

LAPORAN TUGAS BESAR 2 IF2211

STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN WIKIRACE



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S. T, M.Sc
Asisten pembimbing : Bintang Dwi Marthen

Disusun oleh:
Kelas 02 - Kelompok 17 - GO Blocks

Aurelius Justin Philo Fanjaya (13522020)
Marzuli Suhada M(13522070)
Fedrianz Dharma (13522090)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

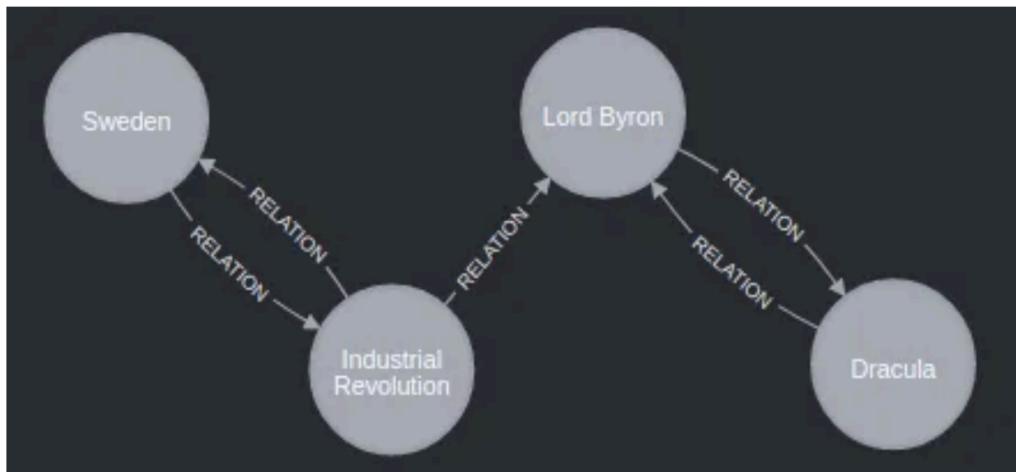
DAFTAR ISI.....	1
BAB I.....	3
BAB II.....	3
2.1 Penjelajahan Graf.....	3
2.2 Algoritma IDS.....	4
2.3 Algoritma BFS.....	5
2.4 Pembangunan Website.....	7
2.4.1 Front End.....	7
2.4.2 Back End.....	8
BAB III.....	10
3.1 Langkah-Langkah Pemecahan Masalah.....	10
3.1.1 Menentukan artikel yang terhubung dengan suatu artikel.....	10
3.1.2 Menelusuri tautan artikel hingga mendapatkan path/rute terpendek.....	10
3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma IDS dan BFS.....	11
3.2.1 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma BFS.....	11
3.2.2 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS.....	12
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	12
3.3.1 Fitur Fungsional.....	12
3.3.2 Arsitektur Aplikasi Web.....	13
3.4 Ilustrasi Kasus.....	14
3.4.1 BFS.....	14
3.4.2 IDS.....	15
BAB IV.....	17
4.1 Spesifikasi Teknis Program.....	17
4.1.1 Struktur Data.....	17
4.1.2 Fungsi dan Prosedur.....	17
4.2 Tata Cara Penggunaan Program.....	25
4.3 Hasil Pengujian.....	27
4.3.1 Test Case 1.....	27
4.3.2 Test Case 2.....	28
4.3.3 Test Case 3.....	29
4.3.4 Test Case 4.....	30
4.3.5 Test Case 5.....	31
4.3.5 Test Case 6.....	32
4.4 Analisis Hasil Pengujian.....	33
BAB V.....	35
5.1 Kesimpulan.....	35
5.2 Saran.....	35
5.3 Refleksi.....	36

5.4 Tanggapan.....	36
Repository.....	37
Youtube.....	37
DAFTAR PUSTAKA.....	38

BAB I

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)

BAB II

LANDASAN TEORI

2.1 Penjelajahan Graf

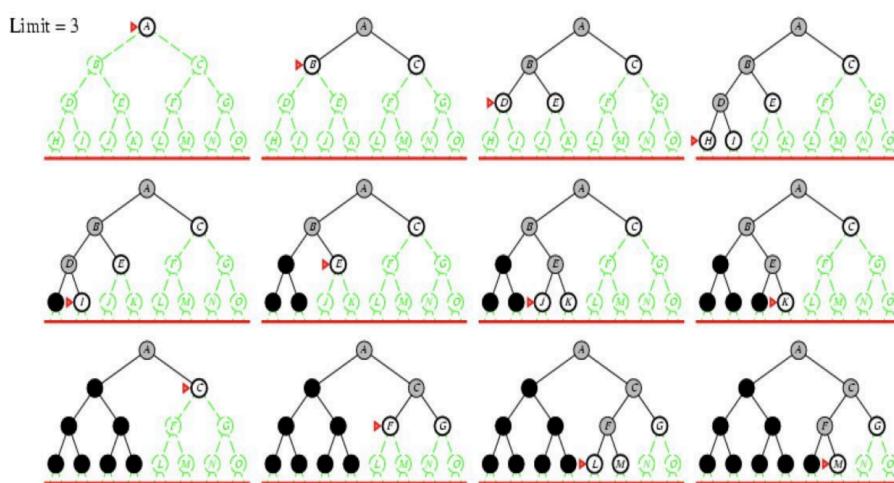
Penjelajahan graf (*graph traversal*) adalah proses mengunjungi (menjelajah) setiap simpul (*node*) atau tepi (*edge*) dalam graf untuk mencapai tujuan tertentu, seperti menemukan simpul tertentu, mencari jalur, menghitung jarak, atau mengidentifikasi komponen yang terhubung. Ada beberapa metode umum untuk menjelajahi graf, yaitu Breadth-First Search (BFS), Depth-First Search (DFS), Iterative-Deepening Search (IDS), dan lain-lain.

Dalam penjelajahan graf, kadang terjadi situasi di mana sebuah simpul dikunjungi lebih dari sekali, dan semakin besar jumlah simpul dalam graf, semakin tinggi pula kemungkinan terjadinya redundansi. Untuk mencegah pencarian yang tidak pernah berakhir ini, setiap simpul harus ditandai statusnya seperti apakah belum dikunjungi, sedang dikunjungi, atau sudah dikunjungi.

2.2 Algoritma IDS

Algoritma IDS adalah kombinasi antara *Depth-First Search* (DFS) dan *Breadth-First Search* (BFS). Ide dasar IDS adalah melakukan penelusuran secara mendalam (DFS) hingga kedalaman tertentu, kemudian meningkatkan kedalaman pencarian secara bertahap hingga menemukan tujuan atau memenuhi kriteria tertentu. IDS sering digunakan dalam konteks pencarian dalam ruang pencarian yang besar, seperti permainan atau masalah lain yang melibatkan sejumlah besar simpul. Skema algoritmanya adalah sebagai berikut.

1. **Tetapkan kedalaman maksimum:** Mulai dengan kedalaman tertentu, biasanya 0 atau 1.
2. **Lakukan DFS hingga kedalaman maksimum:** Jika mencapai batas kedalaman, kembali dan tingkatkan kedalaman maksimum.
3. **Ulangi hingga tujuan ditemukan atau kedalaman maksimum tercapai:** Terus tingkatkan kedalaman maksimum hingga memenuhi kriteria.



Gambar 2.2.1 Urutan Penjelajahan Algoritma IDS Dengan Batas Kedalaman 3

Berikut adalah *pseudocode* dari algoritma IDS.

```
function IDS(root, target):
    depth_limit = 0
    while True:
        # Lakukan Depth-Limited Search (DLS) dengan batas kedalaman
        yang meningkat
        result = DLS(root, target, depth_limit)
        if result == "found":
            return "target found"
        if result == "cutoff":
            depth_limit += 1 # Tingkatkan batas kedalaman
        else:
            return "target not found"

function DLS(node, target, depth_limit):
    if node == target:
        return "found"

    if depth_limit == 0:
        return "cutoff"

    cutoff_occurred = False
    for each child in node.children:
        result = DLS(child, target, depth_limit - 1)
        if result == "found":
            return "found"
        if result == "cutoff":
            cutoff_occurred = True

    if cutoff_occurred:
        return "cutoff"

    return "not found"
```

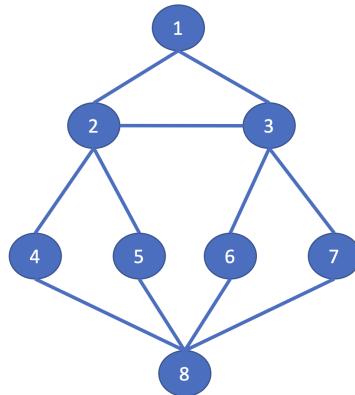
Pada algoritma diatas, IDS bekerja dengan menggabungkan DFS hingga kedalaman tertentu dan kemudian memperluas batas kedalaman jika target tidak ditemukan. IDS berguna dalam pencarian dengan ruang graf yang luas tanpa menggunakan terlalu banyak memori.

2.3 Algoritma BFS

Algoritma BFS adalah algoritma yang mengeksplorasi graf secara berurutan dari simpul awal, bergerak secara lebar dengan mengeksplorasi semua simpul pada tingkat tertentu sebelum melanjutkan ke tingkat berikutnya. BFS menggunakan struktur data antrian (queue) untuk menjaga urutan simpul yang akan dijelajahi. Misalkan ada sebuah

simpul v dari graf G . Akan dilakukan traversal graf G dengan algoritma BFS dimulai dari simpul v . Skema algoritmanya adalah sebagai berikut.

1. Kunjungi simpul v
2. Kunjungi seluruh simpul yang bertetangga dengan simpul v
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sudah dikunjungi sebelumnya
4. Lakukan langkah yang sama hingga seluruh simpul pada graf G telah dikunjungi.



Gambar 2.3.1 Urutan Penjelajahan Algoritma BFS

Berikut adalah *pseudocode* dari algoritma BFS.

```
procedure BFS(input v: simpul)
{ Melakukan traversal graf dengan algoritma pencarian BFS.
Masukan: v adalah simpul awal traversal
Luaran: Seluruh simpul graf dijelajahi dengan algoritma pencarian BFS }

Deklarasi

w : simpul
q : antrian

procedure BuatAntrian(input/output q: antrian)
{ Membuat antrian kosong }

procedure MasukAntrian(input/output q: antrian, input v: simpul)
{ Memasukkan simpul v ke antrian q pada posisi belakang }

procedure HapusAntrian(input/output q: antrian, output v: simpul)
{ Menghapus v dari kepala antrian q }

function AntrianKosong(input q : antrian) → boolean
{ Mengecek apakah antrian q kosong }
```

Algoritma

```
BuatAntrian(q)

write(v)
dikunjungi[v] <- true

MasukAntrian(q,v)
{ Kunjungi semua simpul graf selama antrian belum kosong }

while not AntrianKosong(q) do
    HapusAntrian(q,v)
    for tiap simpul w yang bertetangga dengan simpul v do
        if not dikunjungi[w] then
            write(w)
            MasukAntrian(q,w)
            dikunjungi[w] <- true
        endif
    endfor
endwhile
{ AntrianKosong(q) }
```

Algoritma BFS ini mengunjungi simpul dalam urutan lebar, sehingga semua simpul pada tingkat yang sama dijelajahi sebelum beralih ke tingkat berikutnya. BFS sangat berguna untuk menemukan jalur terpendek dalam graf yang tidak berbobot dan untuk memeriksa apakah graf terhubung. Ini sering digunakan dalam berbagai aplikasi, seperti pencarian dalam jaringan komputer, peta, dan jaringan sosial.

2.4 Pembangunan Website

2.4.1 Front End

Pengembangan Front End website yang saya gunakan adalah React Js. React JS adalah pustaka JavaScript yang dirancang untuk membangun antarmuka pengguna (UI). Dibuat oleh Facebook, React memungkinkan pengembang membuat komponen yang dapat digunakan kembali dan dinamis dalam aplikasi web. Keuntungan utama menggunakan React adalah kemampuan untuk membangun UI yang efisien, responsif, dan mudah dipelihara.

Beberapa konsep utama dalam React JS adalah:

1. Komponen: Unit dasar dalam React. Komponen adalah elemen mandiri yang memiliki logika dan tampilan sendiri. Komponen dapat berupa komponen fungsional atau komponen kelas.

2. JS: Ekstensi sintaksis untuk JavaScript yang memungkinkan penggunaan HTML dalam kode JavaScript. JS membuat kode React lebih intuitif.
3. State dan Props: "State" mengacu pada data yang dikelola oleh komponen dan dapat berubah seiring waktu, sedangkan "props" adalah data yang diteruskan ke komponen dari luar.
4. Virtual DOM: Teknik yang digunakan oleh React untuk mempercepat pembaruan UI dengan membandingkan representasi virtual dari DOM dengan DOM aktual.
5. Hooks: Fitur yang memungkinkan komponen fungsional untuk menggunakan state dan lifecycle methods. Beberapa hooks yang paling umum adalah:
 - useState : Untuk mengelola state dalam komponen fungsional.
 - useEffect : Untuk menangani efek samping, seperti pengambilan data atau pembaruan DOM.
 - useContext : Untuk mengakses konteks global dalam React tanpa prop drilling.

Dengan menggunakan React JS, pengembang dapat membuat aplikasi web yang interaktif dan dinamis. React JS banyak digunakan dalam pengembangan situs web modern dan sering digabungkan dengan teknologi lain seperti Redux, React Router, dan lainnya untuk membangun aplikasi web yang kuat.

2.4.2 Back End

Pengembangan Back End website yang kami buat menggunakan bahasa Go (Golang) dengan library Gin. Gin adalah Framework web development untuk bahasa Go dengan performa tinggi yang menyediakan berbagai fitur seperti:

- JSON Validation
Gin dapat melakukan *parsing* dan validasi terhadap JSON dari sebuah *request*, seperti untuk mengecek ada atau tidaknya sebuah nilai yang dibutuhkan.
- Routes Grouping
Gin dapat melakukan *Grouping* terhadap *routes* yang dapat di *nesting*.
- Error Management

Gin dapat mengumpulkan error yang terjadi ketika melakukan HTTP request.

- Rendering built-in

Gin menyediakan API untuk melakukan rendering terhadap JSON, XML dan HTML.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan utama dari tugas besar ini adalah untuk menentukan path/rute terpendek dari suatu artikel Wikipedia ke suatu artikel Wikipedia lain yang ditentukan. Dari situ, permasalahan yang harus dipecahkan adalah menentukan artikel apa saja yang terhubung (dapat diakses) dari suatu artikel dan menelusuri tautan artikel hingga mendapatkan path/rute terpendek.

3.1.1 Menentukan artikel yang terhubung dengan suatu artikel

Untuk menentukan apa saja artikel yang terhubung dengan suatu artikel, dapat dilakukan *scraping* pada artikel tersebut untuk mendapatkan *link* ke artikel lain. Kemudian *link-link* yang didapatkan tersebut akan dilakukan penelusuran dengan algoritma BFS atau DFS dan dilakukan *scraping* juga. Hal ini akan dilakukan hingga ditemukan *link* yang menuju ke artikel tujuan.

3.1.2 Menelusuri tautan artikel hingga mendapatkan path/rute terpendek

Untuk menelusuri suatu tautan artikel ke artikel tujuan dengan menjamin didapatkannya rute terpendek, dapat digunakan beberapa algoritma *searching*. Algoritma yang digunakan dalam tugas besar ini adalah algoritma BFS (Breadth First Search) dan algoritma IDS (Iterative Deepening Search). Kedua algoritma tersebut menjamin didapatkannya rute terpendek dari suatu artikel ke artikel lainnya jika adalah rute yang menghubungkan artikel tersebut.

3.1.3 Mengurangi lamanya waktu proses pencarian

Dalam melakukan pencarian artikel dengan depth yang tinggi, waktu pencarian akan meningkat secara eksponensial, sehingga akan sangat lama untuk dapat menemukan solusi untuk depth yang cukup tinggi. Untuk mengurangi lamanya waktu proses pencarian, algoritma pencarian menggunakan *goroutine* agar *scraping* dan *looping* dapat dijalankan secara *concurrent*. *Goroutine* yang

diciptakan akan dibatasi menggunakan *buffered channel* untuk mengurangi kemungkinan *timeout* akibat *scraping* yang terlalu banyak pada suatu waktu.

3.1.4 Mengatasi masalah *concurrent read* dan *write*

Untuk mengatasi masalah *concurrent read* dan *write* pada *map* akan digunakan *map* dari *library Sync*, yaitu *Sync.Map*. *Sync.Map* hanya akan memberikan akses pada satu *goroutine thread* pada satu waktu sehingga masalah *concurrent read* dan *write* akan teratas. Untuk mengatasi masalah *concurrent read* dan *write* pada *array* akan digunakan *Mutex* dari *library Sync*. *Mutex* akan menolak akses dari *goroutine thread* jika ada *goroutine thread* yang sedang mengakses *array* tersebut sehingga masalah *concurrent read* dan *write* akan teratas.

3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma IDS dan BFS

3.2.1 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma BFS

- **WikiPages**

WikiPages merupakan *struct* yang digunakan untuk merepresentasikan *link* sebuah artikel pada Wikipedia. WikiPages memiliki atribut URL yang menyimpan *link* artikel dan Title yang menyimpan Title dari *link* artikel.

- **Path / Array of WikiPages**

Path / Array of WikiPages merepresentasikan rute yang telah dikunjungi dari WikiPages awal hingga WikiPages tertentu.

- **Node / Simpul**

Setiap *node* atau simpul pada pohon graf BFS merepresentasikan Array of WikiPages yang ada pada level tertentu.

- **Queue**

Queue digunakan untuk menyimpan antrian rute simpul hidup yang akan diekspan. Pada kasus ini, Queue digunakan untuk menyimpan Path / Array of Wikipages. Queue pada setiap saat dapat merepresentasikan semua Path / Array of WikiPages yang ada pada suatu level tertentu.

- **Map Visited**

Map Visited digunakan untuk mencatat *link* yang pernah dikunjungi / dicek agar *link* tersebut tidak dimasukkan kembali pada Path / Array of WikiPages pada saat *scraping*.

3.2.2 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS

- **Node / Simpul**

Tiap node atau simpul pada pohon graf IDS direpresentasikan oleh tiap halaman Artikel wikipedia yang dibuat dalam *struct* WikiPage yang berisi Title dan URL dari artikel tersebut.

- **Edge / Sisi**

Setiap Sisi dari node pada graf / pohon IDS adalah hubungan antara Parent node dan child node di mana Artikel yang direpresentasikan oleh parent node memiliki link ke Artikel yang direpresentasikan oleh child nodenya.

- **Stack DLS**

Algoritma DLS dapat diimplementasikan dengan stack untuk menyimpan node apa yang akan di-expand berikutnya. Tapi, pada implementasi algoritma program ini, kami mengimplementasikan DLS dengan fungsi rekursif sehingga tidak menggunakan struktur data stack secara eksplisit.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

Arsitektur aplikasi web mencakup berbagai komponen yang saling terkait untuk membentuk keseluruhan sistem. Setiap file dalam aplikasi web memiliki peran dan fungsi tertentu, dan dapat diorganisir berdasarkan arsitektur front-end dan back-end, atau arsitektur berbasis komponen. Pada aplikasi web yang kami bangun, berikut adalah penjelasan mengenai fitur fungsional dan arsitektur aplikasi web yang dibangun.

3.3.1 Fitur Fungsional

Fitur-fitur fungsional dalam aplikasi web ini meliputi:

- **User Interface:** Pengguna dapat memasukkan artikel awal dan akhir untuk melakukan pencarian jalur (WikiRace). Aplikasi web menyediakan fitur interaktif seperti saran otomatis dan tombol-tombol untuk mengendalikan aliran aplikasi.

- **Pencarian Jalur (*path*):** Aplikasi web yang kami bangun memungkinkan pengguna untuk mencari jalur antara dua artikel Wikipedia menggunakan metode pencarian yang berbeda (BFS atau IDS). Hasil pencarian ditampilkan secara visual melalui komponen graf.
- **Pewarnaan dan Visualisasi Graf:** Aplikasi menggunakan komponen Graf untuk menampilkan jalur dalam bentuk graf visual, dengan fitur pewarnaan yang memudahkan pengguna untuk melihat hubungan antara artikel.
- **Informasi Tambahan:** Aplikasi memberikan informasi tambahan kepada pengguna seperti waktu eksekusi, jumlah artikel yang dikunjungi, dan jalur yang ditemukan.
- **Navigasi dan Informasi Umum:** Aplikasi menyediakan komponen navigasi (Navbar) untuk berpindah antar halaman, serta halaman "About Us" dan "How To Use" untuk memberikan informasi tambahan dan panduan penggunaan.

3.3.2 Arsitektur Aplikasi Web

Aplikasi web yang kami bangun menggunakan arsitektur berbasis React Js, dengan pemisahan yang jelas antara komponen-komponen yang berbeda. Berikut adalah elemen utama dari arsitektur aplikasi web ini.

A. Komponen Front-end

- a. **Komponen utama Home:** Ini adalah halaman utama yang mengelola *user interface*, termasuk input artikel, tombol, dan hasil pencarian. Komponen ini menggunakan state dan fungsi pengendali untuk mengelola logika aplikasi.
- b. **Komponen Graf:** Mengambil data jalur dan menampilkan graf dengan berbagai opsi visualisasi. Komponen ini mengelola node dan edge berdasarkan jalur yang diberikan.
- c. **Komponen WikipediaAutosuggest:** Menyediakan saran otomatis saat pengguna memasukkan teks dalam input, dan memungkinkan pengguna memilih saran untuk melanjutkan aliran aplikasi.

- d. **Komponen Navbar:** Menyediakan navigasi untuk berpindah antar halaman.
- e. **Komponen AboutUs dan HowToUse:** Memberikan informasi tambahan tentang aplikasi dan panduan penggunaan.

B. Penggunaan State dan Hooks

Aplikasi menggunakan hook React seperti useState dan useEffect untuk mengelola state dan efek samping dari suatu perlakuan. Ini membantu dalam menjaga interaksi pengguna dan logika aplikasi. Penggunaan state memungkinkan aplikasi untuk merespons perubahan secara dinamis, seperti pembaruan hasil pencarian, perubahan sumber gambar, dan lain-lain.

C. Penggunaan Fetch API

Komponen Home menggunakan fetch untuk mengirim permintaan HTTP ke backend. Ini memungkinkan aplikasi untuk melakukan operasi asinkron, seperti mendapatkan hasil pencarian dari server.

D. Pengaturan CSS dan Styling

Aplikasi mengimpor file CSS untuk mengatur styling dan tampilan aplikasi. Ini mencakup gaya untuk elemen seperti tombol, gambar, input, dan container.

E. Interaksi dan Pengendalian Aplikasi

Berbagai fungsi pengendali digunakan untuk mengatur interaksi pengguna, seperti menangani perubahan input, menukar artikel, dan mengirim permintaan. Ini membantu dalam menjaga aliran aplikasi dan memberikan pengalaman pengguna yang baik.

3.4 Ilustrasi Kasus

3.4.1 BFS

Ketika menggunakan algoritma BFS, misalkan mulai dari artikel Joko Widodo dan artikel tujuannya adalah Elon Musk. Kita mulai dengan mengunjungi artikel Joko Widodo dan melakukan *scraping* pada artikel Joko Widodo untuk mendapat semua *link*

artikel yang ada pada artikel Joko Widodo tersebut. Setiap *link* yang didapatkan akan dicatat pada sebuah *map* dan dimasukkan ke dalam rute saat ini, kemudian dimasukkan (enqueue) ke dalam sebuah *queue*. Lalu, kita akan mengunjungi semua *link* terakhir pada rute yang ada pada *queue* dan melakukan *scraping* secara *concurrent*. Hasil *scraping* tersebut akan dimasukkan pada rute saat ini dan dimasukkan pada sebuah *queue* baru jika *link* yang didapatkan belum ada pada *map*.

Setiap *queue* merepresentasikan semua rute dari *link* awal hingga semua *link*/simpul yang ada pada satu level. Proses akan dilakukan terus menerus hingga mendapatkan solusi rute dari artikel awal hingga tujuan. Jika dilakukan pencarian *single solution*, maka proses pencarian akan dihentikan dan langsung mengembalikan solusi yang didapatkan. Pada contoh kasus artikel awal Joko Widodo dan artikel tujuan Elon Musk, solusi yang didapatkan adalah Joko Widodo, List of international presidential trips made by Joko Widodo, Elon Musk. Namun, jika dilakukan pencarian dengan *multiple solution*, maka proses pencarian akan dilanjutkan pada level tersebut hingga semua rute pada *queue* tersebut selesai diperiksa untuk mendapatkan alternatif solusi lainnya. Pada contoh kasus akan ditemukan 8 solusi lainnya, salah satunya adalah Joko Widodo, The New York Times, Elon Musk.

3.4.2 IDS

Algoritma IDS kami implementasikan dengan menggunakan algoritma DLS dengan *maximum depth* yang bertambah pada tiap iterasi IDS, dimulai dari depth 1. Kami mengimplementasikan algoritma DLS menggunakan pendekatan rekursif sebagai berikut. Misalkan kasus pencarian dengan artikel awal Joko Widodo dan artikel tujuan Elon Musk. Pertama, pada fungsi DLS artikel Joko Widodo akan dicek apakah sudah ada di *Map* atau belum, jika sudah maka akan mengambil *link-link* yang menjadi value di *map* tersebut. Jika belum, maka artikel Joko Widodo akan dikunjungi dan dilakukan *scraping* untuk mendapat semua *link* artikel yang ada pada artikel tersebut dan dimasukkan ke *Map* dengan *key* nama artikel (dalam kasus ini *key* nya adalah Joko Widodo).

Setiap *link* yang didapatkan akan dicek apakah sudah sesuai dengan artikel tujuan, jika belum maka akan dengan memanggil fungsi DLS secara rekursif dengan start artikel tersebut dan tujuan tetap sama hingga menyentuh limit depth atau artikel

ditemukan. Jika pada tiap *path* DLS tidak ditemukan solusinya sampai *depth maximum*, maka akan dilakukan pencarian ulang DLS dengan *depth maximum* bertambah 1 pada iterasi IDS. Jika dilakukan pencarian *single solution*, maka proses pencarian akan dihentikan ketika pertama kali ditemukan solution. Jika dilakukan pencarian *multiple solution*, maka proses pencarian akan dilanjutkan iterasi IDS tersebut hingga selesai untuk mendapatkan solusi lainnya.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur Data

- **WikiPage**

Struktur data WikiPage berisi Title dan URL dari *links* pada Wikipedia.

```
type WikiPage struct {
    Title string
    URL   string
}
```

- **Queue**

Struktur data Queue dipakai pada algoritma BFS untuk menentukan *node* apa yang akan *di-expand* selanjutnya. Setiap node berisi rute dari *node* awal hingga *node* tersebut. WikiPages hasil *scraping* akan dimasukkan ke dalam tiap rute yang ada dan dimasukkan ke belakang *queue* (enqueue). Queue kami implementasikan dengan struktur data slice.

```
queue := make([][]WikiPage, 0)
```

Enqueue kami lakukan dengan meng-*append* node ke belakang slice.

```
queue = append(queue, []WikiPage{start})
```

- **Map**

Struktur data Map digunakan pada algoritma BFS untuk mencatat setiap *links* yang pernah dicek.

```
var visited sync.Map
```

Memasukkan *links* yang pernah dikunjungi ke dalam Map.

```
visited.Store(start.Title, true)
```

Mendapatkan *value* dari *key* pada Map.

```
_, ok := visited.Load(link.Title)
```

4.1.2 Fungsi dan Prosedur

- **BFSGo**

Fungsi BFSGo akan menjalankan pencarian dengan algoritma BFS secara *concurrent* dengan menggunakan bantuan fungsi BFSHelper. Fungsi ini akan melakukan *loop* untuk memproses semua elemen pada Queue secara *concurrent*. Fungsi ini juga akan menangkap hasil yang dikirimkan oleh fungsi BFSHelper untuk mengecek setiap Path yang dihasilkan dan memasukkan Queue yang dihasilkan ke dalam Queue utama. Pencarian akan terus dilakukan hingga sudah tidak ada elemen Queue, kemudian fungsi yang mengembalikan solusi yang ditemukan dan jumlah artikel yang dicek.

Fungsi BFSGo juga menerima argumen multi, jika true artinya fungsi akan mengembalikan *multi solution*, sedangkan jika false maka fungsi akan langsung return ketika menemukan solusi yang pertama.

```
var wg = sync.WaitGroup{}
var max_go int = 150
var guard = make(chan struct{}, max_go)
var solution = make([][]WikiPage, 0)
var m = sync.RWMutex{}

func BFSGo(start, end WikiPage, multi bool) ([][]WikiPage, int) {
    solution = make([][]WikiPage, 0)
    wg = sync.WaitGroup{}
    guard = make(chan struct{}, max_go)
    m = sync.RWMutex{}

    if start.Title == end.Title {
        return [][]WikiPage{{end}}, 1
    }

    queue := make([][]WikiPage, 0)
    var visited sync.Map
    queue = append(queue, []WikiPage{start})
    newPath := make(chan []WikiPage)
    visited.Store(start.Title, true)
    go func() {
        defer close(newPath)
        for len(queue) > 0 {
            tmpqueue := make([][]WikiPage, 0)
            for _, curpath := range queue {
                wg.Add(1)
                guard <- struct{}{}
                go BFSHelper(curpath, end, newPath, &visited,
                &tmpqueue)
            }
        }
    }()
    for i := 0; i < max_go; i++ {
        wg.Add(1)
        guard <- struct{}{}
        go BFSHelper(queue, end, newPath, &visited, &tmpqueue)
    }
    wg.Wait()
    if !multi {
        return solution[0], len(solution)
    }
    return solution, len(solution)
}
```

```
        wg.Wait()
        queue = tmpqueue
        time.Sleep(time.Second * 2)
        if len(solution) > 0 {
            break
        }
    }
    visited.Store(end, true)
} ()
for n := range newPath {
    path := n
    if path == nil {
        continue
    }
    if path[len(path)-1].Title == end.Title {
        fmt.Println(path)
        solution = append(solution, path)
        if !multi {
            return solution, syncMapLen(&visited)
        }
    }
}

return solution, syncMapLen(&visited)
}
```

- **BFSHelper**

Fungsi BFSHelper ini digunakan untuk melakukan *scraping* dengan menggunakan bantuan dari fungsi getWikiLinks dan memasukkan *links* yang didapatkan dari hasil *scraping* ke dalam Path yang kemudian dimasukkan ke dalam Queue jika *link* tersebut belum tercatat pada *map visited*. Fungsi akan menghasilkan Queue baru yang merepresentasikan level berikutnya dan mengirimkan setiap Path / rute yang dihasilkan menggunakan *channel* untuk dicek pada fungsi BFSGo.

```
func BFSHelper(path []WikiPage, end WikiPage, newPath chan<-[]WikiPage, visited *sync.Map, tmpqueue *[][][]WikiPage) {
    defer wg.Done()
    if len(path) == 0 {
        newPath <- nil
        return
    }
    lastPage := path[len(path)-1]
    links, err := getWikiLinks(lastPage)
```

```
if err != nil {
    newPath <- nil
    fmt.Println("error")
    return
}
for _, link := range links {
    _, ok := visited.Load(link.Title)
    if !ok {
        if link.Title != end.Title {
            visited.Store(link.Title, true)
        }
        newPathtmp := append([]WikiPage{}, path...)
        newPathtmp = append(newPathtmp, link)
        newPath <- newPathtmp
        m.Lock()
        *tmpqueue = append(*tmpqueue, newPathtmp)
        m.Unlock()
    }
}
<-guard
}
```

- **getWikiLinks**

Fungsi getWikiLinks digunakan untuk melakukan *scraping* pada suatu *link* untuk mendapatkan semua *link* yang ada yang berada pada domain “en.wikipedia.org”. Fungsi akan mengembalikan *array* yang berisi semua WikiPages yang berada pada suatu *links*.

```
func getWikiLinks(page WikiPage) ([]WikiPage, error) {
    visited2 := make(map[string]bool)
    c := colly.NewCollector(
        colly.AllowedDomains("en.wikipedia.org"),
        colly.Async(true),
    )
    // Add Random User Agents
    extensions.RandomUserAgent(c)

    var wikipages []WikiPage
    var wikipage WikiPage
    c.OnError(func(_ *colly.Response, err error) {
        fmt.Println("Something went wrong: ", err)
    })
    c.OnHTML("a[href]", func(e *colly.HTMLElement) {
        tmp := e.Attr("href")
        // fmt.Println(tmp)
```

```
// time.Sleep(time.Second)
    if strings.HasPrefix(tmp, "/wiki") &&
!strings.HasPrefix(tmp, "/wiki/File:") && !strings.HasPrefix(tmp,
"#") && !strings.HasPrefix(tmp, "https://") &&
!strings.HasPrefix(tmp, "/wiki/Special:") &&
!strings.HasPrefix(tmp, "/wiki/Category:") &&
!strings.HasPrefix(tmp, "/wiki/Wikipedia:") &&
!strings.HasPrefix(tmp, "/wiki/Portal:") &&
!strings.HasPrefix(tmp, "/wiki/Help:") {
    wikipage.URL = "https://en.wikipedia.org" + tmp
    // fmt.Println(wikipage.URL)
    wikipage.Title = strings.TrimPrefix(wikipage.URL,
"https://en.wikipedia.org/wiki/")
    // fmt.Println(wikipage.Title)
    if !visited2[wikipage.Title] {
        wikipages = append(wikipages, wikipage)
        visited2[wikipage.Title] = true
    }
}
})
err := c.Visit(page.URL)
if err != nil {
    return nil, err
}
c.Wait()
return wikipages, err
}
```

• SyncMapLen

Fungsi ini digunakan untuk menghitung jumlah elemen yang berada pada *sync.Map visited* untuk mendapatkan jumlah artikel yang di cek.

```
func syncMapLen(sm *sync.Map) int {
    var i int
    sm.Range(func(k, v interface{}) bool {
        i++
        return true
    })
    return i
}
```

• IDS

Fungsi ini akan memanggil algoritma DLS (DLSmulti/DLSSingle) secara iteratif hingga batas yang ditentukan. Jika pada suatu iterasi sudah

ditemukan solusi, maka iterasi akan diselesaikan pada iterasi tersebut dan solusi akan di-return.

```
func IDS(start, end WikiPage, maxDepth int, multi bool)
([][]WikiPage, int) {
    nodesChecked := 0
    var solution [][]WikiPage
    var cache sync.Map
    for depth := 1; depth <= maxDepth; depth++ {
        var nodes int
        if multi {
            solution, nodes = DLSmulti(start, end, depth, &cache)
        } else {
            solution, nodes = DLSSingle(start, end, depth, &cache)
        }
        nodesChecked += nodes
        if len(solution) > 0 {
            break
        }
    }
    return solution, nodesChecked
}
```

- **DLSmulti**

Fungsi DLSmulti melakukan pencarian DLS terhadap WikiPage secara concurrent dan rekursif. Jika depth sudah maksimal dan belum ditemukan solusi, maka akan return nil. Jika ditemukan solusi, maka akan menunggu hingga DLS pada setiap branch pada iterasi tersebut selesai, kemudian return tiap solusi yang mungkin.

```
func DLSmulti(start, end WikiPage, depth int, cache *sync.Map)
([][]WikiPage, int) {
    var max_go int = 15
    var guard = make(chan struct{}, max_go)
    var wg = sync.WaitGroup{}
    solution := make([][]WikiPage, 0)
    if depth == 0 && start.Title != end.Title {
        return nil, 1
    }
    if start.Title == end.Title {
        path := []WikiPage{start}
        solution = append(solution, path)
    }
    for i := 0; i < max_go; i++ {
        go func() {
            defer wg.Done()
            var result []WikiPage
            var nodes int
            if multi {
                result, nodes = DLSsingle(start, end, depth+1, &cache)
            } else {
                result, nodes = DLSsingle(start, end, depth+1, &cache)
            }
            if len(result) > 0 {
                solution = append(solution, result)
            }
            if nodes > max_go {
                close(guard)
            }
        }()
    }
    wg.Wait()
}
```

```
        return solution, 1
    }

    currentChecked := 1

    var links []WikiPage
    linkstmp, _ := cache.Load(start.Title)
    if linkstmp == nil {
        links, _ = getWikiLinks(start)
        cache.Store(start.Title, links)
    } else {
        links = linkstmp.([]WikiPage)
    }

    for _, link := range links {
        wg.Add(1)
        guard <- struct{}{}
        go func() {
            curSolution, nodesChecked := DLSmulti(link, end,
depth-1, cache)
            currentChecked += nodesChecked
            if curSolution != nil {
                solution = append(solution, curSolution...)
            }
            <-guard
            defer wg.Done()
        }()
    }
    wg.Wait()

    for i, path := range solution {
        solution[i] = append([]WikiPage{start}, path...)
    }
    return solution, currentChecked
}
```

- **DLSsingle**

Fungsi DLSmulti melakukan pencarian DLS terhadap WikiPage secara concurrent dan rekursif. Jika depth sudah maksimal dan belum ditemukan solusi, maka akan return nil. Jika ditemukan solusi, maka akan maka akan langsung return dan menghentikan pencarian pada level tersebut tanpa menunggu pencarian pada branch lain selesai.

```
func DLSSingle(start, end WikiPage, depth int, cache *sync.Map)
([][]WikiPage, int) {
    var max_go int = 15
    var guard = make(chan struct{}, max_go)
    var wg = sync.WaitGroup{}
    solution := make([][]WikiPage, 0)
    if depth == 0 && start.Title != end.Title {
        return nil, 1
    }
    if start.Title == end.Title {
        path := []WikiPage{start}
        solution = append(solution, path)
        return solution, 1
    }
    currentChecked := 1
    var links []WikiPage
    linkstmp, _ := cache.Load(start.Title)
    if linkstmp == nil {
        links, _ = getWikiLinks(start)
        cache.Store(start.Title, links)
    } else {
        links = linkstmp.([]WikiPage)
    }
    for _, link := range links {
        wg.Add(1)
        guard <- struct{}{}
        go func() {
            curSolution, nodesChecked := DLSSingle(link, end,
depth-1, cache)
            currentChecked += nodesChecked
            if curSolution != nil {
                solution = append(solution, curSolution...)
            }
            <-guard
            defer wg.Done()
        }()
        if len(solution) != 0 {
            solution[0] = append([]WikiPage{start}, solution[0]...)
            return solution, currentChecked
        }
    }
    wg.Wait()
    return solution, currentChecked
}
```

}

4.2 Tata Cara Penggunaan Program

1. Build Front end

Dari directory root projek, lakukan pindah ke directory frontend:

cd ./src/frontend

Kemudian, run build frontend dengan command:

yarn

kemudian

yarn build

2. Run Website:

Dari directory frontend, lakukan pindah ke directory backend:

cd ../backend

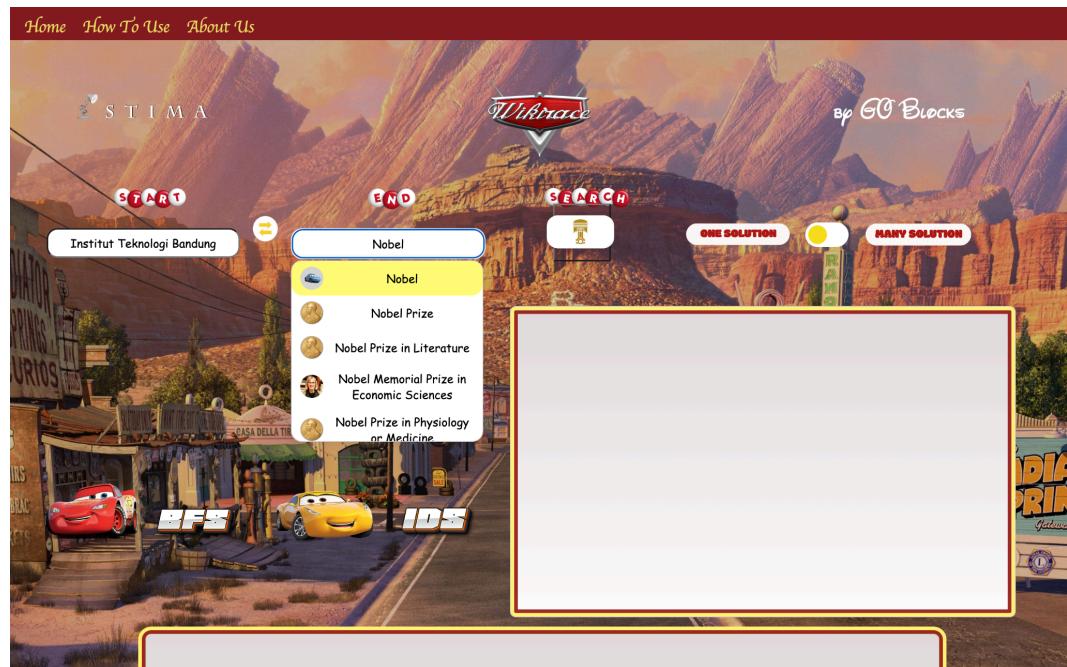
Kemudian, run jalankan website dengan command:

go run .

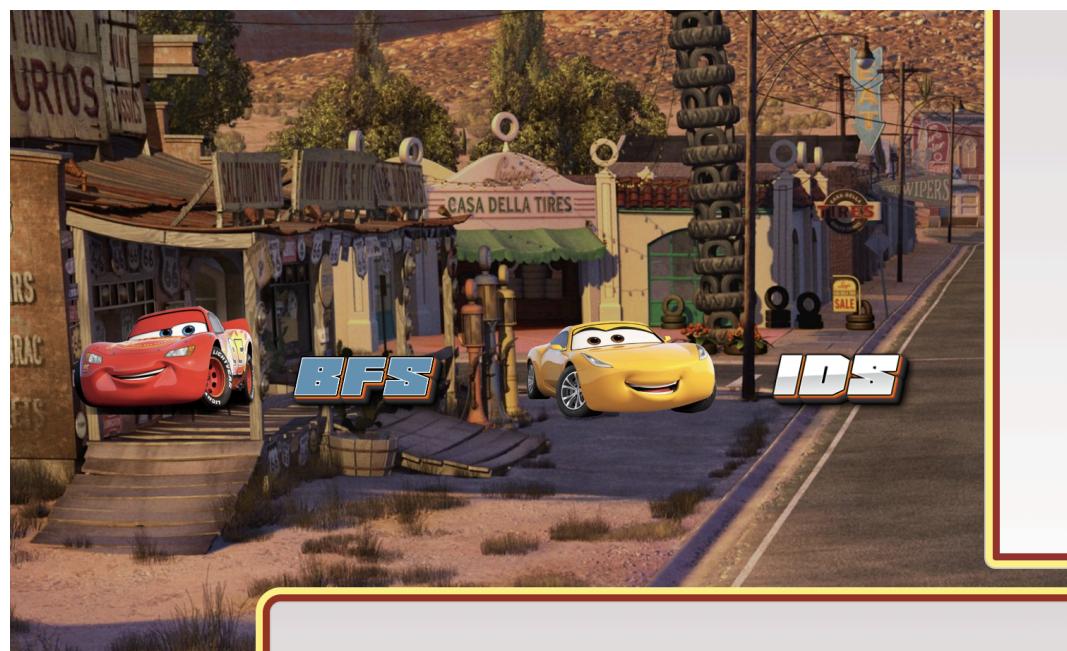
3. Kemudian, buka localhost:3000 pada browser

<http://localhost:3000/>

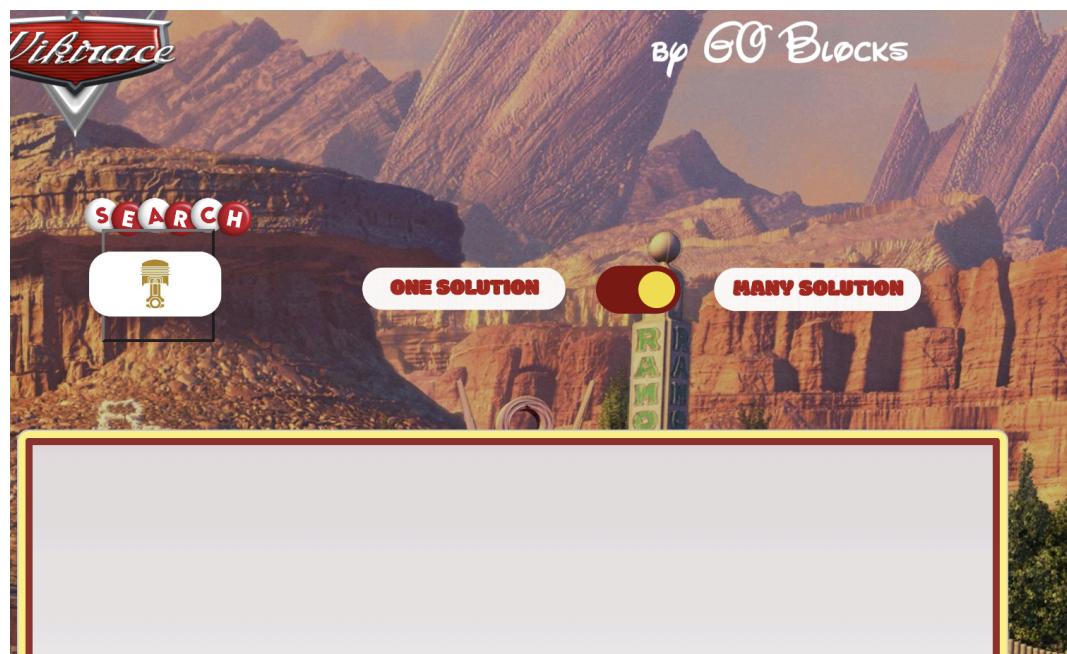
4. Pada website, pilih artikel awal dan tujuan dengan mengetik pada textbox yang tersedia dan memilih rekomendasi yang muncul



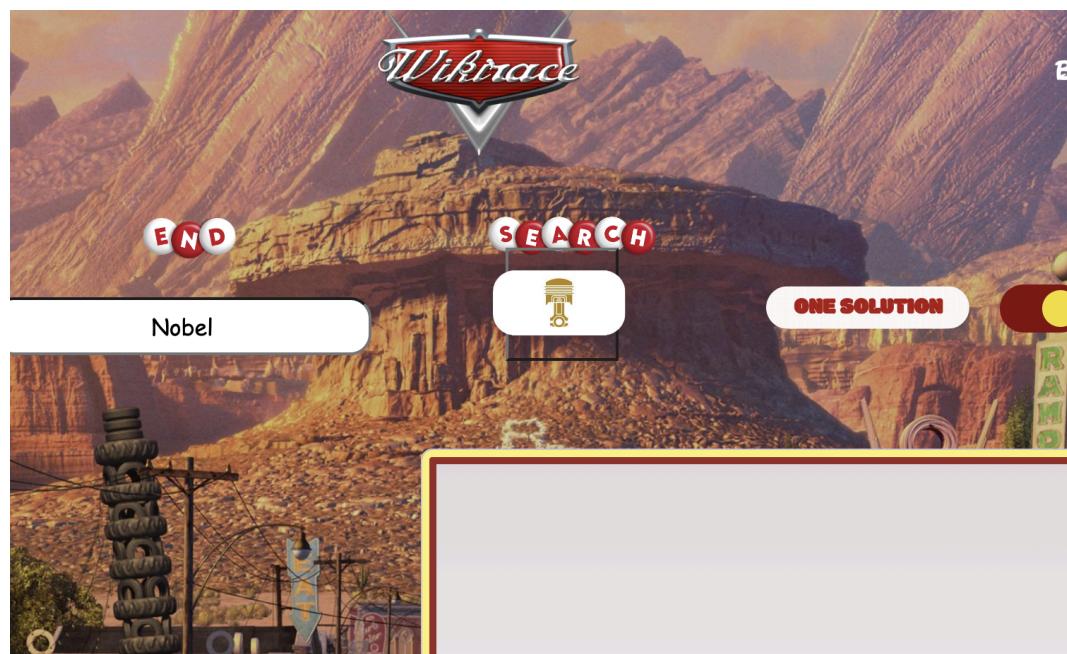
5. Pilih Algoritma yang akan digunakan (BFS / IDS)



6. Pilih Single Solution atau Multiple Solution (One / Many)



7. Klik Tombol Search

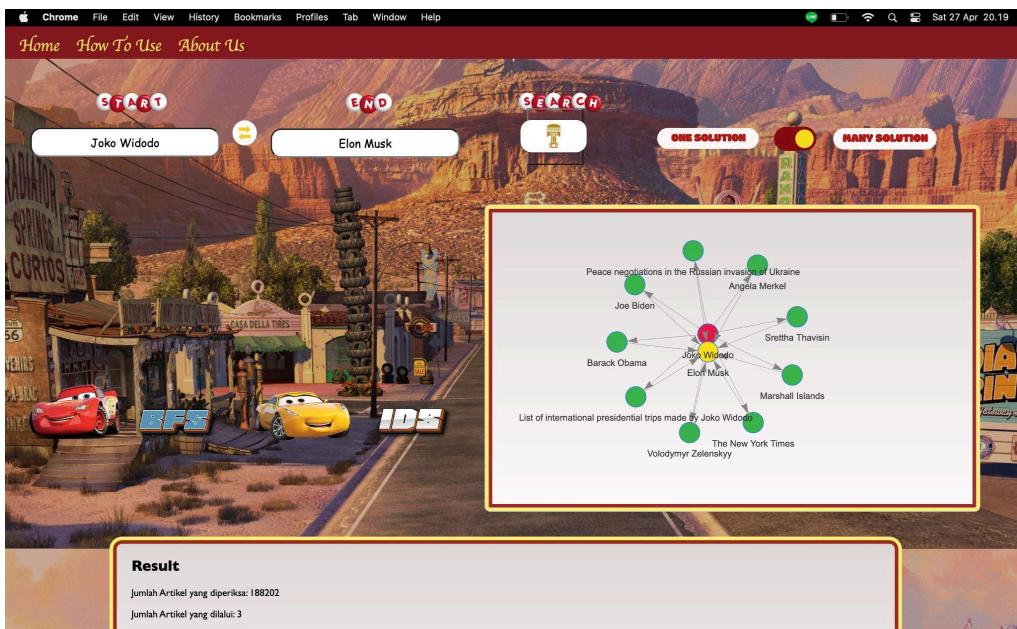
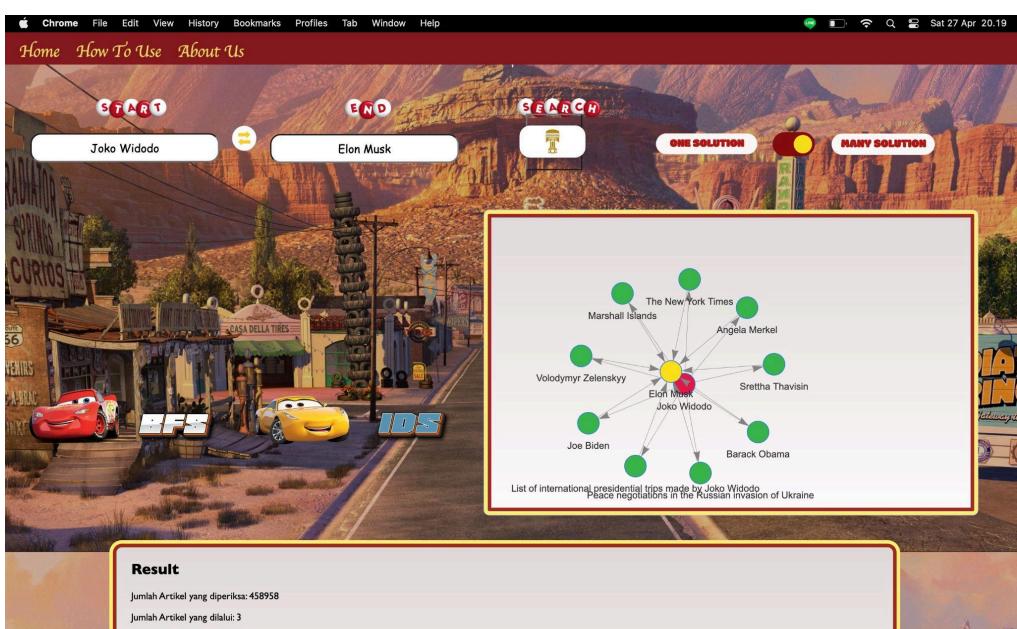


