

Laporan Tugas Besar 2 IF2211 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace
Semester II Tahun 2023/2024



Disusun oleh:
Kelompok 28 - Go-Jo (Golang Enjoiers)

1. Hugo Sabam Augusto 13522129
2. Nicholas Reymond Sihite 13522144
3. Muhammad Rasheed Qais Tandjung 13522158

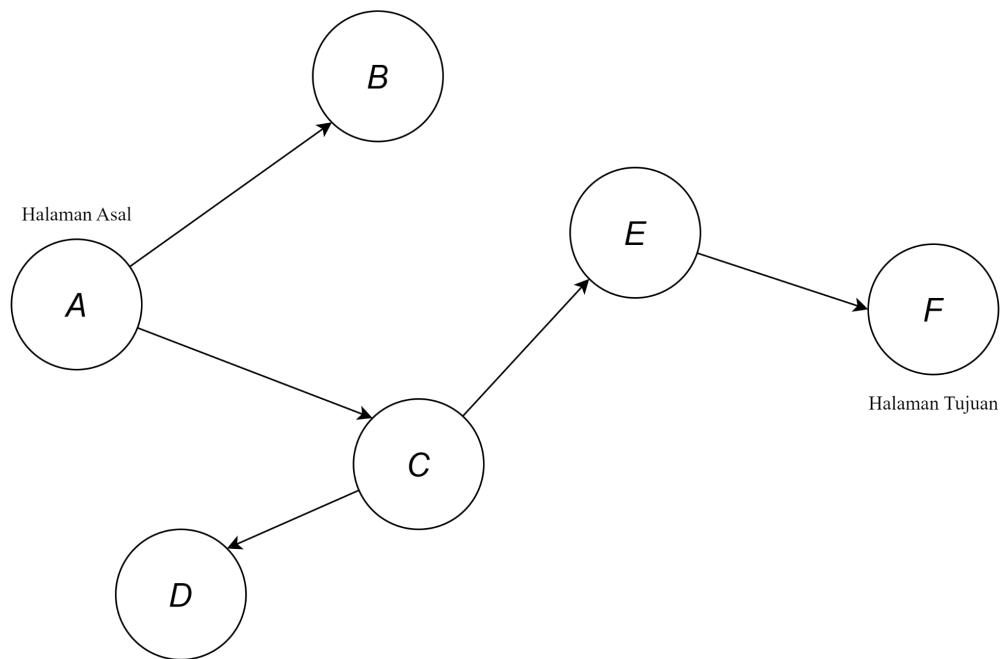
Daftar Isi

Daftar Isi	2
BAB I Deskripsi Masalah	3
BAB II Landasan Teori	4
2.1. Dasar Teori	4
2.1.1. Breadth-First Search	4
2.1.2. Iterative Deepening Search	4
2.2. Pengembangan Website	5
BAB III Analisis Pemecahan Masalah	6
3.1. Langkah Pemecahan Masalah	6
3.1.1. Pemodelan Masalah dalam Algoritma BFS	6
3.1.2. Pemodelan Masalah dalam Algoritma IDS	8
3.2. Arsitektur dan Fitur Website	10
3.3. Contoh Ilustrasi Kasus	11
3.3.1. Ilustrasi Kasus dengan Algoritma BFS	12
3.3.2. Ilustrasi Kasus dengan Algoritma IDS	13
BAB IV Implementasi dan Pengujian	15
4.1. Spesifikasi Teknis Program	15
4.2. Penjelasan Cara Penggunaan Program	29
4.3. Hasil Pengujian	31
4.4. Analisis Hasil Pengujian	39
Bab V Kesimpulan	41
5.1. Kesimpulan	41
5.2. Saran	41
Lampiran	42
Referensi	42

BAB I

Deskripsi Masalah

WikiRace atau Wiki Game adalah permainan yang melibatkan Wikipedia, sebuah ensiklopedia daring gratis yang dikelola oleh berbagai relawan di dunia, di mana 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 1.1. Ilustrasi Permainan WikiRace dari Halaman A ke F

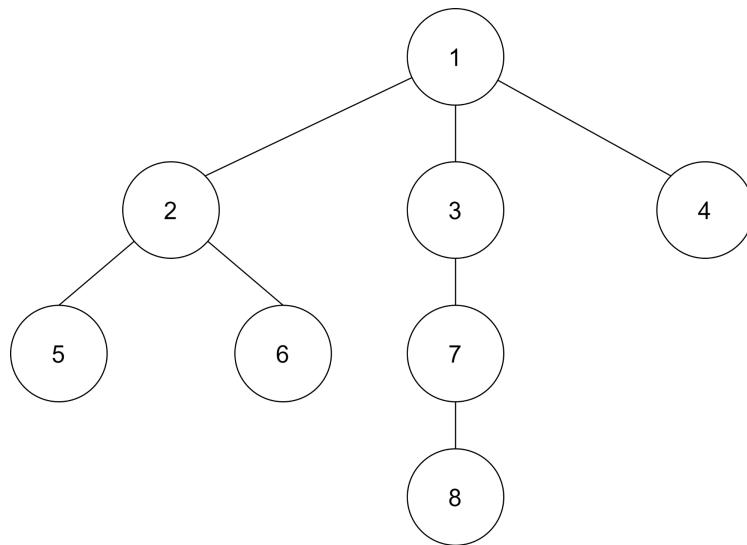
BAB II

Landasan Teori

2.1. Dasar Teori

2.1.1. Breadth-First Search

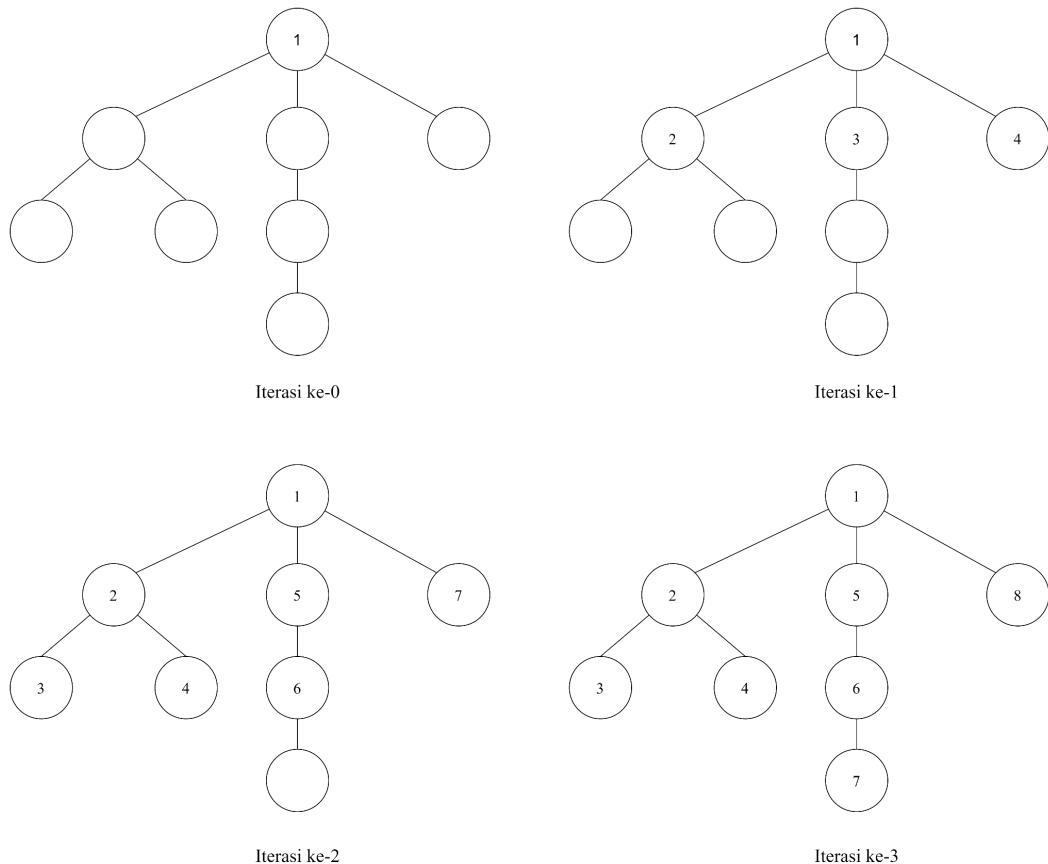
Breadth-First Search (BFS) adalah algoritma pencarian yang digunakan pada suatu graf dengan melakukan pencarian secara ‘melebar’ terlebih dahulu. Pencarian dalam graf dapat digambarkan sebagai sebuah pohon dengan simpul asal pada graf menjadi akar pada pohon. Berikut ilustrasi pencarian dengan algoritma *BFS*.



Gambar 2.1.1.1. Ilustrasi Algoritma BFS

2.1.2. Iterative Deepening Search

Iterative Deepening Search (IDS) adalah algoritma pencarian yang digunakan pada suatu graf dengan melakukan pencarian ‘mendalam’ secara bertahap. Kedalaman akan bertambah untuk setiap tahap dimulai dari kedalaman 0, 1, 2, dan seterusnya. Dengan kata lain, *IDS* merupakan *Depth Limited Search (DLS)* dengan kedalaman yang selalu bertambah. Pencarian dalam graf dapat digambarkan sebagai sebuah pohon dengan simpul asal pada graf menjadi akar pada pohon. Berikut ilustrasi pencarian dengan Algoritma *IDS*.



Gambar 2.1.2.1. Ilustrasi Algoritma IDS

2.2. Pengembangan Website

Website adalah (sekumpulan) halaman yang berada dalam satu domain. Pengembangan *website* adalah proses perancangan, pembangunan, dan optimasi sebuah *website* sehingga dapat berjalan sesuai dengan fungsi yang diinginkan.

Suatu *website* dapat dipecah menjadi dua bagian, yaitu *front-end* dan *back-end*. *Front-end* adalah bagian dari *website* yang terlihat oleh pengguna. Bagian ini meliputi elemen-elemen (contohnya HTTP), *style* (contohnya CSS), dan *script* (contohnya JavaScript). *Back-end* adalah bagian dari website yang seharusnya tidak terlihat oleh pengguna. Bagian ini meliputi algoritma pengolahan data yang dikirim dari *front-end* untuk kemudian dikembalikan. Pada Tugas Besar 2 ini, kami menggunakan HTTP, CSS, dan Go untuk *front-end* dan Go untuk *back-end*.

BAB III

Analisis Pemecahan Masalah

3.1. Langkah Pemecahan Masalah

Permasalahan utama dari tugas besar ini adalah mencari rute terpendek dari sebuah artikel ke artikel lainnya pada salah satu situs pencarian informasi terpopuler, yaitu Wikipedia. Sebuah artikel pada situs Wikipedia tersusun atas banyak *hyperlink* yang mengarah ke artikel-artikel lainnya sehingga rute terpendek yang dimaksud adalah sebuah kumpulan *hyperlink* terurut yang dimulai dari artikel awal dan berakhir di artikel tujuan.

Karena sebuah artikel Wikipedia memiliki sekumpulan *hyperlink* yang mengarah ke artikel lainnya, permasalahan ini dapat dimodelkan sebagai masalah pencarian sebuah simpul tertentu pada graf berarah. Sisi yang keluar dari sebuah simpul menandakan sebuah *hyperlink* yang ada di sebuah artikel, sementara sisi yang memasuki sebuah simpul menandakan sebuah *hyperlink* yang mengarah ke artikel tersebut. Dengan demikian, penyelesaian masalah ini cukup memanfaatkan algoritma traversal graf sederhana untuk melakukan pencarian rute, contohnya algoritma *Breadth-First Search (BFS)* atau *Iterative Deepening Search (IDS)*.

3.1.1. Pemodelan Masalah dalam Algoritma BFS

Penyelesaian masalah dengan algoritma *BFS* diawali dengan memodelkan masalah dari sebuah graf berarah menjadi sebuah pohon dengan akarnya berupa artikel awal. Kedalaman dari pohon tersebut menandakan jumlah *hyperlink* yang perlu dikunjungi dari artikel awal untuk mencapai artikel akhir (Jika artikel ditemukan pada kedalaman 0, artikel awal sama dengan artikel akhir. Jika artikel ditemukan pada kedalaman 1, perlu dikunjungi 1 *hyperlink* dari artikel awal untuk mencapai artikel akhir).

Algoritma *BFS* yang dirancang hanya memiliki tiga komponen: simpul-simpul berupa artikel Wikipedia, sisi-sisi yang keluar dari simpul berupa *hyperlink* yang dimiliki sebuah artikel, dan sebuah struktur data *queue* yang

menyimpan simpul-simpul yang akan dikunjungi. Prosedur pencarian rute dengan algoritma *BFS* adalah sebagai berikut:

1. Inisialisasi sebuah *queue* kosong yang akan menyimpan simpul-simpul yang ingin dikunjungi.
2. Masukkan artikel awal sebagai simpul pertama pada *queue*.
3. Untuk setiap simpul yang ada di *queue*, program akan membaca artikel yang terkait dan memasukkan semua *hyperlink* yang ditemukan ke *queue* mengikuti prosedur berikut:
 - a. Program mengambil simpul paling depan pada *queue* dan membacanya sebagai sebuah artikel.
 - b. Untuk seluruh *hyperlink* pada artikel, masukkan simpul yang terkait ke *queue*.
 - c. Ketika seluruh *hyperlink* sudah dicek, kembali ke langkah a.
4. Program akan terus melakukan *infinite loop* pada langkah ke-3 sampai ditemukan artikel tujuan.
5. Ketika ditemukan artikel tujuan, program mengembalikan rute berupa urutan *hyperlink* dari artikel awal ke artikel tujuan, jumlah lompatan, jumlah *hyperlink* yang dicek, dan durasi waktu pencarian dalam milisekon.

Algoritma *BFS* yang dirancang hanya memiliki tiga komponen: simpul-simpul berupa artikel Wikipedia, sisi-sisi yang keluar dari simpul berupa *hyperlink* yang dimiliki sebuah artikel, dan sebuah struktur data *queue* yang menyimpan simpul-simpul yang akan dikunjungi.

Pencarian rute dilakukan menggunakan metode *web scraping* dengan bahasa pemrograman *Go*, sehingga setiap kali program membaca sebuah artikel perlu dikirim *HTTP request* ke situs Wikipedia. Jumlah *request* yang dapat dilakukan per satuan waktu terbatas, dan untuk kasus-kasus tertentu jumlah *hyperlink* yang perlu dicek dapat mencapai angka jutaan, sehingga pencarian rute dengan algoritma sederhana ini dapat memakan waktu yang sangat lama. Untuk mempercepat waktu pencarian rute, ditambahkan beberapa strategi optimasi antara lain:

1. *Map* artikel yang sudah pernah dikunjungi: Ketika program menemukan *hyperlink* yang sudah pernah dikunjungi sebelumnya, program akan mengabaikan *hyperlink* tersebut dan langsung membaca *hyperlink* selanjutnya.
2. *Multithreading*: Program akan membagi rata tugas yang perlu diselesaikan ke seluruh *thread* yang tersedia di perangkat pengguna sehingga program akan bersifat paralel. Optimasi ini memanfaatkan fitur *goroutine* yang tersedia di bahasa pemrograman *Go*.
3. *Caching*: Program akan menyimpan seluruh artikel yang dikunjungi dan kumpulan *hyperlink*-nya pada sebuah file *cache.txt*. Ketika sebuah artikel yang ingin ditelusuri ditemukan pada *cache*, maka program tidak perlu melakukan *request* ke situs Wikipedia dan cukup menggunakan data yang tersedia pada *cache*.

3.1.2. Pemodelan Masalah dalam Algoritma IDS

Pemodelan masalah dengan algoritma *IDS* direpresentasikan sebagai bentuk pohon, dimana setiap simpul dalam pohon mewakili keadaan yang berbeda yang dapat dicapai dalam proses pencarian. Akar pohon (*root*) berupa keadaan awal atau *hyperlink* sumber yang ditentukan oleh pengguna. Setiap simpul dalam pohon memiliki cabang-cabang yang mewakili tindakan yang dapat diambil dari keadaan yang sesuai. Dalam tugas besar ini, konteks keadaanya adalah pencarian Wikipedia dan cabang cabang ini akan mewakili *hyperlink-hyperlink* dari 1 halaman ke lainnya.

Pada setiap iterasi *IDS*, pencarian akan dilakukan pada pohon pencarian dengan membatasi kedalaman, atau bisa dikatakan tiap iterasi merupakan *DLS/DFS - Limited Search*. Ini berarti pencarian akan dimulai dari *root* dan akan mencari secara rekursi di dalam pohon hingga mencapai kedalaman maksimum yang ditentukan. Jika solusi tidak ditemukan pada kedalaman tersebut, pencarian akan kembali ke akar dan kedalaman akan ditingkatkan untuk iterasi berikutnya.

Prosedur pencarian rute dengan algoritma *IDS* sebagai berikut:

1. Inisialisasi dengan menentukan halaman sumber, halaman tujuan, dan maksimum kedalaman
2. Memasuki iterasi *IDS*, proses pencarian dimulai dari kedalaman 0 hingga kedalaman maksimum dan juga mendeklarasi sebuah variabel *array of strings* kosong.
3. Pada setiap iterasi, pencarian dilakukan menggunakan *DLS* dengan kedalaman (*d*) tertentu
4. Pada setiap langkah dalam *DLS*, fungsi akan memeriksa apakah kedalaman saat ini telah mencapai batas maksimum atau tujuan telah ditemukan sebelumnya. Jika ya, pencarian akan dihentikan
5. Jika belum mencapai batas kedalaman atau tujuan belum ditemukan, *DLS* akan menjelajahi *hyperlink* dari halaman saat ini, untuk setiap *hyperlink* yang valid, *DLS* akan memeriksa apakah *hyperlink* tersebut adalah tujuan yang dicari. Jika ya, pencarian pada jalur tersebut dihentikan dan jalur dari halaman sumber ke halaman tujuan akan disimpan. Apabila tidak, *DLS* akan memanggil dirinya sendiri secara rekursif untuk menjelajahi *hyperlink* tersebut lebih dalam.
6. Setiap terjadi pencarian di rekursif, akan di *append* 1 buah *hyperlink* yang sedang ditelusuri ke dalam *array* dan di lempar sebagai argumen di pemanggilan *rekursif* lainnya
7. Ketika ditemukan artikel tujuan, program mengembalikan rute berupa rute *hyperlink* dari artikel awal ke artikel tujuan. Rute ini akan disimpan dalam sebuah *array of strings*
8. Selain rute, program juga mengembalikan jumlah lompatan, jumlah *hyperlink* yang ditelusuri, dan durasi waktu pencarian dalam milisekon.

Selama proses rekursi, jalur pencarian diperbarui dan diperluas dengan menambahkan halaman-halaman yang telah dikunjungi. Dengan menggunakan rekursivitas, *DLS* dapat secara efisien menjelajahi berbagai jalur pencarian dengan mempertahankan kedalaman yang ditentukan sambil menentukan rute path dari halaman sumber ke halaman tujuan.

Pencarian rute dilakukan sama halnya dengan BFS yang telah dibahas sebelumnya yaitu dengan metode *web scraping* dengan bahasa pemrograman Go. Ketika program membaca artikel, perlu melakukan *request* HTTP ke situs Wikipedia. Namun, jumlah *request* yang dapat dilakukan dalam satu waktu itu terbatas. Akibatnya, untuk beberapa kasus, pencarian rute menggunakan algoritma DLS ini bisa jadi sangat lama untuk di proses. Untuk mengurangi waktu pencarian yang memakan waktu ini, kami melakukan beberapa optimasi, yaitu antara lain:

1. *Caching* : Untuk menghindari pengulangan permintaan HTTP yang berulang, dilakukan caching dari halaman yang telah diambil. Dalam algoritma IDS ini, kami menggunakan caching sederhana dengan menggunakan struktur data map untuk menyimpan halaman web yang telah diambil sebelumnya. Sistem *mutex* juga kami implementasikan untuk caching pada IDS ini karena ketika *goroutine* ingin mengakses cache, bisa terjadi *Reader-Writer problem*, kondisi dimana *goroutine* rebutan cache untuk menulis atau membaca. Dengan adanya *mutex*, sinkronisasi dapat tetap terjaga
2. *Multithreading (Goroutine)* : Untuk mempercepat proses pencarian, dilakukan penggunaan konkurensi dengan penggunaan *Goroutine*. Setiap *hyperlink* yang valid dari halaman ini saat ini diproses secara konkuren. Fungsi *built-in* dari Go ini memungkinkan eksekusi beberapa tugas secara bersamaan sehingga dapat mempercepat proses pencarian. Namun, penggunaan *Goroutine* tidak seefektif itu dalam konteks rekursivitas karena apabila terlalu banyak *Goroutine* yang diciptakan, dapat menyebabkan *overhead* yang berlebihan. Karena itu, perlu diterapkan batasan pada jumlah *goroutine* yang dapat berjalan secara konkuren. Batasan ini dapat membantu mengatur jumlah permintaan HTTP yang dikirimkan secara bersamaan ke situs Wikipedia, sehingga tidak melebihi batas yang diterapkan.

3.2. Arsitektur dan Fitur Website

Bagian *front-end Website* dibangun pada satu halaman HTML dan satu halaman CSS saja. Struktur *website* dimulai dari judul, *form* untuk mengisi halaman Wikipedia awal dan tujuan serta tombol untuk menentukan algoritma yang ingin digunakan, dan diakhiri hasil pencarian, termasuk jika pencarian gagal (judul Wikipedia masukan tidak valid) dan berhasil. Bagian judul hanya berupa sebuah *tag div* dan *h1*. Bagian *form* dibuat dalam sebuah *tag div* yang kemudian masing-masing isinya (*input* dan *button*) dimasukkan ke dalam *tag div* juga. *Button* pemilihan algoritma juga merupakan *button* untuk mengirimkan *request* ke *back-end website*. Ada fitur tambahan yang dapat digunakan untuk menyimpan *cache* ke file *cache.txt*. Untuk mengaktifkannya, cukup tekan tombol yang bertuliskan “save cache” lalu kembali ke halaman utama.

Back-end website sendiri dibangun dengan bahasa Go dengan fitur fungsional dapat mencari rute terpendek antara dua buah halaman Wikipedia dengan algoritma *BFS* atau *IDS* dan dapat menyimpan *cache* pada file *cache.txt*. Di awal akan dilakukan validasi halaman Wikipedia yang dikirim. Jika salah satu atau kedua halaman tidak valid, *back-end* akan mengirimkan informasi bahwa halaman tidak valid dan algoritma *BFS* atau *IDS* tidak diterapkan sama sekali (*front-end* hanya menampilkan pesan halaman Wikipedia tidak valid). Jika kedua halaman *valid*, *back-end* akan menjalankan algoritma sesuai dengan *button* yang ditekan pengguna. Setelah rute didapat, *back-end* akan mengirimkan beberapa informasi yang di antaranya adalah rute dari halaman asal ke tujuan, waktu eksekusi algoritma, derajat pencarian, dan banyaknya artikel yang ditelusuri. Setelah itu, *front-end* akan menampilkan semua informasi tersebut pada bagian bawah *website*.

3.3. Contoh Ilustrasi Kasus

Sebagai contoh kasus, berikut merupakan ilustrasi pencarian dari halaman Wikipedia ‘Indonesia’ ke halaman ‘Elizabeth II’ dengan algoritma *BFS* dan *IDS*.

3.3.1. Ilustrasi Kasus dengan Algoritma *BFS*

Gambar 3.3.1.1. Tangkapan Layar Halaman Artikel ‘Indonesia’

Mengikuti prosedur algoritma pencarian rute *BFS* yang sudah didefinisikan pada Bagian 3.1.1, program akan menginisialisasi sebuah *queue* kosong dan memasukkan artikel ‘Indonesia’ ke *queue* sebagai artikel pertama yang ditelusuri. Lalu program akan mengecek seluruh *hyperlink* pada artikel ‘Indonesia’ dan memasukkannya ke *queue*.

Gambar 3.3.1.2. Tangkapan Layar Hyperlink ‘Elizabeth II’ Ditemukan di Artikel Papua New Guinea’

Salah satu *hyperlink* yang ditemukan di artikel ‘Indonesia’, yang mengarah ke artikel ‘Papua New Guinea’, mengandung *hyperlink* lain yang

mengarah ke artikel tujuan ‘Elizabeth II’. Maka program akan mengembalikan informasi bahwa rute dari ‘Indonesia’ ke ‘Elizabeth II’ berhasil ditemukan dengan urutan *hyperlink* ‘Indonesia’ → ‘Papua New Guinea’ → ‘Elizabeth II’.

3.3.2. Ilustrasi Kasus dengan Algoritma IDS

Memulai pencarian dengan inisialisasi halaman sumber yaitu ‘Indonesia’ dan halaman tujuan yaitu ‘Elizabeth II’. Mengikuti prosedur pencarian dengan algoritma IDS, awalnya akan dicari dahulu apakah halaman tujuan ada di iterasi pertama yaitu kedalaman 1. Apabila di kedalaman 1 tidak ditemukan, maka lanjut ke kedalaman 2 dan seterusnya. Untuk kasus ini, ditemukan halaman tujuan ‘Elizabeth II’ di kedalaman 2 dengan rute yang sudah ditunjukkan pada *screenshot* program di bawah ini

Sebanyak 90 artikel telah ditelusuri selama 587 ms.
Hasil ditemukan pada derajat 2 dengan Algoritma IDS:
Indonesia → Papua New Guinea → Elizabeth II

Gambar 3.3.2.1. Tangkapan Layar Hasil pencarian ‘Indonesia’ ke ‘Elizabeth II’

Ketika algoritma IDS ini sedang memulai pencarian rute pada kedalaman 2, Path yang sudah masuk ke rangkaian hanyalah ‘Indonesia’ saja. Dari halaman sumber ‘Indonesia’, algoritma akan melakukan rekursif sebanyak 2 kali karena iterasi ke-2 itu ketika kedalaman (d) sama dengan 2. Rekursi pertama sebenarnya sudah ditelusuri ketika iterasi pertama, namun untuk iterasi kedua dilakukan pencarian lagi ke dalam sebanyak 1 kali untuk tiap *hyperlink* yang telah ditemukan

Papua New Guinea

Article Talk

From Wikipedia, the free encyclopedia

This article is about the country in Oceania. For the island in general, see [New Guinea](#). For the Indonesian western half of the island, see [Western New Guinea](#). For other uses, see [Papua](#) and [Guinea \(disambiguation\)](#). For the single by the Future Sound of London, see [Papua New Guinea \(song\)](#).

Papua New Guinea^[a] is a country in [Oceania](#) that comprises the eastern half of the island of [New Guinea](#) and its offshore islands in [Melanesia](#) (a region of the southwestern [Pacific Ocean](#) north of Australia). Officially the [Independent State of Papua New Guinea](#)^[13] ([Tok Pisin: Independen Stet bilong Papua Niugini; Hiri Motu: Independen Stet bilong Papua Niu Gini](#)), it shares its only land border with [Indonesia](#) to the west and it is directly adjacent to [Australia](#) to the south and the [Solomon Islands](#) to the east. Its capital, located along its southeastern coast, is [Port Moresby](#). The country is the world's third largest [island country](#), with an area of 462,840 km² (178,700 sq mi).^[14]

At the national level, after being ruled by three external powers since 1883, including nearly 60 years of Australian administration starting during [World War I](#), Papua New Guinea established its sovereignty in 1975, becoming an independent [Commonwealth realm](#) with [Elizabeth II](#) as its queen. Since [Elizabeth II's](#) death in 2022, [Charles III](#) has been the country's king. It is also a member of the [Commonwealth of Nations](#) in its own right.

Independent State of Papua New Guinea
Independen Stet bilong Papua Niugini (Tok Pisin)
Independen Stet bilong Papua Niu Gini (Hiri Motu)




Motto: "Unity in diversity"^[1]

Gambar 3.3.2.2. Tangkapan Layar Hyperlink Artikel ‘Papua New Guinea’

Dalam ilustrasi kasus ini, ditemukan halaman Papua New Guinea yang merupakan cabang dari *root* ‘Indonesia’. Halaman ini sudah diketahui ketika iterasi pertama, dan halaman ini menjadi sebuah *root* untuk cabang-cabang halaman berikutnya pada iterasi ke-2. Ketika halaman ‘Papua New Guinea’ ditelusuri semua cabang-cabangnya (karena sudah pada kedalaman maksimum di iterasi ke-2, yaitu *d* sama dengan 2), ditemukan halaman ‘Elizabeth II’. Pencarian langsung dihentikan dan mengembalikan rute yang sesuai dengan hasil yang dicari yaitu “Indonesia” → ‘Papua New Guinea’ → ‘Elizabeth II’ dengan total loncatan (derajat) yaitu 2.

BAB IV

Implementasi dan Pengujian

4.1. Spesifikasi Teknis Program

Tabel 4.1.1. Spesifikasi Struktur Data, Fungsi, dan Prosedur Program

No	Nama	Deskripsi
1	<pre>type struct wikiGameInfo { Source string Destination string }</pre>	Struktur data yang digunakan untuk menyimpan informasi halaman Wikipedia yang menjadi asal dan tujuan pencarian.
2	<pre>type struct resultStruct { Path []string Degrees int Time int Artikel int }</pre>	Struktur data yang digunakan untuk menyimpan hasil pencarian BFS dan IDS. <i>Path</i> digunakan untuk menyimpan judul-judul halaman Wikipedia menuju tujuan, <i>Degrees</i> digunakan untuk menyimpan derajat kedalaman pencarian, <i>Time</i> digunakan untuk menyimpan waktu eksekusi algoritma, dan <i>Artikel</i> digunakan untuk menyimpan banyak halaman Wikipedia yang dikunjungi.
3	<pre>type ListNode[T any] struct { Value T next *ListNode[T] }</pre>	Tipe bentukan <i>node</i> yang merupakan elemen dari struktur data <i>QueueLinked</i> . Tipe ini memiliki atribut <i>value</i> yang menyimpan data dari <i>node</i> dan atribut <i>next</i> yang menyimpan alamat <i>node</i> selanjutnya. Tipe bentukan ini merupakan <i>template struct</i> sehingga tipe data yang disimpan dapat berupa tipe data apapun.
4	<pre>type QueueLinked[T any] struct { head *ListNode[T] tail *ListNode[T] }</pre>	Struktur data yang merupakan representasi <i>double-ended queue</i> berjenis <i>linked list</i> . Struktur ini menyimpan <i>list of nodes</i> yang merupakan tipe <i>template</i> sehingga struktur data ini juga merupakan tipe <i>template</i> .
5	<pre>type Link struct { title string url string path []string iter int }</pre>	Struktur data yang merepresentasikan sebuah simpul berupa artikel pada pohon pencarian yang berakar di artikel awal. Struktur ini menyimpan <i>hyperlink</i> artikel, judul artikel, rute dari artikel awal ke artikel tersebut, serta kedalaman simpul tersebut pada pohon pencarian.
6	<pre>var cache struct { sync.RWMutex m map[string][]byte size int64 limit int64 }</pre>	Struct <i>cache</i> struktur data yang digunakan untuk menyimpan data caching dalam algoritma IDS. Struktur ini memiliki beberapa parameter, di antaranya <i>m</i> yang merupakan map yang menyimpan konten halaman web dalam bentuk byte array, <i>size</i> yang menyimpan ukuran total cache saat ini, dan <i>limit</i> yang merupakan batasan ukuran maksimum cache yang dapat disimpan. Struct ini juga menggunakan <i>sync.RWMutex</i> untuk mengatur akses ke data cache secara aman dari beberapa goroutine.

7	<code>main()</code>	Prosedur ini digunakan untuk menjalankan program utama.
	<pre>func main() { go func() { log.Println(http.ListenAndServe("localhost:6060", nil)) } ReadCache() http.HandleFunc("/cache", WriteCacheContainer) http.HandleFunc("/", WikiGame) http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("static")))) link := "http://localhost:8080" fmt.Println("Server dimulai pada ", link) http.ListenAndServe(":8080", nil) }</pre>	
8	<code>init()</code>	Prosedur ini digunakan untuk melakukan <i>parsing</i> file HTML.
	<pre>func init() { tmpl = template.Must(template.ParseFiles("main.html")) }</pre>	
9	<code>WikiGame(w http.ResponseWriter, r *http.Request)</code>	Prosedur ini digunakan untuk mengolah masukan dari <i>website</i> dan mengembalikan hasil pengolahan ke <i>website</i> .
	<pre>func WikiGame(w http.ResponseWriter, r *http.Request) { if r.Method != http.MethodPost { tmpl.Execute(w, nil) return } infoSrcDest := wikiGameInfo{ Source: r.FormValue("src"), Destination: r.FormValue("dest"), } algorithm := r.FormValue("algorithm") succeed := true validSrc := false validDest := false sent := true result := fmt.Sprintf("%s -> %s", infoSrcDest.Source, infoSrcDest.Destination) srcLink := fmt.Sprintf("https://en.wikipedia.org/wiki/%s", infoSrcDest.Source) destLink := fmt.Sprintf("https://en.wikipedia.org/wiki/%s", infoSrcDest.Destination) if isValidWikiLink(srcLink) {</pre>	

	<pre> validSrc = true } if isValidWikiLink(destLink) { validDest = true } var emptylist []string finalResult := resultStruct{ Path: emptylist, Degrees: 0, Time: 0, Artikel: 0, } if (validSrc) && (validDest) { if algorithm == "IDS" { finalResult = searchIDS(infoSrcDest.Source, infoSrcDest.Destination, 10) } else { finalResult = BFS(infoSrcDest.Source, infoSrcDest.Destination) } // Create result path result = "" for i, page := range finalResult.Path { result += page if i != len(finalResult.Path)-1 { result += " → " } } } tmpl.Execute(w, struct { Sent bool Success bool ValidSrc bool ValidDest bool InfoSrcDest wikiGameInfo Results resultStruct Result string Algorithm string }{sent, succeed, validSrc, validDest, infoSrcDest, finalResult, result, algorithm}) } </pre>	
10	<pre>isValidWikiLink(url string) → bool</pre> <pre>func isValidWikiLink(url string) bool { resp, err := http.Get(url) if err != nil { fmt.Println("Error:", err) return false } }</pre>	Fungsi ini digunakan untuk memeriksa apakah halaman Wikipedia yang dimasukkan valid atau tidak.

	<pre> } defer resp.Body.Close() return resp.StatusCode >= 200 && resp.StatusCode < 300 } </pre>	
11	<pre> getTitleFromUrl(url string) → func getTitleFromURL(url string) string { if strings.HasPrefix(url, "/wiki/") { if idx := strings.Index(url, "#"); idx != -1 { url = url[:idx] } return strings.TrimPrefix(url, "/wiki/") } return "" } </pre>	Fungsi ini digunakan untuk mendapatkan judul dari website berdasarkan <i>url</i> yang diberikan
12	<pre> getFromCache(url string) → (*goquery.Document, bool) func getFromCache(url string) (*goquery.Document, bool) { cache.RLock() entry, found := cache.m[url] cache.RUnlock() if !found { return nil, false } doc, err := goquery.NewDocumentFromReader(strings.NewReader(string(entry))) if err != nil { return nil, false } return doc, true } </pre>	Fungsi ini mengambil dokumen HTML dari cache berdasarkan <i>url</i> yang diberikan. Jika berhasil, fungsi ini mengembalikan tuple yang terdiri dari dokumen dengan boolean <i>true</i> , jika tidak, mengembalikan <i>nil</i> dan <i>false</i>
13	<pre> cacheDocument(url string, doc *goquery.Document) func cacheDocument(url string, doc *goquery.Document) { html, err := doc.Html() if err != nil { return } cache.Lock() } </pre>	Prosedur ini digunakan untuk menyimpan dokumen HTML ke dalam cache map. Prosedur ini mengambil konten HTML dari dokumen yang diberikan, kemudian menyimpannya dalam cache menggunakan URL sebagai kunci. Prosedur ini juga dilengkapi dengan <i>mutex</i> untuk mencegah <i>Reader-Writer</i> problem dalam penulisan cache dikarenakan konkurensi <i>Goroutine</i> sehingga sinkronisasi tetap terjaga

	<pre> defer cache.Unlock() entrySize := int64(len(html)) if cache.size+entrySize > cache.limit { for key := range cache.m { delete(cache.m, key) cache.size -= int64(len(key)) + int64(len(cache.m[key])) if cache.size+entrySize <= cache.limit { break } } } cache.m[url] = []byte(html) cache.size += entrySize } </pre>	
14.	<pre>isValidLink(link string, input string, path *[]string) → bool</pre>	Fungsi ini digunakan untuk memeriksa apakah link tersebut memenuhi syarat untuk ditelusuri. Apabila suatu link memenuhi syarat, fungsi akan mengembalikan nilai true, jika tidak, mengembalikan nilai false.
	<pre> func isValidLink(link string, input string, path *[]string) bool { for _, visitedLink := range *path { if visitedLink == link { return false } } return strings.HasPrefix(link, "/wiki/") && !strings.Contains(link, ":") && !strings.Contains(link, "Main_Page") && getTitleFromURL(link) != input } </pre>	
15.	<pre>removeChar(url string, c string) → string</pre>	Fungsi ini digunakan untuk mengembalikan string tanpa karakter tertentu c.
	<pre> func removeChar(url string, c string) string { title := strings.TrimPrefix(url, "/wiki/") title = strings.ReplaceAll(title, c, " ") return title } </pre>	
16.	<pre>dls(input string, url string, destination string, maxDepth int, currentDepth int, reachedDestination *bool, path *[]string, finalpath *[]string)</pre>	Prosedur ini digunakan untuk mencari solusi dari pranala awal hingga ke pranala tujuan. Prosedur algoritma pencarian dengan <i>DLS</i> menerima beberapa parameter, termasuk URL awal, URL tujuan, kedalaman maksimum pencarian, kedalaman pencarian saat ini, dan status pencapaian destinasi. Selama pencarian, prosedur memeriksa setiap tautan dalam halaman web yang diakses kemudian memanggil dirinya sendiri secara rekursif untuk menjelajahi tautan yang valid. Pencarian dihentikan jika destinasi ditemukan, batas kedalaman tercapai, atau jika pencarian telah dihentikan secara manual.
	<pre> func dls(input string, url string, destination string, maxDepth int, currentDepth int, reachedDestination *bool, path *[]string, finalpath *[]string) { if currentDepth > maxDepth atomic.LoadInt32(&destinationFound) == 1 *reachedDestination { </pre>	

```

        return
    }

    doc, found := getFromCache(url)
    if !found {
        var err error
        doc, err = goquery.NewDocument(url)
        if err != nil {
            log.Printf("Error loading %s: %v", url, err)
            return
        }
        cacheDocument(url, doc)
    }

    var wg sync.WaitGroup
    stopSearch := make(chan struct{})
    var concurrencyLimit chan struct{}
    concurrencyLimit = make(chan struct{}, goroutineLimit)

    doc.Find("a").Each(func(i int, s *goquery.Selection) {
        link, exists := s.Attr("href")
        if exists && isValidLink(link, input, path) {
            visitedLinks.Lock()
            if !visitedLinks.m[url] {
                visitedLinks.m[url] = true
                visitedLinks.Unlock()
                uniqueLinkCount++
            } else {
                visitedLinks.Unlock()
            }
            newPath := append(*path, getTitleFromURL(link))
            if destination == getTitleFromURL(link) || link == "/wiki/" + destination || destination == removeChar(getTitleFromURL((link)), "_") {
                if !*reachedDestination {
                    *path = newPath //Path success nemu
                    *finalpath = append(*finalpath, input)
                    *finalpath = append(*finalpath, newPath...)
                    for idx, flink := range *finalpath {
                        (*finalpath)[idx] = removeChar(flink, "_")
                    }
                }
                *reachedDestination = true
                if atomic.LoadInt32(&stopSearchClosed) == 0 {
                    close(stopSearch)
                    atomic.StoreInt32(&stopSearchClosed, 1)
                }
                return
            }
            select {
            case <-stopSearch:
                return
            }
        }
    })
}

```

	<pre> case concurrencyLimit <- struct{}{}: wg.Add(1) go func(link string) { defer func() { <-concurrencyLimit wg.Done() }() dls(input, "https://en.wikipedia.org"+link, destination, maxDepth, currentDepth+1, reachedDestination, &newPath, finalpath) }(link) } }) wg.Wait() } </pre>	
17	<pre>searchIDS(source_link string, destination_link string, maxdepth int) → resultStruct</pre>	Fungsi ini melakukan iterasi melalui kedalaman pencarian dari 0 hingga maksimum kedalaman maksimal, dan untuk setiap iterasi, memanggil fungsi dls untuk melakukan pencarian dengan kedalaman tersebut. Jika destinasi ditemukan, fungsi mengembalikan <i>struct</i> resultStruct yaitu hasil pencarian yang mencakup jalur, kedalaman, waktu yang diperlukan, dan jumlah artikel yang dikunjungi.

```

func searchIDS(source_link      string, destination_link      string, maxdepth int)
resultStruct {
    var final_path []string
    degree := 0
    waktu := 0

    if(source_link == destination_link){
        final_path = append(final_path, removeChar(source_link, "_"))
    }else{
        if err != nil {
            log.Fatal(err)
        }

        log.SetOutput(ioutil.Discard)
        var path []string
        url := "https://en.wikipedia.org/wiki/" + source_link
        start := time.Now()

        for i := 0; i < maxdepth; i++ {
            fmt.Println("Searching in depth ", i+1, "...")
            dls(source_link, url, destination_link, i, 0,
&reachedDestination, &path, &final_path, cache)
            if reachedDestination {
                finish := time.Now()
                elapsed := finish.Sub(start)
                waktu = int(elapsed.Milliseconds())
                fmt.Println("Time : ", elapsed)
                fmt.Println("Total unique links:", uniqueLinkCount)
            }
        }
    }
}

```

	<pre> fmt.Println("Path:", source_link, " -> ", strings.Join(final_path, " -> ")) fmt.Println("Destination reached!") fmt.Println("Depth : ", i+1) degree = i + 1 break } else { fmt.Println("Destination not found in depth ", i+1) } } reachedDestination = false } result := resultStruct{ Path: final_path, Degrees: degree, Time: waktu, Artikel: uniqueLinkCount, } uniqueLinkCount = 0 visitedLinks.Lock() defer visitedLinks.Unlock() visitedLinks.m = make(map[string]bool) return result } </pre>	
18	CreateWikiURL(title string) → string	Fungsi ini menerima sebuah judul artikel Wikipedia dan mengembalikan <i>hyperlink</i> yang mengarah ke artikel Wikipedia tersebut.
	<pre> func CreateWikiURL(title string) string { return "https://en.wikipedia.org/wiki/" + title } </pre>	
19	getHTMLDocument(url string) → *goquery.Document	Fungsi ini menerima sebuah <i>hyperlink</i> ke artikel Wikipedia dan mengembalikan dokumen HTML dari artikel tersebut.
	<pre> func getHTMLDocument(url string) *goquery.Document { doc, err := goquery.NewDocument(url) if err != nil { log.Fatal(err) } return doc } </pre>	
20	loadHTML(url string) → (*goquery.Document, string)	Fungsi ini menerima sebuah <i>hyperlink</i> ke artikel Wikipedia dan mengembalikan dokumen HTML serta judul dari artikel tersebut.
	<pre> func loadHTML(url string) (*goquery.Document, string) { var doc *goquery.Document = getHTMLDocument(url) var title string = doc.Find(".mw-page-title-main").First().Text() </pre>	

	<pre> var title_alt string = doc.Find(".firstHeading.mw-first-heading").First().Text() if title == "" { title = title_alt } return doc, title } </pre>	
21	<pre>isValidURL(url string, title string) → bool</pre>	Fungsi ini menerima sebuah <i>hyperlink</i> dan judul artikel dan mengembalikan apakah <i>hyperlink</i> tersebut merupakan <i>hyperlink</i> Wikipedia yang valid (khusus untuk keperluan tugas besar ini).
	<pre> func IsValidURL(url string, title string) bool { return strings.HasPrefix(url, "/wiki/") && !strings.Contains(url, ":") && !strings.Contains(url, "#") && !strings.Contains(url, "#") && url != "/wiki/Main_Page" && title != "View the content page [c]" } </pre>	
22	<pre>WriteAndPrintRoot(iter int, title string, path []string)</pre>	Prosedur ini akan melakukan <i>printing</i> informasi terkait artikel yang sedang dicek program.
	<pre> func WriteAndPrintRoot(iter int, title string, path []string) { fmt.Print(Green) fmt.Printf("%d ", iter + 1) for _ = range iter { fmt.Printf("----") } var pathString string = strings.Join(path, " -> ") fmt.Println(" Root: " + Yellow + title + Reset + " (" + pathString + ")") } </pre>	
23	<pre>CheckFound(title string, path []string, iter int) → bool</pre>	Fungsi ini akan mengecek apakah <i>hyperlink</i> ke artikel tujuan sudah ditemukan dengan mengecek apakah judul artikel yang diberikan sama dengan judul artikel tujuan.
	<pre> func CheckFound(title string, path []string, iter int) bool { var pathString string = strings.Join(path, " -> ") if strings.EqualFold(title, endTitle) { pathString += " -> " + title fmt.Println(Green + "\nPath:\n" + Reset + pathString) found = true ResultPath = append(path, title) ResultDegrees = iter + 1 ResultArtikel = articles } return found } </pre>	
24	<pre>GetLinks(doc *goquery.Document, docTitle string, iter int, path []string)</pre>	Prosedur ini akan melakukan iterasi terhadap seluruh <i>hyperlink</i> pada artikel tertentu dan mengecek sekaligus memasukkan <i>hyperlink</i> tersebut ke <i>hyperlink queue</i> .

		Prosedur ini akan mendapatkan data daftar <i>hyperlink</i> dari dokumen hasil <i>web scraping</i> .
	<pre>func GetLinks(doc *goquery.Document, docTitle string, iter int, path []string) { var linkList []string = []string{} doc.Find("a").EachWithBreak(func(i int, s *goquery.Selection) bool { link, exists := s.Attr("href") title, _ := s.Attr("title") if exists && IsValidURL(link, title) { if CheckFound(title, path, iter) { return false } linkList = append(linkList, TitleToLink(title)) var newURL string = "https://en.wikipedia.org" + link queueMutex.Lock() queue.Enqueue(Link{title, newURL, append(path, title), iter + 1}) queueMutex.Unlock() } return true }) if !found { cacheBFSMutex.Lock() cacheBFS[TitleToLink(docTitle)] = linkList cacheBFSMutex.Unlock() } }</pre>	
25	<pre>GetLinksCache(parent string, iter int, path[] string)</pre>	Prosedur ini akan melakukan iterasi terhadap seluruh <i>hyperlink</i> pada artikel tertentu dan mengecek sekaligus memasukkan <i>hyperlink</i> tersebut ke <i>hyperlink queue</i> . Prosedur ini akan mendapatkan data daftar <i>hyperlink</i> dari <i>cache</i> yang disimpan di <i>disk</i> pengguna.

	<pre> queueMutex.Lock() queue.Enqueue(Link{title, newURL, append(path, title), iter + 1}) queueMutex.Unlock() } } </pre>	
26	BFSTraversal()	<p>Prosedur ini adalah algoritma BFS utama pada program yang akan melakukan <i>infinite loop</i> sampai sebuah rute dari artikel awal ke artikel tujuan ditemukan. Pada setiap iterasi <i>infinite loop</i>, prosedur akan mengambil artikel paling depan pada <i>queue</i> dan mengecek sekaligus memasukkan ke <i>queue</i> semua <i>hyperlink</i> yang ditemukan pada artikel tersebut dengan memanggil prosedur ‘GetLinks’ atau ‘GetLinksCache’.</p> <pre> func BFSTraversal() { for !found { for queue.IsEmpty() { time.Sleep(10 * time.Millisecond) } queueMutex.Lock() var L Link = queue.Dequeue() queueMutex.Unlock() var iter int = L.iter var title string = L.title var url string = L.url var path []string = L.path var hasSeen bool = false seenMutex.RLock() if seen[title] { hasSeen = true } seenMutex.RUnlock() if hasSeen title == "" { continue } articlesMutex.Lock() articles++ articlesMutex.Unlock() if CheckFound(title, path, iter) { return } seenMutex.Lock() seen[title] = true seenMutex.Unlock() cacheBFSMutex.RLock() } } </pre>

	<pre> _, ok := cacheBFS[TitleToLink(title)] cacheBFSMutex.RUnlock() workerSem <- struct{}{} go func() { defer func() { <-workerSem }() if ok { fmt.Println(Yellow + "Cache hit: " + Reset) WriteAndPrintRoot(iter, title, path) GetLinksCache(TitleToLink(title), iter, path) } else { fmt.Println(Red + "Cache miss: " + Reset) WriteAndPrintRoot(iter, title, path) var doc *goquery.Document var docTitle string doc, docTitle = loadHTML(url) GetLinks(doc, docTitle, iter, path) } }() if !ok { time.Sleep(22 * time.Millisecond) } } } </pre>	
27	BFS(startPage string, endPage string) → resultStruct	Fungsi ini merupakan sebuah <i>wrapper function</i> yang berperan sebagai jalur komunikasi antar algoritma dan <i>website</i> . Fungsi akan menerima parameter artikel awal dan artikel akhir dari <i>website</i> , menjalankan pencarian BFS dengan parameter tersebut, dan mengembalikan data-data yang diperlukan oleh <i>website</i> dalam bentuk <i>struct resultStruct</i> .

```

func BFS(startPage string, endPage string) resultStruct {
    articles = 0
    var startURL string = CreateWikiURL(startPage)
    var endURL string = CreateWikiURL(endPage)

    _, startTitle = loadHTML(startURL)

    _, endTitle = loadHTML(endURL)

    seen = make(map[string]bool)

    queue = QueueLinked[Link]{}
    queue.Enqueue(Link{startTitle, startURL, []string{startTitle}, 0})

    found = false

    if strings.EqualFold(startTitle, endTitle) {

```

	<pre> return resultStruct{ Path: []string{startTitle}, Degrees: 0, Time: 0, Artikel: 0, } } start := time.Now() BFSTraversal() duration := time.Since(start) seconds := duration.Seconds() ms := duration.Milliseconds() fmt.Println(Green + "\nTime taken: " + Yellow + strconv.FormatFloat(seconds, 'f', 6, 64) + " sec" + Reset) fmt.Println(Green + "Time taken: " + Yellow + strconv.FormatInt(ms, 10) + " ms" + Reset) return resultStruct{ Path: ResultPath, Degrees: ResultDegrees, Time: int(ms), Artikel: ResultArtikel, } } </pre>	
28	<pre>TitleToLink(title string) → string</pre> <pre>func TitleToLink(title string) string { return strings.Replace(title, " ", "_", -1); }</pre>	Fungsi ini mengubah judul artikel menjadi <i>hyperlink</i> artikel dengan menggantikan karakter ‘ ‘ menjadi ‘ _ ’.
29	<pre>LinkToTitle(link string) → string</pre> <pre>func LinkToTitle(link string) string { return strings.Replace(link, "_", " ", -1); }</pre>	Fungsi ini mengubah <i>hyperlink</i> menjadi judul artikel dengan menggantikan karakter ‘ _ ’ menjadi ‘ ’.
30	<pre>ReadCache()</pre> <pre>func ReadCache() { file, err := os.OpenFile("cache.txt", os.O_RDWR, 0644) if err != nil { log.Fatalf("Error opening file: %v", err) } defer file.Close() scanner := bufio.NewScanner(file) buf := make([]byte, 0, 64*1024) scanner.Buffer(buf, 1024*1024*1024) for scanner.Scan() { line := scanner.Text() } }</pre>	Prosedur akan membaca data <i>cache</i> yang sudah disimpan di <i>cache.txt</i> ke memori.

		<pre> parts := strings.Fields(line) if len(parts) == 0 { continue } key := parts[0] values := parts[1:] cacheBFS[key] = append(cacheBFS[key], values...) } if err := scanner.Err(); err != nil { log.Fatalf("Error reading file: %v", err) } } </pre>
31	WriteCache()	Prosedur ini akan menulis data <i>cache</i> yang ada di memori ke file <i>cache.txt</i> .
	<pre> func WriteCache() { os.Remove("cache.txt") file2, err := os.OpenFile("cache.txt", os.O_CREATE, 0644); if err != nil { log.Fatalf("Error opening file: %v", err) } defer file2.Close() for key, values := range cacheBFS { line := fmt.Sprintf("%s %s\n", key, strings.Join(values, " ")) _, err := file2.WriteString(line) if err != nil { log.Fatalf("Error writing to file: %v", err) } } } </pre>	
32	(q *QueueLinked[T]) Enqueue(item T)	Prosedur ini merupakan <i>method</i> pada tipe bentukan <i>QueueLinked</i> yang memasukkan sebuah elemen bertipe T ke bagian belakang <i>queue</i> .
	<pre> func (q *QueueLinked[T]) Enqueue(item T) { newNode := &ListNode[T]{value: item, next: nil} if q.tail == nil { q.head = newNode q.tail = newNode } else { q.tail.next = newNode q.tail = newNode } } </pre>	
33	(q *QueueLinked[T]) EnqueueHead(item T)	Prosedur ini merupakan <i>method</i> pada tipe bentukan <i>QueueLinked</i> yang memasukkan sebuah elemen bertipe T ke bagian depan <i>queue</i> .
	<pre> func (q *QueueLinked[T]) EnqueueHead(item T) { </pre>	

	<pre> newNode := &ListNode[T]{value: item, next: q.head} if q.head == nil { q.head = newNode q.tail = newNode } else { q.head = newNode } } </pre>	
34	<code>(q *QueueLinked[T]) Dequeue() → T</code> <pre> func (q *QueueLinked[T]) Dequeue() T { if q.head == nil { panic("Queue is empty") } item := q.head.value q.head = q.head.next if q.head == nil { q.tail = nil } return item } </pre>	Fungsi ini merupakan <i>method</i> pada tipe bentukan <i>QueueLinked</i> yang mengembalikan sekaligus menghilangkan elemen paling depan pada <i>queue</i> .
35	<code>(q *QueueLinked[T]) IsEmpty() → bool</code> <pre> func (q *QueueLinked[T]) IsEmpty() bool { return q.head == nil } </pre>	Fungsi ini merupakan <i>method</i> pada tipe bentukan <i>QueueLinked</i> yang mengembalikan apakah <i>queue</i> kosong.

4.2. Penjelasan Cara Penggunaan Program

Program digunakan dengan terlebih dahulu melakukan *clone* terhadap *repository* GitHub pada link [berikut](#) (terlampir juga pada Lampiran). Silakan masukkan perintah berikut pada *terminal*.

```
git clone https://github.com/miannetopokki/Tubes2_Go-Jo.git
```

Selanjutnya, pindah ke folder “src” dengan membuka *terminal* pada folder tersebut atau memasukkan perintah berikut pada *terminal* yang sebelumnya digunakan untuk melakukan *clone*.

```
cd Tubes2_Go-Jo/src
```

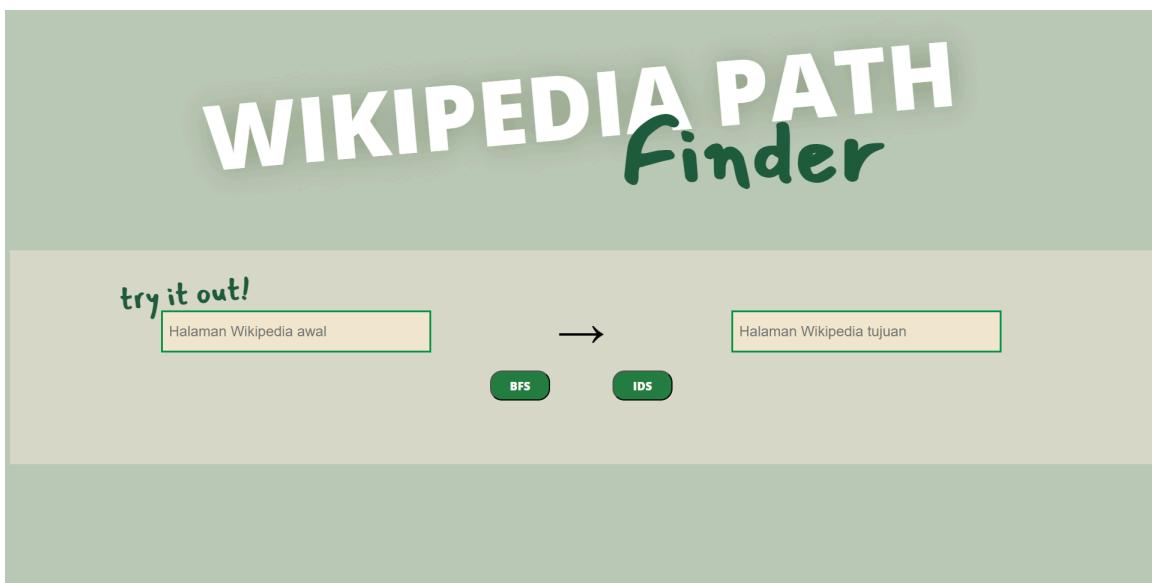
Pada folder “src”, buat file *cache.txt* terlebih dahulu. Setelah itu, lakukan *compile* terhadap semua file .go yang ada dengan memasukkan perintah berikut.

```
go run main.go IDSR.go BFS.go BFSCache.go queueLinked.go
```

Jika ada *firewall*, akan muncul permintaan untuk memperbolehkan atau melarang akses ke internet. Tekan tombol ‘Allow’. Setelah itu, *website* akan berjalan pada alamat berikut.

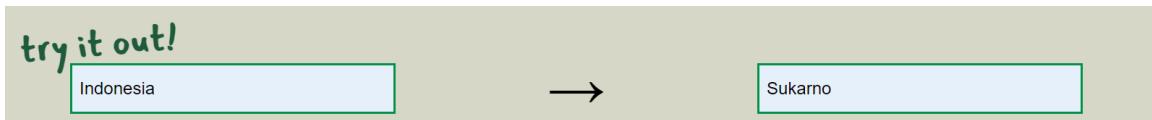
```
https://localhost:8080
```

Website dapat dibuka dengan mengetikkan langsung alamat tersebut atau melakukan [CTRL] + click pada teks yang muncul di *terminal*. Tampilan awal *website* adalah sebagai berikut.



Gambar 4.2.1. Tampilan Awal Website

Selanjutnya, silakan masukkan judul halaman Wikipedia awal dan tujuan masing-masing pada kotak yang berisi teks *placeholder* “Halaman Wikipedia awal” dan “Halaman Wikipedia tujuan” seperti pada gambar berikut.



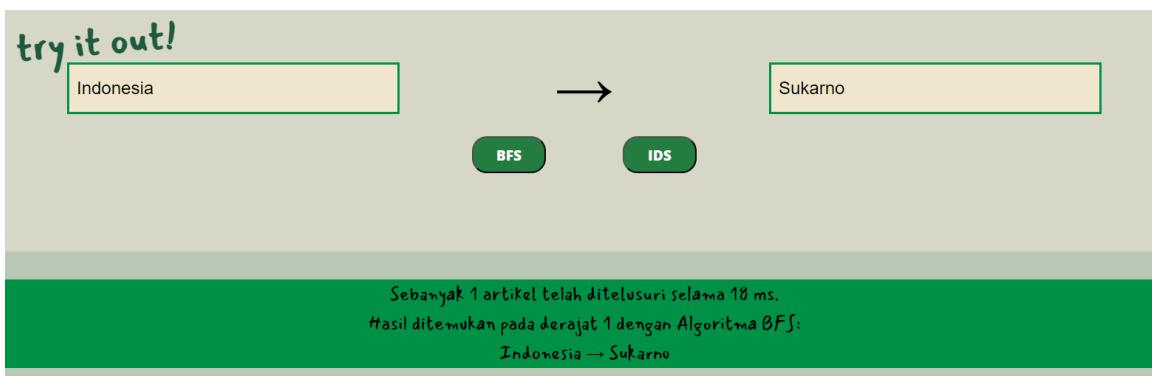
Gambar 4.2.2. Pengisian Halaman Wikipedia

Setelah itu, pilih salah satu algoritma yang ingin dijalankan dengan menekan tombol yang sesuai seperti pada gambar berikut.



Gambar 4.2.3. Pemilihan Algoritma

Hasil (banyak artikel yang dikunjungi, waktu eksekusi, derajat pencarian, dan rute pencarian) akan tampil setelah pencarian selesai dilakukan seperti pada gambar berikut.



Gambar 4.2.4. Hasil Pencarian Rute Halaman Wikipedia Indonesia ke Sukarno

4.3. Hasil Pengujian

Tabel 4.3.1. Hasil Pengujian Program

No	Kasus	
1	Awal : Indonesia Tujuan : Sukarno Algoritma : BFS	

	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Indonesia</div> → <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Sukarno</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; text-align: center; font-size: small;"> Sebanyak 1 artikel telah ditelusuri selama 19 ms. Hasil ditemukan pada derajat 1 dengan Algoritma BFS: Indonesia → Sukarno </div>
Hasil	<p>Artikel : 1 artikel Waktu : 19 ms Derajat : 1 Rute : Indonesia → Sukarno</p>
2	<p>Awal : Indonesia Tujuan : Sukarno Algoritma : IDS</p>
	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Indonesia</div> → <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Sukarno</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; text-align: center; font-size: small;"> Sebanyak 1 artikel telah ditelusuri selama 186 ms. Hasil ditemukan pada derajat 1 dengan Algoritma IDS: Indonesia → Sukarno </div>
Hasil	<p>Artikel : 1 artikel Waktu : 186 ms Derajat : 1 Rute : Indonesia → Sukarno</p>
3	<p>Awal : awudbadwabidwabdiwa Tujuan : Singapore Algoritma : IDS</p>

	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 20px;">awudbadwabidwabdiwa</div> → <div style="border: 1px solid black; padding: 5px; margin-right: 20px;">Singapore</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: red; color: white; text-align: center; padding: 5px; margin-top: 10px;">Halaman Wikipedia asal tidak valid!</div>
Hasil	<p>Awal : awudbadwabidwabdiwa Tujuan : tubesstima2 Algoritma : <i>IDS</i></p>
	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 20px;">awudbadwabidwabdiwa</div> → <div style="border: 1px solid black; padding: 5px; margin-right: 20px;">tubesstima2</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: red; color: white; text-align: center; padding: 5px; margin-top: 10px;">Halaman Wikipedia asal tidak valid! Halaman Wikipedia tujuan tidak valid!</div>
Hasil	<p>Awal : The Man Tujuan : Joko_Widodo Algoritma : <i>BFS</i></p>
4	
5	

	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">The Man</div> <div style="margin: 0 auto; width: 40px;">→</div> <div style="border: 1px solid black; padding: 5px; text-align: center;">Joko_Widodo</div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; margin-top: 20px; font-size: small;"> Sebanyak 783 artikel telah ditelusuri selama 10109 ms. Hasil ditemukan pada derajat 3 dengan Algoritma BFS: The Man → United States → Aide-de-camp → Joko Widodo </div>
Hasil	<p>Artikel : 783 artikel Waktu : 10109 ms Derajat : 3 Rute : The Man → United States → Aide-de-camp → Joko Widodo</p>
6	<p>Awal : The Man Tujuan : Joko_Widodo Algoritma : IDS</p>
	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">The Man</div> <div style="margin: 0 auto; width: 40px;">→</div> <div style="border: 1px solid black; padding: 5px; text-align: center;">Joko_Widodo</div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; margin-top: 20px; font-size: small;"> Sebanyak 124 artikel telah ditelusuri selama 5943 ms. Hasil ditemukan pada derajat 3 dengan Algoritma IDS: The Man → Slang → Indonesian slang → Joko Widodo </div>
Hasil	<p>Artikel : 124 artikel Waktu : 5943 ms Derajat : 3 Rute : The Man → Slang → Indonesian slang → Joko Widodo</p>
7	<p>Awal : Pneumonoultramicroscopicsilicovolcanoconiosis Tujuan : Convergence (comics) Algoritma : BFS</p>

	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <p>BFS → IDS</p> <p>Sebanyak 76697 artikel telah ditelusuri selama 1953803 ms. Hasil ditemukan pada derajat 4 dengan Algoritma BFS: Pneumonoultramicroscopicsilicovolcanoconiosis → English language → Acronym → Surnames by country → Convergence (comics)</p>
Hasil	<p>Artikel : 76697 artikel Waktu : 1953803 ms Derajat : 4 Rute : Pneumonoultramicroscopicsilicovolcanoconiosis → English language → Acronym → Surnames by country → Convergence (comics)</p>
8	<p>Awal : Lontong Tujuan : Albert Einstein Algoritma : IDS</p>
	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <p>BFS → IDS</p> <p>Sebanyak 997 artikel telah ditelusuri selama 89462 ms. Hasil ditemukan pada derajat 3 dengan Algoritma IDS: Lontong → Java → UNESCO → Albert Einstein</p>
Hasil	<p>Artikel : 997 artikel Waktu : 89462 ms Derajat : 3 Rute : Lontong → Java → UNESCO → Albert Einstein</p>
9	<p>Awal : Indonesia Tujuan : Elizabeth II Algoritma : BFS</p>

	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Indonesia</div> → <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;">Elizabeth II</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; text-align: center; font-size: small;"> Sebanyak 90 artikel telah ditelusuri selama 1020 ms. Hasil ditemukan pada derajat 2 dengan Algoritma BFS: Indonesia → Papua New Guinea → Elizabeth II </div>
Hasil	<p>Artikel : 90 artikel Waktu : 1020 ms Derajat : 2 Rute : Indonesia → Papua New Guinea → Elizabeth II</p>
10	<p>Awal : Indonesia Tujuan : Elizabeth II Algoritma : IDS</p>
	<h1>WIKIPEDIA PATH Finder</h1> <p>try it out!</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid #ccc; padding: 5px; margin-right: 20px;">Indonesia</div> → <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;">Elizabeth II</div> </div> <div style="display: flex; justify-content: space-around; width: fit-content; margin: auto;"> BFS IDS </div> <div style="background-color: #008000; color: white; padding: 5px; text-align: center; font-size: small;"> Sebanyak 90 artikel telah ditelusuri selama 3202 ms. Hasil ditemukan pada derajat 2 dengan Algoritma IDS: Indonesia → Papua New Guinea → Elizabeth II </div>
Hasil	<p>Artikel : 90 artikel Waktu : 3202 ms Derajat : 2 Rute : Indonesia → Papua New Guinea → Elizabeth II</p>
11	<p>Awal : Sun Tujuan : Pillow Algoritma : BFS</p>

	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <p>Save Cache</p> <p>Sun → Pillow</p> <p>BFS IDS</p> <p>Sebanyak 9031 artikel telah ditelusuri selama 42088 ms. Hasil ditemukan pada derajat 3 dengan Algoritma BFS: Sun → Carbon → Textiles → Pillow</p>
Hasil	<p>Artikel : 9031 artikel Waktu : 42088 ms Derajat : 3 Rute : Sun → Carbon → Textiles → Pillow</p>
12	<p>Awal : Sun Tujuan : Pillow Algoritma : IDS</p>
	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <p>Save Cache</p> <p>Sun → Pillow</p> <p>BFS IDS</p> <p>Sebanyak 8806 artikel telah ditelusuri selama 53255 ms. Hasil ditemukan pada derajat 3 dengan Algoritma IDS: Sun → Oxygen → Textile → Pillow</p>
Hasil	<p>Artikel : 8806 artikel Waktu : 53255 ms Derajat : 3 Rute : Sun → Oxygen → Textile → Pillow</p>
13	<p>Awal : Lontong Tujuan : Kukulkan Algoritma : BFS</p>

	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <p>Lontong → Kukulkan</p> <p>BFS IDS</p> <p>Sebanyak 1862 artikel telah ditelusuri selama 6762 ms. Hasil ditemukan pada derajat 3 dengan Algoritma BFS: Lontong → List of cuisines → Maya cuisine → Kukulkan</p>
Hasil	<p>Artikel : 1862 artikel Waktu : 6762 ms Derajat : 3 Rute : Lontong → List of cuisines → Maya cuisine → Kukulkan</p>
14	<p>Awal : Lontong Tujuan : Kukulkan Algoritma : IDS</p>
	<p>WIKIPEDIA PATH Finder</p> <p>try it out!</p> <p>Lontong → Kukulkan</p> <p>BFS IDS</p> <p>Sebanyak 2060 artikel telah ditelusuri selama 10290 ms. Hasil ditemukan pada derajat 3 dengan Algoritma IDS: Lontong → Indonesian cuisine → Snake → Kukulkan</p>
Hasil	<p>Artikel : 2060 artikel Waktu : 10290 ms Derajat : 3 Rute : Lontong → Indonesian cuisine → Snake → Kukulkan</p>
15	<p>Awal : Indonesia Tujuan : Operating system Algoritma : BFS</p>

	 <p>Hasil</p> <table border="0"> <tr> <td>Artikel</td><td>: 1403 artikel</td></tr> <tr> <td>Waktu</td><td>: 5239 ms</td></tr> <tr> <td>Derajat</td><td>: 3</td></tr> <tr> <td>Rute</td><td>: Indonesia → Geographic coordinate system → KStars → Operating system</td></tr> </table>	Artikel	: 1403 artikel	Waktu	: 5239 ms	Derajat	: 3	Rute	: Indonesia → Geographic coordinate system → KStars → Operating system						
Artikel	: 1403 artikel														
Waktu	: 5239 ms														
Derajat	: 3														
Rute	: Indonesia → Geographic coordinate system → KStars → Operating system														
16	<table border="0"> <tr> <td>Awal</td><td>: Indonesia</td></tr> <tr> <td>Tujuan</td><td>: Operating system</td></tr> <tr> <td>Algoritma</td><td>: IDS</td></tr> </table>  <p>Hasil</p> <table border="0"> <tr> <td>Artikel</td><td>: 1371 artikel</td></tr> <tr> <td>Waktu</td><td>: 8463 ms</td></tr> <tr> <td>Derajat</td><td>: 3</td></tr> <tr> <td>Rute</td><td>: Indonesia → Indonesian language → Writing system → Operating system</td></tr> </table>	Awal	: Indonesia	Tujuan	: Operating system	Algoritma	: IDS	Artikel	: 1371 artikel	Waktu	: 8463 ms	Derajat	: 3	Rute	: Indonesia → Indonesian language → Writing system → Operating system
Awal	: Indonesia														
Tujuan	: Operating system														
Algoritma	: IDS														
Artikel	: 1371 artikel														
Waktu	: 8463 ms														
Derajat	: 3														
Rute	: Indonesia → Indonesian language → Writing system → Operating system														

4.4. Analisis Hasil Pengujian

Berdasarkan hasil pengujian pada bagian 4.3., didapatkan data sebagai berikut untuk kasus yang sama dengan algoritma yang berbeda.

Tabel 4.4.1. Perbandingan Hasil Pengujian BFS vs IDS

Kasus	Waktu		Jumlah Artikel	
	Algoritma yang Lebih Cepat		Algoritma yang Lebih Sedikit Menelusuri	
	BFS	IDS	BFS	IDS
1 vs 2	✓		≡	≡
5 vs 6		✓		✓
9 vs 10	✓		≡	≡
11 vs 12	✓			✓
13 vs 14	✓		✓	
15 vs 16	✓			✓

Dari segi waktu eksekusi algoritma, didapatkan bahwa algoritma *BFS* menemukan solusi lebih cepat dibanding algoritma *IDS* pada 5 dari 6 kasus. Jika dianalisis secara algoritmik, algoritma *IDS* akan memakan waktu yang lebih lama dalam pencarian solusi karena banyaknya pengulangan (iterasi) yang perlu dilakukan.

Dari segi jumlah artikel yang ditelusuri, algoritma *IDS* umumnya menelusuri lebih sedikit artikel dibanding *BFS*. Algoritma *BFS* akan melewati *hyperlink* yang sudah pernah dikunjungi, sehingga jumlah artikel unik yang ditelusuri akan lebih banyak.

Secara garis besar, hasil yang didapatkan algoritma *BFS* dan *IDS* tidak berbeda jauh. Pada hasil-hasil pengujian yang sudah dilakukan, selisih waktu maupun jumlah artikel antar algoritma *BFS* dan *IDS* tergolong cukup kecil. Secara teori, kompleksitas waktu algoritma *BFS* sama dengan kompleksitas waktu algoritma *IDS* dan kedua algoritma hanya berbeda di kompleksitas ruang sehingga hasil yang didapatkan pada pengujian ini masih cukup logis.

Bab V

Kesimpulan

5.1. Kesimpulan

Baik algoritma *BFS* maupun algoritma *IDS* dapat digunakan untuk mencari rute halaman-halaman Wikipedia yang menghubungkan dua halaman Wikipedia tertentu. Algoritma *BFS* lebih efisien secara waktu dibanding *IDS*. Algoritma *IDS* lebih efisien secara ruang dibanding *BFS*.

5.2. Saran

Program berbasis *website* ini diharapkan bisa menjadi salah satu alternatif pencarian rute antara dua buah halaman Wikipedia. Untuk meningkatkan efisiensi waktu, pencarian secara dua arah (dari asal dan tujuan) disarankan untuk diterapkan. Tombol ‘save cache’ dapat dibuat lebih sederhana agar mempermudah pengguna dalam menggunakan program.

Lampiran

Link repository GitHub:

https://github.com/miannetopokki/Tubes2_Go-Jo-Golang-Enjoyers.git

Daftar Pustaka

Kauradmission. 2021. “5 Manfaat Belajar tentang Pengembangan Website” di <https://pmb.ittelkom-pwt.ac.id/5-manfaat-belajar-tentang-pengembangan-website/> (diakses 26 April 2024).

Munir, Rinaldi. 2024. “Breadth/Depth First Search (BFS/DFS) (Bagian 1)” di <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf> (diakses 26 April 2024).

Munir, Rinaldi. 2024. “Breadth/Depth First Search (BFS/DFS) (Bagian 2)” di <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf> (diakses 26 April 2024).