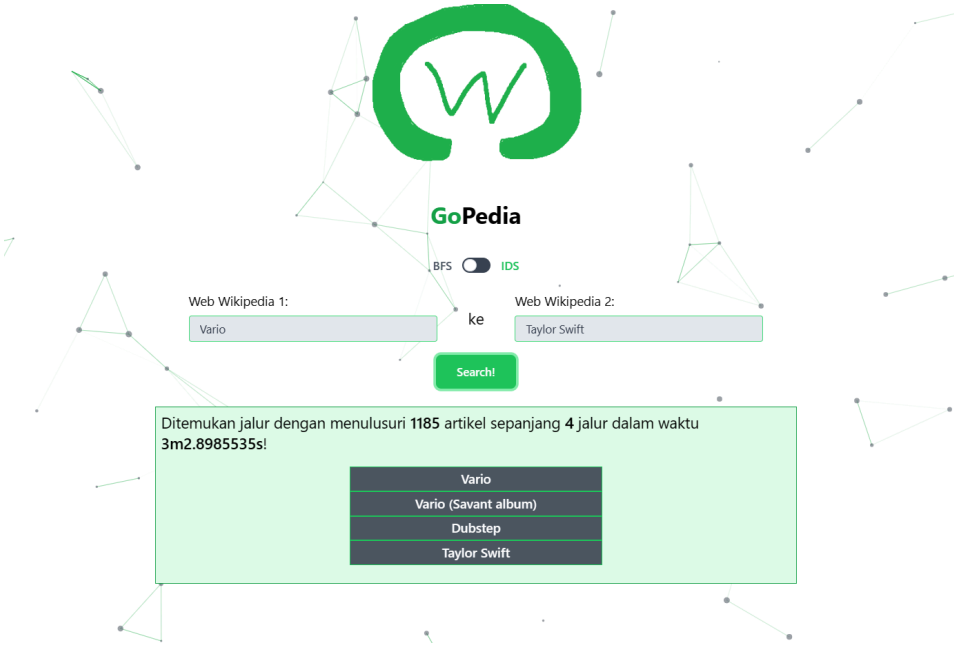
Setelah animasi *loading* menghilang, sebuah kotak akan muncul menampilkan solusi dari permainan Wikirace dengan judul artikel awal hingga artikel akhir, beserta artikel apa saja yang dilalui dan lama waktu proses yang dibutuhkan.

4.3 Pengujian

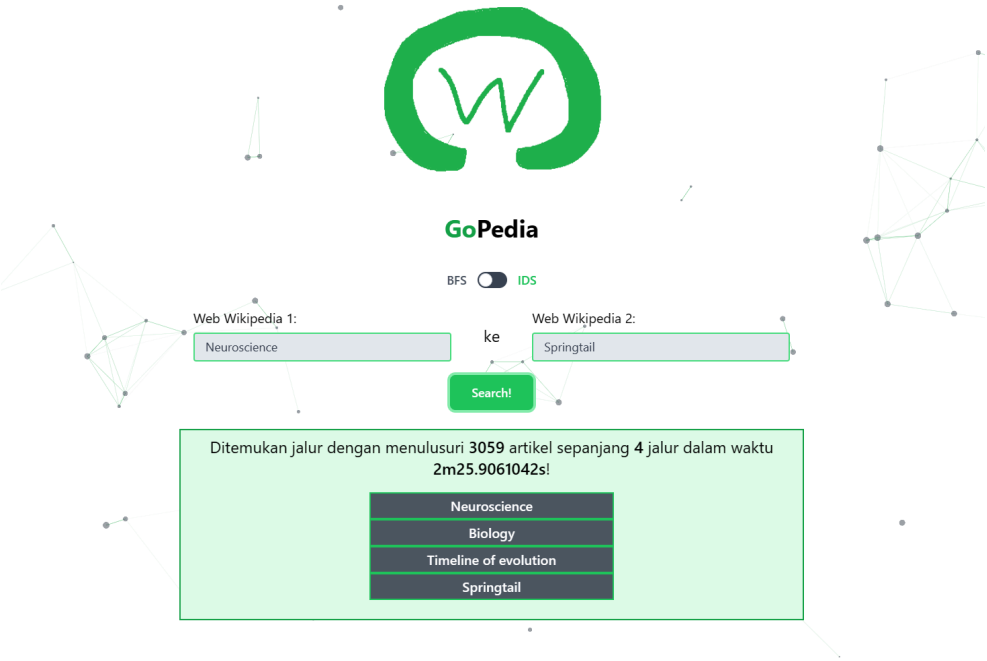
4.3.1 BFS

1. Artikel awal: Lion
   Artikel tujuan: Deer
   Jumlah artikel yang dilalui: 3 (Lion, 10th edition of Systema Naturae, Deer)
   Jumlah artikel yang dikunjungi: 32 tanpa concurrency, 105 dengan concurrency
   Waktu proses: 2.2099352 detik tanpa concurrency, 1.74311 detik dengan concurrency

