Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Tugas Besar 2 IF2211 Strategi Algoritma

Oleh:

Kelompok JoJiKa

1. 13522121 - Jonathan Emmanuel Saragih
2. 13522125 - Satriadhikara Panji Yudhistira
3. 13522128 - Mohammad Andhika Fadillah

IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

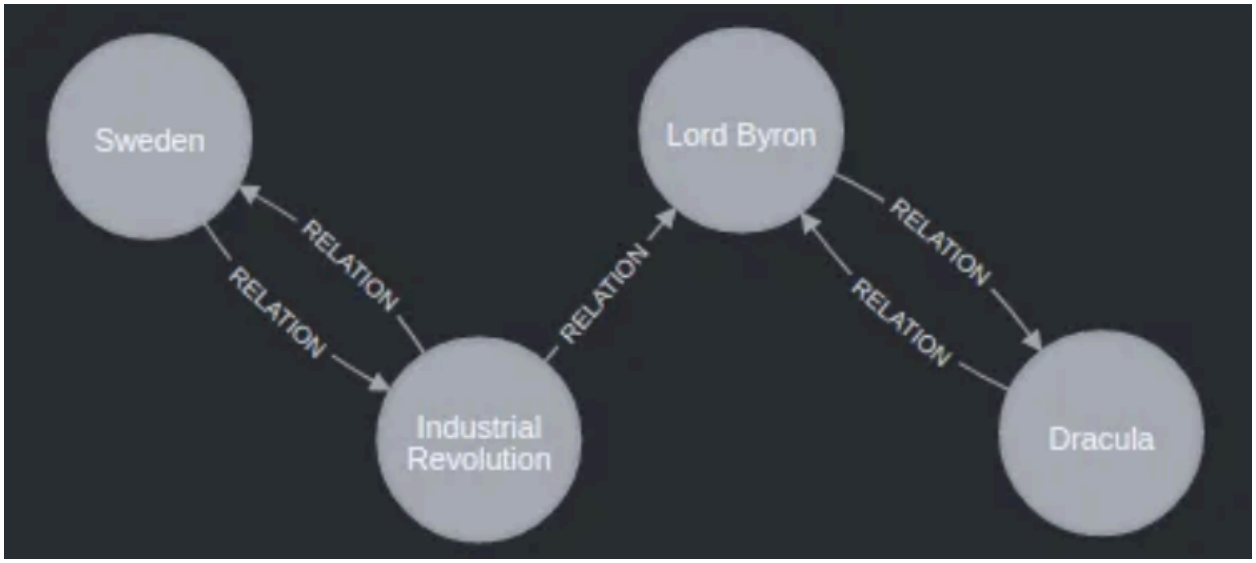
INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

BAB 1

DESKRIPSI MASALAH

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.



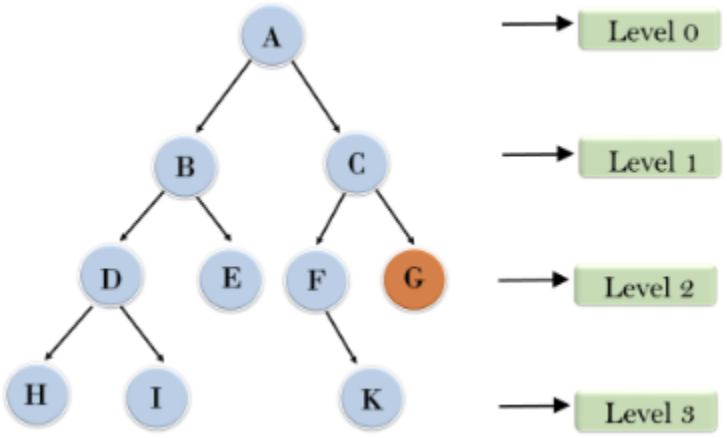
BAB 2
TEORI DASAR

2.1 Penjelajahan Graf

Graf adalah sebuah struktur data yang yang terdiri dari *verteks* atau *node* (simpul) dan juga *edge* (sisi). Sisi pada graf berguna untuk menghubungkan simpul - simpul yang ada pada graf. Dalam tugas ini, graf digunakan untuk memodelkan dan mencari hubungan antar halaman Wikipedia.Setiap halaman diwakili oleh simpul dan hyperlink yang menghubungkan antar halaman sebagai sisi.

Dalam menjelajah Graf, kelompok kami menggunakan Algoritma IDS (Iterative Deepening Search) dan juga BFS (Breadth-First Search). Dalam menjelajahi Wikipedia, penjelajahan digunakan untuk mengunjungi dan mengambil data data yang diperlukan dalam tiap halamannya, yang dapat diakses untuk ke halaman lainnya. Dengan menggunakan graf, pencaharian yang dilakukan menjadi lebih optimal. State dapat disimpan dalam bentuk node dan hubungan hubungannya disimpan sebagai sisi graf.

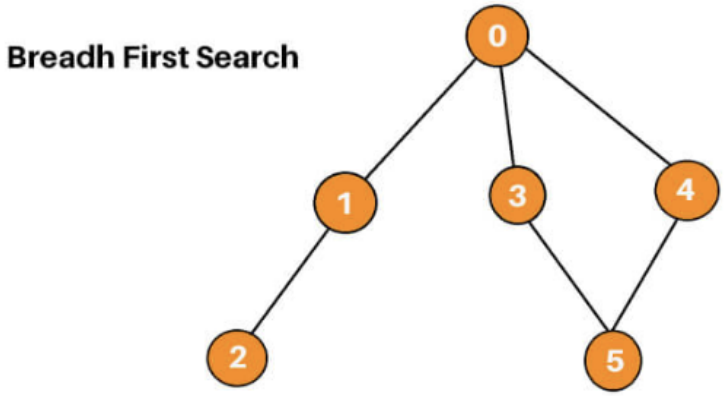
2.2 Iterative Deepening Search (IDS)



Gambar 2.2. Iterative Deepening Search

Iterative Deepening Search (IDS) adalah sebuah strategi pencarian yang menggabungkan pendekatan pencarian kedalaman terbatas (depth first search) dengan pencarian melebar (breadth first search). Metode ini digunakan terutama dalam konteks pencarian pada struktur data seperti pohon pencarian (search tree) atau graf.

2.3 Breadth First Search (BFS)



Gambar 2.3. Breadth First Search

Algoritma Breadth-First Search (BFS) memilih simpul berikutnya secara melebar. Dalam BFS, setelah suatu di simpul diproses, simpul simpul tetangganya akan ditambahkan ke dalam suatu *queue*, lalu akan memilih simpul berikutnya berdasarkan *queue* tersebut. Struktur data *queue* digunakan karena sifat *First In First Out* (FIFO) dari struktur data *queue*. Algoritma ini akan jalan sampai semua simpul sudah dikunjungi atau simpul tujuan sudah ditemukan. Algoritma BFS lebih optimal daripada DFS dalam pemecahan masalah rute terpendek dalam graph tidak berbobot karena sifat pencariannya yang melebar.

2.4 Website

Dalam membuat website, perlu adanya pemilihan teknologi yang cocok dan sesuai dengan kebutuhan dibuatnya website. Dalam tugas kali ini, terdapat teknologi yang dipakai yaitu sebagai berikut :

1. Vite: Sebagai alat build modern, Vite memiliki kecepatan yang sangat cepat dalam proses development. Hal tersebut bisa terjadi karena adanya fitur hot module replacement. Kecepatan ini sangat menguntungkan dalam tahap pengembangan, memungkinkan perubahan instan terhadap kode yangmemiliki dampak langsung pada output tanpa delay.

2. Tailwind CSS: Teknologi TailWind menerapkan pendekatan utility-first, sehingga dengan adanya teknologi tersebut, kelompok kami dapat mengimplementasikan desain responsif dengan cepat dan efisien. Kegunaan utamanya adalah mempercepat proses styling dengan menyediakan kelas-kelas yang mudah digunakan langsung pada elemen HTML.

3. React: Pemilihan Kami memiliki React sebagai library utama kami karena kemampuannya mengelola state dan UI secara efektif. React membantu dalam mengembangkan komponen yang dinamis dan mudah untuk di-maintain.

