

LAPORAN TUGAS BESAR II

IF2211 STRATEGI ALGORITMA



Disusun oleh:

Kelompok 13 Chibye

Ignatius Jhon Hezkiel Chan	13522029
Raden Francisco Trianto Bratadiningrat	13522091
Suthasoma Mahardhika Munthe	13522098

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
BAB I DESKRIPSI TUGAS.....	5
1.1 Pendahuluan.....	5
1.2 Spesifikasi Program.....	5
BAB II LANDASAN TEORI.....	6
2.1 Graph Traversal.....	6
2.2 Algoritma Breadth First Search (BFS).....	7
2.3 Algoritma Depth First Search (DFS).....	8
2.4 Algoritma Depth Limited Search (DLS).....	9
2.5 Iterative Deepening Search (IDS).....	9
2.6 Kompleksitas Algoritma.....	10
2.7 Aplikasi Web.....	11
BAB III ANALISIS PEMECAHAN MASALAH.....	12
3.1 Langkah-langkah Pemecahan Masalah.....	12
3.2 Proses Pemetaan Masalah Menjadi Elemen Algoritma IDS dan BFS.....	12
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web.....	13
3.4 Contoh Ilustrasi Kasus.....	14
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	18
4.1 Spesifikasi Teknis Program.....	18
4.2 Penjelasan Tata Cara Penggunaan Program.....	22
4.3 Hasil Pengujian.....	27
4.4 Analisis Hasil Pengujian.....	29
BAB V KESIMPULAN DAN SARAN.....	31
5.1 Kesimpulan.....	31
5.2 Saran.....	31
5.3 Refleksi.....	31
LAMPIRAN.....	32
DAFTAR PUSTAKA.....	33

DAFTAR GAMBAR

- Gambar 1.1.1 Ilustrasi Graf WikiRace
- Gambar 2.1.1 Contoh Graph Traversal
- Gambar 2.2.1 Contoh Pencarian pada BFS
- Gambar 2.2.2 Tingkat simpul pada Graf
- Gambar 2.3.1 Contoh Pencarian pada DFS
- Gambar 2.3.2 Pencarian DFS sebagai pohon
- Gambar 2.5.1 Pencarian IDS
- Gambar 3.3.1 Diagram Arsitektur Aplikasi Web
- Gambar 3.4.1.1 Pencarian BFS dengan kedalaman 1
- Gambar 3.4.1.2 Pencarian BFS dengan kedalaman 2
- Gambar 3.4.1.3 Hasil Pencarian BFS
- Gambar 3.4.2.1 Pencarian IDS dengan Batas 2
- Gambar 3.4.2.2 Pencarian IDS dengan Batas 3
- Gambar 3.4.2.3 Hasil Pencarian IDS
- Gambar 4.2.1.1 Tampilan Aplikasi Web
- Gambar 4.2.2.1 Tampilan Hasil Pencarian “Animal” menuju “Gravity”
- Gambar 4.2.3.1 Interface Program
- Gambar 4.2.3.2 Tampilan Jika Masukkan Tidak Valid

DAFTAR TABEL

Tabel 4.1.2.1 Fungsi pada Program

Tabel 4.1.3.1 Prosedur pada Program

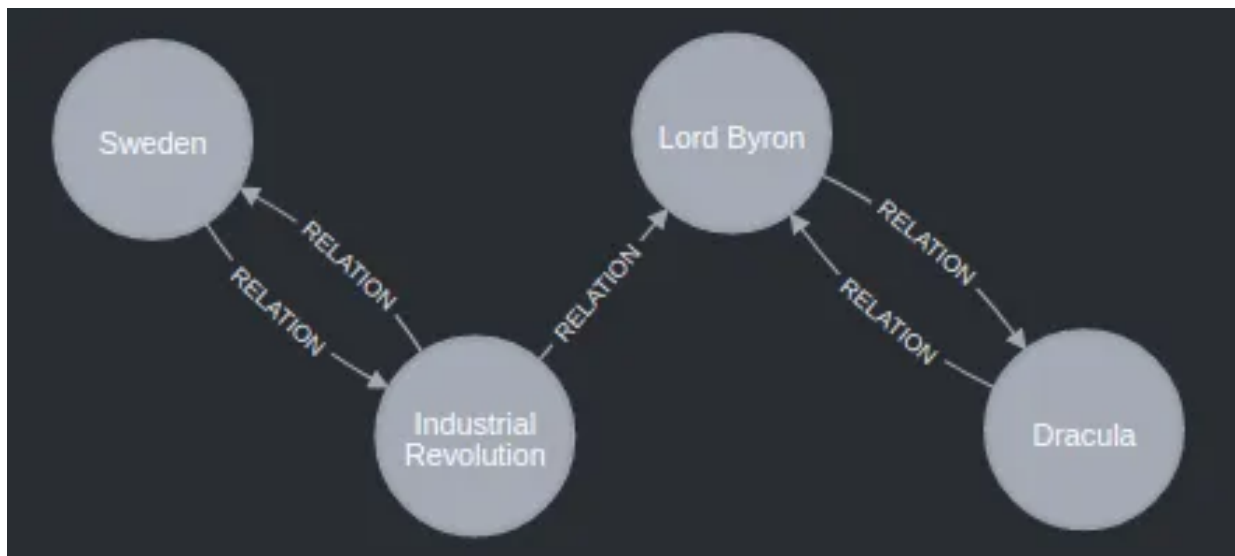
Tabel 4.3.1 Pengujian Program

BAB I

DESKRIPSI TUGAS

1.1 Pendahuluan

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 1.1.1 Ilustrasi Graf WikiRace

(Sumber: https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZsIicJCWQ.png)

1.2 Spesifikasi Program

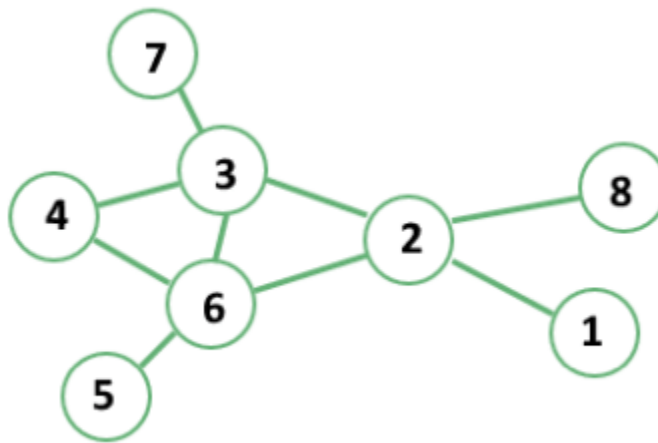
Buatlah program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace. Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms). Program berbasis web, sehingga perlu dibuat bagian frontend dan backend dari program.

BAB II

LANDASAN TEORI

2.1 Graph Traversal

Algoritma graph traversal adalah algoritma yang merupakan proses melewati suatu graph dengan mengunjungi simpul-simpul dengan cara yang sistematis. Tujuan dari algoritma ini adalah untuk mengakses atau memanipulasi setiap simpul dalam suatu graph dengan cara yang efisien dan sistematis. Terdapat dua jenis traversal graph, yaitu pencarian melebar (breadth first search/BFS) dan pencarian mendalam (depth first search/DFS).



Gambar 2.1.1 Contoh Graph Traversal

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

Graph Traversal adalah proses pencarian solusi persoalan yang direpresentasikan dengan suatu graf. Namun perlu diperhatikan bahwa graph harus terhubung agar dapat digunakannya graph traversal. Algoritma pencarian solusi berbasis graf dapat dibagi menjadi dua, yaitu uninformed dan informed Search.

2.1.1 Uninformed Search / Blind Search

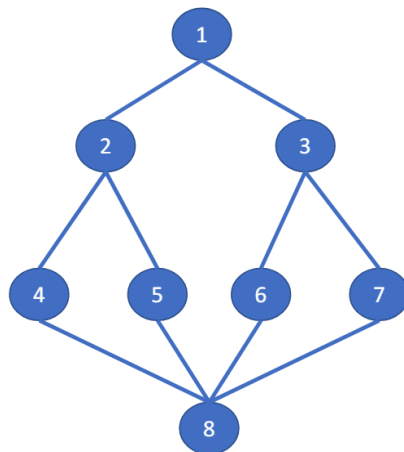
Uninformed search adalah algoritma pencarian yang tidak memiliki informasi tambahan sehingga kadang disebut blind search. Contoh dari Uninformed search antara lain: DFS, BFS, IDS, DLS, dan Cost Search.

2.1.2 Informed Search

Informed search adalah algoritma pencarian yang memiliki informasi tambahan, yaitu mengetahui non-goal state yang lebih menjanjikan daripada yang lain. Contoh dari Informed search antara lain: Best First Search dan A*

2.2 Algoritma Breadth First Search (BFS)

Algoritma Breadth-First Search atau BFS merupakan salah satu algoritma graph traversal utama yang mengunjungi semua simpul yang terhubung pada suatu graf secara bertingkat hingga solusi ditemukan atau seluruh graf sudah dilalui. Graf ini mengunjungi semua simpul pada satu tingkat sebelum berlanjut untuk mengunjungi simpul yang berada pada tingkat selanjutnya.