4.3 Hasil Pengujian

4.3.1 Test Case 1

Tabel 4.3.1 Hasil dari test case 1

Joko Widodo - Elon Musk

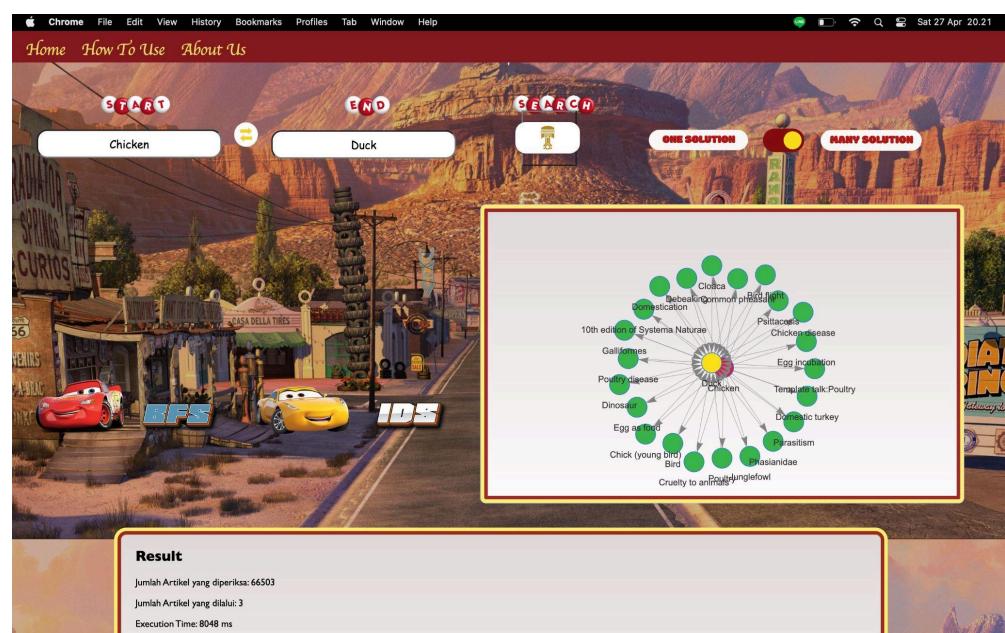
Algoritma	Hasil
BFS	
IDS	

4.3.2 Test Case 2

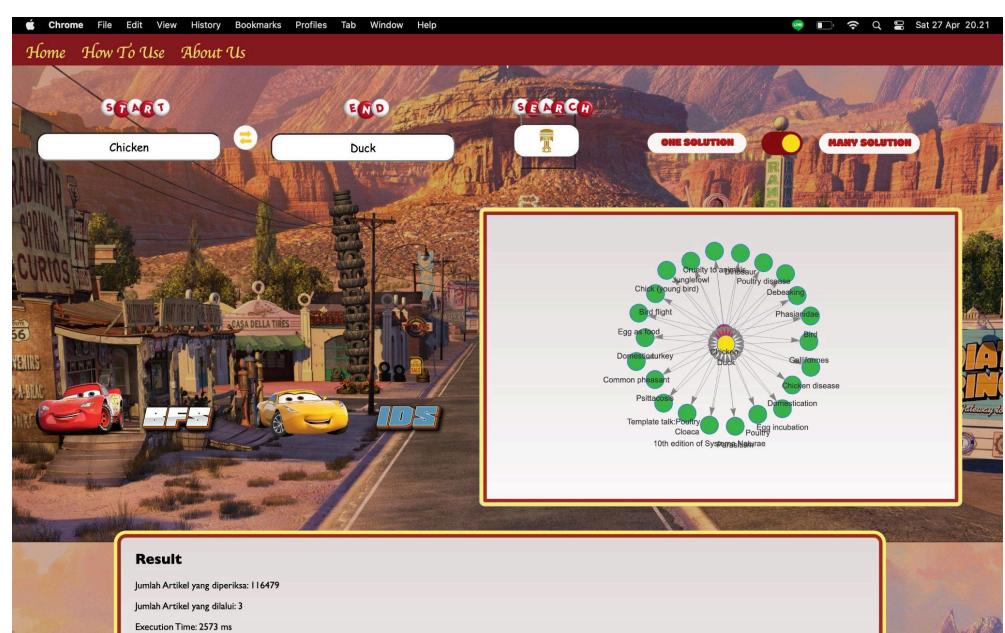
Tabel 4.3.2 Hasil dari test case 2

Chicken - Duck	
Algoritma	Hasil

BFS



IDS

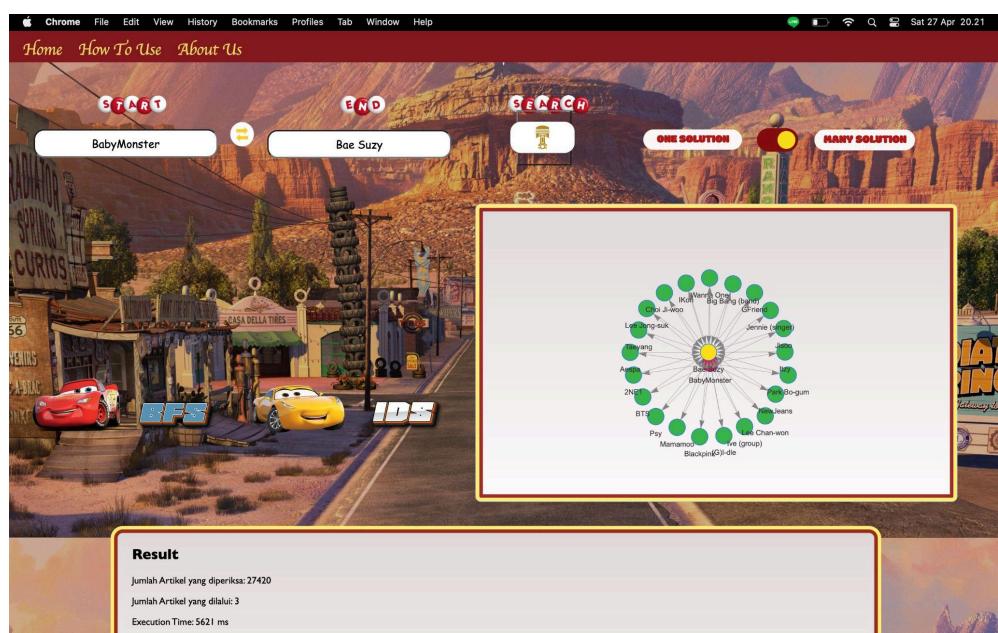


4.3.3 Test Case 3

Tabel 4.3.3 Hasil dari test case 3

BabyMonster - Bae Suzy	
Algoritma	Hasil

BFS



IDS



4.3.4 Test Case 4

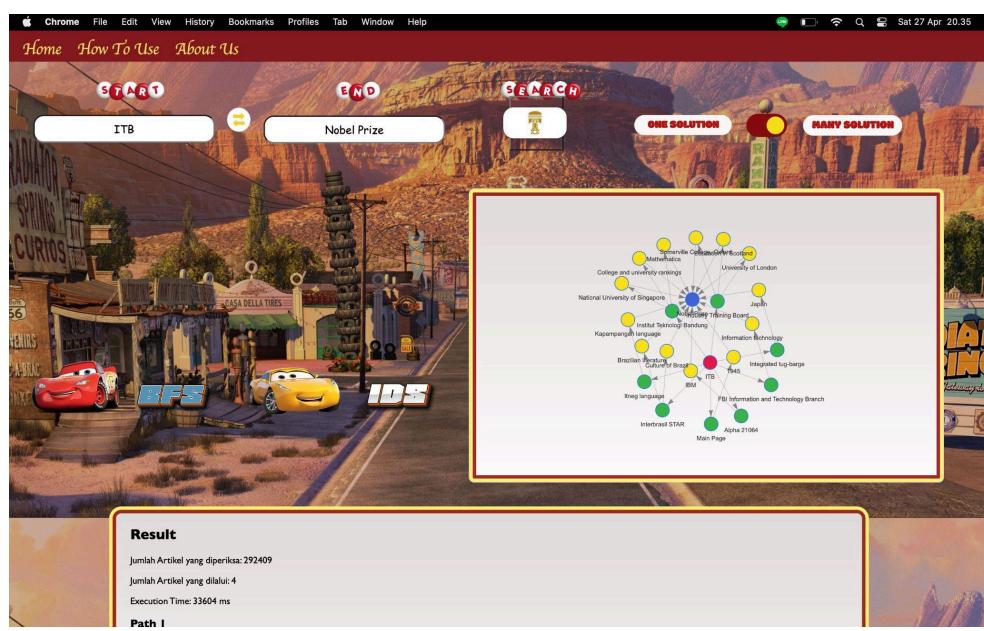
Tabel 4.3.4 Hasil dari test case 4

ITB - Nobel Prize

Algoritma

Hasil

BFS



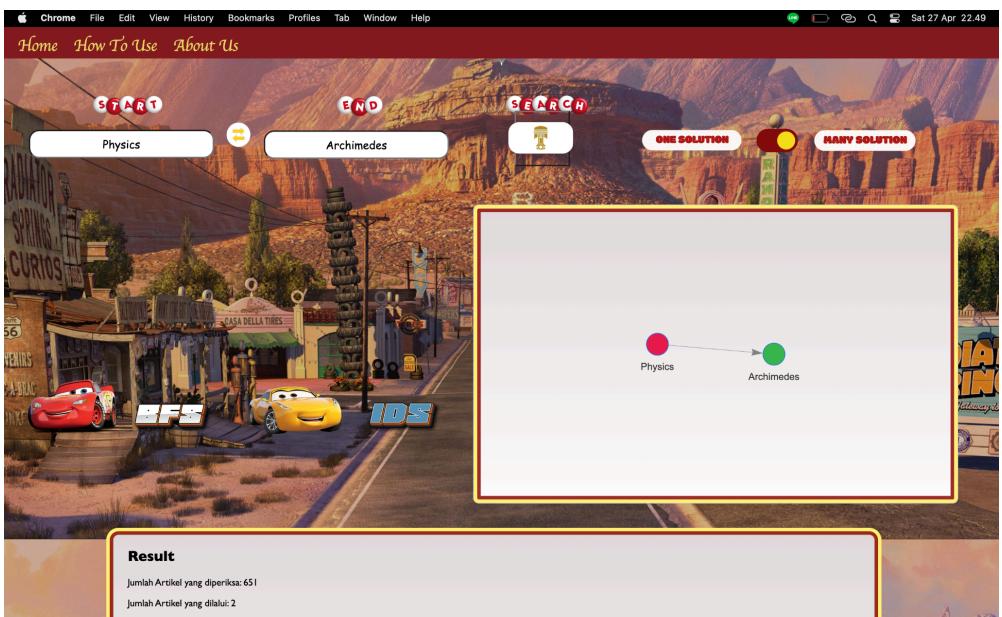
IDS



4.3.5 Test Case 5

Tabel 4.3.5 Hasil dari test case 5

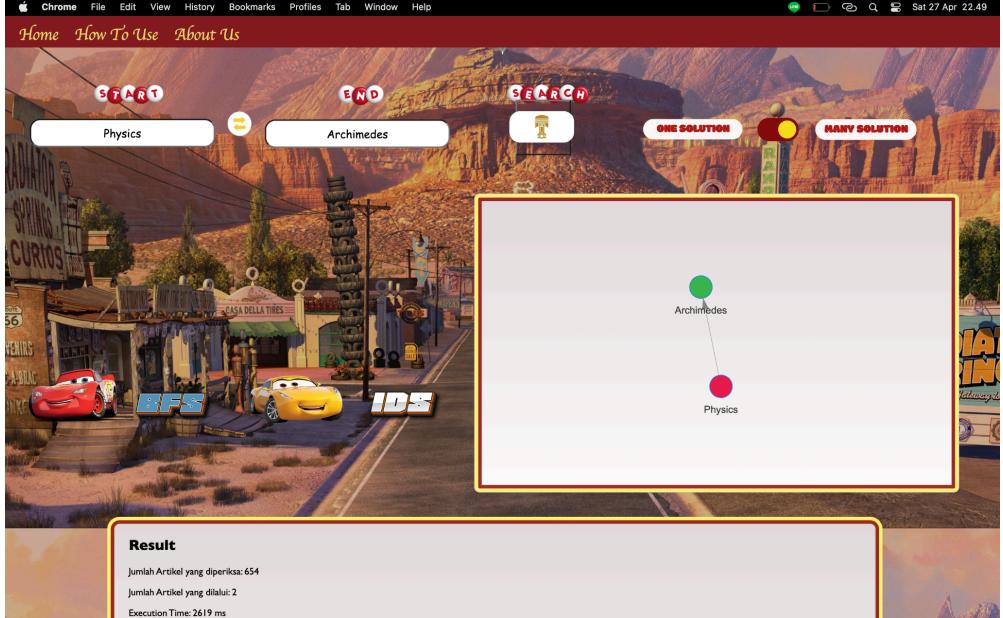
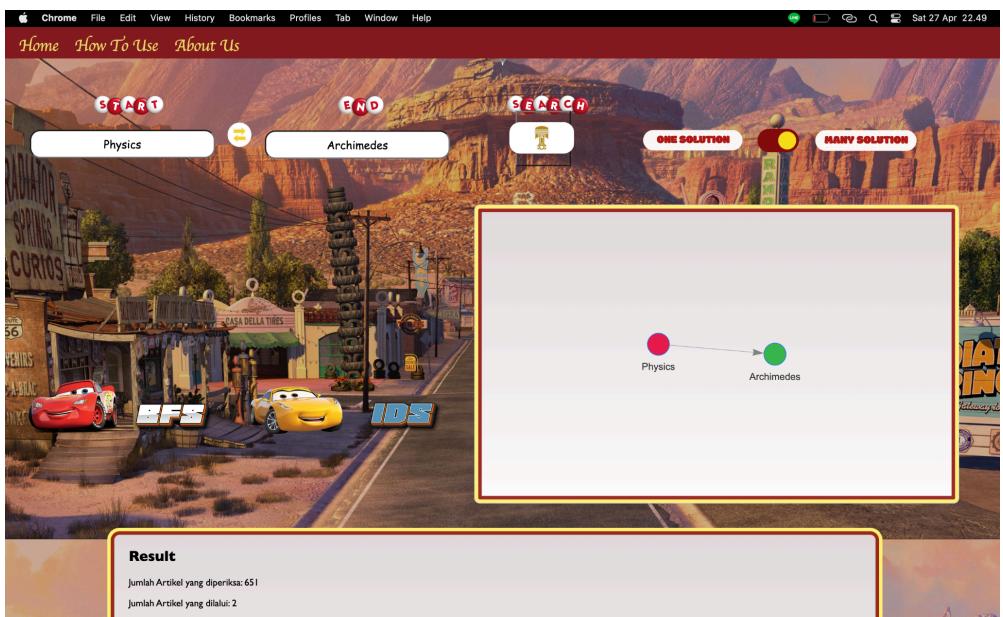
Algoritma	Physics - Archimedes
-----------	----------------------

BFS	
IDS	

4.3.5 Test Case 6

Tabel 4.3.5 Hasil dari test case 6

Algoritma	Physics - Archimedes
-----------	----------------------

BFS	
IDS	

4.4 Analisis Hasil Pengujian

Pencarian dengan menggunakan algoritma IDS lebih cepat dari algoritma BFS karena adanya penggunaan *caching* sehingga *link* yang pernah di-*scrape* tidak perlu di-*scrape* kembali. IDS juga lebih efisien dibanding BFS dari segi data yang harus disimpan. Pada BFS, setiap Node / Simpul akan menyimpan rute yang telah dilalui dari Node awal hingga Node saat itu. Sehingga Queue pada BFS

harus menyimpan *Array of Array of WikiPage* dari setiap levelnya, sedangkan pada IDS hanya disimpan Node dari pada iterasi tersebut saja.

Pada beberapa kasus BFS akan lebih cepat karena pada IDS, tiap iterasi DLS akan diulang dari awal. Meskipun hasil *scraping* disimpan pada *map*, tetapi akan tetap dilakukan DLS pada setiap child artikel tersebut dari awal.

BFS akan selalu mengecek semua kemungkinan rute yang ada pada setiap levelnya sehingga kurang efektif.

Berdasarkan hasil tes pada pencarian dengan data yang sedikit (depth 1 / 2), algoritma BFS menghasilkan waktu eksekusi yang lebih lambat karena kami melakukan *sleep* pada setiap level untuk mengurangi kemungkinan diblokir pada pencarian dengan data yang banyak. Sedangkan pada pencarian dengan data yang banyak (depth > 2) algoritma BFS dan IDS tidak menghasilkan perbedaan waktu yang cukup signifikan karena waktu *sleep* menjadi tidak terlalu berpengaruh terhadap waktu eksekusi yang besar.

BAB V

KESIMPULAN, SARAN

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma ini, kami berhasil membuat aplikasi web yang dapat melakukan pencarian rute dari sebuah artikel pada wikipedia ke artikel tujuan menggunakan algoritma *Breadth-First Search* (BFS) dan *Iterative-Deepening Search* (IDS). Pengembangan aplikasi web dibuat melalui Visual Studio dengan bahasa pemrograman Go dengan pengembangan GUI menggunakan React Js.

Dari pengerjaan tugas besar ini, kami mendapatkan beberapa kesimpulan, yaitu sebagai berikut.

- Penggunaan *goroutine* untuk mempercepat proses pencarian dengan cara melakukan beberapa proses secara *concurrent* dengan menggunakan *multiple thread*.
- Algoritma IDS dan BFS dapat memastikan solusi dengan *path* terpendek jika ditemukan solusi.
- Caching pada algoritma IDS dapat mempercepat waktu pencarian dengan *trade-off* meningkatnya kompleksitas ruang.
- Pencarian dengan algoritma IDS dan BFS secara concurrent tidak menghasilkan perbedaan waktu eksekusi yang signifikan, terutama pada jumlah data yang besar.

5.2 Saran

Saran dari kelompok kami terhadap tugas besar ini adalah tugas yang mengharuskan kami untuk melakukan scraping sehingga seringkali diblokir oleh wikipedia (too many request) ketika melakukan testing dan debugging yang cukup menyulitkan. Mungkin untuk tugas berikutnya dapat mempertimbangkan hal tersebut. Selain itu, terdapat spek atau penjelasan QnA yang cukup penting yang baru di-update hanya beberapa hari sebelum *deadline* sehingga tidak sempat untuk mengimplementasikannya.

5.3 Refleksi

Sebelum melakukan implementasi langsung, seharusnya kami sudah melakukan perancangan terstruktur sehingga tidak terjadi perbedaan-perbedaan pemikiran atau perancangan yang menghambat progress.

5.4 Tanggapan

Pada tugas besar ini, kami belajar banyak hal baru yang menjadi tantangan seperti bahasa go (golang), Framework back-end (Gin), Framework front-end (React), concurrency (goroutine), serta *web scraping* terhadap Wikipedia (gocolly). Hal-hal tersebut sangat menarik dan tentunya menambah pengalaman kami dalam bidang-bidang tersebut.

LAMPIRAN

Repository

Link Repository dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok 17 “GO Blocks” adalah sebagai berikut.

https://github.com/zultopia/Tubes2_GO-Blocks.git

Youtube

Link video Youtube dari Tugas Besar 2 IF2211 Strategi Algoritma kelompok 17 “GO Blocks” adalah sebagai berikut.

<https://www.youtube.com/watch?v=7LrM5IDkDvI>

DAFTAR PUSTAKA

Breadth/Depth First Search (BFS/DFS) Bagian 1. Terakhir diakses 27 April 2024 :
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

Breadth/Depth First Search (BFS/DFS) Bagian 2. Terakhir diakses 27 April 2024 :
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>

How to Build Wikiracer. Terakhir diakses 27 April 2024 :
<https://medium.com/@parulbaweja8/how-i-built-wikiracer-f493993fbdd>

Kakak Web Scraping. Terakhir diakses 27 April 2024 :
<https://github.com/PuerkitoBio/goquery>

Six Degrees of Wikipedia. Terakhir diakses 27 April 2024 :
<https://www.thewikipediagame.com/>

Wikirace Solver Dengan Single Solution. Terakhir diakses 27 April 2024 :
<https://wiki.spaceface.dev/>

Wikirace Solver Dengan Multiple Solutions. Terakhir diakses 27 April 2024 :
<https://www.sixdegreesofwikipedia.com/>