

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



Disusun Oleh:

Randy Verdian (13522067)

Abdul Rafi Radityo Hutomo (13522099)

Rayendra Althaf Taraka Noor (13522107)

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

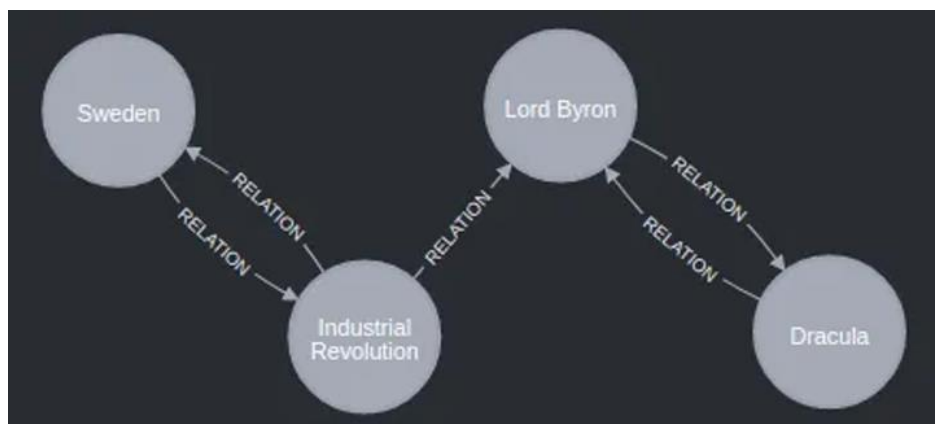
## **Daftar Isi**

<b>BAB I DESKRIPSI TUGAS.....</b>	<b>3</b>
<b>BAB II LANDASAN TEORI.....</b>	<b>4</b>
2.1. Dasar Teori .....	4
2.2. Penjelasan Singkat Mengenai Aplikasi Web yang Dibangun. ....	5
<b>BAB III ANALISIS PEMECAHAN MASALAH .....</b>	<b>6</b>
3.1 Proses Pemetaan dan Langkah-langkah Pemecahan Masalah .....	6
3.2 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun .....	16
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN .....</b>	<b>19</b>
4.1 Spesifikasi Teknis dan Implementasi Program .....	19
4.1.1 Spesifikasi Teknis Program .....	19
4.1.2 Penjelasan Implementasi Algoritma .....	22
4.2 Penjelasan Tata Cara Penggunaan Program.....	28
4.3 Hasil Pengujian .....	29
4.4 Analisis Hasil Pengujian .....	31
<b>BAB V KESIMPULAN DAN SARAN .....</b>	<b>33</b>
<b>LAMPIRAN .....</b>	<b>33</b>
<b>DAFTAR PUSTAKA.....</b>	<b>33</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.



Gambar 1. Ilustrasi Graf WikiRace

## BAB II

### LANDASAN TEORI

#### 2.1. Dasar Teori

Graf Dinamis merupakan sebuah variasi graf dimana simpul-simpul dan sisi-sisinya dapat bertambah dan berkurang seiring dengan pembaruan yang diberikan. Penjelahan graf dinamis sendiri tidak jauh berbeda dengan penjelajahan graf statis. Proses penjelajahan graf dinamis akan mengunjungi simpul-simpul dalam graf yang akan berkembang suatu kondisi tertentu untuk mencapai informasi yang diinginkan dari graf tersebut. Adapun algoritma yang digunakannya beragam, berikut beberapa algoritma yang umum digunakan untuk penjelajahan graf dinamis:

- Breadth First Search (BFS)

Algoritma BFS merupakan algoritma penjelajahan graf yang dimulai dari sebuah simpul awal, selanjutnya akan dilakukan penjelajahan ke simpul di sekitarnya dimana urutan pengunjungan simpul didapat berdasarkan tingkat kedalamannya. Pada umumnya, implementasi algoritma BFS dilakukan dengan memasukkan simpul awal ke dalam sebuah antrian. Selanjutnya, simpul awal akan dikeluarkan dari antrian lalu semua simpul yang bertetanggaannya akan dimasukkan ke dalam antrian tersebut. Berikutnya, bagian depan antrian akan dikeluarkan dan dimasukkan semua simpul yang bertetanggaannya dengan simpul tersebut ke dalam antrian. Langkah tersebut kemudian akan terus diulangi hingga tujuan ditemukan atau antrian habis.

- Iterative Deepening Search (IDS)

Algoritma IDS merupakan algoritma yang dibuat sebagai upaya mendapatkan *depth-first search's space-efficiency* dan *breadth-first search's fast search*. Penjelajahan graf

ini dilakukan dengan melakukan pengulangan pada Depth Limited Search dengan menambah batas kedalaman sampai target ditemukan. Algoritma ini memiliki keunggulan dalam batas memorinya yang lebih kecil dibanding dfs dan efisien untuk graf yang dangkal. Namun, untuk menemukan jalur terpendek algoritma ini tidak seefisien BFS dan tidak efisien untuk graf yang dalam karena setiap pengulangan akan mengulangi penjelajahan yang sama.

## **2.2. Penjelasan Singkat Mengenai Aplikasi Web yang Dibangun.**

Website yang dibuat terdiri dari Frontend dan Backend, dimana Frontend menggunakan bahasa Typescript dan framework NextJS dan TailwindCSS. Kemudian Backend menggunakan bahasa Go language dan framework Gin. Website menerima input *Start Page* dan *Target Page* dengan menggunakan autocomplete dari wikipedia. Kemudian menerima juga parameter Algoritma yang digunakan. Setelah menerima tiga parameter itu, kemudian data dikirim pada Backend yang mana selanjutnya akan diolah oleh Backend kemudian dikirim ke Frontend dan menampilkan path-path yang merupakan solusi dari Wikirace beserta *visited count*, *path length*, *execution time*, dan *number of solutions*.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1 Proses Pemetaan dan Langkah-langkah Pemecahan Masalah**

Untuk menyelesaikan permasalahan pada tugas besar kali ini, sebagai langkah awal mendapatkan solusi perlu dilakukan pemetaan elemen-elemen permasalahan menjadi komponen-komponen pada algoritma penjelelahan graf dinamis sebagai berikut:

- Pohon Ruang Status (State Space Tree) : Graf yang dibangun secara dinamis seiring dengan berjalannya penelusuran web
- Simpul : Artikel Wikipedia
  - Akar : Artikel awal
  - Daun : Artikel yang ingin dituju
- Operator : Melakukan penelusuran hyperlink
- Ruang Solusi : Himpunan dari kumpulan artikel wikipedia  $A_i$  yang terurut yang dimulai dengan artikel awal dan diakhiri dengan artikel yang dituju sehingga untuk setiap artikel  $A_i$  memiliki hyperlink yang menuju ke artikel  $A_{i+1}$

Ketika pemetaan komponen-komponen graf dinamis sudah terbentuk sebagaimana diatas, langkah-langkah untuk mendapatkan komponen tersebut dari website Wikipedia serta pengolahannya akan bergantung sesuai dengan algoritma yang digunakan. Pada tugas kali ini akan ada 2 pilihan algoritma yang bisa digunakan untuk menyelesaikan permasalahan yang diberikan yakni *Breadth First Search* dan *Iterative Deepening Search*. Lebih spesifiknya, berikut adalah perincian langkah-langkah yang dilakukan oleh kedua algoritma tersebut:

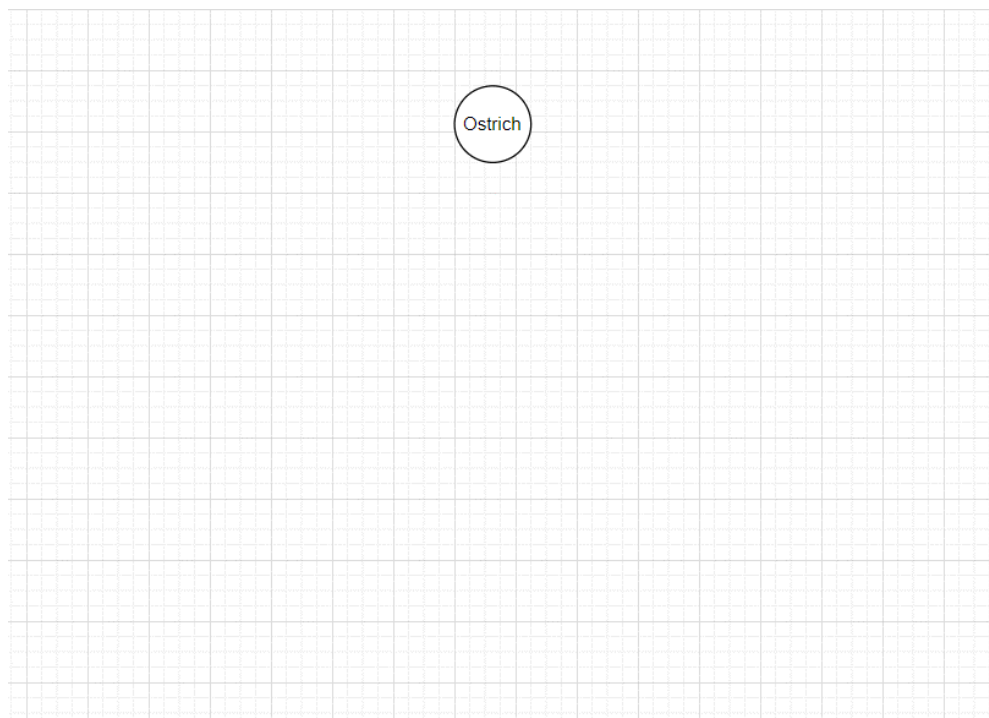
#### **Algoritma BFS**

Pada tugas besar ini algoritma diimplementasikan dengan rincian tahapan sebagai berikut beserta ilustrasi kasus untuk pencarian link wikipedia dari artikel Ostrich ke artikel Camel:

### 1. Inisialisasi Pohon Ruang Status

Pohon Ruang Status diinisialisasi dengan satu simpul, yaitu simpul akar yang pada permasalahan ini adalah artikel wikipedia awal.

Sehingga pada ilustrasi kasus terbentuk Pohon Ruang Status sebagai berikut

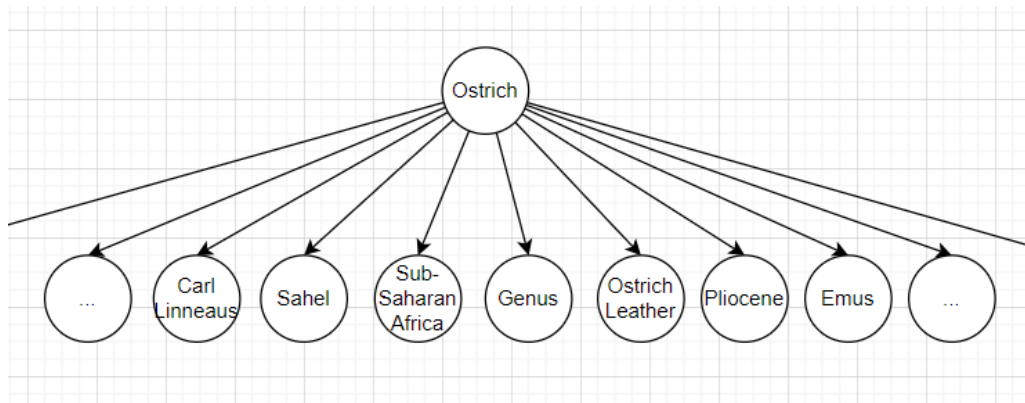


### 2. Pembangkitan Status

Pembangkitan Status dilakukan dengan cara mengaplikasikan operator, yaitu melakukan penelusuran dari hyperlink yang ada pada artikel Wikipedia. Dengan masing-masing hyperlink yang dapat ditelusuri dari artikel Wikipedia tersebut berperan menjadi simpul anak dari simpul yang mengandung hyperlink tersebut.

Pembangkitan status pada algoritma Breadth First Search dilakukan pada simpul yang memiliki kedalaman sama terlebih dahulu.

Pada Ilustrasi kasus pengaplikasian operator pada Ostrich menghasilkan hasil sebagai berikut (tidak semua artikel yang mungkin ditunjukkan) dengan urutan dibangkitkannya simpul berurut dari kiri ke kanan (berarti Carl Linneaus dibangkitkan sebelum Sahel kemudian Sub-Saharan Africa dan seterusnya).

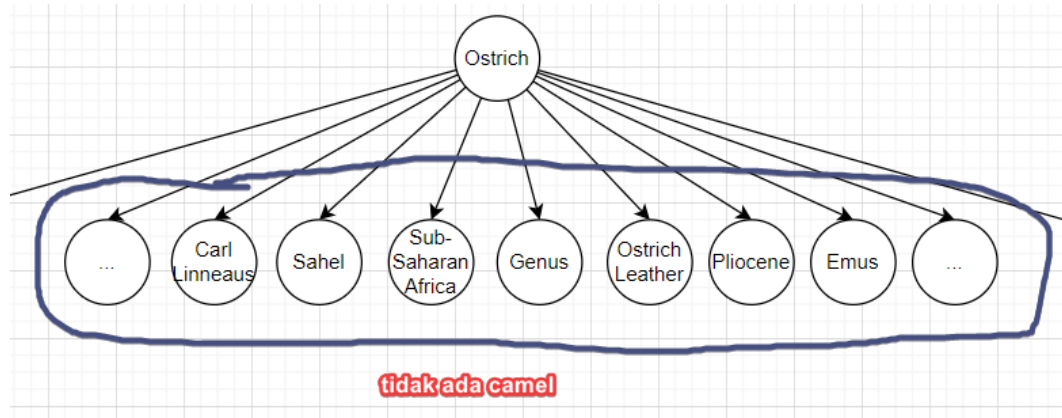


### 3. Pengecekan solusi

Pengecekan solusi dilakukan seiring dengan pembangkitan status, dan apabila dilakukan BFS untuk mencari sebuah solusi saja, maka program akan berhenti ketika telah ditemukan satu solusi. Namun, pada tugas besar ini algoritma yang dirancang adalah untuk mencari semua solusi terpendek, sehingga ketika simpul daun telah ditemukan, proses pembangkitan status akan terus dilakukan hingga semua simpul pada kedalaman yang sama telah diaplikasikan operator dan didapat seluruh simpul anaknya.

Pada ilustrasi kasus, semua anak dari ostrich tidak ada satupun yang merupakan artikel yang dituju, yaitu camel, sehingga belum ditemukan solusi dan penacarian dilanjutkan kembali.





#### 4. Pembangkitan Status Kembali

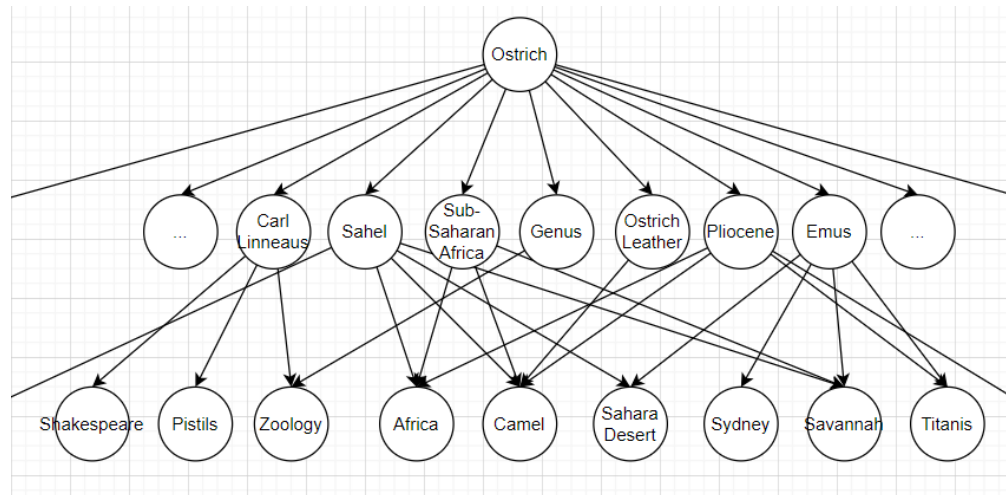
Algoritma BFS terus dilanjutkan hingga ditemukan solusi, sehingga apabila pada pengecekan solusi tidak ditemukan solusi maka akan dilakukan pembangkitan status lagi. Pada algoritma BFS, urutan pembangkitan status dari suatu simpul dilakukan dengan urutan yang sama dengan urutan dibangkitkannya simpul. Sehingga pada ilustrasi kasus simpul ditelusuri hyperlink dari semua simpul sebelum Carl Linneaus, kemudian hyperlink dari Carl Linneaus, hyperlink dari Sahel, hyperlink dari Sub-Saharan Africa, hyperlink dari Genus, dan seterusnya.

Perlu diperhatikan bahwa untuk pembangkitan status yang tidak berasal dari simpul akar, ada kemungkinan bahwa hasil aplikasi operator akan menghasilkan sisi yang mengarah kembali ke simpul pada kedalaman yang sama atau pada kedalaman yang lebih dangkal dari kedalaman simpul saat itu, pada algoritma BFS tidak digambarkan sisi antara kedua simpul tersebut, sebab sisi tersebut tidak mungkin menjadi bagian dari solusi dengan kedalaman terendah.

Selain itu, juga ada kemungkinan bahwa ada simpul yang dibangkitkan yang sebelumnya telah dibangkitkan juga dari hasil aplikasi operator pada simpul yang memiliki kedalaman yang sama dengan simpul yang saat ini sedang ditelusuri. Pada kasus ini, untuk menyelesaikan permasalahan mencari semua

solusi dengan kedalaman terendah, sisi antara kedua simpul tersebut tetap dimasukkan ke dalam pohon ruang status karena berpotensi menjadi bagian dari solusi paling optimal.

Pada Ilustrasi kasus didapatkan pohon ruang status yang baru sebagai berikut



Perlu diperhatikan bahwa pada ilustrasi tidak semua simpul digambarkan. Selain itu, karena telah ditemukan simpul tujuan pada kedalaman saat ini, maka telah algoritma BFS telah selesai dan semua solusi terpendek telah didapatkan.

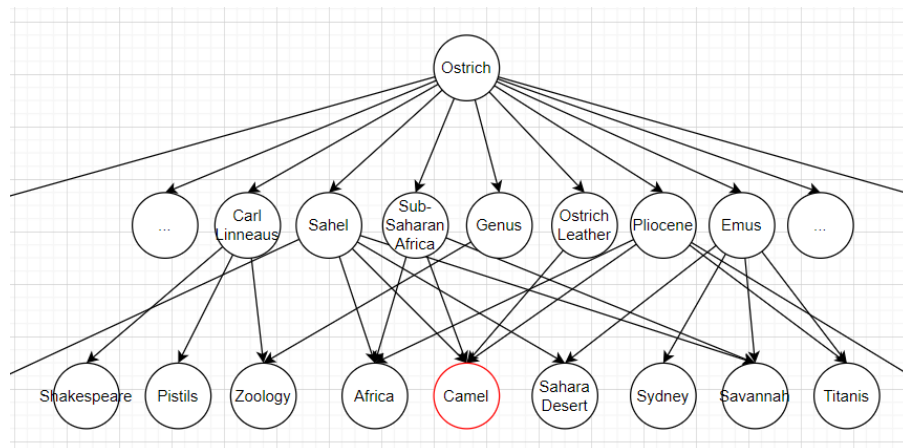
##### 5. Penelusuran solusi

Untuk didapatkan pohon yang hanya mengandung solusi dilakukan penelusuran pada graf yang telah dibuat dimulai dari simpul akhir kembali ke simpul akar. Kemudian, simpul-simpul yang dilalui pada proses tersebutlah yang menjadi pohon solusi.

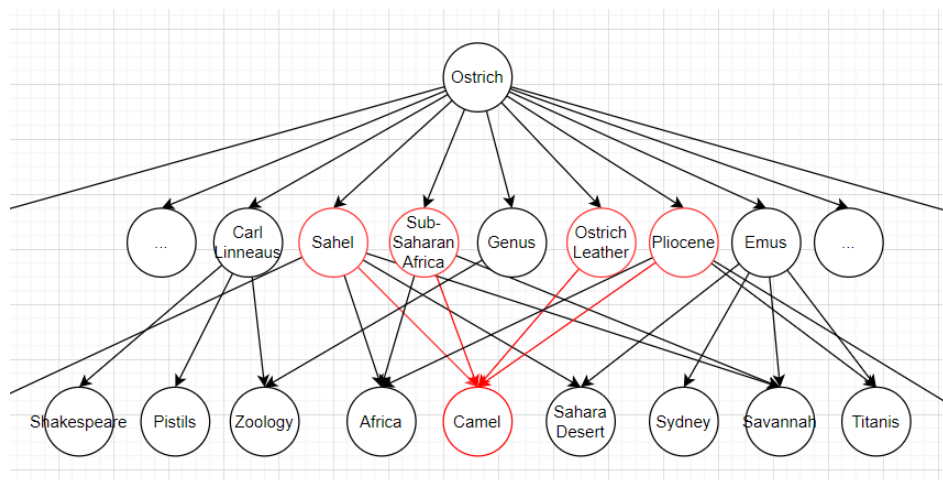
Pembentukan pohon solusi pada ilustrasi dapat digambarkan sebagai berikut :

Pohon solusi yang sedang dibentuk digambarkan dengan warna merah

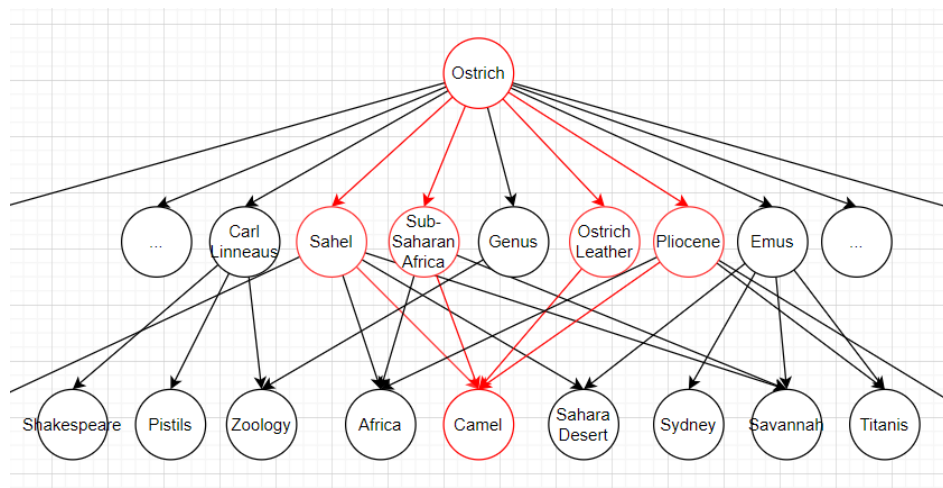
Iterasi 1 :



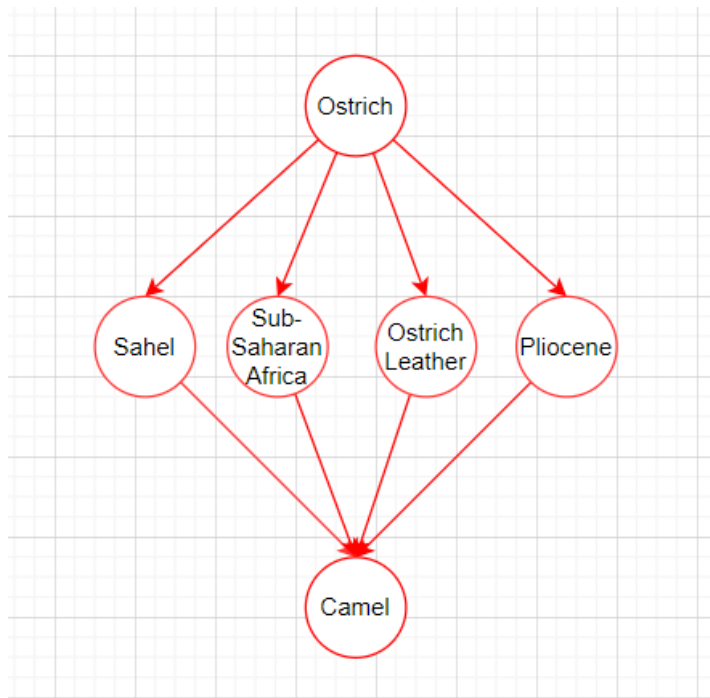
Semua Parent dari camel menjadi bagian dari pohon solusi pada iterasi 2 :



Pada iterasi 3, semua simpul ditelusuri kembali ke simpul akar :



Karena penelusuran balik telah dilakukan hingga simpul akar maka pembentukan pohon solusi telah selesai dan didapat pohon solusi sebagai berikut :



Pohon solusi ini menggambarkan 4 solusi, yaitu Ostrich-Sahel-Camel, Ostrich-Sub Saharan Africa-Camel, Ostrich-Ostrich Leather-Camel dan Ostrich-Pliocene-Camel.

### Algoritma IDS

Pada tugas besar ini, implementasi yang dilakukan untuk Algoritma IDS terbagi menjadi beberapa langkah sebagai berikut:

#### 1. Inisialisasi Pohon Ruang Status

Pada langkah ini cukup dilakukan penambahan simpul akar ke dalam pohon ruang status. Simpul akar ini juga akan ditandai dengan nilai 0 sebagai jaraknya dari simpul akar (dirinya sendiri).

#### 2. Pembangkitan Ruang Status

Pada langkah ini akan dilakukan perluasan pohon ruang status dari simpul-simpul terluar. Simpul-simpul yang baru akan didapatkan dari kumpulan hyperlink ke artikel

Wikipedia yang terdapat pada artikel Wikipedia yang berada pada simpul-simpul dengan kedalaman terbesar yang ada dalam pohon ruang status.

### 3. Penelusuran Solusi dengan DLS

Penelusuran pada tahap ini akan dilakukan dari simpul akar menuju semua simpul yang sudah terbentuk pada graf dinamis yang dibuat. Penelusuran akan dilakukan secara mendalam terlebih dahulu hingga mencapai kedalaman tertentu pada graf yang telah dibuat. Selama proses ini, pada setiap simpul juga akan dibuat sebuah map bernama parent yang berisikan simpul-simpul lain yang memiliki kedalaman tepat 1 di bawah simpul tersebut dan memiliki graf yang menghubungkan keduanya. Diberikan juga pada setiap simpul sebuah nilai yang menandakan jaraknya dari simpul akar.

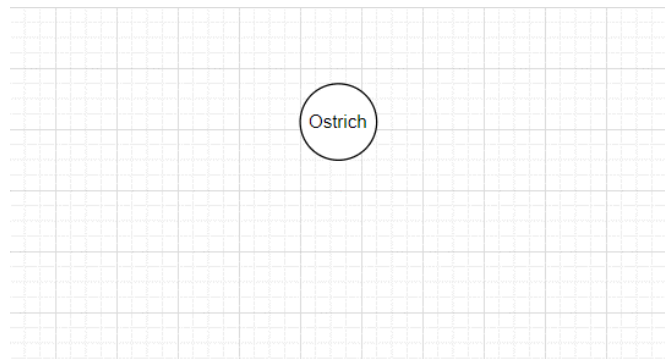
### 4. Pengulangan Pembangkitan Ruang Status dan Penelusuran Solusi

Selama simpul tujuan belum ditemukan, proses kedua dan ketiga dari algoritma ini akan terus dilakukan dengan mengubah batas kedalaman yang digunakan oleh proses Depth Limited Search. Jika tujuan sudah ditemukan hasil akhir bisa didapat dengan mereduksi map parent menjadi map parent yang terhubung dengan simpul tujuan dan membalikinya.

Untuk memberikan gambaran yang lebih baik, berikut contoh ilustrasi kasus untuk pencarian dari artikel Wikipedia "Ostrich" ke "Camel":

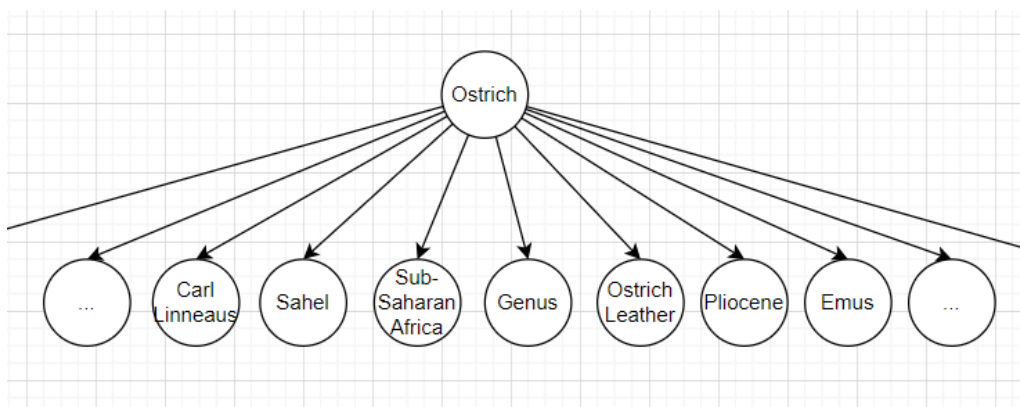
#### 1. Inisialisasi Pohon Ruang Status

Pada proses ini akan ditambahkan simpul artikel awal yakni simpul Ostrich sehingga didapat pohon sementara sebagaimana gambar dibawah.



## 2. Pembangkitan Ruang Status

Sebagaimana penjelasan pada bagian sebelumnya, pada langkah ini akan ditambah simpul-simpul baru dari simpul-simpul yang berhubungan dengan simpul-simpul yang terluar dan untuk kasus ini adalah simpul akar itu sendiri. Ketika proses selesai dilakukan berikut kurang lebih ilustrasi pohon ruang status terkini:

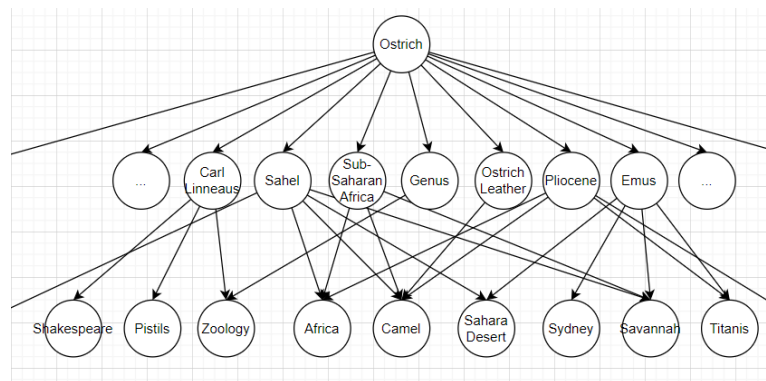


## 3. Penelusuran Solusi dengan DLS

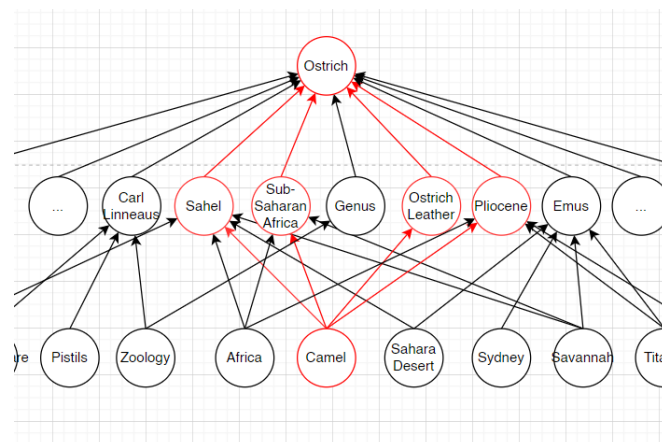
Pada langkah ini akan dilakukan DLS dengan batas kedalaman 1. Simpul-simpul yang baru dibuat kemudian diberi nilai parent dan kedalaman. Untuk kasus ini semua simpul baru akan memiliki kedalaman 1 dan hanya memiliki 1 parent yakni "Ostrich".

## 4. Pengulangan Pembangkitan Ruang Status dan Penelusuran Solusi

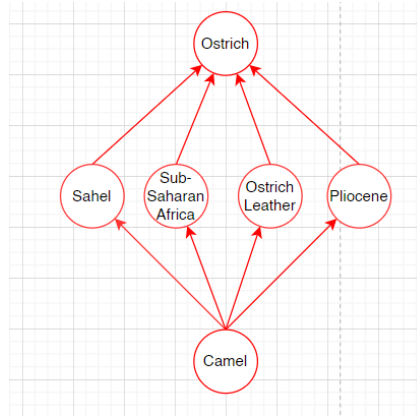
Karena simpul tujuan, artikel "Camel", belum ditemukan, akan dilakukan kembali langkah ke-2 dan ke-3 sampai hasil ditemukan. Pengulangan langkah ke-2 akan menghasilkan pohon baru sebagai berikut:



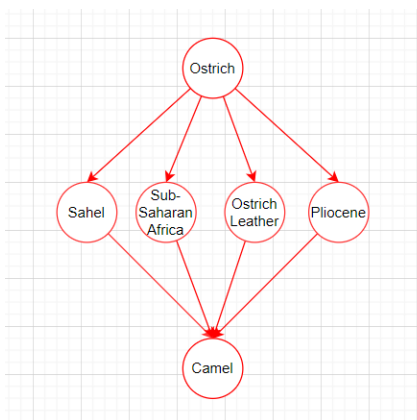
Kemudian dilakukan juga pengulangan dari proses ke-3 sehingga setiap node baru akan memiliki kedalaman 2 dan memiliki graf parent sebagai berikut:



Pada pengulangan kali ini simpul tujuan sehingga pengulangan diselesaikan kemudian hasil pun bisa didapat dari mereduksi graf parent dengan hanya mengambil simpul yang bisa diraih dari simpul tujuan. Berikut ilustrasi graf parent yang sudah tereduksi:



Graf tersebut kemudian dibalik sehingga didapat graf hasil sebagai berikut:



### 3.2 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

#### Fitur Fungsional

No	Fitur Fungsional	Penjelasan
F01	Perangkat lunak dapat menerima masukkan artikel wikipedia awal dan artikel tujuan dari pengguna	Perangkat lunak dapat menerima masukkan artikel wikipedia awal dan artikel tujuan dengan menggunakan fitur autocomplete
F02	Perangkat lunak dapat mendapatkan solusi berupa jalur dari artikel awal ke artikel tujuan	Perangkat lunak dapat memberikan judul/tautan artikel wikipedia secara terurut dari artikel awal hingga artikel tujuan sehingga setiap elemen memiliki hipertaut ke elemen berikutnya serta panjang solusi tersebut merupakan solusi optimal/terpendek



F03	Perangkat lunak dapat mendapatkan semua solusi terpendek dan memvisualisasikannya	Perangkat lunak dapat mendapatkan semua solusi terpendek dengan memvisualisasikannya dalam bentuk graf
F04	Perangkat lunak dapat menunjukkan informasi terkait eksekusi program	Perangkat lunak mampu menampilkan waktu yang dibutuhkan untuk mencari solusi, panjang solusi, jumlah solusi yang ditemukan, dan tautan yang dikunjungi

Aplikasi Web yang dibangun memiliki arsitektur Web yang terdiri dari komponen Front-End dan Back-End. Komponen Front End atau client side, adalah bagian dari aplikasi web yang dapat dilihat dan diinteraksikan oleh pengguna. Bagian ini bertanggung jawab untuk menampilkan informasi dan memungkinkan pengguna untuk berinteraksi dengan aplikasi. Back end, atau sisi server, adalah bagian dari aplikasi web yang berjalan di server. Bagian ini bertanggung jawab untuk memproses data, menjalankan logika aplikasi, dan berkomunikasi dengan database jika menggunakan database.

Pada Tugas Besar ini, bagian Front-End diimplementasikan menggunakan framework Node.js dengan menggunakan bahasa TypeScript. Aplikasi Front-End bertanggung jawab dalam menerima input page wikipedia awal dan akhir dari pengguna, kemudian mengirimkan POST request kepada komponen Back-End lalu menampilkan hasil algoritma pada web page. Visualisasi graf pada Front-End dilakukan dengan menggunakan library react-vis-network-graph. Komponen Back-End pada Tugas Besar ini dibangun dengan bahasa Go menggunakan framework Gin. Komponen Back-End menyediakan endpoint pada <http://localhost:8080/api/submit> yang menyediakan handler untuk request POST dengan body json berisi startPage (page wikipedia untuk memulai pencarian), endPage (page wikipedia yang dituju), dan algorithm (bernilai BFS/IDS, yaitu algoritma yang ingin digunakan). Kemudian, server akan mengirimkan response dengan body yang berisi paths(graf dengan representasi

map of parent nodes), time (waktu yang dibutuhkan untuk mengeksekusi program dalam milisekon), path\_length (panjang solusi), visitedCount (jumlah node yang dikunjungi).

Kemudian juga terdapat program Dockerfile dan docker-compose.yml yang digunakan untuk membuat dan menjalankan aplikasi server yang ditulis dalam Go di dalam Docker container. Dengan menggunakan Docker dan Docker Compose seperti ini, dapat dengan mudah membangun dan menjalankan aplikasi server Go dalam *environment* yang terisolasi.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Spesifikasi Teknis dan Implementasi Program

##### 4.1.1 Spesifikasi Teknis Program

Spesifikasi Teknis scraping wikipedia

Struktur Data

Struktur Data	Deklarasi	Kegunaan
regexp	string	Regular Expression digunakan untuk matching dengan format link wikipedia
set	map[string][]bool	Map yang secara efektif digunakan sebagai set yang berisi seluruh hyperlink dari sebuah tautan halaman Wikipedia

Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
titleToUrl	string -> string	Regular Expression digunakan untuk matching dengan format link wikipedia
urlToTitle	string -> string	Mengolah url Wikipedia menjadi judul artikel
getUntil	string -> string	Mengolah judul artikel Wikipedia menjadi tautan
getHyperlinks	string -> map[string][]bool	Mengembalikan set yang berisi seluruh hipertaut dari sebuah artikel wikipedia

Spesifikasi Teknis untuk pencarian BFS

Struktur Data

Struktur Data	Deklarasi	Kegunaan
Queue	Deque	Menggunakan Deque yang digunakan sebagai queue dalam pemrosesan elemen BFS
QueueItem	name : string int : depth	Sebagai elemen pada queue, perlu menyimpan juga informasi depth untuk mengetahui
Graf	map[string][]string	Representasi dengan map dengan key sebagai node dan child sebagai parentnya

Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
-----------------	-----------	----------

existInGraph	string, *graf -> boolean	Mengecek apakah sebuah node ada di dalam graf
BfsMultiThread	string, string -> graf, int, int, int	1. Menggunakan algoritma BFS untuk mengembalikan pohon solusi, waktu yang dibutuhkan, jumlah simpul yang dilewati dan panjang kedalaman solusi

## Spesifikasi Teknis untuk pencarian IDS

### Struktur Data

Struktur Data	Deklarasi	Kegunaan
Graf	map[string][]string	Representasi graf dalam bentuk map dimana key mewakili sebuah node dan value berisikan array ke node-node lain yang menunjukkan keberadaan sisi berarah dari node key ke node-node di value
ParentGraf	map [string][]string	Representasi graf dalam bentuk map dimana key mewakili sebuah node dan value berisikan array ke node-node lain yang menunjukkan keberadaan sisi berarah dari node-node pada value ke node key
ClosestDistance	map[string]int	Key pada map ini berisikan node-node pada graf dan memiliki value bernilai jarak terdekat dari node tersebut ke node awal

### Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
idsProccess	string, string, int, Graf -> ParentGraf, ClosestDistance, int	Melakukan pemanggilan ke dls dengan batas kedalaman yang diberikan dan jika node tujuan belum didapatkan fungsi ini akan memanggil dirinya sendiri dengan kedalaman maksimal yang ditambahkan. Fungsi ini mengembalikan parent graf dari titik-titik yang berhubungan node awal dan node tujuan, ClosestDistance dari node-node tersebut, dan banyaknya node yang sudah dijelajahi.
dls	string, string, map[int] int, int, Graf, ParentGraf, map[string] bool, map[string] bool, ClosestDistance	Fungsi ini melakukan penjelajahan graf dengan algoritma Depth Limited Search dengan kedalaman yang diberikan.

## Spesifikasi Teknis kakas Multithreading

### Struktur Data

Struktur Data	Deklarasi	Kegunaan
Mutex	state : int sema : int	Dari library bawaan synchronization Go, berguna untuk melakukan operasi yang bersifat mutually exclusive seperti akses read/write map
WaitGroup	counter : int waiter_count : int sema : int	Dari library bawaan synchronition Go, digunakan untuk mensinkronisasi thread agar tidak terjadi race condition

#### Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
go	func -> void	Secara teknis bukan fungsi atau prosedur, tetapi digunakan untuk membuat sebuah thread untuk melakukan func yang dipass

#### Spesifikasi kaskas Komponen Back-End

#### Fungsi/Prosedur

##### Struktur Data

Struktur Data	Deklarasi	Kegunaan
FormData	startPage : string targetPage : string Algorithm : string	Struktur untuk menyimpan body json request dari front end

#### Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
enableCORS	httpHandler -> httpHandler	Menyalakan Cross Origin Resource Sharing agar front-end dapat mengirim request kepada Back-End meskipun berasal dari port yang berbeda
jsonContent-TypeMiddleware	httpHandle -> httpHandler	Untuk mengatur Header JSON sehingga memiliki content-type application/json
validateFormData	FormData -> error	Untuk memvalidasi FormData dari Front-End
resultToJson	graf	Memparsing graf ke bentuk yang lebih mudah untuk diproses di front-end
submitHandler	*gin.context	Prosedur untuk menghandle dan memberikan response atas POST request yang dikirimkan oleh front-end
main	void	Main program, melakukan pengaturan router dan handler

#### Spesifikasi kaskas Komponen Front-End

## Fungsi/Prosedur

### Struktur Data

Struktur Data	Deklarasi	Kegunaan
MapItem	[key: string]: string[]	Merepresentasikan data peta (mapData) yang digunakan dalam pembuatan graf
stringToIntMap	[key: string]: number	Memetakan judul halaman Wikipedia ke ID node dalam graf
graph	nodes edges	Membangun graf yang akan dirender dalam komponen Network Graph

## Fungsi/Prosedur

Fungsi/Prosedur	Deklarasi	Kegunaan
getImageUrlFromResponse	data: WikiResponse	Menghasilkan URL gambar yang akan ditampilkan dalam graf
convertToSnakeCase	name: string	Mengonversi judul halaman Wikipedia ke dalam format yang sesuai untuk penggunaan dalam URL
getImageURL	title: string	Mendapatkan URL gambar untuk setiap node dalam graf
convertMapToGraph	mapData: MapItem[], startPage: string, endPage: string	Membangun graf berdasarkan data peta dan mengatur level kedalaman setiap node berdasarkan jaraknya dari halaman awal
handleStartSuggestionClick	suggestion: string	Menangani pemilihan saran untuk halaman awal
handleTargetSuggestionClick	suggestion: string	Menangani pemilihan saran untuk halaman target
fetchStartPageSuggestions	searchTerm: string	Menampilkan saran pencarian untuk halaman awal
fetchTargetPageSuggestions	searchTerm: string	Menampilkan saran pencarian untuk halaman target

### 4.1.2 Penjelasan Implementasi Algoritma

Algoritma BFS diimplementasikan pada bahasa Go Lang sesuai dengan algoritma yang telah dijelaskan pada Bab III dengan tahapan beserta source code sebagai berikut

```

130 graph := make(map[string][]QueueItem)
131
132 // visited contains urls that are already scraped
133 visited := make(map[string]bool)
134
135 // Create a queue for storing solutions (paths)
136 var theQueue deque.Deque[QueueItem]
137
138 // Initialize the queue with starting page
139 theQueue.PushFront(QueueItem{name: start, depth: 1})
140

```

a. Inisialisasi

Pada tahap ini diinisialisasikan struktur data yang akan digunakan, graph adalah graf yang akan dibentuk secara dinamis seiring dengan dilakukannya web scrapping.

Visited adalah set yang digunakan untuk menyimpan tautan yang telah dikunjungi/discrapping sehingga tidak discrapping kembali. Kemudian, pada algoritma BFS ini digunakan struktur data Queue untuk melakukan pemrosesan simpul secara sekuensial. Inisialisasi diakhiri dengan melakukan pushFront/Enqueue simpul awal yang dalam kasus ini adalah tautan artikel wikipedia awal ke dalam queue

```

// keep looping until queue length is 0 and solution is not found or solution is found but
// haven't moved on, onto the next depth
for theQueue.Len() != 0 && (!solFound || (solFound && theQueue.Front().depth == solLength-1)) {

    currentDepth = theQueue.Front().depth
    fmt.Printf("Currently at depth : %d, with %d item in queue\n", currentDepth, theQueue.Len())

    // for loop where, every iteration is tied to it's own thread
    for i := 0; i < threadCount; i++ {
        wg.Add(1)
        queueLock.Lock()

        // skip if current depth is done
        if theQueue.Len() == 0 || theQueue.Front().depth != currentDepth {
            wg.Done()
            queueLock.Unlock()
            break
        }
    }
}

```

b. Main loop

Main loop program sendiri adalah proses iteratif dari pencarian, dimana program akan terus melakukan pencarian hingga kondisi berhenti telah terpenuhi. Pada source code tersebut dapat dilihat bahwa program akan terus berlanjut ketika panjang queue tidak 0 (seharusnya selalu berlaku), !solFound atau belum ditemukan solusi, atau ketika sudah ditemukan solusi tetapi masih berada pada kedalaman yang sama dengan solusi – 1 sehingga masih berpotensi untuk menghasilkan solusi lain yang memiliki jarak yang sama.

c. Proses mendapatkan hipertaut

```
var currentHyperlinks map[string]bool

currentItem := theQueue.PopFront()
currentLink := currentItem.name
queueLock.Unlock()

currentHyperlinks = getHyperlinks(currentLink)
QueriedPage += 1
mapLock.Lock()
visited[currentLink] = true
mapLock.Unlock()
```

Pemrosesan tautan/simpul yang berada pada queue dimulai dengan pertama-tama melakukan dequeue untuk mendapatkan elemen yang perlu diproses. Kemudian, dilakukan scraping dan didapatkan semua anak dari simpul tersebut, yaitu semua hipertaut yang menuju ke artikel wikipedia lainnya dari laman wikipedia tersebut.

d. Pembangkitan status



```

// if solution found, add to graph, no need to iterate further
if currentHyperlinks[end] {
    solFound = true
    solLength = currentDepth + 1
    graphLock.Lock()
    graph[end] = append(graph[end], QueueItem{name: currentLink, depth: currentDepth})
    graphLock.Unlock()
} else if !solFound {
    // for every link inside the hyperlink
    for iter := range currentHyperlinks {
        graphLock.Lock()

        // construct the node if it doesn't exist and add edges in the graph
        if existInGraph(iter, &graph) {
            if currentDepth == graph[iter][0].depth {
                graph[iter] = append(graph[iter], QueueItem{name: currentLink, depth: currentDepth})
            }
        } else {
            graph[iter] = []QueueItem{{name: currentLink, depth: currentDepth}}
        }
        graphLock.Unlock()
        mapLock.Lock()

        // if the link hasn't been explored, add it to the queue
        if !visited[iter] {
            visited[iter] = true
            mapLock.Unlock()
            if iter != end {
                queueLock.Lock()
                var newItem QueueItem
                newItem.name = iter
                newItem.depth = currentDepth + 1
                theQueue.PushBack(newItem)
                queueLock.Unlock()
            }
        } else {
            mapLock.Unlock()
        }
    }
}

```

Kemudian, dari hasil hipertaut yang telah didapat tadi, pertama dicek terlebih dahulu apakah terdapat artikel tujuan, apabila ada maka solusi tersebut dimasukkan ke dalam graf dan proses selesai. Apabila artikel tujuan tidak ada dalam hipertaut maka akan ke dalam percabangan else if. Terdapat dua kemungkinan, yaitu yang pertama saat sudah ditemukannya solusi, yang berarti simpul yang ada di kedalaman selanjutnya tidak perlu dieksplor lagi karena jaraknya sudah sama dengan jarak dari akar ke solusi sehingga tidak mungkin memberikan solusi yang lebih singkat. Kemudian, kemungkinan kedua adalah apabila belum ditemukan solusi yang berarti simpul hasil hipertaut tersebut semuanya masih memiliki kemungkinan untuk menjadi bagian dari solusi. Sehingga, pertama pada graf ditambahkan simpul (jika belum ada) dan sisi yang berhubungan dengan tautan asalnya. Lalu, apabila tautan tersebut belum pernah

dimasukkan ke dalam queue, maka tautan tersebut akan dienqueue untuk diproses.

Penggunaan queue pada proses BFS penting untuk menjaga keterurutan pemrosesan simpul sehingga program tidak akan melakukan pengecekan kedalaman selanjutnya sebelum kedalaman saat itu selesai dieksplorasi terlebih dahulu.

e. Pembentukan pohon solusi

```
// construct the solution tree, using bfs from the result back to the starting node
resultGraph := make(map[string][]string)
var outputQueue deque.Deque[string]
outputQueue.PushBack(end)
for outputQueue.Len() != 0 {
    for _, item := range graph[outputQueue.Front()] {
        resultGraph[outputQueue.Front()] = append(resultGraph[outputQueue.Front()], item.name)
        if item.name != start {
            outputQueue.PushBack(item.name)
        }
    }
    outputQueue.PopFront()
}
resultGraph[start] = []string{}

return resultGraph, (elapsedTime), QueriedPage, int(solLength)
```

Kemudian, dari graf yang didapat dibuat pohon solusi dengan cara yang telah dijelaskan pada bab 3, implementasinya pada program juga menggunakan queue kembali dengan elemen awal adalah artikel tujuan.

Implementasi algoritma IDS juga dilakukan pada bahasa Go Lang. Sedangkan untuk pembagiannya, seperti yang sudah dijabarkan pada bab 3, algoritma terbagi menjadi 4 langkah utama.

a) Inisialisasi Pohon Ruang Status

Tahap inisialisasi memasukkan node awal pada graf tidak perlu dilakukan karena akan terjadi dengan sendirinya ketika algoritma dijalankan. Sehingga yang diperlukan hanya memberi nilai parent kosong dan jarak simpul awal ke simpul awal yakni nol.

```
closestDist[source] = 0
visitedCount = 0
parent := make(map[string][]string)
var emptyArrayOfString []string
parent[source] = emptyArrayOfString
```

## b) Pembangkitan Ruang Status

Pada pembangkitan status pertama yang dilakukan, pembaruan node hanya dilakukan dari simpul awal.

```
wg.Add(1)
go getHyperlinks(sourceUrl, &nearbyNode)
wg.Wait()
```

Sedangkan jika dilakukan pengulangan, pembaruan node akan dilakukan hanya dari node terluar.

```
// do getHyperlink for all children
if currentDepth > 1 {
    threadCount = 0
    for _, node := range currentNearby {
        readLock.Lock()
        readCount += 1
        if readCount == 1 {
            writeLock.Lock()
        }
        readLock.Unlock()
        _, nearbyExist := (*nearbyNode)[node]
        readLock.Lock()
        readCount -= 1
        if readCount == 0 {
            writeLock.Unlock()
        }
        readLock.Unlock()

        if !nearbyExist {
            threadLock.Lock()
            if threadCount >= 250 {
                wg.Wait()
                time.Sleep(time.Second)
                threadCount = 0
            }
            threadLock.Unlock()
            wg.Add(1)
            threadCount += 1
            go getHyperlinks(node, nearbyNode)
        }
    }
    wg.Wait()
}
```

## c) Penelusuran Solusi dengan DLS

Penelusuran disini mirip dengan DFS, tetapi bukan sampai tujuan ditemukan tetapi dilakukan hingga ke titik batas kedalaman tertentu.

```
// visit all the child node
for _, node := range currentNearby {
    PathExist := (*closestDist)[node]
    if !PathExist {
        (*closestDist)[node] = (*closestDist)[source] + 1
    }
    if (*closestDist)[node] >= (*closestDist)[source]+1 {
        if (*closestDist)[node] > (*closestDist)[source]+1 {
            (*closestDist)[node] = (*closestDist)[source] + 1
            (*parent)[node] = []string{}
        }
        (*parent)[node] = append((*parent)[node], source)
        if target == node {
            (*nodeVisited)[node] = true
        } else if !(*nodeVisited)[node] {
            dls(node, target, currentDepth-1, maxDepth, nearbyNode,
                parent, nodeVisited, childVisited, closestDist)
        }
    }
}
}
```

#### d) Pengulangan Pembangkitan Ruang Status dan Penelusuran Solusi

Pada bagian ini program akan melakukan loop dengan menambahkan kedalaman maksimum selama tujuan belum ditemukan.

```
if !nodeVisited[target] && maxDepth < 10 {
    return idsProccess(source, target, maxDepth+1, nearbyNode)
```

## 4.2 Penjelasan Tata Cara Penggunaan Program

### 4.2.1. Menjalankan Program

1. Clone repository Front-end dan Back-end
2. Masuk ke setiap folder dan buka di kedua terminal yang berbeda
3. Ketik command `npm install npm@latest -g` untuk install requirements
4. Untuk frontend masuk ke folder src, kemudian jalankan `npm run dev`
5. Hasil program dijalankan di server <http://localhost:3000>
6. Untuk backend masuk ke folder src
7. Jika tidak menggunakan docker, jalankan command `go run server.go`

8. Jika menggunakan docker, jalankan command `docker-compose build` dan `docker-compose up`


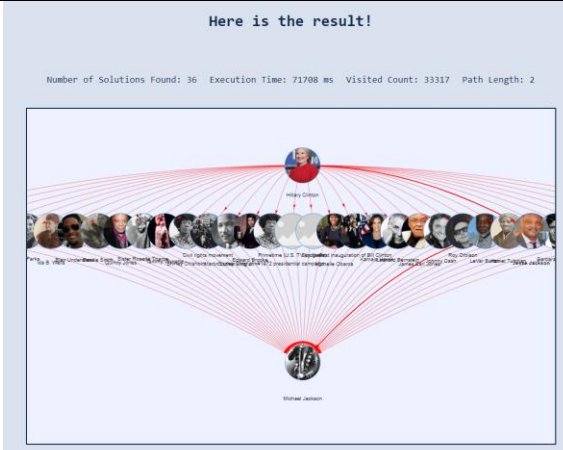
9. Server berjalan pada <http://localhost:8080>

### 4.2.2 Menggunakan Program



1. Input start page dan target page pada website
2. Input algoritma yang ingin digunakan dalam melakukan pencarian path
3. Setelah itu, path-path solusi WikiRace ditvisualisasikan pada graf

### 4.3 Hasil Pengujian

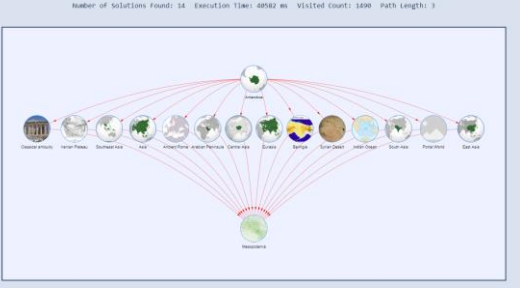

- Pengujian ke-1:

Artikel Awal	Hilary Clinton
Artikel Tujuan	Michael Jackson
Hasil Pengujian BFS	Hasil Pengujian IDS
	
Waktu Eksekusi: 84807 ms	Waktu Eksekusi: 71708 ms

- Pengujian ke-2:

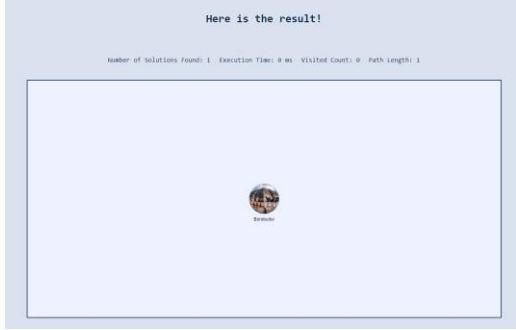
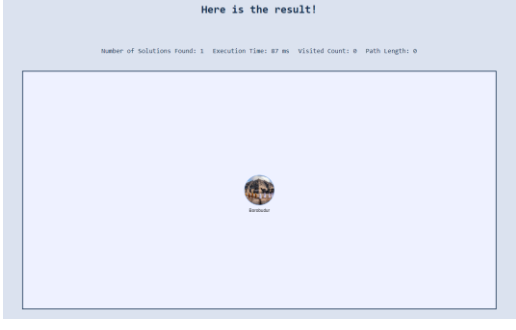
Artikel Awal	Car
Artikel Tujuan	Seat
Hasil Pengujian BFS	Hasil Pengujian IDS
	
Waktu Eksekusi: 20021 ms	Waktu Eksekusi: 22839 ms

- Pengujian ke-3:

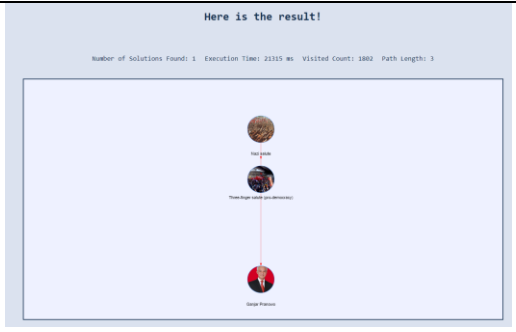
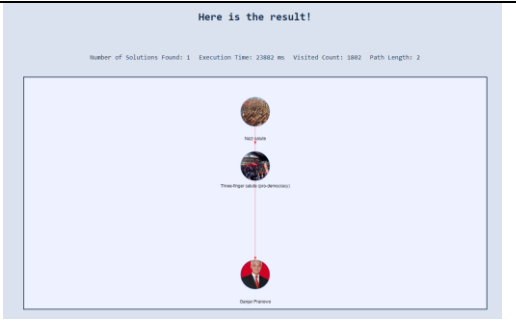
Artikel Awal	Anarctica
Artikel Tujuan	Mesopotamia
Hasil Pengujian BFS	Hasil Pengujian IDS
	
Waktu Eksekusi: 40582 ms	Waktu Eksekusi: 45302 ms

- Pengujian ke-4:

Artikel Awal	
Artikel Tujuan	
Hasil Pengujian BFS	Hasil Pengujian IDS

	
Waktu Eksekusi: 0 ms, (pada server 521.3 mikrosecond)	Waktu Eksekusi: 87 ms

- Pengujian ke-5:

Artikel Awal	Nazi Salute
Artikel Tujuan	Ganjar Pranowo
Hasil Pengujian BFS	Hasil Pengujian IDS
	
Waktu Eksekusi: 21315 ms	Waktu Eksekusi: 23882 ms

#### 4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian, dapat diperhatikan bahwa penggunaan algoritma BFS cenderung memiliki waktu eksekusi yang lebih cepat. Hal ini cukup sesuai menimbang kompleksitas waktu dari sebuah eksekusi dengan algoritma IDS dengan kedalaman n akan mirip dengan penjumlahan kompleksitas waktu BFS dengan kedalaman akhir 1 dengan BFS kedalaman 2 dan seterusnya sampai BFS kedalaman n. Walaupun begitu, pada pelaksanaan tes yang identik, waktu eksekusi tes-tes tersebut dapat bernilai berbeda, bahkan waktu eksekusi IDS bisa lebih

cepat dari BFS. Peristiwa tersebut terjadi dikarenakan dalam pelaksanaan *web scraping* akan muncul beberapa faktor yang menyebabkan ketidakstabilan kecepatan *scraping*. Diantara faktor-faktor tersebut yakni kecepatan internet, kemampuan perangkat, dan yang paling signifikan ialah kebijakan server yang di-*scrape* untuk membatasi pengaksesan dalam jangka waktu tertentu.

Pada pengujian waktu masing-masing komponen didapati bahwa bottleneck dari program ada pada proses request html untuk diparsing, yang berarti sering kali komponen pemrosesan harus menunggu. Berdasarkan beberapa kali percobaan didapati bahwa waktu yang dibutuhkan untuk request satu laman HTML sekitar 0.5 – 0.8 detik. Mengingat jumlah simpul yang dikunjungi yang bisa mencapai ratusan ribu, waktu *scraping* tersebut akan sangat menghambat pencarian. Oleh karena itu, dalam melakukan pencarian solusi dirasa diperlukan melakukan pendekatan multithreading agar dapat melakukan request html di saat yang bersamaan.



## **BAB V**

### **KESIMPULAN DAN SARAN**

Website WikiQuesters menggunakan algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS) dalam pencarian solusi untuk permainan WikiRace. Meskipun secara teoritis BFS cenderung memiliki waktu eksekusi yang lebih cepat, dalam pengujian praktisnya, waktu eksekusi bisa bervariasi. Faktor-faktor seperti kecepatan internet, kemampuan perangkat, dan kebijakan server dapat mempengaruhi waktu eksekusi. Selain pendekatan multithreading, terdapat saran untuk mempertimbangkan optimasi lain seperti caching data yang telah di-scrape untuk mengurangi jumlah permintaan ke server yang sama. Kemudian juga melakukan perbaikan design UI/UX agar lebih enak dipandang dan kenyamanan pengguna.

### **LAMPIRAN**

Link Repository :

- Front-End : [https://github.com/randyver/Tubes2\\_FE\\_WikiQuesters/](https://github.com/randyver/Tubes2_FE_WikiQuesters/)
- Back-End : [https://github.com/randyver/Tubes2\\_BE\\_WikiQuesters/](https://github.com/randyver/Tubes2_BE_WikiQuesters/)

Link Video : <https://youtu.be/IFaoY4O8fGY>

### **DAFTAR PUSTAKA**

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tubes2-Stima-2024.pdf>