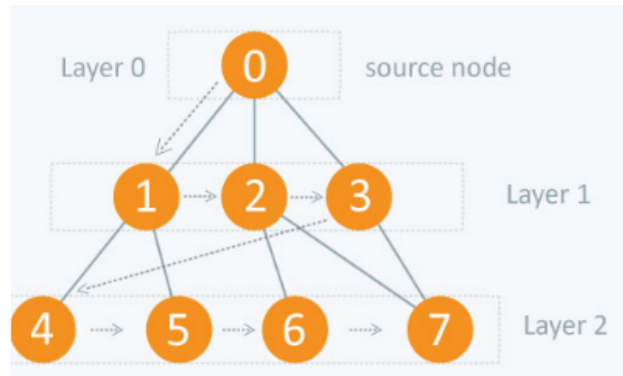
Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 2.2.1 Contoh Pencarian pada BFS

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

Pada gambar di atas, dapat dilihat contoh penggunaan algoritma BFS. Pencarian dengan algoritma BFS dapat diilustrasikan sebagai penggunaan struktur data queue (terlihat pada kolom Q pada gambar), dimana simpul pada urutan tingkat yang sama akan berurutan dalam queue dan simpul pada tingkat selanjutnya ditambahkan sehingga suatu tingkat bawah hanya diperiksa jika seluruh tingkat atas diperiksa terlebih dahulu.

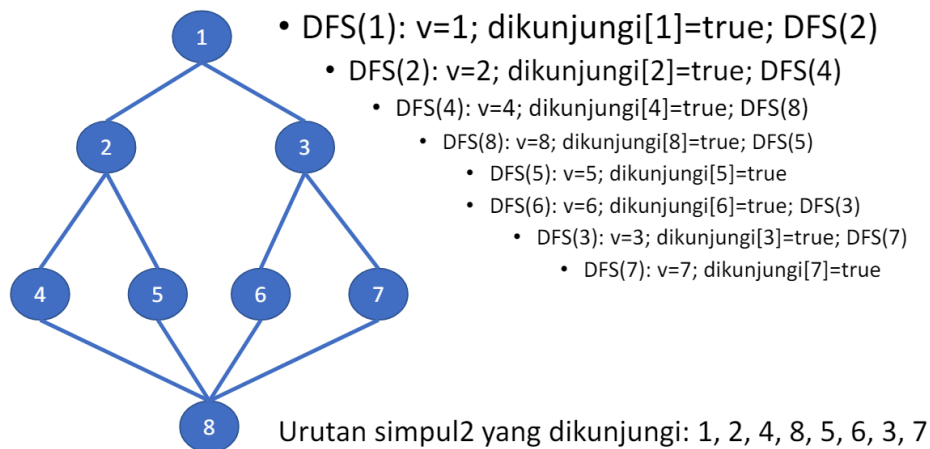


Gambar 2.2.2 Tingkat simpul pada Graf

(Sumber: <https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial>)

2.3 Algoritma Depth First Search (DFS)

Algoritma Depth First Search atau DFS merupakan salah satu algoritma graph traversal utama yang mengunjungi semua simpul yang terhubung pada suatu graf secara mendalam hingga solusi ditemukan atau seluruh graf sudah dilalui. Graf akan mengunjungi semua anak dari suatu simpul hingga semua anak dan anak dari anak telah dilalui, kemudian graf akan melanjutkan dan mengunjungi simpul yang setingkat dengan simpul awal.

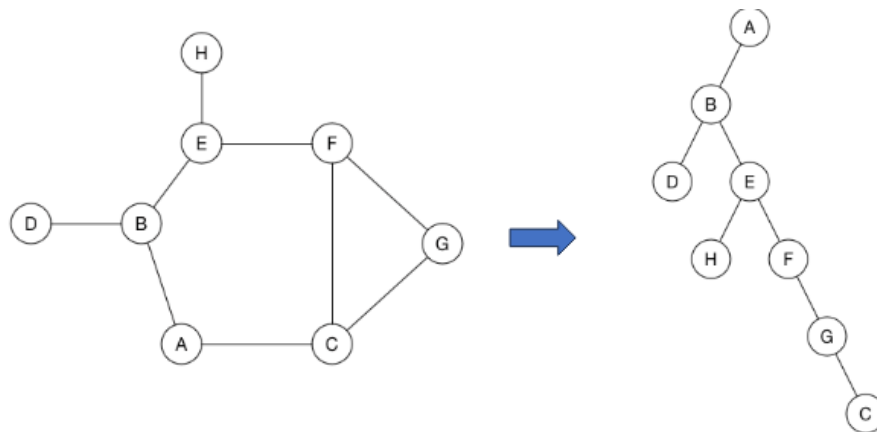


Gambar 2.3.1 Contoh Pencarian pada DFS

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

Pada gambar di atas, dapat dilihat contoh penggunaan algoritma DFS. Dapat dilihat pencarian mengunjungi anak dari suatu simpul hingga tidak ada anak yang perlu dikunjungi.

Setelah tidak ada anak yang dapat dikunjungi maka pencarian dilanjutkan pada simpul lainnya dari tingkat sebelumnya.



Gambar 2.3.2 Pencarian DFS sebagai pohon

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

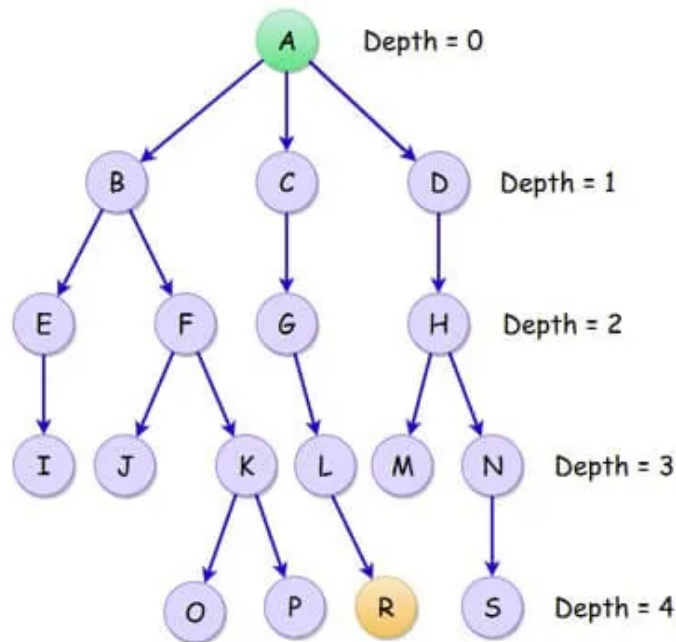
Berbeda dengan BFS, pencarian DFS yang bersifat mendalam dapat diilustrasikan menggunakan struktur data stack dimana anak dari simpul akan dikunjungi terlebih dahulu dan simpul pada tingkat paling atas akan dicek setelah semua anak telah dikunjungi.

2.4 Algoritma Depth Limited Search (DLS)

Algoritma Depth Limited Search atau DLS mirip dengan DFS melakukan pencarian secara mendalam. Namun terdapat perbedaan, yaitu DLS memiliki pembatasan kedalaman saat melakukan pencarian di mana pembatasan membatasi tingkat kedalaman simpul yang dikunjungi sebelum berhenti. DLS sendiri mengatasi permasalahan yang dialami BFS yaitu ruang status yang besar serta permasalahan pada DFS yaitu tidak dijamin solusi memiliki langkah minimum dan mungkin pencarian yang bersifat infinite.

2.5 Iterative Deepening Search (IDS)

Algoritma Iterative Deepening Search atau IDS merupakan algoritma yang terdiri dari serangkaian DLS dengan peningkatan nilai kedalaman maksimum hingga solusi ditemukan. IDS dapat dilihat sebagai gabungan BFS dan DFS.



Gambar 2.5.1 Pencarian IDS

(Sumber: <https://www.educba.com/iterative-deepening-depth-first-search/>)

Terlihat pada gambar di atas, adalah contoh graf beserta kedalaman simpul-simpul. IDS dimulai dengan kedalaman maksimum 0, yaitu melakukan pengecekan pada simpul A. Karena tidak ada solusi yang ditemukan, maka kedalaman maksimum akan dinaikan menjadi 1 dan dilakukan pencarian DLS dengan batas 1 sehingga mengunjungi simpul A, B, C dan D. Ketika solusi masih belum ditemukan maka kedalaman maksimum akan terus ditingkatkan hingga seluruh graf sudah dilalui atau solusi sudah ditemukan.

2.6 Kompleksitas Algoritma

Kompleksitas waktu algoritma BFS dalam notasi Big O adalah $O(b^d)$ di mana, b adalah jumlah maksimum pencabangan yang mungkin dari suatu simpul dan d adalah kedalaman dari solusi terbaik yang dicapai. Kelemahan BFS ada pada kompleksitas ruang yang dimilikinya yaitu $O(b^d)$ juga. Pertumbuhan ruang yang eksponensial sangat tidak dianjurkan pada sumber daya yang kecil.

Kompleksitas waktu yang dimiliki algoritma IDS juga sama yaitu $O(b^d)$. Namun, kompleksitas ruang lebih baik daripada BFS, yaitu $O(bd)$ sehingga algoritma ini tidak memakan banyak memori untuk mencapai solusi yang diinginkan.

2.7 Aplikasi Web

Aplikasi berbasis web, atau sering disebut aplikasi web, adalah program perangkat lunak yang disimpan pada suatu server yang kemudian memberi layanan melalui prambanan internet melalui bentuk website dan API (Application Program Interface). Karena menggunakan internet maka aplikasi dapat mudah digunakan oleh pengguna tanpa diperlukannya untuk mengunduh dan melakukan instalasi aplikasi.

Aplikasi Web terdiri dari dua bagian, yaitu frontend dan backend. frontend adalah bagian yang mengatasi tampilan yang dengan mudah dapat digunakan oleh pengguna namun tidak melakukan pemrosesan data yang besar. backend adalah bagian yang mengatasi data-data dan operasi yang perlu dilakukan suatu aplikasi web. Agar frontend dan backend dapat berkomunikasi, diperlukan adanya API yang menerima permintaan beserta data dari frontend. Data ini kemudian diproses dan API akan mengembalikan hasil operasi dari backend kepada frontend, dimana hasil akan ditampilkan kepada pengguna.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Permasalahan WikiRace adalah mencari rangkaian tautan yang berhubungan dari suatu halaman wikipedia menuju halaman target dengan jumlah tautan sekecil mungkin. Untuk menjalankan program, bagian frontend dan backend perlu dinyalakan secara terpisah. Jika kedua bagian aplikasi web sudah berjalan, maka program dapat digunakan.

Pengguna memasukkan judul halaman Wikipedia asal dan judul halaman Wikipedia yang dicari atau target dari pencarian. Pengguna kemudian memilih algoritma yang akan digunakan untuk menyelesaikan permasalahan tersebut. Terdapat dua pilihan, yaitu algoritma Breadth First Search atau BFS dan algoritma Iterative Deepening Search atau IDS. Data tersebut akan diberikan dari frontend kepada backend yang akan melakukan algoritma pencarian dan scrapping dari Wikipedia.

Halaman Wikipedia asal akan menjadi simpul awal jika melihat permasalahan sebagai sebuah graf. Tautan-tautan menuju halaman lain akan menjadi simpul tingkat bawahnya dan menghasilkan graf yang sangat besar dengan semakin banyak simpul yang diperlukan untuk dikunjungi. Jika pencarian berhasil menemukan halaman target, maka pencarian berhasil dan rangkaian tautan dikembalikan untuk ditampilkan oleh frontend kepada pengguna dalam bentuk graf dengan judul halaman sebagai simpul.

3.2 Proses Pemetaan Masalah Menjadi Elemen Algoritma IDS dan BFS

Mapping elemen-elemen dari permasalahan WikiRace berdasarkan algoritma graph traversal adalah sebagai berikut:

- a. Simpul: semua halaman Wikipedia
- b. Sisi: suatu halaman memiliki isi berupa tautan ke halaman lain maka membentuk sisi

Dengan elemen-elemen tersebut terlihat bahwa terbentuknya suatu graf sehingga dapat digunakan graf traversal. Jumlah tautan total pada Wikipedia English adalah 6.700.000 lebih

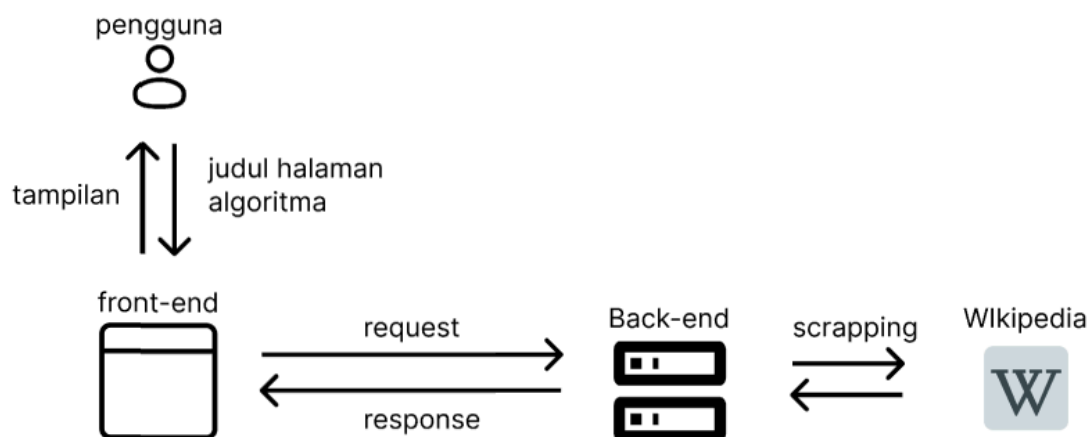
halaman dengan rata-rata jumlah tautan per halaman sekitar 200 hingga 1000 tautan. Hal tersebut menyebabkan graf yang terbuat sangat besar dan kompleks sehingga pemilihan algoritma sangat mempengaruhi kecepatan pencarian hasil.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Web

Fitur Fungsional dari Aplikasi Web adalah sebagai berikut:

- Melakukan Pencarian dengan Algoritma BFS
- Melakukan Pencarian dengan Algoritma IDS
- Terdapat Search bar dengan fitur suggestion dengan menggunakan API Wikipedia
- Hasil pencarian ditampilkan dengan menggunakan graf
- Pengguna dapat memilih algoritma yang digunakan untuk pencarian

Arsitektur Aplikasi Web kami, terbagi menjadi dua bagian, frontend dan backend. Untuk memperjelas interaksi antara pengguna, frontend, backend dan Wikipedia dapat melihat diagram berikut:



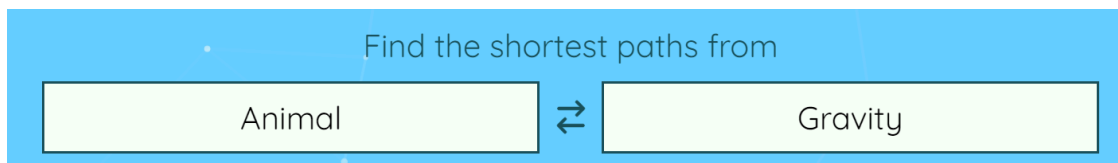
Gambar 3.3.1 Diagram Arsitektur Aplikasi Web

Pengguna dapat mengakses frontend dan memberikan data berupa judul halaman wikipedia asal dan tujuan. Pengguna juga memberikan tipe algoritma yang digunakan untuk menyelesaikan WikiRace. Data tersebut kemudian digunakan untuk mengirim request ke backend yang diterima oleh API. API kemudian akan melakukan scraping dari halaman-halaman dari domain en.wikipedia.org untuk mendapatkan tautan-tautan menuju halaman lain. Kemudian

API melakukan pencarian menggunakan algoritma yang dipilih baik BFS maupun IDS terhadap halaman-halaman tersebut. Hasil dari pencarian kemudian akan dikembalikan melalui response API yang dapat diterima oleh frontend. frontend kemudian menampilkan hasil tersebut dalam bentuk graf yang dapat dilihat oleh pengguna.

3.4 Contoh Ilustrasi Kasus

Untuk memperjelas analisis pemecahan masalah, kami melakukan ilustrasi kasus dengan penyelesaiannya. Misalkan kasus dimana judul halaman asal adalah “Animal” dan judul halaman hasil adalah “Gravity”.

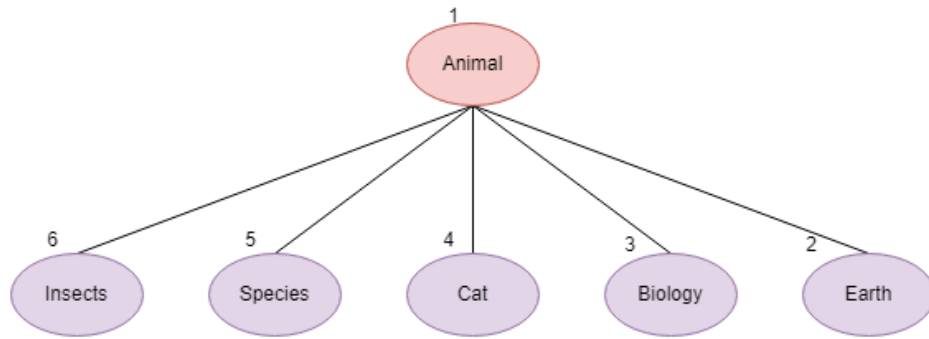


Gambar 3.4.1 Ilustrasi Kasus

(Sumber: <https://www.sixdegreesofwikipedia.com/?source=Animal&target=Gravity>)

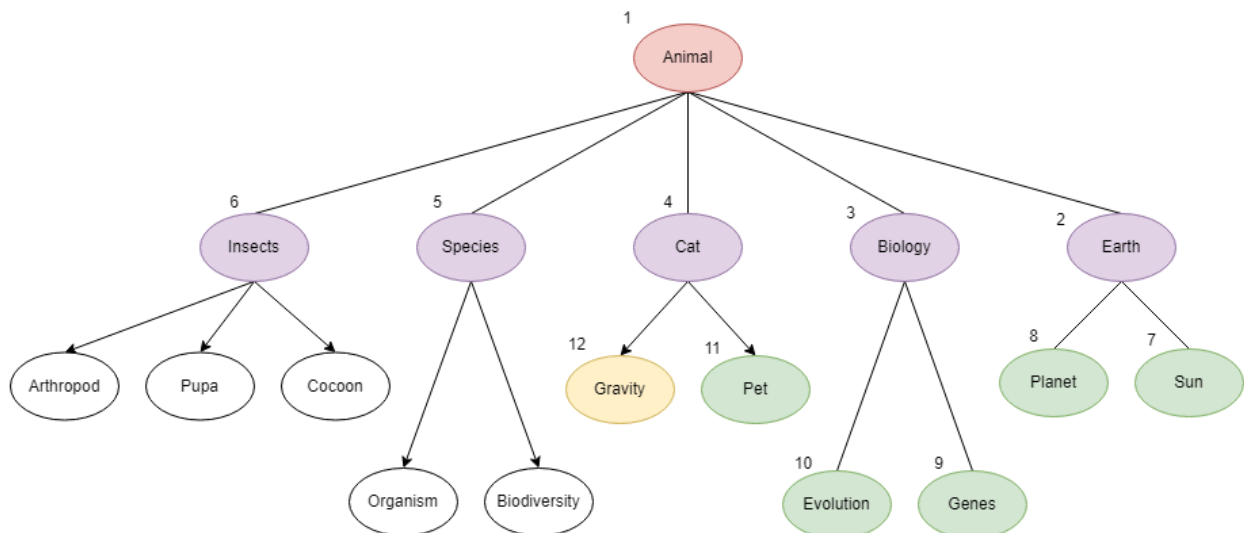
3.4.1 Ilustrasi Kasus Menggunakan Algoritma BFS

Proses BFS akan melakukan pencarian sesuai dengan kedalamannya saat dibentuk dalam pohon ruang status. Awalnya dalam pembentukan pohon ruang status akan ditambahkan terlebih dahulu root pohon yaitu tautan page awal. Namun, pada persoalan kali ini, pohon ruang status dibentuk secara ilusi. Pembentukan pohon ruang status akan dibuat menggunakan struktur data antrian. Awalnya di dalam antrian dimasukkan link awal yang telah disebutkan di atas. Kemudian, setiap elemen di dalam antrian akan diperiksa apakah sudah mencapai tujuan atau belum. Jika belum mencapai tujuan, semua link ke page lain yang termuat di dalam page tersebut akan ditambahkan ke dalam antrian. Pada ilustrasi di bawah page Animal dikunjungi dan sisi yang menunjuk ke simpul lain menunjukkan bahwa dari page Animal ada beberapa link menuju page lain.



Gambar 3.4.1.1 Pencarian BFS dengan kedalaman 1

Kemudian sesuai dengan urutan pada antrian elemen akan dikeluarkan satu per satu untuk diperiksa. Jika elemen tersebut bukan tujuan atau solusi maka link yang ada di dalamnya akan diperiksa jika sebelumnya belum pernah dikunjungi (hal ini menunjukkan bahwa link tersebut juga belum ditambahkan ke antrian). Dalam kasus ini, pada kedalaman 1 yaitu elemen yang dinomori dengan urutan 2 sampai 6 bukan solusi.



Gambar 3.4.1.2 Pencarian BFS dengan kedalaman 2

Kemudian kedalaman selanjutnya, mulai dari elemen 7 dan seterusnya diperiksa. Pada kasus ini, solusi akan dicapai saat mencapai elemen ke 12. Elemen lain yang belum diperiksa tidak akan dilanjutkan jika pada suatu elemen solusi sudah ditemukan. Namun, pada kedalaman yang sama dapat dilakukan jika harus mendapatkan semua solusi dengan kedalaman yang sama.

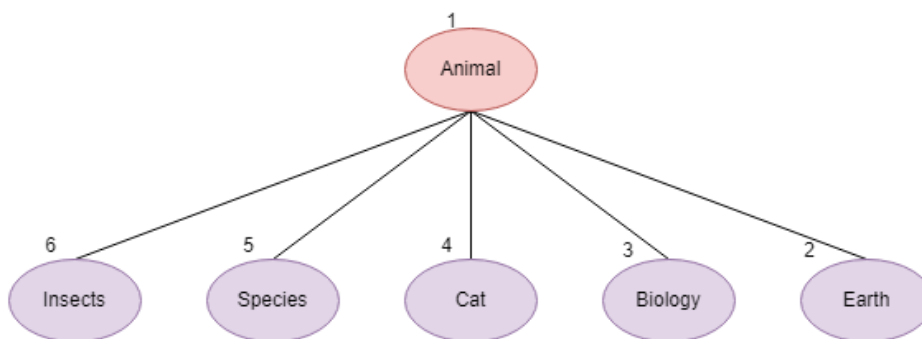


Gambar 3.4.1.3 Hasil pencarian BFS

Solusi yang didapatkan dapat dilihat pada Gambar 3.4.1.3 di atas. Namun, pada kasus nyata, jumlah tautan yang termuat di dalam satu page dapat berisi kurang lebih 200 - 1000 tautan. Proses multithread diperlukan karena proses akses dan parsing elemen html dapat memakan waktu sekitar 200 ms.

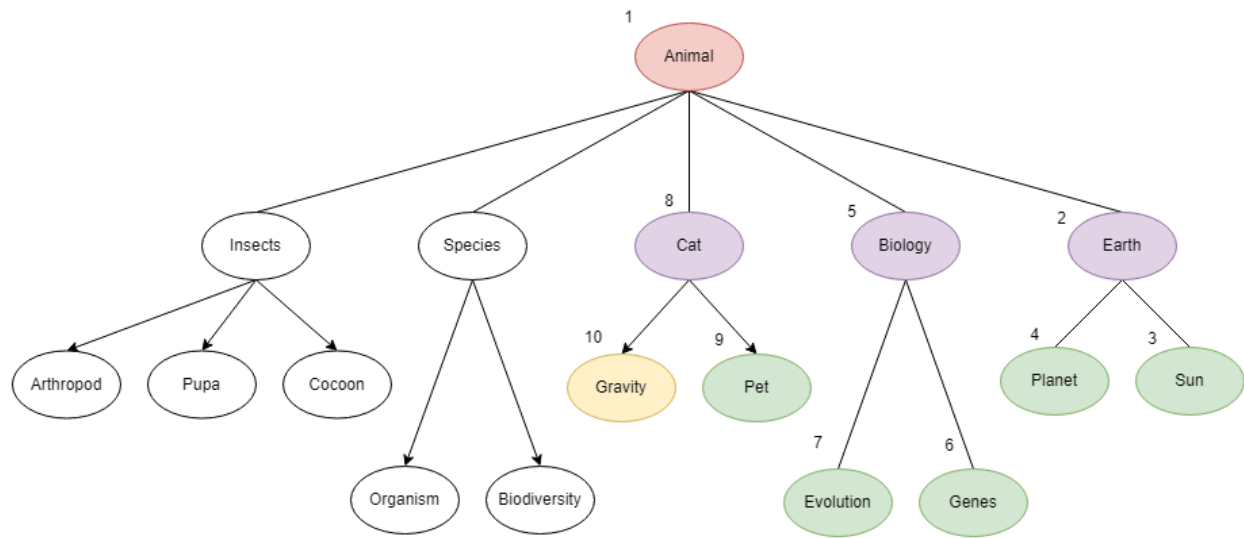
3.4.2 Ilustrasi Kasus Menggunakan Algoritma IDS

Dalam pemecahan persoalan tersebut menggunakan metode IDS, pencarian dimulai dengan batas pencarian yaitu 0 pada DLS. Ketika DLS memiliki batas 0, maka dia hanya melakukan pengecekan tautan awal dengan tautan hasil sehingga untuk kasus tersebut, pada DLS melakukan perbandingan “Animal” dengan “Gravity”. Karena gagal, IDS melanjutkan untuk memanggil DLS dengan batas yang baru, yaitu 1. DLS yang memiliki batas 1, akan mengunjungi simpul yang merupakan hasil dari kunjungan kepada tautan asal, yaitu “Animal”. DLS pada tingkat 1 dapat dilihat pada gambar berikut.



Gambar 3.4.2.1 Pencarian IDS dengan Batas 2

Pada gambar tersebut dimisalkan hanya terdapat 5 tautan pada halaman “Animal” menuju halaman lain. Dapat dilihat juga urutan pengunjungan DLS dari 1 hingga 6. Karena target masih belum ditemukan, maka IDS meningkatkan batas pencarian menjadi 2. Berikut adalah gambar urutan pencarian pada DLS dengan batas 2.



Gambar 3.4.2.2 Pencarian IDS dengan Batas 3

Dapat dilihat pada gambar, bahwa saat kunjungan ke-10, DLS berhasil menemukan halaman tujuan, sehingga IDS berhasil digunakan untuk menyelesaikan ilustrasi kasus tersebut. Dapat dilihat urutan kunjungan pada simpul-simpul yang bersifat DLS, menyebabkan tidak perlu dikunjungi simpul “Species” dan “Insects” beserta simpul anaknya.



Gambar 3.4.2.3 Hasil pencarian IDS

Ilustrasi ini membantu, dalam menjelaskan algoritma IDS dan BFS. Namun perlu diperhatikan bahwa ilustrasi ini tidak lengkap karena dalam satu halaman terdapat sekitar 200-1000 tautan menuju halaman lain. Dalam 200-1000 tautan tersebut, masing-masing juga memiliki 200-1000 tautan di dalamnya sehingga semakin jauh hubungan suatu tautan asal dengan targetnya maka jumlah halaman yang perlu dikunjungi akan meningkat secara eksponensial.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur data

Struktur data yang digunakan pada program ini dapat dilihat di bawah ini.

1. Struktur data `path.Path` (diambil dari *package path*) adalah struktur data yang memanfaatkan kemampuan `golang` dalam menyediakan pointer. Data ini digunakan dengan struktur *linked list*. Struktur data ini akan menyimpan data dari *parent (url parent)* sehingga tidak ada redudansi data. Setiap *vertice* akan memiliki Path-nya masing-masing.
2. Struktur data `Vertice` (diambil dari *package queue*) adalah struktur data yang akan menyimpan *url*-nya dan Path yang dimilikinya.
3. Struktur data `ListVertice` (diambil dari *package queue*) adalah implementasi lain *queue* untuk algoritma BFS yang digunakan. `ListVertice` juga menggunakan mutex yang disediakan `golang`, yaitu `RWMutex` yang mengizinkan beberapa proses secara bersamaan melakukan *read* dari variabel data ini. Sementara modifikasi atau *write* hanya akan dikhususkan pada satu proses.
4. `Solutions` (diambil dari *package solution*) menyimpan solusi yang ditemukan dari proses BFS. Struktur data ini memanfaatkan juga memanfaatkan `RWMutex`.
5. `Visited` (diambil dari *package visit*) berperilaku seperti set. Struktur data ini digunakan untuk mencegah redudansi pada proses BFS. Untuk menghindari *race condition*, digunakan `RWMutex` pada bagian ini juga.
6. `Slice` atau array pada `golang` yang dimanfaatkan untuk menyimpan sekumpulan data sejenis.

4.1.2 Fungsi

Tabel 4.1.2.1 Fungsi pada Program

Fungsi	Deskripsi
ExtractPage()	Fungsi ini melakukan scrape langsung pada page dan melakukan parsing pada body untuk mendapatkan sekumpulan link ke page lain dan mengembalikan list of link pada proses BFS. Selain itu, link yang belum tersimpan pada set visited akan ditambahkan ke dalam visited.
ExtractPageIDS()	Mirip dengan fungsi ExtractPage() di atas. Hanya saja dikhususkan pada proses IDS.
BFS()	Fungsi ini akan melakukan proses pencarian dengan algoritma BFS dan mengembalikan solusi dari hasil proses pencarian.
IDS()	Memulai proses untuk pencarian solusi dengan algoritma IDS dan mengembalikan hasil pencarian jika ditemukan.

4.1.3 Prosedur

Tabel 4.1.3.1 Prosedur pada Program

Prosedur	Deskripsi
ProcessPage()	Memproses setiap page dan melakukan pengecekan apakah link tujuan ada pada page tersebut atau tidak untuk algoritma BFS. Jika tidak ada maka semua link yang belum pernah dikunjungi akan ditambahkan ke dalam set Visited dan ditambahkan ke antrian.
DLS()	Melakukan proses DLS untuk yang dibuat secara rekursif.
ResetRequestCounter()	Melakukan reset pada untuk membatasi jumlah proses pada proses IDS.

4.1.4 Algoritma

Pseudocode untuk proses BFS

```
function BFS(input start, dest : url) -> solution
{ melakukan proses pencarian menggunakan algoritma BFS }

DEKLARASI
solusi : solution
visited : VisitedSet
antrian : Queue

ALGORITMA
antrian.enqueue(start) { menambahkan url awal ke dalam antrian }
while !solusi.found() and !antrian.isEmpty() do
    currNode = antrian.dequeue()
    currNode.setPath()
    links = getUnvisitedLink(currNode.getUrl(), visited)
    if dest ada pada links do
        solusi.setSolusi(dest) { solusi.found = true }
    else
        for semua url pada links do
            queue.enqueue(url)
        endif
    endwhile
return solusi
```

Pseudocode untuk proses IDS

```
procedure IDS(input start, dest : url, input maxDepth : int) -> array of string,
integer
{ melakukan proses pencarian menggunakan algoritma IDS }
{ mengembalikan hasil pencarian, boolean apakah ditemukan atau tidak dan jumlah
  halaman yang dikunjungi }

DEKLARASI
pageVisited : integer
path : array of string
```

```
limit : int
```

ALGORITMA

```
pageVisited = 0
```

```
limit = 0
```

```
While limit <= maxDepth do
```

```
    { Lakukan Pencarian DLS untuk setiap limit }
```

```
    DLS(start, dest, limit, array{start}, address pageVisited, address path)
```

```
    if ukuran path != 0 do
```

```
        return path, pageVisited
```

```
    else
```

```
        pageVisited = 0
```

```
        limit = limit + 1
```

```
endwhile
```

```
return null, pageVisited
```

4.1.5 Aplikasi Web

Sesuai dengan deskripsi tugas, untuk bagian backend kami dibatasi dengan bahasa pemrograman yaitu Golang atau Go. Untuk melakukan *web scraping* pada Wikipedia kami menggunakan library Golang, yaitu Gocolly (<https://go-colly.org/>) untuk membantu dalam meningkatkan kemudahan dan efisiensi dalam scraping. Karena dibutuhkan pembuatan API untuk menghubungkan frontend dan backend maka kami menggunakan web framework pada Golang, yaitu Gin (<https://gin-gonic.com/>) yang kami atur untuk menyediakan layanan API pada localhost port 8080 (<http://localhost:8080/>).

Untuk bagian frontend, kami memilih untuk menggunakan framework React (<https://react.dev/>), karena sifatnya yang ringan dan versatile untuk membuat sebuah frontend yang sederhana. Kami mengatur sehingga frontend melayani layanan web pada localhost port 3000 (<http://localhost:3000/>). Pada frontend kami menggunakan API Wikipedia untuk menyediakan sugesti judul halaman yang valid untuk domain <https://en.wikipedia.org>.

4.2 Penjelasan Tata Cara Penggunaan Program

4.2.1 Cara Menjalankan Program (Tanpa Docker)

Untuk menjalankan program, diperlukan dua terminal. Terminal ke-1 akan menjalankan bagian frontend. Terminal ke-2 akan menjalankan bagian backend. Aplikasi hanya dapat berjalan jika kedua bagian berjalan pada terminal yang berbeda. Langkah-langkah untuk menjalankan kedua bagian program adalah sebagai berikut:

- Menjalankan frontend untuk pertama kali
 1. Buka terminal ke-1
 2. Ketik “**git clone https://github.com/NoHaitch/Tubes2_FE_Chibye/**”
 3. Ketik “**npm install**”
 4. Ketik” **npm start**”
- Menjalankan backend untuk pertama kali
 1. Buka terminal ke-2
 2. Ketik “**git clone https://github.com/NoHaitch/Tubes2_BE_Chibye/**”
 3. Ketik “**cd src**”
 4. Ketik “**go get**”
 5. Ketik “**go run .**”
- Jika menjalankan program bukan pertama kali.

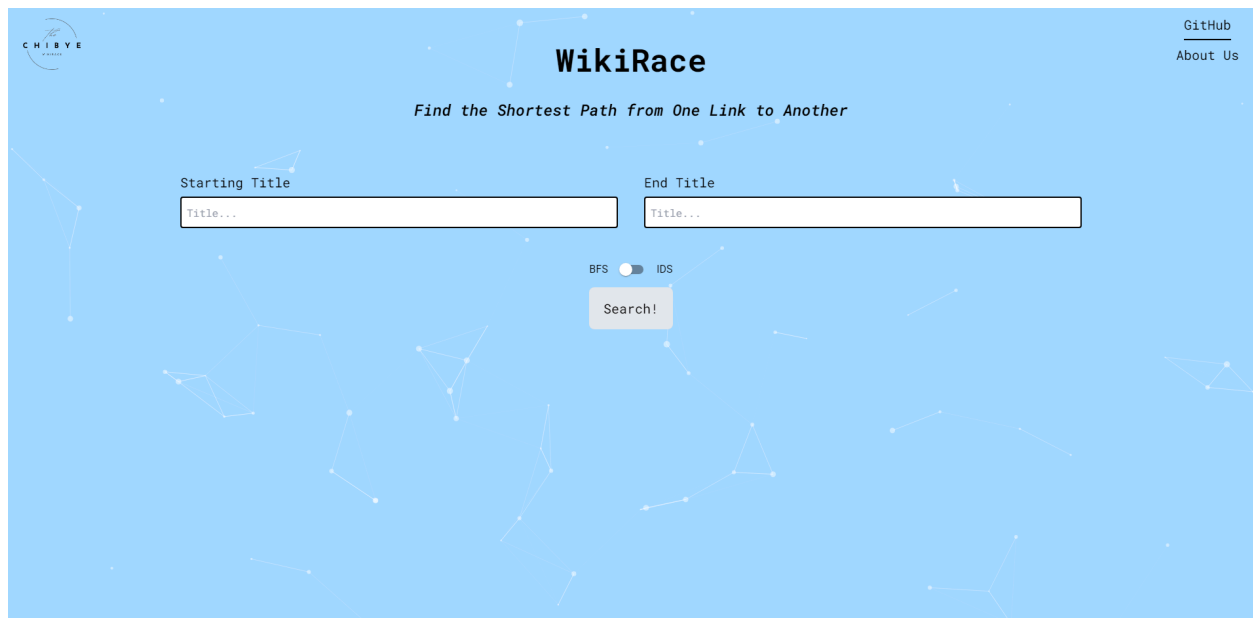
Jika langkah 1, 2, dan 3 pada tata cara menjalankan frontend sudah pernah dilakukan, maka untuk menjalankan frontend, hanya perlu mengikuti langkah 4. Jika langkah 1 hingga 4 sudah pernah dilakukan, maka untuk menjalankan backend hanya diperlukan mengikuti langkah 3 dan 5.

Setelah kedua bagian, frontend dan backend, sudah dijalankan maka aplikasi web siap digunakan. Untuk membuka aplikasi, buka tautan <http://localhost:3000/> pada browser. Tampilan aplikasi web yang sudah dijalankan dengan benar dapat dilihat pada gambar berikut. Tautan <http://localhost:8080/> juga dapat dikunjungi untuk melakukan pengecekan apakah API pada backend sudah berjalan dengan lancar.

4.2.2 Cara Menjalankan Program (Dengan Docker)

Dengan Docker, aplikasi Web dapat dipastikan berjalan untuk setiap jenis OS dan mesin komputasi. Docker menjalankan aplikasi pada sebuah “Container”, yakni sebuah *environment* terisolasi yang memiliki semua hal dan dependensi yang dibutuhkan kode/aplikasi untuk berjalan. Untuk menjalankan Program dengan menggunakan Docker, lakukan langkah-langkah berikut ini.

1. Buat sebuah folder baru di PC anda sebagai folder parent dari folder FrontEnd dan folder BackEnd. Kita sebut Tubes2_Chibye
2. Clone Repositori FrontEnd Website Chibye dan Repositori BackEnd Chibye dalam Tubes2_Chibye
3. Pindahkan file “docker-compose.yml” yang ada di dalam folder FrontEnd, keluar ke folder Tubes2_Chibye sehingga file .yml berada pada direktori yang sama dengan folder FrontEnd dan folder BackEnd.
4. Buka Terminal anda pada directory Tubes2_Chibye dan ketik “docker compose up --build”
5. Jika berikutnya anda ingin menjalankan docker lagi dan sebelumnya sudah pernah menjalankan langkah 4, maka cukup ketik “docker compose up”
6. Kini, aplikasi Web dengan fungsionalitas FrontEnd dan BackEnd sudah berjalan. Kunjungi <http://localhost:3000/> untuk membuka aplikasi Web WikiRace.

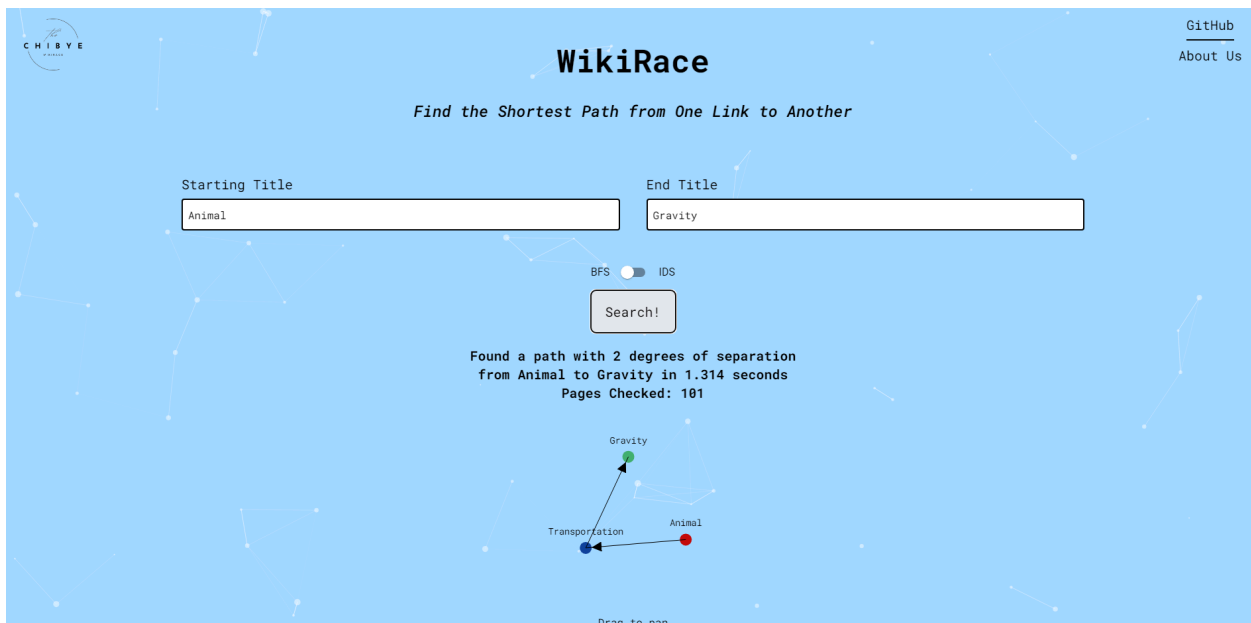


Gambar 4.2.1.1 Tampilan Aplikasi Web

4.2.2 Cara Penggunaan Program

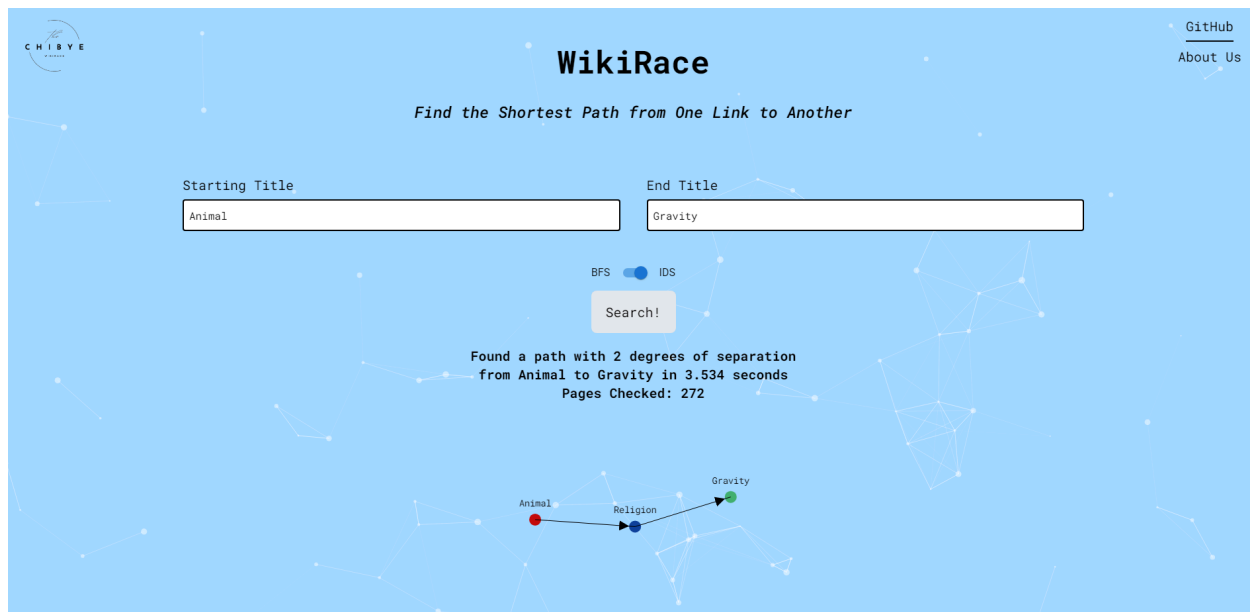
Jika program sudah berjalan dengan benar, maka berikut adalah langkah penggunaan aplikasi web untuk menyelesaikan persoalan WikiRace.

1. Mengisi Starting Title dengan judul halaman asal
2. Klik judul halaman yang sesuai pada sugesti yang diberikan
3. Mengisi End Title dengan judul halaman hasil
4. Klik judul halaman yang sesuai pada sugesti yang diberikan
5. Pilih algoritma yang digunakan untuk pencarian
6. Tekan tombol Search untuk memulai pencarian solusi
7. Tunggu hingga solusi ditampilkan dalam bentuk graf



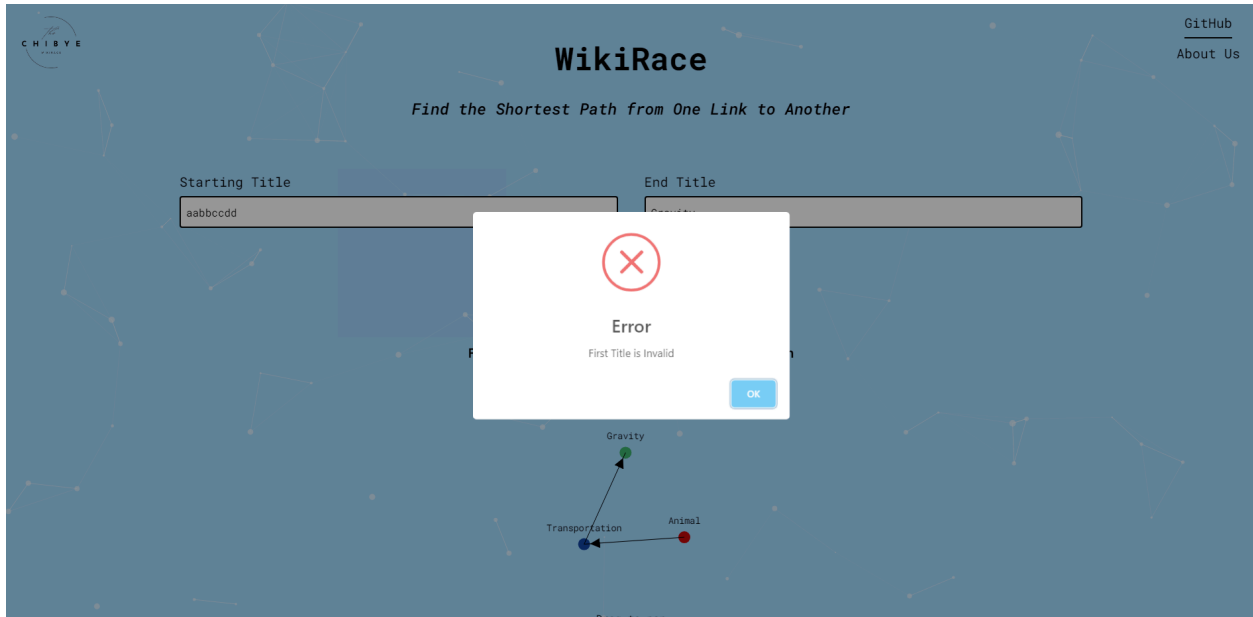
Gambar 4.2.2.1 Tampilan Hasil Pencarian “Animal” menuju “Gravity”

4.2.3 Interface Program



Gambar 4.2.3.1 Interface Program

Interface program terdiri dari 2 kotak teks untuk pencarian judul, 1 tombol pemilihan algoritma, dan 1 tombol search. Kotak teks digunakan untuk mengisi judul halaman asal dan tujuan, dimana Starting Title menunjuk pada judul halaman asal dan End Title menunjuk pada judul halaman tujuan atau target. Secara normal (*Default*) algoritma akan menggunakan BFS seperti pada gambar 4.2.2.1. Namun jika tombol pemilihan algoritma diklik maka algoritma akan berubah dan menggunakan algoritma IDS seperti pada gambar 4.2.3.1. Jika tautan halaman asal dan tujuan sudah diisi, maka dapat menggunakan tombol search untuk mencari solusi dari persoalan WikiRace. Namun jika tautan asal atau tautan tujuan belum diisi atau diisi dengan judul halaman yang tidak valid maka tidak dapat dilakukan pencarian. Jika pencarian dapat dilakukan maka hasil dari penyelesaian persoalan akan ditampilkan dalam bentuk graf. Simpul pada graf dapat diklik untuk menuju halaman Wikipedia yang berjudul tersebut. Simpul graf juga dapat ditarik untuk merubah posisi simpul-simpul pada graf.



Gambar 4.2.3.2 Tampilan Jika Masukkan Tidak Valid

4.2.4 Fitur-Fitur Program


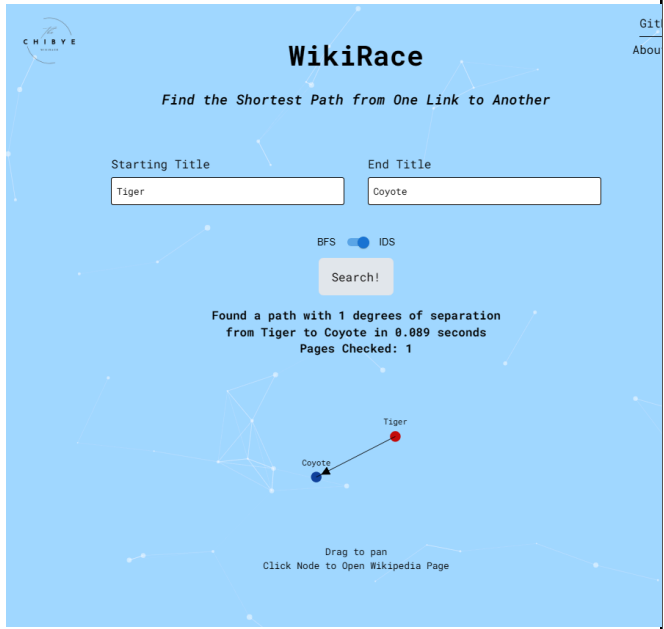
Fitur-fitur yang disediakan oleh aplikasi web kami adalah sebagai berikut:

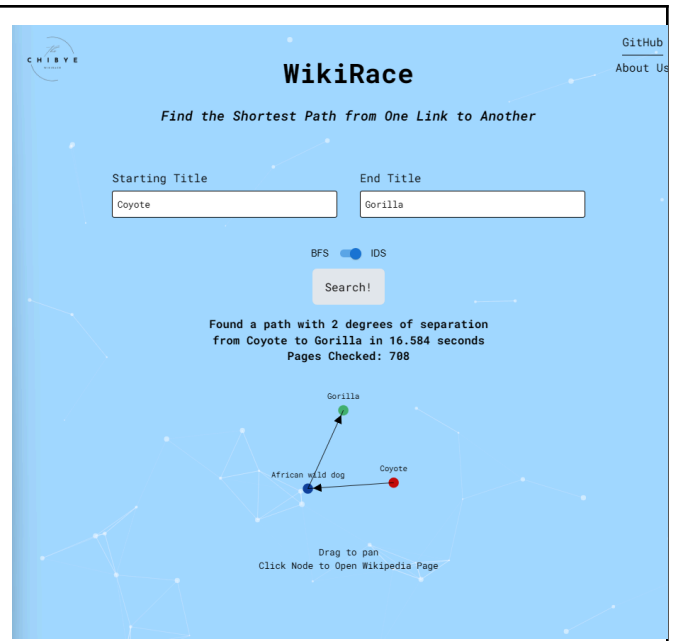
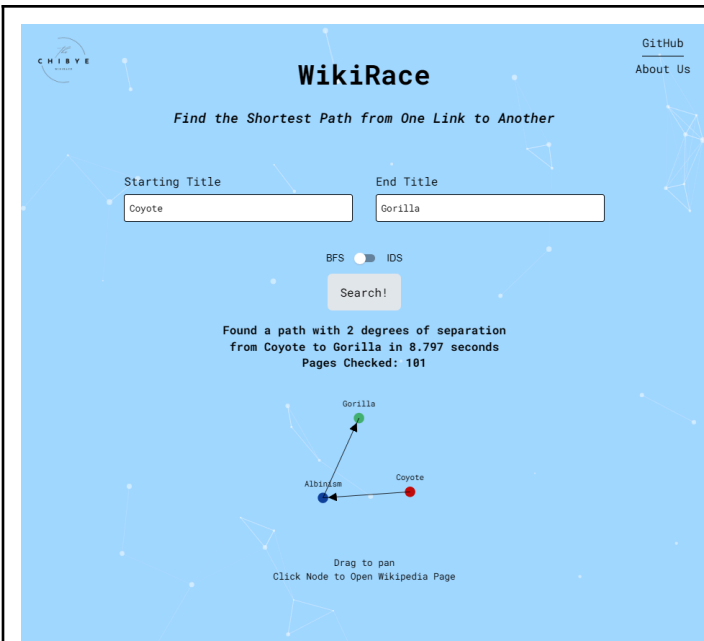
- a. Melakukan pencarian dengan algoritma BFS
- b. Melakukan pencarian dengan algoritma IDS
- c. Kotak teks memberikan sugesti judul halaman Wikipedia
- d. Terdapat Validasi judul halaman Wikipedia
- e. Menunjukkan hasil pencarian dalam bentuk graf
- f. Menunjukkan jumlah halaman yang dikunjungi
- g. Menunjukkan waktu yang diperlukan untuk mencari solusi
- h. Menunjukkan jumlah halaman yang perlu dilalui untuk mencapai halaman tujuan
- i. Penggunaan Caching pada algoritma IDS
- j. Penggunaan Konsep Concurrency dalam algoritma IDS dan BFS

k. Dapat digunakan pada Docker

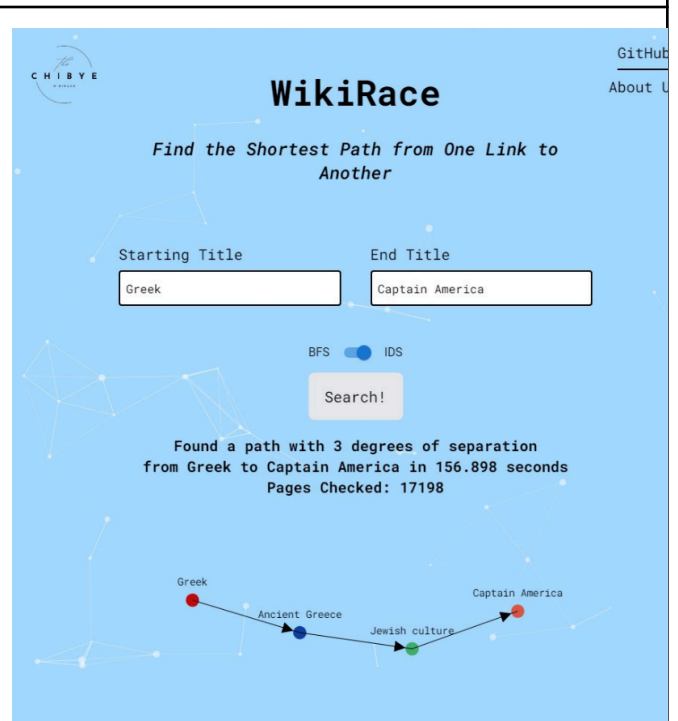
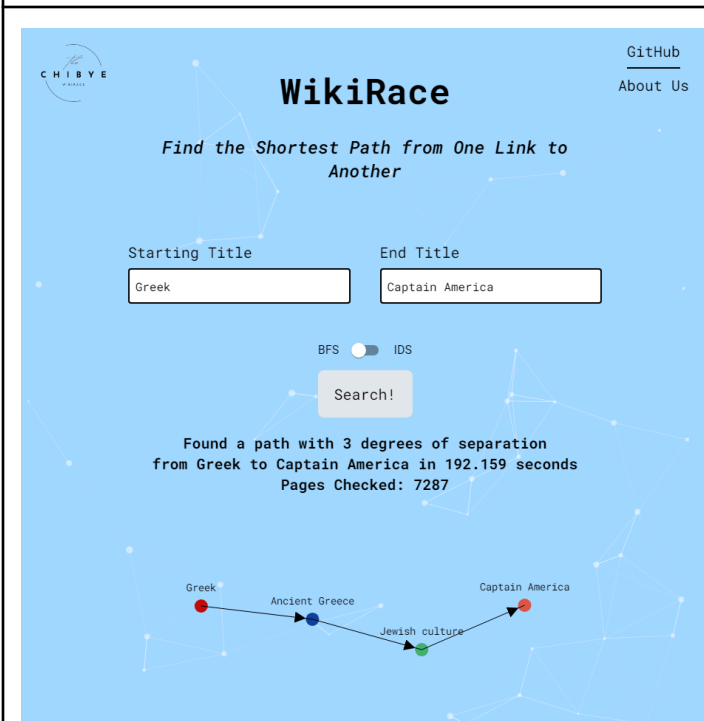
4.3 Hasil Pengujian

Tabel 4.3.1 Pengujian Program

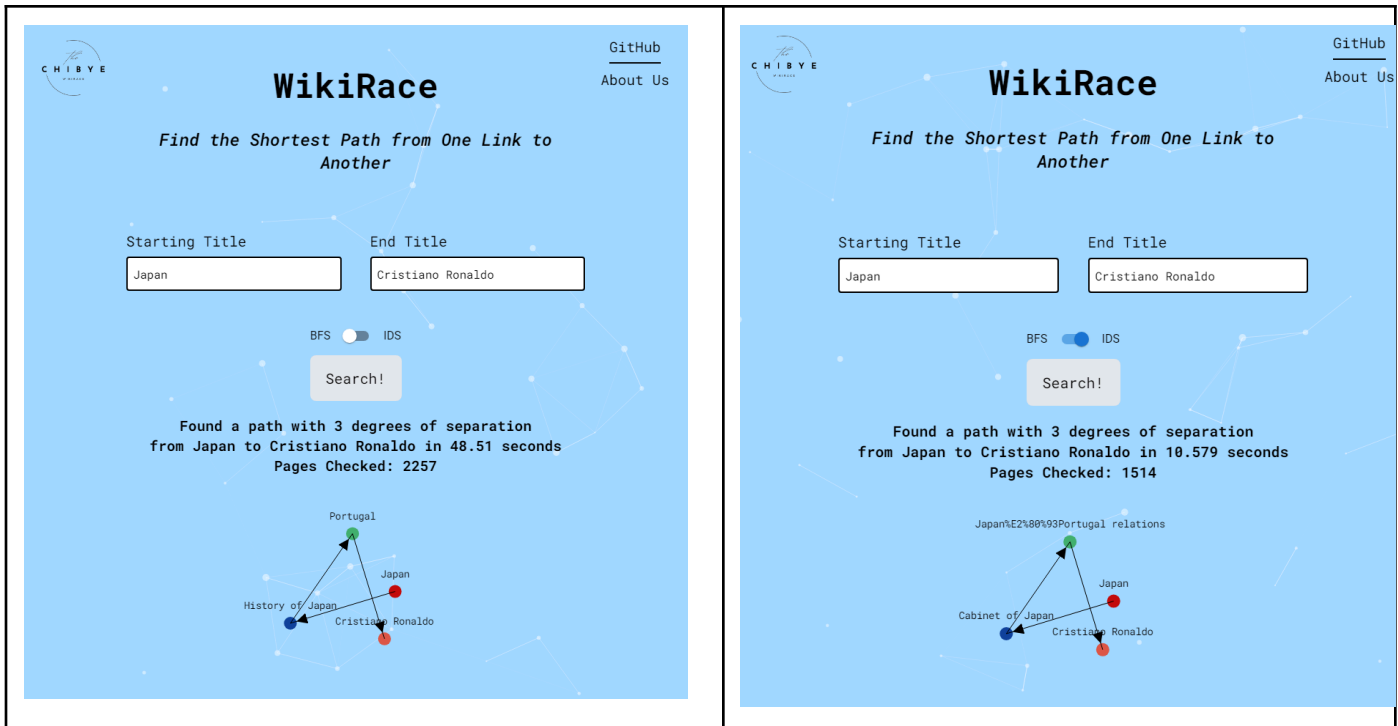
BFS	IDS
<p><i>Test Case Pertama</i> URL awal: Tiger URL tujuan: Coyote</p>	
	
<p><i>Test Case Kedua</i> URL awal: Tiger URL tujuan: Coyote</p>	



Test Case Ketiga
URL awal: Greek
URL tujuan: Captain America



Test Case Keempat
URL awal: Japan
URL tujuan: Cristiano Ronaldo



4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian di atas dan beberapa pengujian lain yang tidak ditampilkan di laporan, algoritma BFS cenderung lebih cepat daripada algoritma IDS pada kasus kedalaman pencarian yang relatif lebih besar. Proses BFS lebih cepat saat kedalaman pencarian mencapai level 2 atau lebih.

Namun, saat kedalaman hanya pada level 1, IDS lebih cepat karena setelah melakukan sekali *scrape* pada laman langsung diperiksa dan dicocokkan dengan tautan tujuan. Kegagalan dalam akses tautan juga sering dialami karena adanya batasan *request* dari wikipedia sehingga pemanfaatan *concurrency* pada bahasa go tidak dapat menjalankan banyak proses sekaligus. Hal ini kemudian menyebabkan proses pencarian menjadi lebih lama karena jumlah proses yang dapat dijalankan secara bersamaan dibatasi.

Pengujian secara berkala untuk mengakses satu tautan wikipedia dan memproses tautan lain yang ada di dalamnya membutuhkan waktu antara 200 ms sampai 2 detik bergantung pada kualitas internet dan kemampuan sumber data komputasi seperti perangkat yang digunakan.

Berdasarkan pengujian di atas, waktu yang dibutuhkan untuk mencapai solusi bertambah secara eksponensial. Sesuai dengan kompleksitas waktu yang telah dijelaskan pada bagian 2.6, waktu yang diperlukan untuk mencari solusi pada kedalaman yang berbeda memiliki perbedaan sangat jelas.

Percobaan untuk mengulangi pengujian terhadap *test case* yang sama beberapa kali menunjukkan perbedaan waktu yang signifikan juga. Hal ini terjadi karena faktor eksternal seperti internet dan sumber daya komputer yang digunakan kurang cepat dan kuat.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar 2 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2023/2024 ini, kami diminta untuk membuat program yang dapat menyelesaikan persoalan WikiRace menggunakan algoritma BFS dan IDS. Kami berhasil membuat aplikasi berbasis web yang dapat menyelesaikan WikiRace dengan kecepatan yang cukup baik. Kami menggunakan bahasa pemrograman Golang dalam membuat sebuah API yang dapat melakukan pencarian menggunakan algoritma BFS dan IDS, yang hasilnya ditampilkan melalui frontend yang berbasis React.

5.2 Saran

Saran untuk kelompok ini diantaranya :

1. Memulai pengerjaan dari lebih awal
2. Meningkatkan komunikasi progress antar anggota
3. Mendalami pengetahuan mengenai bahasa golang sehingga dapat optimasi algoritma yang lebih baik

5.3 Refleksi

Refleksi yang kami dapatkan dari tugas ini antara lain adalah untuk meningkatkan kinerja pengerjaan tugas besar dengan cara pembagian tugas dan pengerjaan tugas yang lebih cepat sehingga tugas dikerjakan dengan lebih baik dan lebih teratur. Kami juga merasa kurangnya komunikasi antar anggota, mengganggu kelancaran dari pengerjaan tugas ini. Hal yang kami dapatkan dari tugas ini adalah pengalaman penting dalam pengembangan aplikasi berbasis web dan permasalahan web scraping. Kami juga mendapat pengalaman dalam pengembangan algoritma BFS dan IDS yang teroptimasi.

LAMPIRAN

Link Repository frontend: https://github.com/NoHaitch/Tubes2_FE_Chibye

Link Repository backend: https://github.com/NoHaitch/Tubes2_BE_Chibye

Link Video:  Tubes 2 STIMA - Algoritma BFS IDS dalam WikiRace - Kelompok Chibye

DAFTAR PUSTAKA

Rinaldi Munir. Breadth/Depth First Search (Bagian 1). Diakses pada 24 April 2024 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

Rinaldi Munir. Breadth/Depth First Search (Bagian 2). Diakses pada 24 April 2024 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>

Geeksforgeeks. Breadth First Search or BFS for a Graph. Diakses pada 22 April 2024 dari <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

Geeksforgeeks. Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS). Diakses pada 22 April 2024 dari <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/>

Techtarget. web application (web app). Diakses pada 25 April 2024 dari <https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app>