

**LAPORAN TUGAS BESAR II**  
**IF2211 STRATEGI ALGORITMA**  
**PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN WIKIRACE**

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma  
pada Semester II Tahun Akademik 2023/2024.



Oleh Kelompok gomugomuno:  
Thea Josephine Halim                    13522012  
Imam Hanif Mulyarahman                13522030  
Nabila Shikoofa Muida                 13522069

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

## DAFTAR ISI

DAFTAR ISI	1
BAB I	
DESKRIPSI MASALAH	4
BAB II	
LANDASAN TEORI	5
2.1. Dasar Teori	5
2.1.1. Penjelajahan Graf	5
2.1.2. Penjelajahan Graf dalam WikiRace	5
2.1.3. Algoritma Iterative Deepening Search	7
2.1.4. Algoritma Breadth First Search	7
2.2. Aplikasi Pembangunan Web	10
2.2.1. Integrated Development Environment (IDE)	11
2.2.2. Framework	11
BAB III	
ANALISIS PEMECAHAN MASALAH	13
3.1. Langkah Pemecahan Masalah	13
3.1.1. Scraping	13
3.1.2. Pencarian Solusi	13
3.1.3. Visualisasi	13
3.2. Pemetaan Masalah pada Algoritma	14
3.2.1. Algoritma IDS	14
3.2.2. Algoritma BFS	14
3.3. Fitur Fungsional dan Arsitektur Website	15
3.3.1. Fitur Fungsional	15
3.3.2. Arsitektur Website	16
3.4 Ilustrasi Kasus	17
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	18
4.1. Implementasi Program	18
4.1.1. Package ids	18
4.1.1.1. Struktur Data Tree	19
4.1.1.2. Function IDS Paralel	20
4.1.1.3. Function DLS Paralel	21
4.1.2. Package bfs (termasuk BFSfunction.go)	21
4.1.2.1. Node Data Struct	22
4.1.2.2. Function testing print untuk debug dan initialBFS	23

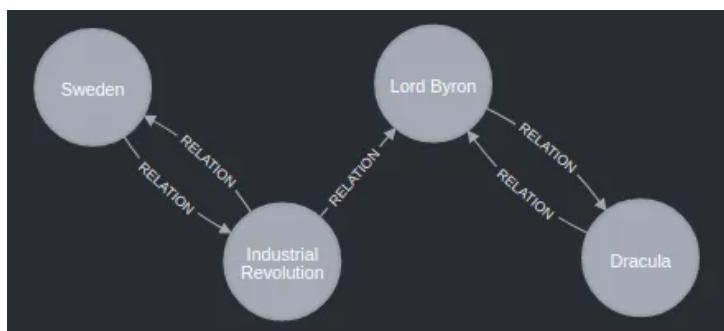
4.1.2.3. Main Algorithm for BFS	23
4.1.2.4. Function multiBFS	24
4.1.2.5. Function ParallelBFS	26
4.1.3. Package Scraper (library gocolly)	27
4.1.3.1. Function Scraper	28
4.1.3.2. Function Convert	28
4.1.4. Package Main	28
4.1.4.1. Function isValidLink	28
4.1.4.2. Function printResult	29
4.1.4.3. Function executeAlgorithm	30
4.1.4.4. Function main	30
4.2. Struktur Data dan Implementasi	31
4.3. Cara Penggunaan Program	32
4.4. Tampilan Antarmuka	33
4.5. Hasil Pengujian	34
4.5.1. Pengujian Breadth First Search	34
4.5.2. Pengujian Iterative Deepening Search	36
4.6. Analisis Hasil Pengujian	39
4.6.1 Faktor yang Mempengaruhi Kecepatan Algoritma	39
4.6.2 Analisis Hasil Pengujian	41
BAB V	43
KESIMPULAN DAN SARAN	43
5.1. Kesimpulan	43
5.2. Saran	43
5.3. Refleksi	43
LAMPIRAN	45
Repository	45
Video	45
DAFTAR PUSTAKA	45



## **BAB I**

### **DESKRIPSI MASALAH**

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



Gambar 1. Ilustrasi Graf WikiRace

(Sumber: [https://miro.medium.com/v2/resize:fit:1400/1\\*jxmEbVn2FFWzbZsIicJCWQ.png](https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWzbZsIicJCWQ.png))

Pada tugas besar kali ini, akan dibuat sebuah website sederhana yang dapat melakukan WikiRace sendiri dengan memanfaatkan algoritma BFS (Breadth First Search) dan IDS (Iterative Deepening Search). Website akan menerima masukan input kata utama sebagai simpul awal dimulainya WikiRace dan input kata akhir sebagai tujuan yang ingin dicari. Berdasarkan dari jenis algoritma yang dipilih oleh pengguna, website akan melakukan WikiRace dan menampilkan hasil total jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute perjalanan artikel, dan lama eksekusi hingga ditemukan solusi tersebut. Perlu dilakukan juga analisis mendalam mengenai perbedaan algoritma BFS dan IDS dalam pencarian solusi terpendek WikiRace. Waktu algoritma pemrosesan termasuk scraping harus dipastikan untuk tidak melebihi 5 menit. Perbandingan akan dilakukan berdasarkan waktu kompleksitas algoritma dan ketepatan solusi yang dihasilkan.

## BAB II

### LANDASAN TEORI

#### 2.1. Dasar Teori

##### 2.1.1. Penjelajahan Graf

Graf adalah kumpulan *node* atau simpul yang saling terhubung oleh *edges* yang melambangkan relasi antara simpul. Graf terdiri dari kumpulan simpul/*vertices* (V) dan kumpulan *edges* (E), sehingga graf bisa dituliskan dengan  $G(V, E)$ . Graf umumnya digunakan sebagai visualisasi hubungan antar objek. Lebih lanjut, graf dapat digunakan untuk pencarian jalan seperti persoalan labirin dan pencarian elemen. Penjelajahan graf, atau graf traversal dapat dilakukan dalam beberapa cara, yaitu pencarian secara melebar dan pencarian secara mendalam. Salah satu contoh pencarian secara melebar adalah Breadth First Search (BFS) dan contoh pencarian secara mendalam misalnya *Depth First Search* (DFS) dan *Iterative Deepening Search* (IDS).

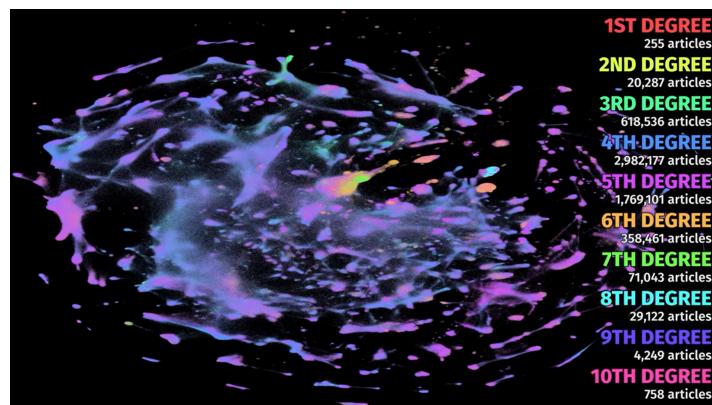
Algoritma pencarian solusi berdasarkan graf dapat diklasifikasikan berdasarkan tingkat informasi yang digunakan. Pada kategori tanpa informasi (*uninformed/blind search*), seperti *Depth First Search* (DFS), *Breadth First Search* (BFS), *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search* tidak ada informasi tambahan yang diperoleh. Sebaliknya, pada pencarian berbasis heuristik atau *informed search*, seperti *Best First Search* dan A\*, informasi heuristik dimanfaatkan untuk mengevaluasi non-goal state yang dianggap lebih menjanjikan.

##### 2.1.2. Penjelajahan Graf dalam WikiRace

Dalam WikiRace, sebuah node melambangkan sebuah judul laman Wikipedia, yang didapatkan dengan *scraping links* pada website. Dalam Wikirace, sebuah node bisa menjadi anak ataupun parent, atau keduanya. Sebuah title node disebut memiliki anak jika di dalam laman tersebut terdapat link lain (tidak buntu). Sebuah node memiliki parent jika ia dapat diakses dari laman parent-nya. Akan tetapi, terdapat juga kasus *unreachable node*, yaitu node yang tidak memiliki parent ataupun anak, sehingga tidak dapat diakses dari link lain ataupun mengakses link lain dari dirinya sendiri. Selain itu, terdapat juga kumpulan *orphan nodes* yaitu node yang tidak punya parent sehingga tidak dapat diakses dari link lain. Sebaliknya,

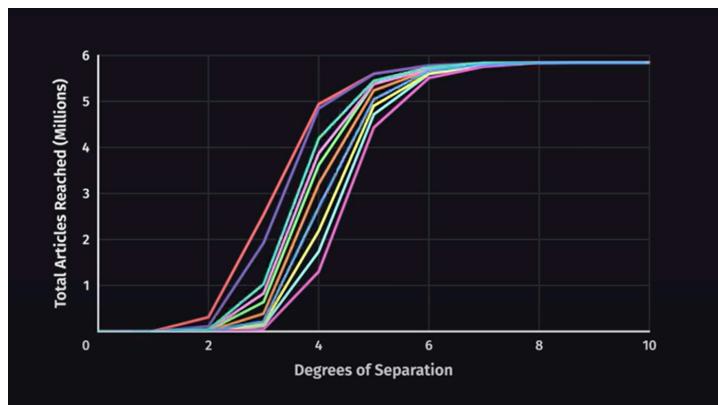
*dead end nodes* adalah nodes yang tidak punya anak, sehingga kita tidak dapat lanjut dari node tersebut, kasus ini sangat jarang terjadi.

Video referensi yang dilakukan *channel adumb* di [Youtube](#), memberikan banyak informasi detail terkait WikiRace dan sangat direkomendasikan untuk ditonton untuk memperjelas pemahaman dasar WikiRace. Pada video tersebut dijelaskan bahwa keterkaitan antar *link* pada Wikipedia mengikuti prinsip *Six Degrees of Separation*, konsep yang menyatakan bahwa setiap individu di dunia ini dapat terhubung dengan individu lain tidak lebih dari enam langkah. Pada video digunakan contoh Wikipedia “Pluto” dan ditemukan bahwa terjadi penurunan jumlah link yang dicapai pada degree kelima. Awalnya akan terjadi lonjakan yang sangat besar di degree awal, lalu mulai menurun karena kebanyakan link sudah pernah dicapai.



Gambar 2.1.1 Penyebaran Link pada Pluto

Gambar di bawah menggambarkan hasil percobaan pertumbuhan jumlah link dari setiap degree, baik untuk link Wikipedia dengan jumlah anak yang sedikit maupun banyak. Tampak bahwa pola pertumbuhan link yang drastis terjadi pada degree dua atau tiga.



Gambar 2.1.2 Pertumbuhan Penyebaran Link

### 2.1.3. Algoritma *Iterative Deepening Search*

IDS (Iterative Deepening Search) hampir sama dengan DFS (Depth First Search) tetapi dengan implementasi derajat kedalaman. IDS melakukan serangkaian DFS dengan peningkatan nilai kedalaman-cutoff hingga solusi ditemukan. Dengan asumsi simpul sebagian besar berada di level bawah, sehingga tidak menjadi masalah ketika simpul pada level atas dibangkitkan secara berulang kali.

```
Depth ← 0
Iterate
    result ← DLS (problem, depth)
    stop: result ≠ cutoff
    depth ← depth+1
→ result
```

Iterative Deepening Search (IDS) memiliki beberapa properti yang penting. Pertama, IDS adalah metode pencarian yang lengkap jika kedalaman batas ( $b$ ) dari pencarian terbatas. Kedua, IDS juga optimal jika langkah yang diambil ke sebuah solusi memiliki biaya yang sama. Dalam hal kompleksitas waktu, IDS memiliki kompleksitas waktu  $O(b^d)$ , di mana  $b$  adalah faktor percabangan maksimum dan  $d$  adalah kedalaman solusi. Sedangkan kompleksitas ruang IDS adalah  $O(bd)$ , di mana  $b$  adalah faktor percabangan maksimum dan  $d$  adalah kedalaman solusi.

### 2.1.4. Algoritma *Breadth First Search*

Algoritma BFS akan dimulai pada simpul yang sudah kita tentukan, lalu mulai melakukan penjelajahan *traversal* ke cabang-cabangnya. Eksplorasi harus dilakukan dengan satu per satu, setiap simpul tetangga dilakukan pengecekan dahulu baru lanjut ke anak simpul tetangga. Dalam implementasinya, BFS menggunakan sistem queue (antrian). Simpul tetangga yang ditemukan dari simpul yang dicek akan di-*enqueue* ke dalam queue simpul hidup, yaitu kumpulan simpul yang perlu dilakukan pengecekan untuk selanjutnya. Akan tetapi, agak sedikit berbeda dengan BFS pada tree, untuk menghindari *cycle* yang berpotensi melakukan pengecekan simpul yang sama, kita memerlukan sebuah variabel boolean

“visited” yang akan bernilai true jika simpul sudah pernah dikunjungi. Secara garis besar, BFS akan dilakukan dengan urutan sebagai berikut:

- a. Inisialisasi dengan enqueue simpul awal/asal ke queue simpul hidup dan ganti boolean “visited” menjadi true.
- b. Lakukan iterasi proses BFS selama queue tidak kosong, dequeue simpul pertama dari *queue*
- c. Jika simpul yang sedang dicek sama dengan simpul yang kita cari, hentikan pencarian, jika tidak, cari simpul tetangga dari simpul pertama tersebut dan enqueue ke dalam queue simpul hidup sembari mengganti variable “visited” menjadi *true*.
- d. Lakukan proses terus-menerus hingga queue kosong yang menandakan tidak ada solusi atau ditemukan solusi.

Adapun skema umum algoritma BFS antara lain sebagai berikut

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.
  Masukan : v adalah simpul awal kunjungan
  Keluaran : semua simpul yang dikunjungi dicetak ke layar}

Deklarasi
w : integer
q : antrian

procedure BuatAntrian(input/output q : antrian)
{ membuat antrian kosong, kepala(q) diisi nilai 0 }

procedure MasukAntrian(input/output q:antrian, input v: integer)
{ memasukkan v ke dalam antrian q pada posisi belakang }

procedure HapusAntrian(input/output q:antrian, input v:integer)
{ menghapus v dari kepala antrian q }

function AntrianKosong(input q:antrian) -> boolean
{ true jika antrian q kosong, false jika sebaliknya }
```

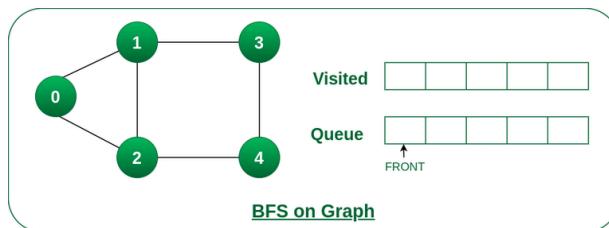
**Algoritma**

```
BuatAntrian(q)
write(v)
dikunjungi(v) <- true
MasukAntrian(q, v)

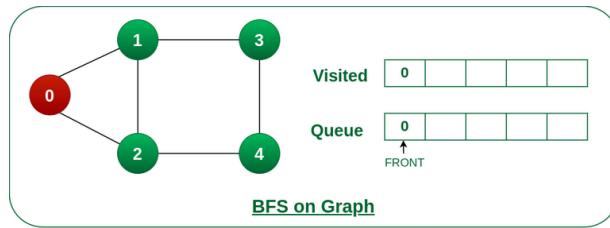
{ Kunjungi semua simpul graf selama antrian belum kosong }
while not AntrianKosong(q) do
    HapusAntrian(q, v)
    for tiap simpul q yang bertetangga dengan v do
        if not dikunjungi(w) then
            write(w)
            MasukAntrian(q, w)
            dikunjungi(x) <- true
        endif
    endfor
endwhile

{AntrianKosong(q) }
```

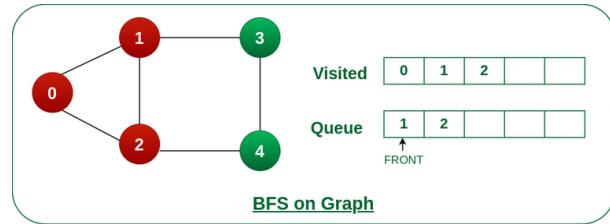
Untuk memperjelas, akan digunakan gambar proses BFS yang diambil dari <https://www.geeksforgeeks.org/>. Disediakan graf seperti gambar di bawah, algoritma pengecekan akan dimulai dari simpul 0 dan akan ditelusuri hingga ditemukan simpul 3. Perlu diingat bahwa pencarian dihentikan jika simpul yang dicek (*di-dequeue*) adalah simpul tujuan, bukan karena terdapat simpul tujuan di simpul tetangga yang akan *di-enqueue*.



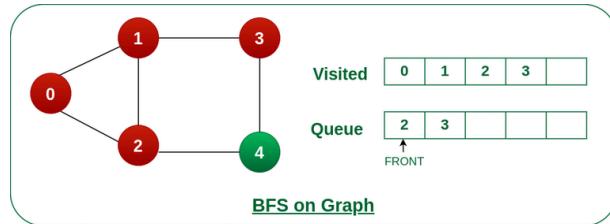
Gambar 2.3.1 Graph Awal dengan Queue kosong



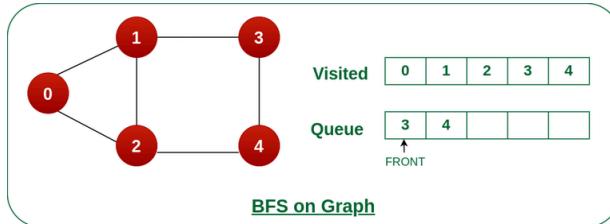
Gambar 2.3.2 Iterasi Pertama



Gambar 2.3.3 Iterasi Kedua



Gambar 2.3.4 Iterasi Ketiga



Gambar 2.3.5 Simpul 3 Ditemukan

## 2.2. Aplikasi Pembangunan Web

Pada Tugas Besar II Strategi Algoritma ini, program diimplementasikan dalam bahasa Go. Bahasa Go atau yang lebih familiar disebut Golang adalah bahasa pemrograman open-source yang dikembangkan oleh Google. Go menjadi salah satu bahasa pemrograman paling populer karena program diketik secara statis, bukan secara

dinamis seperti JavaScript dan Python. Program yang diketik statis tidak akan berjalan sampai kesalahan sudah diperbaiki. Berbeda dengan bahasa yang diketik secara dinamis tetap bisa dijalankan bahkan jika ada error.

Dirangkum dari Flexiple, fungsi Golang yang bisa dijalankan oleh developer antara lain dapat membuat berbagai aplikasi, menulis kode yang bersih, dapat dipelihara, dan efisien, serta membuat serta memelihara dokumentasi aplikasi perangkat lunak. Selain itu, Golang juga dapat membuat dan mengelola rangkaian pengujian otomatis, melakukan debugging, dan memecahkan masalah perangkat lunak. Pengembang Golang juga mampu mengembangkan aplikasi perangkat lunak yang dapat diskalakan dan berkinerja tinggi, serta memelihara aplikasi di area produksi.

### **2.2.1. Integrated Development Environment (IDE)**

Dalam pembuatan tugas besar ini, IDE yang digunakan adalah Visual Studio Code. VSCode merupakan IDE yang ideal untuk pengembangan perangkat lunak dengan berbagai bahasa pemrograman, termasuk namun tidak terbatas pada JavaScript, TypeScript, Python, Go, dan banyak lagi. Kami memilih VSCode karena ia menawarkan ketersediaan ekstensi yang luas, fitur linting dan debugging yang kuat, integrasi dengan Git, serta dukungan platform yang luas, menjadikannya pilihan yang populer bagi pengembang yang mencari lingkungan pengembangan yang fleksibel dan berkinerja tinggi.

### **2.2.2. Framework**

Website pada tugas besar kali ini menggunakan framework React untuk membantu pemecahan komponen-komponen website agar mudah *di-reuse* dan modular. React dipilih karena memiliki ekosistem yang kuat dengan banyaknya pustaka, alat-alat pengembangan, dan dokumentasi yang memudahkan pengembangan aplikasi web yang kompleks.

Salah satu fitur unggulan ReactJS yaitu JSX. JSX adalah ekstensi sintaks Javascript yang memungkinkan untuk penggunaan HTML di Javascript. Dengan JSX, kamu dapat memodifikasi Document Object Model (DOM), yaitu sebuah API untuk

mengatur struktur halaman web. Modifikasi DOM dilakukan untuk menambahkan konten dinamis pada halaman website.

Virtual DOM berguna untuk melihat bagian dari DOM asli yang diubah. Ketika developer mengupdate DOM pada JSX, ReactJS akan membuat salinan DOM aslinya. Fitur ini bertugas untuk membuat kode seperti dengan me-render seluruh halaman yang dilakukan oleh pengguna pada setiap perubahannya. Ketika ada bagian yang perlu diubah, React JS akan mengubah bagian tersebut saja sehingga pengguna tidak perlu melakukan reload satu halaman untuk melihat perubahan.

Single way data flow adalah suatu teknik yang hanya memiliki satu cara untuk mentransfer data ke seluruh aplikasi. Dengan teknik ini, kontrol terhadap data jadi lebih baik. Di ReactJS, aliran data dari satu bagian ke bagian lainnya hanya terjadi secara satu arah. Hal ini menjadikan proses debug jadi lebih mudah, meminimalisir terjadinya error, dan lebih efisien karena library mengetahui batasan setiap bagian dari sistem.

Lalu untuk penggeraan bonus tampilan graf, kami menggunakan pustaka React D3 Graph. D3.js adalah pustaka JavaScript untuk membuat visualisasi data yang dinamis dan interaktif menggunakan HTML, CSS, dan SVG. D3 mengikat data ke DOM dan elemennya, memungkinkan kita untuk memanipulasi visualisasi dengan mengubah data.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

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

Pada tugas besar kali ini, kami mengawali penggerjaan dengan analisis permasalahan untuk memudahkan penggerjaan. Analisis permasalahan tersebut menghasilkan beberapa bagian yang dijabarkan sebagai berikut:

##### **3.1.1. Scraping**

Pada bagian scraping ini, kami mengasumsikan sebuah link yang dapat dicapai sebagai sebuah string dengan cara cut/trimming link yang didapatkan setelah “/wiki/”. Perlu diingat bahwa judul link yang dibuat hanya diambil dari link yang bisa diklik pada laman dan merupakan link *direct* menuju laman Wikipedia lain. Kami juga mengabaikan duplikasi link yang mungkin muncul selama proses scraping.

Hasil scraping ini akan digunakan untuk membangun struktur objek tree pada IDS dan node pada BFS. Sebuah judul pada laman akan dijadikan sebuah node dan bergantung pada algoritmanya, urutan kunjungan node bisa berbeda-beda. Hal ini akan dijelaskan lebih lanjut pada poin pemetaan masalah pada algoritma.

##### **3.1.2. Pencarian Solusi**

Pada pencarian solusi, akan dibuat fungsi pencarian yang menerapkan algoritma BFS dan IDS. Penjelasan mengenai kedua algoritma akan dijelaskan pada poin di bawah. Pada algoritma, digunakan struktur data *queue* untuk BFS dan *tree* untuk IDS.

##### **3.1.3. Visualisasi**

Pada bagian pembuatan website, kami akan membuat website lokal yang dapat menerima *input* berupa dua buah judul Wikipedia yang valid dengan pilihan metode pencarian antara BFS dan IDS. Kemudian 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 *millisecond*.

## 3.2. Pemetaan Masalah pada Algoritma

### 3.2.1. Algoritma IDS

Pada algoritma IDS, dibuat sebuah struktur data bernama *tree*. Sebuah *tree* memiliki atribut *value* yang bertipe string dan atribut *subtree* yang bertipe array dinamis berisi *pointer* ke *tree*. Atribut *value* berisi judul dari suatu halaman wikipedia dan atribut *subtree* berisi *pointer* ke halaman wikipedia lain.

Konsep utama yang digunakan dalam algoritma IDS adalah konsep penelusuran secara rekursif menggunakan tipe data pohon. Setiap *tree* memiliki kumpulan *subtree* yang akan ditelusuri secara DFS. DFS akan dilakukan sampai tingkat kedalaman tertentu. Apabila tidak menemukan solusi pada kedalaman tersebut, maka tingkat kedalaman akan bertambah dan penelusuran akan dilakukan dari akar lagi.

### 3.2.2. Algoritma BFS

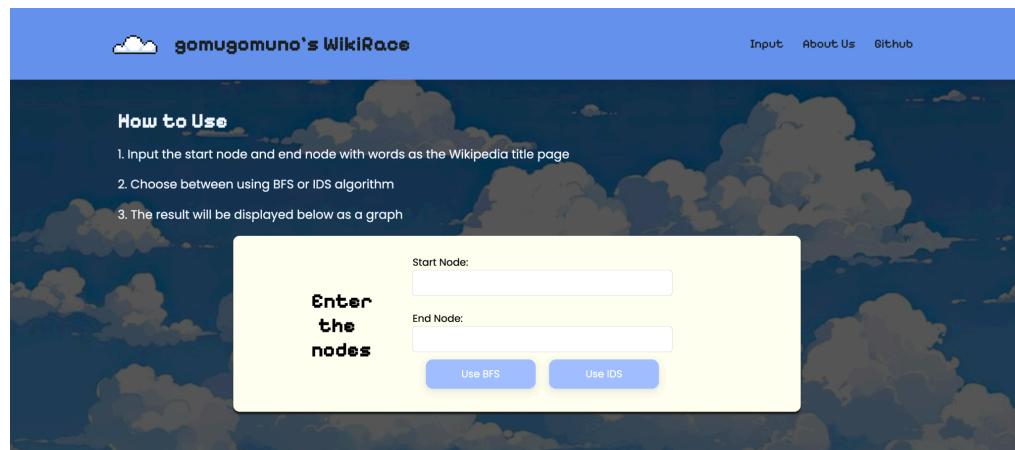
Pada algoritma BFS, dibuatlah sebuah struktur data bernama *node*. Sebuah *node* memiliki atribut *title* dan *parents*. Atribut *title* berisi string nama link yang dijadikan *node* tersebut, sedangkan *parents* sebuah *node* berisi *dynamic array of strings* yang diisi oleh *title* *node* yang telah dilewati sebelumnya hingga tercapailah *title* sekarang. Sebagai contoh, dari laman *title* “Vector”, kita bisa *traverse* ke laman “Euclidean\_vector”, dilanjut dengan laman “Mathematics”. Maka dari itu dapat kita tuliskan sebagai berikut: sebuah *node* dengan *title* Mathematics, memiliki *parent* “Vector” dan “Euclidean\_vector”. Konsep *node* ini akan kita gunakan terus dalam BFS untuk *track* *node* *parent* yang telah kita lalui serta pencegahan memasukkan kembali *node* yang sudah dikunjungi ke dalam list simpul hidup.

Seperti yang kita ketahui, algoritma BFS menggunakan konsep *queue* dalam pengunjungan *node-node* yang ditemukan. Akan dibuat sebuah *slice* simpul hidup yang berisi string calon simpul yang akan dilalui, dan akan dilakukan *dequeue* *node* satu per satu sembari membangkitkan simpul-simpul hidup dari *node* tersebut (dengan *scraping*). Simpul-simpul hidup yang dibangkitkan akan dimasukkan ke dalam *slice* simpul hidup dengan *enqueueing/append* dari belakang.

### 3.3. Fitur Fungsional dan Arsitektur Website

Pada tugas besar ini akan digunakan website sebagai media visualisasi WikiRace dengan kedua algoritma BFS dan IDS. Pengguna akan memasukkan kata awal sebagai *starting node* dan kata tujuan sebagai *end node* yang dicari. Pengguna dapat memilih di antara kedua algoritma dalam pemrosesannya, yaitu dengan memiliki antara tombol IDS atau BDS. Website akan melakukan proses di balik layar sebelum menampilkan hasilnya ke layar dalam bentuk graf.

#### 3.3.1. Fitur Fungsional



Gambar 3.3.1 Antarmuka I

Fitur-fitur fungsional yang terdapat dalam website kami yaitu sebagai berikut

- Textbox*, digunakan untuk memasukkan input judul artikel awal (*start node*) dan judul artikel tujuan (*end node*)
- Button*, pengguna dapat memilih jenis algoritma untuk dijalankan antara BFS dan IDS
- Hyperlink*, website kami juga memungkinkan pengguna untuk menavigasi ke halaman web lain melalui hyperlink yang disediakan



Gambar 3.3.1 Antarmuka II

- d. *Clickable Card*, menampilkan profile pengguna beserta hyperlink menuju github masing-masing



### 3.3.2. Arsitektur Website

Main page pada website ada pada file index.js dan semua komponen akan diimpor di situ. Komponen yang ada pada website adalah About (berisi komponen Card), Content, Footer, GraphComponent, Header, dan How to Use. Komponen About digunakan untuk section About Us, komponen Content berisi input form dan proses untuk *handling* proses algoritma, komponen Footer untuk sign *copyright*, komponen GraphComponent yang memanfaatkan *library react d3 graph* untuk mengubah array result yang sudah di JSONify menjadi sebuah graf. Komponen Header digunakan untuk menampilkan navigation bar, sedangkan komponen How to Use menunjukkan cara penggunaan website.

Pada main.go, algoritma akan dijalankan dan akan didapatkan hasil berupa *result([ ] string)*, *elapsed(int64)*, *shortestlength(int)*, dan *numofcheckednodes(int)*. Keempat hasil ini akan di-decode menjadi JSON *struct response*. Pada Content.js akan terdapat function handleRace yang bertugas untuk *fetch data* masukan user dan mengirimkannya ke algoritma backend. Dengan menggunakan state untuk hasil-hasil yang didapatkan, kita bisa menampilkannya pada website dengan mudah. Di Content.js juga akan dilakukan pengecekan handling error. Pada pengeraaan ini kita cek supaya nilai kedua node tidak kosong ataupun sama (0 degree).

### **3.4        Ilustrasi Kasus**

Selain algoritma IDS dan BFS, mungkin kita juga bisa menggunakan algoritma lain seperti Depth First Search (DFS). Algoritma DFS cenderung mengikuti jalur tertentu sejauh mungkin sebelum beralih ke jalur lain, yang dapat menyebabkan kondisi ngeloop (perulangan tak berujung) jika tidak diimplementasikan dengan benar. Oleh karena itu, pengguna perlu memperhatikan dengan cermat penggunaan DFS dan memastikan bahwa input yang diberikan sesuai dengan kondisi yang diinginkan untuk menghindari masalah perulangan tak berujung.

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1. Implementasi Program**

Program WikiRace dibagi menjadi beberapa submodule, di antaranya adalah algoritma IDS, BFS, scraper gocolly, dan main.

##### **4.1.1. Package ids**

Secara keseluruhan, berikut adalah langkah-langkah pemecahan masalah dengan algoritma IDS:

- Buat sebuah tree (akar) dengan value berisi masukan judul halaman wikipedia dan subtree kosong.
- Scrape semua link yang bisa didapatkan dari masukan judul tersebut dan jadikan subtree dari judul tersebut.
- Untuk setiap subtree yang dimiliki judul tersebut, cek apabila value sama dengan hasil yang kita cari, apabila tidak sama maka lakukan langkah kedua pada secara rekursif sampai batas kedalaman yang ditentukan.
- Apabila solusi belum ditemukan maka tingkat kedalaman akan ditambah satu tingkat.
- Lakukan proses diatas untuk semua bagian tree atau sampai solusi ditemukan.

Untuk package ids, terdapat fungsi DLS yang akan melakukan DFS dengan kedalaman tertentu. Fungsi IDS akan memanggil fungsi DLS untuk setiap tingkat kedalaman yang akan bertambah sampai menemukan solusi.

#### 4.1.1.1. Struktur Data Tree

```
● ● ●
1 package ids
2
3 import (
4     "fmt"
5     "strings"
6     "Tubes2_gomugomuno/Scrape"
7     "sync"
8     "time"
9 )
10
11 type Tree struct {
12     Value string
13     SubTree []*Tree
14 }
15
16 func createTree(startTitle string) *Tree{
17     return &Tree{
18         Value: startTitle,
19         SubTree: nil,
20     }
21 }
22
23 // Membuat anak-anak pohon
24 func (t *Tree) AddSubtree(value string) {
25     child := &Tree{Value: value}
26     t.SubTree = append(t.SubTree, child)
27 }
28
29 func (t *Tree) displayTree(){
30     fmt.Println(t.Value)
31     for _, sub := range t.SubTree {
32         fmt.Print("    ")
33         sub.displayTree()
34     }
35 }
36
37 func (t *Tree) displayTreeWithLevel(level int){
38     spasi := level
39     for spasi > 0{
40         fmt.Print("    ")
41         spasi--;
42     }
43     fmt.Println(t.Value)
44     for _, sub := range t.SubTree {
45         sub.displayTreeWithLevel(level+1)
46     }
47 }
48 func (t *Tree) getSumAll() int {
49     sum := 1
50     for _, sub := range t.SubTree {
51         sumSub := sub.getSumAll()
52         sum = sum + sumSub
53     }
54     return sum
55 }
56
57
```

#### 4.1.1.2. Function IDS Paralel

```
● ● ●
1 func IDSParalel(startTitle string, endTitle string) ([][]string, int64, int, int, int) {
2     start := time.Now()
3     var result [][]string
4     var wg sync.WaitGroup
5     maxNumThread := 300
6     pengali := 0
7     startTitle = Scrape.Convert(startTitle)
8     root := createTree(startTitle)
9     childRoot := Scrape.Scraper(startTitle)
10    numThreads := len(childRoot)
11    endTitle = Scrape.Convert(endTitle)
12    path := []string{startTitle}
13    sisa := numThreads
14    resultavailable := &result
15    fmt.Println(numThreads)
16    for _, child := range childRoot {
17        root.AddSubtree(child)
18        if strings.EqualFold(child, endTitle) {
19            *resultavailable = append(*resultavailable, []string{startTitle, child})
20        }
21    }
22    if numThreads > maxNumThread {
23        pengali = numThreads / maxNumThread
24        sisa = numThreads % maxNumThread
25    }
26    iterasi := 1
27    for len(*resultavailable) == 0 {
28        fmt.Println("Iterasi : ", iterasi)
29        for x := 0; x < pengali; x++ {
30            for i := 0; i < maxNumThread; i++ {
31                wg.Add(1)
32                go func(akar *Tree) {
33                    defer wg.Done()
34                    DLSParalel(akar, endTitle, path, iterasi, resultavailable)
35                }(root.SubTree[x*maxNumThread + i])
36            }
37            wg.Wait()
38        }
39        for r := 0; r < sisa; r++ {
40            wg.Add(1)
41            go func(akar *Tree) {
42                defer wg.Done()
43                DLSParalel(akar, endTitle, path, iterasi, resultavailable)
44            }(root.SubTree[pengali*maxNumThread + r])
45        }
46        wg.Wait()
47        iterasi++
48    }
49    elapsed := time.Since(start).Milliseconds()
50    return *resultavailable, elapsed, iterasi, root.getSumAll(), len(result)
51 }
```

#### 4.1.1.3. Function DLS Paralel

```
1 func DLSParalel(root *Tree, endUrl string, path []string, iterasi int, resultavailable *[][]string) {
2     if iterasi == 1 {
3         hasil_scrape := Scrape.Scraping(root.Value)
4         if hasil_scrape == nil {
5             fmt.Println("gaada anak")
6             return
7         } else {
8             for _, sub := range hasil_scrape {
9                 root.AddSubtree(sub)
10                // fmt.Println("visited " + sub)
11                if strings.EqualFold(sub, endUrl) {
12                    fmt.Println("Ketemuuu pathnya", iterasi)
13                    path := append(path, root.Value)
14                    fmt.Println(path)
15                    // *resultavailable = append(*resultavailable, path)
16                    *resultavailable = [][]string{path}
17                    return
18                }
19            }
20        }
21    } else {
22        path = append(path, root.Value)
23        for _, child := range root.SubTree {
24            // fmt.Println(child.Value)
25            DLSParalel(child, endUrl, path, iterasi-1, resultavailable)
26            if len(*resultavailable) > 0 {
27                break
28            }
29        }
30    }
31 }
32 }
33 }
34 }
```

#### 4.1.2. Package bfs (termasuk BFSfunction.go)

Secara keseluruhan, berikut adalah langkah-langkah pemecahan masalah dengan algoritma BFS:

- Masukkan input judul pada queue
- Scrape semua link yang bisa didapatkan dari start judul inputan dan simpan ke dalam slice of strings.
- Dequeue elemen pertama dari queue dan jadikan sebagai currentNode
- Cek apabila currentNode sama dengan hasil yang kita cari, apabila tidak sama scrape currentNode dan masukkan link yang didapat pada queue dengan enqueueing.
- Lakukan proses dequeue dan enqueue bergantian hingga queue kosong atau solusi ditemukan.

- Kembalikan hasil yang didapat, jika solusi kosong maka tidak ada jalur yang mungkin ditempuh dari laman start ke finish.

Pada awalnya kami ingin melakukan implementasi iterasi hingga queue kosong sehingga bisa mendapatkan seluruh solusi yang mungkin. Strategi yang kami pikirkan adalah algoritma BFS seperti biasa, dan menyimpan solusi yang ditemukan pada sebuah slice of slice of strings. Variable *shortestlengthavailable* akan menjadi batas untuk penghentian pengecekan. Pengecekan akan terus berlangsung hingga queue kosong (*most likely not*) atau node yang dicek jumlah parentnya melebihi *shortestlengthavailable*. Akan tetapi, setelah melakukan testing dan percobaan lebih lanjut, untuk memenuhi 5 menit cukuplah sulit terutama untuk solusi *4 degrees of separation*.

#### 4.1.2.1. Node Data Struct

```

1 type Node struct {
2     Title    string // The title of the Wikipedia link
3     Parents []string // Titles of parent nodes traversed to reach this node
4 }
5
6 // Node constructor, user defined
7 func createNode(startTitle string, NodeParent []string) *Node{
8     return &Node{
9         Title: startTitle,
10        Parents: NodeParent,
11    }
12 }
13

```

#### 4.1.2.2. Function *testing print* untuk debug dan *initialBFS*

```
● ● ●
1 func printAllQueue(queue []*Node){
2     fmt.Println("Simpul hidup:")
3     for i := 0; i < len(queue); i++ {
4         fmt.Println("Title:", queue[i].Title)
5         fmt.Println("Parents:")
6         fmt.Print("- ")
7         for j := 0; j < len(queue[i].Parents); j++ {
8             fmt.Println(queue[i].Parents[j])
9         }
10    fmt.Println()
11 }
12 }
13
14 func (n *Node) Print() {
15     fmt.Println("Title:", n.Title)
16     fmt.Println("Parents:")
17     if len(n.Parents) > 0 {
18         fmt.Println(strings.Join(n.Parents, ", "))
19     } else {
20         fmt.Println("<none>")
21     }
22 }
```

#### 4.1.2.3. Main Algorithm for BFS

```
● ● ●
1 package bts
2
3 import (
4     "Tubes2_gomugomuno/Scrape"
5     "fmt"
6     "sync"
7     "time"
8     "context"
9 )
10
11 func printStrings(slice []string) {
12     for _, s := range slice {
13         fmt.Println(s)
14     }
15 }
16
17
18 // return the list of Nodes from startTitle created inside a queue
19 func initialBFS(startTitle string) ([]*Node){
20     firstNode := createNode(startTitle,nil) // node pertama
21     var queue []*Node          // simpul hidup
22     tempparent := Scrape.Scaper(startTitle)
23     for i := 0; i<len(tempparent); i++{
24         var tempparent []string = append(firstNode.Parents, startTitle)
25         queue = append(queue, createNode(tempparent[i],tempparent))
26     }
27     return queue
28 }
```

#### 4.1.2.4. Function *multiBFS*

Code	Deskripsi
<pre> 1 2 // find a solution inside a queue (the queue has been divided by the number of threads) 3 func multiBFS(queue []<b>*Node</b>, startNode string, endNode string, 4     shortestLengthAvailable *int, resultAvailable [][]string, 5     numOfCheckedNodesAvailable *int, elapsedAvailable int64, ctx context.Context) { 6     var result [][]string           // list of answers 7     var hasFound bool = false 8     var foundOneSol bool = false 9     var numOfCheckedNodes := 0 10    shortestLength := 0 11    visited := make(map[string]string) 12    secondToLast := make(map[string]bool) 13    start := time.Now() </pre>	<p>Inisialisasi variable awal dan timer start.</p> <p>Sebagai keterangan tambahan:</p> <ul style="list-style-type: none"> <li>• result akan menyimpan hasil solusi yang didapatkan</li> <li>• Boolean hasFound akan digunakan untuk penanda bahwa sebuah solusi telah ditemukan. Awalnya hasFound digunakan untuk menangani kasus <i>multiple solution</i>.</li> <li>• Integer numOfCheckedNodes diisi dengan jumlah nodes atau laman yang dicek (di seluruh thread routine) hingga sebuah solusi ditemukan.</li> <li>• Integer shortestLength adalah banyak degree</li> <li>• Map Visited berisi key judul laman dan value element terakhir parent setiap kali dilakukan kunjungan pada currentNode. Tujuannya adalah menghindari kasus siklis dan append node yang sudah pernah dikunjungi pada queue</li> <li>• Map secondToLast digunakan untuk optimisasi kecepatan untuk <i>multiple solution</i>. Cara kerjanya adalah skip append link apabila link tersebut merupakan salah satu dari link kedua dari belakang dari solusi yang sudah kita temukan (jika</li> </ul>

	di append kita hanya akan mendapatkan solusi yang sama).
--	--

```

1 for len(queue) > 0 {           //while queue is not empty
2     select{
3         case :: ctx.Done():
4             return
5         default:
6             numofcheckednodes++
7             currentNode := queue[0] //current branch is the start of the queue, dequeue
8             fmt.Println(len(currentNode.Parents))
9             fmt.Println(resultavailable)
10            fmt.Println("">>>>>>>>>> Skrg kita cek Node: ")
11            currentNode.Print()
12            fmt.Println()
13            queue = queue[1:]
14        }

```

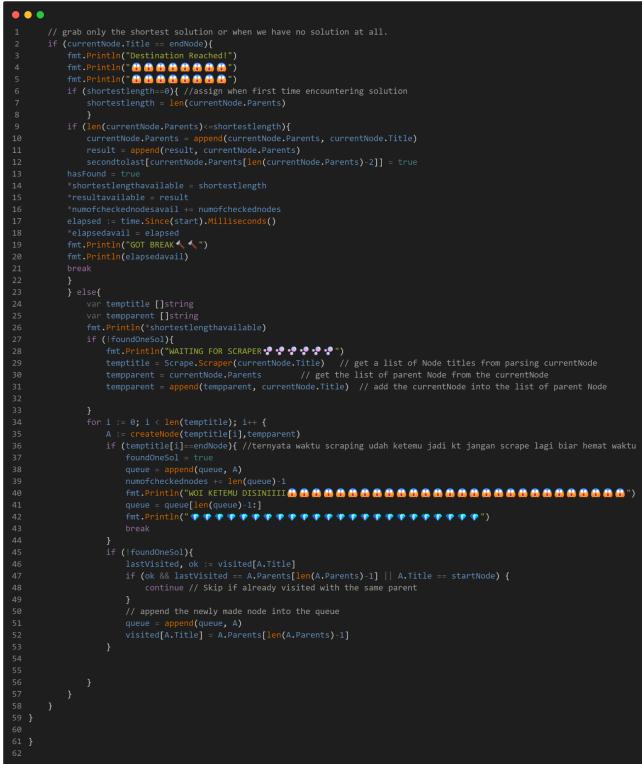
Selama queue simpul hidup tidak kosong, akan dilakukan proses dequeue dan checking. Setiap akan melakukan looping dequeue akan dilakukan dequeue element pertama pada queue untuk dijadikan currentNode, sisakan indeks ke-1 dan seterusnya pada queue.

```

1 // stop if its checking the node with the same parents length as the shortestlengthavailable in parallelBFS
2 if (len(currentNode.Parents) >= shortestlengthavailable && shortestlengthavailable != 0){
3     fmt.Println("shortestlengthavailable")
4     fmt.Println(resultavailable)
5     fmt.Println("THREAD DIHENTIKANNNN●●●●●●●●●●●")
6     "numofcheckednodesavail" == numofcheckednodes
7     return
8 }
9 if _, ok := secondlast[currentNode.Title]; ok {
10    fmt.Println("THREAD DIHENTIKANNNN22222●●●●●●●●●●")
11    continue
12 }
13 if (len(currentNode.Parents) >= shortestlength) && hasFound{
14     fmt.Println("ALREADY GOT BETTER ANSWER STOP ●●●●")
15     "numofcheckednodesavail" == numofcheckednodes
16     return
17 }
18 // grab only the shortest solution

```

Beberapa case handling untuk segera terminate thread apabila sudah ada solusi yang ditemukan. Hal ini terjadi ketika panjang parent currentNode sekarang sama dengan atau lebih dari shortestlengthavailable dan shortestlengthavailable bukanlah 0 (sudah pernah menemuka solusi). Apabila panjang parent yang dicek sekarang masih kurang dari shortestlength, akan tetap dilakukan iterasi walaupun sudah ditemukan sebuah solusi. Hal ini dilakukan untuk memastikan solusi yang didapatkan benar-benar optimal terpendek karena bisa saja karena jumlah link yang sedikit, ditemukan dahulu pada kedalaman degree yang besar. Padahal ternyata pada thread lain bisa ditemukan solusi yang lebih pendek. Masalah routine ini juga bisa mempengaruhi hasil solusi yang ditemukan. Mungkin sekali terjadi perbedaan solusi setiap kali menjalankan iterasi, sebab sebuah thread bisa mendahului thread lain, dan

	<p>algoritma BFS hanya akan mengambil solusi yang pertama didapatkan dan teroptimal.</p>
 <pre> 1 // grab only the shortest solution or when we have no solution at all. 2 if (currentNode.Title == endNode){ 3     fmt.Println("-----") 4     fmt.Println("      ") 5     fmt.Println("      ") 6     if (shortestLength==0) //assign when first time encountering solution 7         shortestLength = len(currentNode.Parents) 8     } 9     if (len(currentNode.Parents)&lt;shortestLength){ 10         currentNode.Parents = append(currentNode.Parents, currentNode.Title) 11         result = append(result, currentNode.Parents) 12         secondtolast(currentNode.Parents[len(currentNode.Parents)-2]) = true 13     hasFound = true 14     shortestLengthAvailable = shortestLength 15     tresultAvailable = result 16     numofcheckednodes = numofcheckednodes 17     elapsed = time.Since(startTime).Milliseconds() 18     elapsedavail = elapsed 19     elapsedavail = elapsed 20     fmt.Println("GOT BREAK&lt; ^-^&gt;") 21     fmt.Println(elapsedavail) 22     break 23 } 24 } else{ 25     var tempTitle []string 26     var tempParent []string 27     fmt.Println("shortestLengthavailable") 28     if (foundOneSol){ 29         fmt.Println("WAITING FOR SCRAPER&lt; ^-^&gt;") 30         tempTitle = Scrape.Scrape(currentNode.Title) // get a list of Node titles from parsing currentNode 31         tempParent = currentNode.Parents // get the list of parent Node from the currentNode 32         tempParent = append(tempParent, currentNode.Title) // add the currentNode into the list of parent Node 33     } 34     for l := 0; l &lt; len(tempTitle); l++ { 35         A = createNode(tempTitle[l],tempParent) 36         if (tempTitle[l]==endNode){ //ternyata waktu scraping udah ketemu jadi kt jangan scrape lagi biar hemat waktu 37             foundOneSol = true 38             queue = append(queue, A) 39             numofcheckednodes += len(queue)-1 40             fmt.Println("WOW KETEMU DISINI!!!") 41             queue = queue[:len(queue)-1] 42             fmt.Println(queue) 43             break 44         } 45         if (!foundOneSol){ 46             lastVisited, ok := visited[A.Title] 47             if (ok &amp;&amp; lastVisited == A.Parents[len(A.Parents)-1]    A.Title == startNode) { 48                 continue // skip if already visited with the same parent 49             } 50             // append the newly made node into the queue 51             queue = append(queue, A) 52             visited[A.Title] = A.Parents[len(A.Parents)-1] 53         } 54     } 55 } 56 } 57 } 58 } 59 } 60 } 61 } 62 }</pre>	

#### 4.1.2.5. Function *ParallelBFS*

Fungsi *ParallelBFS* ini akan menjadi kunci pemanggil semua algoritma BFS. Awalnya akan dilakukan scraping dan pengubahan hasil judul laman menjadi node dengan pemanggilan fungsi *initialBFS*. Pada *ParallelBFS* akan juga akan dibuat variable yang sama dengan tujuan penampung jawaban dari semua thread. Bersifat seperti channel, tetapi di sini akan sharing satu variable yang sama, dan akan diubah isinya dengan passing parameter pointer ke fungsi *multiBFS*. Sebelumnya akan ditentukan dahulu indeks untuk pembagian queue menjadi rata. Iterasi for loop pertama pada fungsi ini digunakan untuk membagi rata panjang queue ke dalam masing-masing thread, dilanjutkan dengan iterasi kedua for loop untuk membagi rata sisa hasil pembagian indeks. Selanjutnya akan dilakukan looping sebanyak jumlah thread untuk memulai

goroutine memanggil fungsi *multiBFS*. Program akan menunggu hingga semua thread selesai dieksekusi dengan waitgroup, lalu akan mengembalikan hasilnya ke main.go untuk diproses menjadi JSON.

```

● ○ ●
1 func ParallelBFS(startTitle string, endNode string) ([][]string, int64, int, int, int){
2     ctx, cancel := context.WithTimeout(context.Background(), 5*time.Minute)
3     defer cancel() // Cancel context to release resources when done
4
5
6     var shortestlength int = 0
7     var result [][] string
8     var numofcheckednodes int = 0
9     var numThreads int = 500
10    var elapsed int64
11    var wg sync.WaitGroup
12    endNode = Scrape.Convert(endNode)
13    shortestlengthavail := &shortestlength
14    resultavailable := &result
15    numofcheckednodesavail := &numofcheckednodes
16    elapsedavail := &elapsed
17    array := make([]int, numThreads)
18    queue := initialBFS(startTitle)
19    //divide queue into equal number of nodes per thread
20    for i := 0; i < numThreads; i++ {
21        array[i] = len(queue) / numThreads
22    }
23    for i := 0; i < len(queue)%numThreads; i++ {
24        array[i]++
25    }
26
27    startIndex := 0
28
29    for i := 0; i < numThreads; i++ {
30        // perform go routine here
31        endIndex := startIndex + array[i]
32        wg.Add(1)
33        go func(start, end int) {
34            defer wg.Done()
35            tes := queue[start:end]
36            multiBFS(tes, startTitle, endNode, shortestlengthavail, resultavailable, numofcheckednodesavail, elapsedavail, ctx)
37        }(startIndex, endIndex)
38        startIndex = endIndex
39    }
40    wg.Wait()
41    return result, elapsed, shortestlength, numofcheckednodes, len(result)
42 }
```

#### 4.1.3. Package Scraper (library gocolly)

Scraping Wikipedia kali ini dibatasi pada Wikipedia English (en) dan akan mengambil daftar judul link yang didapatkan dari string judul masukan user. Masukan string judul dari user masih menggunakan *space*, sehingga perlu dilakukan konversi dengan menggunakan fungsi *Convert* untuk mengkonversi spasi menjadi *underscore* (\_). Dengan menggunakan gocolly, akan diambil judul-judul link yang di-trim setelah */wiki/*. Setelah itu juga diperlukan penyaringan link-link agar tidak mengandung simbol “:/%”, hal ini penting untuk mengurangi jumlah link diperiksa karena diasumsikan tidak mungkin dimasukkan oleh user. Dilanjutkan dengan *append* jika link yang didapatkan tidak terdapat pada daftar list link yang kita punya. Pengecekan duplikasi link juga

diperlukan dengan membuat sebuah map *linksMap* yang menyimpan key judul link dan boolean true (selalu di set true). Penggunaan map ini akan mempercepat proses pengecekan duplikasi O(1) daripada harus iterasi list of links dengan kompleksitas O(n).

#### 4.1.3.1. Function *Scraper*

```
● ● ●
1 package Scrape
2
3 import (
4     "strings"
5     "github.com/gocolly/colly"
6 )
7
8
9 func Scraper(title string) []string {
10     c := colly.NewCollector()
11
12     linksMap := make(map[string]bool) // Map to store links
13     result := []string{}
14     title = Convert(title)
15     c.OnHTML("div.mw-page-container a[href^='/wiki/']", func(h *colly.HTMLElement) {
16         link := h.Request.AbsoluteURL(h.Attr("href"))
17         link = strings.TrimPrefix(link, "https://en.wikipedia.org/wiki/")
18
19         if !strings.ContainsAny(link, ":/%") && link != Convert(title) && !linksMap[link]{
20             linksMap[link] = true // Store link in map
21             result = append(result, link)
22         }
23     })
24
25     c.Visit("https://en.wikipedia.org/wiki/" + title)
26     return result
27 }
```

#### 4.1.3.2. Function *Convert*

```
● ● ●
1 // Input without underscore
2 // Output with underscore
3 func Convert(input string) string {
4     converted := strings.ReplaceAll(input, " ", "_")
5     return converted
6 }
7
```

### 4.1.4. Package Main

#### 4.1.4.1. Function *isValidLink*

Melakukan pengecekan apakah link masukan valid atau benar-benar ada pada Wikipedia. Fungsi *isValidLink* akan mengembalikan nilai true jika link Wikipedia untuk judul tersebut ada (*exists*).

```
● ○ ●
1 package main
2
3 import (
4     "encoding/json"
5     "Tubes2_gomugomuno/BFS"
6     "Tubes2_gomugomuno/IDS"
7     "Tubes2_gomugomuno/Scrape"
8     "fmt"
9     "net/http"
10 )
11
12 // Checks if the given Wikipedia link is valid
13 func isValidLink(title string) bool {
14     title = Scrape.Convert(title)
15     url := fmt.Sprintf("https://en.wikipedia.org/wiki/%s", title)
16     resp, err := http.Head(url)
17     if err != nil {
18         fmt.Println("Error:", err)
19         return false
20     }
21     defer resp.Body.Close()
22
23     if resp.StatusCode == http.StatusOK {
24         fmt.Println("Link is valid")
25         return true
26     } else {
27         fmt.Println("Link is invalid")
28         return false
29     }
30 }
31
```

#### 4.1.4.2. Function *printResult*

Fungi *printResult* digunakan untuk testing debugging, menampilkan solusi yang didapatkan.

```
● ○ ●
1 // Print all results found
2 func printResult(result [][]string) {
3     fmt.Println("Printing result:")
4     for _, innerSlice := range result {
5         fmt.Print("(")
6         for i, str := range innerSlice {
7             fmt.Print(str)
8             // If not the last element in the slice, add a comma
9             if i < len(innerSlice)-1 {
10                 fmt.Print(",")
11             }
12         }
13         fmt.Println(")")
14     }
15 }
```

#### 4.1.4.3. Function executeAlgorithm

Fungsi *executeAlgorithm* akan melakukan fetching input data user dari form website, melakukan pengecekan apakah link masukan user valid dengan *isValidLink*, memastikan start dan finish title tidak kosong ataupun sama. Setelah mendapatkan hasilnya berdasarkan algoritma yang terpilih, executeAlgorithm akan struct response menjadi JSON agar bisa dipanggil di bagian frontend

```
1 // Processing data received from the frontend
2 func executeAlgorithm(w http.ResponseWriter, r *http.Request, algorithm AlgorithmFunc) {
3     startNode := r.URL.Query().Get("startNode")
4     endNode := r.URL.Query().Get("endNode")
5
6     if startNode == "" || endNode == "" {
7
8         http.Error(w, "Missing start or end node", http.StatusBadRequest)
9         return
10    }
11
12    if startNode == endNode{
13        http.Error(w, "Start and end node cannot be the same.", http.StatusBadRequest)
14        return
15    }
16
17    if !isValidLink(startNode) {
18        http.Error(w, "Invalid start node", http.StatusNotFound)
19        return
20    }
21
22    if !isValidLink(endNode) {
23        http.Error(w, "Invalid end node", http.StatusNotFound)
24        return
25    }
26
27    result, elapsed, shortestlength, numofcheckednodes, path := algorithm(startNode, endNode)
28    fmt.Println("⚡ ",elapsed)
29
30    response := struct {
31        Result    [][]string `json:"result"`
32        Elapsed   int64      `json:"elapsed"`
33        Shortest  int        `json:"shortestlength"`
34        Checked   int        `json:"numofcheckednodes"`
35        Path      int        `json:"path"`
36    }{result, elapsed, shortestlength, numofcheckednodes, path}
37
38    json.NewEncoder(w).Encode(response)
39 }
```

#### 4.1.4.4. Function *main*

Pada fungsi Main inilah semua algoritma dan packages akan dipanggil. Dengan route API tertentu akan dipanggil algoritma yang cocok antara BFS dan IDS. Di sini juga dilakukan set access control supaya website diberi permission untuk tidak di-block oleh firewall dan bisa melakukan fetch dan post antara website dan algoritma.

```

1
2 // Connecting the backend with the frontend
3 func main() {
4     handleAlgorithm := func(endpoint string, algorithm func(http.ResponseWriter, *http.Request)) {
5         http.HandleFunc(endpoint, func(w http.ResponseWriter, r *http.Request) {
6             fmt.Println("Enter", endpoint)
7             w.Header().Set("Access-Control-Allow-Origin", "http://localhost:3000")
8             w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")
9             w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
10            algorithm(w, r)
11        })
12    }
13
14    handleAlgorithm("/api/bfs", func(w http.ResponseWriter, r *http.Request) {
15        executeAlgorithm(w, r, bfs.ParallelBFS)
16    })
17
18    handleAlgorithm("/api/ids", func(w http.ResponseWriter, r *http.Request) {
19        executeAlgorithm(w, r, ids.IDS)
20    })
21
22    fmt.Println("Server is running on :3000")
23    http.ListenAndServe(":3000", nil)
24 }

```

## 4.2. Struktur Data dan Implementasi

```

.
├── README.md
├── doc
│   └── gomugomuno.pdf
└── src
    ├── frontend
    │   ├── .gitignore
    │   ├── package.json
    │   ├── package-lock.json
    │   ├── README
    │   └── public
    │       └── src
    │           ├── App.js
    │           ├── App.css
    │           ├── index.js
    │           ├── index.css
    │           ├── package-lock.json
    │           ├── package.json
    │           └── About
    │               ├── About.js
    │               └── About.css
    └── Content

```

```

    |   |   |   └── Content.js
    |   |   |   └── Content.css
    |   |   └── Footer
    |   |   |   └── footer.js
    |   |   |   └── footer.css
    |   |   └── GraphComponent
    |   |   |   └── GraphComponent.js
    |   |   |   └── GraphComponent.css
    |   |   └── Header
    |   |   |   └── Header.js
    |   |   |   └── Header.css
    |   |   └── How To Use
    |   |   |   └── HTU.js
    |   |   |   └── HTU.css
    |   └── backend
    |   |   └── go.mod
    |   |   └── go.sum
    |   |   └── Dockerfile
    |   |   └── main.go
    |   |   └── BFS
    |   |   |   └── BFS.go
    |   |   |   └── BFSfunction.go
    |   |   └── IDS
    |   |   |   └── ids.go
    |   └── Scrape
    |   |   └── scraper.go

```

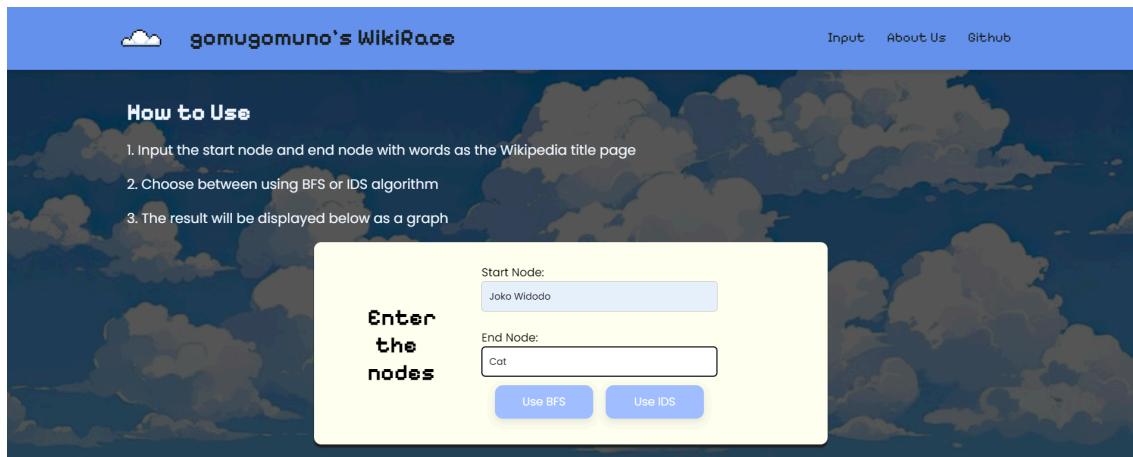
### 4.3. Cara Penggunaan Program

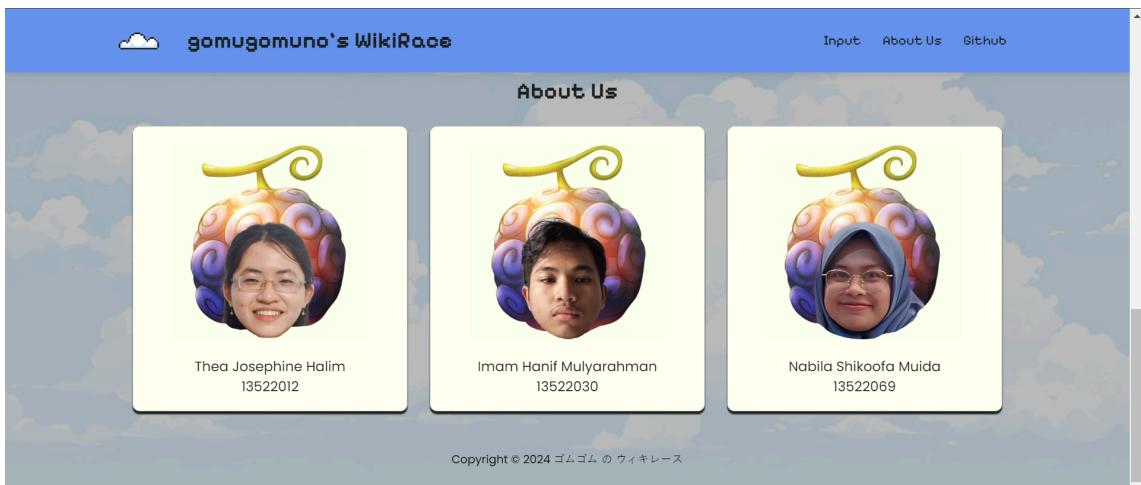
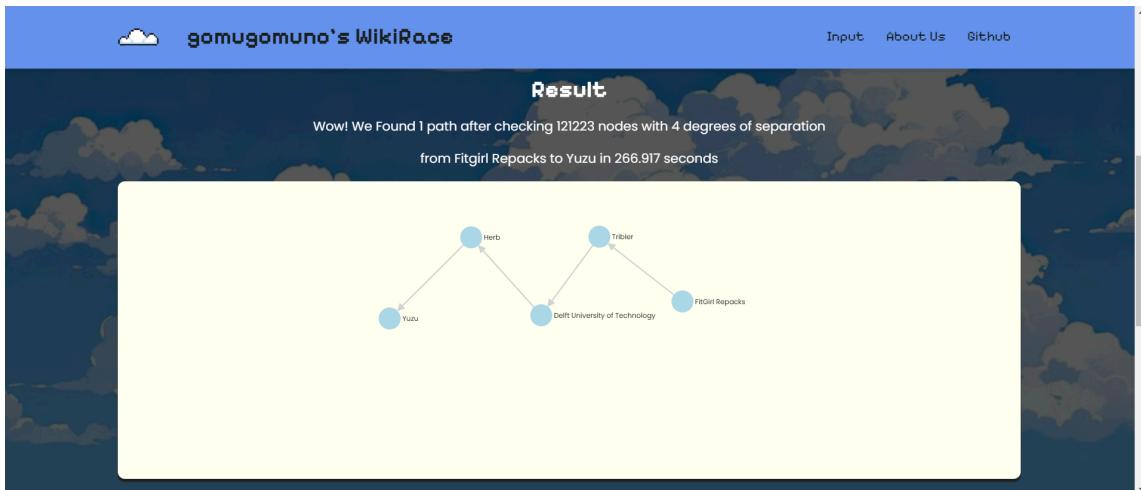
- 1) Buka IDE Visual Studio Code
- 2) Buka dua terminal dan navigasi terminal pertama ke folder frontend dan lainnya backend.
- 3) Lakukan “npm install” pada folder frontend, lalu ketikkan “npm start”. Jika terjadi error, ulangi lagi dengan “npm install react-d3-graph@2.6.0 --legacy-peer-deps” dan “npm start”

- 4) Pada folder backend ketikkan “go run main.go” untuk start server setelah website menyala dan terbuka di browser localhost:3000. Jika terjadi error 404, ulangi starting server dengan melakukan run ulang main.go.
- 5) Apabila muncul peringatan firewall block yang menghambat main, select allow.
- 6) Program siap dijalankan bila pesan “Server is running on :3000” pada terminal backend.
- 7) Masukkan inputan judul laman awal dan akhir pada form inputan. Website akan memberikan peringatan apabila salah satu dari inputan kosong atau keduanya sama (0 degree).
  - a) Pilih algoritma yang ingin digunakan untuk searching lalu tekan tombol yang sesuai, hasil akan muncul dalam < 5 menit dalam bentuk graf.
  - b) Ulangi proses masukan untuk mencoba judul-judul lain.

#### 4.4. Tampilan Antarmuka

Antarmuka website dibagi menjadi 3 bagian, penjelasan cara penggunaan (how to use), *content* (input node dan tampilan hasil), serta about us. Tersedia juga navigation bar ke *section* input dan *about us*, serta link menuju repository github.





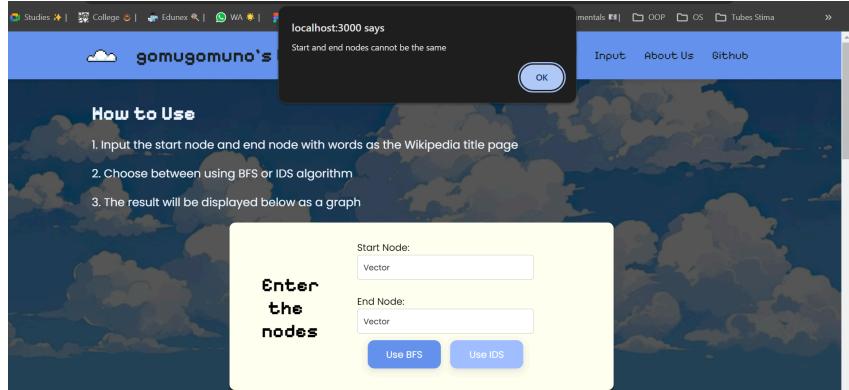
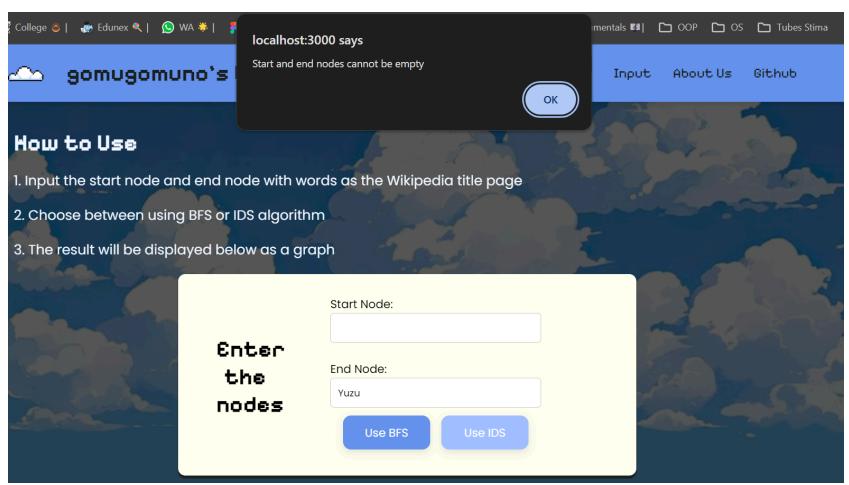
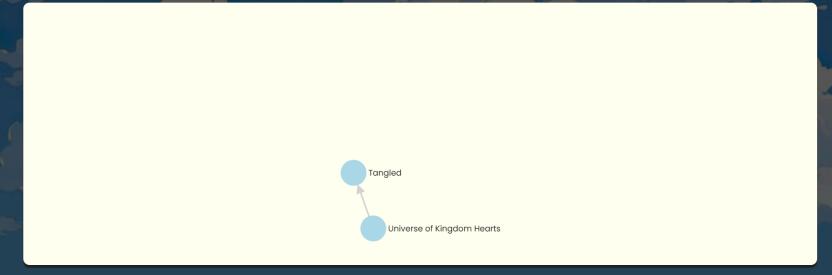
## 4.5. Hasil Pengujian

Hasil Pengujian dari 0 hingga 4 degrees, berdasarkan QNA nomor 38 dengan jumlah hop tidak akan lebih dari 4 (max 4 degrees of separation).

### 4.5.1. Pengujian Breadth First Search

Tabel 4.4.1 Hasil Pengujian BFS

Input/Output	Keluaran
--------------	----------

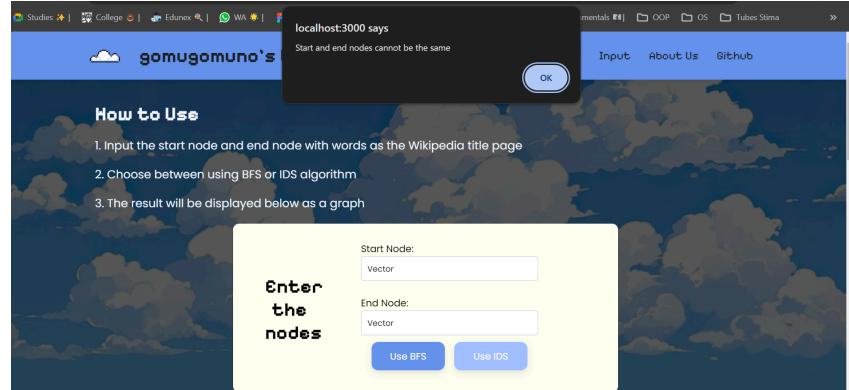
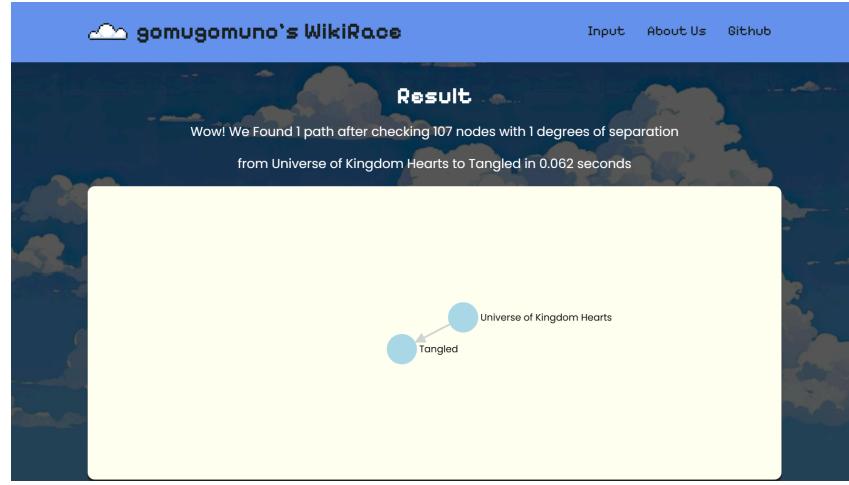
	 <p>0 degree</p> <p>Start: Vector</p> <p>End: Vector</p>
	 <p>Start Node: <input type="text"/></p> <p>End Node: <input type="text"/> Yuzu</p> <p>Use BFS Use IDS</p>
1 degree	<p>Result</p> <p>Wow! We Found 1 path after checking 9463 nodes with 1 degrees of separation from Universe of Kingdom Hearts to Tangled in 0.227 seconds</p>  <p>Tangled</p> <p>Universe of Kingdom Hearts</p>

<p>2 degrees</p> <p>Start: Joko Widodo</p> <p>End: YouTube</p>	<p><b>Result</b></p> <p>Wow! We Found 1 path after checking 10572 nodes with 2 degrees of separation from Joko Widodo to YouTube in 2.289 seconds</p>
<p>3 degrees</p> <p>Start: Vector</p> <p>End: Knowledge</p>	<p><b>Result</b></p> <p>Wow! We Found 1 path after checking 225415 nodes with 3 degrees of separation from Vector to Knowledge in 0.684 seconds</p>
<p>4 degrees</p> <p>Start: Fitgirl Repacks</p> <p>End: Yuzu</p>	<p><b>Result</b></p> <p>Wow! We Found 1 path after checking 121223 nodes with 4 degrees of separation from FitGirl Repacks to Yuzu in 266.917 seconds</p>

#### 4.5.2. Pengujian Iterative Deepening Search

Tabel 4.4.2 Hasil Pengujian IDS

Start/End	Keluaran

	<p>0 degree</p> <p>Start: Vector</p> <p>End: Vector</p> 
	<p>1 degree</p> <p>Start: Universe of Kingdom Hearts</p> <p>End: Tangled</p> 

<p>2 degrees</p> <p>Start: Joko Widodo</p> <p>End: YouTube</p>	<p>gomugomuno's WikiRace</p> <p><b>Result</b></p> <p>Wow! We Found 1 path after checking 376358 nodes with 2 degrees of separation from Joko Widodo to YouTube in 8.765 seconds</p> <pre> graph TD     Joko[Widodo] --&gt; John[John Lee Ka-chiu]     John --&gt; YouTube[YouTube]   </pre>
<p>3 degrees</p> <p>Start: Vector</p> <p>End: Knowledge</p>	<p>gomugomuno's WikiRace</p> <p><b>Result</b></p> <p>Wow! We Found 1 path after checking 33977 nodes with 3 degrees of separation from Vector to Knowledge in 0.898 seconds</p> <pre> graph TD     Vector[Vector] --&gt; VectorMath[Vector (mathematics and physics)]     VectorMath --&gt; Mathematics[Mathematics]     Mathematics --&gt; Knowledge[Knowledge]   </pre>
<p>4 degrees</p> <p>Start: Fitgirl Repacks</p> <p>End: Yuzu</p>	<p>gomugomuno's WikiRace</p> <p><b>Result</b></p> <p>Wow! We Found 1 path after checking 12249009 nodes with 4 degrees of separation from Fitgirl to Yuzu in 300.189 seconds</p> <pre> graph TD     Yuzu[Yuzu] --&gt; Herb[Herb]     Herb --&gt; TU[Delft University of Technology]     TU --&gt; Tribler[Tribler]     Tribler --&gt; Fitgirl[Fitgirl]   </pre>

## 4.6. Analisis Hasil Pengujian

### 4.6.1 Faktor yang Mempengaruhi Kecepatan Algoritma

- Kecepatan internet

Kecepatan internet yang tinggi mempercepat proses scraping website.

Kedua algoritma sering melakukan scraping sehingga dibutuhkan kecepatan scraping yang tinggi.

- Banyaknya Link pada Suatu Laman Wikipedia

Jumlah link yang berada pada setiap halaman menentukan kecepatan algoritma. Semakin banyak link pada suatu laman maka semakin banyak pula link yang perlu di-scraping. Hal ini meningkatkan waktu yang dibutuhkan untuk mencari lintasannya.

- Letak link yang dicari

Letak artikel setelahnya yang dicari juga dapat mempengaruhi cepat atau lambatnya algoritma. Apabila ternyata jalur yang benar untuk link yang dicari di bagian atas sebuah website dapat mempercepat proses pencarian karena algoritma scraping membaca dan memproses konten halaman web dari atas ke bawah dalam satu artikel.

- Multi Thread

Penggunaan multithread go routine bisa mempercepat proses hingga berkali-kali lipat tergantung pada jumlah thread. Pada algoritma BFS, go routine digunakan untuk mengecek bagian-bagian initial queue yang berbeda dalam satu waktu secara langsung. Akan tetapi, perlu diperhatikan juga bahwa penggunaan thread yang terlalu banyak, terutama jika melakukan multithreading setiap kali melakukan iterasi dapat menyebabkan overhead ataupun *ter-block* oleh Wikipedia, sehingga kita perlu membatasi jumlah thread yang digunakan. Kita juga harus berhati-hati agar tidak mengakses suatu variabel penting secara bersamaan agar tidak terjadi deadlock, sehingga perlu menanganinya dengan mutex.

Walaupun begitu, go routine menjadi kunci penting terutama pada tugas kali ini, karena kita harus melakukan proses yang sama secara berulang-ulang. Bila proses yang berulang ini bisa kita lakukan dalam satu

waktu secara sekaligus, kita bisa mempercepat algoritma hingga berkali-kali lipat.

- Caching

Penggunaan cache akan mengurangi jumlah scraping web yang dilakukan selama run time sehingga mempercepat program. Penggunaan cache lebih efektif pada IDS karena algoritma ini melakukan DLS secara berulang. Dengan menggunakan cache, waktu untuk scrapping dapat dipangkas sehingga menghasilkan algoritma yang lebih cepat.

Contoh caching ini bisa kita lihat dari melakukan pencarian yang sama dua kali. Pada proses yang pertama kita mungkin membutuhkan waktu lebih lama untuk scraping karena cache device yang masih kosong atau *miss*. Ketika kita sudah melakukan proses pencarian, dalam pencarian yang kedua kita mungkin masih memiliki cache data pencarian sebelumnya, sehingga proses menjadi lebih cepat.

Berikut adalah contoh nyata konsep caching. Akan dilakukan pencarian dari laman Indonesia ke Korea dengan menggunakan IDS. Pada percobaan pertama didapatkan hasilnya dalam waktu 1.266 detik. Sedangkan pada percobaan kedua yang dilakukan langsung setelah percobaan pertama didapatkan hasilnya hanya 898 millisecond.





#### 4.6.2 Analisis Hasil Pengujian

Berdasarkan pengujian yang telah dilakukan pada beberapa degree, kita dapat memetakannya dalam tabel perbandingan sebagai berikut:

Tabel 4.6.2.1 Perbandingan Algoritma

Degrees	Start and End	Kompleksitas Waktu (ms)		Artikel yang Diperiksa	
		IDS	BFS	IDS	BFS
1 Degree	Start: Koji Kondo End: Universe of The Legend of Zelda	66	271	294	4577
	Start: Vector End: vectorman	71	118	51	131
	Start: Superman End : Batman	20	625	623	4419
2 Degree(s)	Start: Nabila End: Japan	982	82	922	325
	Start: Cloud End: Sky	10585	2143	46951	5195
	Start: Nintendo End: Philips	22	2852	198	7038
3 Degree(s)	Start: Open world End : Legislator	10195	1857	82432	694
	Start: Vector End: Knowledge	898	684	33977	225485
	Input: Saman End : Roger	16569	23404	124151	6208

Berdasarkan tabel perbandingan yang kita lakukan, kita melihat bahwa penggunaan algoritma IDS ataupun BFS tidak selalu tetap untuk test case yang berbeda-beda. Pada 1 degree of separation IDS lebih cepat daripada BFS karena pada iterasi pertama IDS langsung melakukan pengecekan terhadap link dan mengembalikan solusi apabila sudah ditemukan, sedangkan pada algoritma BFS perlu dilakukan pembacaan initial queue dari node masukan menggunakan fungsi *initialBFS*, sehingga memerlukan waktu lebih lama untuk diproses. Hal ini juga menjelaskan mengapa jumlah node yang dicek oleh BFS lebih banyak.

Pada degree lain, posisi link yang dicari pada laman mungkin bisa berada di paling atas ataupun di bawah laman. Ditambah juga dengan adanya faktor multithreading dengan menggunakan routine sehingga hasil yang didapat mungkin tidak konsisten karena thread yang berjalan bersamaan. Faktor internet dan lain-lain mungkin juga membuat ketidakstabilan pada hasil yang didapat.

Dalam hal kompleksitas waktu, IDS dan BFS sama yaitu  $O(b^d)$  dengan  $b$  adalah jumlah anak dan  $d$  adalah kedalaman. Akan tetapi, pada kompleksitas ruang/memori IDS menggunakan memori yang lebih sedikit yaitu  $O(b \times d)$  daripada BFS  $O(b^d)$ . Hal ini dikarenakan IDS melakukan pengecekan hingga kedalam dan tidak menyimpan informasi queue simpul hidup, sedangkan BFS perlu menyimpan queue simpul hidup yang mungkin panjangnya hingga ribuan bahkan ratusan dalam kasus WikiRace ini.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Tugas besar 2 IF2211 Strategi Algoritma mengungkit masalah mengenai WikiRace, di mana kita harus mencari jalur terdekat dan tercepat dengan menggunakan algoritma *Breadth First Search* (BFS) dan *Iterative Deepening Search* (IDS). Algoritma BFS akan melakukan pengecekan dengan konsep queue, dequeue link dari queue simpul hidup, lakukan pengecekan, jika bukan yang dicari akan dilakukan scraping untuk enqueue semua link ke queue lagi. Proses dilakukan hingga queue kosong atau sudah ditemukan solusi. Pada IDS akan melakukan pengecekan dengan konsep pohon. Pada simpul akar akan dilakukan pencarian secara DFS dengan tingkat kedalaman tertentu. Apabila tidak ditemukan solusi maka tingkat kedalaman ditingkatkan 1 level. Proses dilakukan hingga menemukan solusi atau seluruh bagian pohon dikunjungi.

#### **5.2. Saran**

Selama penggerjaan tugas besar ini kami memiliki beberapa saran untuk penggerjaan tugas besar berikutnya, di antaranya:

- Mencari referensi lebih banyak lagi sebagai bahan literasi penambah wawasan.
- Membuat sheets berisi link-link terintegrasi yang dibutuhkan selama penggerjaan tugas serta jadwal yang jelas terkait waktu pertemuan maupun *soft deadline* untuk setiap task.
- Mendebug program *as soon as possible* dan menyamakan persepsi anggota terhadap target tugas besar

#### **5.3. Refleksi**

Tugas Besar 2 IF2211 Strategi Algoritma Semester II Tahun 2023/2024 mengajarkan kami banyak hal, terutama dalam hal kerjasama. Kerjasama kami diuji pada tugas besar ini karena waktu penggerjaan yang bertepatan dengan libur lebaran. Kami belajar bagaimana cara bekerja sama walaupun tidak bisa bertemu secara langsung dan memastikan bahwa kami memiliki pemikiran yang sama. Kami juga belajar untuk

membagi waktu antara waktu untuk penggerjaan tugas dan waktu untuk acara keluarga. Selain itu, kami juga belajar bahasa pemrograman yang benar-benar baru buat kami, yaitu Golang. Ditambah dengan keharusan implementasi pada website, tugas besar ini pastinya menjadi tantangan besar buat kami semua. Melalui tugas besar ini kami juga menyadari alasan penggunaan Golang karena adanya kemampuan untuk melakukan routine, yaitu melakukan hal yang sama berkali-kali dalam waktu yang sama. Ini memberi perubahan besar dalam kecepatan pemrosesan, sebab kami dapat membagi proses menjadi X dan mempercepatnya X kali lipat.

## LAMPIRAN

### Repository

[https://github.com/nabilashikoofa/Tubes2\\_gomugomuno](https://github.com/nabilashikoofa/Tubes2_gomugomuno)

### Video

<https://youtu.be/mPReoDKy7Lw>

## DAFTAR PUSTAKA

Adumb. (2024, 30 Maret). I Made a Graph of Wikipedia... This Is What I Found [Video].

<https://www.youtube.com/watch?v=JheGL6uSF-4>

Asisten, S. A. (2024). Tugas Besar 2 IF2211 Strategi Algoritma 2024. docx. Diakses dari Google Dokumen: <https://docs.google.com/document>

GeeksforGeeks. (n.d.). Graph Data Structure and Algorithms. Diakses pada tanggal 3 April 2024, dari <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

JohnWatsonRooney. (2023, 3 Februari). Web Scraping with GO... Easy AND Fast?! [Video]. YouTube. <https://www.youtube.com/watch?v=wUSgA8WEy4Q&t=335s>

Munir, R. (2023). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1). Diakses dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir>

Munir, R. (2023). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2). Diakses dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir>