

IF2211 Strategi Algoritma  
**PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN WIKIRACE**

**Laporan Tugas Besar 2**  
Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma  
pada Semester 2 (dua) Tahun Akademik 2023/2024



**Oleh**  
**Muhammad Davis Adhipramana**                   **13522157**  
**Auralea Alvinia Syaikha**                       **13522148**  
**Pradipta Rafa Mahesa**                       **13522162**

**Kelompok Wiki Scrapper**

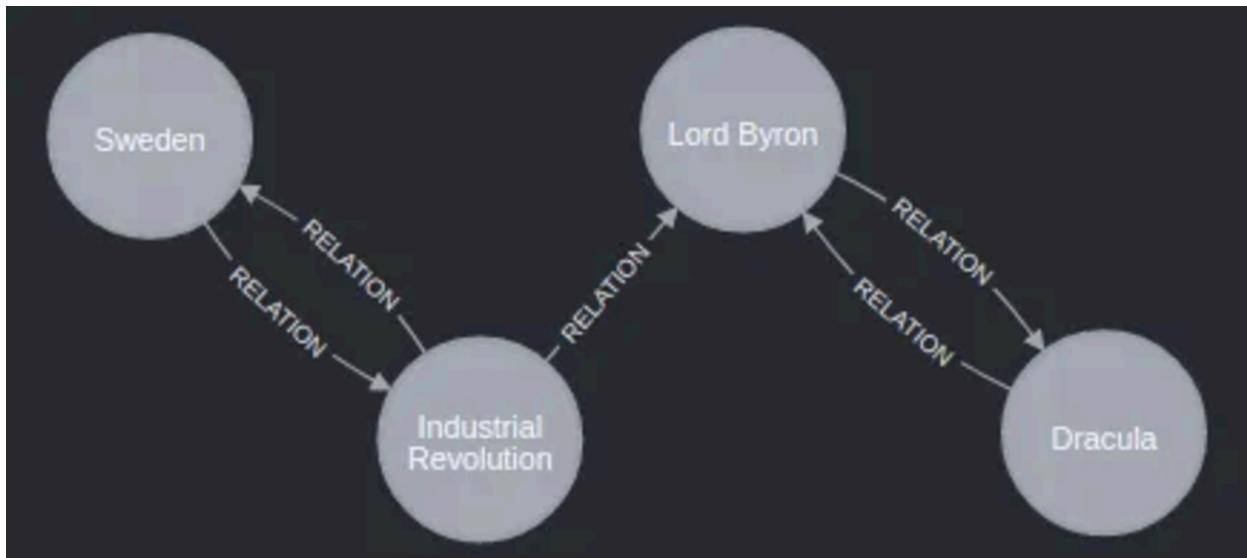
**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**BANDUNG**  
**2024**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB 1 DESKRIPSI TUGAS</b>	<b>3</b>
<b>BAB 2 LANDASAN TEORI</b>	<b>5</b>
2.1 Graf	5
2.2 Algoritma Breadth-First Search (BFS)	5
2.3 Algoritma Iterative Deepening Search (IDS)	6
2.4 Apikasi Web WikiRace	7
<b>BAB 3 ANALISIS PEMECAHAN MASALAH</b>	<b>8</b>
3.1 Langkah-langkah Pemecahan Masalah	8
3.1.1 Langkah-langkah Pemecahan Masalah menggunakan Algoritma BFS (Breadth-First Search)	
8	
3.1.2 Langkah-langkah Pemecahan Masalah menggunakan Algoritma IDS (Iterative Deepening Search)	9
3.2 Proses Pemetaan Masalah	9
3.2.1 Proses Pemetaan Masalah menjadi Elemen Algoritma BFS (Breadth-First Search)	9
3.2.2 Proses Pemetaan Masalah menjadi Elemen Algoritma IDS (Iterative Deepening Search)	10
3.3 Fitur Fungsional & Arsitektur Aplikasi Web yang Dibangun	10
3.3.1 Fitur Fungsional	10
3.3.2 Arsitektur Aplikasi Web yang Dibangun	11
<b>BAB 4 IMPLEMENTASI DAN PENGUJIAN</b>	<b>11</b>
4.1 Spesifikasi Teknis Program	11
4.1.1 prioqueue.go (Struktur Data)	11
4.1.2 links.go (Fungsi)	15
4.1.3 bfs.go (Fungsi)	16
4.1.4 main.go (Prosedur)	17
4.2 Tata Cara Penggunaan Program	17
4.3 Hasil Pengujian	20
4.4 Analisis Pengujian	23
<b>BAB 5 KESIMPULAN DAN SARAN</b>	<b>24</b>
5.1 Kesimpulan	24
5.2 Saran	24
<b>LAMPIRAN</b>	<b>25</b>
<b>DAFTAR PUSTAKA</b>	<b>26</b>

# BAB 1 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.



## Spesifikasi Tugas Besar 2

- Buatlah program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace.
- Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan.
- Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms).
- Program cukup mengeluarkan salah satu rute terpendek saja (cukup satu rute saja, tidak perlu seluruh rute kecuali mengerjakan bonus).
- Program berbasis web, sehingga perlu dibuat front-end dan back-end (tidak perlu di-deploy).
- Repository front-end dan back-end diizinkan untuk dipisah maupun digabung dalam repository yang sama.
- Program wajib dapat mencari rute terpendek kurang dari 5 menit untuk setiap permainan.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus. Akan tetapi, anggota kelompok tidak boleh sama dengan anggota kelompok pada tugas-tugas Strategi Algoritma sebelumnya.

- Program harus mengandung komentar yang jelas serta mudah dibaca.
- Mahasiswa dilarang menggunakan kode program yang didapatkan dari internet (alasan menggunakan kakas seperti GitHub Copilot tidak diterima). Mahasiswa harus membuat program sendiri, diperbolehkan untuk belajar dari program yang sudah ada.
- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.
- Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.
- Setiap kelompok harap mengisi nama kelompok dan anggotanya pada pranala [bit.ly/kelompoktubes2stima24](http://bit.ly/kelompoktubes2stima24), paling lambat Kamis, 4 April pukul 22.11 WIB.
- Diwajibkan untuk memilih asisten meskipun tidak melakukan asistensi, karena asisten yang dipilih akan menjadi asisten saat asistensi (opsional) dan demo tugas besar. Pemilihan asisten dapat dilakukan pada link berikut, paling lambat Kamis, 4 April pukul 22.11 WIB.
- Program disimpan dalam repository yang bernama Tubes2\_NamaKelompok (bila digabung) dan Tubes2\_FE/BE\_NamaKelompok (bila dipisah) dengan nama kelompok sesuai dengan yang di sheets diatas. Berikut merupakan struktur dari isi repository tersebut:
  - a. Folder src berisi program yang dapat dijalankan
  - b. Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf
  - c. README untuk tata cara penggunaan yang minimal berisi:
    - i. Penjelasan singkat algoritma IDS dan BFS yang diimplementasikan
    - ii. Requirement program dan instalasi tertentu bila ada
    - iii. Command atau langkah-langkah dalam meng-compile atau build program
    - iv. Author (identitas pembuat)

## BAB 2 LANDASAN TEORI

### 2.1 Graf

Dalam ilmu komputer, graf adalah struktur data yang terdiri dari kumpulan simpul (nodes) yang terhubung oleh sisi (edges). Graf sering digunakan untuk merepresentasikan berbagai situasi yang melibatkan hubungan antar entitas. Graf terdiri dari dua elemen utama:

- Simpul

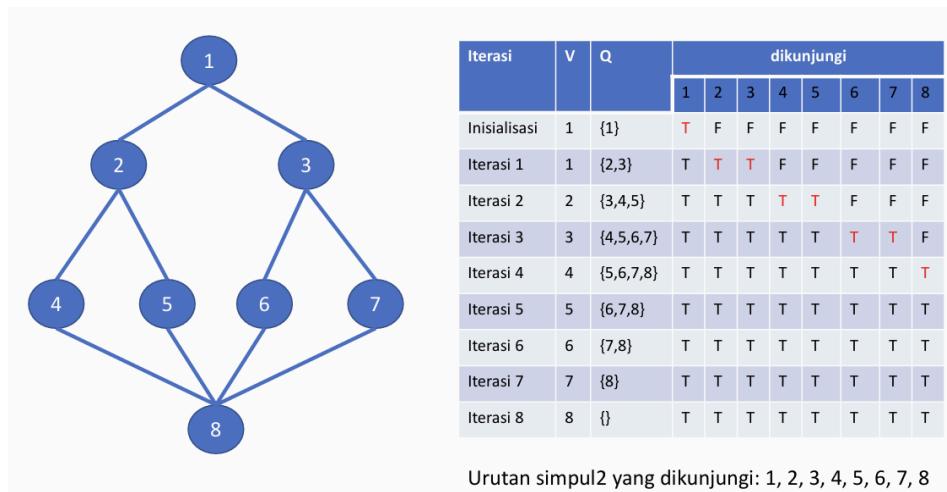
Simpul mewakili entitas atau objek dalam graf.

- Sisi

Sisi menghubungkan pasangan simpul dan mewakili relasi antar entitas

Graf dapat digunakan dalam berbagai domain, seperti ilmu komputer, matematika, sains sosial, dan lainnya. Penggunaan graf mencakup pemodelan jaringan komputer, rute perjalanan, analisis jaringan, perencanaan proyek, dan sebagainya. Dalam konteks algoritma penelusuran graf, dua teknik yang umum digunakan adalah Depth-First Search (DFS) dan Breadth-First Search (BFS).

### 2.2 Algoritma *Breadth-First Search (BFS)*



Sumber: [Homepage Rinaldi Munir \(itb.ac.id\)](http://Homepage.Rinaldi.Munir(itb.ac.id))

Algoritma Breadth-First Search (BFS) adalah salah satu teknik penelusuran graf yang digunakan untuk mengunjungi semua simpul pada tingkat kedalaman tertentu

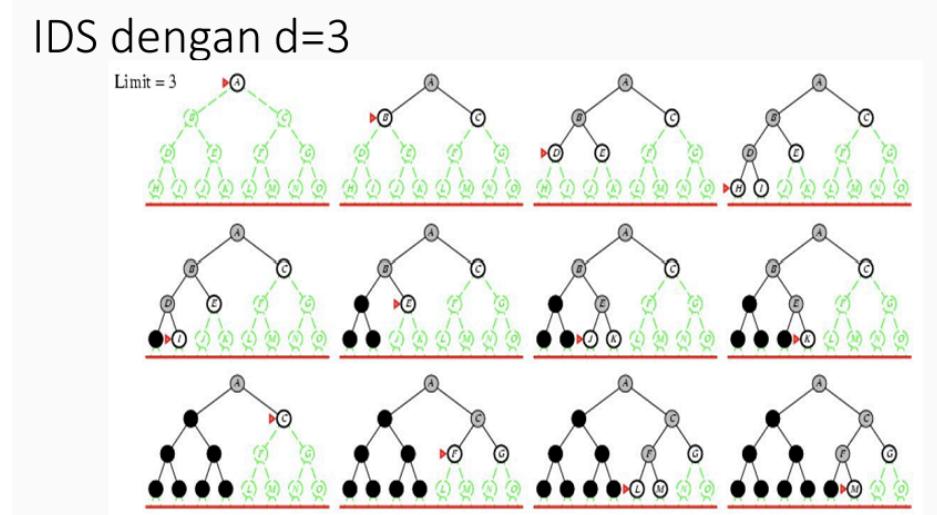
sebelum melanjutkan ke tingkat kedalaman berikutnya. Prinsip utama dari BFS adalah menjelajahi secara melebar, artinya algoritma ini akan mengunjungi semua simpul yang terhubung pada tingkat kedalaman saat ini sebelum melanjutkan ke tingkat kedalaman berikutnya.

Implementasi BFS biasanya menggunakan struktur data antrian (queue). Proses dimulai dengan memasukkan simpul awal ke dalam antrian. Selanjutnya, algoritma akan mengambil simpul pertama dari antrian, mengunjungi semua simpul tetangga yang belum dikunjungi, dan memasukkan simpul-simpul tersebut ke dalam antrian. Langkah ini akan terus berlanjut sampai antrian kosong, yang menandakan bahwa semua simpul yang dapat dicapai dari simpul awal telah dikunjungi.

Salah satu keunggulan utama dari BFS adalah kemampuannya untuk menemukan jalur terpendek (dalam jumlah tepat dari langkah atau sisi) antara dua simpul dalam graf tak berbobot atau graf berbobot non-negatif. Hal ini karena BFS mengunjungi simpul-simpul berdasarkan jarak (tingkat kedalaman) dari simpul awal. Selain itu, BFS juga berguna dalam pencarian lintasan dan penemuan struktur graf seperti komponen terhubung.

Kompleksitas waktu BFS adalah  $O(V + E)$ , di mana  $V$  adalah jumlah simpul (nodes) dan  $E$  adalah jumlah sisi (edges) dalam graf. BFS sangat berguna dalam berbagai aplikasi seperti pemodelan jaringan, navigasi dalam permainan, pemrosesan bahasa alami, dan optimasi jaringan. Pemahaman yang baik tentang prinsip dan implementasi BFS akan membantu dalam menyelesaikan berbagai masalah terkait graf dengan efisien.

### 2.3 Algoritma *Iterative Deepening Search* (IDS)



Sumber: [Homepage Rinaldi Munir \(itb.ac.id\)](http://Homepage.Rinaldi.Munir(itb.ac.id))

Algoritma Iterative Deepening Search (IDS) adalah metode penelusuran graf yang menggabungkan konsep dari Depth-First Search (DFS) dan Breadth-First Search (BFS) untuk mencari jalur terpendek dalam ruang pencarian. IDS bekerja dengan melakukan serangkaian penelusuran DFS dengan batasan kedalaman yang semakin meningkat hingga mencapai solusi yang diinginkan atau mencapai kedalaman maksimum yang ditentukan.

Implementasi IDS dimulai dengan melakukan DFS dengan batasan kedalaman 1. Jika solusi tidak ditemukan pada kedalaman tersebut, maka dilanjutkan dengan DFS batasan kedalaman 2, kemudian batasan kedalaman 3, dan seterusnya. Proses ini berlanjut hingga solusi ditemukan atau batasan kedalaman maksimum tercapai.

IDS memiliki keunggulan utama dalam hal efisiensi. Meskipun IDS secara konseptual mirip dengan DFS yang terbatas pada kedalaman tertentu, IDS dapat menemukan solusi dengan kompleksitas waktu yang lebih baik dalam beberapa kasus. Hal ini terjadi karena IDS mencoba menjelajahi ruang pencarian secara bertahap, mulai dari kedalaman yang rendah hingga kedalaman yang lebih dalam, sehingga lebih efisien dalam menemukan solusi pada graf yang sangat dalam atau memiliki cabang-cabang yang sangat panjang.

Meskipun IDS dapat menjadi alternatif yang efisien dalam mencari solusi pada ruang pencarian yang besar, algoritma ini tetap memiliki kompleksitas waktu yang bergantung pada kedalaman maksimum yang ditentukan. Selain itu, implementasi IDS memerlukan penyimpanan tambahan untuk melacak jalur pencarian pada setiap iterasi, meskipun hanya pada kedalaman tertentu. Meskipun demikian, IDS merupakan algoritma yang kuat dan dapat diaplikasikan dalam berbagai masalah yang melibatkan pencarian solusi optimal dalam ruang pencarian yang kompleks.

## 2.4 Apikasi Web *WikiRace*

Aplikasi web Wikirace merupakan sebuah permainan atau kompetisi daring di mana peserta berusaha untuk menavigasi dari satu halaman Wikipedia ke halaman lainnya dengan menggunakan tautan internal Wikipedia sebagai jalur navigasi. Dalam konteks permainan Wikirace, penggunaan algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS) dapat memberikan wawasan tentang bagaimana peserta dapat menemukan jalur terpendek antara dua halaman Wikipedia yang ditentukan.

Dalam Wikirace, BFS dapat diimplementasikan dengan memperlakukan setiap halaman Wikipedia sebagai simpul. Setiap tautan internal dari suatu halaman akan dianggap sebagai sisi yang menghubungkan simpul-simpul tersebut. Mulai dari halaman awal (simpul awal), BFS akan secara sistematis menjelajahi tautan internal ke halaman-halaman terhubung lainnya, memeriksa satu level (kedalaman) pada suatu waktu sebelum melanjutkan ke level berikutnya.

Dalam Wikirace, IDS akan memulai dengan DFS batasan kedalaman 1, kemudian bertahap meningkatkan batasan kedalaman jika solusi tidak ditemukan. Proses ini terus diulang hingga jalur terpendek ditemukan atau batasan kedalaman maksimum tercapai.

## BAB 3 ANALISIS PEMECAHAN MASALAH

### 3.1 Langkah-langkah Pemecahan Masalah

#### 3.1.1 Langkah-langkah Pemecahan Masalah menggunakan Algoritma BFS (*Breadth-First Search*)

Dalam implementasi algoritma BFS, kami menggunakan pendekatan *prioqueue* dengan skala prioritas yang ditentukan oleh jarak string antara tautan target dan tautan saat ini dengan langkah-langkah seperti berikut.

1. Membandingkan string target URL dengan string URL yang sedang diakses menggunakan algoritma *Levenshtein distance*. Algoritma *Levenshtein distance* digunakan untuk mengukur seberapa mirip dua buah string, dalam hal ini tautan target dan tautan saat ini. Algoritma ini mengkalkulasi seberapa mudah suatu string diubah menjadi string target.
2. Menghitung jarak Levenshtein antara string dari tautan saat ini dan tautan target. Semakin rendah jaraknya, maka semakin tinggi prioritasnya dalam antrian.
3. *Path* yang telah diambil dalam penelusuran akan dilacak dengan menyimpan *path* tersebut dalam sebuah *map* yang memiliki *key* berupa *string* dan *value* berupa *list of string*. *Key* tersebut adalah link yang nantinya akan di-*scrap* nantinya.
4. Setiap kali sebuah URL di-*scrap*, link yang pertama kali ditemukan pada URL tersebut akan disimpan ke dalam sebuah *map* yang memiliki *key* berupa *string* dan *value* berupa *boolean*. Hal ini dibuat dengan tujuan agar link yang ditemukan tersebut tidak diakses kembali.
5. Proses ini akan terus dilakukan hingga saat melakukan scraping target url dapat ditemukan, dan program akan mengembalikan *Path* yang disimpan

### **3.1.2 Langkah-langkah Pemecahan Masalah menggunakan Algoritma IDS (*Iterative Deepening Search*)**

Dalam implementasi algoritma IDS, kami menggunakan pendekatan looping dari algoritma DLS dalam dalam setiap depth dari depth = 2 sampai maxDepth yang digabungkan dengan concurrency golang Langkah-langkah dari algoritma IDS adalah seperti berikut.

1. Melakukan loop DLS dimulai dari 2 hingga maxDepth
2. Dari page URL awal (URL1) cari URL yang paling pertama muncul (URL2) pada page tersebut apabila URL tersebut adalah target maka selesai, apabila tidak lanjut kepada page URL2 lalu mencari URL pertama yang muncul pada page tersebut (URL3).
3. Hal tersebut diulang sebanyak jumlah depth, apabila depth sudah habis maka harus melakukan backtrack lalu mencari URL terdekat selanjutnya, yang dalam contoh ini apabila depth habis pada URL3 adalah URL kedua yang muncul pada page URL2 (URL4)
4. Lakukan hal tersebut hingga semua URL pada page URL2 sudah dijelajahi
5. Mengulangi hingga semua link pada page URL1 telah dijelajah atau ditemukannya target

## **3.2 Proses Pemetaan Masalah**

### **3.2.1 Proses Pemetaan Masalah menjadi Elemen Algoritma BFS (*Breadth-First Search*)**

1. Halaman Wikipedia dimodelkan sebagai graf berarah dengan simpul menyatakan halaman Wikipedia dan sisi menyatakan link ke halaman Wikipedia.
2. Penelusuran dimulai dari halaman awal.
3. Ekstrak semua link yang terdapat pada halaman tersebut dan menambahkannya ke dalam antrian untuk dikunjungi selanjutnya.
4. Jika pada halaman awal tidak ditemukan link target, maka penelusuran akan lanjut ke kedalaman selanjutnya, dan akan melakukan proses berulang seperti berikut:
  - a. Ambil link pertama dari antrian
  - b. Kunjungi halaman web yang sesuai dengan link tersebut
  - c. Ekstrak semua link dari halaman tersebut dan tambahkan ke dalam antrian
5. Jika halaman target telah ditemukan, maka proses ini akan berhenti.

### **3.2.2 Proses Pemetaan Masalah menjadi Elemen Algoritma IDS (Iterative Deepening Search)**

1. Halaman Wikipedia dimodelkan sebagai graf berarah dengan simpul menyatakan halaman Wikipedia dan sisi menyatakan link ke halaman Wikipedia.
2. Penelusuran dimulai dari halaman awal.
3. Pengecekan akan dilakukan pada link pertama yang ditemukan pada halaman awal.
4. Jika, link pertama tersebut bukanlah link target, maka penelusuran akan lanjut ke kedalaman selanjutnya hingga mencapai batas kedalaman yang telah ditentukan.
5. Jika sudah mencapai batas dan link target belum ditemukan, maka penelusuran akan lanjut ke simpul tetangga.
6. Proses akan terus berulang hingga link target ditemukan.

## **3.3 Fitur Fungsional & Arsitektur Aplikasi Web yang Dibangun**

### **3.3.1 Fitur Fungsional**

Pada aplikasi web yang dibangun, terdapat beberapa fitur fungsional sebagai berikut.

#### **1. Mesin Pencari untuk Pencarian Konten**

Mesin pencari merupakan salah satu fitur yang krusial dalam memungkinkan pengguna untuk menemukan konten secara cepat dan efisien. Dengan adanya mesin pencari, pengguna dapat dengan mudah mencari topik atau informasi yang mereka butuhkan tanpa harus menghabiskan waktu untuk menelusuri per kata. Fitur ini membantu meningkatkan pengalaman pengguna dengan memberikan akses yang cepat dan langsung ke konten yang relevan.

#### **2. Penggunaan API Wikipedia untuk Autocomplete**

Penggunaan API Wikipedia untuk fitur Autocomplete dilakukan untuk meningkatkan pengalaman pengguna. Dengan adanya fitur ini, saat pengguna mulai mengetik dalam kotak pencarian, mereka akan diberikan rekomendasi atau sugesti otomatis berdasarkan entri Wikipedia yang relevan. Hal ini tidak hanya mempercepat proses pencarian, tetapi juga membantu pengguna dalam menemukan topik yang tepat dengan lebih mudah.

### 3. Navigasi yang Intuitif antara Halaman dan Kategori Konten

Navigasi yang intuitif adalah kunci untuk memastikan pengguna dapat dengan mudah menjelajahi berbagai halaman dan kategori konten yang tersedia dalam aplikasi web. Dengan menyediakan navigasi yang jelas dan terstruktur, pengguna dapat dengan cepat beralih antara berbagai halaman atau kategori, menjadikan pengalaman mereka lebih lancar dan efisien. Ini juga membantu pengguna untuk menemukan lebih banyak konten yang mungkin menarik minat mereka tanpa kesulitan dalam navigasi.

### 3.3.2 Arsitektur Aplikasi Web yang Dibangun

#### 1. FrontEnd

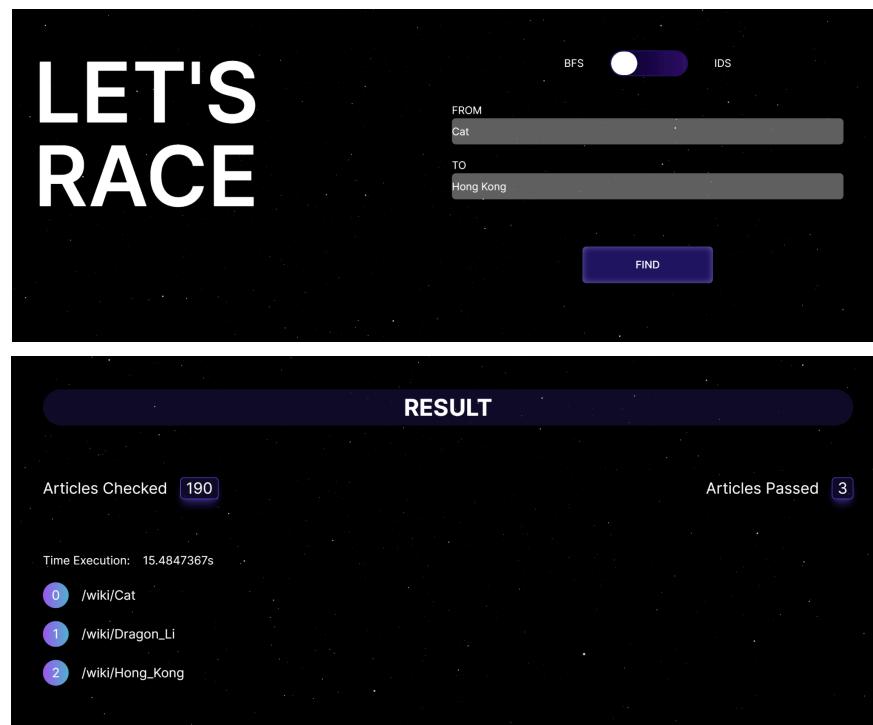
Menggunakan framework frontend seperti Next.js karena routing yang dinamis. Kami juga menggunakan Tailwind CSS untuk *styling* karena kustomisasinya yang mudah.

#### 2. BackEnd

Menggunakan Golang karena dalam Golang terdapat library yang bernama goroutine yang mudah digunakan dan cepat.

### 3.4 Contoh Ilustrasi Kasus

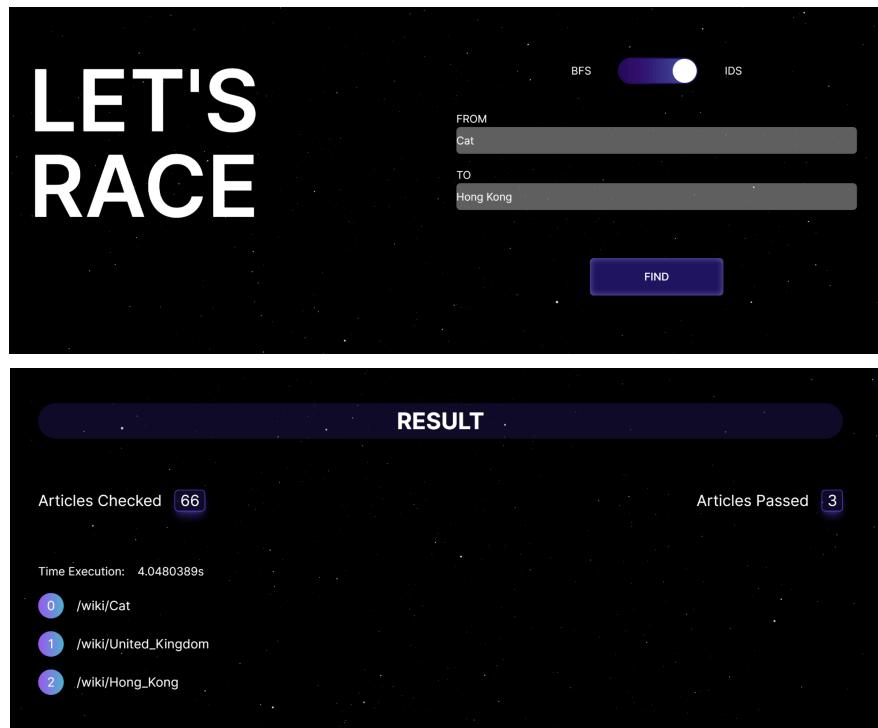
#### 1. Algoritma *Breadth-First Search* (BFS)



(Kasus ‘Cat’ ke ‘Hong Kong’ dengan = 2)

Dalam penerapan algoritma BFS, kami menggunakan metode prioqueue dengan skala prioritas yang ditentukan oleh jarak string antara tautan target dan tautan saat ini. Langkah-langkah yang dilakukan adalah sebagai berikut: pertama, membandingkan string URL target dengan string URL saat ini menggunakan algoritma Levenshtein distance, yang berguna untuk menentukan seberapa mirip kedua string tersebut. Algoritma ini mengukur seberapa mudah suatu string dapat diubah menjadi string target. Selanjutnya, kami menghitung jarak Levenshtein antara string dari tautan saat ini dan tautan target. Semakin kecil jaraknya, semakin tinggi prioritasnya dalam antrian. Kami melacak path yang telah diambil dalam penelusuran dengan menyimpannya dalam sebuah map, di mana key-nya adalah string tautan yang akan di-scraped nantinya, dan nilai-nya adalah daftar string. Ketika melakukan proses scraping, setiap kali sebuah URL di-scraped, link pertama yang ditemukan pada URL tersebut akan disimpan dalam map dengan key berupa string dan value berupa boolean, untuk menghindari pengaksesan kembali terhadap link yang sama. Proses ini akan terus dilakukan hingga URL target dapat ditemukan saat melakukan scraping, dan program akan mengembalikan Path yang telah disimpan.

## 2. Algoritma Iterative Deepening Search (IDS)



(Kasus ‘Cat’ ke ‘Hong Kong’ dengan  $degree = 2$ )

Dalam penerapan algoritma IDS, kami menggabungkan pendekatan looping dari algoritma DLS pada setiap kedalaman dari 2 hingga maxDepth, dengan memanfaatkan konkurensi dalam bahasa pemrograman Golang. Langkah-langkah algoritma IDS adalah sebagai berikut: pertama, kami melakukan looping dari kedalaman 2 hingga maxDepth. Dari URL awal (URL1), kami mencari URL pertama yang muncul (URL2) pada halaman tersebut. Jika URL tersebut merupakan target, pencarian berakhir; jika tidak, kami melanjutkan ke halaman URL2 dan mencari URL pertama yang muncul pada halaman tersebut (URL3). Proses tersebut diulang sebanyak depth yang ditentukan; jika depth habis, kami melakukan backtrack dan mencari URL terdekat berikutnya. Misalnya, jika depth habis pada URL3, kami mencari URL kedua yang muncul pada halaman URL2 (URL4). Proses ini terus diulang hingga semua URL pada halaman URL2 telah dijelajahi. Kami mengulangi langkah-langkah ini hingga semua tautan pada halaman URL1 telah dijelajahi atau target telah ditemukan.

## BAB 4 IMPLEMENTASI DAN PENGUJIAN

### 4.1 Spesifikasi Teknis Program

#### 4.1.1 prioqueue.go (Struktur Data)

File prioqueue.go mengimplementasikan struktur data antrian prioritas (priority queue) dalam bahasa Go. Struktur data ini digunakan untuk mengatur elemen-elemen dengan prioritas tertentu. Dalam file ini, terdapat beberapa tipe data yang didefinisikan. Target adalah tipe data yang merepresentasikan target URL.



```
1 import (
2     "fmt"
3     "log"
4     "os"
5     "strconv"
6     "sync"
7     "unicode/utf
8 ) 8"
9
10
11 // Target Url Dat
12 type Target struct{
13     name string
14 }
15
16 // Item data for compariso
17 type Item struct {
18     key    string
19     priority int
20     depth int
21 }
22
23 // Queue
24 type PriorityQueu  struct {
25     items []Item
26 }
27
28 // Priority Queu
29 type Prioqueue struct {
30     treshold int // Used to preserve some Enqueue before declining prioritie
31     target Targets
32     pq    PriorityQueu
33     sync.Mutex
34 }
35
36
37 // Length of the Priority Queu
38 func (pq *Prioqueue) Length() int {
39     return len(pq.pq.items)
40 }
41
42 // Defined the target url and insert into the prioqueue
43 func (pq *Prioqueue) ConstructTarge (target string) {
44     var t Target    t
45     t.name = target
46     pq.target = t
47 }
```

```

1 // Initiate Prioqueue, set target as the url target
2 func (pq *Prioqueue) Init(target string) {
3     pq.Lock()
4     defer pq.Unlock()
5     pq.pq.items = make([]Item, 0)
6     pq.ConstructTarget(target)
7     pq.threshold = 10 // Limit of Enqueue without prioritization
8             s
9 }
10
11 // Compare Two strings using Levenshtein distance algorithm
12 func StringCompare(s1, s2 string) int {
13     minlengthThreshold := 32
14     if len(s1) == 0 {
15         return utf8.RuneCountInString(s2)
16     }           g
17     if len(s2) == 0 {
18         return utf8.RuneCountInString(s1)
19     }           g
20     if s1 == s2 {
21         return 0
22     }
23     if len(s1) > len(s2) {
24         s1, s2 = s2, s1
25     }
26
27     ls1 := len(s1)
28     ls2 := len(s2)
29     var x []int
30     if ls1+1 > minlengthThreshold {
31         x = make([]int, ls1+1)
32     } else {
33         x = make([]int, minlengthThreshold )
34         x = x[:ls1+1]   d
35     }
36     for i := 1; i < len(x); i++ {
37         x[i] = int(i)
38     }
39     for i := 1; i <= ls2; i++ {
40         prev := int(i)
41         for j := 1; j <= ls1; j++ {
42             current := x[j-1] // match
43             if s2[i-1] != s1[j-1] {
44                 current = min(min(x[j-1]+1, prev+1), x[j]+1)
45             }
46             x[j-1] = prev
47             prev = current
48         }
49         x[ls1] = prev
50     }
51     return int(x[ls1])
52 }
53
54 // Return the lowest of two integer
55 func min(a, b int) int {
56     if a < b {
57         return a
58     }
59     return b
60 }
61
62 // Determine the Priority of a key
63 // The Lower the distance by StringCompare, the higher the priority
64 func (pq *Prioqueue) priorityDecision(key string) int {
65     return StringCompare(key, pq.target.name)
66 }           e
67

```

```

1 // Keeping the order of the queu
2 #! Prioritize by depth, then prioritize by priorit
3 func(pq *Prioqueue) ReSortLis (item Item){
4     id := 0          t
5     for i:= 0; i < pq.Length();i++{
6         temp := pq.pq.items[i]
7         if(item.depth > temp.depth){
8             id = i + 1
9             continue
10            }
11
12         if(item.depth == temp.depth){
13             if item.priority < temp.priority {
14                 id = i
15                 break
16             } else {
17                 id = i + 1 // Move to the next positio
18                 continue n
19             }
20         } else if(item.depth < temp.depth){
21             id = i
22             break
23         }
24     }
25
26     if(pq.Length() > 0){
27         pq.pq.items = append(pq.pq.items[:id], append([]Item{item}, pq.pq.items[id:]...)...)
28     }else{
29         pq.pq.items = append([]Item{}, item)
30     }
31 }
32
33 // Write Prioqueue data onto a fil
34 func writeFilePriou (filename string, data []Item, parent string) {
35     file, err := os.OpenFile(filename, os.O_APPEND|os.O_WRONLY|os.O_CREATE, 0644)
36     if err != nil {
37         log.Fatal("Error opening fil , err")
38     } e:
39     defer file.Close()
40
41     for _, link := range data {
42         // Write each link to the fil
43         e, err := file.WriteString("Link: " +link.key + " Depth : "+ strconv.Itoa(link.depth) +
44 " Priority: "+ strconv.Itoa(link.priority)+ " Parent : " + parent + "\n")
45         if err != nil {
46             log.Fatal("Error writing to fil , err")
47         } e:
48
49         //fmt.Println("Links appended to", filename
50     } e)
51
52 // Enqueue a new key and its depth to the queu
53 func (pq *Prioqueue) Enqueue(key string, depth int) {
54     pq.Lock()
55     defer pq.Unlock()
56     if pq.pq.items == nil {
57         pq.pq.items = make([]Item, 0)
58     }
59
60     removeWik := key[5:]
61     priority := pq.priorityDecisio (removeWik )
62     n           i
63     if(priority == 99){
64         return
65     }
66     item := Item{key, priority, depth}
67
68     //Boundary se
69     if(pq.Length() > 5000 ){
70         if(priority < 20){
71             pq.ReSortLis (item);
72         }else{ t
73             return;
74         }
75     }else {
76         pq.ReSortLis (item);
77     } t
78 }
79

```

```
1 //Dequeue The Prio List
2 //Return The Link, Priority, and Depth of the most prioritized Item
3 func (pq *Priqueue) Dequeue() (string,int,int) {
4     pq.Lock()
5     defer pq.Unlock()
6     if len(pq.pq.items) == 0 {
7         return "",99,99
8     }
9     root := pq.pq.items[0]
10    pq.pq.items = pq.pq.items[1:]
11    return root.key,root.priority,root.depth
12 }
13 /*
14 Display Data of Priqueue
15 @status : Full (Display Full Data)
16 $status : ListOnly (Display Only The List)
17 $status: Length (Display Only the Length)
18 */
19 func (pq *Priqueue) Log(status string){
20
21
22     if(status == "full"){
23         fmt.Println("=====PRIQUEUE DATA====")
24         fmt.Println("#Data: ")
25         fmt.Println(pq.pq.items)
26         fmt.Println("Length: ")
27         fmt.Println(len(pq.pq.items))
28         fmt.Println()
29
30     }else if(status == "ListOnly"){
31         fmt.Println("==y==PRIQUEUE DATA==")
32         fmt.Println("#Data: ")
33         fmt.Println(pq.pq.items)
34     }else{
35         fmt.Println("=====PRIQUEUE DATA====")
36         fmt.Println("#Length: ")
37         fmt.Println(len(pq.pq.items))
38     }
39 }
```

#### 4.1.2 links.go (Fungsi)

Fungsi	Keterangan
<pre>func extractLinks(doc *goquery.Document, visited SafeMap, targeturl string, links *[]string, start, end int) bool</pre>	Fungsi ini digunakan untuk mengekstrak tautan dari bagian dokumen. Fungsi ini mencari elemen dengan id "mw-content-text" dan mengekstrak semua tautan di dalamnya. Jika tautan cocok dengan URL target, fungsi akan mengembalikan true.
<pre>func ignoreLink(link string) bool</pre>	Fungsi ini digunakan untuk mengecek apakah tautan harus diabaikan atau tidak. Tautan yang diabaikan adalah tautan yang mengandung karakter "%" atau tautan yang diawali dengan string tertentu seperti "/wiki/File:", "/wiki/Help:", dll.
<pre>func getListOfLinksMult(targeturl, url string, visited map[string]bool, httpClient *http.Client) ([]string, bool)</pre>	Fungsi ini digunakan untuk mendapatkan daftar tautan dari URL tertentu. Fungsi ini mengambil URL, mengambil halaman web, mem-parsing HTML, dan mengekstrak semua tautan. Jika tautan cocok dengan URL target, fungsi akan mengembalikan true.
<pre>func getListOfLinks1(targeturl, url string, visited SafeMap) ([]string, bool)</pre>	Fungsi ini digunakan untuk mendapatkan daftar tautan dari URL tertentu. Fungsi ini mengambil URL, dan mengekstrak semua tautan. Jika tautan cocok dengan URL target, fungsi akan mengembalikan true. Memanfaatkan concurrency dengan membagi page menjadi 4 kuadran dan mencari link dari masing masing kuadran lalu menggabungkannya dan mengembalikannya
<pre>func isin(link string, array []string) bool</pre>	Fungsi ini digunakan untuk mengecek apakah suatu string ada dalam array string atau tidak. Fungsi ini membagi array menjadi empat bagian dan memproses setiap bagian secara paralel. Jika string ditemukan dalam salah satu bagian, fungsi akan mengembalikan true.

### 4.1.3 bfs.go (Fungsi)

Fungsi	Keterangan
<pre>func BFSWithPriorityQueue(startURL, targetURL string, counter *int) []string</pre>	Fungsi ini digunakan untuk mencari jalur terpendek dari URL awal (startURL) ke URL tujuan (targetURL). Fungsi ini juga menghitung jumlah URL yang telah dikunjungi (counter). Fungsi ini menggunakan beberapa goroutine (sejenis thread di Go) untuk melakukan pencarian secara paralel.
<pre>func FindCorrectPath(currentUrl string, queue [][]string) []string</pre>	Fungsi ini digunakan untuk mencari jalur yang benar dari antrian jalur yang ada. Fungsi ini menerima URL saat ini (currentUrl) dan antrian jalur (queue) sebagai parameter, dan mengembalikan jalur yang mengandung URL saat ini sebagai elemen terakhir.
<pre>func RemovePathFromQueue(queue [][]string, deleted []string) [][]string</pre>	Fungsi ini digunakan untuk menghapus jalur tertentu dari antrian jalur. Fungsi ini menerima antrian jalur (queue) dan jalur yang akan dihapus (deleted) sebagai parameter, dan mengembalikan antrian jalur baru setelah jalur yang dihapus telah dihilangkan.

### 4.1.4 main.go (Prosedur)

Fungsi	Keterangan
<pre>func postDataHandler(w http.ResponseWriter, r *http.Request)</pre>	Fungsi ini adalah handler untuk permintaan HTTP POST.
<pre>func getDataHandler(w http.ResponseWriter, r *http.Request)</pre>	Fungsi getDataHandler adalah handler untuk permintaan HTTP GET. Fungsi ini memeriksa apakah metode permintaan adalah GET.

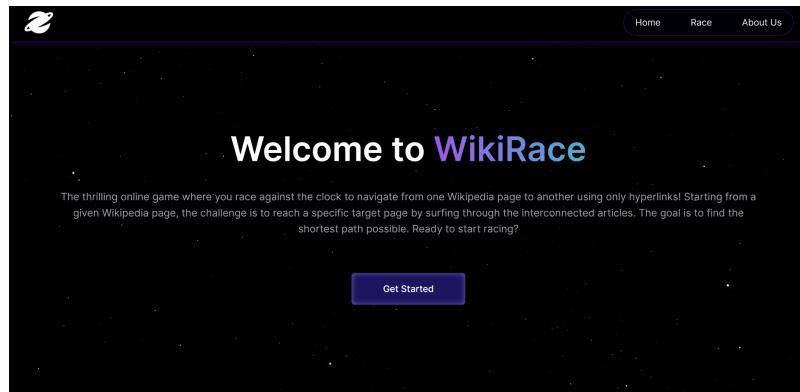
<pre>func processData()</pre>	Fungsi processData digunakan untuk memproses data yang diterima dari channel UrlData. Fungsi ini melakukan pencarian jalur antara dua URL menggunakan algoritma BFS atau IDS, tergantung pada nilai data.Algorithm.
<pre>func main()</pre>	Fungsi main adalah fungsi utama dalam program Go yang akan dieksekusi pertama kali saat program dijalankan

#### 4.1.5 ids.go (Fungsi)

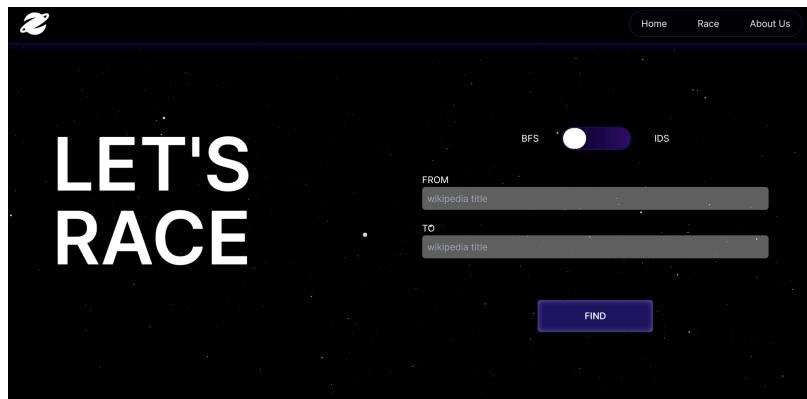
Fungsi	Keterangan
<pre>func IDS(startURL, targetURL string, depthLimit int, counter *int) []string</pre>	Fungsi IDS adalah fungsi yang mengembalikan nil apabila depthLimit <0 {startURL} apabila startURL = targetURL. Apabila tidak tempat mengloop penggunaan DLS berdasarkan depth mulai dari 2 hingga depthLimit. Sebelum loop mengembalikan {startURL} apabila startURL = targetURL.
<pre>func DLS(currentURL, targetURL string, depthLimit int, visited *SafeMap, mutex *sync.Mutex, counter *int) [] string</pre>	Fungsi DLS adalah fungsi utama penerapan algoritma Depth Limited Search dari currentURL ke targetURL dengan depth = depthLimit dengan memanfaatkan getlistsoflink1 untuk mendapatkan links yang terdapat pada suatu page. Mengembalikan nil apabila tidak ditemukan target setelah menelusuri semua link hingga depthLimit.

## 4.2 Tata Cara Penggunaan Program

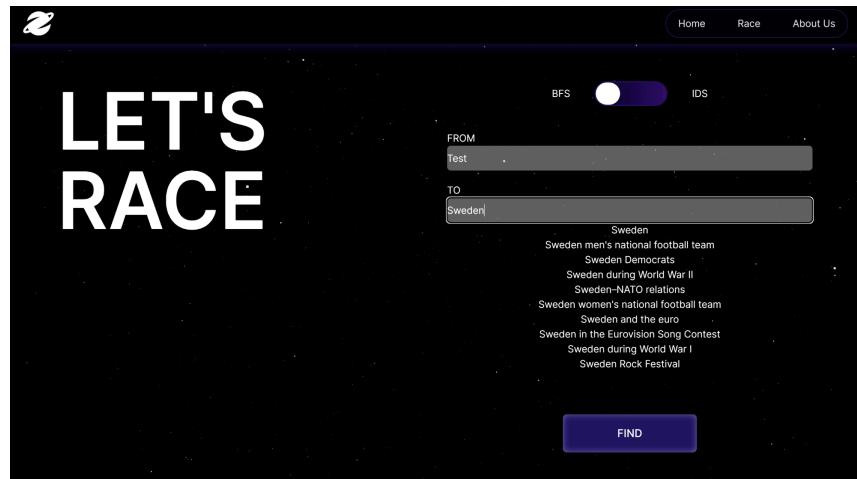
1. Ketika halaman localhost dibuka, pengguna akan berada di halaman *homepage* dengan tampilan seperti berikut.



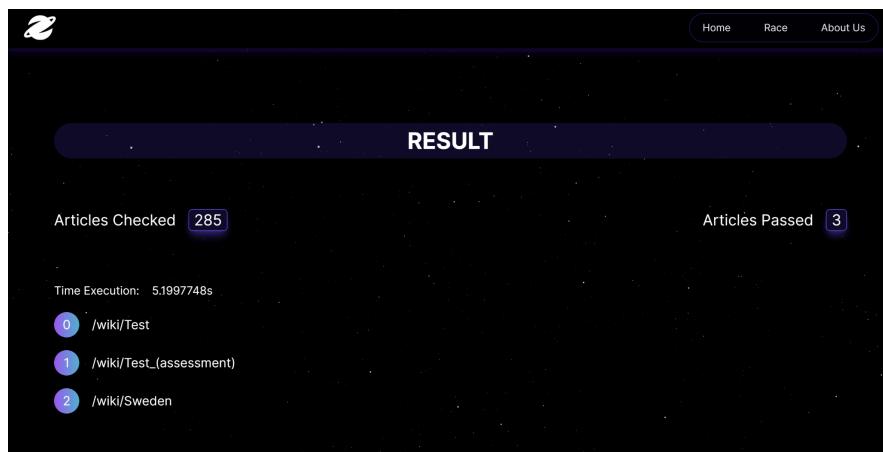
2. Untuk memulai permainan, pengguna dapat menekan tombol “Get Started” pada *homepage* atau menekan tombol “Race” pada *navigation bar* yang terdapat di pojok kanan atas dari halaman
3. Setelah menekan tombol “Get Started” atau “Race”. Pengguna akan diarahkan ke halaman permainan dengan tampilan seperti berikut.



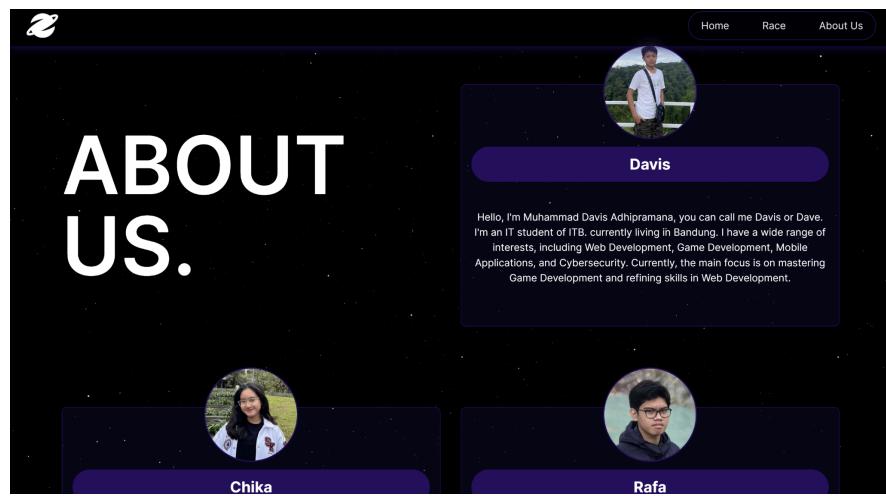
4. Setelah itu, pengguna dapat memilih algoritma yang akan dipakai untuk permainan (algoritma BFS ataupun IDS). Lalu, pengguna dapat meng-*input* judul halaman *Wikipedia* awal dan judul halaman *Wikipedia* tujuan. Jika sudah mengisi semua inputan, maka pengguna dapat menekan tombol “FIND” untuk memulai penelusuran. Berikut adalah contohnya.



- Setelah tombol “FIND” ditekan, maka program akan menampilkan hasil penelusuran yang berisi jumlah artikel yang di cek, jumlah artikel yang dilalui, waktu eksekusi program, dan detail judul artikel yang dilalui seperti berikut.



6. Selain dari halaman permainan, kami juga memiliki halaman “About Us” yang berisi deskripsi singkat dari semua kontributor dalam pembuatan program ini.



### 4.3 Hasil Pengujian

Algoritma BFS	Algoritma IDS
Dari: Guallatiri Ke: Volcano Derajat: 1	
<b>RESULT</b> Articles Checked 1 Articles Passed 2  Time Execution: 732.8044ms 0 /wiki/Guallatiri 1 /wiki/Volcano	<b>RESULT</b> Articles Checked 1 Articles Passed 2  Time Execution: 329.1875ms 0 /wiki/Guallatiri 1 /wiki/Volcano

Dari: Hawaii Ke: Honolulu Derajat: 1

## RESULT

Articles Checked 1

Articles Passed 2

Time Execution: 1.0684709s

0 /wiki/Hawaii

1 /wiki/Honolulu

## RESULT

Articles Checked 1

Articles Passed 2

Time Execution: 303.9152ms

0 /wiki/Hawaii

1 /wiki/Honolulu

Dari: Test Ke: Sweden Derajat: 2

## RESULT

Articles Checked 285

Articles Passed 3

Time Execution: 5.1997748s

0 /wiki/Test

1 /wiki/Test\_(assessment)

2 /wiki/Sweden

## RESULT

Articles Checked 3

Articles Passed 3

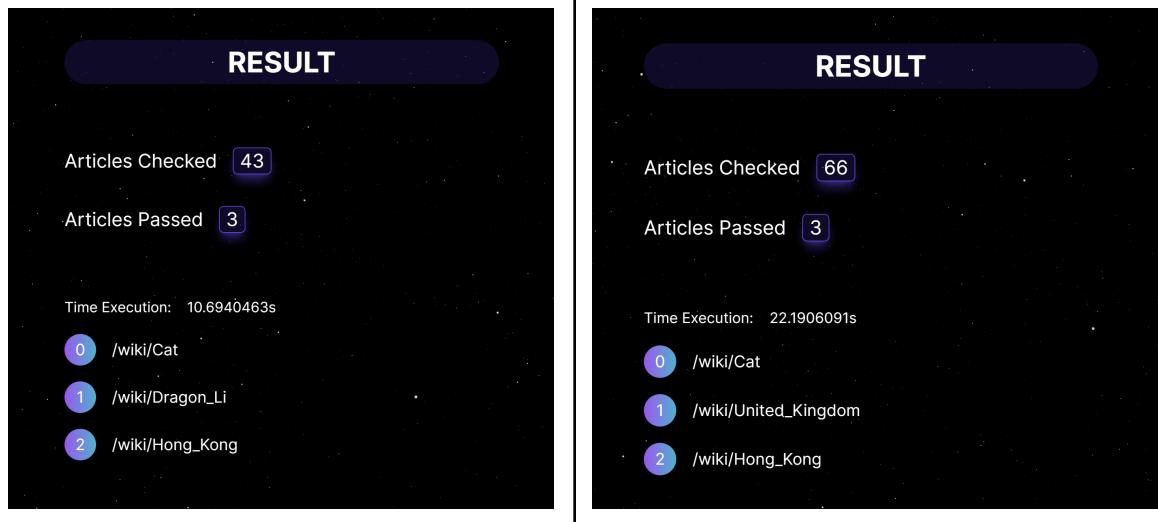
Time Execution: 535.5813ms

0 /wiki/Test

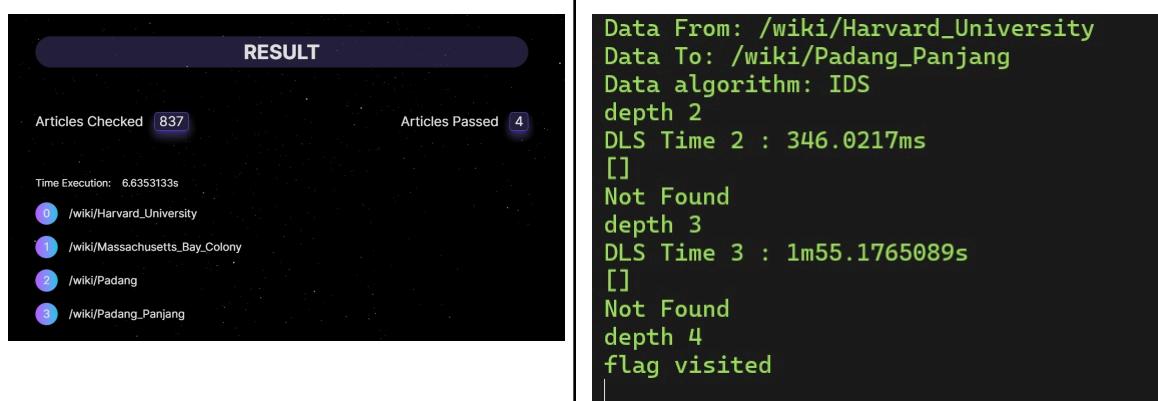
1 /wiki/Test\_(assessment)

2 /wiki/Sweden

Dari: Cat Ke: Hong Kong Derajat: 2

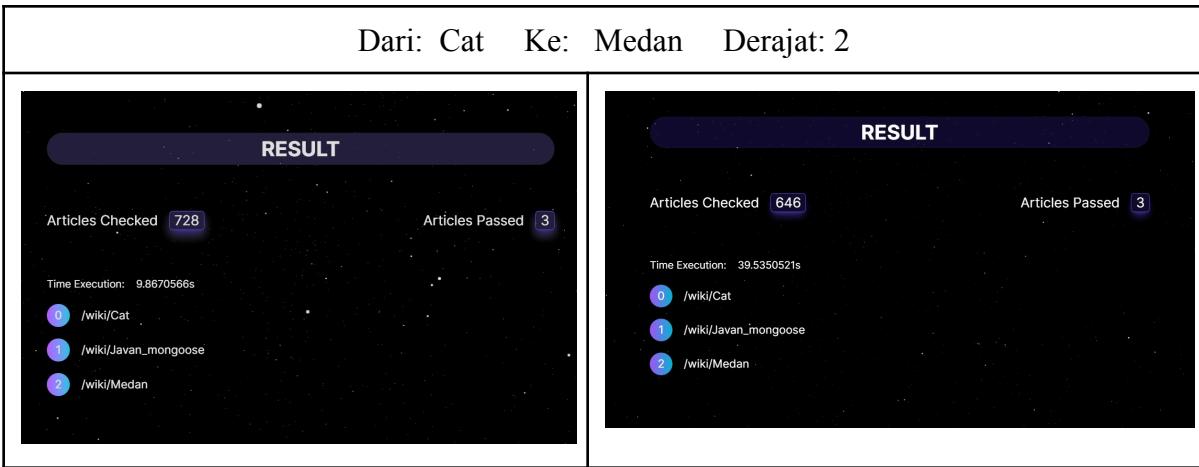


Dari: Harvard University Ke: Padang Panjang Derajat: 3



Catatan:

Untuk kasus  $degree \geq 3$  pada algoritma IDS masih membutuhkan waktu yang cukup lama untuk mengeksekusi program, Pada gambar diatas terlihat bahwa algoritma ini sudah memakan waktu sekitar kurang lebih 2 menit untuk penelusuran di  $depth$  ke-2



## 4.4 Analisis Pengujian

*Algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS)* adalah dua pendekatan yang berbeda dalam penelusuran graf. Meskipun keduanya memiliki tujuan yang sama, yaitu mencari jalur terpendek atau solusi dalam sebuah ruang pencarian, perbedaan dalam kinerja keduanya terletak pada cara mereka mengeksplorasi ruang pencarian tersebut.

Pada *degree 1*, di mana hanya ada satu langkah dari simpul awal ke target atau target itu sendiri terletak pada simpul awal, kedua algoritma seringkali memberikan hasil yang serupa dalam hal waktu eksekusi. Ini karena keduanya hanya perlu melakukan sedikit eksplorasi dalam graf.

Namun, ketika mencapai *degree* yang lebih dalam, seperti *degree 2* hingga *4*, perbedaan antara keduanya menjadi lebih nyata. BFS cenderung lebih cepat karena ia mengeksplorasi semua simpul pada level yang sama sebelum melanjutkan ke level berikutnya. Dalam konteks scraping Wikipedia, di mana setiap artikel bisa memiliki banyak link ke artikel lain, BFS cenderung lebih efisien dalam menelusuri dan mengekstrak data, karena ia menjangkau semua koneksi di setiap level sebelum melangkah ke level berikutnya.

Di sisi lain, IDS, meskipun memiliki keunggulan dalam space complexity karena hanya menyimpan satu jalur dalam memori pada suatu waktu, menjadi lebih lambat karena harus melakukan iterasi berulang kali ke dalam kedalaman pencarian, yang bisa menjadi sangat lambat saat mencapai batas kedalaman yang besar seperti pada Wikipedia.

Dengan demikian, dalam konteks scraping Wikipedia di mana jumlah node yang perlu dieksplorasi sangat besar, BFS lebih disukai karena kinerjanya yang lebih cepat dan kemampuannya untuk menangani jumlah koneksi yang besar dengan lebih efisien.

## **BAB 5 KESIMPULAN DAN SARAN**

### **5.1 Kesimpulan**

Dalam laporan ini, kita telah menjelaskan pemanfaatan algoritma IDS (Iterative Deepening Search) dan BFS (Breadth-First Search) dalam permainan WikiRace. WikiRace adalah permainan dimana pemain diminta untuk menavigasi dari satu artikel Wikipedia ke artikel lainnya hanya dengan mengikuti tautan internal. Penggunaan algoritma IDS dan BFS telah memberikan kontribusi yang signifikan dalam meningkatkan pengalaman bermain WikiRace.

Algoritma IDS memungkinkan pemain untuk melakukan pencarian secara mendalam dengan memperluas pencarian secara bertahap ke dalam cabang-cabang tertentu. Ini berguna ketika pemain ingin mencapai tujuan dengan mempertimbangkan jumlah langkah yang mungkin. Sementara itu, BFS memberikan pendekatan yang sistematis dalam menjelajahi tautan internal pada Wikipedia. Dengan BFS, pemain dapat mengeksplorasi artikel-artikel terkait secara terurut, yang membantu dalam mencapai tujuan dengan efisien.

### **5.2 Saran**

Dalam laporan ini, kami telah menyoroti pemanfaatan algoritma IDS dan BFS dalam permainan WikiRace. Algoritma IDS memungkinkan pencarian yang mendalam, sementara BFS memberikan pendekatan yang sistematis. Implementasi algoritma-algoritma ini memerlukan pengembangan program yang responsif dan efisien. Kami merekomendasikan penelitian lebih lanjut pada algoritma lain seperti DFS dan A\*, serta peningkatan antarmuka pengguna. Integrasi dengan platform permainan online atau mobile juga disarankan untuk memperluas jangkauan permainan. Dengan fokus pada pengembangan algoritma dan pemrograman, WikiRace diharapkan dapat memberikan pengalaman bermain yang lebih menarik dan memuaskan.

## LAMPIRAN

1. Link Repository Github

[https://github.com/Loxenary/Tubes2\\_Wiki\\_Scraper](https://github.com/Loxenary/Tubes2_Wiki_Scraper)

2. Link Video

<https://docs.google.com/document/d/1brFB7IZK8K-ndx5GHxgXC-Rr3zAWkRiX9n9GC89iCEY/edit?usp=sharing>

## **DAFTAR PUSTAKA**

Rinaldi Munir. 2022. "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>

Rinaldi Munir. 2022. "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>

“Breadth-First Search Graph Algorithm (With code in C, C++, Java, and Python)”  
[https://www.programiz.com/dsa/graph-bfs#google\\_vignette](https://www.programiz.com/dsa/graph-bfs#google_vignette)

“Goroutines – Concurrency in Golang”  
[Goroutines - Concurrency in Golang - GeeksforGeeks](#)