

# LAPORAN TUGAS BESAR

## IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Disusun oleh:

Dewantoro Triatmojo 13522011

Emery Fathan Zwageri 13522079

Rayhan Fadhlwan Azka 13522095

**Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2024**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	
<b>DESKRIPSI TUGAS</b>	<b>4</b>
<b>BAB II</b>	
<b>LANDASAN TEORI</b>	<b>5</b>
2.1 Graf	5
2.2 Penjelajahan Graf	5
2.3 Breadth First Search (BFS)	5
2.4 Iterative Deepening Search (IDS)	6
2.5 Depth First search (DFS)	6
2.6 Aplikasi Web	7
2.7 Front End	8
2.8 Back End	9
2.9 Komunikasi Aplikasi Web	9
<b>BAB III</b>	
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>10</b>
3.1 Langkah Langkah Pemecahan Masalah	10
3.2 Proses Pemetaan Masalah Elemen Elemen Algoritma BFS dan IDS	11
3.2.1 BFS	11
3.2.2 IDS	11
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun	12
3.3.1 Arsitektur Aplikasi Web	12
3.3.2 Fitur Front-End	12
3.3.1.1 Home Page	13
3.3.1.2 About Page	13
3.3.1.3 Play Page	14
3.3.3 Fitur Back-End	17
3.3.1.1 Middleware CORS	17
3.3.1.2 Caching	18
3.3.1.3 Endpoint /play	20
3.4 Contoh Ilustrasi Kasus	22
<b>BAB IV</b>	
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>25</b>
4.1 Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).	25
4.1.1 BFS	25
4.1.2 IDS	33
4.1.3 Multithreading	36
4.2 Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).	38

4.2.1 Setup Back-End	38
4.2.2 Setup Front-End	39
4.2.3 Menggunakan Aplikasi Web	40
4.3 Hasil pengujian (Screenshot antarmuka dan skenario yang memperlihatkan berbagai kasus yang mencakup seluruh fitur pada aplikasi Anda).	43
4.4 Analisis hasil pengujian	46
<b>BAB V</b>	
<b>KESIMPULAN DAN SARAN</b>	<b>48</b>
5.1 Kesimpulan	48
5.2 Saran	48
<b>LAMPIRAN</b>	<b>49</b>
<b>DAFTAR PUSTAKA</b>	<b>50</b>

# BAB I

## DESKRIPSI TUGAS

WikiRace atau Wiki Game adalah permainan yang melibatkan Wikipedia, sebuah ensiklopedia daring gratis yang dikelola oleh berbagai relawan di dunia, dimana pemain mulai pada suatu artikel Wikipedia dan harus menelusuri artikel-artikel lain pada Wikipedia (dengan mengeklik tautan di dalam setiap artikel) untuk menuju suatu artikel lain yang telah ditentukan sebelumnya dalam waktu paling singkat atau klik (artikel) paling sedikit.

Permasalahan permainan WikiRace dapat terjemahkan dalam bentuk permasalahan graf berarah. Setiap simpul dalam graf melambangkan suatu halaman wikipedia dan setiap sisi yang menghubungkan simpul A ke simpul B melambangkan bahwa ada terdapat hyperlink di halaman wikipedia A yang menuju ke halaman wikipedia B.

Permainan ini memiliki objektif untuk mencari lintasan artikel yang paling pendek antara dua artikel. Terdapat beberapa algoritma untuk mencari jarak terpendek antara dua titik pada sebuah graf. Pada tugas ini kami diperintahkan menggunakan algoritma *Breadth First Search* (BFS) dan *Iterative Deepening Search* (IDS) untuk mencari lintasan terpendek antara dua halaman wikipedia.

Dalam tugas besar kedua ini, kami diperintahkan untuk membuat sebuah aplikasi berbasis web untuk menyelesaikan permainan WikiRace. Aplikasi web yang dibuat harus memanfaatkan teknik *web scraping* untuk memperoleh data-data artikel wikipedia. Selain itu, back-end dari aplikasi web yang dibuat juga harus menggunakan bahasa pemrograman *Go*.

Front-end dari aplikasi web harus bisa menerima input data jenis algoritma, judul artikel awal, dan judul artikel tujuan. Kemudian, setelah input dikirim ke back-end dan diolah, Front-end harus bisa menampilkan jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms). Tak kalah penting, program juga harus dapat mencari rute terpendek dalam waktu kurang dari lima menit karena lebih dari itu `fetch()` akan timeout.

## BAB II

# LANDASAN TEORI

### 2.1 Graf

Sebuah graf dalam Computer science merupakan struktur data non linier yang terdiri dari simpul dan sisi yang menghubungkan 2 garis. Struktur data graf merupakan alat yang baik untuk merepresentasikan dan menganalisis hubungan kompleks antara objek dan entitas. Aplikasi graf berguna di banyak bidang seperti analisis jaringan sosial , rekomendasi sistem, dan jaringan komputer.

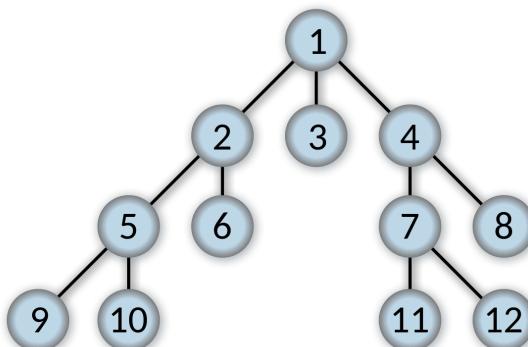
### 2.2 Penjelajahan Graf

Graf traversal adalah metode/algoritma yang digunakan untuk menjelajahi node pada graf. Aplikasi dari penjelajahan graf sendiri bisa berupa *connectivity check*, *finding cycles*, dan *bipartiness check*. Terkadang juga penjelajahan graf bisa mencari *shortest path* tetapi tidak di semua jenis graf.

### 2.3 Breadth First Search (BFS)

Algoritma Breadth First Search adalah algoritma pencarian melebar yang dilakukan dengan mengunjungi node pada level n terlebih dahulu sebelum mengunjungi node-node pada level n+1.

Seperti pada contoh dibawah ini, penjelajahan graf pada BFS dilakukan pada seluruh node pada level yang sama terlebih dahulu sebelum melanjutkan ke level selanjutnya.



Kompleksitas waktu dari BFS adalah  $O(b^d)$  dan kompleksitas ruang dari BFS adalah  $O(b^d)$ .

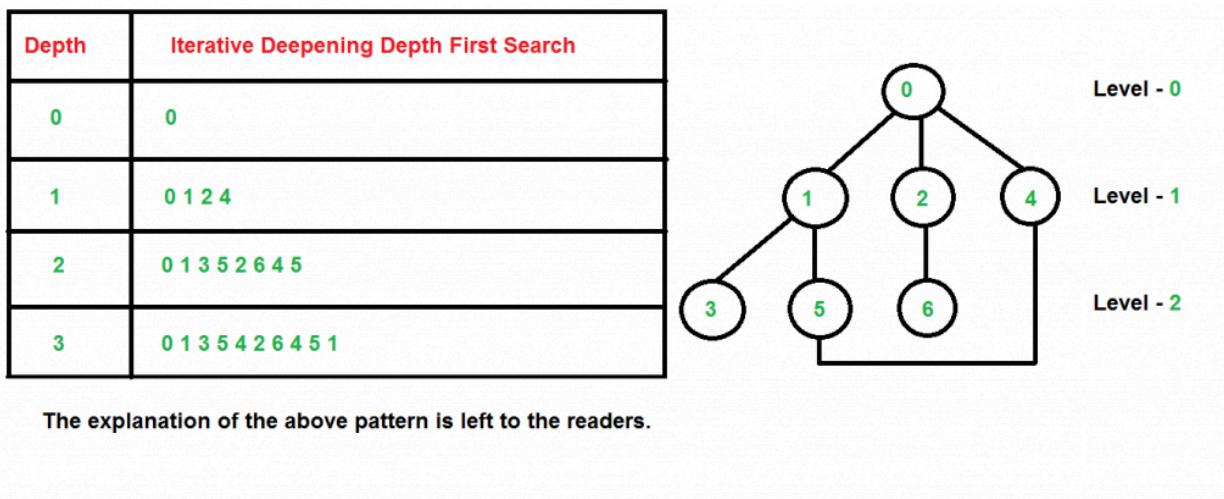
## 2.4 Iterative Deepening Search (IDS)

IDS adalah teknik yang digunakan untuk mengimplementasikan strategi *depth-limited search* (DLS) secara iteratif dengan batas kedalaman yang berbeda-beda, DLS sendiri cara kerjanya mirip seperti DFS namun dengan batasan pada kedalamannya.

Cara kerja IDS adalah dengan melakukan pencarian berulang-ulang dengan meningkatkan batas kedalaman pada setiap iterasi. Pencarian dimulai dengan batas kedalaman minimum, kemudian ditingkatkan satu per satu hingga solusi ditemukan atau batas maksimum kedalaman tercapai.

Keuntungan dari IDS adalah kemampuannya untuk menjelajahi ruang pencarian dengan cara yang efisien. Dengan menggunakan DLS secara iteratif, IDS dapat menemukan solusi dengan memperhitungkan kedalaman yang berbeda-beda tanpa harus melakukan pencarian ulang dari awal setiap kali batas kedalaman ditambahkan.

IDS sering digunakan dalam implementasi pencarian ruang keadaan pada masalah-masalah seperti pencarian jalur terpendek dalam graf, pencarian solusi dalam permainan, atau dalam berbagai aplikasi lain yang memerlukan pencarian solusi dalam ruang keadaan yang kompleks. Contoh penjelajahan dengan IDS adalah sebagai berikut



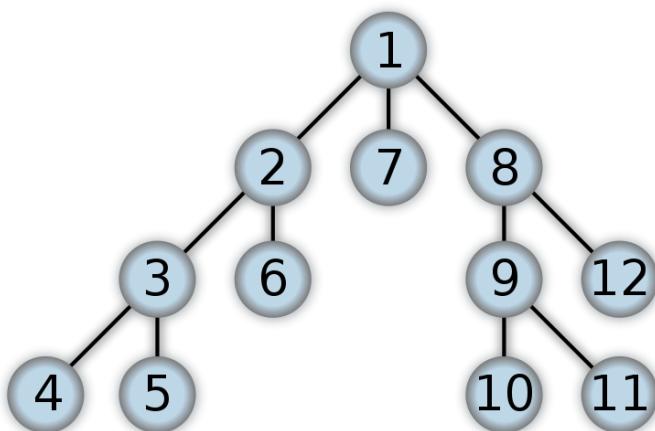
Pada contoh ini, penjelajahan dilakukan secara iteratif mulai dari level 0,1,2, dst.

## 2.5 Depth First search (DFS)

DFS adalah salah satu algoritma pencarian yang digunakan dalam kecerdasan buatan dan teori graf. DFS adalah metode pencarian yang melibatkan penjelajahan struktur data secara mendalam sebelum menelusuri jalur-jalur yang lain.

Cara kerja DFS adalah dengan memilih satu cabang dari pohon pencarian dan menelusurinya sejauh mungkin sebelum mundur dan mencoba cabang lain. Algoritma ini mengikuti prinsip “go deep before going wide” di mana ia mencoba menelusuri setiap cabang sejauh mungkin sebelum kembali ke cabang yang belum dieksplorasi.

DFS sering digunakan dalam berbagai aplikasi, termasuk pencarian jalur terpendek dalam graf, pemecahan masalah kecerdasan buatan, analisis struktur data seperti pohon, dan banyak lagi. Salah satu kelemahan DFS adalah jika digunakan pada graf yang sangat dalam atau memiliki sirkuit, algoritma tersebut dapat mengalami masalah seperti *stack overflow* atau menghabiskan waktu yang lama untuk menemukan solusi jika solusi berada di cabang yang terdalam.



## 2.6 Aplikasi Web

Aplikasi web (atau aplikasi web) adalah perangkat lunak aplikasi yang diakses menggunakan browser web. Aplikasi web dikirimkan di World Wide Web kepada pengguna dengan koneksi jaringan aktif. Sebelumnya pembuatan aplikasi web cukup rumit karena ada banyak hal-hal kecil yang harus diperhatikan. Sekarang, pembuatan aplikasi web disederhanakan dengan penggunaan *framework* aplikasi web. Kerangka kerja ini memfasilitasi pengembangan aplikasi yang cepat dengan menambahkan banyak berbagai lapisan abstraksi sehingga memungkinkan untuk tim pengembang fokus pada bagian-bagian aplikasi tanpa harus menyelesaikan masalah pengembangan umum.

Arsitektur web adalah struktur konseptual World Wide Web. WWW atau internet adalah media yang terus berubah yang memungkinkan komunikasi antara pengguna yang berbeda dan interaksi teknis (interoperabilitas) antara sistem dan subsistem yang berbeda. Dasarnya adalah berbagai komponen dan format data, yang biasanya disusun dalam tingkatan dan dibangun di atas satu sama lain. Secara keseluruhan, mereka membentuk infrastruktur internet, yang

dimungkinkan oleh tiga komponen inti yaitu protokol transmisi data (TCP/IP, HTTP, HTTPS), format representasi (HTML, CSS, XML), dan standar pengalamatan (URI, URL). Terdapat beberapa arsitektur web seperti “client-server model”, “three-tier model”, dan “Service-oriented architectures”.

Pada awalnya, web hanya terdiri dari arsitektur dua tingkat yaitu klien dan server. Klien dan server berbagi tugas dan layanan yang seharusnya dilakukan oleh sistem. Misalnya, klien mungkin meminta layanan dari server; server menjawab permintaan dengan menyediakan layanan. Mengambil situs web menggunakan alamat URL yang mengarahkan ke server untuk memuat situs di browser klien adalah contoh model dua lapis, juga dikenal sebagai model klien-server.

Model tiga tingkat mencakup logika aplikasi antara klien dan server, yang menangani pemrosesan data dan memungkinkan interaksi pada tingkat tertentu. Misalnya, server aplikasi dapat memproses data sementara server database didedikasikan hanya untuk penyimpanan data. Dengan cara ini, konten dapat dimuat dan disimpan secara dinamis. Bahasa skrip JavaScript sering kali bertanggung jawab atas perilaku klien.

Saat ini, web digunakan untuk jaringan struktur TI yang didistribusikan secara global. Setiap sistem TI dapat terdiri dari sub bagian yang komponennya saling terhubung melalui struktur atau arsitektur tetap. Pikirkan intranet dan perangkat lunak internal perusahaan. Aplikasi TI dan web modern jauh lebih kompleks daripada model klien-server. Layanan web terdistribusi, yang disusun sebagai arsitektur berorientasi layanan (SOA), menawarkan banyak fungsi dan unit fungsional modular yang dapat ditambahkan. Dengan SOA, proses bisnis dapat diotomatisasi dengan sistem yang terlibat berkomunikasi satu sama lain - sebagian tanpa campur tangan manusia - dan melakukan tugas tertentu. Contohnya termasuk perbankan online, e-commerce, e-learning, pasar online, dan aplikasi bisnis cerdas. Arsitektur ini tidak hanya jauh lebih kompleks tetapi juga dapat diperluas secara modular. Mereka dikenal sebagai arsitektur N-tier dan sejauh ini digunakan terutama dalam sektor bisnis.

Dalam tugas besar ini, kami hanya akan menggunakan arsitektur client-server model yaitu dengan membuat front end & back end saja.

## 2.7 Front End

Front end mengacu pada bagian dari sebuah aplikasi atau situs web yang diakses dan digunakan langsung oleh pengguna akhir. Ini termasuk segala yang terlihat dan dirasakan oleh pengguna, seperti antarmuka pengguna, layout, dan elemen desain. Front end seringkali melibatkan penggunaan bahasa pemrograman seperti HTML, CSS, dan JavaScript untuk membuat halaman web yang interaktif dan responsif.

Dalam tugas besar kali ini, kami menggunakan framework Next.js untuk membangun aplikasi web bagian front-end. Hal ini dikarenakan beberapa hal seperti kemudahan untuk develop dalam waktu cepat, terdapat banyak APInya yang bermanfaat, dan yang terakhir terdapat banyak library bagus yang dapat digunakan dengan Next.js (seperti UI Library).

## 2.8 Back End

Back end merujuk pada bagian dari aplikasi atau situs web yang tidak terlihat oleh pengguna tetapi mengelola logika bisnis, database, dan pemrosesan data. Ini melibatkan pengembangan server, database, dan logika aplikasi yang menjalankan proses di balik layar untuk memungkinkan fungsi yang kompleks dan interaksi dengan pengguna melalui front end.

Dalam tugas besar kedua ini, kami menggunakan framework Gin untuk membangun aplikasi web bagian back-end. Hal ini dikarenakan beberapa hal penting seperti performa, kemudahan untuk setup endpoint & middleware dalam waktu cepat, terdapat banyak API yang sudah disediakan oleh Gin sehingga kerja semakin cepat, dan terakhir banyak library & plugin yang dapat digunakan bersama dengan Gin.

## 2.9 Komunikasi Aplikasi Web

Front end dan Back end tidak tersambung secara langsung, oleh karena itu diperlukan sebuah mekanisme komunikasi antara Front end dan Back end. Dalam aplikasi web yang kami buat, kami menggunakan REST API.

REST (Representational State Transfer) adalah gaya arsitektur yang umum digunakan untuk merancang API. RESTful API, atau REST API, adalah API yang mengikuti prinsip-prinsip REST untuk mentransfer representasi sumber daya (resource) melalui protokol komunikasi standar, seperti HTTP. REST API menggunakan metode HTTP seperti GET, POST, PUT, DELETE untuk melakukan operasi pada sumber daya, dan sering menggunakan format data seperti JSON atau XML untuk pertukaran data. Ini memungkinkan berbagai aplikasi untuk berkomunikasi dengan satu sama lain secara efisien dan terstruktur melalui internet.

Pada back-end tersedia sebuah endpoint /play yang menerima POST request dimana front end bisa mengirim data input seperti jenis algoritma, artikel mulai wikipedia, dan artikel target wikipedia.

## **BAB III**

# **ANALISIS PEMECAHAN MASALAH**

### **3.1 Langkah Langkah Pemecahan Masalah**

Dalam Wikipedia dari satu page wiki bisa ke seluruh page yang ada di dalam page tersebut melalui link. Untuk mencari jalur terpendek antara dua link wikipedia melalui link link yang ada di dalamnya, kami menggambarkan suatu page sebagai node dan terhubung dengan link yang ada dalam page tersebut. Lalu dari suatu page, kita bisa pergi ke page yang linknya terdapat pada page tersebut. Namun dari page yang ada di dalam page tersebut belum tentu bisa mengunjungi page yang sebelumnya secara langsung, maka Kami menggambarkan masalah pada permainan wikiracer ini dengan struktur data graf berarah tak berbobot. Langkah-langkah dalam memecahkan masalah tersebut adalah sebagai berikut :

1. Menentukan page wikipedia yang menjadi page awal dan page wikipedia yang menjadi target. Lalu akan dicari jarak terpendek dari page awal ke page target tersebut.
2. Dalam mencari path atau jalur untuk mencari jarak terpendek kita dapat mengumpulkan seluruh link pada page wikipedia yang mengarah pada link wikipedia lain. Hal ini dilakukan dengan menggunakan kakas scraping go Colly, pengumpulan link-link ini bertujuan sebagai penunjuk dari node parent ke childnya, dengan parent adalah page wikipedia saat ini, dan childnya adalah link-link yang terdapat pada laman tersebut.
3. Pengumpulan link atau scraping dapat dilakukan dengan optimasi yaitu dengan menggunakan multithreading. Setelah diamati, pengumpulan link pada satu website wikipedia saja dapat memakan waktu 0,5 detik. Jika tidak melakukan multithreading, maka waktu yang akan dibutuhkan untuk mencari jalur dari page awal ke page target akan menjadi sangat lama. Saat kami melakukan percobaan mencari rute dari laman “Car” ke laman “Seat” pada wikipedia, jika tidak menggunakan multithreading, waktu yang dibutuhkan 10 menit, namun jika dilakukan multithreading rute yang sama dapat dicari dalam waktu kurang dari 20 detik.
4. Dalam menentukan link yang akan di visit, kami menggunakan algoritma IDS serta BFS. Link-link diantara page awal dan page target akan terus di visit sampai page target ditemukan.

## 3.2 Proses Pemetaan Masalah Elemen Elemen Algoritma BFS dan IDS

### 3.2.1 BFS

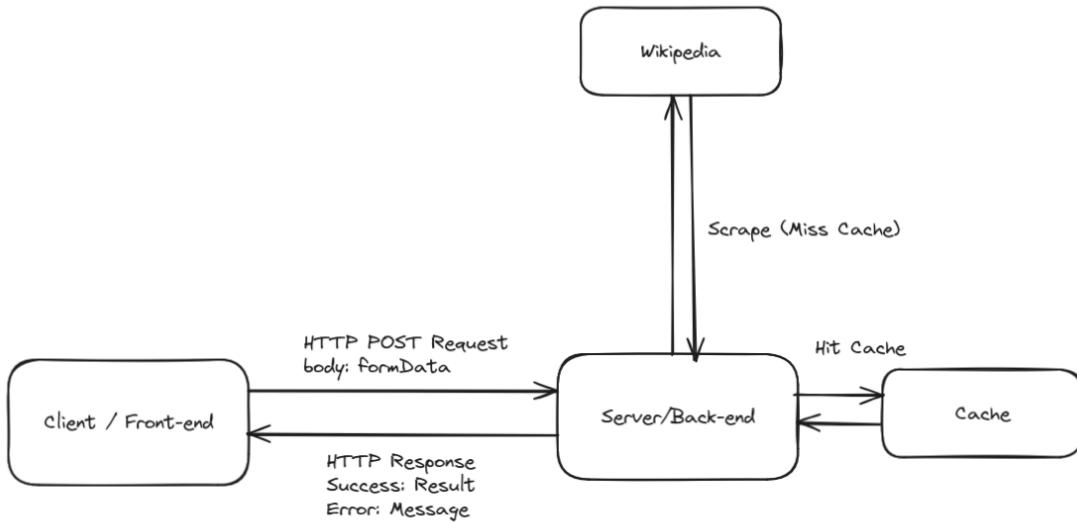
Pada implementasi BFS kali ini *Pages* dianggap sebagai *nodes*, lalu jarak dari node awal ke suatu node disimpan dalam *array* jarak yang awal mula nya diset *maxInt* selain *startURL* yang diset 0 . Graf direpresentasikan dalam bentuk *adjacency list* yang diimplementasikan dalam map yang key nya merupakan node dan valuenya adalah *array of string*. Total pengunjungan node disimpan dalam variabel *traversed*. Antrian node yang akan diproses disimpan dalam *queue* lalu solusi disimpan dalam solution tree dengan nama parent. Solution tree ini yang nantinya akan dikunjungi menggunakan DFS untuk menampilkan solusi.

### 3.2.2 IDS

Pada implementasi IDS, pages bisa dianggap sebagai nodes, dan link-link yang ada pada nodes adalah children dari page tersebut. Penelusuran secara IDS melakukan penelusuran secara bertahap dengan meningkatkan kedalaman pencarian pada setiap iterasinya. Selain itu pada IDS juga dilakukan pengecekan ketika link yang sudah ditelusuri pada iterasi sebelumnya, maka link tersebut tidak perlu di scrapping lagi, dan hanya melakukan load dari cache. Penelusuran dengan IDS memanfaatkan rekursi yang cara kerja rekursi sendiri menggunakan stack secara tidak langsung. Penelusuran rekursi ini akan berhenti ketika parameter IDS *startURL* = *endURL*, lalu path dari *startURL* sampai *endURL* akan di append ke *resultPath*.

## 3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

### 3.3.1 Arsitektur Aplikasi Web

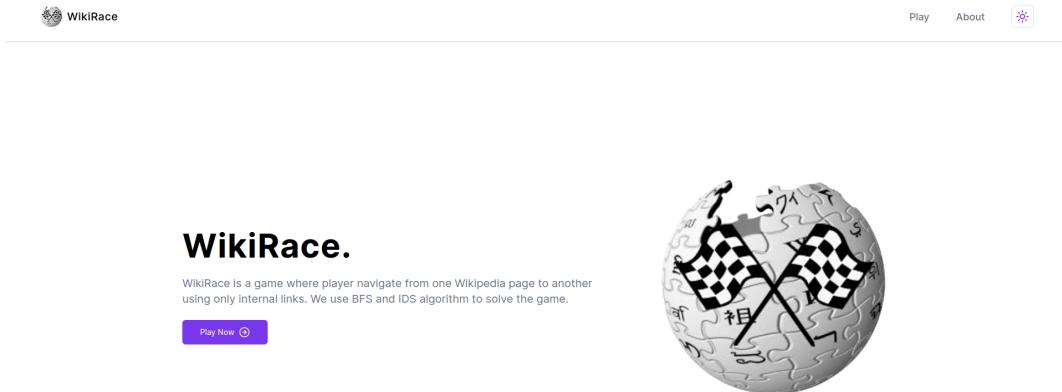


Aplikasi web penyelesaian permainan wikiracer menggunakan arsitektur client-server model. Client/front-end sebagai interface user untuk memilih nilai input permainan wiki racer. Server/back-end sebagai lingkungan untuk melakukan penyelesaian permainan wikiracer dan juga menyimpan cache.

### 3.3.2 Fitur Front-End

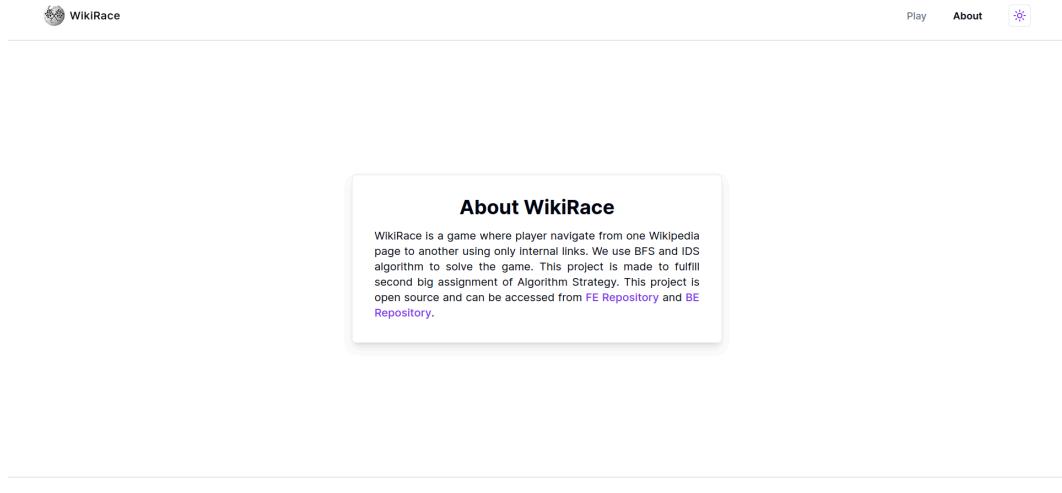
Implementasi front-end dari aplikasi web menggunakan framework [Next.js](#). Pembuatan UI menggunakan library [shadcn/ui](#) agar cepat dan mudah. Form state handling menggunakan library [React Hook Form](#). Query pada client menggunakan library [tanstack/react-query](#). Dalam aplikasi web yang dibuat, terdapat 3 halaman utama yaitu halaman home, halaman about, dan halaman play.

### 3.3.1.1 Home Page



Halaman home merupakan landing page ketika pengguna pertama kali mengunjungi website WikiRace. Halaman home berisi judul halaman beserta deskripsi singkat mengenai Wikirace. Pada halaman home page juga terdapat tombol menuju ke halaman play,

### 3.3.1.2 About Page



Halaman about merupakan halaman yang berisi informasi mengenai project ini mulai dari alasan dibuat beserta link repository front-end dan back-end dari project ini.

### 3.3.1.3 Play Page

The screenshot shows the 'WikiRace' play page. At the top, there's a navigation bar with a logo, 'WikiRace', 'Play', 'About', and a gear icon. The main area is titled 'WikiRace'. It has two dropdown menus: 'Algorithm' (set to 'Select algorithm') and 'Solution Option' (set to 'Select solution option'). Below these are input fields for 'Start' (labeled 'Insert start') and 'Target' (labeled 'Insert target'), each with a placeholder 'Wikipedia title of the starting page' or 'Wikipedia title of the target page'. A large purple button at the bottom right contains the text 'Find Shortest Path' with a gear icon.

Halaman play adalah halaman dimana pengguna dapat memainkan permainan WikiRace. Pertama pengguna memasukkan pilihan algoritma menggunakan dropdown select. Pengguna juga dapat memasukkan pilihan untuk mendapatkan single solution atau multiple solution. Setelah itu pengguna dapat memasukkan judul artikel wikipedia mulai pada input field start dan judul artikel wikipedia target pada input field target. Setelah itu user bisa mencari solusi dari permainan WikiRace dengan menekan tombol “Find Shortest Path”.

This screenshot shows the same 'WikiRace' play page as above, but with validation errors. The 'Algorithm' dropdown is empty ('Select algorithm'). The 'Solution Option' dropdown is also empty ('Select solution option'). The 'Start' input field is empty ('Insert start') and has a red error message 'Please enter a starting wikipedia page'. The 'Target' input field is empty ('Insert target') and has a red error message 'Please enter a target wikipedia page'. The large purple 'Find Shortest Path' button remains at the bottom.

Form juga menyediakan sebuah validasi yang dilakukan di front-end. Validasi pada front-end hanya mencakup form harus diisi (tidak boleh kosong). Label pada form field akan berubah warnanya menjadi merah ketika terdapat error.

Setelah validasi pada front end sukses, maka aplikasi web akan mengirimkan HTTP POST request ke endpoint backend /play dengan body formdata yang berisi data yang dimasukkan pengguna.

## WikiRace

Algorithm

BFS

Algorithm used to find the shortest wikipedia path from start to target.

Solution Option

Single

Get single or multiple path solutions

Start

Ini bukan artikel valid wkwkw

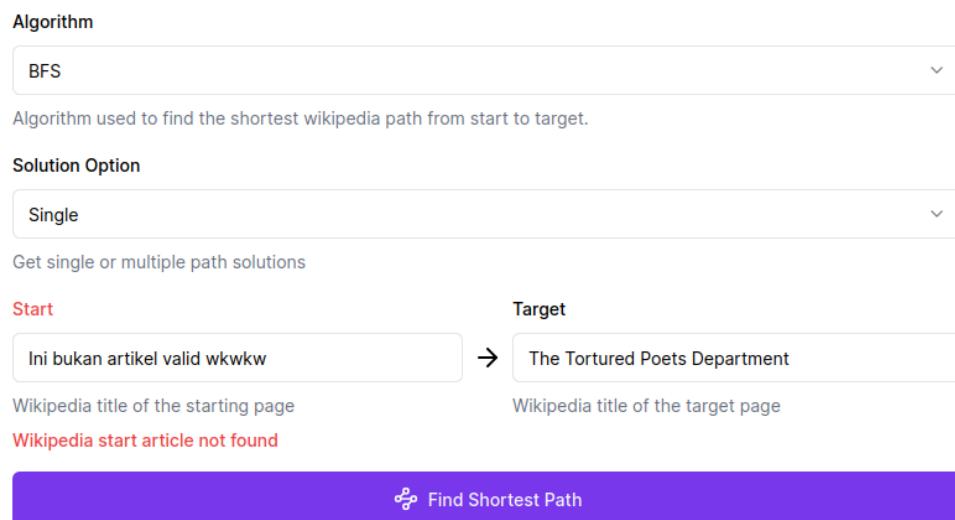
Wikipedia title of the starting page  
Wikipedia start article not found

Target

The Tortured Poets Department

Wikipedia title of the target page

 Find Shortest Path

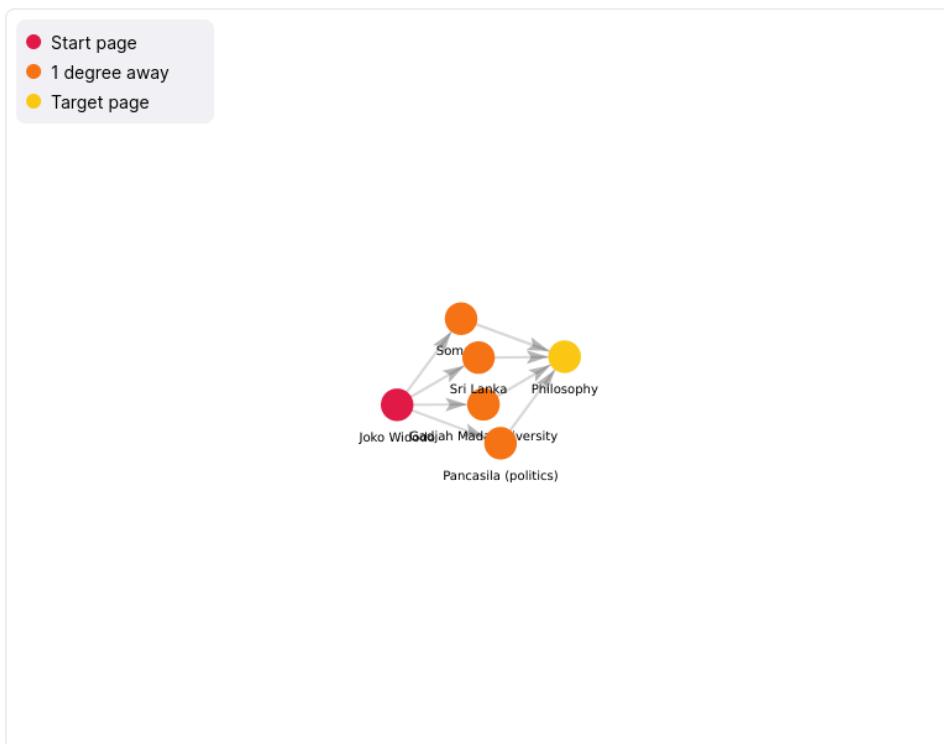


Backend juga menyediakan validasi. Back end akan memvalidasi apakah judul artikel wikipedia yang dikirim valid atau tidak, jika tidak akan mengirim error bad request dengan body berisi informasi judul dan deskripsi error serta form field mana yang error!

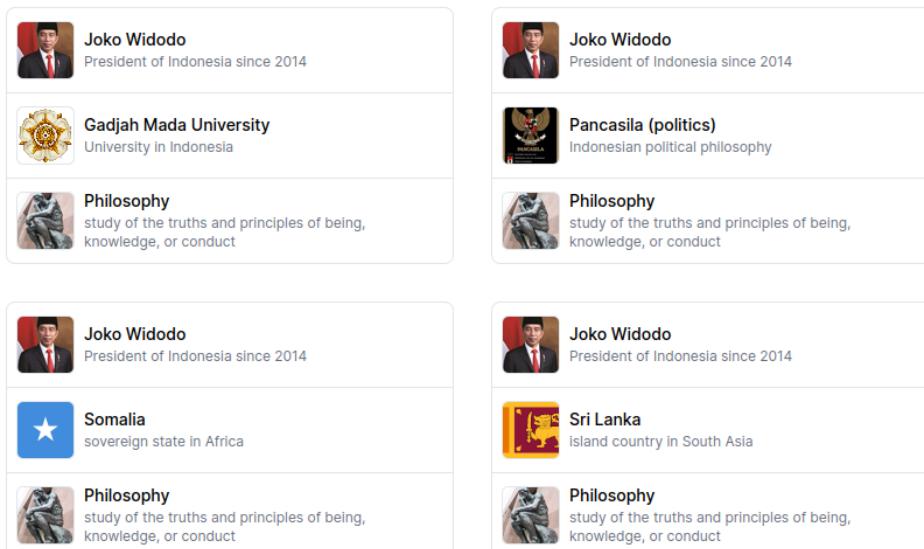
## Information

Traversed a total of **182850 articles** and found a total of **4** path solutions in **7.68 seconds** where the shortest path from **Joko Widodo** to **Philosophy** is **2 articles**

# Graph Visualization



## Individual Paths



Jika hasil pencarian sudah ketemu maka website akan mengeluarkan informasi total artikel yang dilalui, panjang lintasan terpendek, durasi pencarian, graf hasil pencarian, dan list semua solusi lintasan terpendek artikel.

### 3.3.3 Fitur Back-End

Implementasi back-end menggunakan bahasa [Go](#) dengan framework [Gin](#). Scraping artikel wikipedia dilakukan dengan memanfaatkan library [GoColly](#).

```
func RegisterRoutes() http.Handler {
    // Initialize gin router
    r := gin.Default()

    // CORS settings middleware
    r.Use(middleware.CORS())

    // Initialize database connection
    db.InitDB()

    // Endpoint for calculating wikirace shortest path
    r.POST("/play", controllers.PlayHandler)

    return r
}
```

Back-end hanya menerima satu endpoint, yaitu POST request di /play. Back-end juga menggunakan satu middleware, yaitu middleware untuk mengatur aturan CORS.

#### 3.3.1.1 Middleware CORS

```
func CORS() gin.HandlerFunc {
    return cors.New(cors.Config{
        AllowOrigins:   []string{os.Getenv("FE_URL")},
        AllowMethods:   []string{"GET", "POST", "PUT", "DELETE", "OPTIONS"},
        AllowHeaders:   []string{"Origin", "Content-Type", "Accept", "Authorization"},
        ExposeHeaders:  []string{"Content-Length"},
        AllowCredentials: true,
        MaxAge:         24 * 3600,
    })
}
```

Cross-origin resource sharing (CORS) merupakan mekanisme yang memungkinkan halaman web mengakses sumber daya tidak terbatas dari server di domain yang sama. Gambar diatas merupakan aturan CORS yang dipakai oleh back-end agar bisa menerima request dari front-end.

### 3.3.1.2 Caching

Caching dalam aplikasi web menyimpan data JSON yang berisi map of array dengan key URL suatu halaman wikipedia dan valuenya adalah array string yang berisi link link yang ada dalam halaman wikipedia tersebut. Cache akan coba untuk dimuat ketika backend dijalankan dan disimpan pada global variable Cache.

```
// Define global variable for storing cache
var Cache map[string][]string

func InitCache() {
    // Load the cache from the file
    fmt.Println("Reading cache, loading...")
    err := ReadMapFromFile("./cache/cache.json")

    if err != nil {
        fmt.Println("Error reading cache file")
        // Cache = make(map[string][]string)
        return
    }
    fmt.Println("Cache loaded")
}
```

```
func ReadMapFromFile(filename string) error {
    // Read the file
    data, err := os.ReadFile(filename)
    if err != nil {
        return err
    }

    // Unmarshal the JSON data into a map
    err = json.Unmarshal(data, &Cache)
    if err != nil {
        return err
    }

    return nil
}
```

Menulis global variable Cache pada cache, dapat menggunakan fungsi UpdateMapToFile().

```
func UpdateMapInFile(filename string) error {
    // Read the existing data from the file
    data, err := os.ReadFile(filename)
    if err != nil {
        return err
    }

    // Unmarshal the JSON data into a map
    err = json.Unmarshal(data, &Cache)
    if err != nil {
        return err
    }

    // Merge the existing map with the new map
    for key, value := range Cache {
        if _, exists := Cache[key]; !exists {
            Cache[key] = value
        }
    }

    // Convert the merged map to JSON
    jsonData, err := json.Marshal(Cache)
    if err != nil {
        return err
    }

    // Write the JSON data back to the file
    err = os.WriteFile(filename, jsonData, 0644)
    if err != nil {
        return err
    }

    return nil
}
```

### 3.3.1.3 Endpoint /play

```
// Route handler for /play
func PlayHandler(c *gin.Context) {
    // Validate request data
    var reqJSON PlayRequest
    err := c.ShouldBind(&reqJSON)
    if err != nil {
        c.JSON(400, gin.H{"error": "Bad Request", "message": err.Error()})
        return
    }

    // Get data
    algorithm := reqJSON.Algorithm
    startTitle := reqJSON.Start
    targetTitle := reqJSON.Target

    // Get start wikipedia URL (and validate title)
    startURL, err := utils.GetWikipediaURLFromTitle(startTitle)
    if err != nil {
        c.JSON(400, gin.H{"error": "Bad Request", "message": "Wikipedia start article not found", "errorFields": []FieldError{{"start", "Wikipedia start article not found"}}})
        return
    }
    // Get target wikipedia URL (and validate title)
    targetURL, err := utils.GetWikipediaURLFromTitle(targetTitle)
    if err != nil {
        c.JSON(400, gin.H{"error": "Bad Request", "message": "Wikipedia target article not found", "errorFields": []FieldError{{"target", "Wikipedia target article not found"}}})
        return
    }

    // Solve
    result, err := Solve(algorithm, startURL, targetURL)
    if err != nil {
        c.JSON(500, gin.H{"error": err.Error()})
        return
    }

    // Return the result
    c.JSON(200, result)
}
```

API Endpoint /play menerima request dengan metode POST sebagai endpoint untuk melakukan perhitungan pencarian lintasan terpendek dengan struktur data & validasi parsing sebagai berikut.

```
// Result Request Data Structure
type PlayRequest struct {
    Algorithm string `form:"algorithm" binding:"required,oneof='IDS' 'BFS'"` // Algorithm type (IDS or BFS)
    Start     string `form:"start" binding:"required"`                      // Start wikipedia article title
    Target    string `form:"target" binding:"required"`                     // Target wikipedia article title
}
```

Endpoint /play bisa mengirimkan response Error 400 (Bad Request) atau Error 500 (Internal Server Error) dengan struktur data respon (JSON) sebagai berikut.

```
type FieldError struct {
    Field   string `json:"field"`      // Form field that caused the error
    Message string `json:"message"`    // Error message
}

// Error Response Data Structure
type PlayErrorResponse struct {
    Error     string     `json:"error"`      // Error message
    Message   string     `json:"message"`    // Error message
    ErrorFields []FieldError `json:"errorFields"` // List of fields that caused the error
}
```

Endpoint /play juga bisa mengirimkan response Success (200) dengan struktur data respon (JSON) sebagai berikut.

```
// Result Response Data Structure
type PlaySuccessResponse struct {
    TotalTraversed int           `json:"totalTraversed"` // Total article traversed
    ShortestPathLength int         `json:"shortestPathLength"` // Shortest path length
    Duration float32              `json:"duration"` // Duration of the search
    Articles []models.Article     `json:"articles"` // Articles data used in paths: Path[]
    Paths    []models.Path         `json:"paths"` // List of the shortest paths from start to target found
}
```

Endpoint /play menerima body yang berisi formdata yang akan di parse dengan tiga key yaitu algorithm, start, dan target. Field algorithm memiliki dua validasi, yaitu required dan harus memiliki value IDS atau BFS. Field start dan target hanya memiliki satu validasi yaitu required. Jika validasi dari request body tidak terpenuhi maka back-end akan mengirimkan Error 400: Bad Request.

```
// Get Wikipedia URL from title.
//
// Returns the wikipedia URL of the article with the given title.
//
// Returns error if the title is invalid or the article is not found.
func GetWikipediaURLFromTitle(title string) (string, error) {
    // Validate title
    res, err := http.Get("https://en.wikipedia.org/wiki/" + title)

    // Http protocol error / too many redirects
    if err != nil {
        return "", err
    }

    // Article not found
    if res.StatusCode == 404 {
        return "", fmt.Errorf("start article not found")
    }

    return res.Request.URL.String(), nil
}
```

Validasi saat parsing form data saja tidak cukup. Start title dan target title juga masih harus divalidasi apakah title dari wikipedia yang ada di form data request merupakan artikel wikipedia dengan judul yang valid atau tidak. Untuk memvalidasi ini, back-end akan melakukan GET request pada artikel yang ingin diuji (tujuan tertulis pada gambar diatas). Jika respon dari GET request tersebut adalah 404 artinya article tidak ketemu dan fungsi akan mengembalikan

error. Selain itu, program akan mengirimkan URL wikipedia yang sudah di parse (spasi, dan karakter-karakter lain).

Setelah semua validasi selesai, PlayHandler akan menghitung solusi permainan wikirace dari data input yang sudah tervalidasi. Jika perhitungan terjadi error, maka back-end akan mengirim Error 500: Internal Server Error. Jika perhitungan berhasil, maka back-end akan mengirimkan data hasil solusi permainan wikirace.

### 3.4 Contoh Ilustrasi Kasus

1. Kasus Inauguration of Joko Widodo ke Indonesia dengan IDS multiple path dan single path

The screenshot shows the WikiRace application interface with two search results side-by-side. Both results use the 'IDS' algorithm and have 'Multiple' solution options selected. The 'Start' field contains 'Inauguration of Joko Widodo' and the 'Target' field contains 'Indonesia'. The first result, under 'Information', states: 'Traversed a total of 76354 articles and found a total of 137 path solutions in 9.92 seconds where the shortest path from Inauguration of Joko Widodo to Indonesia is 3 articles'. The second result, under 'Information', states: 'Traversed a total of 460 articles and found a total of 1 path solutions in 8.41 seconds where the shortest path from Inauguration of Joko Widodo to Indonesia is 3 articles'. Both results have a purple 'Find Shortest Path' button at the bottom.

2. Kasus Iowa ke Guitar dengan BFS single path dan BFS multiple path

## WikiRace

**Algorithm**

BFS

Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**

Single

Get single or multiple path solutions

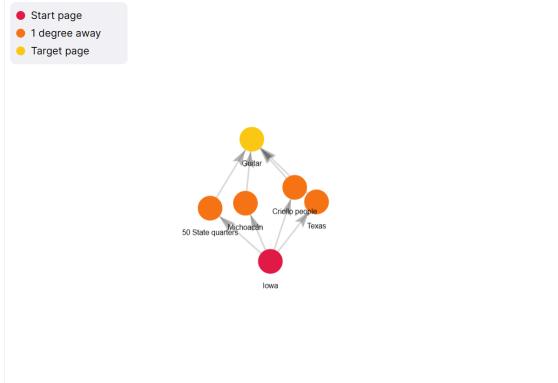
**Start** Iowa **Target** Guitar

Wikipedia title of the starting page Wikipedia title of the target page

**Find Shortest Path**

Traversed a total of **229645 articles** and found a total of **4** path solutions in **48.24 seconds** where the shortest path from **Iowa** to **Guitar** is **2 articles**

### Graph Visualization



### Information

Traversed a total of **396383 articles** and found a total of **1** path solutions in **30.08 seconds** where the shortest path from **Iowa** to **Guitar** is **2 articles**

## 3. Kasus Duck ke Chicken dengan BFS Multiple path dan IDS multiple path

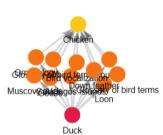
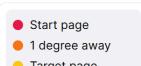
### Information

Traversed a total of **38784 articles** and found a total of **10** path solutions in **4.59 seconds** where the shortest path from **Duck** to **Chicken** is **2 articles**

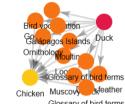
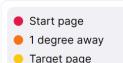
### Information

Traversed a total of **61014 articles** and found a total of **10** path solutions in **5.13 seconds** where the shortest path from **Duck** to **Chicken** is **2 articles**

### Graph Visualization



### Graph Visualization



#### 4. Kasus Joko Widodo ke Philosophy dengan BFS single path dan IDS Single path

**Algorithm**

BFS

Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**

Single

Get single or multiple path solutions

Start	Target
Joko Widodo	→ Philosophy
Wikipedia title of the starting page	
Wikipedia title of the target page	

Find Shortest Path

-----

**Algorithm**

IDS

Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**

Single

Get single or multiple path solutions

Start	Target
Joko Widodo	→ Philosophy
Wikipedia title of the starting page	
Wikipedia title of the target page	

Find Shortest Path

#### Information

Traversed a total of **6895 articles** and found a total of **1** path solutions  
in **9.69 seconds** where the shortest path from **Joko Widodo** to  
**Philosophy** is **2 articles**

#### Information

Traversed a total of **14315 articles** and found a total of **1** path solutions  
in **10.44 seconds** where the shortest path from **Joko Widodo** to  
**Philosophy** is **2 articles**

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1 Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).**

##### **4.1.1 BFS**

###### **A. Fungsi BFS**

Menerima parameter startURL,targetURL, dan mengembalikan nilai array of array of string(solusi) dan total pengunjungan (integer). Kami disini membuat dua fungsi bfs, multiple dan single. Single untuk menyelesaikan singlepath solution dan multi untuk menyelesaikan multiple path solution. Bedanya adalah untuk bfs single jika sudah menemukan target url pencarian langsung selesai saat itu juga. Tidak perlu memroses queue.

Penjelasan: Awal mula traversed diset 1 karena awalnya pasti mengunjungi diri sendiri. Lalu jika startURL sama dengan targetURL maka program akan langsung mengembalikan lintasan solusi berisi hanya startURL dan total traversed. Lalu disini kami menggunakan multithreading. Thread di set ke 300 (tidak boleh terlalu banyak agar tidak diban wikipedia). Awal mulanya distance startURL diset 0 dan parent startURL nil, lalu queue diinisiasi dengan size maximal integer yaitu  $2^{31}-1$ . Lalu distance targetURL diset maximal integer. Setelah itu masukan startURL ke dalam queue setelah itu ada while loop yang berhenti saat queue telah diproses semua. Jadi pada program kami setiap 1 pengulangan while kami memroses satu depth node. *Scraping* dilakukan secara paralel karena jika tidak parallel akan sangat lama, saat discrap maka URL yang didapatkan akan dianggap sebagai tetangga dari node yang diproses. Node yang diproses hanyalah node distance dari startURL nya lebih kecil dari targetURL. Setelah itu barulah kita memroses isi queue dan mengecek mengunjungi node yang bertetangga dengannya. Jika distancenya lebih besar dari distance node yang diproses ditambah 1 artinya node tersebut belum diproses lalu node tersebut kita masukan ke queue dan parent dari node tersebut kita tambahkan isinya dengan node yang kita proses sekarang. Jika distance node tetangga sama dengan node yang diproses ditambah 1 maka node yang diproses sudah pernah dikunjungi dan parent dari node tetangga ditambah isinya dengan node yang diproses. Kami juga melakukan prunning jika pada kedalaman ke 1 telah menemukan solusi maka langsung saja diberhentikan prosesnya karena solusinya sudah pasti unik. Selanjutnya pembatasan juga dilakukan untuk

bfsSingle jika menemukan targetURL. Setelah selesai memproses semua yang ada di dalam queue maka queue digantikan dengan node untuk layer selanjutnya.

Setelah while loop selesai terbentuk lah solution tree yang merupakan jalur lintasan dari setiap node menuju startURL, Lalu kami menggunakan DFS untuk mengubah solusi dari tree tersebut menjadi array of array solution. Karena dari tree yang terbalik maka kita akan reverse isi arraynya. Lalu kembalikan matrix solusi dan total pengunjungan

Penjelasan tambahan:

Dist di set maxInt artinya belum pernah dikunjungi. Kami tidak menggunakan visited karena lebih fleksibel sekalian untuk mengukur shortest path.

```

1  func BFSMulti(startURL string, targetURL string) ([][]string, int) {
2      fmt.Println("Solving with BFS")
3      fmt.Println("Start URL:", startURL)
4      fmt.Println("Target URL:", targetURL)
5
6      traversed := 1
7
8      if startURL == targetURL {
9          return [][]string{{startURL}}, 1
10     }
11
12     cache := make(map[string][]string)
13     cache, err := readMapFromFile("./cache/cache.json")
14
15     // cache, err := readMapFromFile("../cache/cache.json")
16
17     if err != nil {
18         fmt.Println("error reading cache file")
19         cache = make(map[string][]string)
20     }
21
22
23     gm := NewGoRoutineManager(maxConcurrentBFS)
24     maxInt := math.MaxInt32
25     adj := make(map[string][]string)
26     parent := make(map[string][]string)
27     parent[startURL] = nil
28     q := Queue{Size: maxInt}
29     dist := make(map[string]int)
30     dist[startURL] = 0
31     dist[targetURL] = maxInt
32     q.Enqueue(startURL)
33
34     for !q.IsEmpty() {
35         var isFirst bool
36         isFirst = false
37         for i := 0; i < q.GetLength(); i++ {
38             i := i
39             gm.Run(func() {
40                 check := dist[q.Elements[i]] < dist[targetURL]
41                 // mu.Unlock()
42                 if check {
43                     rwmu.RLock()
44                     cacheCheck := cache[q.Elements[i]]
45                     rwmu.RUnlock()
46                     if cacheCheck == nil {
47                         links := utils.GetAllInternalLinks(q.Elements[i])
48                         rwmu.Lock()
49                         adj[q.Elements[i]] = links
50                         cache[q.Elements[i]] = links
51                         rwmu.Unlock()
52                     } else {
53                         rwmu.Lock()
54                         adj[q.Elements[i]] = cacheCheck
55                         rwmu.Unlock()
56                     }
57                     fmt.Println(q.Elements[i], "dist: ", dist[q.Elements[i]], "target URL: ", dist[targetURL])
58
59                 }
60             })
61         }
62     }
63 }
```

```

● ● ●
1  wg.Wait()
2      // length := q.GetLength()
3      for i := 0; i < length; i++ {
4          u := q.Elements[i]
5          if dist[u] >= dist[targetURL] {
6              continue
7          }
8          for _, v := range adj[u] {
9              if v != startURL && dist[v] == 0 {
10                  dist[v] = maxInt
11              }
12          }
13          for i := 0; i < len(adj[u]); i++ {
14              traversed++
15
16              if dist[adj[u][i]] > dist[u]+1 {
17                  dist[adj[u][i]] = dist[u] + 1
18                  q.Enqueue(adj[u][i])
19                  parent[adj[u][i]] = nil
20                  parent[adj[u][i]] = append(parent[adj[u][i]], u)
21              }
22
23              if dist[targetURL] != maxInt {
24                  isFound = true
25                  break
26              }
27          }
28          if isFound {
29              break
30          }
31      }
32      if(isFound){
33          break
34      }
35  q.Elements = q.Elements[length:]
36
37 }
38
39 paths := make([][]string, 0)
40 path := make([]string, 0)
41
42 paths = dfs(paths, path, parent, targetURL)
43
44 for i := 0; i < len(paths); i++ {
45     paths[i] = reverse(paths[i])
46 }
47 updateMapInFile(cache, "./cache/cache.json")
48 // updateMapInFile(cache, "../cache/cache.json")
49 paths = paths[:1]
50 return paths, traversed
51
52 }

```

```

1  func BFSSingle(startURL string, targetURL string) ([][]string, int) {
2      fmt.Println("Solving with BFS")
3      fmt.Println("Start URL:", startURL)
4      fmt.Println("Target URL:", targetURL)
5      // runtime.GOMAXPROCS(runtime.NumCPU())
6      // var adj [][]int
7      traversed := 1
8
9      if startURL == targetURL {
10          return [][]string{{startURL}}, 1
11      }
12
13      cache := make(map[string][]string)
14      // cache, err := readMapFromFile("../cache/cache.json")
15      cache, err := readMapFromFile("./cache/cache.json")
16
17
18      if err != nil {
19          fmt.Println("error reading cache file")
20          cache = make(map[string][]string)
21      }
22
23      gm := NewGoRoutineManager(maxConcurrentBFS)
24      maxInt := math.MaxInt32
25      adj := make(map[string][]string)
26      parent := make(map[string][]string)
27
28      parent[startURL] = nil
29      q := Queue{Size: maxInt}
30      dist := make(map[string]int)
31      dist[startURL] = 0
32      dist[targetURL] = maxInt
33      q.Enqueue(startURL)
34      isFound := false
35      for !q.IsEmpty() {
36          length := min(q.GetLength(), 200)
37          for i := 0; i < length; i++ {
38              i := i
39              gm.Run(func() {
40                  check := dist[q.Elements[i]] < dist[targetURL]
41                  // mu.Unlock()
42                  if check {
43                      rwmu.RLock()
44                      cacheCheck := cache[q.Elements[i]]
45                      rwmu.RUnlock()
46                      if cacheCheck == nil {
47                          links := utils.GetAllInternalLinks(q.Elements[i])
48                          rwmu.Lock()
49                          cache[q.Elements[i]] = links
50                          adj[q.Elements[i]] = links
51                          rwmu.Unlock()
52                      } else {
53                          rwmu.Lock()
54                          adj[q.Elements[i]] = cacheCheck
55                          rwmu.Unlock()
56                      }
57                      fmt.Println(q.Elements[i], "dist: ", dist[q.Elements[i]], "target URL: ", dist[targetURL])
58                  }
59              })
60          }
61      }
62      wg.Wait()

```



```
1  for i := 0; i < length; i++ {
2      u := q.Elements[i]
3      if dist[u] >= dist[targetURL] {
4          continue
5      }
6      for _, v := range adj[u] {
7          if v != startURL && dist[v] == 0 {
8              dist[v] = maxInt
9          }
10     }
11     for i := 0; i < len(adj[u]); i++ {
12         traversed++
13
14         if dist[adj[u][i]] > dist[u]+1 {
15             dist[adj[u][i]] = dist[u] + 1
16             q.Enqueue(adj[u][i])
17             parent[adj[u][i]] = nil
18             parent[adj[u][i]] = append(parent[adj[u][i]], u)
19         }
20
21         if dist[targetURL] != maxInt {
22             isFound = true
23             break
24         }
25     }
26     if isFound {
27         break
28     }
29
30 }
31 if(isFound){
32     break
33 }
34 q.Elements = q.Elements[length:]
35 }
36
37 paths := make([][]string, 0)
38 path := make([]string, 0)
39
40 paths = dfs(paths, path, parent, targetURL)
41
42 for i := 0; i < len(paths); i++ {
43     paths[i] = reverse(paths[i])
44 }
45 updateMapInFile(cache, "./cache/cache.json")
46 // updateMapInFile(cache, "../cache/cache.json")
47 paths = paths[:1]
48 return paths, traversed
49
50 }
```

## Struktur Data dan variabel pada BFS

Struktur Data/Variabel	Penjelasan
<code>adj := make(map[string][]string)</code>	Adjacency list yang memetakan node pada neighboursnya
<code>parent := make(map[string][]string)</code>	Menyimpan path untuk solusi
<code>q := Queue{Size: maxInt}</code>	Antrian node untuk diproses
<code>traversed := 1</code>	Total pengunjungan diinisiasi 1 karena sudah pasti mengunjungi node awal
<code>paths := make([][]string, 0)</code>	Matrix string yang digunakan untuk menyimpan solusi
<code>dist := make(map[string]int)</code>	Map yang memetakan jarak dari sebuah key atau node ke start url
<code>path := make([]string, 0)</code>	Sebuah dump variabel untuk rekursif mengisi solusi

## Struktur data dan method pada queue

Struktur Data dan method	penjelasan/Implementasi
<pre>type Queue struct {     Elements []string     Size     int }</pre>	Queue memiliki variabel element yang merupakan array of string dan size yang merepresentasikan ukurannya.
<pre>func (q *Queue) Enqueue(elem string)</pre>	<pre>if q.GetLength() == q.Size {     fmt.Println("Overflow")     return } q.Elements = append(q.Elements, elem)</pre>

func (q *Queue) Dequeue() string	<pre>if q.IsEmpty() {     fmt.Println("UnderFlow")     return "" }  element := q.Elements[0] if q.GetLength() == 1 {     q.Elements = nil     return element }  q.Elements = q.Elements[1:] return element // Slice off the element once it is dequeued.</pre>
func (q *Queue) GetLength() int	return len(q.Elements)
func (q *Queue) IsEmpty() bool	return len(q.Elements) == 0
func (q *Queue) Peek() (string, error)	<pre>if q.IsEmpty() {     return "", errors.New("empty queue") }  return q.Elements[0], nil</pre>

## B. Fungsi DFS

Fungsi DFS menerima parameter array of array of string paths, array of string path, map from string to array of string dan string. DFS mengembalikan array of array of string. Fungsi ini bekerja secara rekursif untuk mentraverse parent dari targetURL yang ketika mengunjungi parent yang paling tinggi yaitu startURL program akan mereturn lintasan yang ada.

Fungsi ini dibuat untuk membantu *printing solution dari parent solution* pada BFS. Alasan penggunaan DFS untuk printing solution adalah agar membuat solusi yang lebih elegan terutama untuk multiple solution.

```

1 func dfs(paths [][]string, path []string, parent map[string][]string, end string) [][]string {
2     if parent[end] == nil {
3         path = append(path, end)
4         // *path = append(*path, end)
5
6         paths = append(paths, path)
7         // *paths = append(*paths, *path)
8         path = path[:len(path)-1]
9         // *path = (*path)[:len(*path)-1]
10    } else {
11        for i := 0; i < len(parent[end]); i++ {
12            path = append(path, end)
13            // *path = append(*path, end)
14            paths = dfs(paths, path, parent, parent[end][i])
15            path = path[:len(path)-1]
16            // *path = (*path)[:len(*path)-1]
17        }
18    }
19    return paths
20 }

```

### 4.1.2 IDS

Cara kerja fungsi IDS adalah dengan cara melakukan iterasi mulai dari depth 1, yaitu mencari child dari startURL. Lalu, IDS akan memanggil fungsi DLS (Depth Limited Search), yaitu fungsi penjelajahan yang cara kerjanya mirip seperti DFS, tetapi dengan batasan kedalaman. Pada fungsi DLS, pertama akan ditambahkan iterasi jumlah traversed, lalu dicek apakah parameter startURL = endURL, jika startURL = endURL, maka site tersebut adalah site yang dituju, dan program akan meng append path dari startURL sampai endURL, ke resultPath dan melakukan return,. atau jika depth == 0 maka program akan melakukan return. Lalu DLS akan mengecek, apakah link tersebut pernah di scrape pada IDS iterasi sebelumnya, make child dari link tersebut akan di load dari cache, jika tidak, maka akan dilakukan scrape dan childnya disimpan ke cache. Setelah itu, untuk setiap link pada website, akan dijalankan DLS lagi secara rekursi dengan depthnya dikurangi 2. Dalam program IDS ini, kami tidak mengecek visited, tetapi mengecek pada cache. Hal ini dikarenakan jika menggunakan multipath, maka ada kemungkinan node yang sama bisa divisit lebih dari sekali pada jalur yang berbeda.

Perbedaan fungsi IDS untuk single path dan multi path hanya terdapat pada bagian awal

DLS, untuk single path seluruh DLS yang sedang berjalan akan me return ketika sudah ditemukan minimal 1 path yang terhubung dari startURL ke endURL.



```
1 func IDS(startURL string, targetURL string, isSingle bool) ([][]string, int32) {
2     targetFound = 0
3     resultPath := make([][]string, 0)
4
5     path := make([]string, 0)
6     var totalTraversed int32 = 0
7     gm := NewGoRoutineManager(maxConcurrent)
8
9     depth := 1
10    for {
11        fmt.Println("====")
12        fmt.Println("depth : ", depth)
13        fmt.Println("====")
14        // wg.Add(1)
15        if isSingle {
16            DLSSingle(startURL, targetURL, path, &resultPath, depth, gm, &totalTraversed)
17        } else {
18            DLS(startURL, targetURL, path, &resultPath, depth, gm, &totalTraversed)
19        }
20
21        wg.Wait()
22        if len(resultPath) > 0 {
23            fmt.Println("found")
24
25            for i := range resultPath {
26                resultPath[i] = append([]string{startURL}, resultPath[i]...)
27            }
28
29            // cache.UpdateMapInFile("./cache/cache.json")
30            // updateMapInFile(cache, "../cache/cache.json")
31            return resultPath, totalTraversed
32        }
33
34    }
35
36    path = path[:0]
37    // if depth > 10 {
38    //     break
39    // }
40    depth++
41    totalTraversed = 0
42 }
43 return nil, 0
44
45 }
```

```
1  func DLS(startURL string, targetURL string, path []string, resultpath *[][]string, depth int, gm *goRoutineManager, totalTraversed *int32) {
2      atomic.AddInt32(totalTraversed, 1)
3      if startURL == targetURL {
4          mu.Lock()
5          *resultpath = append(*resultpath, path)
6          mu.Unlock()
7          return
8      }
9      if depth == 0 {
10         return
11     }
12     var links []string
13     if depth > 1 {
14         links = (cache.Cache)[startURL]
15     } else {
16         rwmu.RLock()
17         check := (cache.Cache)[startURL]
18         rwmu.RUnlock()
19         if check == nil {
20             links = utils.GetAllInternalLinks(startURL)
21             rwmu.Lock()
22             (cache.Cache)[startURL] = links
23             rwmu.Unlock()
24         } else {
25             links = check
26         }
27     }
28     fmt.Println("current processed : ", startURL)
29     for _, link := range links {
30         currpath := append(path, link)
31         // capture the link so each goroutine is unique
32         link := link
33         gm.Run(func() {
34             DLS(link, targetURL, currpath, resultpath, depth-1, gm, totalTraversed)
35         })
36     }
37 }
38 }
```

```

1  func DLSSingle(startURL string, targetURL string, path []string, resultpath *[][]string, depth int, gm *goRoutineManager, totalTraversed *int32) {
2
3     if atomic.LoadInt32(&targetFound) != 0 {
4         return
5     }
6     atomic.AddInt32(totalTraversed, 1)
7     if startURL == targetURL {
8         mu.Lock()
9         atomic.StoreInt32(&targetFound, 1)
10        *resultpath = append(*resultpath, path)
11        fmt.Println(targetFound)
12        mu.Unlock()
13    }
14
15    return
16 }
17 if depth == 0 {
18     return
19 }
20
21 var links []string
22 if depth > 1 {
23
24     links = (cache.Cache)[startURL]
25
26 } else {
27     rwmu.RLock()
28     check := (cache.Cache)[startURL]
29     rwmu.RUnlock()
30     if check == nil {
31
32         links = utils.GetAllInternalLinks(startURL)
33         fmt.Println("links : ")
34         rwmu.Lock()
35         (cache.Cache)[startURL] = links
36         rwmu.Unlock()
37     } else {
38         links = check
39     }
40 }
41
42 fmt.Println("current processed : ", startURL)
43
44 for _, link := range links {
45     currpath := append(path, link)
46
47     // capture the link so each goroutine is unique
48     link := link
49
50     gm.Run(func() {
51
52         DLSSingle(link, targetURL, currpath, resultpath, depth-1, gm, totalTraversed)
53     })
54
55 }
56
57 }
58
59 }
60

```

### 4.1.3 Multithreading

Pada kode BFS dan IDS, kami menggunakan multithreading agar waktu yang dibutuhkan ketika scraping lebih cepat, hal ini karena jika tidak dilakukan multithreading, satu kali scraping bisa mencapai 0.5 detik. Disini kami menggunakan goroutine dengan maksimal concurrent thread antara 100 sampai 300, tergantung spesifikasi laptop yang menjalankan. Penggunaan multithreading ini dilakukan saat pemanggilan rekursi DLS pada IDS dengan memanggil fungsi gm.Run, sedangkan bagian BFS multithreading dilakukan saat melakukan scraping dan mengisi elemen pada adjacent node. Pada kode IDS dan BFS sebelumnya juga terdapat penggunaan

mutex dan waitgroup. Mutex digunakan ketika melakukan write dan read pada shared resource, dalam kasus ini ketika melakukan write dan read pada cache, mutex digunakan agar tidak terjadi *race condition* yaitu ketika sistem melakukan write pada elemen yang sama secara berbarengan, yang memungkinkan terjadinya deadlock. Lalu, terdapat juga waitgroup, waitgroup adalah library bawaan Go yang bertujuan agar ketika dilakukan multithreading, sebuah proses tidak akan melanjutkan ke bagian selanjutnya sebelum seluruh thread yang berjalan telah selesai. Dalam IDS waitgroup digunakan agar sebuah thread tidak akan mendahului melanjutkan IDS iterasi selanjutnya, dan harus menunggu seluruh thread selesai sebelum melanjutkan. Pada BFS, waitgroup digunakan agar program akan menyelesaikan seluruh scraping sebelum melanjutkan ke pemrosesan algoritma selanjutnya.



```
1 package controllers
2
3 import "sync"
4
5 var mu sync.Mutex
6 var rwmu sync.RWMutex
7
8 type goRoutineManager struct {
9     goRoutineCnt chan bool
10 }
11
12 // goRoutineManager to limit amount of concurrent running goRoutine
13
14 func (g *goRoutineManager) Run(f func()) {
15     select {
16         case g.goRoutineCnt <- true:
17             wg.Add(1)
18             go func() {
19                 f()
20                 <-g.goRoutineCnt
21                 wg.Done()
22             }()
23         default:
24             f()
25     }
26 }
27
28
29 func NewGoRoutineManager(goRoutineLimit int) *goRoutineManager {
30     return &goRoutineManager{
31         goRoutineCnt: make(chan bool, goRoutineLimit),
32     }
33 }
34
```

## 4.2 Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).

### 4.2.1 Setup Back-End

- Pertama, clone repository back-end dengan melakukan

```
git clone https://github.com/dewodt/Tubes2\_BE\_MCCF
```

- Pastikan anda sudah di root directory dari folder back-end. Masuk ke dalam directory source code program dengan melakukan

```
cd src
```

- Untuk **development** (dengan hot reload)

- Build docker image

```
docker compose -f "docker-compose.development.yml"  
build
```

- Run docker container

```
docker compose -f "docker-compose.development.yml" up
```

- Untuk **production** (tanpa hot reload)

- Build docker image

```
docker compose -f "docker-compose.production.yml"  
build
```

- Run docker container

```
docker compose -f "docker-compose.production.yml" up
```

5. API Endpoint jalan pada <http://localhost:8080>
6. Untuk menghentikan proses tersebut, cari id container terlebih dahulu

```
docker ps
```

Lalu hentikan prosesnya dengan menjalankan

```
docker stop <container_id>
```

#### 4.2.2 Setup Front-End

1. Pertama, clone repository front-end dengan melakukan

```
git clone https://github.com/dewodt/Tubes2\_FE\_MCCE
```

2. Pastikan anda sudah di root directory dari folder front-end. Masuk ke dalam directory source code program dengan melakukan

```
cd src
```

3. Untuk **development** (dengan hot reload)

- a. Build docker image

```
docker compose -f "docker-compose.development.yml"  
build
```

- b. Run docker container

```
docker compose -f "docker-compose.development.yml" up
```

7. Untuk **production** (tanpa hot reload)

- a. Build docker image

```
docker compose -f "docker-compose.production.yml"  
build
```

- b. Run docker container

```
docker compose -f "docker-compose.production.yml" up
```

- 8. Front-end jalan pada <http://localhost:3000>
- 9. Untuk menghentikan proses tersebut, cari id container terlebih dahulu

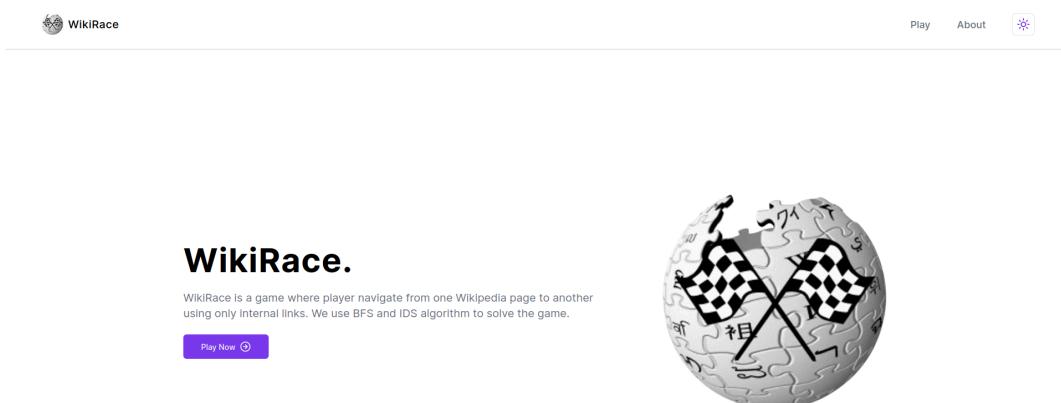
```
docker ps
```

Lalu hentikan prosesnya dengan menjalankan

```
docker stop <container_id>
```

#### 4.2.3 Menggunakan Aplikasi Web

- 1. Setelah melakukan setup [front-end](#) dan setup [back-end](#), kunjungi <http://localhost:3000>. Anda akan berada di halaman home.



- Untuk mengunjungi about page, Anda bisa klik button about pada navigation bar di atas. Di halaman ini juga terdapat informasi mengenai link repository front-end dan back-end.

The screenshot shows the 'About' page of the WikiRace website. At the top, there is a navigation bar with the WikiRace logo, 'Play', 'About', and a gear icon. Below the navigation bar, there is a large central box with a rounded rectangle containing the title 'About WikiRace'. The text inside the box describes the game: 'WikiRace is a game where player navigate from one Wikipedia page to another using only internal links. We use BFS and IDS algorithm to solve the game. This project is made to fulfill second big assignment of Algorithm Strategy. This project is open source and can be accessed from [FE Repository](#) and [BE Repository](#)'. At the bottom of the page, there is a copyright notice: 'Copyright © 2024 WikiRace'.

- Untuk memulai permainan wikirace, kunjungi halaman /play. Kemudian masukkan data input permainan wikirace. Kemudian submit. Jika terdapat error, maka perbaiki masukan Anda.

The screenshot shows the main play page of the WikiRace website. At the top, there is a navigation bar with the WikiRace logo, 'Play', 'About', and a gear icon. Below the navigation bar, the title 'WikiRace' is displayed in a large, bold font. There are several input fields and dropdown menus:

- A dropdown menu labeled 'Algorithm' with the option 'Select algorithm'.
- A dropdown menu labeled 'Solution Option' with the option 'Select solution option'.
- Text input fields for 'Start' and 'Target' with placeholder text 'Wikipedia title of the starting page' and 'Wikipedia title of the target page' respectively.
- A purple button at the bottom labeled 'Find Shortest Path'.

- Jika perhitungan berhasil, akan muncul hasil perhitungan dari permainan wikirace.

## Information

Traversed a total of **182850 articles** and found a total of **4** path solutions in **7.68 seconds** where the shortest path from **Joko Widodo** to **Philosophy** is **2 articles**

## Graph Visualization



## Individual Paths

	<b>Joko Widodo</b> President of Indonesia since 2014
	<b>Gadjah Mada University</b> University in Indonesia
	<b>Philosophy</b> study of the truths and principles of being, knowledge, or conduct

	<b>Joko Widodo</b> President of Indonesia since 2014
	<b>Pancasila (politics)</b> Indonesian political philosophy
	<b>Philosophy</b> study of the truths and principles of being, knowledge, or conduct

	<b>Joko Widodo</b> President of Indonesia since 2014
	<b>Somalia</b> sovereign state in Africa
	<b>Philosophy</b> study of the truths and principles of being, knowledge, or conduct

	<b>Joko Widodo</b> President of Indonesia since 2014
	<b>Sri Lanka</b> island country in South Asia
	<b>Philosophy</b> study of the truths and principles of being, knowledge, or conduct

## 4.3 Hasil pengujian (Screenshot antarmuka dan skenario yang memperlihatkan berbagai kasus yang mencakup seluruh fitur pada aplikasi Anda).

Skenario pengujian BFS dengan single path :

### WikiRace

**Algorithm**  
BFS  
Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**  
Single  
Get single or multiple path solutions

**Start**  
Manchester United F.C.  
Wikipedia title of the starting page

**Target**  
Ronaldo (Brazilian footballer)  
Wikipedia title of the target page

**Find Shortest Path**

### Graph Visualization

The graph visualization shows the shortest path from Manchester United F.C. to Ronaldo (Brazilian footballer). The path consists of three nodes: Manchester United F.C. (red), Intercontinental Cup (football) (orange), and Ronaldo (Brazilian footballer) (yellow). The orange node is connected to both red and yellow nodes by arrows. A legend indicates: Red circle = Start page, Orange circle = 1 degree away, Yellow circle = Target page.

### Information

Traversed a total of **14909 articles** and found a total of **1** path solutions in **2.82 seconds** where the shortest path from **Manchester United F.C.** to **Ronaldo (Brazilian footballer)** is **2 articles**

### Individual Paths

Manchester United F.C.  
association football club in Manchester, England

Hasil Pengujian BFS dengan multiple path :

## WikiRace

Algorithm

BFS

Algorithm used to find the shortest wikipedia path from start to target.

Solution Option

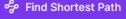
Multiple

Get single or multiple path solutions

Start Target

Humber Cinemas → Prince Edward Viaduct

Wikipedia title of the starting page Wikipedia title of the target page

 Find Shortest Path

### Information

Traversed a total of **4977 articles** and found a total of **4** path solutions  
in **1.59 seconds** where the shortest path from **Humber Cinemas** to  
**Prince Edward Viaduct** is **2 articles**

### Individual Paths

 <b>Humber Cinemas</b> former cinema in Toronto, Ontario, Canada	 <b>Humber Cinemas</b> former cinema in Toronto, Ontario, Canada
 <b>Toronto</b> capital and largest city of the province of Ontario, Canada	 <b>Bloor Street</b> major thoroughfare in Toronto
 <b>Prince Edward Viaduct</b> truss arch bridge system in Toronto	 <b>Prince Edward Viaduct</b> truss arch bridge system in Toronto
 <b>Humber Cinemas</b> former cinema in Toronto, Ontario, Canada	 <b>Humber Cinemas</b> former cinema in Toronto, Ontario, Canada
 <b>Danforth Music Hall</b> music venue and former cinema in Toronto, Ontario, Canada	 <b>The Royal Conservatory of Music</b> Canadian non-profit music education institution
 <b>Prince Edward Viaduct</b> truss arch bridge system in Toronto	 <b>Prince Edward Viaduct</b> truss arch bridge system in Toronto

Hasil pengujian IDS dengan multiple path :

## WikiRace

Algorithm  
IDS

Solution Option  
Multiple

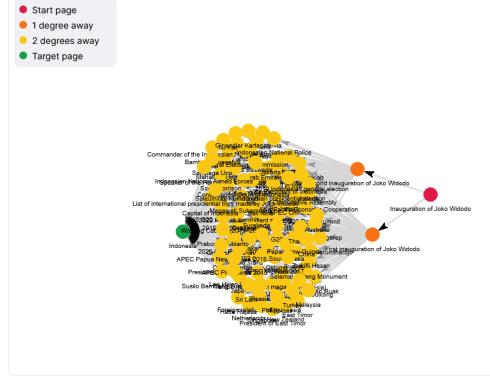
Start: Inauguration of Joko Widodo → Target: indonesia

Find Shortest Path

### Information

Traversed a total of 91578 articles and found a total of 147 path solutions in 1.57 seconds where the shortest path from Inauguration of Joko Widodo to indonesia is 3 articles

## Graph Visualization



Kasus perbandingan IDS dan BFS dengan kasus yang sama :

## WikiRace

Algorithm  
IDS

Solution Option  
Single

Start: New car smell → Target: Venus flytrap

Find Shortest Path

### Information

Traversed a total of 219855 articles and found a total of 1 path solutions in 4.59 seconds where the shortest path from New car smell to Venus flytrap is 3 articles

## WikiRace

Algorithm  
BFS

Solution Option  
Single

Start: New car smell → Target: Venus flytrap

Find Shortest Path

### Information

Traversed a total of 1931143 articles and found a total of 1 path solutions in 35.31 seconds where the shortest path from New car smell to Venus flytrap is 3 articles

Pada skenario single path, solusi dengan IDS lebih cepat (4.5 detik vs 35.3 detik)

**WikiRace**

**Algorithm**: BFS  
Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**: Multiple  
Get single or multiple path solutions

**Start**: New car smell  
Wikipedia title of the starting page

**Target**: Venus flytrap  
Wikipedia title of the target page

**Find Shortest Path**

**Algorithm**: IDS  
Algorithm used to find the shortest wikipedia path from start to target.

**Solution Option**: Multiple  
Get single or multiple path solutions

**Start**: New car smell  
Wikipedia title of the starting page

**Target**: Venus flytrap  
Wikipedia title of the target page

**Find Shortest Path**

**Information**  
Traversed a total of 852860 articles and found a total of 5 path solutions in 62.95 seconds where the shortest path from New car smell to Venus flytrap is 3 articles

**Information**  
Traversed a total of 7905960 articles and found a total of 5 path solutions in 67.05 seconds where the shortest path from New car smell to Venus flytrap is 3 articles

Pada skenario multiple path, BFS lebih cepat (62.9 detik vs 67 detik)

## 4.4 Analisis hasil pengujian

Dari hasil pengujian kami, Untuk jalur tunggal, IDS kemungkinan lebih cepat dari BFS, hal ini disebabkan IDS menelusuri suatu branch sampai selesai terlebih dahulu secara rekursif sehingga memungkinkan pemrosesan secara IDS lebih cepat menemukan target dibandingkan dengan BFS pada beberapa kasus, sedangkan BFS memproses node per lapis artinya BFS mengunjungi node berurutan sesuai kedalamannya sehingga terkadang lebih lambat untuk single path dari IDS.

Dalam pembuatan algoritma IDS kami, dalam setiap iterasi node yang telah dikunjungi sebelumnya tidak perlu lagi di scraping karena sudah disimpan dari cache sehingga pemrosesan node yang bukan node terdalam pada iterasi tersebut tidak terlalu memakan banyak waktu.

Untuk multi jalur, Biasanya BFS lebih cepat karena BFS mengunjungi lapis demi lapis Sehingga jika target URL ketemu dia akan membatasi pencarian hanya pada maksimal sedalam target URL, namun pada IDS, dia perlu melakukan iterasi pada tingkat kedalaman yang lebih rendah sebelum melakukan iterasi pada tingkat kedalaman target URL sehingga jika harus menelusuri seluruh node, BFS akan lebih cepat.

Untuk kompleksitas ruang nya IDS lebih efisien dari BFS karena IDS menyimpan setiap pengunjungan branch lalu dihapus lagi karena secara rekursif, dengan kompleksitas ruangnya adalah  $O(bd)$  dengan  $b$  adalah branching factor, dalam kasus ini branching factor adalah jumlah

rata-rata link yang terdapat pada suatu page wikipedia, dan  $d$  adalah kedalaman penjelajahan. Sedangkan BFS kompleksitas ruangnya adalah  $O(b^d)$  karena perlu menyimpan seluruh node. Untuk kompleksitas waktunya IDS mempunyai kompleksitas waktu  $O(b^d)$ , sedangkan BFS juga memiliki kompleksitas waktu  $O(b^d)$ . Hal ini dibuktikan dengan meningkatnya lama waktu penjelajahan seiring dengan meningkatnya kedalaman, sebagai contoh, penjelajahan dengan kedalaman 1, membutuhkan waktu 1 detik, kedalaman 2 membutuhkan waktu 15 detik, dan kedalaman 3 membutuhkan waktu minimal 1 menit. Peningkatan waktu yang dibutuhkan ini sesuai dengan kompleksitas waktu kedua algoritma tersebut.

# **BAB V**

## **KESIMPULAN DAN SARAN**

### **5.1 Kesimpulan**

Pada permainan Wikiracer masalah dapat digambarkan sebagai graf yang berarah tapi tak memiliki bobot. Masalah ini dapat diselesaikan dengan menggunakan teknik pengunjungan graf, tidak perlu menggunakan dijkstra atau bellman-ford karena graf tidak memiliki bobot. Maka dari itu teknik traversal yang dapat menghitung jarak terpendek pada graf yang tidak memiliki bobot dapat digunakan seperti BFS dan IDS. IDS kompleksitas waktunya sama dengan BFS tapi kompleksitas ruang nya lebih kecil. Sehingga IDS lebih efisien.

### **5.2 Saran**

Saran kedepannya untuk pemberian spek lebih diperhatikan dan jelas. Seperti info untuk caching dan lain lain.

# LAMPIRAN

Repository Back-End:

[https://github.com/dewodt/Tubes2\\_BE\\_MCCF](https://github.com/dewodt/Tubes2_BE_MCCF)

Repository Front-End:

[https://github.com/dewodt/Tubes2\\_FE\\_MCCF](https://github.com/dewodt/Tubes2_FE_MCCF)

Link Video:

 Video Tugas Besar 2 Stima MCCF

## **DAFTAR PUSTAKA**

Laaksonen, A. (2018). Competitive Programmer's Handbook. Diakses pada 25 april 2024.

Munir, Rinaldi. 2024. “Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1)”.  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>  
(Diakses pada 25 April 2024).

Munir, Rinaldi. 2024. “Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2)”.  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>  
(Diakses pada 25 April 2024).

Penulis StackPath. 2024. “What is a Web Application?”.  
<https://www.stackpath.com/edge-academy/what-is-a-web-application/>  
(Diakses pada 25 April 2024).

Penulis Ryte. 2021. “Web Architecture”.  
[https://en.ryte.com/wiki/Web\\_Architecture#Service-oriented\\_architectures\\_.28SOA.29](https://en.ryte.com/wiki/Web_Architecture#Service-oriented_architectures_.28SOA.29)  
(Diakses pada 25 April 2024).