

# LAPORAN TUGAS BESAR 2 IF2211

## STRATEGI ALGORITMA

*Pemanfaatan Algoritma Backtracking dalam pembuatan game Wikirace*



**Disusun oleh:**  
**Kelompok 54 Dinasti Pak Lurah**

**Muhamad Rafli Rasyiidin** (13522088)

**M. Hanief Fatkhan Nashrullah** (13522100)

**Indraswara Galih Jayanegara** (13522119)

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

## KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan ini guna memenuhi Tugas Besar 2 mata kuliah IF2211 Strategi Algoritma.

Kami juga ingin mengucapkan terima kasih kepada semua pihak yang telah membantu penulis dalam penyelesaian laporan ini, terutama kepada Dr. Nur Ulfa Maulidevi serta Dr. Rinaldi Munir selaku dosen mata kuliah IF2211 Strategi Algoritma dan para asisten dari Lab IRK yang telah membimbing kami dalam penyelesaian makalah ini.

Laporan ini merupakan dokumen pelengkap untuk projek Tugas Besar 2 IF2211 Strategi Algoritma. Laporan ini menjelaskan landasan teori serta pendekatan *backtracking* yang diimplementasikan untuk menghasilkan solusi dari permainan *Wikirace*.

Penulis berharap laporan ini dapat memberikan manfaat dan memotivasi pembaca untuk terus menjelajahi serta mendalami dunia menarik dari Strategi Algoritma.

Bandung, 26 April 2024

Tim Penulis

## **BAB I**

### **DESKRIPSI TUGAS**

WikiRace atau Wiki Game adalah permainan online yang menggunakan Wikipedia sebagai sumber informasi. Dalam permainan ini, pemain diminta untuk mencari jalan dari satu artikel Wikipedia ke artikel lain yang telah ditentukan, biasanya dengan aturan mencapai artikel akhir dalam waktu sesingkat mungkin atau dengan menggunakan jumlah tautan (klik) paling sedikit. Konsep permainan ini mirip dengan "Six Degrees of Kevin Bacon" yang menghubungkan aktor Hollywood dengan Kevin Bacon melalui sejumlah film yang dimainkan.

Cara bermain WikiRace umumnya melibatkan langkah-langkah berikut:

1. **Memilih Artikel Awal:** Pemain memulai dari suatu artikel Wikipedia yang telah ditentukan sebagai titik awal. Artikel ini biasanya dipilih secara acak atau berdasarkan kategori atau topik tertentu.
2. **Tujuan Akhir:** Pemain juga diberi artikel akhir atau target yang harus dicapai. Tujuan ini bisa berupa artikel apa pun di Wikipedia, dan seringkali tidak ada batasan topik.

WikiRace menjadi populer di kalangan pengguna Wikipedia dan penggemar permainan online karena menyajikan tantangan unik yang menggabungkan pencarian informasi dengan strategi navigasi web. Permainan ini tidak hanya menyenangkan untuk dimainkan, tetapi juga dapat meningkatkan pemahaman tentang struktur dan konten Wikipedia.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1. Algoritma *Backtracking***

Algoritma backtracking adalah pendekatan rekursif untuk menyelesaikan masalah yang melibatkan pemilihan dari sejumlah pilihan yang mungkin. Tujuan dari algoritma ini adalah untuk mencari solusi untuk masalah secara eksploratif, di mana pada setiap langkah, kita membuat keputusan tentang pilihan yang mungkin, dan jika pilihan tersebut tidak mengarah pada solusi yang valid, kita mundur (backtrack) ke langkah sebelumnya untuk mencoba pilihan lain.

Berikut adalah langkah-langkah umum dalam algoritma backtracking:

- 1. Pemilihan Pilihan:** Pada setiap langkah, algoritma harus memilih opsi dari sekumpulan pilihan yang tersedia.
- 2. Validasi Pilihan:** Setelah memilih suatu opsi, algoritma harus memeriksa apakah opsi tersebut valid untuk langkah saat ini atau tidak.
- 3. Rekursi:** Jika opsi yang dipilih valid, maka algoritma akan melanjutkan untuk memanggil dirinya sendiri (rekursi) untuk langkah selanjutnya dari masalah.
- 4. Backtrack:** Jika opsi yang dipilih tidak valid atau tidak menghasilkan solusi yang benar, algoritma akan mundur (backtrack) ke langkah sebelumnya. Pada tahap ini, algoritma akan mencoba opsi lain yang mungkin dari langkah sebelumnya.
- 5. Mengulang dan Memeriksa Solusi:** Algoritma akan mengulang langkah-langkah ini sampai solusi ditemukan atau semua kemungkinan telah dieksplorasi tanpa hasil yang valid.

Contoh penerapan algoritma backtracking termasuk permasalahan seperti penyelesaian puzzle Sudoku, pencarian jalur dalam labirin, penataan ratu pada papan catur (masalah delapan ratu), dan banyak masalah optimasi lainnya di mana kita perlu mengeksplorasi kombinasi berbagai pilihan untuk mencapai solusi yang benar. Keunggulan utama dari algoritma backtracking adalah kemampuannya untuk menemukan solusi secara sistematis dengan menghindari pembuatan

keputusan yang tidak diperlukan, sehingga cocok digunakan untuk masalah-masalah yang dapat dipetakan dalam bentuk pohon keputusan.

## **2.2. Penjelasan Aplikasi Web**

### **a. React**

Aplikasi web yang kami buat, dibuat menggunakan React, sebuah framework JavaScript yang sangat populer untuk membangun antarmuka pengguna yang dinamis dan responsif. Dengan menggunakan React, kami dapat mengembangkan aplikasi web yang efisien dan mudah di-maintain, serta memungkinkan penggunaan komponen-komponen yang dapat digunakan ulang untuk mempercepat pengembangan.

### **b. Tailwind CSS**

Kami menggunakan framework Tailwind CSS karena memiliki beberapa keunggulan yang membuatnya menonjol dalam pengembangan web. Pertama, Tailwind CSS menggunakan pendekatan "utility-first", di mana kami membangun antarmuka dengan menggabungkan kelas-kelas kecil yang mewakili gaya tertentu seperti ukuran font, warna, padding, dan sebagainya. Pendekatan ini memberikan fleksibilitas besar dalam menyesuaikan tampilan tanpa perlu menulis CSS khusus. Selain itu, meskipun menggunakan kelas-kelas bawaan, Tailwind CSS sangat dapat disesuaikan.

## **BAB III**

### **APLIKASI PEMECAHAN MASALAH**

#### **3.1. Langkah-langkah Pemecahan masalah**

Proses awal dalam masalah ini melibatkan web scraping, di mana kita menggunakan teknik untuk mengambil semua link yang ada di sebuah halaman Wikipedia. Link-link yang berhasil diambil kemudian disimpan dalam suatu struktur data seperti larik (array) atau antrian (queue). Dalam konteks ini, larik atau antrian tersebut berfungsi sebagai kumpulan link yang akan dikunjungi berdasarkan algoritma pencarian yang dipilih.

1. Web Scraping: Tahap pertama adalah melakukan web scraping pada halaman Wikipedia. Dalam web scraping, kita menggunakan teknik untuk mengumpulkan informasi dari halaman web, termasuk semua link yang terdapat di halaman tersebut. Untuk Wikipedia, kita dapat mengambil link dari daftar tautan yang terkandung di dalam artikel.
2. Penyimpanan Link: Setelah link-link berhasil ditemukan, langkah berikutnya adalah menyimpannya dalam suatu struktur data. Pilihan yang umum adalah menggunakan larik (array) atau antrian (queue). Larik akan berisi semua link yang sudah dikumpulkan, sementara antrian akan digunakan untuk mengatur urutan kunjungan link-link tersebut.
3. Pemilihan Algoritma: Setiap struktur data (larik atau antrian) akan digunakan dengan algoritma pencarian yang berbeda, tergantung pada kebutuhan:
  - Breadth-First Search (BFS): Jika kita menggunakan antrian (queue), ini sering digunakan dengan algoritma BFS. Pencarian BFS akan mengunjungi semua link pada level saat ini sebelum beralih ke level berikutnya. Dalam konteks web scraping, ini berarti mengunjungi semua link pada halaman sebelum melanjutkan ke link-link yang terdapat di halaman lain.
  - Iterative Deepening Search (IDS): Jika kita menyimpan link dalam larik (array), kita bisa menggunakan algoritma IDS. IDS adalah varian dari Depth-First Search (DFS) yang bertujuan untuk mencari

solusi secara efisien dengan menghindari kelemahan DFS. Dengan IDS, kita mencari secara bertahap dengan meningkatkan kedalaman pencarian pada setiap iterasi.

4. Perbandingan dengan Target Link: Setiap link yang dikunjungi selama proses pencarian akan dibandingkan dengan target link yang ingin dikunjungi. Tujuan dari proses ini adalah untuk menemukan jalur atau informasi tertentu yang terkait dengan target yang telah ditentukan.

Dengan menggunakan kombinasi web scraping dan algoritma pencarian yang tepat, kita dapat melakukan eksplorasi yang efisien dan sistematis terhadap halaman Wikipedia atau sumber informasi lainnya untuk mencapai tujuan tertentu, seperti mengumpulkan informasi atau mencari jalur tertentu dalam struktur hyperlink.

### **3.2. Pemetaan Masalah**

Proses yang terjadi adalah seperti yang ada pada bagian 3.1 yaitu mengunjungi link pertama yang akan dikunjungi, lalu link yang ada di dalam halaman web tersebut akan dimasukkan ke dalam antrian (Queue) atau List (larik) yang digunakan untuk menjalankan algoritma yang dipilih

#### **a. BFS**

Pada algoritma ini, program akan mengunjungi seluruh link pada halaman web pertama dan memasukkan link-link tersebut ke dalam queue, lalu kita akan melakukan visit ke dalam link yang berada pada queue tersebut. Link yang ada pada halaman selanjutnya akan kita masukkan pada queue yang sama sesuai dengan algoritma BFS yang mana cara kerja BFS mirip seperti queue. Kita akan mengunjungi link-link yang berada pada Queue paling depan. Setiap kita melakukan visit kepada sebuah link, kita juga akan melakukan perbandingan dengan judul yang kita tuju.

Jika link yang terdapat pada suatu halaman merupakan link yang kita tuju, maka program akan berhenti dan memberikan link-link yang perlu dikunjungi untuk mencapai link tujuan dari

link awal. Jika tidak ditemukan link yang dituju, maka program akan lanjut mencari link tujuan hingga ditemukan atau hingga mencapai batas tertentu.

**b. IDS**

IDS atau Iterative Deepening Search adalah salah satu algoritma pencarian dalam kecerdasan buatan yang menggabungkan konsep dari dua jenis algoritma pencarian, yaitu Depth-First Search (DFS) dan Breadth-First Search (BFS). Tujuan utama dari IDS adalah untuk memiliki keuntungan dari kedua algoritma ini tanpa memilikinya sendiri, sehingga lebih efisien dalam beberapa situasi.

Pada Algoritma ini kita melakukan hal yang berbeda daripada apa yang dilakukan oleh BFS karena pada algoritma ini pertama kita akan mengunjungi link dengan depth 0, lalu pada depth 0 kita akan melakukan pencarian secara BFS sampai seluruh link pada depth tersebut tidak ada lagi, selanjutnya kita akan mengunjungi depth 1 dan melakukan pencarian lagi secara BFS, sampai seluruh link yang ada pada depth 1 selesai dikunjungi. Kita juga melakukan perbandingan pada Algoritma ini sama seperti yang ada pada BFS. Jika ada link yang kita kunjungi memiliki judul yang sama dengan judul yang kita masukkan sebagai tujuan maka program akan mengeluarkan jalur (path) dari judul awal sampai dengan judul akhir.

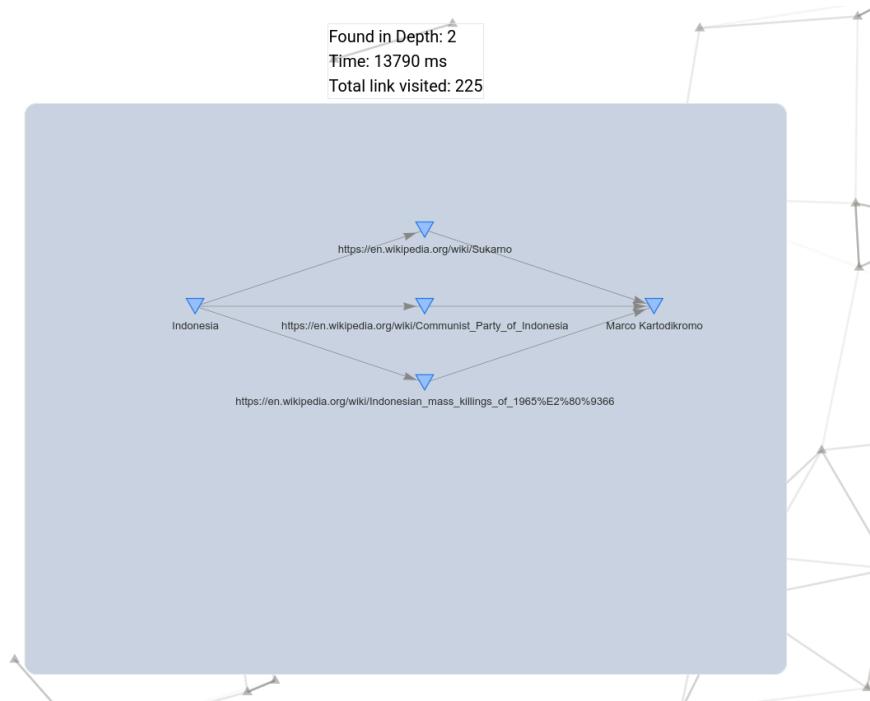
### **3.3. Fitur Fungsional Web**

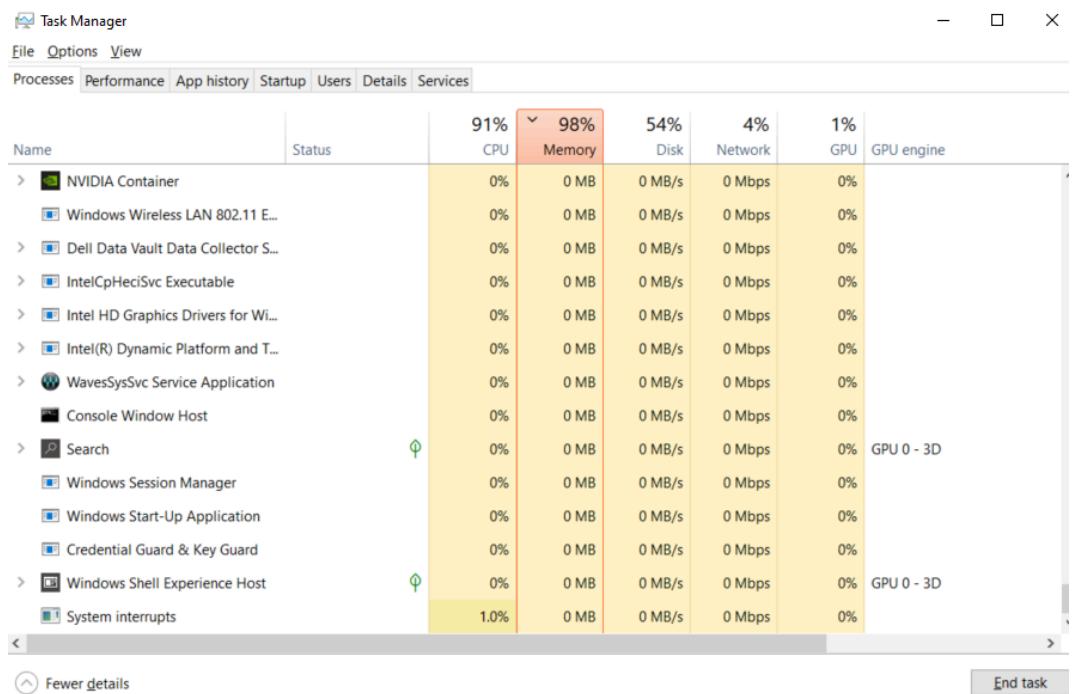
Terdapat beberapa fitur yang dimiliki oleh website kami, yaitu:

1. Routing untuk berganti page
2. Tombol switch untuk berganti metode pencarian, IDS atau BFS
3. Tombol switch untuk berganti jenis pencarian, pencarian menyeluruh atau pencarian biasa
4. Penampilan node graph yang saling terhubung. Node graph tersebut merepresentasikan link yang harus dikunjungi dari link awal agar dapat mencapai link tujuan

### **3.4. Contoh Ilustrasi Kasus**

Misalkan terdapat pengguna yang ingin mencari laman “Marco Kartodikromo” dari laman “Indonesia”. Pengguna dapat mencari jalur setiap laman dengan menggunakan web yang telah kami buat. Dengan memasukkan judul artikel, tipe algoritma, dan tipe pencarian, dapat ditemukan jalur di antara keduanya.





Namun, tidak semua jalur dapat dicari. Kasus untuk mencari terlalu banyak, tetapi tidak ketemu dan harus dipaksa untuk berhenti maka hasilnya seperti ini memory akan berada di atas 90% selama beberapa menit, untuk kasus penulis, device harus di-restart terlebih dahulu agar memory-nya normal.

## **BAB IV**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1. Implementasi Algoritma *IDS* dan *BFS***

##### **1. bfs.go**

Kode di bawah ini merupakan penerapan dari algoritma BFS.

Pertama, program akan masuk ke prosedur crawlerBFS. Kemudian prosedur akan menginisialisasi *queue* sebagai tempat untuk menyimpan antrean link yang akan dikunjungi. Lalu, prosedur crawlerBFS akan memanggil prosedur pembantu yaitu BFS untuk melakukan pencarian secara dengan algoritma BFS. Dalam prosedur ini, akan diinisialisasi *link* yang sudah dikunjungi dan *queue* yang akan dikunjungi pada depth selanjutnya. Pertama, bfs akan memeriksa *link* yang ada pada *queue* pertama pada kalang *queue Iteration*. Kemudian, prosedur BFS akan masuk ke tahap OnHTML. Bagian ini akan memeriksa seluruh *link* yang ada pada *queue* yang baru diperiksa. *Link* yang baru diperiksa akan dimasukkan ke dalam *queueChild*. Setelah *queue* diperiksa, prosedur akan masuk ke tahap OnScrapped yaitu setelah OnHTML. Pada tahap ini akan diperiksa apakah *queue* sudah kosong atau belum dan apakah *path* menuju target sudah ditemukan atau belum. Jika sudah kosong, *queue* dengan *queueChild* akan ditukar isinya sehingga *queueChild* akan menjadi *queue* yang akan diproses selanjutnya. Jika sudah ditemukan, prosedur akan menghentikan *queueIteration* kemudian kembali ke crawlerBFS dan kembali ke pemanggil dari crawlerBFS.

```
1 func BFS(start string, target string, path *safeorderedmap.SafeOrderedMap[[]string], queue
2         *safeorderedmap.SafeOrderedMap[bool], depth *int32, visitCount *int32, searchAll bool, timer
3         *time.Time) {
4     var mutex sync.Mutex
5     var inserter emptyNFound
6     visits := safeorderedmap.New[bool]()
7     queueChild := safeorderedmap.New[bool]()
8     var found int32
9
10    c := colly.NewCollector(
11        colly.AllowedDomains("en.wikipedia.org"),
12        colly.Async(true),
13    )
14
15    c.Limit(&colly.LimitRule{DomainGlob: "*", Parallelism: 2, RandomDelay: 25 * time.Millisecond})
16
17    c.OnRequest(func(r *colly.Request) {
18    })
19
20    c.OnError(func(r *colly.Response, err error) {
21        log.Println("Something went wrong:", err)
22        os.Exit(1)
23    })
24
25    c.OnResponse(func(r *colly.Response) {
26    })
27
28    c.OnHTML("a[href]", func(h *colly.HTMLElement) {
29        link := h.Attr("href")
30        pathInserter, _ := path.Get(h.Request.AbsoluteURL(link))
31        if link == "/wiki/"+target {
32            *timer = time.Now()
33            fmt.Println("Found target link at depth", atomic.LoadInt32(depth)+1, ":", link,
34            time.Since(*timer))
35            if linkNotInside(h.Request.AbsoluteURL(link), h.Request.URL.String(), path) {
36                path.Add(h.Request.AbsoluteURL(link), append(pathInserter, h.Request.URL.String()))
37            }
38            atomic.AddInt32(&found, 1)
39        }
40        if strings.HasPrefix(link, "/wiki/") &&
41            !strings.Contains(link, "File:") &&
42            !strings.Contains(link, "Help:") &&
43            !strings.Contains(link, "Category:") &&
```

```

1      !strings.Contains(link, "Wikipedia:") &&
2      !strings.Contains(link, "Talk:") &&
3      !strings.Contains(link, "Special:") &&
4      !strings.Contains(link, "Portal:") &&
5      !strings.Contains(link, "Template:") &&
6      !strings.Contains(link, "MediaWiki:") &&
7      !strings.Contains(link, "User:") &&
8      !strings.Contains(link, "_talk:") &&
9      link != "/wiki/Main_Page" &&
10     h.Request.AbsoluteURL(link) != h.Request.URL.String() {
11         mutex.Lock()
12         visited, _ := visits.Get(h.Request.AbsoluteURL(link))
13         if !visited {
14             queueChild.Add(h.Request.AbsoluteURL(link), true)
15         }
16         mutex.Unlock()
17         if linkNotInside(h.Request.AbsoluteURL(link), h.Request.URL.String(), path) {
18             path.Add(h.Request.AbsoluteURL(link), append(pathInserter, h.Request.URL.String()))
19         }
20     }
21 })
22
23 c.OnScraped(func(r *colly.Response) {
24     mutex.Lock()
25     atomic.AddInt32(&visitCount, 1)
26     queue.Delete(r.Request.URL.String())
27     if queue.Size() == 0 {
28         inserter.empty = true
29     } else {
30         inserter.empty = false
31     }
32     if atomic.LoadInt32(&found) >= 1 {
33         inserter.found = true
34     } else {
35         inserter.found = false
36     }
37     if queue.Size() == 0 && atomic.LoadInt32(&found) != 1 {
38         queue, queueChild = queueChild, queue
39         queueChild = safeorderedmap.New[bool]()
40         atomic.AddInt32(&depth, 1)
41         fmt.Println("Searching at depth:", atomic.LoadInt32(&depth))
42     }
43 })

```

```

1
2     mutex.Unlock()
3     controlBFS <- inserter
4   })
5
6 queueIteration:
7   for {
8     mutex.Lock()
9     queue.Each(func(key string, value bool) {
10       visits.Add(key, true)
11       c.Visit(key)
12     })
13     mutex.Unlock()
14     controller := <-controlBFS
15
16     if controller.empty && controller.found || (atomic.LoadInt32(&found) >= 1 &&
17       time.Since(*timer) > 500*time.Millisecond && !searchAll) || atomic.LoadInt32(depth) > 9 {
18       if controller.empty && controller.found {
19         atomic.AddInt32(depth, -1)
20       }
21       c.AllowedDomains = []string{""}
22       break queueIteration
23     }
24   }
25
26 func crawlerBFS(start string, target string, path *safeorderedmap.SafeOrderedMap[[string]], depth
27   *int32, visitCount *int32, searchAll bool, timer time.Time) {
28   var wg sync.WaitGroup
29   callerTimer := timer
30   queue := safeorderedmap.New[bool]()
31   queue.Add("https://en.wikipedia.org/wiki/" + start, true)
32   wg.Add(1)
33   go func() {
34     defer wg.Done()
35     BFS(start, target, path, queue, depth, visitCount, searchAll, &callerTimer)
36   }()
37   wg.Wait()
38
39   _, targetInPath := path.Get("https://en.wikipedia.org/wiki/" + target)
40   if atomic.LoadInt32(depth) == -1 && targetInPath {
41     atomic.StoreInt32(depth, 1)
42   }

```

## 2. ids.go

Kode di bawah ini merupakan kode dari algoritma IDS. Implementasi dari IDS lebih sederhana dibandingkan dengan BFS. Dalam *module* colly, kedalaman dari pencarian dapat dibatasi. Pembatasan ini membuat implementasi hanya perlu memanggil colly secara berulang-ulang dengan kedalaman yang berbeda dengan menaikkan nilai kedalaman di setiap iterasi. Pada dasarnya, bagian ini hanya memanggil prosedur DLS (*Depth limited search*) secara berulang ulang sampai solusi ditemukan. Implementasi dari DLS juga cukup sederhana, setiap prosedur mencapai bagian OnHTML, *link* yang baru dimuat akan langsung dikunjungi, sehingga akan seperti seperti algoritma DFS. Ketika kedalaman sudah mencapai batas yang ditetapkan oleh *module* colly, program akan menuju *link* lain yang masih bisa dikunjungi pada kedalaman yang sama. Jika tidak ada yang bisa dikunjungi, prosedur akan kembali ke kedalaman sebelumnya lalu kembali melakukan pencarian.



```

1
2     c.OnResponse(func(r *colly.Response) {
3         fmt.Println("Visited", r.Request.URL)
4         hasVisited, _ := visited.Get(r.Request.URL.String())
5         if !hasVisited {
6             visited.Add(r.Request.URL.String(), true)
7             atomic.StoreInt32(visitCount, atomic.LoadInt32(visitCount)+1)
8         }
9     })
10
11    c.OnHTML("a[href]", func(e *colly.HTMLElement) {
12        link := e.Attr("href")
13        pathInserter, _ := path.Get(e.Request.AbsoluteURL(link))
14
15        if link == "/wiki/"+target {
16            *timer = time.Now()
17            fmt.Println("Found target link at depth", depth+1, ":", link, time.Since(*timer))
18            if linkNotInside(e.Request.AbsoluteURL(link), e.Request.URL.String(), path) {
19                path.Add(e.Request.AbsoluteURL(link), append(pathInserter, e.Request.URL.String()))
20            }
21            atomic.AddInt32(linkFound, 1)
22            inserter.done = true
23            inserter.found = depth
24            controlIDS <- inserter
25            return
26        }
27        if strings.HasPrefix(link, "/wiki/") &&
28            !strings.Contains(link, "File:") &&
29            !strings.Contains(link, "Help:") &&
30            !strings.Contains(link, "Category:") &&
31            !strings.Contains(link, "Wikimedia:") &&
32            !strings.Contains(link, "Talk:") &&
33            !strings.Contains(link, "Special:") &&
34            !strings.Contains(link, "Portal:") &&
35            !strings.Contains(link, "Template:") &&
36            !strings.Contains(link, "MediaWiki:") &&
37            !strings.Contains(link, "User:") &&
38            !strings.Contains(link, "_talk:") &&
39            e.Request.AbsoluteURL(link) != e.Request.URL.String() &&
40            link != "/wiki/Main_Page" {
41                if linkNotInside(e.Request.AbsoluteURL(link), e.Request.URL.String(), path) {
42                    path.Add(e.Request.AbsoluteURL(link), append(pathInserter, e.Request.URL.String()))
43                }
44                e.Request.Visit(link)
45

```

```
1      }
2  })
3
4  c.OnScraped(func(r *colly.Response) {
5  })
6
7  c.Visit("https://en.wikipedia.org/wiki/" + start)
8  c.Wait()
9  inserter.found = -1
10 inserter.done = true
11 controlIDS <- inserter
12 }
13
14 func crawlerIDS(start, target string, path *safeorderedmap.SafeOrderedMap[[]string], depth *int32,
15   visitCount *int32, searchAll bool, timer time.Time) {
16   var linkFound int32
17   callerTimer := timer
18   visited := safeorderedmap.New[bool]()
19   i := 0
20   incrementLoop:
21     for {
22       fmt.Println("Searching at depth:", i)
23       go crawlerDLS(start, target, i, visitCount, &linkFound, path, visited, &callerTimer)
24       controlFlow := &controlIDS
25       if controlFlow.found != -1 && controlFlow.found < i && controlFlow.done ||
26       (atomic.LoadInt32(&linkFound) >= 1 && time.Since(callerTimer) > 500*time.Millisecond && !searchAll) ||
27       atomic.LoadInt32(depth) > 9 {
28         if controlFlow.found != -1 && controlFlow.found < i && controlFlow.done {
29           atomic.AddInt32(depth, -1)
30         }
31         break incrementLoop
32       }
33       if controlFlow.done && controlFlow.found == -1 {
34         i++
35         atomic.StoreInt32(depth, int32(i))
36     }
37 }
```

## 4.2. Penggunaan Program

### 1. Install Dependencies

```
npm install
```

Dependencies yang ada adalah sebagai berikut:

Kalo diatas tidak bisa silahkan di-install dependencies dibawah ini secara manual

```
"@fortawesome/fontawesome-svg-core": "^6.5.2",
"@fortawesome/free-brands-svg-icons": "^6.5.2",
"@fortawesome/free-regular-svg-icons": "^6.5.2",
"@fortawesome/free-solid-svg-icons": "^6.5.2",
"@fortawesome/react-fontawesome": "^0.2.0",
"@testing-library/jest-dom": "^5.17.0",
"@testing-library/react": "^13.4.0",
"@testing-library/user-event": "^13.5.0",
"@tsparticles/react": "^3.0.0",
"@tsparticles/slim": "^3.3.0",
"axios": "^1.6.8",
"axios-jsonp": "^1.0.4",
"react": "^18.3.0",
"react-dom": "^18.2.0",
"react-router-dom": "^6.23.0",
"react-scripts": "5.0.1",
"react-switch": "^7.0.0",
"react-tsparticles": "^2.12.2",
"react-vis-network-graph": "^3.0.1",
"wait-on": "^7.2.0",
"web-vitals": "^2.1.4"
```

### 2. Jalankan frontend

pada folder src pada frontend jalankan perintah

```
npm run start
```

### 3. Jalankan backend

sebelumnya install dulu pm2

```
npm install pm2@latest -g
```

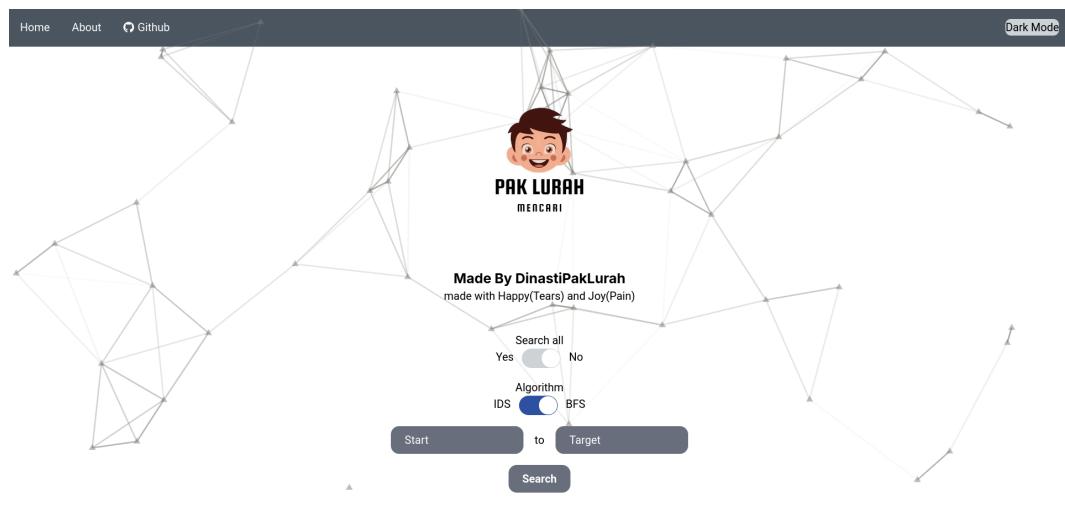
pada folder backend jalankan perintah

```
go build .
```

```
pm2 start backend.exe
```

#### 4.3. Interface dan Fitur Program

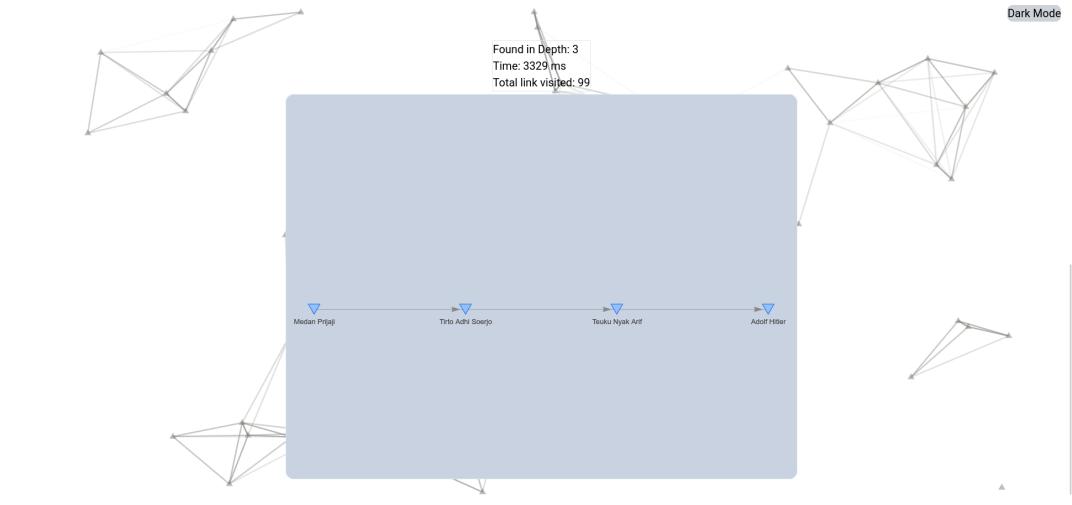
Program ini memiliki fitur dasar seperti pencarian jalur, pilihan algoritma, menampilkan waktu pencarian, dan juga jumlah *link* yang dikunjungi dalam pencarian jalur. Selain fitur dasar, terdapat fitur lain seperti visualisasi dengan graf dan juga pilihan untuk mencari seluruh jalur terpendek. Berikut adalah tampilan utama dari program kami.



Tampilan Utama Program

Cara menggunakan program ini cukup sederhana, cukup pilih pengaturan pencarian sesuai dengan yang diinginkan, lalu masukkan judul artikel tujuan dan judul artikel awal. Akan muncul beberapa sugesti judul artikel yang ada pada wikipedia. Setelah memilih judul dan pengaturan, tekan tombol search dan tunggu sampai graf jalur muncul pada bagian bawah halaman. Setiap *node* pada graf dapat diklik untuk mengunjungi halaman wikipedia yang bersesuaian.

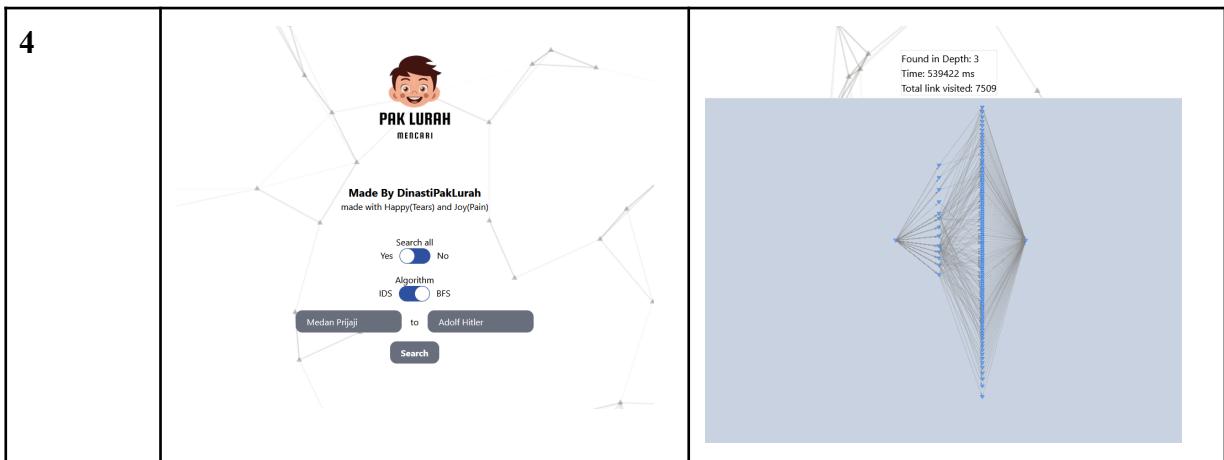
Lama pencarian akan bergantung pada kecepatan internet dan juga jumlah memory (RAM) yang tersedia pada device. Sangat disarankan untuk menggunakan device dengan memory di atas 16GB dan kecepatan internet di atas 3MB/s.



Tampilan Jalur Hasil Pencarian

#### 4.4. Hasil Pengujian

Testcase	Input	Output
1	<p>Search all Yes <input checked="" type="checkbox"/> No</p> <p>Algorithm IDS <input checked="" type="checkbox"/> BFS</p> <p>Java <span style="border: 1px solid black; padding: 2px;">to</span> Malaysia</p> <p>Search</p>	<p>Found in Depth: 2 Time: 10659 ms Total link visited: 444</p>
2	<p>Made By DinastiPakLurah made with Happy(Tears)and Joy(Pain)</p> <p>Search all Yes <input checked="" type="checkbox"/> No</p> <p>Algorithm IDS <input checked="" type="checkbox"/> BFS</p> <p>Ring of Fire <span style="border: 1px solid black; padding: 2px;">to</span> Street dance</p> <p>Search</p>	<p>Found in Depth: 3 Time: 156570 ms Total link visited: 1881</p>
3	<p>Search all Yes <input checked="" type="checkbox"/> No</p> <p>Algorithm IDS <input checked="" type="checkbox"/> BFS</p> <p>Adolf Hitler <span style="border: 1px solid black; padding: 2px;">to</span> Indonesia</p> <p>Search</p>	<p>Found in Depth: 2 Time: 15840 ms Total link visited: 1321</p>



#### 4.5. Analisis Hasil Pengujian

Pada testcase 1, kami mencoba menjalankan program dengan link awal adalah Java dan link tujuan adalah Malaysia. Metode pencarian yang digunakan adalah BFS dengan jenis pencarian menyeluruh. Hasil yang didapatkan, dapat dilihat pada kolom output, testcase 1. Dari gambar tersebut, didapat bahwa kedalaman yang diperlukan untuk menemukan Malaysia dari Java adalah 2 dengan jumlah link yang dikunjungi sebanyak 444. Output node yang dihasilkan sedikit rusak karena jumlah link pada depth 1 yang mengandung link tujuan ada cukup banyak. Waktu yang diperlukan untuk mendapatkan seluruh link tersebut adalah 10 detik. Waktu tersebut cukup lama karena link pada depth 1 cukup banyak sehingga program akan berjalan lebih lama karena perlu mengecek seluruh link tersebut.

Pada testcase 2, kami mencoba menjalankan program dengan link awal Ring of Fire dan link tujuan adalah Street dance. Metode pencarian yang digunakan adalah IDS dengan jenis pencarian biasa. Hasil yang didapat dapat dilihat pada kolom output, testcase 2. Dari hasil tersebut, dapat kita lihat bahwa ada 1 jalur yang dapat diikuti untuk mencapai Street dance dari Ring of Fire. Kedalaman pencarian yang diperlukan untuk menemukan link tujuan adalah 3 dengan total link yang dikunjungi adalah 1881. Waktu yang diperlukan untuk menemukan hasil tersebut adalah 156 detik. Dari perbandingan testcase 1 dan testcase 2, kita dapat melihat

bahwa waktu yang diperlukan akan meningkat jika kedalaman yang digunakan dan link yang dikunjungi bertambah.

Pada testcase 3, kami mencoba menjalankan program dengan link awal adalah Adolf Hitler dengan link tujuan adalah Indonesia. Metode pencarian yang digunakan adalah BFS dengan jenis pencarian biasa. Hasil yang didapat dari input tersebut dapat dilihat pada kolom output, testcase 3. Dari hasil tersebut, dapat dilihat bahwa Indonesia dapat diperoleh pada kedalaman 2 dari Adolf Hitler. Terdapat 1321 link yang dikunjungi dengan total waktu 15 detik. Dari hasil testcase 1 dan testcase 3, dapat kita lihat bahwa testcase 3 memiliki waktu yang cukup mirip dengan testcase 1, tetapi jumlah link yang dikunjungi lebih banyak, hampir seperti testcase 2. Hal tersebut dapat terjadi karena link yang dikunjungi memerlukan waktu scraping lebih cepat. Selain itu, kecepatan jaringan dan spesifikasi juga dapat berpengaruh terhadap kecepatan pencarian.

Pada testcase 4, kami mencoba menjalankan program dengan link awal adalah Medan Prijaji dengan link tujuan adalah Adolf Hitler. Metode pencarian yang digunakan adalah BFS dengan jenis pencarian menyeluruh. Hasil yang didapat dari input tersebut dapat dilihat pada kolom output, testcase 4. Dari hasil tersebut, dapat dilihat bahwa kedalaman yang diperlukan untuk mendapatkan Adolf Hitler dari Medan Prijaji adalah 3 dengan total link yang dikunjungi adalah 7509. Waktu yang diperlukan untuk menemukan link tujuan adalah 530 detik. Waktu tersebut cukup lama karena link yang dikunjungi sangat banyak. Performa perangkat dan kecepatan internet juga berpengaruh terhadap waktu pencarian. Selain itu, jika kita melakukan pencarian terlalu cepat, ada kemungkinan alamat IP kita akan di-*block* untuk sementara sehingga kita tidak dapat mengunjungi link Wikipedia. Waktu yang lama tersebut juga merupakan salah satu cara untuk menghindari *block* dari Wikipedia.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Kesimpulan tugas besar kali ini adalah algoritma IDS dan BFS untuk melakukan Backtracking belum sepenuhnya optimal karena keterbatasan dalam pemahaman library go-colly yang ada di dalam bahasa golang. Selain itu, batasan mencari solusi di bawah 5 menit sangat memberatkan karena banyak sekali faktor yang harus diperhatikan, seperti memory, cpu, internet, dan algoritma. Kami telah mencoba berbagai solusi algoritma, tetapi belum menemukan solusi yang paling optimal untuk mencari solusi.

#### **5.2. Saran**

Menurut kami hal yang perlu dilakukan untuk orang-orang yang ingin mencoba membuat apa yang telah kami kembangkan adalah dengan mengoptimalkan algoritma backtracking yang ada. Juga kami menyarankan untuk eksplor lebih banyak dalam hal pembuatan web karena kesulitan dalam membuat web sendiri dapat mengganggu proses penggerjaan pada tugas besar kali ini. Saran lainnya adalah batasan yang tidak memberatkan

Menurut kami, tugas ini bukan sesuatu yang baik untuk dijadikan menjadi tugas. Bukan karena hal ini terlalu sulit, tetapi terlalu banyak variabel yang perlu diperhitungkan untuk membuat web crawler. IP address yang diblokir, memory laptop yang kurang, serta *bandwidth* internet yang cukup besar (Salah satu anggota kami pernah menghabiskan 14GB dalam 6 jam crawling) menjadi halangan yang cukup besar bagi kami. Tugas yang menyenangkan bukanlah tugas yang mudah, akan tetapi tugas di mana kami memiliki kontrol penuh terhadap apa yang perlu kami kerjakan (atau kami hanya skill issue).



## **LAMPIRAN**

Link Github:

[https://github.com/MRafliRasyiidin/Tubes2\\_DinastiPakLurah](https://github.com/MRafliRasyiidin/Tubes2_DinastiPakLurah)

## DAFTAR PUSTAKA

- Levitin, A. (2011). Introduction to the Design and Analysis of Algorithms (3rd ed.). Pearson.
- Cormen et al. (2009). Introduction to Algorithms 3rd ed. Massachusetts Institute of Technology.
- Ward, Beau.** "Backtracking Algorithms Explained." *freeCodeCamp*, 2022. [Online].  
<https://www.freecodecamp.org/news/backtracking-algorithms-explained/>.
- Rinaldi Munir (2021). Algoritma Backtracking (Bagian 1).  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf>
- Rinaldi Munir (2021). Algoritma Backtracking (Bagian 2).  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf>