

LAPORAN TUGAS BESAR 2 IF2211
STRATEGI ALGORITMA
SEMESTER II TAHUN 2023/2024

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Disusun oleh:
Kelompok 16 - WikiDiDi

Melati Anggraini	(13522035)
Andhita Naura Hariyanto	(13522060)
Diana Tri Handayani	(13522104)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

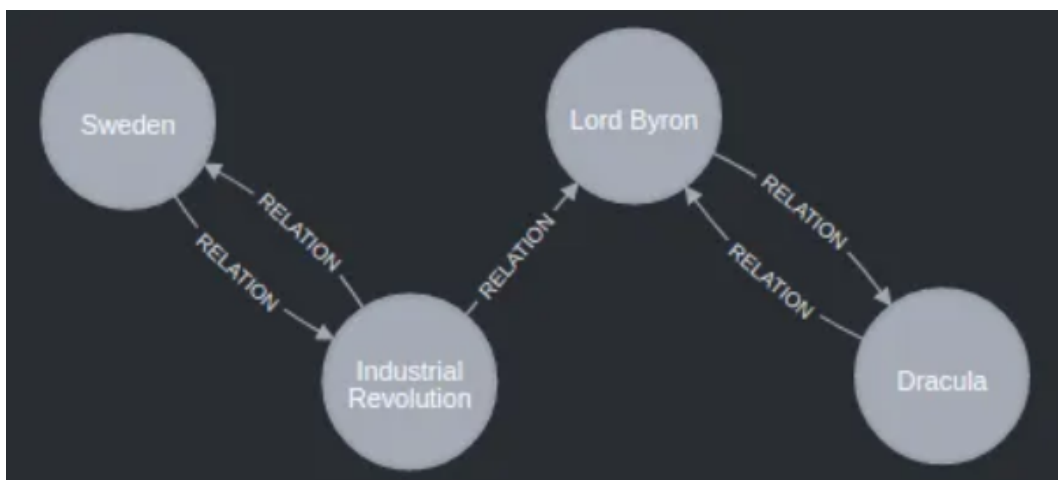
DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
BAB II.....	4
2.1. Dasar Teori.....	4
2.1.1. Penjelajahan Graf.....	4
2.1.2. Algoritma Breadth-First Search.....	5
2.1.3. Algoritma Iterative-Deepening Search.....	6
2.2. Aplikasi Web yang Dibangun.....	7
BAB III.....	8
3.1. Langkah-Langkah Pemecahan Masalah.....	8
3.2. Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS dan BFS.....	10
3.2.1. Breadth-First Search.....	10
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	11
3.4. Contoh Ilustrasi Kasus.....	11
BAB IV.....	13
4.1. Spesifikasi Teknis Program.....	13
4.1.1. Breadth-First Search.....	13
4.1.2. Iterative-Deepening Search.....	16
4.2. Tata Cara Penggunaan Program.....	18
4.3. Hasil Pengujian.....	19
4.4. Analisis Hasil Pengujian.....	20
BAB V.....	22
5.1. Kesimpulan.....	22
5.2. Saran.....	22
5.3. Refleksi.....	23
LAMPIRAN.....	24
DAFTAR PUSTAKA.....	25

BAB I

DESKRIPSI MASALAH

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



Gambar

1.1 Ilustrasi Graf WikiRace (Sumber:

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

Spesifikasi Tugas Besar 2 :

- Buatlah program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace.
- Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan.
- Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms).
- Program cukup mengeluarkan salah satu rute terpendek saja (cukup satu rute saja, tidak perlu seluruh rute kecuali mengerjakan bonus).

- Program berbasis web, sehingga perlu dibuat front-end dan back-end (tidak perlu di-deploy).
- Repository front-end dan back-end diizinkan untuk dipisah maupun digabung dalam repository yang sama.
- Program wajib dapat mencari rute terpendek kurang dari 5 menit untuk setiap permainan.

BAB II

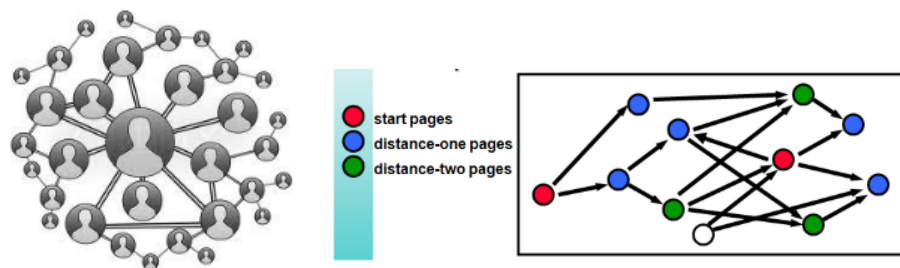
LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Penjelajahan Graf

Penjelajahan graf berarti mengunjungi simpul-simpul dalam graf dengan cara yang sistematis untuk menemukan solusi dari permasalahan yang direpresentasikan dengan graf. Algoritma pencarian solusi berbasis graf dibagi menjadi dua tipe, yaitu tanpa informasi (uninformed/blind search) dan dengan informasi (informed search). Algoritma dengan tipe tanpa informasi berarti pencarian solusi tidak mempertimbangkan informasi tambahan. Algoritma-algoritma yang termasuk ke dalam algoritma tipe tanpa informasi adalah DFS, BFS, Depth Limited Search, Iterative Deepening Search, dan Uniform Cost Search. Algoritma bertipe dengan informasi menggunakan pencarian berbasis heuristik dengan mengetahui non-goal state yang lebih menjanjikan daripada yang lain. Algoritma-algoritma yang termasuk ke dalam algoritma tipe ini adalah Best First Search, A*.

Pencarian solusi dengan representasi graf memiliki dua pendekatan, yaitu graf statis dan graf dinamis. Pendekatan graf statis berarti permasalahan sudah dalam bentuk struktur data graf. Pendekatan graf dinamis berarti permasalahan baru dibentuk sebagai struktur data graf ketika proses pencarian dilakukan.



Gambar 2.1.1.1 Representasi Permasalahan dalam Bentuk Graf untuk Jaringan Laman Web dan Graf Sosial (Sumber :

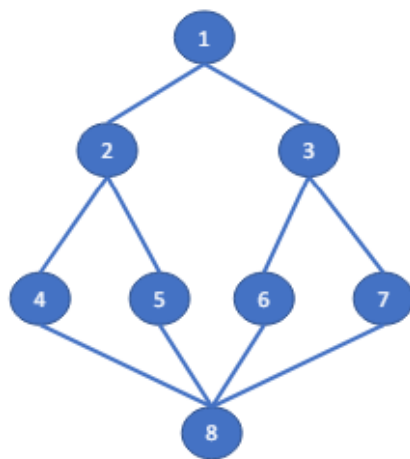
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Ba g1-2024.pdf>)

2.1.2. Algoritma Breadth-First Search

Breadth-First Search (BFS) adalah metode pencarian berbasis graf yang melakukan pencarian solusi secara melebar. Breadth-First Search akan melakukan pencarian dengan mengunjungi satu per satu seluruh simpul yang bertetangga terlebih dahulu hingga goal state tercapai.

Algoritma Breadth-First Search memiliki tiga struktur data :

1. Matriks ketetanggaan yang menyatakan ketetanggaan antara dua simpul
2. Antrian q untuk menyimpan simpul yang telah dikunjungi
3. Tabel boolean “dikunjungi” yang menyatakan apakah suatu simpul telah dikunjungi atau belum.



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

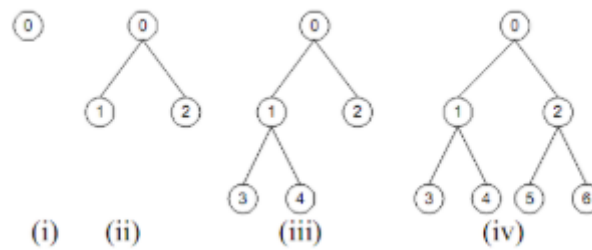
Urutan simpul yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 2.1.2.1 Proses Pencarian Solusi dengan Algoritma Breadth-First Search

(Sumber :

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Ba g1-2024.pdf>)

Pencarian solusi dengan algoritma Breadth-First Search dapat diterapkan pada permasalahan yang direpresentasikan dalam graf statik maupun graf dinamis. Pada permasalahan yang direpresentasikan dalam graf dinamis, pembangkitan status baru pada proses pembentukan pohon ruang status dilakukan pada level yang setara terlebih dahulu.



Gambar 2.1.2.2 Proses Pembangkitan Status pada Pembentukan Pohon Ruang Status dalam Pencarian Solusi Graf Dinamis secara BFS

(Sumber :

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba%20g2.pdf>)

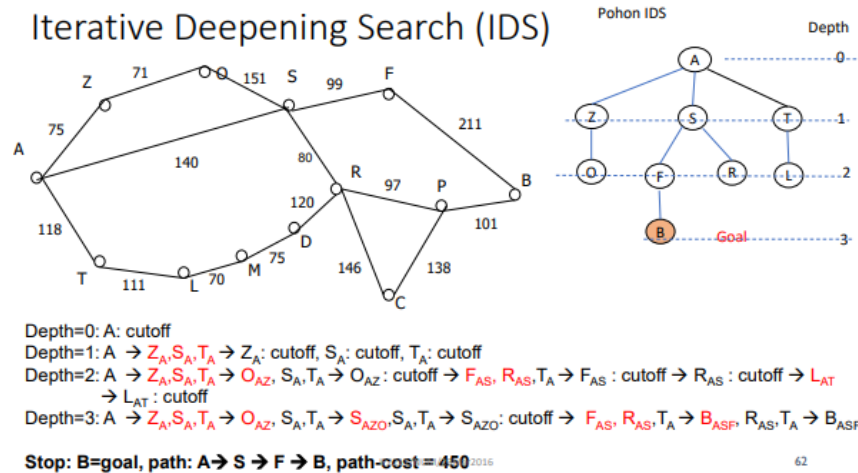
2.1.3. Algoritma Iterative-Deepening Search

Iterative Deepening Search (IDS) adalah sebuah strategi pencarian yang menggabungkan konsep dari Depth-First Search (DFS) dengan peningkatan nilai kedalaman yang diiterasi secara bertahap. IDS bertujuan untuk mengatasi keterbatasan DFS, yaitu masalah terkait kedalaman yang mungkin sangat panjang atau berupa sirkuit sehingga dapat mempengaruhi efisiensi dan keberhasilan pencarian. Persoalan yang menggunakan solusi IDS, biasanya diasumsikan simpul sebagian besar ada di level atau kedalaman bawah, sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali.

Pencarian dimulai dengan menelusuri pohon ruang status atau graf menggunakan algoritma Depth-First Search dengan batasan kedalaman 1 dan batasan kedalamannya akan meningkat secara perlahan hingga simpul tujuan ditemukan atau semua simpul telah ditelusuri. Secara singkat, DLS merupakan IDS dengan batasan kedalaman tertentu secara terurut dari kedalaman satu. Apabila tujuan belum ditemukan pada kedalaman tersebut, maka nilai kedalaman akan ditambah. Lalu, dilakukan kembali.

IDS menjamin penemuan jalur terpendek jika memang simpul tujuan ada pada graf atau pohon ruang status karena memiliki batasan kedalaman yang akan ditingkatkan secara perlahan. IDS memiliki kompleksitas ruang yang serupa dengan DFS karena hanya perlu menyimpan simpul-simpul pada batasan kedalaman yang

sedang ditelusuri dan menghindari kompleksitas ruang eksponensial dari BFS karena pendalaman dalam penelusurannya dilakukan secara iteratif.



Gambar 2.1.3.1 Ilustrasi Penelusuran Simpul dengan Iterative-Deepening Search

(Sumber :

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Ba g2.pdf>)

2.2. Aplikasi Web yang Dibangun

Aplikasi web WikiDidi dibangun dengan menggunakan tech stack JavaScript untuk frontend dengan styling CSS (menggunakan framework React) dan Go yang digunakan untuk pengembangan backend. Aplikasi web dibangun dengan fitur utama melakukan pencarian jalur antara dua laman Wikipedia menggunakan dua jenis algoritma, Breadth-First Search dan Iterative-Deepening Search. Aplikasi web akan menampilkan jalur terpendek, durasi pencarian, dan visualisasi jalur yang dihasilkan.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Langkah–langkah pemecahan masalah dengan algoritma Breadth-First Search :

1. Inisialisasi sebuah queue kosong yang nantinya akan menjadi queue untuk menyimpan seluruh simpul hidup hasil ekspansi penelusuran.
2. Inisialisasi map ‘dikunjungi’ yang diberi nama ‘visited’ untuk menandai sudah dikunjunginya suatu laman Wikipedia.
3. Inisialisasi *array of string* yang diberi nama path untuk menyimpan jalur penelusuran laman-laman Wikipedia.
4. Dilakukan pencarian seluruh laman Wikipedia yang tertaut dengan laman Wikipedia awal pencarian dengan menggunakan *web scraping*.
5. Seluruh link yang memenuhi kondisi diterimanya hasil web scraping akan di-enqueue ke dalam queue simpul hidup.
6. Dilakukan dequeue tautan paling awal dari queue simpul hidup.
7. Apabila hasil dequeue sama atau cocok dengan laman Wikipedia yang menjadi target pencarian, pencarian selesai dan program mengembalikan pencatatan laman-laman yang dilalui dari laman awal menuju laman tujuan.
8. Apabila hasil dequeue tidak sama dengan laman Wikipedia yang menjadi target pencarian, ulangi langkah pencarian dimulai dari langkah tiga hingga laman yang di-dequeue sama dengan laman target atau queue simpul hidup kosong.

Langkah–langkah pemecahan masalah dengan algoritma Iterative-Deepening Search :

1. Inisialisasi Node awal dan Node tujuan yang akan dihubungkan dalam suatu graf pohon.
2. Inisialisasi *map* dengan key berupa string nama suatu laman Wikipedia dan value berupa boolean untuk menandai laman atau Node tersebut sudah dilalui sehingga terhindar dari terbentuknya sirkuit.

3. Inisialisasi *map* dengan key berupa string nama suatu laman Wikipedia dan value berupa integer untuk menandai seberapa banyak laman tersebut dikunjungi. Hal ini digunakan untuk menghitung jumlah kunjungan laman dalam algoritma IDS.
4. Membuat fungsi Depth Limited Search (DLS) yaitu mencari seluruh laman terkait menggunakan *web scrapping*, hingga kedalaman tertentu sesuai parameter.
5. Melakukan DLS secara iteratif dari kedalaman satu hingga kedalaman maksimal (*max depth*) sesuai kesepakatan atau hingga laman tujuan yang dicari ditemukan.
6. Apabila laman ditemukan tujuan tidak ditemukan hingga kedalaman maksimal, akan dikeluarkan hasil nil.

Langkah-langkah mencari seluruh laman Wikipedia yang tertaut pada sebuah laman Wikipedia :

1. Fungsi menerima input judul laman Wikipedia.
2. Dapatkan bentuk HTML dari laman Wikipedia yang sedang ditelusuri
3. Cari elemen anchor pada bagian #mw-context-text untuk mencari hyperlink dari laman Wikipedia yang sedang ditelusuri.
4. Melakukan parsing bagian program yang mengandung elemen anchor dengan goquery.
5. Jika ditemukan, melakukan pencarian elemen anchor yang beratribut "href".
6. Jika link berhasil ditemukan, lakukan parsing terhadap link.
7. Inisialisasi sebuah list of link kosong untuk menyimpan seluruh link Wikipedia yang valid.
8. Apabila komponen host dan komponen path dari link hasil parsing memenuhi kondisi yang menyatakan link tersebut merupakan laman artikel Wikipedia, link akan diappend ke links.
9. Web scraper akan mengembalikan judul-judul artikel dari laman Wikipedia yang valid.
10. Apabila tidak ada link yang memenuhi akan dikirim pesan link valid tidak ditemukan.

3.2. Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma IDS dan BFS

3.2.1. Breadth-First Search

Pencarian solusi dari permasalahan Wikirace dengan algoritma BFS menggunakan konsep algoritma searching dengan graf dinamis. Pembentukan pohon ruang status direpresentasikan struktur data queue yang berisikan seluruh tetangga simpul. Simpul merepresentasikan judul dari laman Wikipedia yang sedang ditelusuri. Tetangga-tetangga dari simpul berarti seluruh hyperlink yang ada pada laman Wikipedia yang sedang ditelusuri. Algoritma Breadth-First Search juga menggunakan struktur data map, tepatnya map[string]bool, untuk menandai laman-laman yang sudah dikunjungi oleh simpul ekspansi. Program juga menggunakan struktur data map, tepatnya map[string]string untuk melakukan jalur penyimpanan penelusuran tiap laman Wikipedia.

Node awal : Laman Wikipedia asal

Node : Laman Wikipedia yang sedang ditelusuri

Goal state : Laman Wikipedia target

Queue : Penyimpanan seluruh simpul hidup yang dilakukan dengan prinsip FIFO (First In First Out)

Tabel boolean dikunjungi : Struktur data map[string]bool untuk menandai simpul-simpul yang telah dikunjungi

3.2.2. Iterative-Deepening Search

Pencarian solusi dengan algoritma IDS menggunakan konsep pencarian dalam graf pohon dinamis. Pembentukan pohon IDS direpresentasikan dalam tipe data buatan 'Node' yang memiliki atribut *Name*, *Children*, dan *Parent*. Setiap laman akan diubah menjadi node. Sehingga *Name* berupa string akan memuat judul laman Wikipedia, *Children* berupa list atau slice dari *Node akan memuat seluruh hyperlink yang ada pada laman terkait, dan *Parent* berupa *Node yang akan menyimpan asal laman tersebut didapatkan. Apabila laman tersebut adalah laman paling awal atau *startPage*, maka atribut *Parent* akan memuat null atau nil. Setelah Node awal hingga

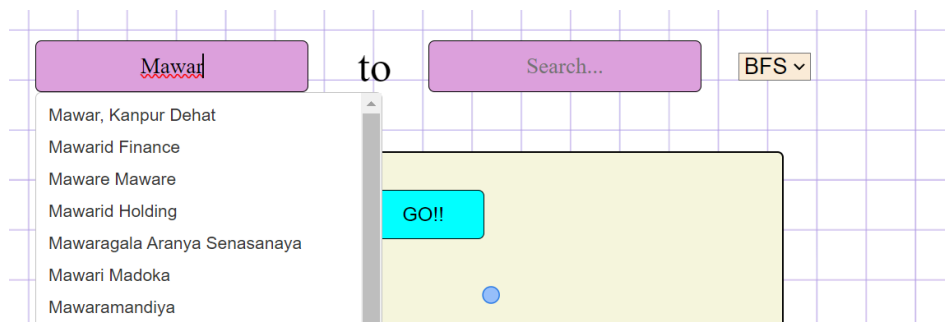
Node tujuan berhasil dibangkitkan dan terbentuk sebuah graf pohon, maka solusi didapatkan dengan menelusuri letak Node tujuan melalui parent hingga Node awal.

Untuk membangkitkan graf pohon dinamis tersebut, dibuat fungsi `IterativeDeepeningWikirace` yang melakukan iterasi fungsi `depthLimitedSearch` atau algoritma DLS mulai dari kedalaman satu hingga kedalaman maksimal. Kedalaman maksimal berupa variabel konstan di dalam program (`maxDepth = 5`). Iterasi berhenti ketika laman atau Node tujuan ditemukan atau telah mencapai kedalaman maksimal tetapi tidak ditemukannya laman atau Node tujuan. Di dalam fungsi `depthLimitedSearch` dilakukan pemanggilan fungsi `getLinksCaching` untuk melakukan *web scrapping* agar didapatkan seluruh hyperlink dari suatu laman.

3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

1. Fitur search yang menggunakan autocomplete wikipedia

Fitur ini memungkinkan pengguna melihat artikel-artikel yang ada di wikipedia. Tujuan lainnya untuk mengurangi random input yang tidak terkait atau tersedia di wikipedia.



2. Fitur Input judul artikel

Fitur ini memungkinkan pengguna untuk memasukkan input judul artikel yaitu `startPage` dan `targetPage`.

3. Fitur direct link ke github dan youtube

Fitur ini memungkinkan pengguna untuk direct link ke github pembuatan web ini serta video penjelasan.

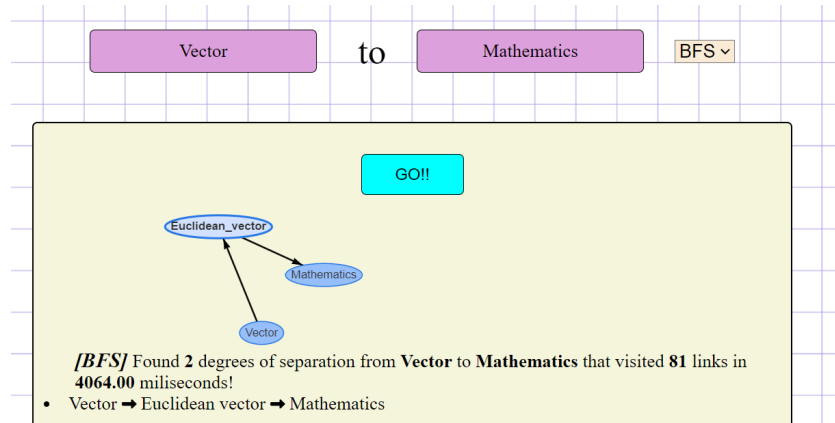
4. Fitur pencarian kecocokan bfs dan ids

Fitur ini memungkinkan pengguna untuk memilih jenis searching yang akan dilakukan untuk mencari kecocokan. Pemilihan dilakukan melalui dropdown yang tersedia.

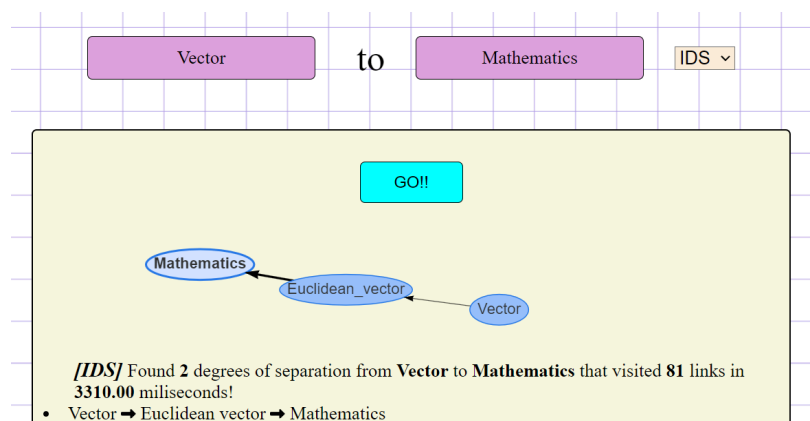
3.4. Contoh Ilustrasi Kasus

Berikut merupakan contoh ilustrasi kasus pencarian jalur antara laman Wikipedia dengan judul “Vector” dan laman Wikipedia dengan judul Mathematics :

- Algoritma Breadth-First Search



- Algoritma Iterative-Deepening Search



BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

4.1.1. Breadth-First Search

Dalam proses penyelesaian permasalahan mencari jalur antara laman Wikipedia asal dan laman Wikipedia target, algoritma Breadth-First Search dalam program menggunakan beberapa struktur data :

1) Queue

Queue pada program digunakan untuk menyimpan hasil ekspansi dari seluruh simpul yang sedang ditelusuri. Ketika queue simpul hidup yang di-*dequeue* dari queue simpul hidup masih memiliki tetangga/hipertaut, atau jika simpul hidup yang di-*dequeue* bukanlah laman target yang user input, akan dilakukan *enqueue* untuk seluruh hipertaut Wikipedia yang ada pada laman tersebut. Queue akan berlaku sebagai tempat penyimpanan bagi simpul hidup yang nantinya digunakan untuk proses pencarian solusi hingga tercapainya laman Wikipedia target.

2) Map

Struktur data map pada program ini digunakan untuk beberapa hal. Pertama, map digunakan untuk menyimpan informasi telah dikunjungi suatu simpul dengan map[string]bool. Judul-judul dari laman Wikipedia yang telah ditelusuri akan masuk menjadi key dalam map dengan value yang bernilai true. Hal ini mencegah program melakukan pemeriksaan kembali terhadap simpul yang sebelumnya telah dikunjungi oleh program. Kemudian, map juga digunakan untuk menyimpan laman Wikipedia yang telah dimasukkan ke dalam queue simpul hidup. Judul-judul dari laman Wikipedia yang telah dimasukkan ke dalam simpul hidup akan menjadi key dalam map dengan value yang bernilai true. Sistem ini menjadi salah satu usaha optimasi program untuk meningkatkan efisiensi pencarian solusi agar program tidak

harus melakukan ekspansi kembali untuk simpul yang sebelumnya telah masuk ke dalam queue simpul hidup.

3) List

Struktur data list atau *slice* digunakan untuk menyimpan data yang secara dinamis berubah ukurannya serta secara aktif dilakukan penambahan atau pengurangan data. Dalam algoritma IDS, list digunakan untuk menyimpan kumpulan judul laman hasil *web scrapping*, serta untuk menyimpan jalur hasil pencarian.

Pencarian dengan algoritma Breadth-First Search pada program dilakukan menggunakan dua fungsi, yaitu `breadthFirstSearch` dan `getLinks`.

1) *breadthFirstSearch()*

Fungsi `breadthFirstSearch` menerima parameter input yaitu `startPage` dengan tipe string yang merepresentasikan laman awal Wikipedia, `currentPage` dengan tipe string yang merepresentasikan laman Wikipedia yang sedang ditelusuri fungsi, `targetPage` dengan tipe string merepresentasikan laman Wikipedia target, `visited` dengan tipe `map[string]bool` yang merepresentasikan tabel boolean ‘dikunjungi’, `appended` dengan tipe `map[string]bool` yang merepresentasikan status sudah atau belumnya suatu laman Wikipedia masuk ke dalam simpul hidup program, `path []string` yang merepresentasikan jalur dari hasil `breadthFirstSearch` yang akan didapat dengan memasukkan setiap laman yang di-*dequeue* dari queue simpul hidup, `queue` dengan tipe `*Queue` yang merepresentasikan *container* simpul hidup, dan `parent` dengan tipe `map[string]string` yang akan digunakan untuk melakukan *tracking* simpul ‘parent’ dari simpul yang sedang ditelusuri.

Fungsi `breadthFirstSearch` dibangun secara rekursif dengan beberapa basis. Basis pertama adalah apabila laman Wikipedia yang sedang ditelusuri cocok dengan atau merupakan laman target, fungsi akan mengembalikan keluaran berupa *array of array of string* yang merepresentasikan jalur terpendek untuk mencapai link target tujuan dari tautan asal. Basis berikutnya adalah apabila laman Wikipedia tidak lagi memiliki hipertaut di dalamnya dan

queue simpul hidup telah habis. Apabila basis ini tercapai, fungsi akan mengembalikan nilai nil. Kemudian, rekurens dari fungsi ini adalah mencari seluruh hipertaut yang ada pada laman Wikipedia yang sedang ditelusuri dengan menggunakan fungsi `getLinks`. Laman Wikipedia kemudian dimasukkan ke dalam map `visited` dengan value `true`. Kemudian seluruh hipertaut yang valid akan di-*parsing* dan di-*enqueue* ke dalam queue simpul hidup.

```

1 func breadthFirstSearch(startPage string, currentPage string, targetPage string, visited map[string]bool, appended map[string]bool, path []string, queue *Queue, parent map[string]string) []string {
2     if currentPage == targetPage {
3         var path []string
4         path = append(path, currentPage)
5         paths := [][]string{
6             for currentPage := startPage {
7                 currentPage = parent[currentPage]
8                 path = append([]string(currentPage), path...)
9             }
10            paths = append(paths, path)
11            return paths
12        }
13        fmt.Println(currentPage)
14        visited[currentPage] = true
15        links, err := getLinks(currentPage)
16        if err != nil {
17            fmt.Println("Error pada page: ", currentPage, ":", err)
18            return nil
19        }
20        if links != nil {
21            for _, link := range links {
22                if !visited[link] || !appended[link] {
23                    appended[link] = true
24                    queue.Enqueue(link)
25                    parent[link] = currentPage
26                }
27            }
28        }
29        if queue.QueueEmpty() {
30            frontLink := queue.Queue()
31            path = append(path, frontLink)
32            result := breadthFirstSearch(startPage, frontLink, targetPage, visited, appended, path, queue, parent)
33            if result != nil {
34                return result
35            }
36        }
37        // Search the parent path
38        visited[currentPage] = false
39        return nil
40        // Output: list of Hyperlink yg jadi path atau nil klo gak ketemu
41    }

```

2) `getLinks()`

Fungsi ini menerima masukan judul laman Wikipedia dalam bentuk string. Kemudian fungsi akan mencari elemen anchor pada bagian `#mw-context-text` untuk mencari hyperlink dari laman Wikipedia yang sedang ditelusuri. Selanjutnya, melakukan parsing bagian program yang mengandung elemen anchor dengan `goquery`. Jika ditemukan bagian program yang mengandung elemen “anchor”, akan dilakukan pencarian elemen anchor yang beratribut “`href`”. Jika tautan berhasil ditemukan, akan dilakukan parsing terhadap tautan. Kemudian, inisialisasi sebuah *list of link* kosong untuk menyimpan seluruh link Wikipedia yang valid. Web scraper akan mengembalikan judul-judul artikel dari laman Wikipedia yang valid. Apabila tidak ada link yang memenuhi akan dikirim pesan link valid tidak ditemukan.


```

1 func getLinks(pageTitle string) ([]string, error) {
2     urlString := "https://en.wikipedia.org/wiki/" + pageTitle
3     doc, err := goquery.NewDocument(urlString)
4     if err != nil {
5         return nil, err
6     }
7
8     var links []string
9     doc.Find("a[href]").Each(func(i int, s *goquery.Selection) {
10        link, exists := s.Attr("href")
11        if exists {
12            // Parse the link to handle encoded characters properly
13            parsedLink, err := url.Parse(link)
14            if err != nil {
15                fmt.Println("Error parsing link:", err)
16                return
17            }
18
19            if parsedLink.Host == "" && strings.HasPrefix(parsedLink.Path, "/wiki/") && !strings.Contains(parsedLink.Path, ":") && !strings.Contains(parsedLink.Path, ".") {
20
21                decodedPath, err := url.PathUnescape(parsedLink.Path)
22                if err != nil {
23                    fmt.Println("Error decoding link path:", err)
24                    return
25                }
26
27                links = append(links, strings.TrimPrefix(decodedPath, "/wiki/"))
28            }
29        })
30    })
31    if len(links) == 0 {
32        return nil, errors.New("no valid links found")
33    }
34    return links, nil
35 }

```

4.1.2. Iterative-Deepening Search

Algoritma Iterative-Deepening Search dibuat dengan menggunakan beberapa struktur data berikut:

1) Pohon atau Tree

Struktur data Pohon atau Tree dibuat dengan bantuan tipe data buatan yaitu Node. Struktur data ini cocok digunakan karena dapat menggambarkan kedalaman, *parent*, dan *child* dengan baik untuk membuat pohon IDS. Sehingga algoritma dapat dibuat sesuai dengan konsep IDS.

```

type Node struct {
    Name      string
    Children  []*Node
    Parent    *Node
}

```

2) Map

Struktur data map digunakan untuk menyimpan suatu variabel atau key tertentu yang memiliki nilai atau value terkait. Pada algoritma IDS, struktur data map digunakan untuk menyimpan data laman yang sudah pernah dikunjungi dengan ditandai nilai boolean. Selain itu juga digunakan untuk menyimpan data laman seberapa banyak telah dikunjungi dengan ditandai nilai integer.

3) List atau *slice*

Struktur data list atau *slice* digunakan untuk menyimpan data yang secara dinamis berubah ukurannya serta secara aktif dilakukan penambahan atau pengurangan data. Dalam algoritma IDS, list digunakan untuk menyimpan kumpulan judul laman hasil *web scrapping*, serta untuk menyimpan jalur hasil pencarian.

Dalam algoritma IDS dibuat 3 buah fungsi yaitu fungsi *getLinkCaching()* untuk melakukan *web scrapping* dengan memanfaatkan *cache memory* untuk mempercepat pencarian, fungsi *depthLimitedSearch()* untuk melakukan pencarian secara prioritas vertikal dengan batas kedalaman sesuai parameter, serta fungsi *iterativeDeepeningWikirace()* untuk memanggil fungsi *depthLimitedSearch()* secara iterasi dari batas kedalaman satu hingga kedalaman maksimal. Berikut adalah langkah-langkah secara rinci dari setiap fungsi:

1) *getLinkCaching()*

Fungsi ini menerima masukan judul laman Wikipedia dalam bentuk string. Dari masukan tersebut akan dilakukan pengecekan dalam cache data apakah terdapat data dari *scraping* sebelumnya. Jika ada langsung kembalikan data berupa *slice of string* tersebut. Jika tidak ada akan dilakukan pengambilan data laman menggunakan *goquery*. Setelah itu, data akan divalidasi untuk dipastikan data yang tersimpan berupa hyperlink menuju laman Wikipedia lain. Simpan dan kunci data yang telah divalidasi dalam cache data. Kembalikan *slice of string* yang memuat data hyperlink yang telah tervalidasi.

2) *depthLimitedSearch()*

Fungsi ini menerima masukan *currentPage* berupa **Node*, *targetPage* berupa **Node*, *depthLimit* berupa *int*, *visited* berupa *map[string]bool*, *totalVisit* berupa *map[string]int*, dan *parentNode* berupa **Node*. Fungsi ini pertama-tama akan mengecek apakah *depthLimit* telah mencapai nol sehingga rekursif dapat dihentikan, serta juga akan mengecek apakah *currentPage* sama dengan *targetPage* yang juga dapat menghentikan rekursif. Setelah itu, fungsi

akan meng-*update* map *visited* dan *totalVisit* dari *currentPage* menjadi *true* dan ditambah satu karena tertanda telah dikunjungi. Dilakukan *web scrapping* dengan fungsi *getLinkCaching()*. Hasil scrapping akan diiterasi dan secara rekursif untuk ditelusuri hingga *targetPage* ditemukan atau mencapai batas kedalaman. Apabila rekursif telah berhenti akan didapatkan Node letak *targetPage* dalam graf pohon yang terbangun.


3) *iterativeDeepeningWikirace()*

Fungsi ini menerima masukan *startPage* berupa string, *targetPage* berupa string, dan *maxDepth* berupa integer. Fungsi akan melakukan inisiasi *startNode* dan *targetNode* dengan mengubah *startPage* dan *targetPage* menjadi Node. Inisiasi juga dilakukan pada map *visited*, map *totalVisit*, serta variabel integer *totalVisitedPages*. Dilakukan iterasi terhadap fungsi *depthLimitedSearch()* dari kedalaman satu hingga kedalaman maksimal atau hingga ditemukan Node tujuan. Setelah Node tujuan ditemukan akan diubah dalam bentuk *slice of string* path dan dikembalikan bersama dengan *totalVisitedPages*.

4.2. Tata Cara Penggunaan Program

Berikut adalah tata cara memulai program WikiDiDi :

1. Lakukan git clone pada repository dari program dengan memasukkan command `git clone github.com/andhitanh/Tubes2_WikiDiDi` pada terminal.
2. Buka folder frontend dan masukkan command berikut
`npm install`
`Npm run start`
3. Buka folder go-backend dan masukkan command berikut
`go run bfsweb.go scrap.go queue.go ids.go`
4. Masukkan input di kedua Searchbox lalu pilih metode pencarian pada dropdown sebelah kanan



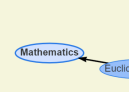

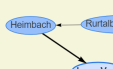


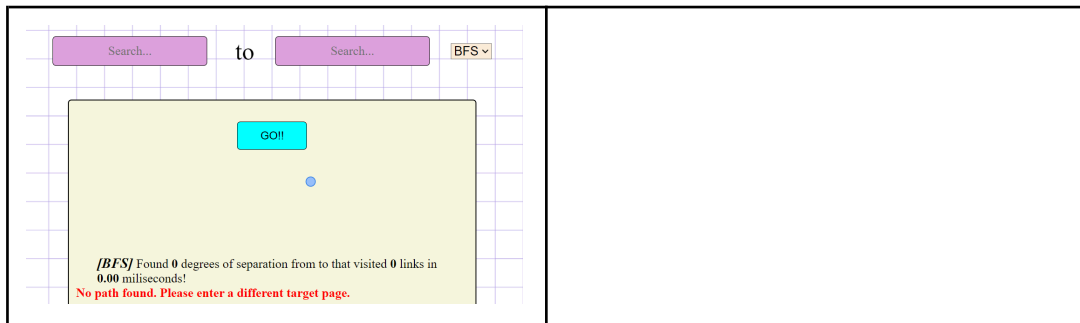
to
BFS ▾

5. Klik button Go untuk memulai pencarian

to
BFS ▾

4.3. Hasil Pengujian

BFS	IDS
<div> <input type="text" value="Joko Widodo"/> to <input type="text" value="Prabowo Subianto"/> BFS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[BFS] Found 1 degrees of separation from Joko_Widodo to Prabowo_Subianto that visited 80 links in 4894.00 milliseconds!</p> <ul style="list-style-type: none"> Joko Widodo → Prabowo Subianto 	<div> <input type="text" value="Joko Widodo"/> to <input type="text" value="Prabowo Subianto"/> IDS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[IDS] Found 1 degrees of separation from Joko_Widodo to Prabowo_Subianto that visited 80 links in 3767.00 milliseconds!</p> <ul style="list-style-type: none"> Joko Widodo → Prabowo Subianto
<div> <input type="text" value="Vector"/> to <input type="text" value="Mathematics"/> BFS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[BFS] Found 2 degrees of separation from Vector to Mathematics that visited 81 links in 4064.00 milliseconds!</p> <ul style="list-style-type: none"> Vector → Euclidean vector → Mathematics 	<div> <input type="text" value="Vector"/> to <input type="text" value="Mathematics"/> IDS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[IDS] Found 2 degrees of separation from Vector to Mathematics that visited 81 links in 3510.00 milliseconds!</p> <ul style="list-style-type: none"> Vector → Euclidean vector → Mathematics
<div> <input type="text" value="Rurtalbahn_GmbH"/> to <input type="text" value="Lars_Vogt"/> BFS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[BFS] Found 2 degrees of separation from Rurtalbahn_GmbH to Lars_Vogt that visited 381 links in 16031.00 milliseconds!</p> <ul style="list-style-type: none"> Rurtalbahn GmbH → Heimbach → Lars Vogt 	<div> <input type="text" value="Rurtalbahn_GmbH"/> to <input type="text" value="Lars_Vogt"/> IDS ▾ </div> <div> <input type="button" value="GO!!"/> </div>  <p>[IDS] Found 2 degrees of separation from Rurtalbahn_GmbH to Lars_Vogt that visited 381 links in 14988.00 milliseconds!</p> <ul style="list-style-type: none"> Rurtalbahn GmbH → Heimbach → Lars Vogt



4.4. Analisis Hasil Pengujian

Algoritma Breadth-First Search dan Iterative-Deepening Search pada setiap pengujian menghasilkan derajat dan rute yang sama. Hal ini menunjukkan kedua algoritma tersebut sama-sama berhasil untuk menyelesaikan permasalahan WikiRace dalam pencarian rute dari suatu laman Wikipedia ke laman Wikipedia lain. Namun, apabila tidak dilakukan input apa-apa maka output yang dihasilkan berupa pesan bahwa tidak ditemukan rute apapun atau nol derajat.

Berdasarkan pengamatan dan hasil pengujian, secara besaran waktu yang dihasilkan, algoritma Iterative-Deepening Search dan Breadth-First Search cenderung menghasilkan waktu yang serupa namun bersifat fluktuatif. Beberapa kasus uji menyatakan hasil pencarian dengan Iterative-Deepening Search memberikan waktu penemuan solusi lebih cepat daripada Breadth-First Search. Beberapa kasus uji juga menyatakan hasil pencarian dengan Breadth-First Search memberikan waktu penemuan solusi lebih cepat daripada Iterative-Deepening Search. Beberapa faktor yang berkontribusi terhadap fluktuatifnya hasil pencarian adalah digunakannya optimisasi dengan caching pada web scraping di pencarian solusi dengan algoritma Iterative-Deepening Search.

Selain itu, perbandingan kompleksitas waktu keduanya sangat bergantung pada struktur dan sifat dari graf yang sedang diproses. BFS memiliki kompleksitas waktu yang lebih pasti dan seringkali lebih cepat dalam menemukan solusi. Karena BFS melakukan *backtracking* yang lebih sedikit dibandingkan IDS. Sedangkan IDS dapat menjadi lebih efisien dalam kasus-kasus ketika kedalaman solusi yang diharapkan tidak terlalu dalam ataupun simpul pada level atas tidak terlalu banyak.

Sedangkan, secara jumlah laman Wikipedia yang dikunjungi selalu sama. Hal tersebut terjadi karena algoritma IDS mencari secara iteratif pada setiap kedalaman

yang menyebabkan setiap simpul pada level atas akan dikunjungi. Sama seperti BFS yang akan mengunjungi setiap simpul pada level atas terlebih dahulu. Jumlah laman Wikipedia yang dikunjungi ini dapat menggambarkan kompleksitas ruang yang dibutuhkan dalam keberjalanan algoritma. Dari hasil pengujian tersebut menunjukkan kompleksitas ruang kedua algoritma tersebut sama besar.

Berdasarkan teori kompleksitas ruang BFS adalah $O(V)$, di mana V adalah jumlah simpul (node) dalam graf. BFS menyimpan semua simpul yang telah dikunjungi dalam struktur data antrian (queue) sebelum mengunjungi simpul-simpul anak dari simpul tersebut. Dengan demikian, BFS membutuhkan ruang yang sebanding dengan jumlah simpul dalam graf. Sedangkan kompleksitas ruang IDS bervariasi. Dalam kasus terburuk, IDS memerlukan ruang $O(b*d)$, dengan b adalah faktor bercabang rata-rata dari simpul-simpul dalam graf, dan d adalah kedalaman solusi yang ditemukan. Apabila diasumsikan simpul tersebar secara merata, perkalian rata-rata cabang dengan kedalaman menghasilkan jumlah seluruh simpul. Sehingga kompleksitas ruang BFS dan IDS dapat memerlukan besar yang sama. Hal ini bersesuaian dengan hasil pengujian di atas.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1.Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma ini, kami berhasil membuat aplikasi web yang mengimplementasikan algoritma Breadth-First Search dan Iterative-Deepening Search untuk menyelesaikan permasalahan pencarian *path* terpendek dari dua laman Wikipedia yang diterima sebagai input. Kami juga berhasil membangun aplikasi web yang memanfaatkan tech stack Go untuk pengembangan backend dan JavaScript dengan styling CSS (framework React) untuk pengembangan frontend.

Berdasarkan pengamatan yang dilakukan dalam pembuatan program, algoritma BFS dan IDS dapat menyelesaikan permasalahan pencarian path secara efisien. Algoritma BFS dapat menghasilkan rute terpendek dari penelusurannya. Akan tetapi, BFS memiliki kecenderungan memakan banyak waktu dan memori akibat perlu dibangkitkannya seluruh simpul yang bertetangga dengan simpul yang sedang ditelusuri dalam proses pencarian sehingga diperlukan optimasi agar pembangkitan simpul dapat dilakukan dengan lebih cepat. Algoritma IDS dapat juga menghasilkan rute terpendek dari penelusurannya. Akan tetapi dibandingkan BFS rata-rata waktu yang dibutuhkan lebih lama. Sedangkan secara memori, algoritma IDS menghabiskan ruang yang sama dengan algoritma BFS. Dengan penambahan optimisasi pencarian solusi pada algoritma IDS, waktu yang diperlukan untuk mencari solusi dengan algoritma IDS dapat mendekati waktu yang diperlukan untuk mencari solusi dengan algoritma BFS.

5.2.Saran

Saran-saran dalam pengembangan penyelesaian persoalan dalam tugas besar ini :

1. Penting untuk melakukan pemahaman mendalam dan eksplorasi mengenai seluruh tech stack yang akan digunakan, dalam hal ini adalah Go untuk pengembangan backend dan JavaScript serta framework React untuk pengembangan frontend agar proses penyelesaian permasalahan lebih efektif dan efisien.

2. Penting untuk memahami dan mendalami materi algoritma Breadth-First Search dan Iterative-Deepening Search guna menghasilkan perencanaan dan pengembangan yang optimal
3. Penting untuk melakukan perencanaan matang untuk menghasilkan alur kerja yang teratur dan optimal

5.3.Refleksi

Pengerjaan tugas besar akan bisa lebih dioptimalkan lagi dengan melakukan perancangan yang lebih matang dan manajemen waktu yang lebih baik dari tiap anggota kelompok.

LAMPIRAN

Link repository :

https://github.com/andhitanh/Tubes2_WikiDiDi.git

Link video :

<https://youtu.be/zFxV5iflvdY?si=YPsx9tRzlLWlPvzE>

DAFTAR PUSTAKA

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

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