2. Artikel awal: Vario
   Artikel tujuan: Taylor Swift
   Jumlah artikel yang dilalui: 4 (Vario, Vario (savant album), Dubstep, Taylor Swift)
   Jumlah artikel yang dikunjungi: 1185 tanpa concurrency, 161 dengan concurrency
   Waktu proses: 3 menit 2.895535 detik tanpa concurrency, 2.0508269 detik dengan concurrency

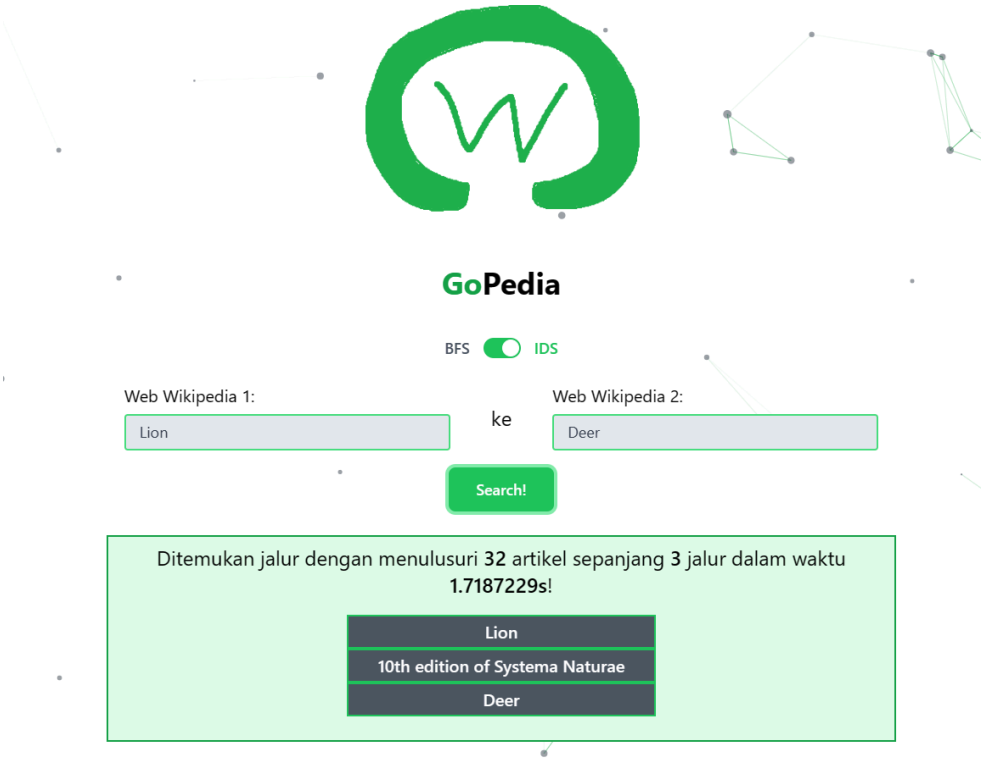


3. Artikel awal: Neuroscience
   Artikel tujuan: Springtail
   Jumlah artikel yang dilalui: 4 (Neuroscience, Biology, Timeline of evolution, Springtail)
   Jumlah artikel yang dikunjungi: 3059
   Waktu proses: 2 menit 25.9061042 detik



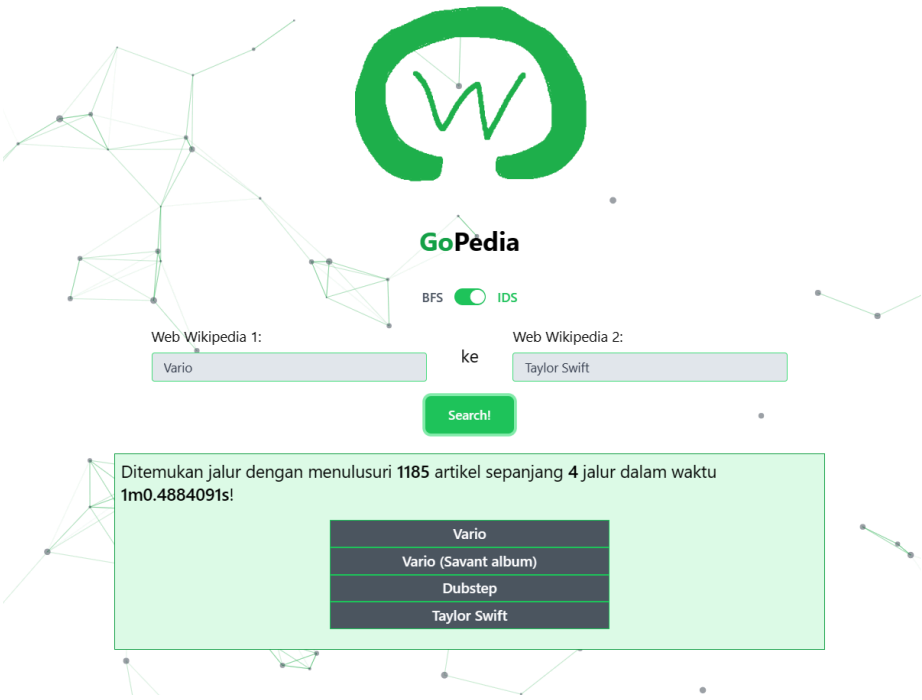
4.3.2 IDS

1. Artikel awal: Lion
   Artikel tujuan: Deer
   Jumlah artikel yang dilalui: 3 (Lion, 10th edition of Systema Naturae, Deer)
   Jumlah artikel yang dikunjungi: 32 artikel
   Waktu proses: 2.0338597 detik tanpa concurrency, 1.7187 detik dengan concurrency

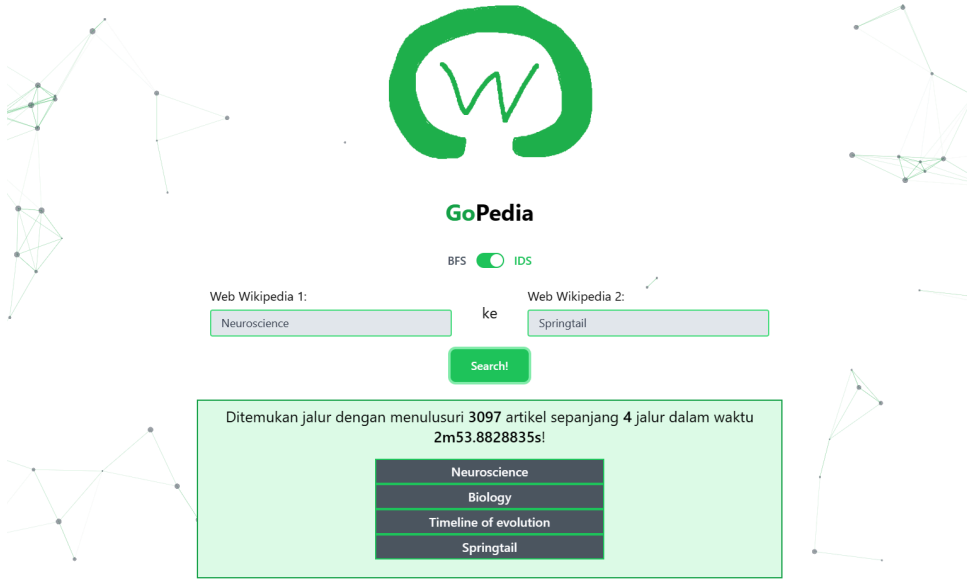


2. Artikel awal: Vario

Artikel tujuan: Taylor Swift
Jumlah artikel yang dilalui: 4 (Vario, Vario (savant album), Dubstep, Taylor Swift)
Jumlah artikel yang dikunjungi: 1185
Waktu proses: 1 menit 0.4884091 detik tanpa concurrency, 48.471 detik dengan concurrency



3. Artikel awal: Neuroscience
   Artikel tujuan: Springtail
   Jumlah artikel yang dilalui: 4 (Neuroscience, Biology, Timeline of evolution, Springtail)
   Jumlah artikel yang dikunjungi: 3097
   Waktu proses: 2 menit 53.8828835 detik



4.4 Analisis Hasil

Secara teori BFS dan IDS memiliki *time complexity* yang sama, yaitu $O(b^d)$ dan *space complexity* $O(b^d)$ untuk BFS dan O(bd) untuk IDS. Nilai b adalah jumlah node/titik dan nilai d adalah kedalaman. Berdasarkan 3 contoh kasus di atas, dapat diamati bahwa algoritma BFS secara umum lebih cepat daripada IDS pada kasus dengan *depth* yang besar (lebih dari sama dengan 3) dan lebih lambat jika *depth* lebih dari tiga. Waktu IDS yang lebih lama untuk *depth* yang lebih tinggi disebabkan karena penerapan IDS yang mengulang kembali iterasi dari *node* artikel awal untuk setiap penambahan level, sementara BFS akan melanjutkan langsung ke level yang lebih dalam meskipun pencarian dilakukan secara melebar awalnya. BFS menyimpan data dengan jumlah yang bertambah secara eksponensial seiring bertambahnya waktu yang menyebabkan penggunaan RAM yang meningkat secara eksponensial juga. Di lain sisi, penerapan IDS menggunakan sistem *cache* untuk menelusuri judul yang sudah pernah di-*scrape* sehingga penggunaan RAM dan memorinya tidak terlalu besar.

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 berbasi website dengan menggunakan bahasa Golang untuk membuat Wikirace, yaitu mesin pencari jalur terpendek antara dua judul artikel pada wikipedia, dengan algoritma BFS dan DFS. Kami menggunakan bahasa Javascript untuk bagian frontend dan golang untuk backend. Dari pengujian yang telah dilakukan, algoritma BFS memiliki kecepatan pencarian yang lebih baik dibandingkan algoritma IDS, terutama saat level kedalaman artikel yang dituju cukup tinggi. Hal ini dikarenakan algoritma IDS akan memulai pencarian dari artikel awal lagi saat level kedalaman pencarian ditambahkan. Namun algoritma IDS memiliki penggunaan memori yang cukup lebih baik pada kebanyakan kasus karena program tidak perlu mencari melebar dahulu sebelum selesai mencari mendalam, dan saat pencarian terakhir (melebar) tidak ditemukan, program melakukan pencarian mendalam kembali dengan level yang baru sehingga pada kebanyakan kasus penggunaan memorinya lebih sedikit.

5.2 Saran dan Refleksi

Dalam pembuatan program ini, dipelajari berbagai hal, baik dalam pengembangan *frontend* maupun *backend*. Dalam pengembangan *frontend*, belajar banyak terkait penggunaan React, dalam membuat web interaktif, fetch data dari API *backend*, dan penggunaan pustaka luasnya React, terutama animasi *background* yang dipakai dalam GoPedia. Saran untuk kedepannya adalah memisahkan bagian kode menjadi komponen-komponen React (selain untuk *background* yang sudah menjadi komponen), supaya kode utama menjadi lebih mudah dibaca dan dipelihara.
Dalam sisi *backend*, dipelajari tentang aplikasi BFS serta IDS, keunikan bahasa golang, serta cara membuat API sebagai output data untuk *frontend*. Saran untuk kedepannya, sebelum mulai memrogram, berfikir dahulu struktur kode yang baik. Karena struktur kode yang baik akan sangat memudahkan memrogram dalam jangka panjang maupun pendek. Selain itu, khusus untuk IDS, belajar concurrency lebih baik lagi karena concurrency berupa konsep yang lumayan kompleks.

LAMPIRAN

- Tautan *repository* Github:
  https://github.com/bagassambega/Tubes2_BE_GoPedia
  https://github.com/bagassambega/Tubes2_FE_GoPedia

DAFTAR PUSTAKA

[1] Rinaldi Munir, “Matematika Diskrit”, Penerbit Informatika Bandung, Oktober 2020
[2] Rinaldi Munir, salindia perkuliahan IF2211 Strategi Algoritma Institut Teknologi Bandung, https://informatika.stei.itb.ac.id/~rinaldi.munir/
[3] geeksforgeeks.org, “*Iterative Deepening Search (IDS) or Iterative Deepening Depth First Search*), diakses pada 21 April 2024. https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/
[4] Yaroslav Bai, “Best Practices: Why Use Golang For Your Project”, diakses pada 25 April 2024, https://www.uptech.team/blog/why-use-golang-for-your-project