

**LAPORAN TUGAS BESAR II IF2211 STRATEGI ALGORITMA  
SEMESTER II 2023/2024  
PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN  
WIKIRACE**

Kelompok 15 / lebaran-ya-balapan

Ibrahim Ihsan Rasyid	13522018
Zaki Yudhistira Candra	13522031
Kayla Namira Mariadi	13522050



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

## DAFTAR ISI

<b>LAPORAN TUGAS BESAR II IF2211 STRATEGI ALGORITMA</b>	<b>1</b>
<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	<b>3</b>
<b>DESKRIPSI MASALAH</b>	<b>3</b>
<b>BAB II</b>	<b>4</b>
<b>LANDASAN TEORI</b>	<b>4</b>
A. Graph Traversal	4
B. Algoritma Breadth-First Search (BFS)	4
C. Algoritma Depth-First Search (DFS)	5
D. Algoritma Iterative Deepening Search (IDS)	5
E. Aplikasi Web Menggunakan Bahasa Go	5
<b>BAB III</b>	<b>7</b>
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>7</b>
A. Langkah-Langkah Pemecahan Masalah	7
B. Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS dan BFS	7
C. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.	8
D. Contoh Ilustrasi Kasus	9
<b>BAB IV</b>	<b>10</b>
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>10</b>
A. Spesifikasi Teknis Program	10
a. Struktur Data pada Program	10
b. Fungsi dan Prosedur yang Dibangun	10
B. Sebagai fungsi Tata Cara Penggunaan Program	13
1. Prosedur penggunaan program	13
2. Batasan penggunaan program	13
C. Hasil Pengujian	14
a. Pengujian BFS	14
b. Pengujian IDS	16
D. Analisis Hasil Pengujian	17
<b>BAB V</b>	<b>19</b>
<b>KESIMPULAN DAN SARAN</b>	<b>19</b>
A. Kesimpulan	19
B. Saran	19
C. Refleksi	19
<b>LAMPIRAN</b>	<b>20</b>
<b>DAFTAR PUSTAKA</b>	<b>24</b>

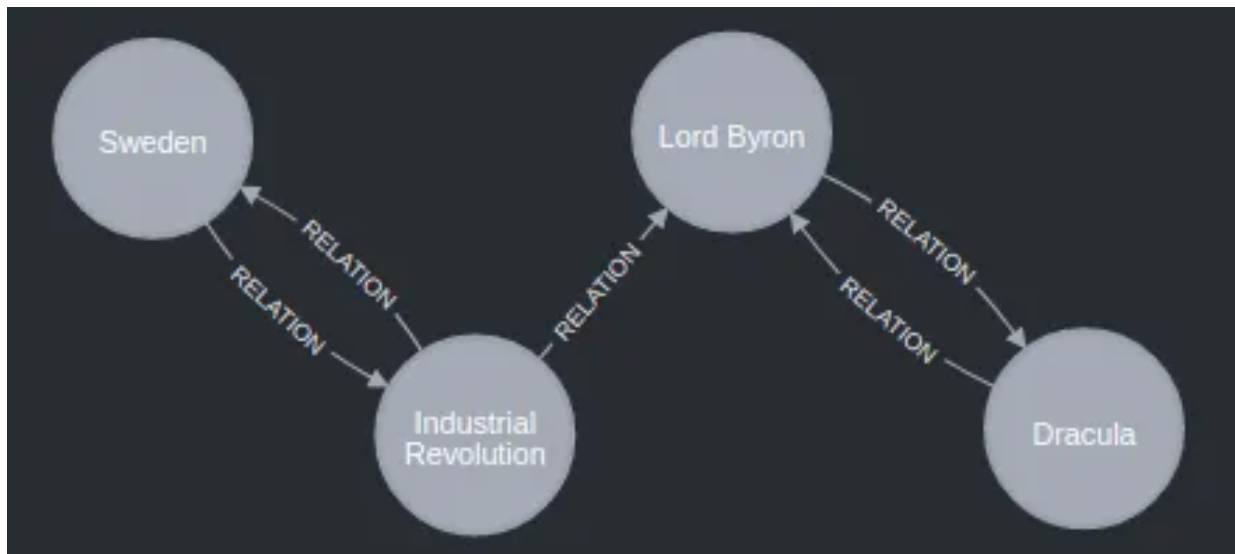
## BAB I

### 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.

Pada tugas besar ini, kami diminta untuk membuat implementasi dari WikiRace berbasis web menggunakan bahasa Go sebagai backend dengan memanfaatkan algoritma IDS dan BFS. Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms). Program akan mengeluarkan salah satu rute terpendek saja (cukup satu rute saja, tidak perlu seluruh rute kecuali mengerjakan bonus) yang ditemukan.

Berikut merupakan contoh visualisasi permasalahan permainan WikiRace dengan artikel awal adalah Sweden dan artikel tujuan adalah Dracula



(Sumber: [https://miro.medium.com/v2/resize:fit:1400/1\\*jxmEbVn2FFWybZslicJCWQ.png](https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZslicJCWQ.png))

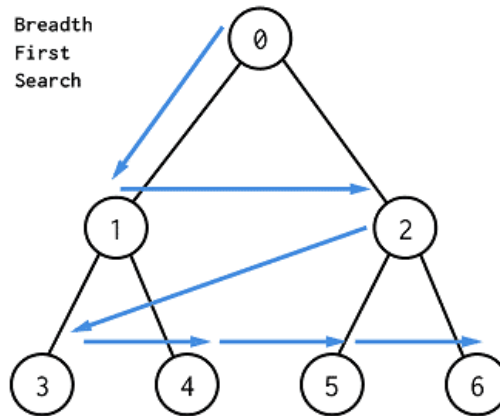
## BAB II

### LANDASAN TEORI

#### A. Graph Traversal

Graph Traversal adalah proses mengunjungi setiap simpul di dalam graf dengan cara yang sistematis. Tujuan dari graph traversal adalah untuk menemukan, mengakses, atau memanipulasi simpul pada graf. Terdapat beberapa algoritma dalam melakukan graph traversal, namun yang akan kami bahas hanya Breadth-First Search (BFS) dan Iterative Deepening Search (IDS). Kedua algoritma tersebut berbeda dari prioritas algoritma tersebut dalam memilih simpul yang akan dikunjungi setelahnya.

#### B. Algoritma Breadth-First Search (BFS)

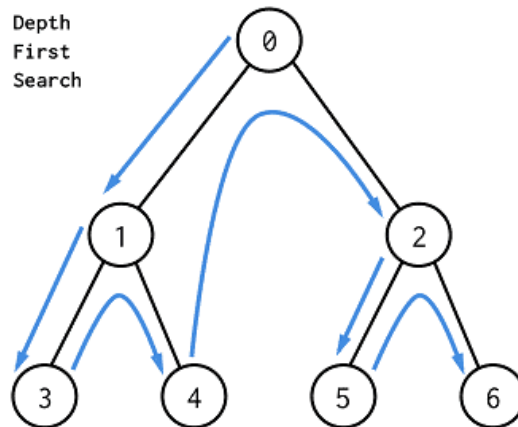


Gambar 2.1. Breadth First Search

Algoritma Breadth-First-Search (BFS) adalah salah satu algoritma pencarian pada graf yang umum digunakan. BFS memilih simpul berikutnya secara melebar, dimulai dengan memproses suatu simpul  $v$ , lalu mengunjungi semua simpul yang bertetangga dengan  $v$  dan menambahkan simpul tersebut ke queue, dilanjutkan dengan mengunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang telah dikunjungi sebelumnya pada queue, demikian seterusnya hingga mencapai simpul yang ingin dituju atau semua simpul sudah dikunjungi. Struktur data queue digunakan karena sifat First In First Out (FIFO) dari struktur data queue.

Kompleksitas waktu dari BFS adalah  $O(b^d)$ , dan kompleksitas ruangnya adalah  $O(b^d)$ , dengan  $b$  adalah banyaknya simpul tetangga paling banyak dari suatu simpul dan  $d$  adalah kedalaman dari graf.

### C. Algoritma Depth-First Search (DFS)



Gambar 2.3. Depth First Search

Algoritma Depth-First-Search adalah salah satu algoritma pencarian pada graf selain BFS yang umum digunakan. DFS memilih simpul berikutnya secara mendalam, dimulai dari simpul  $v$ , lalu mengunjungi simpul  $w$  yang bertetangga dengan simpul  $v$  dan menambahkan simpul tersebut ke stack, lalu proses diulangi mulai dari simpul  $w$ . Ketika mencapai sebuah simpul  $u$  sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul  $w$  yang belum dikunjungi. Struktur data stack digunakan karena sifat Last In First Out (LIFO) dari struktur data stack.

Kompleksitas waktu dari DFS adalah  $O(b^m)$ , sedangkan kompleksitas ruangnya adalah  $O(bm)$ , dengan  $b$  adalah banyaknya simpul tetangga paling banyak dari suatu simpul dan  $m$  adalah kedalaman dari graf.

### D. Algoritma Iterative Deepening Search (IDS)

Algoritma Iterative Deepening Search (IDS) adalah algoritma yang melakukan serangkaian DFS dengan kedalaman cutoff tertentu yang ditingkatkan secara iteratif hingga solusi ditemukan. IDS menggabungkan efisiensi ruang dari DFS dan kecepatan pencarian dari BFS.

Kompleksitas waktu dari DFS adalah  $O(b^d)$ , sedangkan kompleksitas ruangnya adalah  $O(bd)$ , dengan  $b$  adalah banyaknya simpul tetangga paling banyak dari suatu simpul dan  $d$  adalah kedalaman dari graf.

### E. Aplikasi Web Menggunakan Bahasa Go

Go merupakan bahasa pemrograman high-level yang dikembangkan oleh Google. Go merupakan bahasa yang statically typed, dan memiliki syntax yang mirip dengan

bahasa C, namun memiliki memory safety, garbage collection, structural typing, serta CSP-style concurrency.

Go diciptakan untuk meningkatkan produktivitas pemrograman di era 2007 ketika sudah banyak digunakan multicore, networked machine, dan codebase yang besar. Pengembang Go merasa tidak sesuai dengan bahasa-bahasa yang digunakan di Google, namun ingin mempertahankan beberapa karakteristik yang berguna dari bahasa tersebut: static typing dan efisiensi run-time milik C, readability dan usability milik Python, dan performa networking dan multiprocessing yang baik.

Pada tugas besar ini, digunakan beberapa *package* yaitu *fmt*, *sync*, *time*. Untuk kebutuhan api dan scraper juga digunakan *gocolly* dan *fiber*. Dalam implementasi program juga digunakan *goroutine* yang memungkinkan dijalankannya program secara konkuren dengan memanfaatkan multiple core CPU. Go juga memungkinkan untuk menjalankan banyak *goroutine* secara bersamaan, yang bisa saling berkomunikasi melalui saluran (*channels*).

#### **F. Pengembangan *Frontend* Website dengan React.js**

React adalah library JavaScript untuk mengembangkan antarmuka pengguna (*front-end*). React dibangun dari beberapa komponen yang dapat digunakan ulang dan kemampuan pembaruan tanpa perlu me-refresh halaman. React memastikan antarmuka pengguna responsif dan efisien. Untuk mengambil (*fetch*) data API, digunakan *axios* yang merupakan sebuah library untuk melakukan proses HTTP request pada website dengan React JS. *Axios* dapat memudahkan pengelolaan interaksi antara client dan server web. Pada Tugas Besar ini, digunakan pula *Tailwind CSS* yang dihubungkan dengan React sebagai *styling*.

### **BAB III**

#### **ANALISIS PEMECAHAN MASALAH**

##### **A. Langkah-Langkah Pemecahan Masalah**

Permasalahan yang akan diselesaikan dalam Tugas Besar 2 ini adalah pencarian rute terpendek dari satu artikel pada Wikipedia ke artikel lainnya. Pemecahan masalah dilakukan dengan memanfaatkan algoritma IDS atau BFS. Pemecahan masalah diawali dengan meminta masukan judul artikel awal dan judul artikel tujuan pada web. Untuk *autocomplete* mesin pencarian, digunakan axios untuk mengambil data yang berkaitan dengan masukan Url dari Wikipedia API. Masukan url awal dan url tujuan dipastikan merupakan url yang ada pada Wikipedia dengan domain <https://en.wikipedia.org>. Pencarian hanya dilakukan pada Wikipedia berbahasa Inggris karena menurut kami, artikel yang tersedia pada Wikipedia berbahasa Inggris lebih lengkap. Lalu, pengguna juga memilih algoritma yang ingin digunakan dalam pencarian tersebut, serta ADA fitur opsional yaitu memilih jumlah maksimal hasil yang ingin ditampilkan (khusus BFS). Pengguna kemudian memilih algoritma pencarian dan menekan tombol “Search” untuk melakukan pencarian.

Data dari API tersebut kemudian diintegrasikan dengan backend menggunakan fiber. Dari masukan tersebut, algoritma diimplementasikan. Pembangkitan graf dilakukan melalui scraping pada laman Wikipedia menggunakan gocolly serta goroutine. Ketika program menerima masukan judul dan url awal, program akan menjadikan url awal sebagai simpul awal, lalu laman-laman lain yang berkaitan sebagai simpul tetangganya, seterusnya selama pencarian berjalan. Pencarian berhenti ketika dalam pemeriksaan ditemukan simpul tujuan.

Hasil pencarian kemudian dikirim ke web melalui method Post pada api. Web kemudian akan menampilkan jumlah laman yang dikunjungi, jumlah laman yang dilalui hingga url tujuan (kedalaman/derajat), rute penjelajahan artikel, waktu eksekusi, dan graf berarah yang memvisualisasikan rute hasil.

##### **B. Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS dan BFS**

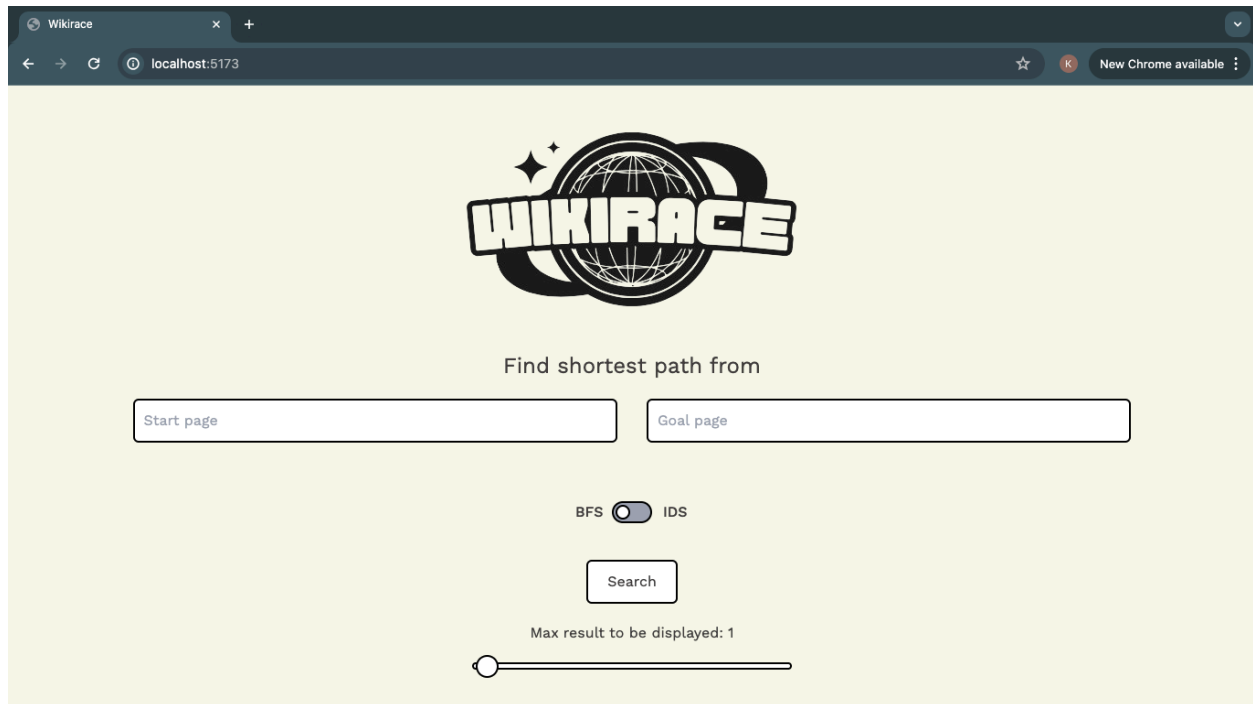
Pada pencarian menggunakan algoritma IDS, program akan memulai pencarian dari url awal, lalu dilanjutkan ke url lain yang berhubungan dengan url saat ini berdasarkan hasil scraping. Pencarian dilakukan dengan algoritma DLS dengan kedalaman cutoff awal 0. Apabila pada kedalaman tersebut tidak ditemukan url tujuan, maka kedalaman cutoff akan ditingkatkan dan operasi DLS akan diiterasi. Pada operasi DLS, pencarian dilakukan dengan mencari salah satu jalur hingga mencapai kedalaman cutoff, lalu beralih ke jalur lain. Proses terus dilakukan hingga url tujuan ditemukan.

Sedangkan pada pencarian menggunakan algoritma BFS, program akan melakukan pencarian dengan memeriksa setiap tetangga dari simpul awal, lalu beralih ke

salah satu simpul tetangga sesuai dengan aturan BFS. Proses terus dijalankan hingga url tujuan ditemukan.

### C. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.

Berikut adalah tampilan aplikasi web yang kami bangun



Gambar 3.1. Tampilan Web

Untuk bagian frontend, kami menggunakan React Js dengan *styling* menggunakan Tailwind CSS. Source code frontend terdapat pada direktori `src/frontend`. Pada direktori tersebut terdapat folder `src` yang berisikan komponen-komponen serta aset yang dipakai untuk merender web. Frontend dijalankan pada <http://localhost:5173/>.

Untuk bagian backend, kami menggunakan bahasa Go. Source code backend terdapat pada direktori `src/backend`. Pada direktori tersebut terdapat folder `models` berisi struktur data yang digunakan, folder `api` berisi program untuk integrasi dengan frontend, dan folder `algorithm` berisi program `ids`, `bfs`, dan `scraper`. Untuk kebutuhan scraping, digunakan *package* `gocolly` untuk mengekstrak link artikel wikipedia dan `waitgroup` go serta `goroutine`. Sedangkan untuk kebutuhan api, kami menggunakan `fiber`. Backend dijalankan pada <http://localhost:8080/>.

Terdapat beberapa fitur-fitur fungsional dalam aplikasi website Wikirace ini, yaitu

1. Masukan judul artikel wikipedia beserta fitur *suggestion*

Pada web yang kami buat, pengguna memasukkan serta memilih judul artikel awal pada input box dengan keterangan “Start page”, dan memasukkan artikel tujuan pada input box “Goal page”. Pengguna dapat memilih *suggestion*



berupa judul, gambar, dan deskripsi artikel yang diambil dari Wikipedia API sehingga judul artikel tidak perlu ditulis manual secara lengkap.

2. Pemilihan algoritma pencarian

Pengguna memilih jenis algoritma yang digunakan dengan mengklik toggle bar di antara BFS dan IDS.

3. Pilihan jumlah hasil maksimal yang ingin ditampilkan

Khusus algoritma BFS, terdapat slider dengan range 0-200 agar pengguna dapat memilih jumlah maksimal rute hasil yang ingin ditampilkan. Jika bernilai 0, web akan menampilkan semua hasil pencarian. Secara default, jumlah diset 1, dimana web hanya akan menampilkan 1 rute terpendek.

4. Visualisasi rute pencarian dengan graf berarah

Web juga memvisualisasikan graf berarah rute dari artikel awal ke artikel tujuan.

5. Tampilan rute pencarian berupa list

Setiap kumpulan rute yang ditemukan ditampilkan pada kotak berisi judul artikel yang dilalui pada rute tersebut.

#### **D. Contoh Ilustrasi Kasus**

1. Ilustrasi Kasus dengan BFS

Apabila diperlukan untuk mencari laman *wikipedia* perantara antara suatu laman awal dan laman target

2. Ilustrasi Kasus dengan IDS

Sama seperti kasus BFS.

## BAB IV IMPLEMENTASI DAN PENGUJIAN

### A. Spesifikasi Teknis Program

#### 1. Backend

##### a. Struktur Data pada Program

Tabel 4.1.1. Tabel struktur data (src/backend/models/\*)

Struktur Data	Deskripsi
<pre>type Page struct {     Title string `json:"title"`     Url string  `json:"url"` }</pre>	<p>Digunakan untuk merepresentasikan sebuah artikel wikipedia, memiliki atribut:</p> <ul style="list-style-type: none"> <li>- Title: bertipe string, menyimpan judul artikel dari halaman web</li> <li>- Url: bertipe string, menyimpan Url dari halaman web</li> </ul>
<pre>type Response struct {     Accessed int    `json:"accessed"`     N_step   int    `json:"n_step"`     Steps   [][]Page `json:"steps"`     Time    float64 `json:"time"` }</pre>	<p>Digunakan untuk merepresentasikan hasil pencarian yang dijadikan respons terhadap frontend, memiliki atribut:</p> <ul style="list-style-type: none"> <li>- Accessed: bertipe integer, menyimpan jumlah artikel yang telah diakses</li> <li>- N_step: bertipe integer, menyimpan jumlah langkah</li> <li>- Steps: bertipe <i>slice of slice</i> of Page, menyimpan kumpulan rute-rute yang ditemukan</li> <li>- Time: bertipe float64, menyimpan runtime pencarian</li> </ul>

##### b. Fungsi dan Prosedur yang Dibangun

Berikut adalah tabel berisi prosedur yang dibuat serta implementasinya pada program kami, untuk implementasi lengkapnya dapat dilihat di *repository* kami atau tangkapan layar yang dilampirkan pada bab LAMPIRAN:

Tabel 4.1.2. Tabel fungsi dan prosedur bagian bfs (src/backend/algorithm/bfs.go)

Nama fungsi atau prosedur	Penjelasan prosedur dan cara kerja
RunBFS()	<p>Merupakan fungsi untuk menjalankan algoritma <b>BFS</b> atau <b>Breadth First Search</b>. Menerima masukan berupa laman mulai, laman target, jumlah maksimal hasil, dan jumlah konkurensi yang diperbolehkan. Program akan berhenti apabila jumlah hasil yang diperoleh mencapai suatu angka yang dimasukkan oleh pengguna dan apabila semua solusi yang mungkin telah ditemukan. Program akan mencari semua solusi yang mungkin apabila masukan dari pengguna berupa angka 0 untuk jumlah hasil.</p> <p>Program akan memproses <i>child node</i> secara paralel sehingga diciptakan sistem</p>

	<p><i>mutex lock</i> dan <i>channel</i> untuk mencegah menghilangnya data dan <i>race conditions</i>.</p> <p>Program bekerja dengan menjadikan laman mulai sebagai <i>root node</i> dan mengiterasi secara rekursif untuk setiap <i>child node</i> yang diciptakan oleh <i>scraping</i> dari <i>root node</i>. Setiap <i>child node</i> yang ditemukan dan belum pernah diproses, akan dimasukkan kedalam <i>queue</i> untuk diproses pada iterasi selanjutnya. Apabila ditemukan laman target, akan digunakan <i>map</i> untuk melacak jejak <i>root node</i> asal sampai ditemukan laman mulai.</p>
runBFSHelper()	Merupakan fungsi yang membantu RunBFS() utama, fungsi ini melakukan <b>BFS</b> tanpa konkurensi. Fungsi ini memiliki <i>run time</i> yang jauh lebih lama ketimbang fungsi lain yang akan dibahas di bawah.
runBFSGoRoutine()	Merupakan fungsi yang membantu RunBFS() dalam menjalankan <b>BFS</b> secara konkuren atau paralel. Menerima jumlah operasi paralel maksimal untuk mencegah pemblokiran oleh <i>wikipedia</i> .

Tabel 4.1.3. Tabel fungsi dan prosedur pada *package "algorithm"* bagian ids (src/backend/algorithm/ids.go)

Nama fungsi atau prosedur	Penjelasan prosedur dan cara kerja
IDS()	Merupakan fungsi utama yang dipanggil untuk metode IDS. Fungsi menerima parameter berupa laman awal dan laman tujuan dan akan mengembalikan Result yang berisi banyaknya laman yang dikunjungi, jumlah langkah untuk mencapai laman tujuan, urutan langkah dari laman awal hingga laman akhir, dan waktu eksekusi. Karena IDS bisa dibilang DLS dengan kedalaman yang ditambah secara iteratif, dibuatlah fungsi DLS untuk membantu pencarian, dan akan memanggil fungsi DLS dengan depth mula-mula 0. Pemanggilan fungsi DLS akan dilakukan terus menerus dengan depth yang bertambah hingga ditemukan satu solusi
DLS()	Merupakan proses pencarian dengan algoritma DFS dengan kedalaman yang dibatasi. Menerima parameter berupa laman awal, laman tujuan, laman saat ini, dan kedalaman. Apabila kedalaman adalah 0 atau tidak ditemukan laman tujuan pada kedalaman tersebut, maka fungsi akan mengembalikan nil. Apabila laman tujuan ditemukan, maka fungsi akan mengembalikan langkah-langkah dari laman awal hingga laman akhir.

Tabel 4.1.4. Tabel fungsi dan prosedur pembantu (src/backend/algorithm/utills.go)

Nama fungsi atau prosedur	Penjelasan prosedur dan cara kerja
Scraper(pageUrl string)	Merupakan fungsi untuk mengekstrak link pada halaman wikipedia. Menerima masukan berupa URL halaman wikipedia yang ingin di- <i>scrape</i> , bertipe string. Program dimulai dengan menginisiasi sebuah collector dengan gocolly.

	Channel of Page juga dimanfaatkan untuk menyalurkan data selama proses scraping. Setiap “a” elemen pada halaman wikipedia yang beratribut href akan diekstrak, sebelum disalurkan ke channel, dicek apakah link mengandung string “/wiki”. Link yang mengandung “/Main_Page” dan “:” juga diabaikan untuk mempercepat pencarian. Scraper ini juga memanfaatkan goroutine dan waitgroup, dimana counter waitgroup akan ditambah dan tidak akan dikurangi sebelum proses scraping selesai. Fungsi mengembalikan hasil bertipe slice of Page.
GetTitle(url string)	Merupakan fungsi untuk mengekstrak/ <i>scrape</i> dan mengembalikan judul dari halaman masukan wikipedia dengan gocolly

Tabel 4.1.5. Tabel fungsi dan prosedur berkaitan api (src/backend/api/api.go)

Nama fungsi atau prosedur	Penjelasan prosedur dan cara kerja
handleBFS(c *fiber.Ctx)	Merupakan fungsi untuk handle endpoint method POST dengan endpoint “/bfs”. Melakukan parsing dari body context fiber (berisi request JSON dari fe), menyimpannya pada map(string to string), dan memanfaatkan data tersebut sebagai parameter fungsi RunBFS. Hasil kemudian dibentuk sebagai tipe Response dan dikembalikan sebagai JSON.
handleIDS(c *fiber.Ctx)	Merupakan fungsi untuk handle endpoint method POST dengan endpoint “/ids”. Melakukan parsing dari body context fiber (berisi request JSON dari fe), menyimpannya pada map(string to string), dan memanfaatkan data tersebut sebagai parameter fungsi IDS. Hasil kemudian dibentuk sebagai tipe Response dan dikembalikan sebagai JSON.
Init()	Menginisiasi objek fiber untuk menangani requests dan responses. menggunakan middleware cors untuk menangani permintaan lintas domain (Cross-Origin Resource Sharing). Juga digunakan middleware cors untuk menetapkan header HTTP yang tepat untuk memungkinkan akses request dari <a href="http://localhost:5173">http://localhost:5173</a> tanpa batasan CORS. Lalu, terdapat 2 rute, “/bfs” dan “/ids” dengan method POST yang akan di-handle dengan kedua fungsi di atas. Kemudian, aplikasi fiber disiapkan untuk menangani permintaan HTTP pada port 8080.

Pada src/backend.main.go, terdapat fungsi main() sebagai fungsi utama yang memanggil Init untuk inisiasi backend.

## 2. Frontend

### a. App.jsx

Komponen utama dalam aplikasi frontend. Terdapat beberapa komponen yang di-*import* dalam file ini, diantaranya Dashboard, dan Result.

### b. Dashboard.jsx

Komponen dashboard, merangkai Query input pengguna, switch jenis algoritma, dan slider jumlah rute masukan.

- c. Graph.jsx  
Komponen untuk visualisasi graf rute dengan menggunakan *library* d3js.
- d. Query.jsx  
Komponen menangani input masukan judul wikipedia serta melakukan fetch dari “<https://en.wikipedia.org/w/api.php?>” untuk fitur *suggestion*
- e. Result.jsx  
Komponen yang menampilkan hasil pencarian, termasuk Graph dan ResultBox
- f. ResultBox.jsx  
Komponen yang berisi judul-judul wikipedia dari sebuah rute pencarian.
- g. Slider.jsx  
Komponen untuk mengatur dan menerima masukan jumlah maksimal hasil yang ingin ditampilkan
- h. Switch.jsx  
Komponen untuk mengatur jenis algoritma yang digunakan

## **B. Sebagai fungsi Tata Cara Penggunaan Program**

### **1. Prosedur penggunaan program**

- a. Buka laman utama program
- b. Masukkan judul dari laman *wikipedia* pertama sebagai poin mulai pada *input box* sebelah kiri
- c. Masukkan judul dari laman *wikipedia* kedua sebagai poin *finish* pada *input box* sebelah kanan
- d. Pilih algoritma yang ingin digunakan, berupa *switch* IDS atau BFS
- e. (Opsional) Bila menggunakan BFS, masukkan jumlah hasil *path* yang ingin ditampilkan atau dicari pada *slider*
- f. Klik *Search* dan tunggu program selesai mengeksekusi
- g. Cermati hasil yang diperoleh

### **2. Batasan penggunaan program**

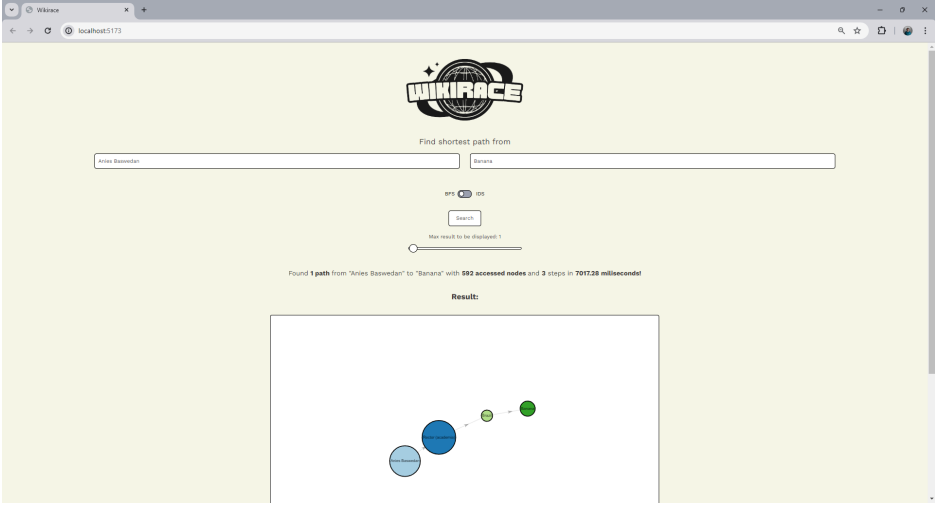
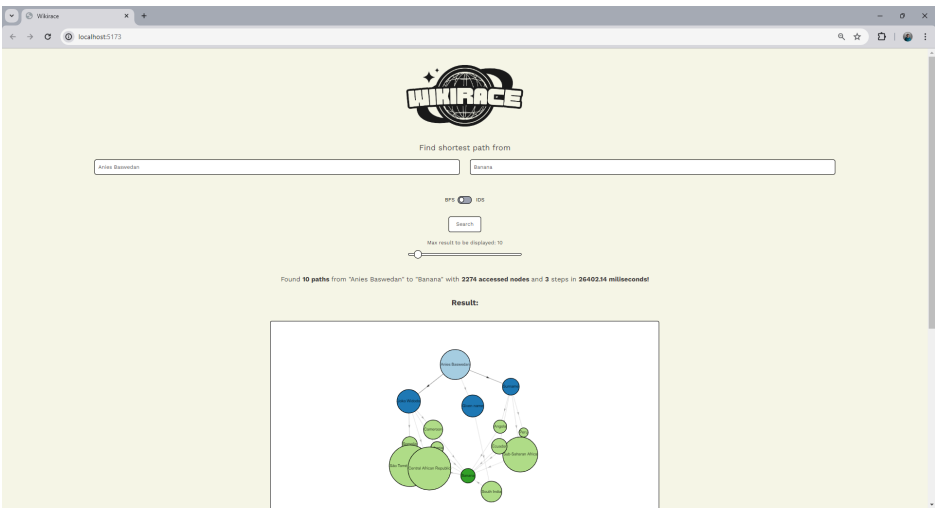
- a. Masukan harus disesuaikan dengan *autocomplete* yang disediakan oleh *search*

## C. Hasil Pengujian

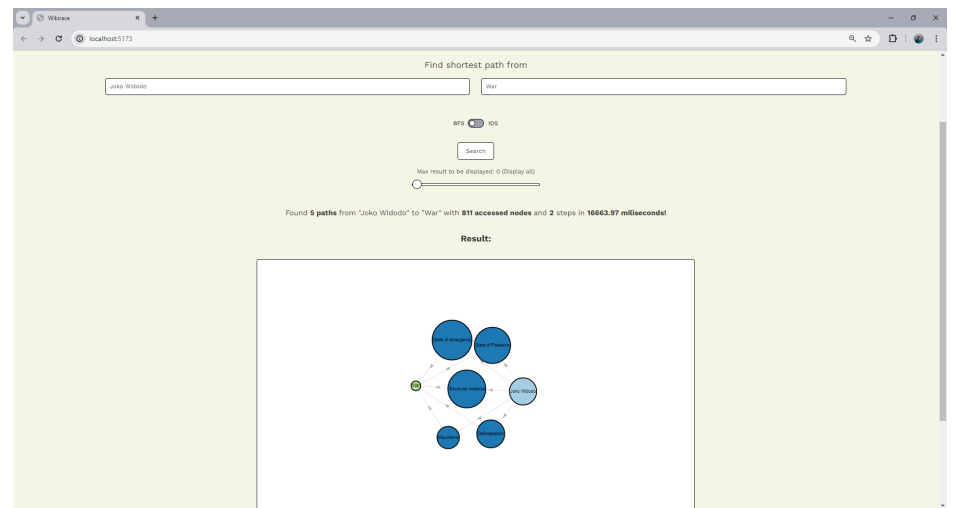
### a. Pengujian BFS

Berikut adalah tabel berisi pengujian dengan menggunakan algoritma BFS :

Tabel 4.2.1 Tabel fungsi dan prosedur pada *package* “algorithm” bagian bfs

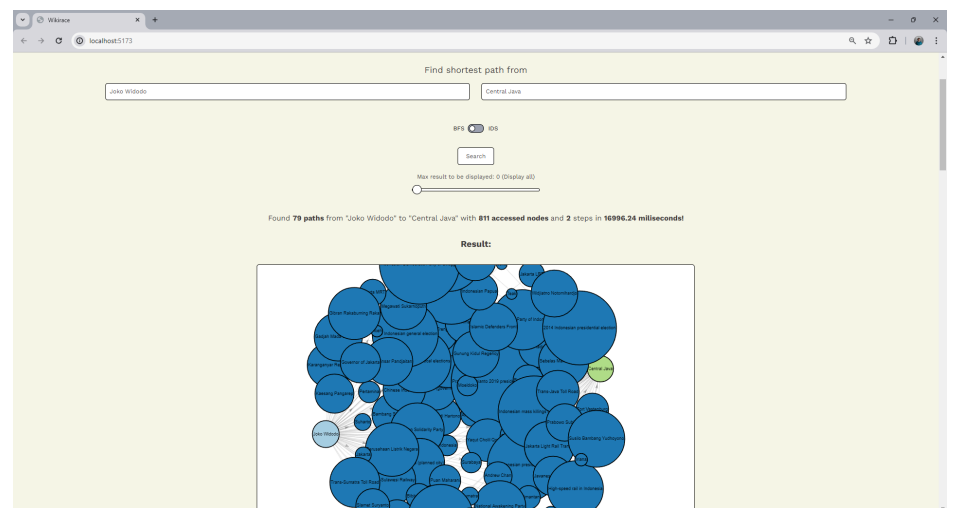
Laman Mulai - Laman Target, keterangan	Tangkap layar hasil
Anies Baswedan - Banana Maks hasil = 1	 <p>7017 ms</p>
Anies Baswedan - Banana Maks hasil = 10	 <p>26402 ms</p>

Joko Widodo - War  
Maks hasil = 0 (Tidak ada batasan)



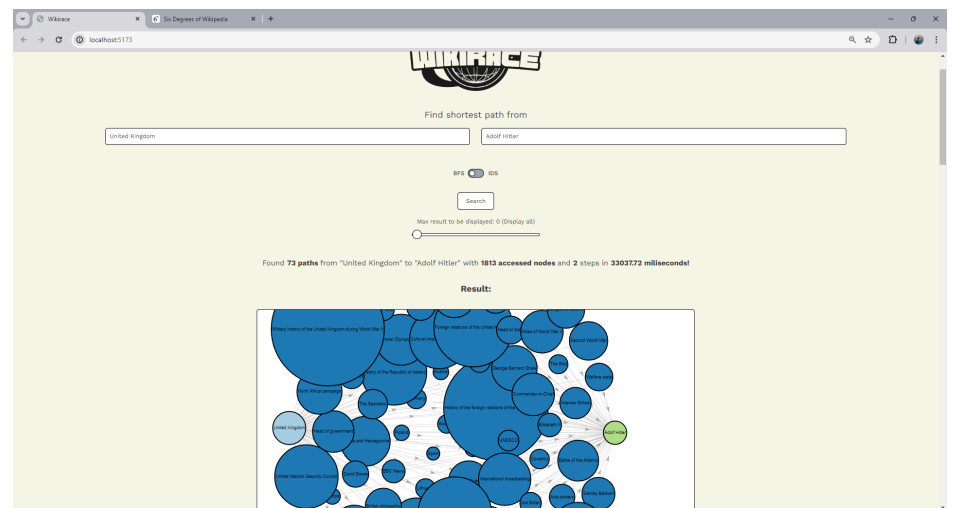
16663 ms

Joko Widodo - Central Java  
Maks hasil = 0 (Tidak ada batasan)



16996 ms

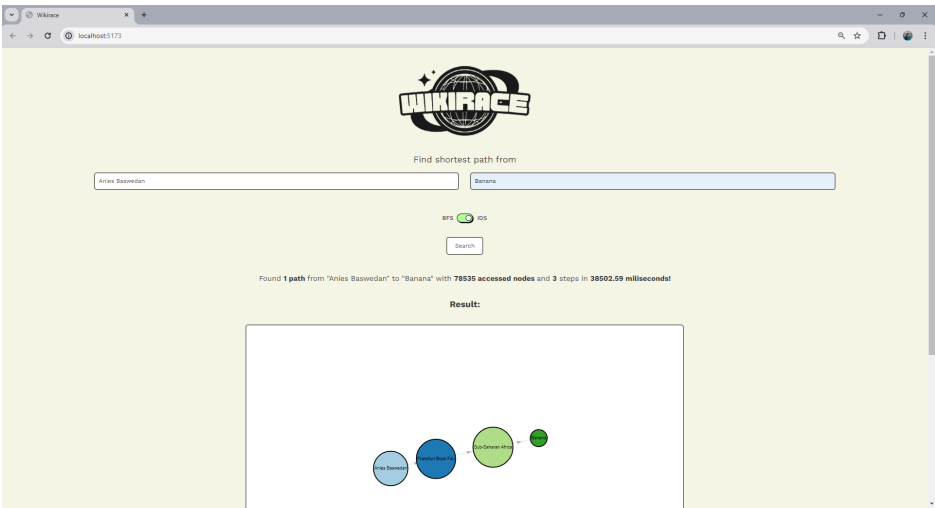
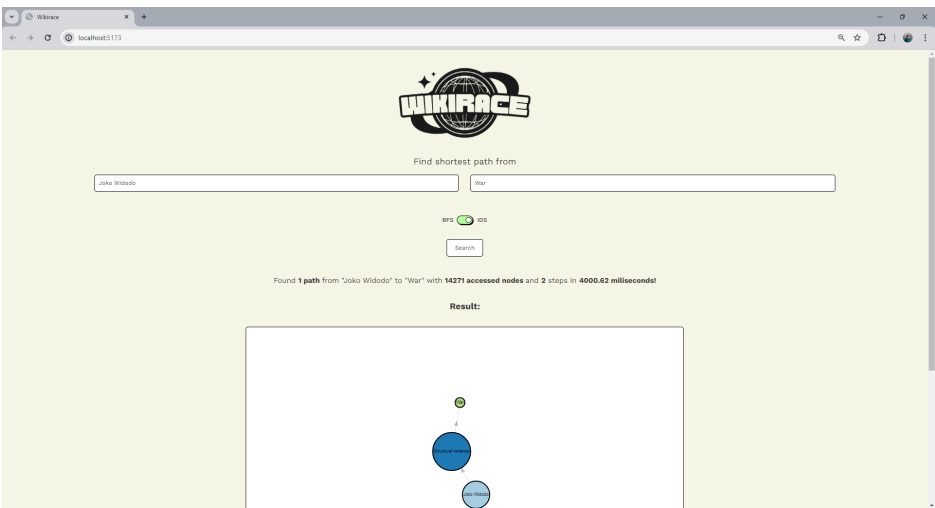
United Kingdom - Adolf Hitler  
Maks hasil = 0 (Tidak ada batasan)



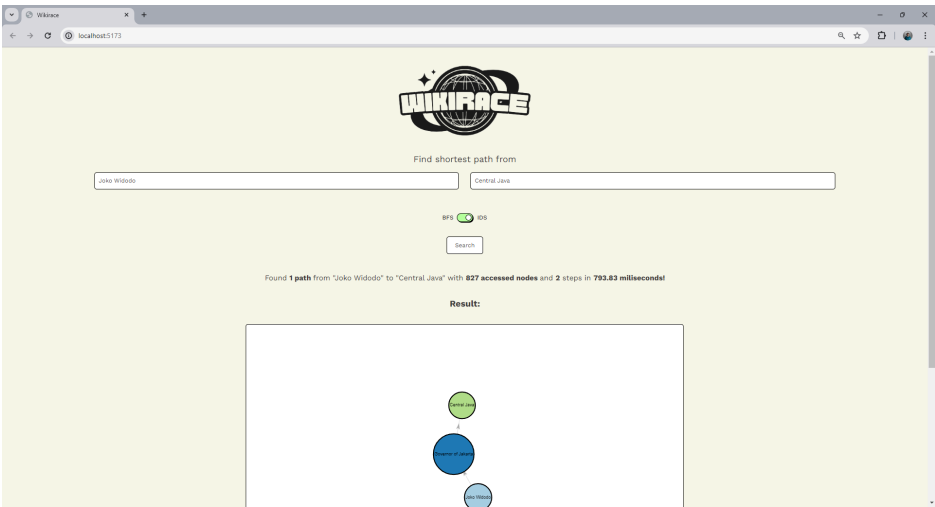
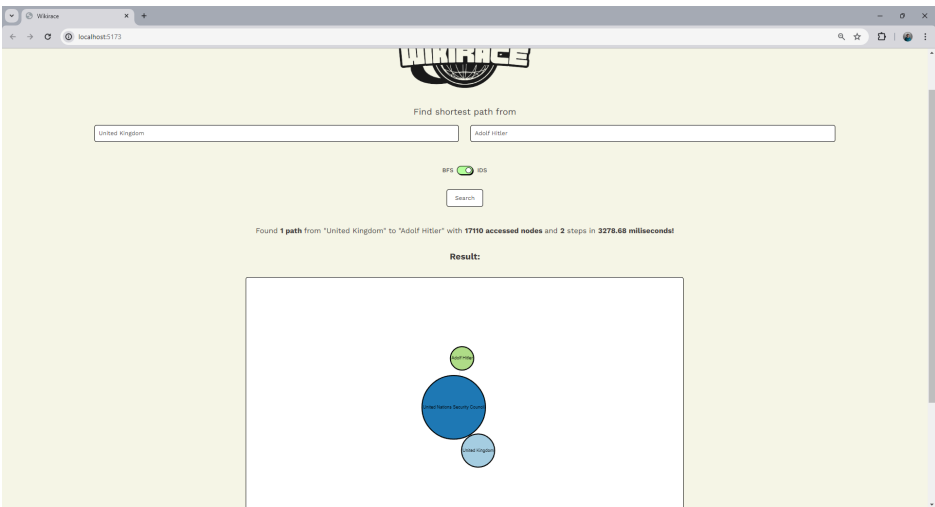
	33037 ms
--	----------

**b. Pengujian IDS**

Tabel 4.2.2 Tabel fungsi dan prosedur pada *package* “algorithm” bagian ids

Laman Mulai - Laman Target, keterangan	Tangkap layar hasil
Anies Baswedan - Banana	 <p>38502 ms</p>
Joko Widodo - War	 <p>4000 ms</p>



Joko Widodo - Central Java	 <p>793 ms</p>
United Kingdom - Adolf Hitler	 <p>3278 ms</p>

#### D. Analisis Hasil Pengujian

Tabel 4.3.1 Tabel perbandingan IDS dan BFS dalam rute, banyak laman diakses, kedalaman, dan waktu eksekusi

Test Case	Elemen	IDS	BFS
1 (Anies Baswedan - Banana)	Rute	Anies Baswedan → Frankfurt Book Fair → Sub-Saharan Africa → Banana	Anies Baswedan → Rector (academia) → Brazil → Banana
	Banyak laman yang diakses	78535	592
	Kedalaman	3	3

	Waktu eksekusi (dalam ms)	38502	7017
Joko Widodo - War	Rute	Joko Widodo → Structural Violence → War	Joko Widodo → Structural Violence → War
	Laman yang diakses	14271	811
	Kedalaman	2	2
	Waktu eksekusi	4000 ms	16663 ms
Joko Widodo - Central Java	Rute	Joko Widodo → Governor of Jakarta → Central Java	Joko Widodo → Governor of Jakarta → Central Java
	Laman yang diakses	827	811
	Kedalaman	2	2
	Waktu eksekusi	3278 ms	16996 ms
United Kingdom - Adolf Hitler	Rute	United Kingdom → United Nation Security Conference → Adolf Hitler	United Kingdom → United Nation Security Conference → Adolf Hitler
	Laman yang diakses	17130	1813
	Kedalaman	2	2
	Waktu eksekusi	3278 ms	33037 ms

Dari tabel analisis di atas, terdapat tren yang konsisten yaitu algoritma IDS selalu lebih cepat dari algoritma BFS, tetapi untuk jumlah laman yang dilalui selalu lebih banyak IDS. Terdapat beberapa hal yang dapat menjelaskan fenomena tersebut :

1. BFS akan terus berlanjut hingga mencari solusi yang mungkin, sedangkan IDS berhenti pada satu solusi
2. IDS akan memprioritaskan kedalaman dan melakukan *backtrack*, sehingga jumlah laman yang diakses akan relatif banyak
3. Urutan iterasi laman pada *parent node*. Bisa jadi jawaban ditemukan pada iterasi bagian ‘depan’ apabila iterasi dimulai dari ‘depan’ terlebih dahulu

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **A. Kesimpulan**

Algoritma IDS dan BFS merupakan salah dua algoritma pencarian graf traversal. Salah satu implementasi dari algoritma ini adalah penggunaannya dalam menyelesaikan permasalahan permainan WikiRace, dan pada tugas besar ini, algoritma telah dapat menyelesaikan persoalan tersebut. Meski demikian, terdapat beberapa kekurangan dan kelebihan dari masing-masing algoritma. Kelebihan dari algoritma IDS dan BFS adalah keduanya dapat dipastikan mencapai simpul tujuan dalam sebuah pencarian karena keduanya menjamin completeness (selama simpul tujuannya memang ada). Kekurangan dari BFS adalah kompleksitas ruangnya yang besar sehingga meskipun performa dioptimasi dengan dijalankan secara concurrent, terdapat masalah keterbatasan RAM dari gawai pengguna. Kekurangan dari IDS adalah kompleksitas waktunya yang besar dikarenakan iterasi dalam menerapkan DLS sebelum menemukan simpul tujuan.

#### **B. Saran**

Terdapat beberapa saran dalam pengerjaan tugas besar selanjutnya :

- Perlu dilakukan pendalaman dan pemahaman terlebih dahulu terhadap penggunaan bahasa Go dan sifat concurrency yang digunakan dalam pengembangan tubes ini.
- Perlunya perencanaan terlebih dahulu mengenai struktur dari program dan kelas-kelas apa saja yang akan digunakan untuk menyelesaikan persoalan.
- Perlu perencanaan terlebih dahulu mengenai algoritma BFS dan DFS yang ingin dikembangkan dapat menyelesaikan persoalan dengan optimal.

#### **C. Refleksi**

Dalam pengerjaan tugas ini sebaiknya dikerjakan dengan perancangan yang matang agar tidak merubah alur kode di tengah-tengah pengerjaan.

## LAMPIRAN

```
func RunBFS(pageUrl string, target string, max_goroutine int, max_result int) models.Response{ // Run
BFS function

    visited := make(map[string]bool)
    step := make(map[models.Page]models.Page)
    visited[pageUrl] = true

    var main_page models.Page

    main_page.Title = GetTitle(pageUrl)
    main_page.Url = pageUrl

    var input = []models.Page{main_page}

    // ret, depth, count := runBFSHelper(pageUrl, target, visited, step, input, 1, 0)
    ret, depth, count, runtime := runBFSGoRoutine(pageUrl, target, visited, input, step, 1,
max_goroutine, max_result) // Main BFS function
    fmt.Println(depth, " ", "count : ", count)

    for i, j := 0, len(ret)-1; i < j; i, j = i+1, j-1 { // Result reversal for valid output
        ret[i], ret[j] = ret[j], ret[i]
    }

    for i := range ret { // appending result with start page
        ret[i] = append(ret[i], main_page)
        ret[i] = reverse(ret[i])
    }

    if max_result != 0 {
        if max_result < len(ret){
            ret = ret[:max_result]
        }
    }

    var return_val models.Response

    return_val.N_step = depth
    return_val.Accessed = count
    return_val.Time = runtime
    return_val.Steps = ret

    return return_val
}
```

Gambar A.1 Source code algoritma BFS I

```

func runBFSGoRoutine(start string, target string, visited map[string]bool, to_be_processed []models.Page,
step map[models.Page]models.Page, depth int, max_go int, max_res int) ([]models.Page, int, int,
float64){
    st := time.Now()

    guard := make(chan struct{}, max_go)
    var lock sync.Mutex
    var wg sync.WaitGroup
    // Go routine guard, wait group, and mutex definition

    var next_process []models.Page // Used as queue
    var result [][]models.Page // Path result slice
    added := make(map[string]bool) // Validator for added url
    // Required data initialization

    for i := range to_be_processed {
        guard <- struct{}{}
        wg.Add(1)
        // Go routine constraints prep

        go func (page models.Page) {
            defer func(){
                <-guard
                wg.Done()
                // Routine temination
            }()

            temp := Scraper(page.Url) // Scraping

            lock.Lock() // Lock mutex due to writing data

            visited[page.Url] = true

            for j := range temp {
                if !visited[temp[j].Url]{
                    val := step[temp[j]]
                    if !val{
                        step[temp[j]] = page
                    }
                    if !added[temp[j].Url]{
                        next_process = append(next_process, temp[j])
                        added[temp[j].Url] = true
                    }
                }
                if temp[j].Url == target { // Found result process
                    fmt.Println("found from " + page.Url)
                    var path_i []models.Page
                    if page.Url == start {

                    } else {
                        path_i = getPath(start, step, page)
                    }
                    path_i = reverse(path_i)
                    path_i = append(path_i, temp[j])
                    path_i = reverse(path_i)
                    // Adding path to result slice of path

                    result = append(result, path_i)
                    break // Break out of loop since a result is found
                }
            }
        }(to_be_processed[i])

        if(len(result) >= max_res && max_res != 0){ // if a maximum amount on answer is not omitted
            break
        }
    }

    wg.Wait() // Wait for go routines termination
    // fmt.Println(len(result)) debugging purpose
    // fmt.Println(time.Since(st))

    total_time := time.Since(st).Seconds()

    if len(result) == 0 {
        depth++ // depth increment
        return runBFSGoRoutine(start, target, visited, next_process, step, depth, max_go, max_res)
    } else {
        return result, depth, len(visited), total_time
    }
}

```

Gambar A.2 Source code algoritma BFS II

```

func IDS(startTitle string, targetTitle string, startUrl string, targetUrl string) models.Result {
    var result models.Result
    var steps []models.Page

    var startPage models.Page
    startPage.Title = startTitle
    startPage.Url = startUrl

    var targetPage models.Page
    targetPage.Title = targetTitle
    targetPage.Url = targetUrl
    depth := 0

    start := time.Now()
    for {
        visited := make(map[string]bool)
        // fmt.Println(startPage)
        // fmt.Println(targetPage)
        path := DLS(startPage, targetPage, visited, steps, depth)
        if path != nil {
            end := time.Now()
            exe_time := end.Sub(start)
            result.Accessed = len(visited)
            result.N_step = depth
            result.Time = exe_time.Seconds()

            encountered := make(map[string]bool)
            res := []models.Page{}
            for _, i := range path {
                if (!encountered[i.Url]) {
                    encountered[i.Url] = true
                    i.Title = GetTitle(i.Url)
                    res = append(res, i)
                }
            }
            result.N_step = len(res) - 1

            result.Steps = res

            return result
        }
        depth++
    }
}

func DLS(currentPage models.Page, targetPage models.Page, visited map[string]bool, steps []models.Page,
depth int) []models.Page {
    if currentPage.Url == targetPage.Url {
        return []models.Page{currentPage}
    }
    if depth <= 0 {
        return nil
    }

    childUrl := Scraper(currentPage.Url)
    for _, page := range childUrl {
        if !visited[page.Url] {
            visited[page.Url] = true
            steps = append(steps, page)
            next := DLS(page, targetPage, visited, steps, depth-1)
            if next != nil {
                return append([]models.Page{currentPage}, next...)
            }
            steps = steps[:len(steps)-1]
        }
    }
    return nil
}

```

Gambar A.3 Source code algoritma IDS

- Tautan repository Github : [https://github.com/ibrahim-rasyid/Tubes2\\_lebaran-ya-balapan](https://github.com/ibrahim-rasyid/Tubes2_lebaran-ya-balapan)
- Tautan video : -

## DAFTAR PUSTAKA

Levitin, Anany. *Introduction to the Design & Analysis of Algorithms*. Addison-Wesley, 2003.

Munir, Rinaldi. *Algoritma Greedy 2021*. Diakses pada 27 April 2024

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2006-2007/BFS%20dan%20DFS.pdf>

<https://okanexe.medium.com/colly-web-scraping-in-golang-a-practical-tutorial-for-beginners-6e35cb3bd608>