4. TypeScript: Kami memilih untuk menggunakan TypeScript karena teknologi ini dapat memberikan layer tambahan keamanan dalam bentuk type checking yang dapat dilakukan sebelum kode dijalankan. Hal ini sangat membantu dalam mengurangi potensi bug dan meningkatkan kualitas kode pada aplikasi skala besar.

Dengan mengintegrasikan teknologi-teknologi tersebut, kelompok berhasil membangun sebuah website yang sesuai kebutuhan yang diperlukan. Setiap teknologi yang kami pilih, memiliki kelebihan yang bisa membuat website pada tugas ini menghasilkan hasil yang lebih baik.

BAB 3
ANALISIS PEMECAHAN MASALAH

3.1 Langkah - langkah Pemecahan Masalah

Dalam membangun solusi berbasis GO yang digunakan untuk memecahkan masalah Wikipedia Race, kelompok kami melakukan langkah langkah berikut,

1. Memahami persoalan yang ingin diselesaikan, serta dekomposisi masalah tersebut menjadi elemen-elemen dalam persoalan graph.
2. Menemukan algoritma dasar BFS dan IDS yang sesuai dengan persoalan Wikipedia Race.
3. Merancang tampilan dari *Website*
4. Mempelajari syntax dan semantics dari bahasa GO, serta framework untuk tampilan website yang akan digunakan untuk menampilkan solusi.
5. Membuat algoritma BFS dan IDS dalam bahasa GO.
6. Membuat Website menggunakan framework yang telah ditentukan
7. Menghubungkan Front-end dan Back-end yang menggunakan algoritma BFS dan IDS yang telah dibuat

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

3.2.1 Breadth First Search

Dalam pemecahan persoalan Wikipedia Race dengan algoritma BFS, Algoritma Breadth First Search (BFS) diterapkan untuk menelusuri halaman-halaman Wikipedia. Penggunaan *queue* memungkinkan pencatatan link-link yang akan dikunjungi selanjutnya, sementara peta (visited) digunakan untuk menandai URL yang sudah dikunjungi. Selain itu, peta kosong path diinisialisasi untuk menyimpan jalur dari startURL ke setiap URL yang sudah dikunjungi.

3.2.2 Iterative Deepening Search

Dalam pemecahan persoalan Wikipedia Race dengan algoritma IDS, algoritma ini menggunakan konsep algoritma searching dan graf dinamis. Dalam kasus ini, Node merepresentasikan halaman - halaman yang ada pada wikipedia, dimana *parent akan* menelusuri jalur yang dilalui, dan *children* merupakan daftar node anak yang merepresentasikan halaman yang dapat diakses dari halaman saat ini.

Dengan adanya algoritma IDS, program ini akan menelusuri halaman halaman wikipedia dengan menggunakan Depth-Limited Search (DLS) dengan kedalaman yang

akan meningkat secara bertahap. DLS sendiri merupakan pencaharian halaman halaman berdasarkan kedalaman tertentu. Jika kedalaman adalah 0 maka pencaharian akan berhenti. Pencaharian akan terus berjalan hingga hasil ditemukan dan proses tersebut akan diulangi dengan cara rekursif.

Node	Halaman wikipedia
Edge	Hubungan antar halaman (hyperlink)

3.3 Fitur fungsional dan arsitektur aplikasi web yang dibangun

Dalam tugas besar ini, terdapat beberapa fitur fungsionalitas yang dibuat untuk menjalankan program ini.

1. Pencarian jalur atau path :
    a. Algoritma BFS : Digunakan untuk mencari jalur terpendek antara satu atau lebih halaman wikipedia dengan algoritma BFS.
    b. Algoritma IDS : Digunakan untuk mencari jalur terpendek antara satu atau lebih halaman wikipedia dengan algoritma IDS.
2. Input Pengguna :
    a. Pengguna memasukan start dan end point
    b. Pengguna memilih antara BFS dan IDS untuk mencari path.
3. Tampilan Hasil :
    a. Web akan menampilkan path dari start dan end point, serta akan menampilkan waktu yang diperlukan untuk menemukan path tersebut.

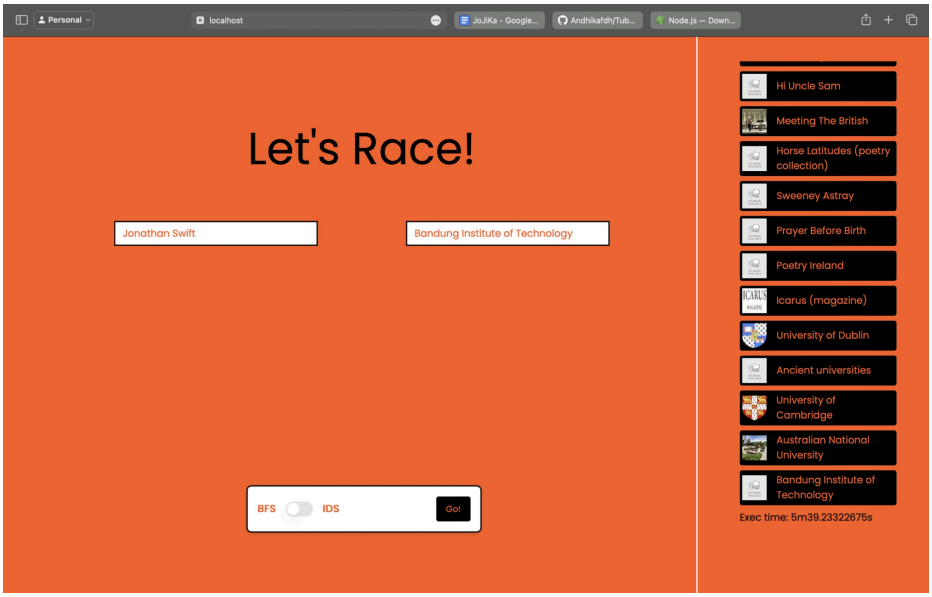
Dalam tugas besar ini, terdapat arsitektur aplikasi yang digunakan :

1. Front End : Front end menggunakan framework Vite, TailWind dan React, dan memakai bahasa pemrograman TypeScript, HTML, dan CSS.
2. Back End : Back end menggunakan bahasa pemrograman go yang berisi logika untuk mencari path menggunakan algoritma IDS dan BFS.

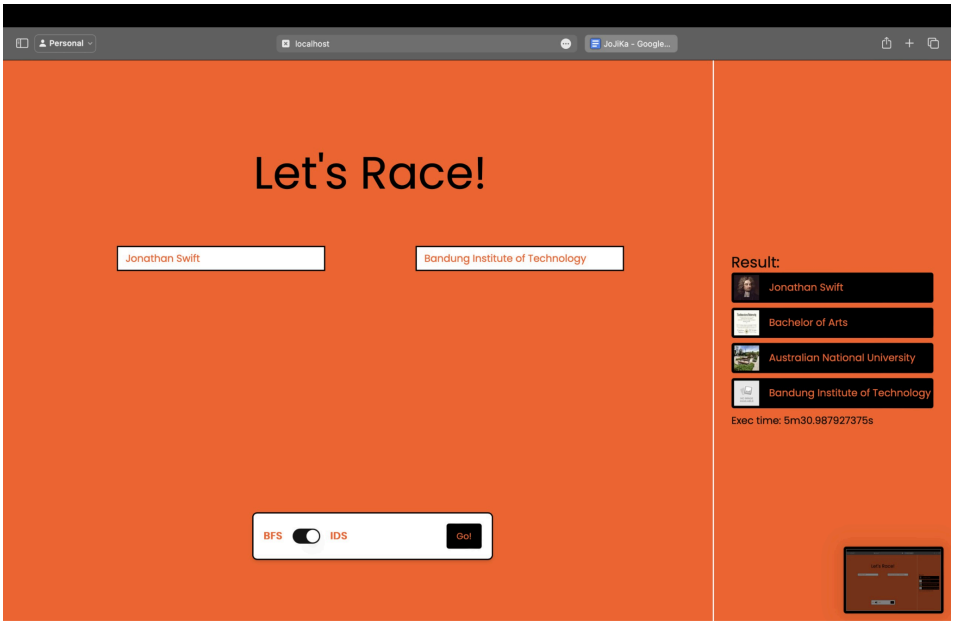
3.4 Contoh Ilustrasi Kasus

Dalam bagian ini, akan dipaparkan berbagai jenis ilustrasi kasus dengan menggunakan algoritma BFS dan IDS

Berikut merupakan contoh penerapan kasus BFS :



Berikut merupakan contoh penerapan kasus IDS :



BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi teknis program

4.1.1 Struktur Data

4.1.1.1 BFS.go

Link : Struktur data tersebut merepresentasikan sebuah link. Sebuah link memiliki url dengan tipe data string
Berikut Implementasi dari Link dalam bahasa GO :

```
type Link struct {
    URL string
}
```

4.1.1.2 IDS.go

Node : Struktur data tersebut merepresentasikan sebuah node dalam pohon (tree). Setiap node memiliki ID unik, referensi ke parent node, dan daftar referensi ke child node.
Berikut Implementasi dari Node dalam bahasa GO :

```
// Struktur Node untuk menyimpan informasi halaman Wikipedia
type Node struct {
    ID       string  // ID unik untuk setiap node yang biasanya adalah URL halaman
    Parent   *Node   // Referensi ke parent node
    Children []*Node // Slice untuk menyimpan referensi ke child node
}
```

4.2.2 Fungsi & Prosedur

4.2.2.1 IDS.go

GetLinks : Fungsi GetLinks mengambil link dari halaman Wikipedia berdasarkan input. Jika link sudah di-cache, langsung diambil dari cache. Jika tidak, link di-scan sesuai kriteria, lalu disimpan ke cache sebelum dikembalikan.
Berikut Implementasi dari GetLinks dalam bahasa GO :

```go
// Fungsi GetLinks untuk mengambil link dari halaman Wikipedia menggunakan Colly
func GetLinks(input string, parent *Node) []*Node {
    if nodes, ok := linkCache[input]; ok { // Cek jika link sudah ada di cache
        return nodes
    }

    var nodes []*Node
    collector := colly.NewCollector() // Membuat instance collector baru dari Colly

    // Mengatur filter untuk link yang ditemukan di halaman
    collector.OnHTML("a[href]", func(e *colly.HTMLElement) {
        href := e.Attr("href")
        // Validasi link yang sesuai dengan kriteria tertentu
        if strings.HasPrefix(href, "/wiki/") && !strings.Contains(href, ":") &&
            !strings.Contains(href, "Main_Page") && !strings.Contains(href, "UTF-8") {
            nodes = append(nodes, &Node{ID: href, Parent: parent})
        }
    })

    // Memulai scraping
    err := collector.Visit(fmt.Sprintf("https://en.wikipedia.org/wiki/%s", input))
    if err != nil {
        log.Fatal("Gagal mengambil halaman: ", err)
    }

    linkCache[input] = nodes // Menyimpan hasil ke cache
    return nodes
}
```

IDS : Fungsi IDS (Iterative Deepening Search) adalah implementasi dari strategi pencarian pada pohon yang disebut Depth-Limited Search (DLS). Fungsi ini mencoba mencari node dengan nilai tertentu (goal) mulai dari root dengan memperdalam pencarian satu per satu hingga ditemukan node yang dicari atau seluruh pohon telah dijelajahi.
Berikut Implementasi dari IDS dalam bahasa GO :

```go
// (Iterative Deepening Search - IDS)
func IDS(root *Node, goal string) *Node {
    depth := 0
    for {
        found := DLS(root, goal, depth) // Mencoba mencari dengan kedalaman tertentu
        if found != nil {
            return found // Mengembalikan nilai ditemukan
        }
        depth++ // Meningkatkan kedalaman dan mencoba lagi
    }
}
```

DLS : Fungsi DLS (Depth-Limited Search) adalah implementasi dari pencarian pada pohon dengan batasan kedalaman. Fungsi ini mencari node dengan nilai tertentu (goal) dalam pohon dimulai dari node, namun hanya mencari hingga kedalaman tertentu (depth).
Berikut Implementasi dari DLS dalam bahasa GO :

```go
// (Depth-Limited Search - DLS)
func DLS(node *Node, goal string, depth int) *Node {
    if node.ID == goal {
        return node // Jika ID node sama dengan tujuan, kembalikan node ini
    }
    if depth > 0 {
        node.Children = GetLinks(strings.TrimPrefix(node.ID, "/wiki/"), node) // Mengambil link anak jika belum mencapai batas kedalaman
        for _, child := range node.Children {
            found := DLS(child, goal, depth-1) // Rekursif mencari pada anak dengan kedalaman dikurangi satu
            if found != nil {
                return found
            }
        }
    }
    return nil // Kembalikan nil jika tidak ditemukan
}
```

4.2.2.2 BFS.go

fetchLinks : Fungsi fetchLinks digunakan untuk mengambil dan menyaring link dari halaman web Wikipedia berdasarkan URL halaman yang diberikan.
Berikut Implementasi dari fetchLinks dalam bahasa GO :

```go
func fetchLinks(pageURL string) ([]Link, error) {
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
    )

    var links []Link

    c.OnHTML("a[href]", func(e *colly.HTMLElement) {
        href := e.Attr("href")
        // Filter hanya link yang menuju halaman artikel biasa, tidak termasuk halaman khusus.
        if strings.HasPrefix(href, "/wiki/") && !strings.Contains(href, ":") {
            link := Link{
                URL: "https://en.wikipedia.org" + href,
            }
            links = append(links, link)
        }
    })

    err := c.Visit(pageURL)
    if err != nil {
        return nil, err
    }

    return links, nil
}
```

BFS : Fungsi BFS (Breadth-First Search) melakukan pencarian jalur dari URL awal (startURL) ke URL tujuan (endURL) menggunakan algoritma Breadth-First Search (pencarian lebar).
Berikut Implementasi dari BFS dalam bahasa GO :

```go
func BFS(startURL, endURL string) []Link {
    queue := []Link{{URL: startURL}}
    visited := make(map[string]bool)
    path := make(map[string][]Link)
    path[startURL] = []Link{{URL: startURL}} // Initialize the path for the start URL

    for len(queue) > 0 {
        currentLink := queue[0]
        queue = queue[1:]

        if currentLink.URL == endURL {
            return path[currentLink.URL]
        }

        if visited[currentLink.URL] {
            continue
        }

        visited[currentLink.URL] = true

        links, err := fetchLinks(currentLink.URL)
        if err != nil {
            fmt.Println("Error fetching links:", err)
            continue
        }

        for _, link := range links {
            if !visited[link.URL] {
                newPath := append([]Link(nil), path[currentLink.URL]...) // Copy current path
                newPath = append(newPath, link)                          // Append new link to the path
                path[link.URL] = newPath                                 // Update path for this link
                queue = append(queue, link)

                if link.URL == endURL {
                    return path[link.URL]
                }
            }
        }
    }

    return nil
}
```

4.2 Penjelasan tata cara penggunaan program

Secara keseluruhan, program kami terbagi menjadi dua yaitu frontend dan juga backend. Terdapat langkah yang berbeda yang harus dilakukan untuk menjalankan program tersebut.

Dalam menggunakan backend pada program ini, kelompok kami menggunakan vscode sebagai programming tools. Untuk menjalankan program ini, user harus menginstall go lang pada vscode, dan menginstall gocolly (library eksternal) .Untuk menjalankan program, perlu untuk mengcompile program dengan perintah , Berikut merupakan contoh pengaplikasiannya. :

```
cd src/ && go run .
```

Setelah di run, maka program backend akan berjalan dan menunggu perintah yang didapat dari pengguna yang didapatkan dari frontend.

Dalam menjalankan frontend pada program ini, user harus sudah menjalankan backend dan menginstall NodeJS (npm). Untuk menjalankan program ini, user perlu melakukan perintah berikut :

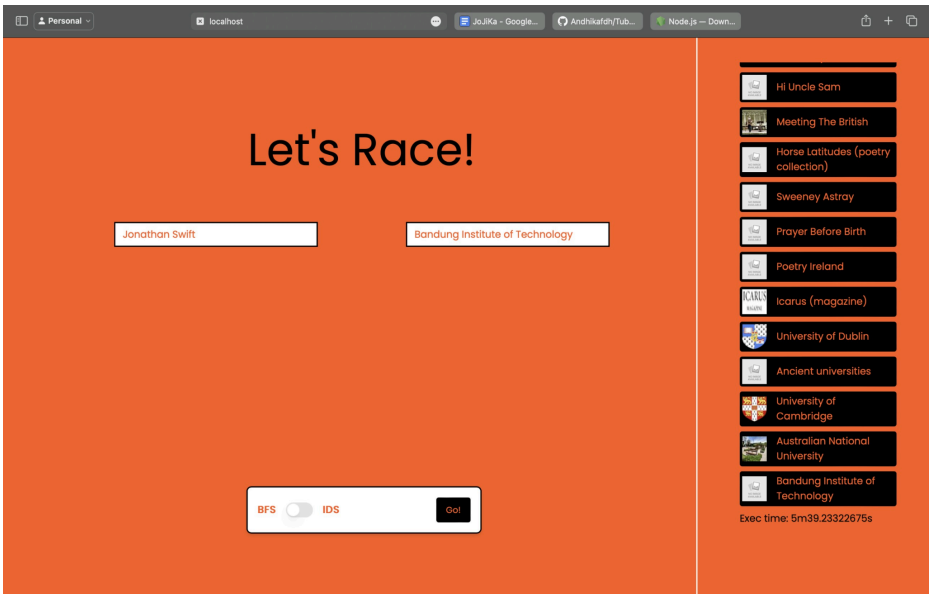
```
cd src/ && npm install && npm run dev
```

Dengan dijalankan perintah berikut, frontend sudah dijalankan dan website untuk memasukkan start dan endpoint bisa digunakan.

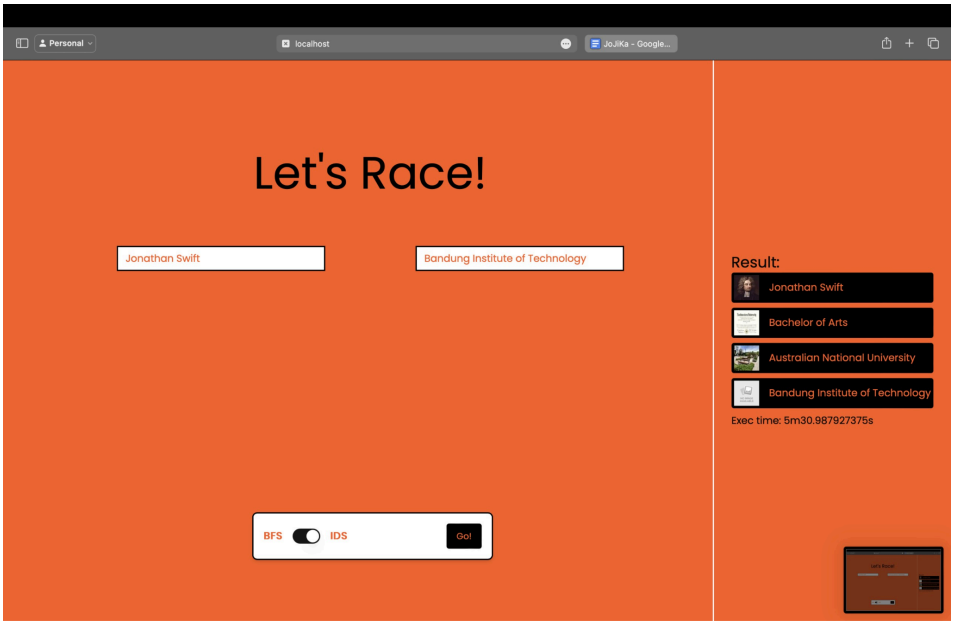
Untuk menggunakan website, user akan memasukan startpoint dan endpoint kemudia memilih algoritma IDS atau BFS untuk digunakan. Setelah menekan tombol “Go!”, program akan mencari path dan kemudian memunculkan hasilnya

4.3 Hasil pengujian

Berikut merupakan hasil pengujian menggunakan algoritma BFS :



Berikut merupakan hasil pengujian menggunakan algoritma IDS :



4.4 Analisis Hasil Pengujian

Dari pengujian yang sudah dilakukan, dapat dilihat bahwa hasil yang didapatkan sudah akurat dan memiliki waktu yang relatif singkat. Dalam hal ini, program IDS dan BFS sudah berhasil dalam mencari path dengan algoritma yang efektif dan sesuai dengan ketentuan tugas.

Untuk mencari path yang memiliki kedalaman diatas 3, program kami masih memiliki kendala yaitu waktu yang relatif lebih lambat. Perlu adanya peningkatan performa baik dari segi algoritma, maupun menangkap link yang perlu untuk dianalisis.

BAB 5
PENUTUP

5.1 Kesimpulan

Dalam tugas besar IF2211 Strategi Algoritma ini, kami berhasil membuat website yang dapat mengaplikasikan algoritma BFS dan IDS dalam mencari rute tercepat diantara dua tautan wikipedia. Website yang kami buat memiliki tampilan yang unik, interaktif dan mudah digunakan, serta berhasil mendapatkan hasil yang cukup optimal.
Dari pengerjaan tugas ini, kami mendapatkan beberapa kesimpulan, yaitu:

1. Algoritma BFS (Breadth First Search) efektif digunakan untuk menemukan rute optimal atau terpendek dalam persoalan yang dapat direpresentasikan sebagai graf tidak berbobot, seperti dalam Wikipedia Race. Namun, BFS memiliki kelemahan dalam penggunaan waktu dan memori karena harus menyimpan semua simpul yang akan dieksplorasi di dalam antrian (queue), sehingga perlu dilakukan optimasi agar pencarian tidak menghasilkan terlalu banyak simpul yang memakan waktu dan memori.
2. Algoritma IDS (Iterative Deepening Search) merupakan penggabungan antara konsep algoritma searching dengan graf dinamis. Dengan menggunakan metode Depth-Limited Search (DLS) secara berulang dengan peningkatan kedalaman, IDS dapat menemukan solusi dengan mempertimbangkan kedalaman pencarian. IDS cocok digunakan untuk persoalan yang memerlukan pencarian jalur dengan ruang seminimal mungkin, meskipun tidak menjamin solusi optimal karena fokusnya pada kedalaman tertentu.

BFS dapat memberikan solusi optimal namun dengan biaya waktu dan memori yang cenderung lebih besar, sementara IDS mempertimbangkan ruang seminimal mungkin namun tidak menjamin solusi optimal karena fokus pada kedalaman tertentu. Pilihan antara kedua algoritma ini tergantung pada kebutuhan spesifik dan kompromi yang diinginkan antara optimalitas solusi dan efisiensi komputasi.

5.2 Saran

Saran yang kami dapatkan dari pengerjaan pengembangan aplikasi ini adalah sebagai berikut,
1. Memerlukan pendalaman dan pemahaman terlebih dahulu terhadap penggunaan bahasa pemrograman GOLang dan tools-tools lain yang digunakan dalam pengembangan program ini.

2. Memerlukan *time management* yang baik agar tidak terburu - buru dalam mengerjakan tugasnya

3. Memerlukan perencanaan terlebih dahulu mengenai algoritma BFS dan IDS yang digunakan.

5.3 Refleksi

Tugas Besar 2 Strategi Algoritma ini sangat menarik dan informatif dalam pelaksanaannya tetapi alangkah baiknya pada saat pengerjaan tugas besar 2 ini direncanakan dengan matang agar bisa optimal dalam proses pengerjaannya.

LAMPIRAN

LINK REPOSITORY

Link repository GitHub Front End : https://github.com/Andhikafdh/Tubes2_FE_JoJiKa

Link repository GitHub Back End : https://github.com/JonathanSaragih/Tubes2_BE_JoJiKa

LINK VIDEO YOUTUBE

https://youtu.be/wuRfr4QZoLc

Daftar Pustaka

Munir, R., MT., & Ulfa Maulidevi, N., S. T, M. Sc. (n.d.). *Breadth/Depth First Search (BFS/DFS) (Bagian 2)* [Slide show]. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

Munir, R., & Ulfa Maulidevi, N., S. T, M. Sc. (n.d.). *Breadth/Depth First Search (BFS/DFS) (Bagian 1)* [Slide show]. https://informatika.stei.itb.ac.id. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf