

**Laporan Tugas Besar 2  
IF2211 Strategi Algoritma WikiRace  
Semester II Tahun 2023/2024**



Disusun Oleh:  
Muhammad Althariq Fairuz 13522027  
Nyoman Ganadipa Narayana 13522066  
Azmi Mahmud Bazeid 13522109

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

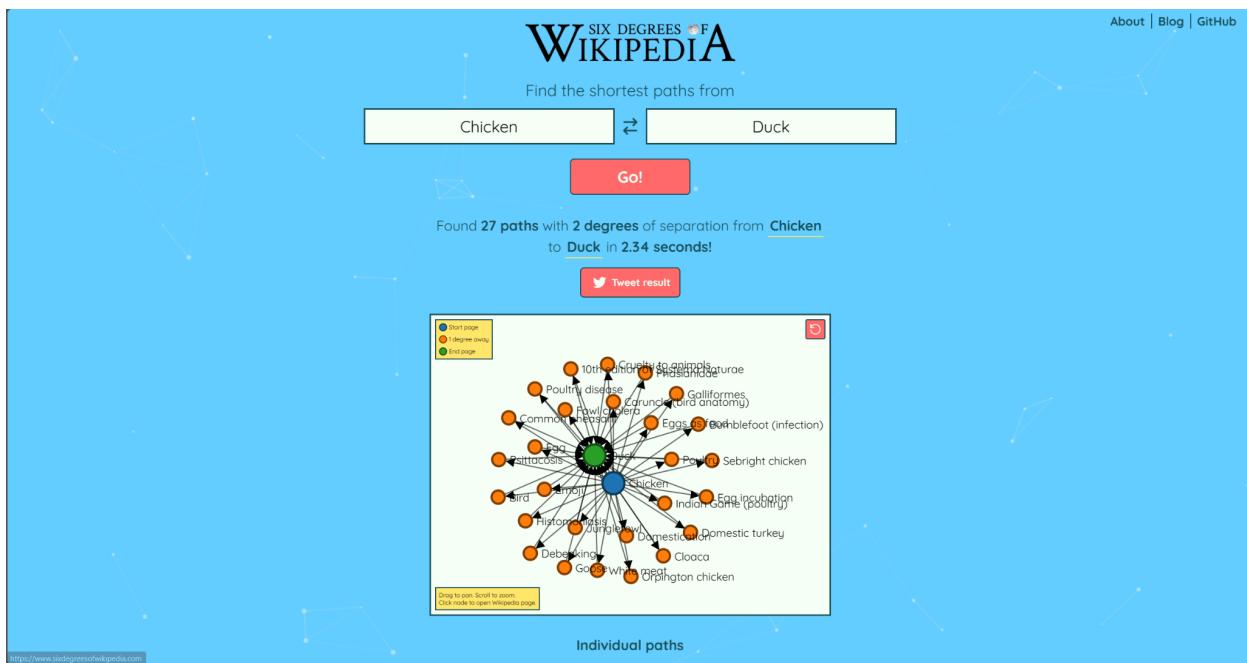
## DAFTAR ISI

|  |           |
|--|-----------|
| <b>BAB 1</b>   | <b>1</b>  |
| <b>Deskripsi Tugas</b>   | <b>1</b>  |
| <b>BAB 2</b>   | <b>2</b>  |
| <b>Landasan Teori</b>  | <b>2</b>  |
| 2.1. Dasar Teori   | 2         |
| 2.2. Penjelasan Mengenai Aplikasi Web yang Dibangun                      | 2         |
| 2.2.1 Aplikasi Web bagian Front End                                      | 2         |
| 2.2.2 Integrasi antara Front End dan Back End                            | 3         |
| 2.2.3 Aplikasi Web bagian Back End                                       | 3         |
| <b>BAB 3</b>   | <b>4</b>  |
| <b>Analisis Pemecahan Masalah</b>  | <b>4</b>  |
| 3.1. Langkah-langkah Pemecahan Masalah                                   | 4         |
| 3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma IDS dan BFS | 4         |
| 3.3. Fitur fungsional dan arsitektur aplikasi web yang dibangun          | 5         |
| 3.4. Ilustrasi Kasus   | 5         |
| <b>BAB 4</b>   | <b>15</b> |
| <b>Implementasi dan Pengujian</b>  | <b>15</b> |
| 4.1. Spesifikasi Teknis Program  | 15        |
| 4.2. Tata Cara Penggunaan Program  | 22        |
| 4.2.1 Setup repository   | 22        |
| 4.3. Hasil dan Analisis Pengujian  | 22        |
| <b>BAB 5</b>   | <b>28</b> |
| <b>Kesimpulan dan Saran</b>  | <b>28</b> |
| 5.1. Kesimpulan  | 28        |
| 5.2. Saran   | 28        |
| <b>Lampiran</b>  | <b>29</b> |
| <b>Daftar Pustaka</b>  | <b>30</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.



**Gambar 1.1** Tampilan permainan WikiRace

## **BAB 2**

### **Landasan Teori**

#### **2.1. Dasar Teori**

Graf adalah struktur data yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Secara sederhana, graf didefinisikan sebagai kumpulan titik yang dihubungkan oleh sisi.

Graf dapat dijelajah dengan menggunakan beberapa macam algoritma. Penggunaan algoritma penjelajahan sebuah graf digunakan sesuai dengan kebutuhan dan sifat dari masalah yang dihadapi.

DFS (Depth-first search) adalah algoritma yang digunakan untuk menjelajahi graf yang memprioritaskan simpul yang paling dalam terlebih dahulu (atau dalam kata lain, simpul yang belum pernah divisit untuk kasus graf yang tidak pohon). DFS menggunakan kompleksitas waktu  $O(V+E)$  (karena harus melakukan iterasi setiap simpul dan untuk setiap simpul harus melakukan iterasi setiap neighbour) dan kompleksitas ruang yang linier terhadap banyaknya simpul (untuk menyimpan simpul mana yang sudah divisit).

BFS (Breadth-First Search) adalah algoritma yang digunakan untuk menjelajahi graf berdasarkan jarak dari source. Algoritma ini memulai pencariannya dari node awal atau root node dan kemudian mengunjungi semua node yang berdekatan dengan node yang dipilih. BFS melakukan perjalanan secara “luas” daripada “mendalam”. Misalnya, jika kita memiliki sebuah pohon, BFS akan mengunjungi semua node di level yang sama sebelum pindah ke level berikutnya.

Iterative Deepening Search (IDS) adalah algoritma pencarian graf yang menggabungkan konsep dari Breadth First Search (BFS) dan Depth First Search (DFS). IDS menggabungkan efisiensi ruang pencarian depth-first dan pencarian cepat breadth-first (untuk node yang lebih dekat ke root). IDS bekerja dengan cara memanggil DFS untuk kedalaman yang berbeda mulai dari nilai awal. Dalam setiap pemanggilan, DFS dibatasi dari melampaui kedalaman yang diberikan dengan peningkatan nilai kedalaman dan berhenti sampai solusi ditemukan.

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

##### **2.2.1 Aplikasi Web bagian Front End**

Pada bagian front-end digunakan library reactjs dan typescript sebagai pembangun dari aplikasi web. Front end memiliki landing page, not found page, dan main page. Pada bagian main page, user diharapkan memasukkan input dari titik awal dan titik akhir suatu judul wikipedia, serta

suatu pilihan untuk metode skema pencarian antara IDS atau BFS, user juga diharapkan memilih antara pilihan tampilkan semua shortest path atau hanya satu saja.

Saat user telah memasukkan input dengan valid, request dilakukan kepada backend dengan endpoint tertentu, kemudian backend memproses masukan tersebut kemudian mengembalikannya sebagai response.

### **2.2.2 Integrasi antara Front End dan Back End**

Tipe data payload dan tipe data response (ekspektasi response) dibentuk di kedua bagian untuk memberikan kenyamanan saat runtime. Backend menerima payload dalam bentuk tipe yang diinginkan backend, frontend pun juga menyimpan tipe tersebut. Frontend menerima response dalam bentuk tipe yang diinginkan oleh frontend, backend pun juga menyimpan tipe-tipe tersebut.

### **2.2.3 Aplikasi Web bagian Back End**

Pada bagian backend, payload yang diterima oleh backend sudah dalam bentuk tipe yang terstruktur, response yang harus dikirim pun juga sudah terstruktur sesuai dengan keinginan dari frontend sebelumnya. Pada bagian ini, request yang dikirimkan oleh frontend diproses dengan memanggil fungsi IDS atau BFS, kemudian merespons dengan memproses hasil IDS dan BFS kembali ke bentuk response yang diinginkan oleh frontend.

## BAB 3

### Analisis Pemecahan Masalah

#### 3.1. Langkah-langkah Pemecahan Masalah

- Modelkan persoalan WikiRace sebagai persoalan graf:  
Kita asosiasikan setiap *wiki article* dengan sebuah node/simpul. Sebuah sisi/edge berarah menghubungkan dua node/simpul dari A ke B jika dan hanya jika ada sebuah hyperlink di *wiki article* A yang ketika diklik menuju ke *wiki article* B.
- Gunakan algoritma traversal/penjelajahan graf:  
Kita gunakan algoritma BFS dan IDS untuk mencari jarak yang terdekat antara simpul source dengan simpul destination

#### 3.2. Proses pemetaan masalah menjadi elemen-elemen algoritma IDS dan BFS

##### a. BFS

Untuk setiap *depth*/kedalaman dimulai dari  $d=0$  (dari root node atau *source*) dan bertambah satu, kita lakukan sebagai berikut:

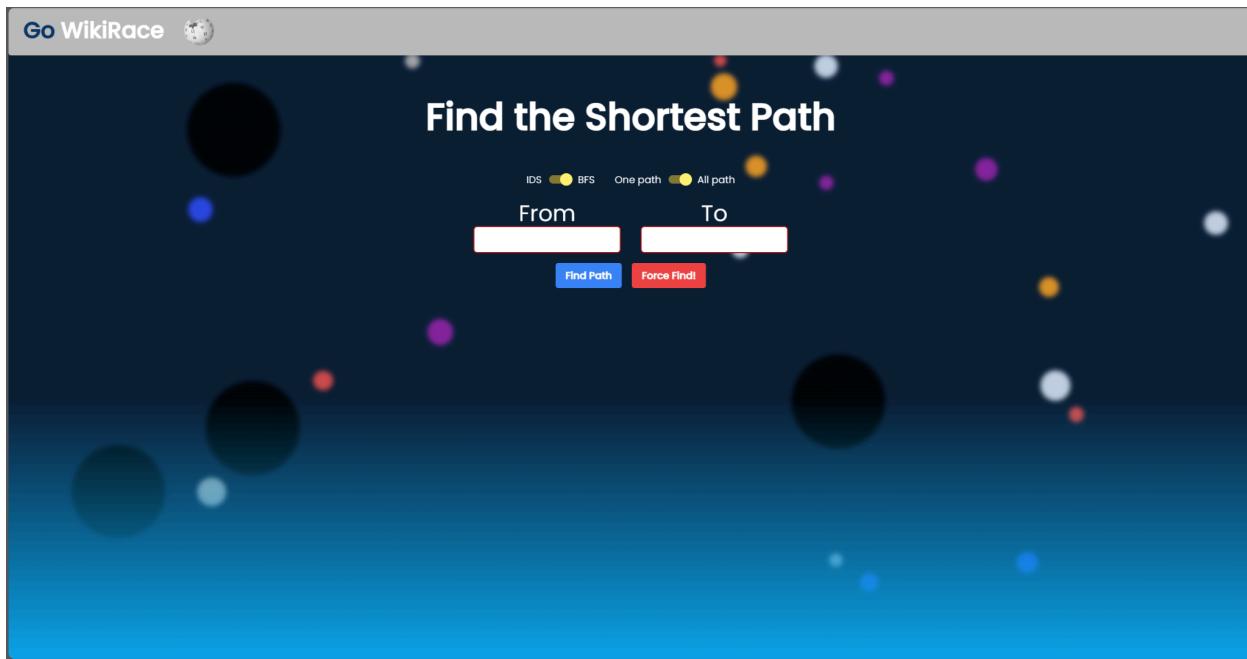
1. Crawling *source*
2. Terima hasil data crawling semua nodes yang di depth d. Simpan dalam sebuah map yang merupakan adjacency list.
3. Setelah semua hasil crawling telah diterima, iterasi setiap node di depth d. Iterasi semua node di depth d. Iterasi setiap neighbour di setiap node.
  - Jika neighbournya adalah *destination*, maka telah ketemu solusi.
  - Jika neighbournya sudah pernah di visit (diindikasikan dengan tercatat jaraknya dari *source* di sebuah variabel map dengan key string dan value jarak dari *source*), maka lanjutkan ke neighbour berikutnya.
  - Jika neighbournya belum pernah di visit, catatlah bahwa pernah divisit (catatlah jarak dari *source* di sebuah variabel map), pushlah node tersebut ke queue, dan buatlah goroutine untuk lakukan crawling node yang bersangkutan.
4. *Increment* d dan balik ke step 2.

##### b. IDS

Untuk setiap *depth*/kedalaman dimulai dari  $d=0$  (dari root node atau *source*) dan bertambah satu, kita lakukan sebagai berikut:

1. Carilah semua solusi hingga depth/kedalaman d menggunakan DLS.
2. Jika terdapat solusi, maka setelah DLS selesai, semua solusi terpendek sudah tercatat. Program selesai.
3. Jika masih belum terdapat solusi, panggil goroutine untuk crawling semua node di kedalaman d.
4. Tunggu crawling semua node dalam kedalaman d selesai, dan simpan dalam sebuah map yang merupakan adjacency list.
5. *Increment* d dan balik ke step 1 .

### 3.3. Fitur fungsional dan arsitektur aplikasi web yang dibangun



**Gambar 3.3.1** Tampilan website yang dibangun

Web yang dibangun memiliki beberapa fitur yang mirip seperti website aslinya (WikiRace), diantaranya:

1. Mencari jarak terdekat antara *source* dan *destination* dengan menggunakan BFS dan IDS.
2. (BONUS) Mencari semua path antara *source* dan *destination* dengan menggunakan BFS dan IDS.
3. (BONUS) Memberikan visualisasi graf yang menggambarkan semua *path* antara *source* dan *destination*.
4. Menampilkan waktu yang dibutuhkan.
5. Menampilkan total nodes yang ada pada graf yang dikonstruksi.

Serta meningkatkan developer experience dengan menggunakan docker setup (BONUS).

### 3.4. Ilustrasi Kasus

3.4.1. Kasus 1: Pencarian semua shortest path dari judul Chicken ke Duck menggunakan BFS

1. User memilih pilihan BFS dan pilihan All path
2. User memasukkan input field from dengan “Chicken” dan input field to dengan “Duck”, kemudian submit
3. Aplikasi frontend mengirimkan request berupa tipe data yang berisi

```
{  
    source = "Chicken"  
    target = "Duck"  
    using_bfs = true  
    all_paths = true  
}
```

4. Aplikasi menangkap request tersebut dan segera melakukan proses pencarian sesuai dengan payload yang dikirimkan, dalam hal ini pemanggilan fungsi traversal.MultiPathBFS(“Chicken”, “Duck”) dan menyimpan hasilnya.
5. Hasil tersebut kemudian diproses sehingga membentuk tipe data Response yang diinginkan frontend, jika proses berjalan dengan lancar, backend akan memberikan response sbb

```
{  
    "data": {  
        "nodes": [  
            {  
                "id": 0,  
                "label": "Chicken",  
                "url": "https://en.wikipedia.org/wiki/Chicken",  
                "level": 0  
            },  
            {  
                "id": 1,  
                "label": "Poultry",  
                "url": "https://en.wikipedia.org/wiki/Poultry",  
                "level": 1  
            },  
            {  
                "id": 2,  
                "label": "Duck",  
                "url": "https://en.wikipedia.org/wiki/Duck",  
                "level": 2  
            },  
            {  
                "id": 3,  
                "label": "Galliformes",  
                "url": "https://en.wikipedia.org/wiki/Galliformes",  
                "level": 3  
            }  
        ]  
    }  
}
```

```
"url": "https://en.wikipedia.org/wiki/Galliformes",
"level": 1
},
{
"id": 4,
"label": "Junglefowl",
"url": "https://en.wikipedia.org/wiki/Junglefowl",
"level": 1
},
{
"id": 5,
"label": "Template talk:Poultry",
"url": "https://en.wikipedia.org/wiki/Template talk:Poultry",
"level": 1
},
{
"id": 6,
"label": "Parasitism",
"url": "https://en.wikipedia.org/wiki/Parasitism",
"level": 1
},
{
"id": 7,
"label": "Chicken disease",
"url": "https://en.wikipedia.org/wiki/Chicken disease",
"level": 1
},
{
"id": 8,
"label": "Cruelty to animals",
"url": "https://en.wikipedia.org/wiki/Cruelty to animals",
"level": 1
},
{
"id": 9,
"label": "Bird flight",
"url": "https://en.wikipedia.org/wiki/Bird flight",
"level": 1
},
{
"id": 10,
"label": "Dinosaur",
"url": "https://en.wikipedia.org/wiki/Dinosaur",
"level": 1
},
```

```
"id": 11,
"label": "Domestic turkey",
"url": "https://en.wikipedia.org/wiki/Domestic_turkey",
"level": 1
},
{
"id": 12,
"label": "Egg incubation",
"url": "https://en.wikipedia.org/wiki/Egg_incubation",
"level": 1
},
{
"id": 13,
"label": "Poultry disease",
"url": "https://en.wikipedia.org/wiki/Poultry_disease",
"level": 1
},
{
"id": 14,
"label": "Cloaca",
"url": "https://en.wikipedia.org/wiki/Cloaca",
"level": 1
},
{
"id": 15,
"label": "Bird",
"url": "https://en.wikipedia.org/wiki/Bird",
"level": 1
},
{
"id": 16,
"label": "Chick (young bird)",
"url": "https://en.wikipedia.org/wiki/Chick_(young_bird)",
"level": 1
},
{
"id": 17,
"label": "Debeaking",
"url": "https://en.wikipedia.org/wiki/Debeaking",
"level": 1
},
{
"id": 18,
"label": "Phasianidae",
"url": "https://en.wikipedia.org/wiki/Phasianidae",
"level": 1
```

```
},
{
  "id": 19,
  "label": "Psittacosis",
  "url": "https://en.wikipedia.org/wiki/Psittacosis",
  "level": 1
},
{
  "id": 20,
  "label": "Common pheasant",
  "url": "https://en.wikipedia.org/wiki/Common_pheasant",
  "level": 1
},
{
  "id": 21,
  "label": "Egg as food",
  "url": "https://en.wikipedia.org/wiki/Egg_as_food",
  "level": 1
},
{
  "id": 22,
  "label": "Category:Bird common names",
  "url": "https://en.wikipedia.org/wiki/Category:Bird_common_names",
  "level": 1
},
{
  "id": 23,
  "label": "10th edition of Systema Naturae",
  "url": "https://en.wikipedia.org/wiki/10th_edition_of_Systema_Naturae",
  "level": 1
},
{
  "id": 24,
  "label": "Domestication",
  "url": "https://en.wikipedia.org/wiki/Domestication",
  "level": 1
}
],
"paths": [
  [0, 1, 2],
  [0, 3, 2],
  [0, 4, 2],
  [0, 5, 2],
  [0, 6, 2],
  [0, 7, 2],
  [0, 8, 2]
]
```

```

[0, 9, 2],
[0, 10, 2],
[0, 11, 2],
[0, 12, 2],
[0, 13, 2],
[0, 14, 2],
[0, 15, 2],
[0, 16, 2],
[0, 17, 2],
[0, 18, 2],
[0, 19, 2],
[0, 20, 2],
[0, 21, 2],
[0, 22, 2],
[0, 23, 2],
[0, 24, 2]
]
},
"time": 7,
"degreesOfSeparation": 2,
"ok": true,
"totalNodesVisited": 355,
"totalNodesCrawled": 355
}

```

contoh output:

```

api-1 | {16 Poultry disease https://en.wikipedia.org/wiki/Poultry_disease 1}
api-1 | {19 Bird https://en.wikipedia.org/wiki/Bird 1}
api-1 | {21 10th edition of Systema Naturae https://en.wikipedia.org/wiki/10th_edition_of_Systema_Naturae 1}
api-1 | {22 Galliformes https://en.wikipedia.org/wiki/Galliformes 1}
api-1 | {3 Poultry https://en.wikipedia.org/wiki/Poultry 1}
api-1 | {4 Chick_(young_bird) https://en.wikipedia.org/wiki/Chick_(young_bird) 1}
api-1 | {9 Psittacosis https://en.wikipedia.org/wiki/Psittacosis 1}
api-1 | {13 Egg_incubation https://en.wikipedia.org/wiki/Egg_incubation 1}
api-1 | {15 Cruelty_to_animals https://en.wikipedia.org/wiki/Cruelty_to_animals 1}
api-1 | {6 Junglefowl https://en.wikipedia.org/wiki/Junglefowl 1}
api-1 | {7 Category:Bird_common_names https://en.wikipedia.org/wiki/Category:Bird_common_names 1}
api-1 | {8 Common_pheasant https://en.wikipedia.org/wiki/Common_pheasant 1}
api-1 | {10 Domestic_turkey https://en.wikipedia.org/wiki/Domestic_turkey 1}
api-1 | {12 Bird_flight https://en.wikipedia.org/wiki/Bird_flight 1}
api-1 | {14 Template_talk:Poultry https://en.wikipedia.org/wiki/Template_talk:Poultry 1}
api-1 | {17 Phasianidae https://en.wikipedia.org/wiki/Phasianidae 1}
api-1 | {18 Egg_as_food https://en.wikipedia.org/wiki/Egg_as_food 1}
api-1 | {0 Chicken https://en.wikipedia.org/wiki/Chicken 0}
api-1 | {2 Debeaking https://en.wikipedia.org/wiki/Debeaking 1}
api-1 | {24 Dinosaur https://en.wikipedia.org/wiki/Dinosaur 1}
api-1 | {11 Cloaca https://en.wikipedia.org/wiki/Cloaca 1}
api-1 | {20 Parasitism https://en.wikipedia.org/wiki/Parasitism 1}
api-1 | {23 Domestication https://en.wikipedia.org/wiki/Domestication 1}
api-1 | {1 Duck https://en.wikipedia.org/wiki/Duck 2}
api-1 | {5 Chicken_disease https://en.wikipedia.org/wiki/Chicken_disease 1}

```

#### **Gambar 3.4.1** Response dari backend

6. Frontend menangkap response tersebut kemudian memunculkan data-data tersebut di web app, serta memvisualisasikannya.

#### 3.4.2. Kasus 2: Pencarian semua shortest path dari judul Chicken ke Duck menggunakan IDS

1. User memilih pilihan IDS dan pilihan All path
2. User memasukkan input field from dengan “Chicken” dan input field to dengan “Duck”, kemudian submit
3. Aplikasi frontend mengirimkan request berupa tipe data yang berisi

```
{  
    source = "Chicken"  
    target = "Duck"  
    using_bfs = true  
    all_paths = false  
}
```

4. Aplikasi menangkap request tersebut dan segera melakukan proses pencarian sesuai dengan payload yang dikirimkan, dalam hal ini pemanggilan fungsi traversal.MultiPathIDS(“Chicken”, “Duck”) dan menyimpan hasilnya.
5. Hasil tersebut kemudian diproses sehingga membentuk tipe data Response yang diinginkan frontend, jika proses berjalan dengan lancar, backend akan memberikan response sbb

```
{  
    "data": {  
        "nodes": [  
            {  
                "id": 0,  
                "label": "Chicken",  
                "url": "https://en.wikipedia.org/wiki/Chicken",  
                "level": 0  
            },  
            {  
                "id": 1,  
                "label": "Duck",  
                "url": "https://en.wikipedia.org/wiki/Duck",  
                "level": 2  
            },  
            {  
                "id": 2,  
                "label": "Shortest Path",  
                "url": "https://en.wikipedia.org/wiki/Shortest_path",  
                "level": 3  
            }  
        ]  
    }  
}
```

```
"label": "Template talk:Poultry",
"url": "https://en.wikipedia.org/wiki/Template talk:Poultry",
"level": 1
},
{
"id": 3,
"label": "Debeaking",
"url": "https://en.wikipedia.org/wiki/Debeaking",
"level": 1
},
{
"id": 4,
"label": "Psittacosis",
"url": "https://en.wikipedia.org/wiki/Psittacosis",
"level": 1
},
{
"id": 5,
"label": "Poultry disease",
"url": "https://en.wikipedia.org/wiki/Poultry disease",
"level": 1
},
{
"id": 6,
"label": "Phasianidae",
"url": "https://en.wikipedia.org/wiki/Phasianidae",
"level": 1
},
{
"id": 7,
"label": "Bird",
"url": "https://en.wikipedia.org/wiki/Bird",
"level": 1
},
{
"id": 8,
"label": "Egg incubation",
"url": "https://en.wikipedia.org/wiki/Egg incubation",
"level": 1
},
{
"id": 9,
"label": "Chicken disease",
"url": "https://en.wikipedia.org/wiki/Chicken disease",
"level": 1
},
```

```
{  
  "id": 10,  
  "label": "Cruelty to animals",  
  "url": "https://en.wikipedia.org/wiki/Cruelty_to_animals",  
  "level": 1  
},  
{  
  "id": 11,  
  "label": "Domestication",  
  "url": "https://en.wikipedia.org/wiki/Domestication",  
  "level": 1  
},  
{  
  "id": 12,  
  "label": "Dinosaur",  
  "url": "https://en.wikipedia.org/wiki/Dinosaur",  
  "level": 1  
},  
{  
  "id": 13,  
  "label": "Cloaca",  
  "url": "https://en.wikipedia.org/wiki/Cloaca",  
  "level": 1  
},  
{  
  "id": 14,  
  "label": "Galliformes",  
  "url": "https://en.wikipedia.org/wiki/Galliformes",  
  "level": 1  
},  
{  
  "id": 15,  
  "label": "Parasitism",  
  "url": "https://en.wikipedia.org/wiki/Parasitism",  
  "level": 1  
},  
{  
  "id": 16,  
  "label": "Bird flight",  
  "url": "https://en.wikipedia.org/wiki/Bird_flight",  
  "level": 1  
},  
{  
  "id": 17,  
  "label": "10th edition of Systema Naturae",  
  "url": "https://en.wikipedia.org/wiki/10th_edition_of_Systema_Naturae",
```

```
        "level": 1
    },
    {
        "id": 18,
        "label": "Common pheasant",
        "url": "https://en.wikipedia.org/wiki/Common_pheasant",
        "level": 1
    },
    {
        "id": 19,
        "label": "Category:Bird common names",
        "url": "https://en.wikipedia.org/wiki/Category:Bird_common_names",
        "level": 1
    },
    {
        "id": 20,
        "label": "Egg as food",
        "url": "https://en.wikipedia.org/wiki/Egg_as_food",
        "level": 1
    },
    {
        "id": 21,
        "label": "Chick (young bird)",
        "url": "https://en.wikipedia.org/wiki/Chick_(young_bird)",
        "level": 1
    },
    {
        "id": 22,
        "label": "Poultry",
        "url": "https://en.wikipedia.org/wiki/Poultry",
        "level": 1
    },
    {
        "id": 23,
        "label": "Domestic turkey",
        "url": "https://en.wikipedia.org/wiki/Domestic_turkey",
        "level": 1
    },
    {
        "id": 24,
        "label": "Junglefowl",
        "url": "https://en.wikipedia.org/wiki/Junglefowl",
        "level": 1
    }
],
"paths": [
```

```

[0, 2, 1],
[0, 3, 1],
[0, 4, 1],
[0, 5, 1],
[0, 6, 1],
[0, 7, 1],
[0, 8, 1],
[0, 9, 1],
[0, 10, 1],
[0, 11, 1],
[0, 12, 1],
[0, 13, 1],
[0, 14, 1],
[0, 15, 1],
[0, 16, 1],
[0, 17, 1],
[0, 18, 1],
[0, 19, 1],
[0, 20, 1],
[0, 21, 1],
[0, 22, 1],
[0, 23, 1],
[0, 24, 1]
]
},
"time": 15,
"degreesOfSeparation": 2,
"ok": true,
"totalNodesVisited": 355,
"totalNodesCrawled": 355
}

```

- Frontend menangkap response tersebut kemudian memunculkan data-data tersebut di web app, serta memvisualisasikannya.

## BAB 4

### Implementasi dan Pengujian

#### **4.1. Spesifikasi Teknis Program**

Program ini secara keseluruhan terdiri dari 2 directory, yaitu /frontend dan /backend. Directory frontend memuat seluruh komponen penyusun aplikasi yang berinteraksi dengan user, serta memuat fungsi-fungsi yang melakukan request kepada backend. Di sisi lain, pada directory backend, terdapat setup menggunakan bahasa goLang yang membuat aplikasi backend, dan

fungsi-fungsinya untuk membuat endpoint dan api, dan handlernya, dan algoritma utama program, yaitu bagian logic yang memungkinkan masalah ini dapat diselesaikan, yaitu fungsi crawl, utilitas semaphore, dan fungsi traversal.

Program go dibuat dalam *module* yang bernama wikirace. *Module* ini memiliki beberapa *package*:

- wikirace/main: Menjalankan server
- wikirace/api: Menerima API requests dan menjalankan algoritma dan mengirim hasilnya balik.
- wikirace/semaphore: Implementasi *semaphore* sendiri agar dapat membatasi jumlah *crawling* secara konkuren.
- wikirace/wikipedia/crawling: Melakukan crawling wiki article dan melakukan parsing hyperlink-hyperlink yang valid.
- wikirace/wikipedia/traversal: Mengandung implementasi algoritma BFS dan IDS.

Algoritma utama program yang mengatur logic terdiri dari beberapa file, diantaranya:

1. bfs.go

File ini berisi algoritma pencarian BFS, baik untuk single path maupun multipath, Traversal yang diimplementasikan juga menggunakan peran semaphore untuk membatasi batas atas threading. Fungsi pada akhirnya ini mengembalikan kedalaman tree, tree, dan level untuk tiap node.



```
func MultiPathBFS(source, destination string) (int, map[string][]string, map[string]int) {
    var distances = make(map[string]int)
    var solutions []string
    var data = make(map[string][]string)
    var request_sem = semaphore.NewSemaphore(maxWorkers)
    ch := make(chan crawling.CrawlResult, 120000)

    queue := Queue{}
    queue.Enqueue(source)
    distances[source] = 0
    go func() {
        defer func() { request_sem.Release() }()
        request_sem.Acquire()
        result := crawling.Crawl(source)
        ch <- result
    }()
    currentDepth, found, closest_distance := 0, false, 0

    for !found {
        fmt.Printf("Searching depth %d...\n", currentDepth)
        size := queue.Len()

        for range size {
            crawlResult, ok := <-ch
            if !ok {
                panic("The goroutine panicked")
            } else {
                data[crawlResult.Name] = crawlResult.Links
            }
        }

        for range size {
            curr := queue.Dequeue()
            curr_dist := distances[curr]

            for _, neighbour := range data[curr] {
                if _, ok := distances[neighbour]; ok {
                    continue
                }
                if neighbour == destination {
                    fmt.Printf("Found solution %s\n", curr)
                    found = true
                    closest_distance = curr_dist + 1
                    solutions = append(solutions, curr)
                    break
                }
                queue.Enqueue(neighbour)
                distances[neighbour] = curr_dist + 1
                go func(neighbour string) {
                    defer func() { request_sem.Release() }()
                    request_sem.Acquire()
                    result := crawling.Crawl(neighbour)
                    ch <- result
                }(neighbour)
            }
            currentDepth++
        }
        fmt.Printf("Total nodes: %d & %d\n", len(data), len(distances))
        distances[destination] = closest_distance
    }
    return closest_distance, data, distances
}
```

**Gambar 4.1.1** Algoritma BFS multipath

```

●●● bfs.go

79  func SingePathBFS(source, destination string) (int, map[string]string, string) {
80      var distances = make(map[string]int)
81      var tree = make(map[string]string)
82      var data = make(map[string][]string)
83      var request_sem = semaphore.NewSemaphore(maxWorkers)
84      ch := make(chan crawling.CrawlResult, 120000)
85
86      queue := Queue{}
87      queue.Enqueue(source)
88      distances[source] = 0
89      go func() {
90          defer func() { request_sem.Release() }()
91          request_sem.Acquire()
92          result := crawling.Crawl(source)
93          ch <- result
94      }()
95
96      currentDepth, found, closest_distance := 0, false, 0
97
98      for !found {
99          fmt.Printf("Searching depth %d...\n", currentDepth)
100         size := queue.Len()
101
102         for range size {
103             crawlResult, ok := <-ch
104             if !ok {
105                 panic("The goroutine panicked")
106             } else {
107                 data[crawlResult.Name] = crawlResult.Links
108
109             }
110         }
111
112         for range size {
113             curr := queue.Dequeue()
114             curr_dist := distances[curr]
115
116             for _, neighbour := range data[curr] {
117                 if _, ok := distances[neighbour]; ok {
118                     continue
119                 }
120                 if neighbour == destination {
121                     fmt.Printf("Found solution %s\n", curr)
122                     found = true
123                     closest_distance = curr_dist + 1
124                     return closest_distance, tree, curr
125                     // break
126                 }
127                 queue.Enqueue(neighbour)
128                 tree[neighbour] = curr //neighbour = child, curr = parent
129                 distances[neighbour] = curr_dist + 1
130
131                 go func(neighbour string) {
132                     defer func() { request_sem.Release() }()
133                     request_sem.Acquire()
134                     result := crawling.Crawl(neighbour)
135                     ch <- result
136                     }(neighbour)
137                 }
138                 currentDepth++
139             }
140             fmt.Printf("Total nodes: %d & %d\n", len(data), len(distances))
141             return closest_distance, tree, ""
142     }

```

Snipped

**Gambar 4.1.2 Algoritma BFS singlepath**

## 2. ids.go

File ini berisi algoritma pencarian IDS untuk single path maupun multipath. Fungsi ini menggunakan semaphore untuk membatasi worker yang digunakan untuk multithreading. Algoritma ini mengembalikan level dari tree pencarian dan solusi/path dari source ke destination. Algoritma ini masih berbasis pada DFS sehingga diperlukan implementasi DFS untuk membantu algoritma pencarian ini.

```
 10  func dfsCache(node string, currentDepth, requiredDepth int, distances map[string]int, data map[string][]string, request_sem *semaphore.Semaphore, wg *sync.WaitGroup, ch chan crawling.CrawlResult, newNodes *int) {
 11    // fmt.Println("Still called...")
 12    if currentDepth == requiredDepth {
 13        _, ok := distances[node]
 14        (*newNodes)++
 15        if !ok {
 16            panic("Distance should have been recorded before crawled")
 17        }
 18        go func() {
 19            defer func() { request_sem.Release() }()
 20            request_sem.Acquire()
 21            result := crawling.Crawl(node)
 22            ch <- result
 23        }()
 24    }
 25  }
 26
 27  nodeDistance := distances[node]
 28  for _, neighbour := range data[node] {
 29      neighbourDistance, ok := distances[neighbour]
 30      if !ok {
 31          panic("Distance of neighbour should have been cached!")
 32      }
 33      if nodeDistance+1 == neighbourDistance {
 34          dfsCache(neighbour, currentDepth+1, requiredDepth, distances, data, request_sem, wg, ch, newNodes)
 35      }
 36  }
 37 }
 38
 39 func multiPathDFSsearch(node, destination string, currentDepth, requiredDepth int, distances map[string]int, data map[string][]string, path *[]string, solutions *[][]string) {
 40    if currentDepth == requiredDepth {
 41        if node == destination {
 42            fmt.Println("Found solution")
 43            copiedSolution := make([]string, len(*path))
 44            copy(copiedSolution, *path)
 45            *solutions = append(*solutions, copiedSolution)
 46        }
 47        return
 48    }
 49
 50    nodeDistance := distances[node]
 51
 52    for _, neighbour := range data[node] {
 53        neighbourDistance, ok := distances[neighbour]
 54        if !ok {
 55            panic("Distance should have been cached!")
 56        }
 57        if nodeDistance+1 == neighbourDistance {
 58            *path = append(*path, neighbour)
 59            multiPathDFSsearch(neighbour, destination, currentDepth+1, requiredDepth, distances, data, path, solutions)
 60            *path = (*path)[:len(*path)-1]
 61        }
 62    }
 63 }
 64 }
```

Snipped

Gambar 4.1.3 Algoritma DFS multipath

ids.go

```

65  func MultiPathIDS(source, destination string) (int, [][]string) {
66      var data = make(map[string][]string)
67      var solutions [][]string
68      var distances = make(map[string]int)
69      var request_sem = semaphore.NewSemaphore(maxWorkers)
70      var wg sync.WaitGroup
71      ch := make(chan crawling.CrawlResult, 10000)
72
73      distances[source] = 0
74
75      currentDepth := 0
76      for {
77          fmt.Printf("Seaching depth %d...\n", currentDepth)
78          var path []string
79          path = append(path, source)
80          multiPathDFSsearch(source, destination, 0, currentDepth, distances, data, &path, &solutions)
81
82          if len(solutions) != 0 {
83              break
84          }
85
86          newNodes := 0
87          dfsCache(source, 0, currentDepth, distances, data, request_sem, &wg, ch, &newNodes)
88          fmt.Printf("Crawling %d links in depth %d\n", newNodes, currentDepth)
89          for range newNodes {
90              node := <-ch
91              data[node.Name] = node.Links
92              for _, neighbour := range node.Links {
93                  if _, ok := distances[neighbour]; !ok {
94                      distances[neighbour] = currentDepth + 1
95                  }
96              }
97          }
98          currentDepth++
99      }
100     return currentDepth, solutions
101 }
102 }
```

Snipped

**Gambar 4.1.4 Algoritma IDS multipath**

ids.go

```

104 func singlePathDFSsearch(node, destination string, currentDepth, requiredDepth int, distances map[string]int, data map[string][]string, path *[]string, isFound *bool) {
105     if currentDepth == requiredDepth {
106         if node == destination {
107             (*isFound) = true
108         }
109         return
110     }
111
112     nodeDistance := distances[node]
113
114     for _, neighbour := range data[node] {
115         neighbourDistance, ok := distances[neighbour]
116         if !ok {
117             panic("Distance should have been cached!")
118         }
119         if nodeDistance+1 == neighbourDistance {
120             *path = append(*path, neighbour)
121             singlePathDFSsearch(neighbour, destination, currentDepth+1, requiredDepth, distances, data, path, isFound)
122             if *isFound {
123                 return
124             }
125             *path = (*path)[:len(*path)-1]
126         }
127     }
128 }
```

Snipped

**Gambar 4.1.5 Algoritma DFS singlepath**

```
●○● ids.go

130 func SinglePathIDS(source, destination string) (int, []string) {
131     var data = make(map[string][]string)
132     var distances = make(map[string]int)
133     var request_sem = semaphore.NewSemaphore(maxWorkers)
134     var wg sync.WaitGroup
135     ch := make(chan crawling.CrawlResult, 10000)
136
137     distances[source] = 0
138
139     currentDepth := 0
140     for {
141         fmt.Printf("Seaching depth %d...\n", currentDepth)
142         var path []string
143         path = append(path, source)
144         isFound := false
145         singlePathDFSsearch(source, destination, 0, currentDepth, distances, data, &path, &isFound)
146
147         if isFound {
148             return currentDepth, path
149         }
150
151         newNodes := 0
152         dfsCache(source, 0, currentDepth, distances, data, request_sem, &wg, ch, &newNodes)
153         fmt.Printf("Craling %d links in depth %d\n", newNodes, currentDepth)
154         for range newNodes {
155             node := <-ch
156             data[node.Name] = node.Links
157             for _, neighbour := range node.Links {
158                 if _, ok := distances[neighbour]; !ok {
159                     distances[neighbour] = currentDepth + 1
160                 }
161             }
162         }
163         currentDepth++
164     }
165 }
166 }
```

Snipped

**Gambar 4.1.6** Algoritma IDS singlepath

### 3. crawl.go

File ini berisi algoritma yang digunakan untuk melakukan web crawling pada website Wikipedia.

```

● ● ● crawl.go

13 type CrawlResult struct {
14     Name string
15     Links []string
16 }
17
18 func Crawl(name string) CrawlResult {
19     name = strings.Replace(name, "%", "%25", -1)
20     wiki_url := fmt.Sprintf("https://en.wikipedia.org/wiki/%s", name)
21
22     // Make an HTTP GET request to the URL
23     response, err := http.Get(wiki_url)
24     if err != nil {
25         log.Fatal("Error fetching URL:", err)
26     }
27
28     for response.StatusCode == 429 {
29         // fmt.Printf("Too many requests. %s\n", name)
30         response, err = http.Get(wiki_url)
31         if err != nil {
32             log.Fatal("Error fetching URL:", err)
33         }
34     }
35     defer response.Body.Close()
36
37     // Load the HTML document
38     doc, err := goquery.NewDocumentFromReader(response.Body)
39     if err != nil {
40         log.Fatal("Error loading HTML:", err)
41     }
42
43     var result_set = make(map[string]bool)
44
45     // Find the <div> with id "bodyContent" and loop through its <a> tags
46     doc.Find("a").Each(func(i int, s *goquery.Selection) {
47         // Get the href attribute of the <a> tag
48         link, exists := s.Attr("href")
49         if exists && len(link) >= 6 && link[:6] == "/wiki/" {
50             result_set[link[6:]] = true
51         }
52     })
53
54     var result_array []string
55
56     for name := range result_set {
57         result_array = append(result_array, name)
58     }
59
60     return CrawlResult{ name, result_array }
61 }

```

Snipped

**Gambar 4.1.7 Algoritma Crawling**

## 4.2. Tata Cara Penggunaan Program

### 4.2.1 Setup repository

Pertama-tama clone repository [https://github.com/ganadipa/Tubes2\\_AzGan](https://github.com/ganadipa/Tubes2_AzGan) dengan melakukan

```
git clone https://github.com/ganadipa/Tubes2_AzGan
```

### 4.2.2 Container docker setup

Kemudian lakukan docker compose up untuk menyalakan container pada docker dengan melakukan

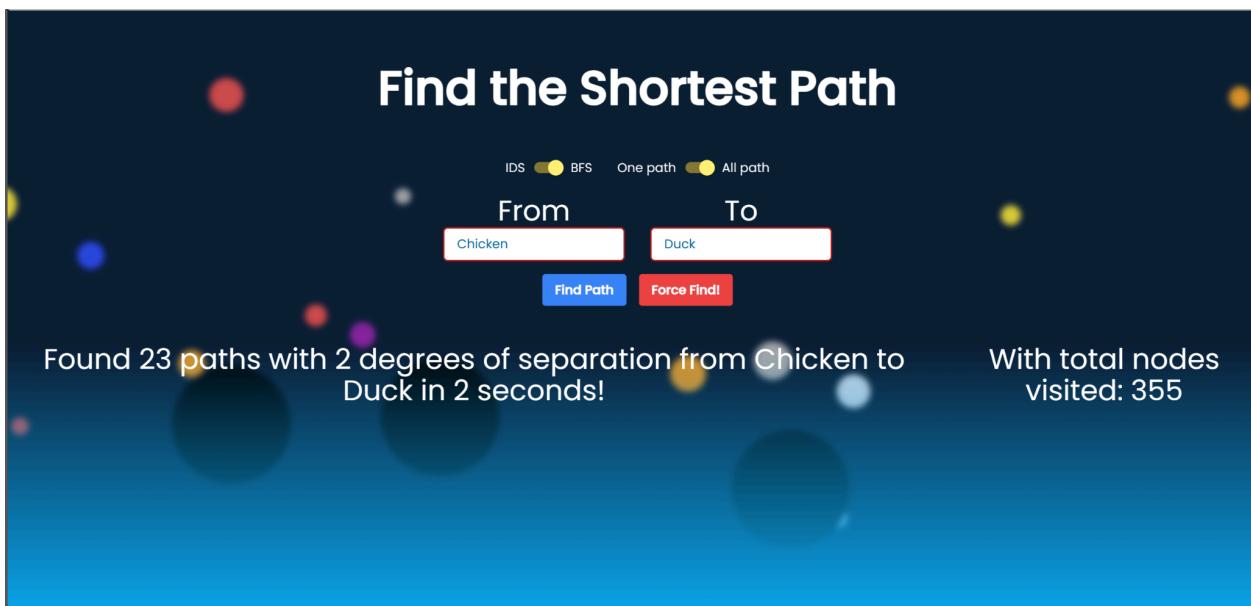
```
docker compose up
```

### 4.2.3 Penggunaan program

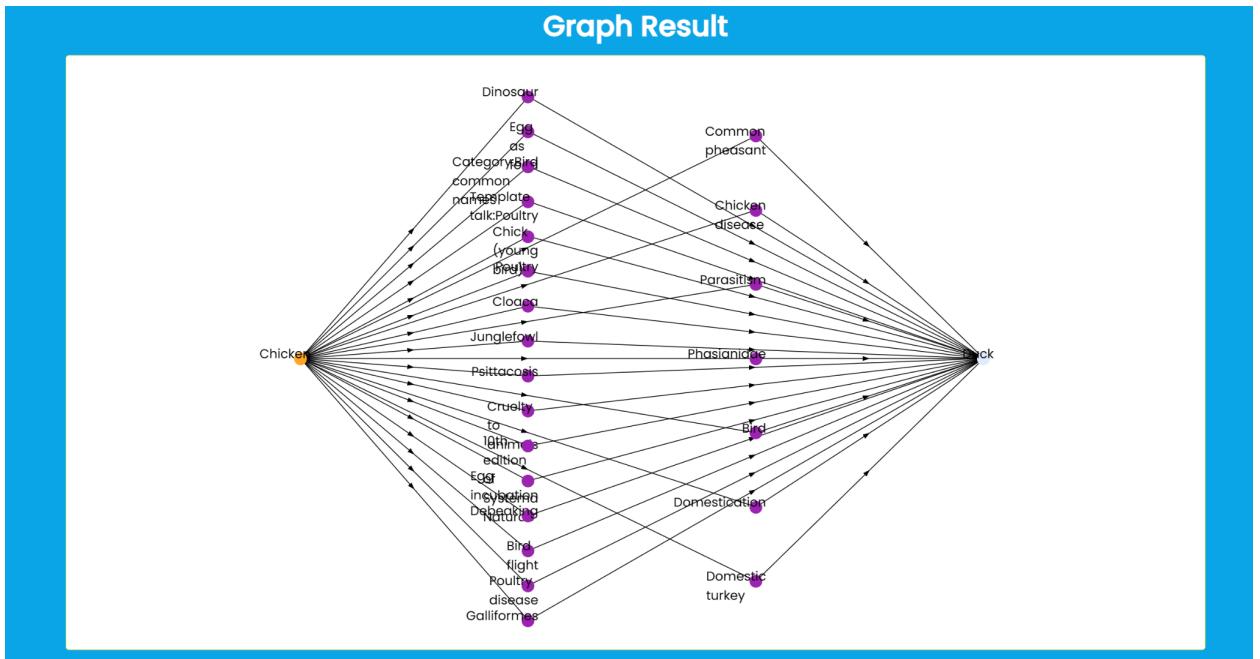
1. Pertama pilih metode yang ingin digunakan untuk skema traversal.
2. Kemudian pilih metode pencarian, antara mencari semua shortest path atau hanya satu shortest path.
3. Masukkan judul source dan target, kemudian gunakan suggestion dari input field tersebut.
4. Submit dan tunggu solusinya.

## 4.3. Hasil dan Analisis Pengujian

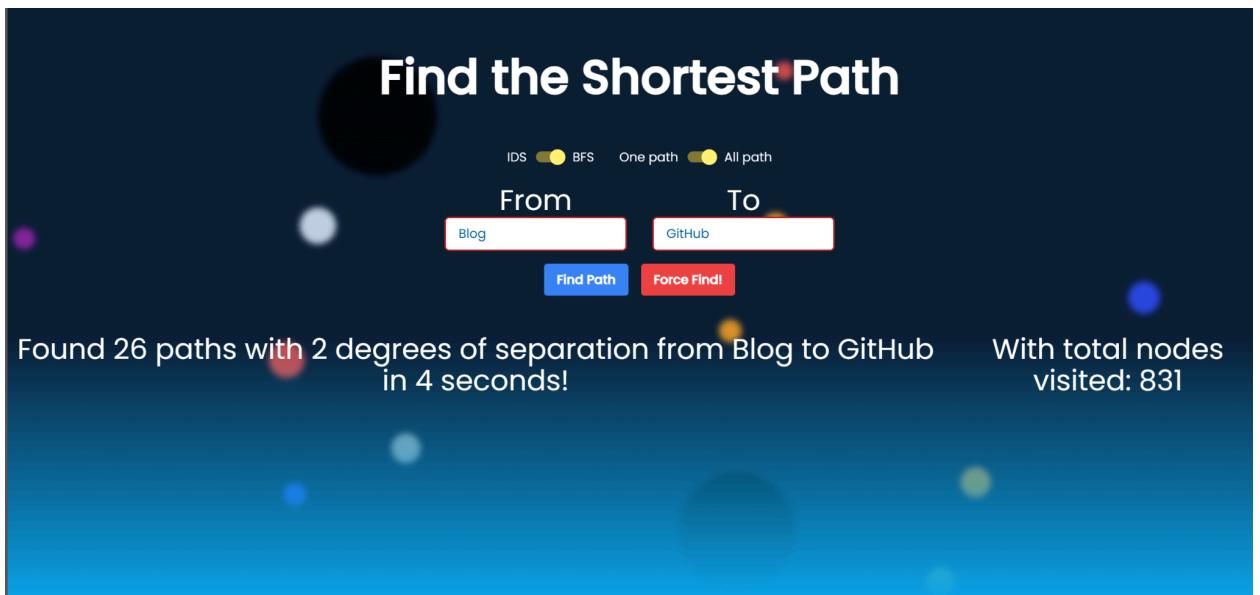
1. BFS



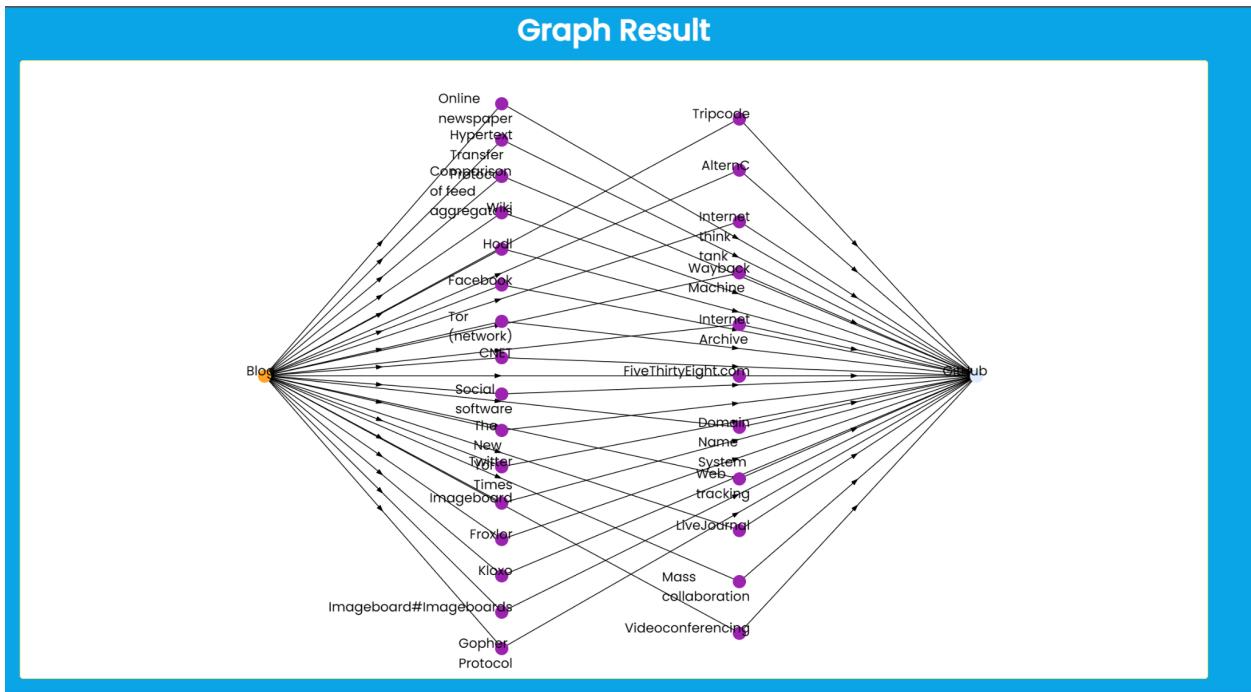
Gambar 4.3.1 Pencarian dari Chicken ke Duck dengan menggunakan algoritma BFS



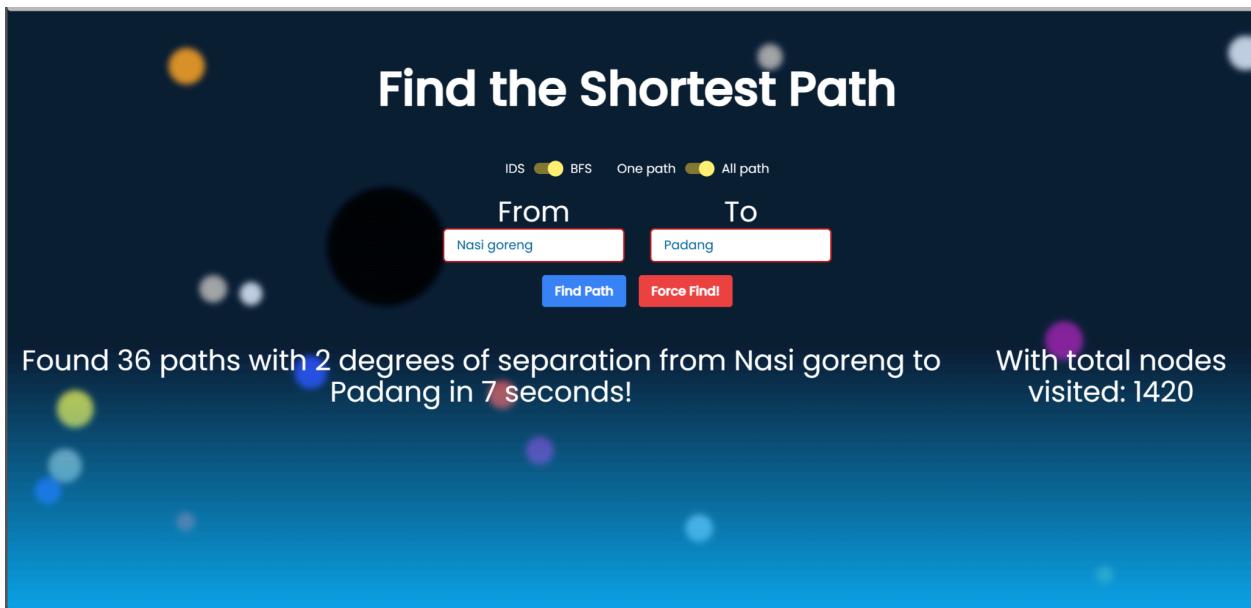
**Gambar 4.3.2** Pencarian dari Chicken ke Duck dengan menggunakan algoritma BFS



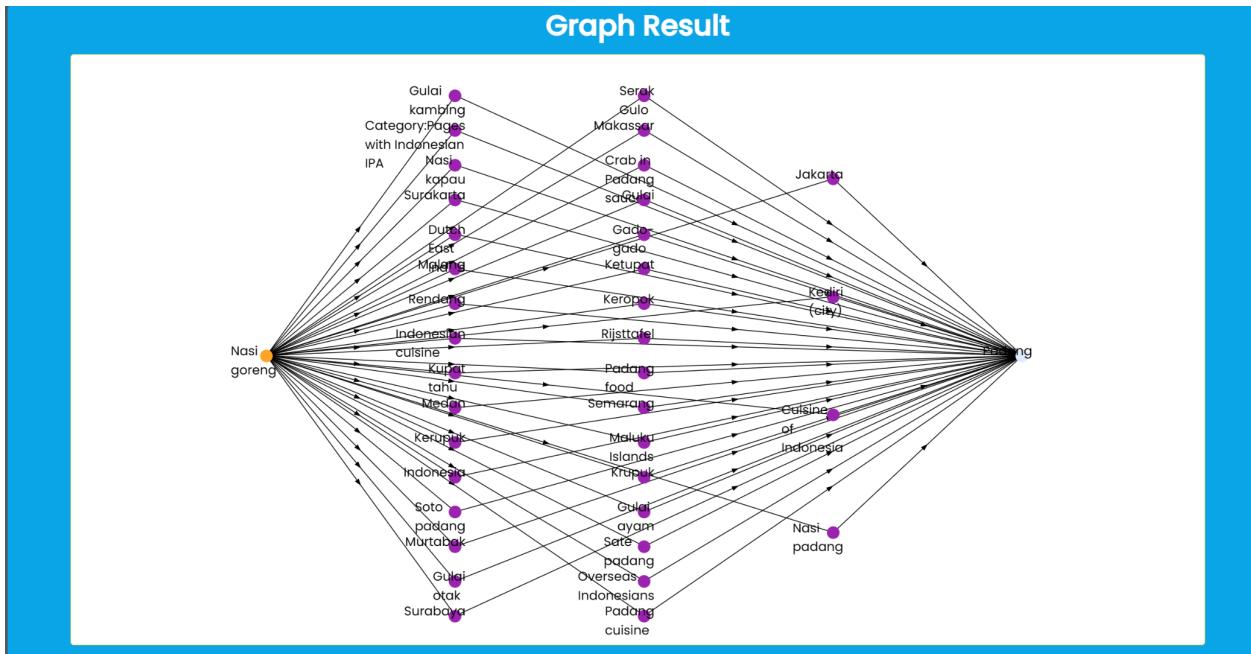
**Gambar 4.3.3** Pencarian dari Blog ke Github dengan menggunakan algoritma BFS



**Gambar 4.3.4** Graph perjalanan dari Blog ke Github dengan BFS

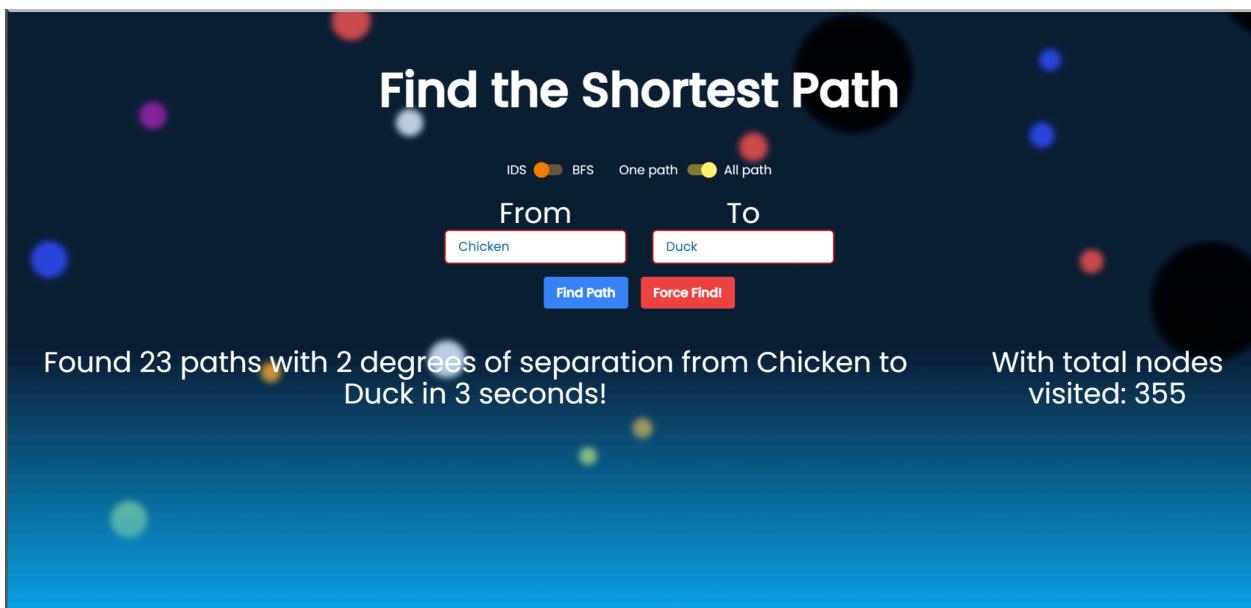


**Gambar 4.3.5** Pencarian dari Nasi goreng ke Padang dengan BFS

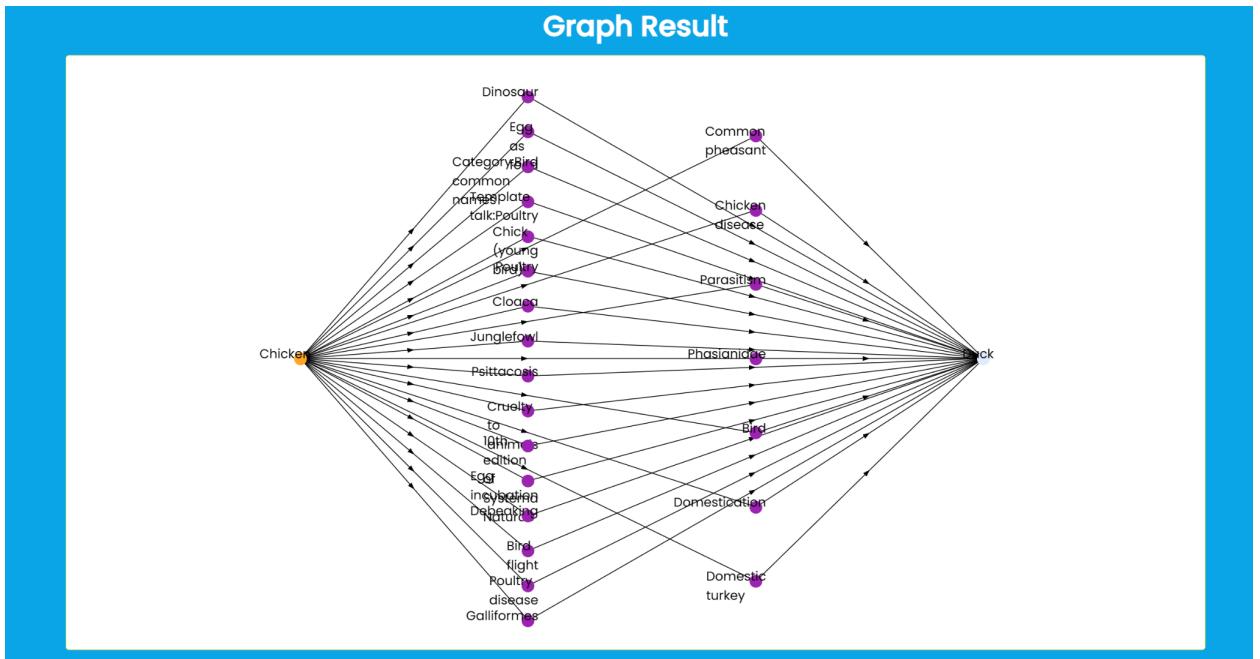


Gambar 4.3.6 Graph perjalanan dari Nasi goreng ke Padang dengan BFS

## 2. IDS



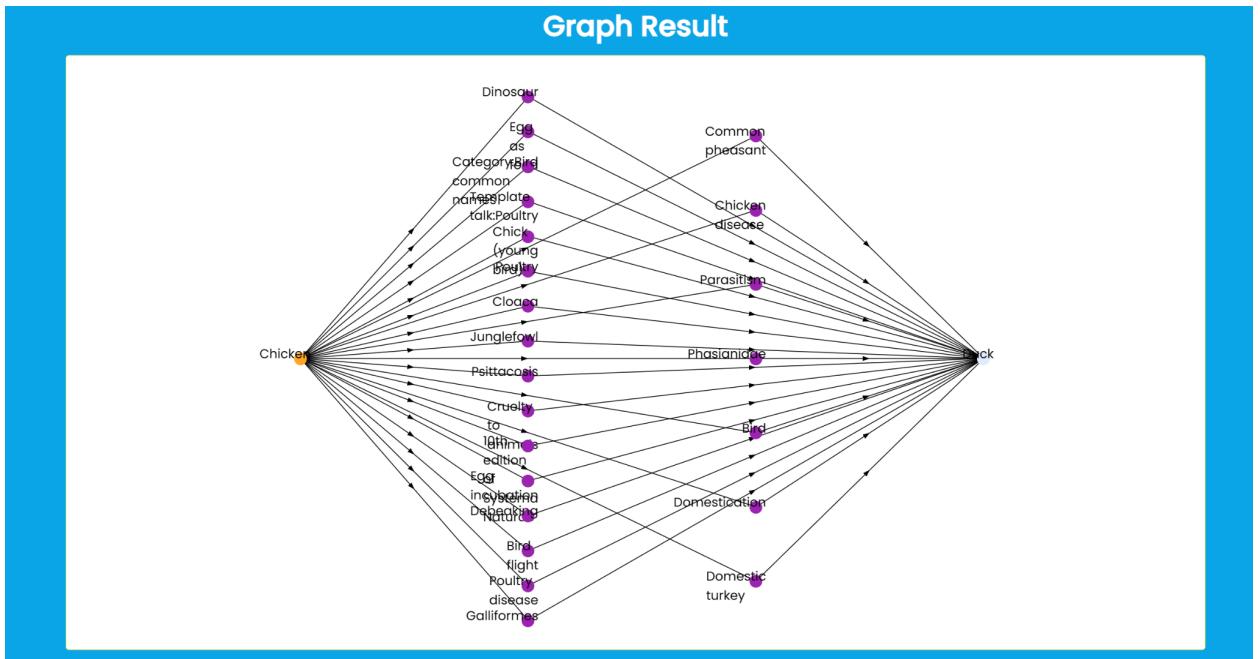
Gambar 4.3.7 Pencarian dari Chicken ke Duck dengan menggunakan algoritma IDS



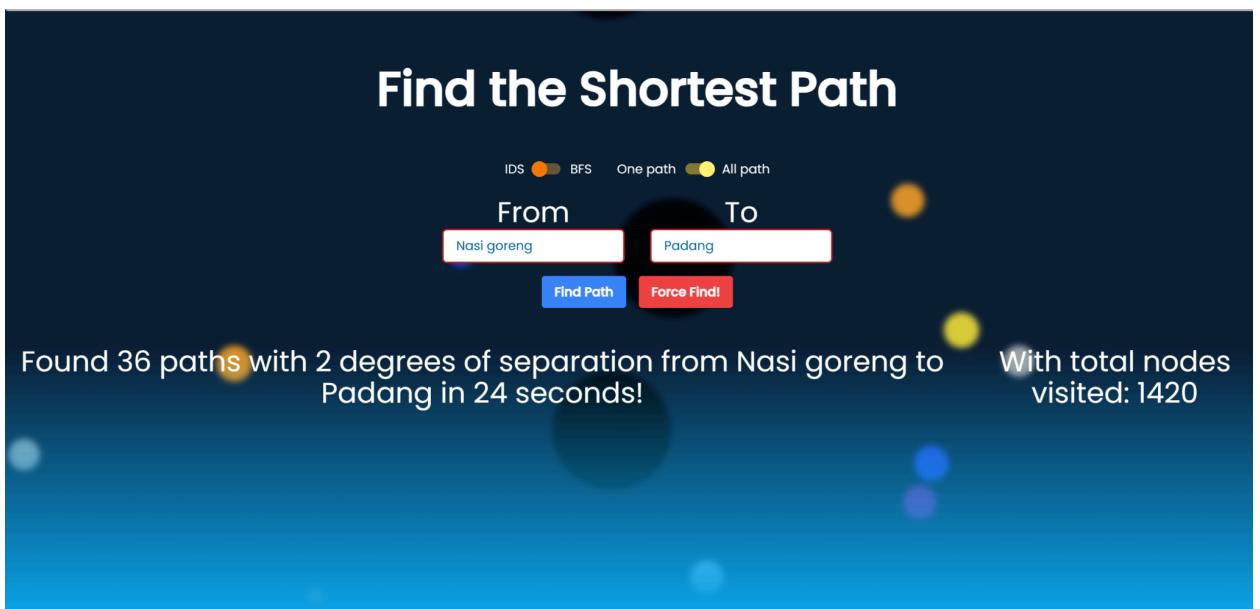
Gambar 4.3.8 Pencarian dari Chicken ke Duck dengan menggunakan algoritma IDS



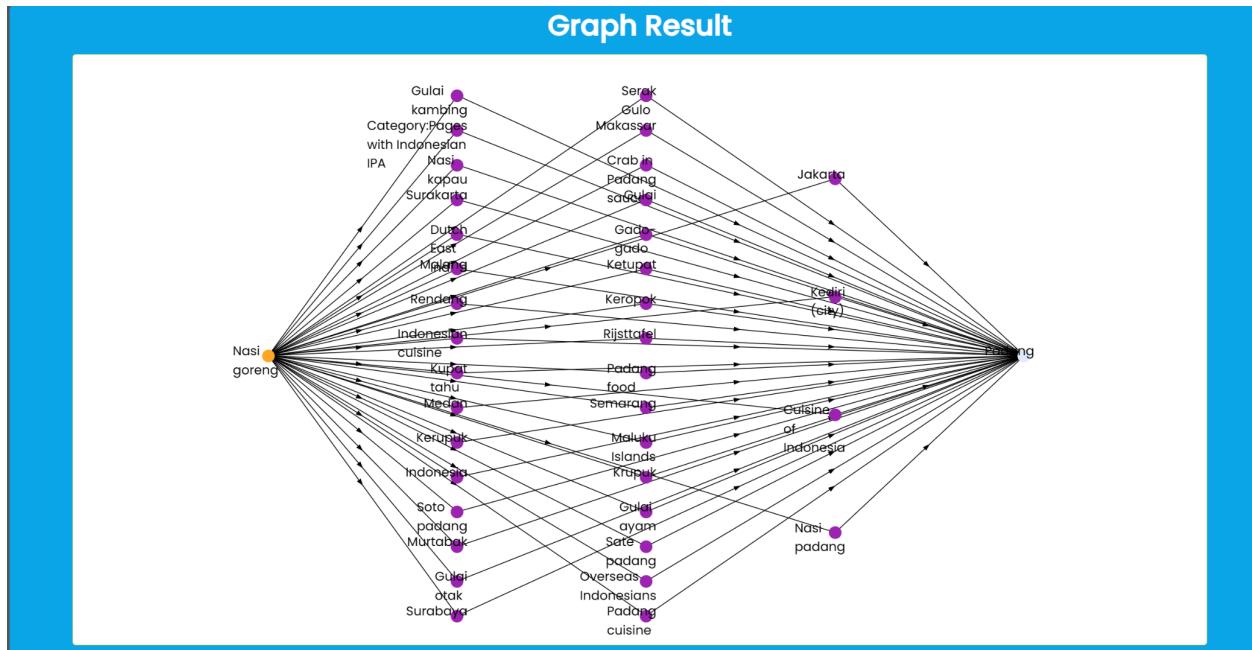
Gambar 4.3.9 Pencarian dari Blog ke Github dengan menggunakan algoritma IDS



**Gambar 4.3.10** Pencarian dari Blog ke Github dengan menggunakan algoritma IDS



**Gambar 4.3.11** Pencarian dari Nasi goreng ke Padang dengan IDS



**Gambar 4.3.12** Graph perjalanan dari Nasi goreng ke Padang dengan IDS

Dari percobaan diatas, dapat disimpulkan bahwa algoritma BFS jauh lebih cepat daripada IDS, Hal ini karena IDS sebenarnya adalah DFS yang dilakukan secara berulang dengan kedalaman yang meningkat tiap pencarinya. Hal ini bisa saja menyebabkan IDS mengunjungi node yang sama, sedangkan BFS melakukan pencarian node di setiap level/depth nya sehingga BFS hanya mengunjungi node sekali saja. Namun, IDS jauh lebih hemat memory karena IDS akan membuang nodes diluar path yang telah dikunjunginya, sedangkan BFS tetap menyimpan node tersebut.

Jika dinotasikan ke dalam notasi big-Oh, dari algoritma yang telah diimplementasikan, maka kami dapat kompleksitas waktu dari kedua algoritma ini adalah  $O(b^d)$  untuk IDS dan BFS, dan kompleksitas ruang untuk algoritma BFS adalah  $O(b^d)$  sementara  $O(bd)$  untuk IDS.

## BAB 5

# Kesimpulan dan Saran

### **5.1. Kesimpulan**

Dari hasil uji coba yang dilakukan dapat disimpulkan bahwa algoritma BFS lebih cepat dibandingkan algoritma IDS karena algoritma BFS dipastikan tidak mengunjungi suatu node lebih dari sekali, sedangkan algoritma IDS bisa mengunjungi suatu node lebih dari sekali seiring bertambahnya depth yang ditelusuri. Namun, algoritma IDS jauh lebih efisien dalam hal memori karena ia membuang semua node yang diluar path yang dikunjunginya, sedangkan BFS tetap menyimpan semua node tersebut.

## **5.2. Saran**

Saran untuk tugas besar 2 kali ini adalah sebelum mengerjakan tugas ini, dianjurkan memahami sintaks dari Go terlebih dahulu karena ini adalah bahasa baru dan belum pernah digunakan/diajarkan selama perkuliahan.

## **Lampiran**

Link *repository* github : [ganadipa/Tubes2\\_AzGan \(github.com\)](https://github.com/ganadipa/Tubes2_AzGan)

Link video youtube :  Tugas Besar 2 Wiki Race - Kelompok AzGan

## **Daftar Pustaka**

Munir, Rinaldi. "Breadth/Depth First Search (BFS/DFS) - Bagian 1 ." Institut Teknologi Bandung, 2020-2021.

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

Munir, Rinaldi. "Breadth/Depth First Search (BFS/DFS) - Bagian 2 ." Institut Teknologi Bandung, 2020-2021.

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