

**LAPORAN TUGAS BESAR II
IF2211 STRATEGI ALGORITMA**

**PEMANFAATAN ALGORITMA IDS DAN BFS DALAM
PERMAINAN WIKIRACE**



Disusun oleh:

Kelompok GasTubes (1)

Habibi Galang Trianda 10023457

Nelsen Putra 13520130

Zachary Samuel Tobing 13522016

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

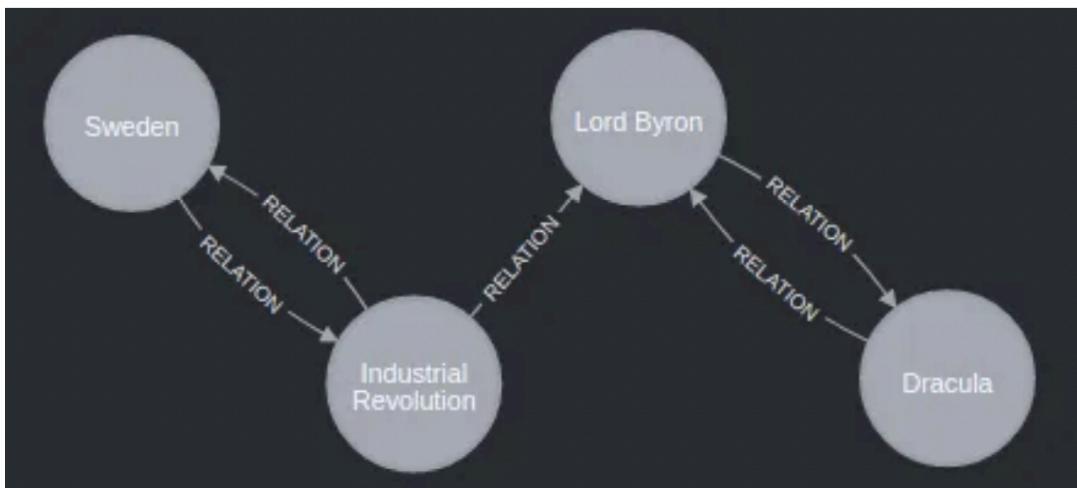
DAFTAR ISI

DAFTAR ISI	<i>i</i>
BAB I	1
BAB II	2
2.1. Dasar Teori	2
2.2. Go Web-based Application Development	5
2.3. Deskripsi Umum Program	6
BAB III	7
3.1. Langkah-Langkah Pemecahan Masalah	7
3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma IDS dan BFS	8
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun	8
3.4. Contoh Ilustrasi Kasus	10
BAB IV	14
4.1. Implementasi Program	14
4.2. Struktur Data dan Spesifikasi Teknis Program	28
4.3. Tata Cara Penggunaan Program	30
4.4. Hasil Pengujian	31
4.5. Analisis Hasil Pengujian	33
BAB V	34
5.1. Kesimpulan	34
5.2. Saran dan Refleksi	35
LAMPIRAN	38
DAFTAR PUSTAKA	39

BAB I

DESKRIPSI TUGAS

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



Gambar 1.1 Ilustrasi Graf WikiRace

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

Tugas yang diminta ialah membuat program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace. Program harus menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Program akan 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 yang dibuat berbasis web, sehingga perlu dibuat *front-end* dan *back-end* (tidak perlu *di-deploy*). Program wajib dapat mencari rute terpendek kurang dari 5 menit untuk setiap permainan.

BAB II

LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Graph Traversal

Graph traversal atau penjelajahan graf adalah proses *traversing* (melintasi) semua simpul atau tepi dalam sebuah graf. Graf dapat berupa struktur data yang terdiri dari simpul-simpul (node atau vertex) yang terhubung oleh tepi (*edge*). Penjelajahan graf merupakan salah satu teknik yang penting dalam ilmu komputer dan matematika terutama dalam algoritma dan struktur data. Ada dua metode utama untuk melakukan penjelajahan graf:

1. Penjelajahan dalam (Depth-First Search/DFS)

Pada metode ini, kita akan mengunjungi sebuah simpul dan kemudian secara rekursif menjelajahi setiap simpul yang terhubung dengan simpul tersebut sebelum kembali ke simpul yang belum dijelajahi sebelumnya. Metode penelusuran graf ini dilakukan traversal secara mendalam. Penelusuran akan dimulai dari sebuah simpul v yang selanjutnya akan diekspansi secara “mendalam”, sehingga penelusuran akan dilakukan ke simpul-simpul daun terlebih dahulu, jika sudah tidak ada simpul yang dapat dikunjungi maka akan dilakukan proses *backtracking*, hal ini bertujuan untuk mencari simpul selanjutnya yang belum dikunjungi.

2. Penjelajahan lintas (Breadth-First Search/BFS)

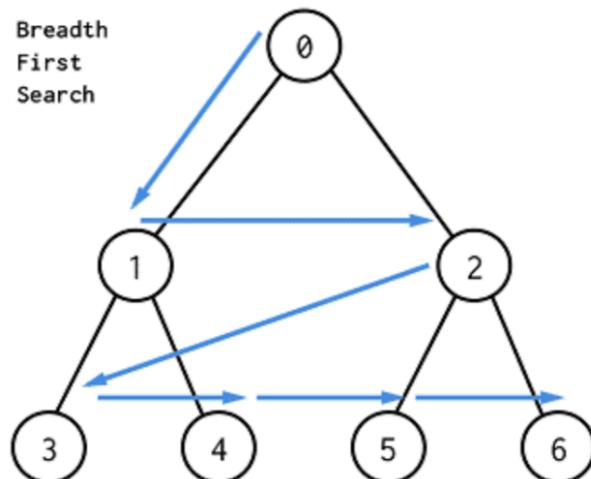
Pada metode ini, kita akan mengunjungi semua simpul yang terhubung dengan simpul awal terlebih dahulu sebelum melanjutkan ke simpul-simpul yang berada pada tingkatan selanjutnya. Metode penelusuran graf ini dilakukan traversal secara melebar. Pencarian dimulai dari sebuah simpul v yang selanjutnya akan “diperlebar” ke simpul-simpul yang bertetanggaan dengan simpul v .

Penjelajahan graf memiliki banyak aplikasi dalam berbagai bidang, termasuk pemodelan jaringan sosial, pemrosesan bahasa alami, optimisasi rute, dan banyak lagi.

2.1.2. Algoritma BFS

Breadth First Search (BFS) adalah metode penelusuran graf secara traversal secara melebar. Pencarian dimulai dari sebuah simpul v yang selanjutnya akan “diperlebar” ke simpul-simpul yang bertetanggaan dengan simpul v . Langkah-langkah yang dilakukan pada *BFS* adalah sebagai berikut:

- a. Kunjungi simpul v
- b. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu
- c. Untuk setiap simpul yang dikunjungi tersebut kunjungi simpul yang belum dikunjungi dan bertetangga simpul tersebut
- d. Ulangi langkah 3 hingga semua simpul dikunjungi



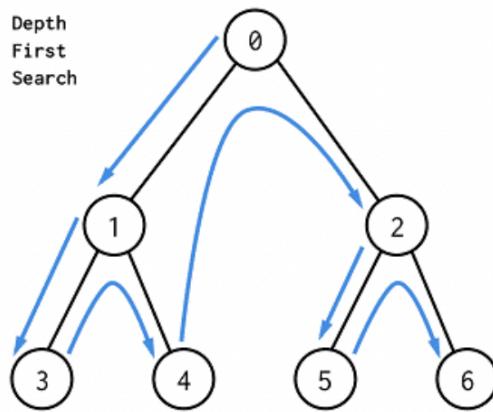
Gambar 2.1.2.1 Ilustrasi Algoritma BFS

2.1.3. Algoritma DFS

Depth First Search (DFS) adalah metode penelusuran graf secara traversal secara mendalam. Penelusuran akan dimulai dari sebuah simpul v yang selanjutnya akan diekspansi secara “mendalam”, sehingga penelusuran akan dilakukan ke simpul-simpul daun terlebih dahulu, jika sudah tidak ada simpul yang dapat dikunjungi maka akan dilakukan proses *backtracking*, hal ini bertujuan untuk mencari simpul selanjutnya yang belum dikunjungi. Berikut langkah-langkah dari algoritma DFS:

- a. Kunjungi simpul v
- b. Kunjungi simpul w yang merupakan simpul tetangga dari v

- c. Ulangi algoritma DFS secara rekursif dengan simpul awal adalah simpul w
- d. Apabila proses pencarian mencapai suatu simpul u sehingga tidak ada lagi simpul tetangga yang belum dikunjungi, dilakukan pencarian runut-balik ke simpul terakhir yang dikunjungi sebelum simpul u dan memiliki simpul tetangga yang belum dikunjungi.
- e. Pencarian berakhir apabila semua simpul telah dikunjungi atau tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai.



Gambar 2.1.3.1 Ilustrasi Algoritma DFS

2.1.4. Algoritma IDS

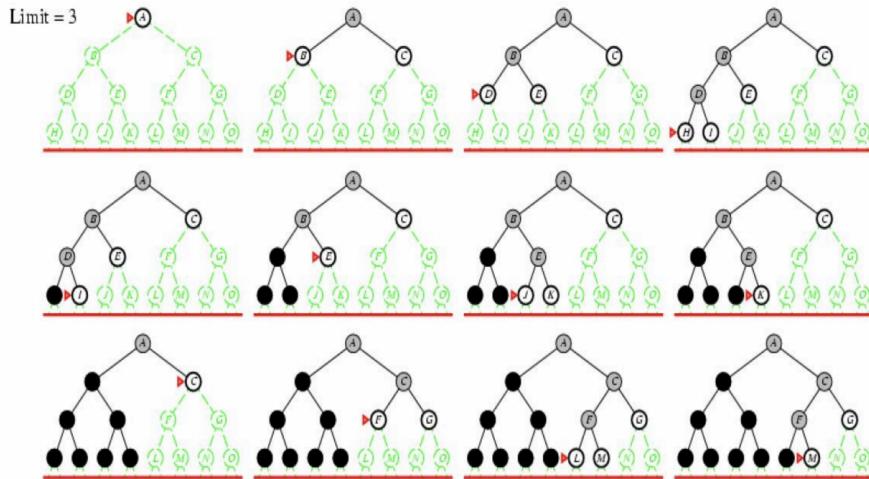
Iterative Deepening Search (IDS) adalah metode penelusuran graf secara traversal dengan melakukan serangkaian DFS, dengan peningkatan nilai kedalaman-cutoff, sampai solusi ditemukan. Asumsi yang umumnya digunakan dalam menerapkan algoritma ini ialah simpul sebagian besar ada di level bawah sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali. Berikut ini merupakan algoritma IDS dalam *pseudocode*.

```

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

Gambar 2.1.4.1 *Pseudocode* Algoritma IDS

Berikut ini merupakan ilustrasi algoritma *Iterative Deepening Search* dengan kedalaman (d) = 3.



Gambar 2.1.4.2 Ilustrasi Algoritma IDS

2.2. Go Web-based Application Development

Web-based application merupakan perangkat lunak yang diakses melalui *browser web* melalui jaringan seperti internet atau intranet. Aplikasi ini tidak diinstal pada komputer atau berjalan secara lokal pada komputer pribadi atau laptop pengguna, berbeda dengan *desktop application* yang berjalan pada komputer pribadi maupun *mobile application* yang berjalan pada *smartphone*. Aplikasi berbasis web berada di *remote server* dan pengguna akan berinteraksi dengannya melalui *browser*. Karena harus diakses pada *web*, maka aplikasi berbasis *web* umumnya harus *compatible* untuk dijalankan di berbagai *operating system* seperti Windows, Linux, iOS, Android, dan MacOS.

Salah satu bahasa yang cukup populer digunakan dalam pengembangan aplikasi berbasis web adalah Go. Go atau yang biasa juga dikenal dengan Golang merupakan bahasa pemrograman *open source* yang dikembangkan oleh *Google*. Go diciptakan dengan tujuan untuk menyediakan bahasa yang mudah dipahami, efisien, dan cepat untuk pengembangan perangkat lunak modern, terutama untuk pengembangan sistem yang berskala besar dan berkinerja tinggi. Go memiliki level yang sama dengan Java dimana bahasa pemrograman ini dihimpun dan diketik dalam bahasa C. Beberapa fitur yang

membuat Go menonjol antara lain adanya pengelolaan memori secara otomatis (garbage collection), kompilasi cepat, dukungan untuk konkurenSI dengan goroutine, penulisan kode yang sederhana dan mudah dipahami, serta pengelolaan dependensi yang terintegrasi melalui *Go Modules*.

2.3. Deskripsi Umum Program

Program yang dibuat akan mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace. Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Kemudian, program akan 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 menggunakan bahasa pemrograman Go untuk *backend* dan React untuk *frontend*. Dalam proses pengembangan aplikasi berbasis web ini, *programmer* biasa menggunakan sebuah IDE buatan Microsoft yang bernama *Visual Studio* untuk mempermudah proses pengembangan dengan bantuan *version control system* untuk membantu pengelolaan versi program bernama *GitHub*.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Untuk menyelesaikan masalah yang sudah disebutkan di atas, diperlukan beberapa langkah-langkah sebagai berikut:

1. Pemahaman Pencarian Solusi sebagai Pohon: Pahami bahwa pencarian solusi dapat diilustrasikan sebagai pohon, dengan artikel awal sebagai akarnya. Setiap clickable link di suatu artikel merupakan simpul anak dari artikel tersebut.
2. Implementasi BFS: Inisialisasi antrian dengan simpul awal sebagai simpul pertama.
3. Implementasi IDS: Mulai dengan kedalaman pencarian 0. Lakukan pencarian dengan kedalaman tersebut menggunakan DLS (Depth-Limited Search), memeriksa artikel child pertama, lalu child pertama artikel tersebut, hingga mencapai batas kedalaman yang ditentukan.
4. Penanganan Solusi: Jika artikel tujuan ditemukan selama pencarian, program akan menghentikan pencarian dan mengembalikan jalur yang diambil dari artikel awal hingga artikel tujuan. Jalur ini akan berisi semua artikel yang dilalui untuk mencapai artikel tujuan.
5. Lakukan pengujian menyeluruh untuk memastikan bahwa algoritma BFS dan IDS berfungsi dengan baik. Uji dengan berbagai skenario masukan dan pastikan bahwa program dapat menelusuri semua artikel sesuai kedalaman dan rutenya dengan waktu minimal.
6. Buat backend menggunakan Go untuk menangani permintaan pencarian dari frontend. Buat antarmuka pengguna menggunakan React untuk memungkinkan pengguna memasukkan artikel awal dan artikel tujuan, serta menampilkan rute yang ditemukan oleh algoritma, beserta jumlah artikel yang dikunjungi, jumlah artikel yang dibutuhkan untuk sampai rute, dan waktu eksekusinya.

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

3.2.1. Pemetaan Masalah ke Algoritma BFS

1. Inisialisasi Antrian: Antrian terdiri dari *array of string* yang melambangkan semua rute yang sejauh ini sudah ditelusuri. Antrian akan digunakan untuk melacak simpul yang akan dieksplorasi dimana setiap antrian pada setiap pemanggilan rekursi melambangkan semua rute dengan kedalaman atau jumlah artikel yang sudah ditelusuri saat ini. Inisialisasikan antrian dengan simpul awal sebagai simpul pertama yang akan dieksplorasi.
2. Ekspansi Simpul: Proses ekspansi simpul saat mencari tetangga dari simpul terakhir dari setiap elemen pada antrian.. Tambahkan semua tetangga yang belum dieksplorasi ke dalam antrian.
3. Penyelidikan Solusi: Saat menambahkan simpul-simpul baru ke dalam antrian, periksa apakah salah satu dari simpul-simpul tersebut adalah solusi. Jika ya, selesaikan algoritma.
4. Pengulangan: Ulangi langkah-langkah 2-3 hingga solusi ditemukan atau hingga antrian kosong, yang menandakan bahwa tidak ada solusi yang ditemukan.

3.2.2. Pemetaan Masalah ke Algoritma IDS

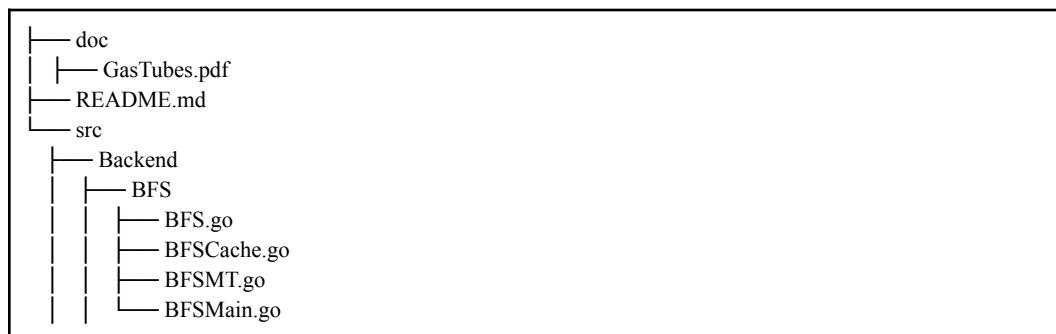
1. Penyesuaian Batas Kedalaman: Set batas kedalaman pencarian ke nilai awal (0).
2. Pencarian dengan Batas Kedalaman: Lakukan pencarian dengan kedalaman tertentu, memeriksa simpul-simpul hingga kedalaman tersebut.
3. Pengecekan Solusi: Saat mencapai batas kedalaman, periksa apakah solusi ditemukan. Jika ya, selesaikan algoritma. Jika tidak, lanjutkan ke langkah berikutnya.
4. Peningkatan Batas Kedalaman: Jika solusi tidak ditemukan pada batas kedalaman tertentu, tingkatkan batas kedalaman dan kembali ke langkah 2.
5. Iterasi: Ulangi langkah-langkah 2-4 hingga solusi ditemukan atau hingga mencapai batas kedalaman maksimum.

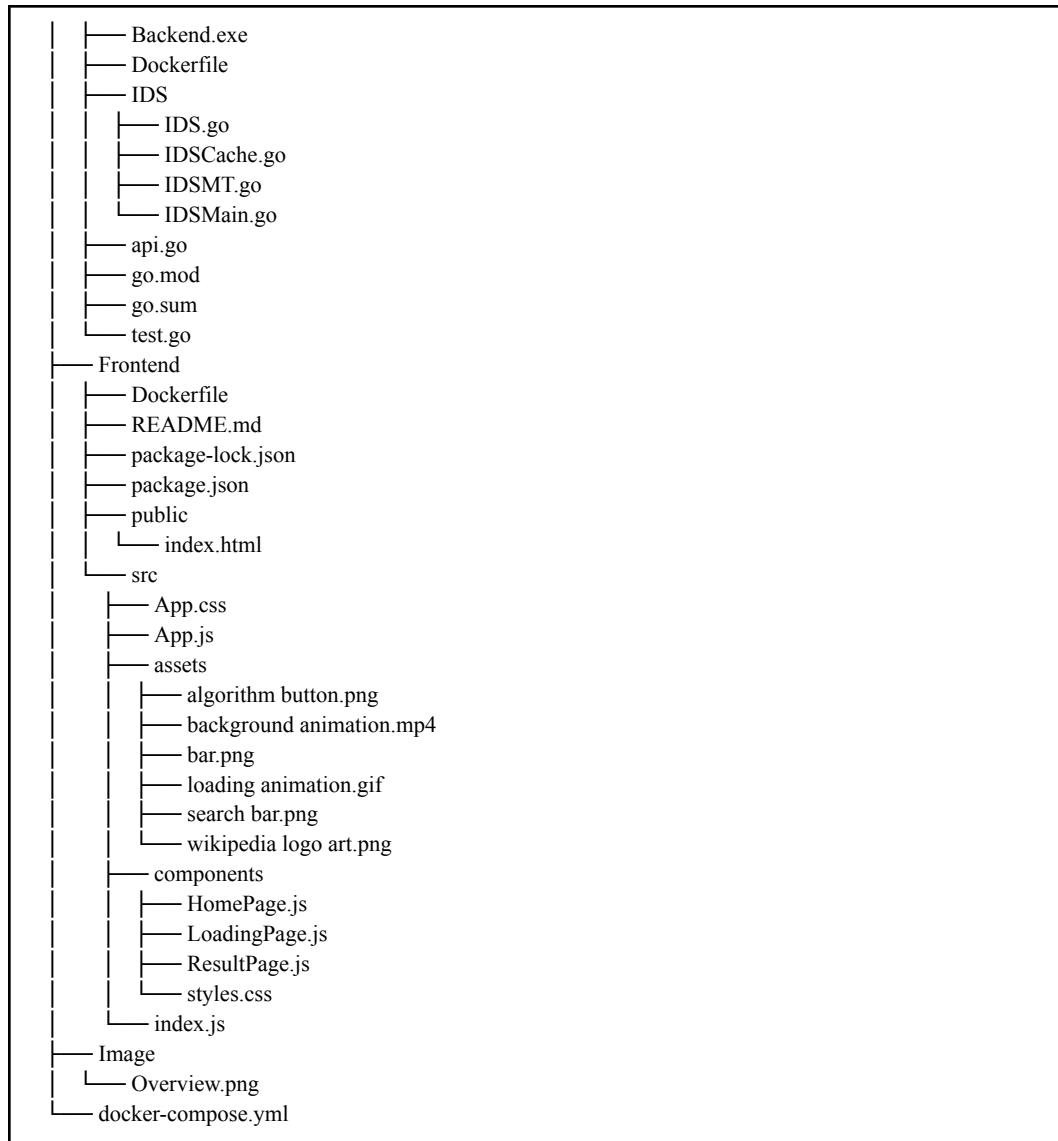
3.3. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

3.3.1. Fitur Fungsional Aplikasi

1. Form Input: Menampilkan formulir yang memungkinkan pengguna untuk memasukkan judul artikel awal dan artikel tujuan.
2. Pilihan Algoritma: Memberikan opsi kepada pengguna untuk memilih algoritma yang akan digunakan dalam pencarian jalur, seperti BFS (Breadth-First Search) atau IDS (Iterative Deepening Search).
3. Hasil Pencarian dalam Graf: Menampilkan hasil pencarian jalur antara artikel awal dan artikel tujuan dalam daftar tautan, di mana setiap tautan merepresentasikan artikel pada jalurnya dengan urutannya.
4. Tampilan Rute: Menampilkan rute atau jalur yang ditemukan dari artikel awal hingga artikel tujuan. Ini bisa berupa daftar artikel yang harus diikuti oleh pengguna untuk mencapai tujuan mereka.
5. Interaktif dan Responsif: Memastikan antarmuka pengguna responsif dan mudah digunakan, dengan dukungan untuk berbagai perangkat dan ukuran layar.
6. Pesan Kesalahan: Hanya memberikan pesan kesalahan dalam *terminal* tetapi program diberhentikan jika artikel yang dimasukkan tidak *valid*.
7. Tombol Pencarian: Tombol yang memungkinkan pengguna untuk memulai pencarian setelah memasukkan informasi yang diperlukan.
8. Waktu Pencarian: Menampilkan waktu yang dibutuhkan untuk menemukan rute antara artikel awal dan artikel tujuan, memberikan pengguna pemahaman tentang kinerja algoritma yang digunakan.
9. Tautan Eksternal: Memberikan tautan eksternal ke artikel di Wikipedia untuk artikel awal, artikel tujuan, dan artikel-artikel yang ada dalam rute yang ditemukan, memungkinkan pengguna untuk langsung mengakses informasi lebih lanjut tentang artikel tersebut.

3.3.2. Arsitektur Aplikasi





3.4. Contoh Ilustrasi Kasus

Kasus penelusuran artikel pada permainan WikiRace ini dapat diilustrasikan dengan graf yang ada pada Gambar 1.1. Secara sederhana kasus ini dapat diilustrasikan sebagai berikut.

Dua orang teman bernama Andi dan Budi sedang memainkan permainan WikiRace. Mereka memilih artikel Wikipedia tentang "Space Exploration" sebagai artikel tujuan mereka. Mereka berdua duduk di komputer masing-masing, dan permainan dimulai.

Langkah-langkah yang akan dilakukan oleh keduanya dalam memainkan permainan ini

ialah:

1. Andi: Dia memulai di artikel Wikipedia tentang "Galaxy". Dari sini, dia mengklik tautan ke artikel tentang "Space Exploration".
2. Budi: Sementara itu, Budi memulai dari artikel Wikipedia tentang "Astronomy". Dia menavigasi ke artikel tentang "Space Exploration" melalui tautan-tautan yang relevan.
3. Andi: Setelah mencapai artikel "Space Exploration", Andi mencoba menemukan tautan yang membawa ke artikel tujuan akhir, "Space Exploration".
4. Budi: Begitu dia sampai di artikel "Space Exploration", Budi melakukan hal yang sama, mencoba menemukan tautan yang akan membawanya ke artikel tujuan akhir.

Pemenang:

Pemenangnya adalah orang pertama yang berhasil mencapai artikel tujuan akhir, "Space Exploration", dengan menavigasi melalui tautan-tautan dalam artikel Wikipedia lainnya.

Contoh Hasil:

Setelah beberapa putaran, Andi berhasil mencapai artikel "Space Exploration" lebih cepat daripada Budi, sehingga dia menjadi pemenangnya.

Tentunya, dalam permainan ini, para pemain bisa melakukan modifikasi dengan memilih artikel tujuan yang berbeda, dan tantangannya bisa menjadi semakin menarik tergantung pada kompleksitas dan kedalaman topik yang dipilih.

Selain itu, kasus ini juga serupa dengan contoh-contoh kasus lainnya yang dapat dilihat pada contoh kasus berikut ini.

1. Kasus 1: Permainan "*Six Degrees of Kevin Bacon*"

Salah satu ilustrasi kasus pencarian lain yang mirip dengan WikiRace adalah permainan "*Six Degrees of Kevin Bacon*". Dalam permainan ini, pemain mencoba untuk menghubungkan aktor Kevin Bacon dengan aktor atau aktris

lainnya melalui hubungan film yang mereka perankan. Pemain harus menemukan jalur yang paling pendek (dalam jumlah film yang paling sedikit) antara Kevin Bacon dan aktor/aktris yang ditentukan.

Permainan ini berdasarkan pada konsep "*Six Degrees of Separation*" yang menyatakan bahwa setiap orang di dunia ini dapat dihubungkan dengan orang lain melalui jaringan hubungan yang terdiri dari tidak lebih dari enam langkah (atau derajat) pengetahuan. Dalam konteks "*Six Degrees of Kevin Bacon*", konsep ini diterapkan pada industri film, dengan Kevin Bacon sebagai pusatnya.

Para pemain bisa mencoba menemukan jalur yang paling pendek antara Kevin Bacon dan aktor/aktris lain dengan menghubungkan film-film yang pernah mereka bintangi dengan aktor/aktris lainnya yang juga bermain dalam film yang sama. Permainan ini menantang pengetahuan tentang film dan menguji kemampuan untuk menemukan hubungan antara berbagai aktor dan film.

2. Kasus 2: Permainan "*Six Degrees of Wikipedia*"

Contoh kasus lain yang mirip dengan WikiRace adalah "*Six Degrees of Wikipedia*". Dalam permainan ini, pemain diberikan dua topik atau artikel yang berbeda di Wikipedia dan harus menemukan jalur terpendek antara kedua topik tersebut hanya dengan mengklik tautan dalam artikel. Pemain mencoba untuk menavigasi melalui artikel-artikel terkait dan menemukan hubungan yang memungkinkan antara topik awal dan akhir.

Misalnya, jika pemain diberikan topik "Albert Einstein" dan "Great Wall of China", pemain harus menemukan jalur terpendek antara kedua topik tersebut dengan mengikuti tautan-tautan dalam artikel Wikipedia. Proses ini membutuhkan keterampilan dalam pencarian dan navigasi online, serta pengetahuan umum tentang berbagai topik yang mungkin muncul dalam artikel Wikipedia.

Permainan "*Six Degrees of Wikipedia*" ini dapat menjadi cara yang menyenangkan untuk menggali pengetahuan tentang berbagai topik yang beragam dan memperluas pemahaman tentang koneksi antara konsep-konsep yang mungkin terlihat tidak terkait pada pandangan pertama.

3. Kasus 3: Permainan "*Geographical Exploration Game*"

Contoh kasus lain yang serupa dengan WikiRace adalah "*Geographical Exploration Game*". Dalam permainan ini, pemain dimulai dengan suatu lokasi geografis, seperti kota, negara, atau landmark terkenal, dan harus menavigasi melalui peta atau sumber informasi geografis lainnya untuk mencapai tujuan tertentu, seperti mencapai lokasi lain yang ditentukan sebelumnya.

Pemain mungkin diberikan tugas untuk mencari rute terpendek antara dua lokasi yang berbeda, menjelajahi tempat-tempat menarik di sepanjang jalur tersebut, atau bahkan menemukan informasi tersembunyi tentang sejarah atau budaya setiap lokasi yang mereka kunjungi. Mirip dengan WikiRace, tujuan dari permainan ini adalah untuk menyelesaikan tugas atau mencapai tujuan dengan menggunakan sumber informasi yang tersedia secara efisien dan dalam waktu secepat mungkin.

Permainan ini dapat memperluas pengetahuan geografis pemain sambil juga menguji keterampilan navigasi dan pemecahan masalah mereka. Selain itu, permainan ini dapat dimainkan dalam berbagai format, mulai dari permainan papan tradisional hingga aplikasi permainan digital yang memanfaatkan teknologi peta interaktif.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Program

Pada program kami, hasil implementasi algoritma BFS dan IDS terdapat pada folder src/Backend/BFS dan src/Backend/IDS. Di dalam folder tersebut terdapat sejumlah file yang berisi algoritma BFS dan IDS yang dibuat dengan metode biasa, dengan bantuan *multithreading*, dan dengan metode *caching*. Secara keseluruhan, berikut ini merupakan implementasi kode dari program *WikiRace Solver* kami.

4.1.1. Fungsi Main (api.go)

```
package main

import (
    "Backend/BFS"
    "Backend/IDS"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "time"

    "github.com/rs/cors"
)

type Result struct {
    Path          []string      `json:"path"`
    NumArticlesVisited int         `json:"numArticlesVisited"`
    NumArticlesChecked int        `json:"numArticlesChecked"`
    ExecutionTime   time.Duration `json:"executionTime"`
}

func main() {
    // Create a new CORS handler allowing requests from localhost:3000
    corsHandler := cors.New(cors.Options{
        AllowedOrigins: []string{"http://localhost:3000"},
        AllowCredentials: true,
    })

    // Wrap the default ServeMux with the CORS handler
    handler := corsHandler.Handler(http.DefaultServeMux)

    http.HandleFunc("/wikirace", algorithmHandler)
    fmt.Println("Server listening on port 8000...")
    log.Fatal(http.ListenAndServe(":8000", handler))
}

func algorithmHandler(w http.ResponseWriter, r *http.Request) {
    algorithmType := r.URL.Query().Get("algorithm")
    if algorithmType == "BFS" {
```

```

        BFSHandler(w, r)
    } else if algorithmType == "IDS" {
        IDSHandler(w, r)
    } else {
        http.Error(w, "Algoritma tidak valid!", http.StatusBadRequest)
    }
}

func BFSHandler(w http.ResponseWriter, r *http.Request) {
    initialPage := r.URL.Query().Get("initial")
    destinationPage := r.URL.Query().Get("destination")

    startTime := time.Now()
    path, articlesVisited, articlesChecked := BFS.CallBFS(initialPage,
destinationPage)
    finishedTime := time.Now()
    executionTime := finishedTime.Sub(startTime)

    formatExecutionTime := time.Duration(executionTime.Nanoseconds() /
int64(time.Millisecond))

    solution := Result{
        Path:                 path,
        NumArticlesVisited:  articlesVisited,
        NumArticlesChecked:  articlesChecked,
        ExecutionTime:       formatExecutionTime,
    }

    jsonResponse, err := json.Marshal(solution)
    if err != nil {
        http.Error(w, "Unable to marshal JSON response",
http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Write(jsonResponse)
}

func IDSHandler(w http.ResponseWriter, r *http.Request) {
    initialPage := r.URL.Query().Get("initial")
    destinationPage := r.URL.Query().Get("destination")

    startTime := time.Now()
    path, articlesVisited, articlesChecked := IDS.IDS(initialPage, destinationPage)
    finishedTime := time.Now()
    executionTime := finishedTime.Sub(startTime)

    formatExecutionTime := time.Duration(executionTime.Nanoseconds() /
int64(time.Millisecond))

    solution := Result{
        Path:                 path,
        NumArticlesVisited:  articlesVisited,
        NumArticlesChecked:  articlesChecked,
        ExecutionTime:       formatExecutionTime,
    }

    jsonResponse, err := json.Marshal(solution)
    if err != nil {

```

```

        http.Error(w, "Unable to marshal JSON response",
http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Write(jsonResponse)
}

```

4.1.2. Fungsi Mendapatkan Link Wikipedia BFS (BFS.go)

```

package BFS

func isInArray(slice []string, value string) bool {
    for _, item := range slice {
        if item == value {
            return true
        }
    }
    return false
}

func lastElement(array []string) string {
    return array[len(array)-1]
}

// rekursif
// semua elemen berupa arrayString yang terdiri dari semua urutan link
// perbandingan langsung dilakukan pada elemen terakhir setiap stringarray pada
// possible solution
func BFS(possibleSolutions [][]string, visited []string, desiredPageName string,
numArticlesVisited int) ([]string, int) {
    // penyimpanan possibleSolution dengan level yang baru
    var tempPossibleSolutions [][]string
    // penelusuran untuk setiap array of string pada possible solutions
    for _, stringArray := range possibleSolutions {
        // tambah visited
        numArticlesVisited++
        // sudah dikunjungi (elemen terakhir dari setiap array of string) ->
        iterasi selanjutnya
        if isInArray(visited, lastElement(stringArray)) {
            continue
        } else {
            // ketemu
            if lastElement(stringArray) == desiredPageName {
                return stringArray, numArticlesVisited
            } else {
                // daftar link dari elemen terakhir (yang akan diexpand)
                linkArray := GetWikipediaLinks(lastElement(stringArray))
                // copy dari yang mau diexpand
                copyStringArray := stringArray
                // untuk setiap link ditambahin
                for _, link := range linkArray {
                    // tambah visited
                    numArticlesVisited++
                    // tambahin link baru
                    copyStringArray = append(copyStringArray, link)
                    // pengecekan juga
                }
            }
        }
    }
}

```

```

                if lastElement(copyStringArray) == desiredPageName
{
                    return copyStringArray, numArticlesVisited
}
// dan dimasukkan dalam semua array sesuai
kebutuhan
tempPossibleSolutions =
append(tempPossibleSolutions, copyStringArray)
// pengembalian nilai jadi awal
copyStringArray = stringArray
}
visited = append(visited, lastElement(stringArray))
}
}
// pemanggilan rekursi
return BFS(tempPossibleSolutions, visited, desiredPageName, numArticlesVisited)
}

func CallBFS(initialPageName string, desiredPageName string) ([]string, int, int) {
possibleSolutions := [][]string{{initialPageName}}
var visited []string
var articlesVisited int
solution, articlesVisited := BFSMT(possibleSolutions, visited, desiredPageName,
articlesVisited)
return solution, articlesVisited, len(solution) - 1
}

```

4.1.3. Fungsi Penelusuran dengan BFS Biasa (BFSMain.go)

```

package BFS

import (
    "log"
    "net/http"
    "strings"

    "github.com/PuerkitoBio/goquery"
)

func GetWikipediaLinks(URL string) []string {
    resp, err := http.Get(URL)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("status code error: %d %s", resp.StatusCode, resp.Status)
    }

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatal(err)
    }

    var awalan = []string{
        "/wiki/Draft:",

```

```

        "/wiki/Module:",
        "/wiki/MediaWiki:",
        "/wiki/Index:",
        "/wiki/Education_Program:",
        "/wiki/TimedText:",
        "/wiki/Gadget:",
        "/wiki/Gadget_Definition:",
        "/wiki/Main_Page",
        "/wiki/Main_Page:",
        "/wiki/Special:",
        "/wiki/Talk:",
        "/wiki/User:",
        "/wiki/Portal:",
        "/wiki/Wikipedia:",
        "/wiki/File:",
        "/wiki/Category:",
        "/wiki/Help:",
        "/wiki/Template:",
    }

    links := []string{}
    // cari link
    doc.Find("a[href]").Each(func(i int, s *goquery.Selection) {
        link, _ := s.Attr("href")
        // wikipedia page
        if strings.HasPrefix(link, "/wiki/") {
            skip := false
            // awalan nggak kepakai (diskip)
            for _, prefix := range awalan {
                if strings.HasPrefix(link, prefix) {
                    skip = true
                    break
                }
            }
            if !skip {
                links = append(links, "https://en.wikipedia.org"+link)
            }
        }
    })
    return links
}

```

4.1.4. Fungsi Penelusuran dengan BFS Multithreading (BFSMT.go)

```

package BFS

import (
    "math"
    "sync"
)

func BFSMT(possibleSolutions [][]string, visited []string, desiredPageName string,
numArticlesVisited int) ([]string, int) {
    var wg sync.WaitGroup
    var mu sync.Mutex
    found := make(chan struct {
        path []string
        num   int
    })

```

```

}, len(possibleSolutions))

for _, stringArray := range possibleSolutions {
    wg.Add(1)
    go func(stringArray []string, visited []string, desiredPageName string,
numArticlesVisited int) {
        defer wg.Done()

        mu.Lock()
        numArticlesVisited++
        lastURL := lastElement(stringArray)
        if isInArray(visited, lastURL) {
            mu.Unlock()
            return
        }
        visited = append(visited, lastURL)
        mu.Unlock()

        if lastURL == desiredPageName {
            found <- struct {
                path []string
                num   int
            }{stringArray, numArticlesVisited}
            return
        }

        linkArray := GetWikipediaLinks(lastURL)
        for _, link := range linkArray {
            copyStringArray := append([]string{}, stringArray...)
            copyStringArray = append(copyStringArray, link)

            wg.Add(1)
            go func(copyStringArray []string, visited []string,
desiredPageName string, numArticlesVisited int) {
                defer wg.Done()
                BFSMT([][]string{copyStringArray}, visited,
desiredPageName, numArticlesVisited)
                }(copyStringArray, visited, desiredPageName,
numArticlesVisited)
            }
            }(stringArray, visited, desiredPageName, numArticlesVisited)
        }
    }

    go func() {
        wg.Wait()
        close(found)
    }()
}

var shortestPath []string
shortestNum := math.MaxInt64
for result := range found {
    if len(result.path) > 0 && result.num < shortestNum {
        shortestPath = result.path
        shortestNum = result.num
    }
}

return shortestPath, shortestNum
}

```

4.1.5. Fungsi Penelusuran dengan BFS Caching (BFSCache.go)

```
package BFS

import (
    "encoding/gob"
    "log"
    "net/http"
    "os"
    "strings"
    "sync"

    "github.com/PuerkitoBio/goquery"
)

var cache map[string][]string
var cacheMutex sync.Mutex

// func init() {
//     cache = make(map[string][]string)
//     loadCache()
// }

func loadCache() {
    cacheMutex.Lock()
    defer cacheMutex.Unlock()
    file, err := os.Open("/app/Backend/cache.gob")
    if err != nil {
        if os.IsNotExist(err) {
            // Cache file doesn't exist, create it
            file, err = os.Create("/app/Backend/cache.gob")
            if err != nil {
                log.Fatal("Error creating cache file:", err)
            }
            defer file.Close()
            return
        }
        log.Fatal("Error opening cache file:", err)
    }
    defer file.Close()

    fileInfo, err := file.Stat()
    if err != nil {
        log.Fatal("Error getting file information:", err)
    }

    if fileInfo.Size() == 0 {
        // Cache file is empty, no need to load anything
        return
    }

    decoder := gob.NewDecoder(file)
    if err := decoder.Decode(&cache); err != nil {
        log.Fatal("Error decoding cache:", err)
    }
}

func saveCache() {
    cacheMutex.Lock()
    defer cacheMutex.Unlock()
```

```

        file, err := os.OpenFile("/app/Backend/cache.gob",
os.O_WRONLY|os.O_CREATE|os.O_APPEND, 0644)
        if err != nil {
            log.Fatal("Error opening cache file:", err)
        }
        defer file.Close()

        encoder := gob.NewEncoder(file)
        if err := encoder.Encode(cache); err != nil {
            log.Fatal("Error encoding cache:", err)
        }
    }

func GetWikipediaLinksCache(URL string) []string {
    // cek yang mau diekspan ada di cache atau tidak
    if links, ok := cache[URL]; ok {
        return links
    }

    resp, err := http.Get(URL)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("status code error: %d %s", resp.StatusCode, resp.Status)
    }

    links := []string{}

    if resp.Request.URL.String() != URL {
        URL = resp.Request.URL.String()
        links = append(links, URL)
    }

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatal(err)
    }

    var awalan = []string{
        "/wiki/Draft:",
        "/wiki/Module:",
        "/wiki/MediaWiki:",
        "/wiki/Index:",
        "/wiki/Education_Program:",
        "/wiki/TimedText:",
        "/wiki/Gadget:",
        "/wiki/Gadget_Definition:",
        "/wiki/Main_Page",
        "/wiki/Main_Page:",
        "/wiki/Special:",
        "/wiki/Talk:",
        "/wiki/User:",
        "/wiki/Portal:",
        "/wiki/Wikipedia:",
        "/wiki/File:",
        "/wiki/Category:",
        "/wiki/Help:",
    }
}

```

```

        "/wiki/Template:",
    }
    // cari link
    doc.Find("a[href]").Each(func(i int, s *goquery.Selection) {
        link, _ := s.Attr("href")
        // wikipedia page
        if strings.HasPrefix(link, "/wiki/") {
            hasPrefix := false
            // awalan nggak kepakai (diskip)
            for _, prefix := range awalan {
                if strings.HasPrefix(link, prefix) {
                    hasPrefix = true
                    break
                }
            }
            if !hasPrefix {
                links = append(links, "https://en.wikipedia.org"+link)
            }
        }
    })
    // cache
    cache[URL] = links
    // simpan dalam data
    saveCache()

    return links
}

```

4.1.6. Fungsi Mendapatkan Wikipedia Link IDS (IDS.go)

```

package IDS

import (
    "log"
    "net/http"
    "strings"

    "github.com/PuerkitoBio/goquery"
)

func GetWikipediaLinks(URL string) []string {
    resp, err := http.Get(URL)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("status code error: %d %s", resp.StatusCode, resp.Status)
    }

    doc, err := goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        log.Fatal(err)
    }
}

```

```

var awalan = []string{
    "/wiki/Draft:",
    "/wiki/Module:",
    "/wiki/MediaWiki:",
    "/wiki/Index:",
    "/wiki/Education_Program:",
    "/wiki/TimedText:",
    "/wiki/Gadget:",
    "/wiki/Gadget_Definition:",
    "/wiki/Main_Page",
    "/wiki/Main_Page:",
    "/wiki/Special:",
    "/wiki/Talk:",
    "/wiki/User:",
    "/wiki/Portal:",
    "/wiki/Wikipedia:",
    "/wiki/File:",
    "/wiki/Category:",
    "/wiki/Help:",
    "/wiki/Template:",
}

links := []string{}
// cari link
doc.Find("a[href]").Each(func(i int, s *goquery.Selection) {
    link, _ := s.Attr("href")
    // wikipedia page
    if strings.HasPrefix(link, "/wiki/") {
        skip := false
        // awalan nggak pakai (diskip)
        for _, prefix := range awalan {
            if strings.HasPrefix(link, prefix) {
                skip = true
                break
            }
        }
        if !skip {
            links = append(links, "https://en.wikipedia.org"+link)
        }
    }
})
return links
}

```

4.1.7. Fungsi Penelusuran dengan IDS Biasa (IDSMain.go)

```

package IDS

func isInArray(slice []string, value string) bool {
    for _, item := range slice {
        if item == value {
            return true
        }
    }
    return false
}

func DLS(currentPageName string, desiredPageName string, visited []string, solution

```

```

[]string, currentDepth int, desiredDepth int, numArticlesVisited int) ([]string, int)
{
    numArticlesVisited++
    // utamain ketemu
    if currentPageName == desiredPageName {
        return solution, numArticlesVisited
    } else {
        // utamain kedalaman
        if currentDepth == desiredDepth {
            return solution, numArticlesVisited
        } else {
            // utamain sudah dikunjungi
            if isInArray(visited, currentPageName) {
                return solution, numArticlesVisited
            } else {
                // tambah daftar yang sudah dikunjungi
                visited = append(visited, currentPageName)
                // dapat link
                links := GetWikipediaLinks(currentPageName)
                for _, link := range links {
                    // tambahkan visited
                    numArticlesVisited++
                    // dibuat array baru agar terpisah dari parameter
                    solution
                    newSolution := append([]string{}, solution...)
                    newSolution = append(newSolution, link)
                    newSolution, _ = DLS(link, desiredPageName,
                    visited, newSolution, currentDepth+1, desiredDepth, numArticlesVisited)
                    if len(newSolution) > len(solution) &&
                    newSolution[len(newSolution)-1] == desiredPageName {
                        return newSolution, numArticlesVisited
                    }
                }
            }
        }
    }
}

func IDS(initialPageName string, desiredPageName string) ([]string, int, int) {
    var found bool = false
    var currentDepth int = 0
    var result []string
    var numArticlesVisited int
    var visited []string
    // Inisialisasi dan dibuat copy karena akan diubah pada DLS
    copyInitialPageName := initialPageName
    // selama belum ketemu dan kedalamannya belum maksimal
    for !found {
        // // termasuk yang awal
        // numArticlesVisited = 1
        // solution diubah setiap DLS
        solution := []string{copyInitialPageName}
        // dapat hasil
        result, numArticlesVisited = DLSMT(copyInitialPageName, desiredPageName,
        visited, solution, 0, currentDepth, numArticlesVisited)
        // cek solusi
        if len(result) != 1 {
            found = true
        }
    }
}

```

```

        }
        // tambah kedalaman maksimal
        currentDepth = currentDepth + 1
        // reset untuk iterasi berikutnya karena diubah dalam DLS
        copyInitialPageName = initialPageName
    }
    return result, numArticlesVisited, len(result) - 1
}

```

4.1.8. Fungsi Penelusuran dengan IDS Multithreading (IDSMT.go)

```

package IDS

import "sync"

func DLSMT(currentPageName string, desiredPageName string, visited []string, solution
[]string, currentDepth int, desiredDepth int, numArticlesVisited int) ([]string, int)
{
    numArticlesVisited++
    // utamain ketemu
    if currentPageName == desiredPageName {
        return solution, numArticlesVisited
    } else {
        // utamain kedalaman
        if currentDepth == desiredDepth {
            return solution, numArticlesVisited
        } else {
            // utamain sudah dikunjungi
            if isInArray(visited, currentPageName) {
                return solution, numArticlesVisited
            } else {
                // tambah daftar yang sudah dikunjungi
                visited = append(visited, currentPageName)
                // dapat link
                links := GetWikipediaLinks(currentPageName)
                var wg sync.WaitGroup
                results := make(chan struct {
                    solution      []string
                    numArticlesVisited int
                }, len(links))

                for _, link := range links {
                    wg.Add(1)
                    go func(link string) {
                        defer wg.Done()
                        // tambahin visited
                        numArticlesVisited++
                        // dibuat array baru agar terpisah dari
parameter solution
solution...)
                        newSolution := append([]string{},

newSolution = append(newSolution, link)
newSolution, _ = DLS(link, desiredPageName,
visited, newSolution, currentDepth+1, desiredDepth, numArticlesVisited)
results <- struct {
                    solution      []string
                    numArticlesVisited int
                }{newSolution, numArticlesVisited}
}

```

```

                }(link)
            }

            go func() {
                wg.Wait()
                close(results)
            }()

            for result := range results {
                if len(result.solution) > len(solution) &&
result.solution[len(result.solution)-1] == desiredPageName {
                    return result.solution,
result.numArticlesVisited
                }
            }
        }

        return solution, numArticlesVisited
    }
}

```

4.1.9. Fungsi Penelusuran dengan IDS Caching (IDSCache.go)

```

package IDS

import (
    "encoding/gob"
    "log"
    "net/http"
    "os"
    "strings"
    "sync"

    "github.com/PuerkitoBio/goquery"
)

var cache map[string][]string
var cacheMutex sync.Mutex

func init() {
    cache = make(map[string][]string)
    loadCache()
}

func loadCache() {
    cacheMutex.Lock()
    defer cacheMutex.Unlock()
    file, err := os.Open("/app/Backend/cache.gob")
    if err != nil {
        if os.IsNotExist(err) {
            // Cache file doesn't exist, create it
            file, err = os.Create("/app/Backend/cache.gob")
            if err != nil {
                log.Fatal("Error creating cache file:", err)
            }
            defer file.Close()
            return
        }
    }
}

```

```

        }
        log.Fatal("Error opening cache file:", err)
    }
    defer file.Close()

    fileInfo, err := file.Stat()
    if err != nil {
        log.Fatal("Error getting file information:", err)
    }

    if fileInfo.Size() == 0 {
        // Cache file is empty, no need to load anything
        return
    }

    decoder := gob.NewDecoder(file)
    if err := decoder.Decode(&cache); err != nil {
        log.Fatal("Error decoding cache:", err)
    }
}

func saveCache() {
    cacheMutex.Lock()
    defer cacheMutex.Unlock()
    file, err := os.OpenFile("/app/Backend/cache.gob",
os.O_WRONLY|os.O_CREATE|os.O_APPEND, 0644)
    if err != nil {
        log.Fatal("Error opening cache file:", err)
    }
    defer file.Close()

    encoder := gob.NewEncoder(file)
    if err := encoder.Encode(cache); err != nil {
        log.Fatal("Error encoding cache:", err)
    }
}

func GetWikipediaLinksCache(URL string) []string {
    // cek yang mau diekspan ada di cache atau tidak
    if links, ok := cache[URL]; ok {
        return links
    }

    resp, err := http.Get(URL)
    if err != nil {
        log.Fatal(err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != 200 {
        log.Fatalf("status code error: %d %s", resp.StatusCode, resp.Status)
    }

    links := []string{}

    if resp.Request.URL.String() != URL {
        URL = resp.Request.URL.String()
        links = append(links, URL)
    }
}

```

```

doc, err := goquery.NewDocumentFromReader(resp.Body)
if err != nil {
    log.Fatal(err)
}

var awalan = []string{
    "/wiki/Draft:",
    "/wiki/Module:",
    "/wiki/MediaWiki:",
    "/wiki/Index:",
    "/wiki/Education_Program:",
    "/wiki/TimedText:",
    "/wiki/Gadget:",
    "/wiki/Gadget_Definition:",
    "/wiki/Main_Page",
    "/wiki/Main_Page:",
    "/wiki/Special:",
    "/wiki/Talk:",
    "/wiki/User:",
    "/wiki/Portal:",
    "/wiki/Wikipedia:",
    "/wiki/File:",
    "/wiki/Category:",
    "/wiki/Help:",
    "/wiki/Template:",
}

// cari link
doc.Find("a[href]").Each(func(i int, s *goquery.Selection) {
    link, _ := s.Attr("href")
    // wikipedia page
    if strings.HasPrefix(link, "/wiki/") {
        hasPrefix := false
        // awalan nggak kepakai (diskip)
        for _, prefix := range awalan {
            if strings.HasPrefix(link, prefix) {
                hasPrefix = true
                break
            }
        }
        if !hasPrefix {
            links = append(links, "https://en.wikipedia.org"+link)
        }
    }
})

// cache
cache[URL] = links
// simpan dalam data
saveCache()

return links
}

```

4.2. Struktur Data dan Spesifikasi Teknis Program

Program ini diimplementasikan dengan bahasa pemrograman Go dan dikembangkan dengan IDE Visual Studio. Struktur data yang digunakan berbasis pada

fungsi. Di dalam program ini, telah terdefinisi beberapa *package*, antara lain *BFS*, *IDS*, dan *Main*. Semua *package* tersebut kemudian kami kembangkan dan lengkapi dengan penjelasan sebagai berikut.

4.2.1. BFS

Fungsi ini berisi strategi cara mencari *file* spesifik dari sebuah *root directory* berdasarkan algoritma BFS. Dalam pencarian menggunakan metode BFS, digunakan struktur data *list of string* untuk mencatat solusi-solusi pencarian. Selain itu, digunakan juga struktur data *queue of string* untuk mencatat *directory* yang akan ditelusuri dan *queue of node* untuk mencatat *parent* dari *node* yang akan ditelusuri.

4.2.2. IDS

Fungsi ini berisi strategi cara mencari *file* spesifik dari sebuah *root directory* berdasarkan algoritma DFS. Dalam pencarian menggunakan metode BFS, digunakan struktur data *list of string* untuk mencatat solusi-solusi pencarian.

4.2.3. Main

Fungsi ini menyediakan *handler* untuk setiap pemanggilan dari *frontend* sekaligus memanggil fungsi BFS dan IDS.

4.3. Tata Cara Penggunaan Program



Gambar 4.3.1 Tampilan Program

Tata cara penggunaan program:

1. Pilih *page* awal [1] dengan memasukkan nama *page* Wikipedia awal yang ingin dimasukkan
2. Pilih *page* akhir [2] dengan memasukkan nama *page* Wikipedia awal yang ingin dituju
3. Tekan tombol BFS [3] atau IDS [4] berdasarkan algoritma yang ingin digunakan
4. Tekan tombol *search* [5] untuk mulai pencarian dan akan menampilkan hasil ketika algoritma telah selesai berjalan

Fitur:

1. Kolom untuk memasukkan *page* awal yang akan dicari
2. Kolom untuk memasukkan *page* akhir yang ingin dituju
3. Tombol algoritma BFS
4. Tombol algoritma IDS
5. Tombol Search

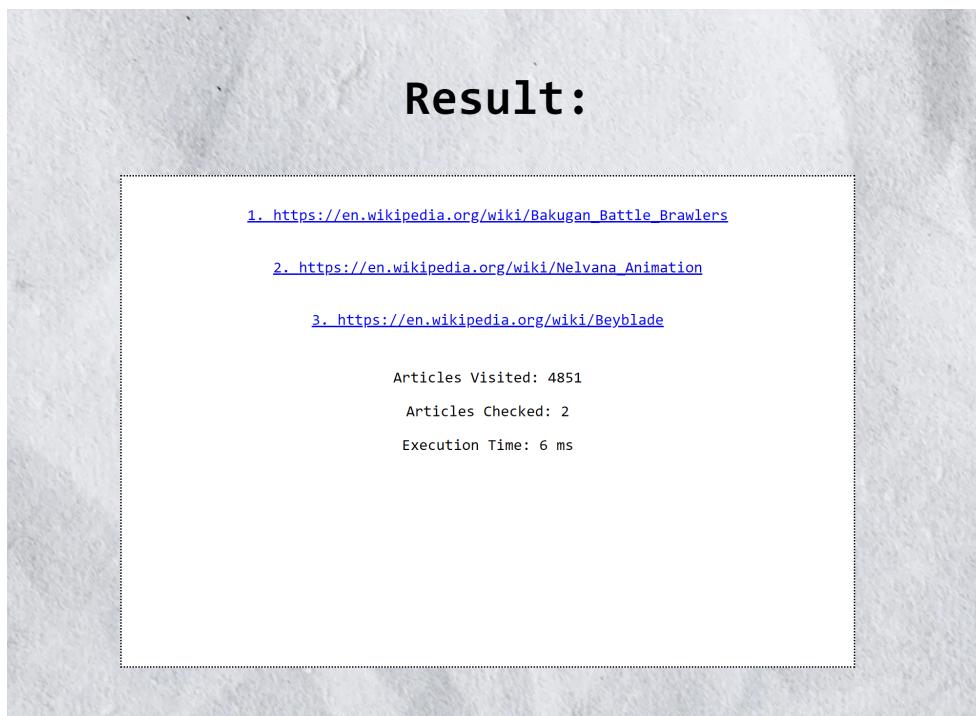
4.4. Hasil Pengujian

4.4.1. Mencari dengan Metode BFS Biasa



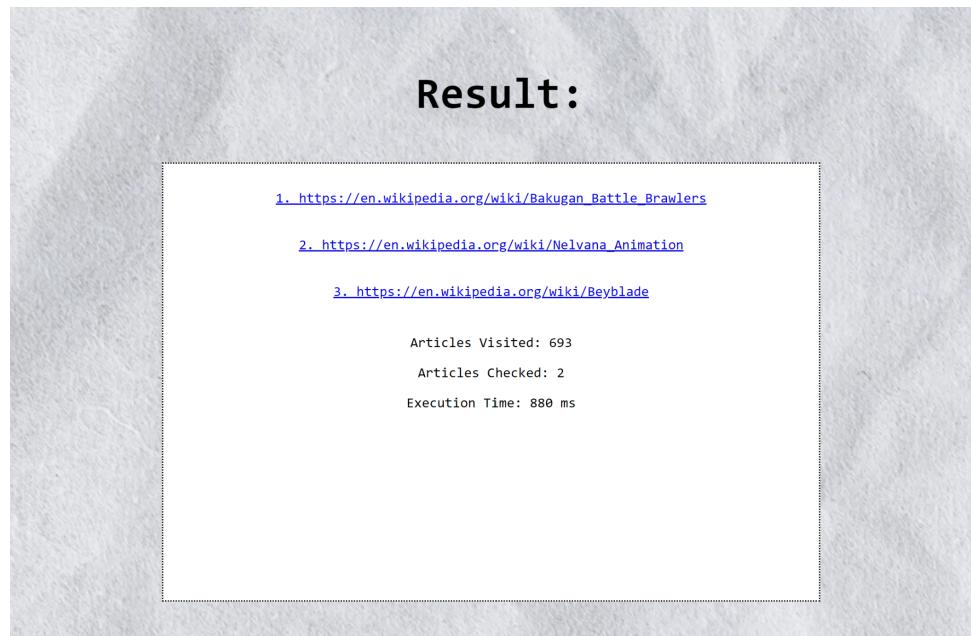
Gambar 4.4.1.1 Mencari dengan Metode BFS biasa

4.4.2. Mencari dengan Metode BFS Caching



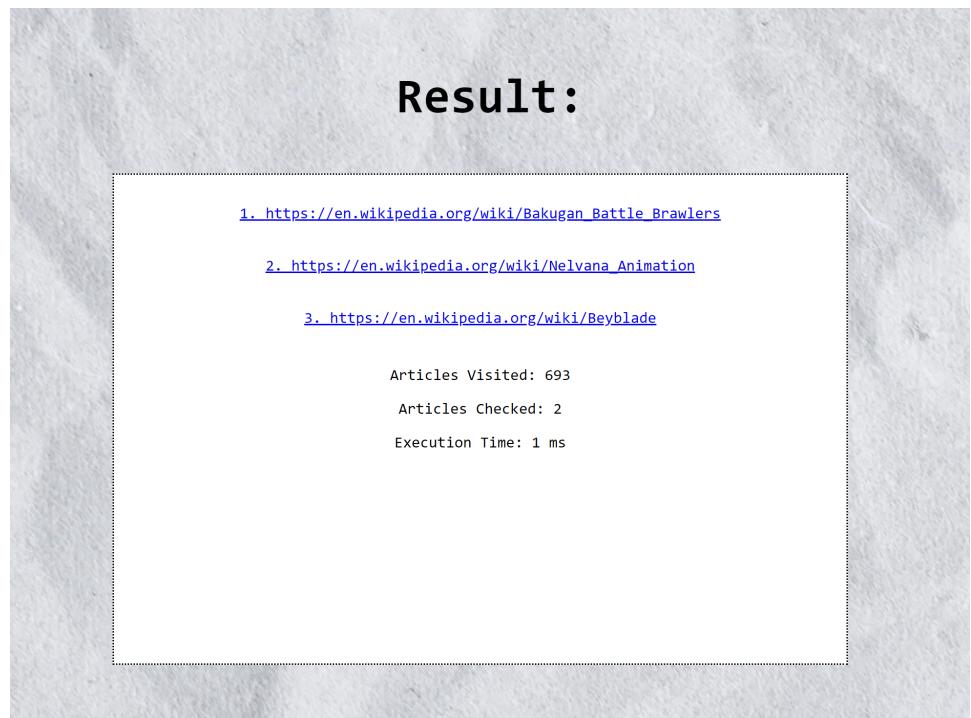
Gambar 4.4.2.1 Hasil *Testing* Mencari dengan Metode BFS Caching

4.4.3. Mencari dengan Metode IDS Biasa



Gambar 4.4.3.1 Hasil *Testing* Mencari dengan Metode IDS Biasa

4.4.4. Mencari dengan Metode IDS Caching



Gambar 4.4.4.1 Hasil *Testing* Mencari dengan Metode IDS Caching

4.4.5. Mencari *page* awal yang Tidak Ada dengan Metode BFS

```
backend-1 | 2024/04/27 06:38:10 status code error: 404 404 Not Found
```

Gambar 4.4.5.1 Hasil *Testing* Mencari *File* yang Tidak Ada dengan Metode BFS

4.5. Analisis Hasil Pengujian

Berdasarkan hasil pengujian program yang kami buat, pencarian *page* dengan algoritma IDS cenderung lebih cepat eksekusinya dibandingkan dengan algoritma BFS. Pada gambar 4.4.3.1 pencarian dengan DFS membutuhkan waktu 880 ms sedangkan pada gambar 4.4.2.1 pencarian *file* dengan konfigurasi yang sama menggunakan BFS membutuhkan waktu 8873 ms. Perbedaan durasi tersebut terjadi karena BFS membutuhkan pemrosesan yang lebih rumit baik pada proses pencarian maupun penunjukan pohon seperti penggunaan *queue* dan penyimpanan *node*.

Dapat dilihat juga pada contoh 4.2 dan 4.4, ketika dibuat dengan metode *caching* setelah melakukan pencarian sebelumnya, dapat dilihat bahwa hasil penjelajahan jauh lebih cepat dibandingkan dengan metode biasa. Ini juga menjadi sangat bermanfaat untuk pencarian yang dilakukan berulang kali sehingga langsung mengambil data yang sudah pernah dijelajahi dan membuat waktu pencarian semakin singkat.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Telah berhasil diimplementasikan sebuah program berupa *WikiRace Solver* yang dirancang dan dikembangkan untuk memenangkan pertandingan *WikiRace*, yakni dengan melakukan penelusuran terhadap artikel-artikel 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. Hal ini sesuai dengan yang diminta dalam spesifikasi Tugas Besar 2 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024. Hal mengenai program *WikiRace Solver* yang berhasil diimplementasikan dalam program ini meliputi:

1. Konsep algoritma BFS dan IDS, serta penerapannya dalam implementasi program *WikiRace Solver*,
2. Analisis desain solusi algoritma BFS dan IDS, serta perbandingan antar tingkat efektivitas program dari masing-masing algoritma,
3. Implementasi aplikasi berbasis web, lengkap dengan integrasi *frontend* dan *backend*,
4. Implementasi program dengan paradigma pemrograman fungsional dengan bahasa Go.
5. Implementasi *Docker* untuk menjalankan *frontend* dan *backend*.

Semua implementasi dari konsep-konsep di atas kemudian berhasil digunakan untuk menyelesaikan seluruh fitur yang ada di dalam spesifikasi. Fitur-fitur tersebut telah terdapat pada program *WikiRace Solver* yang kami buat. Setidaknya terdapat 4 fitur utama dan 2 fitur bonus yang dapat digunakan pada web kami, antara lain:

1. Program dapat menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan,
2. Program dapat memilih algoritma yang digunakan melalui input dari pengguna,

3. 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),
4. Program setidaknya dapat mengeluarkan salah satu rute terpendek yang kurang dari 5 menit di setiap permainan.
5. Program dapat mencari rute terpendek dengan durasi kurang dari satu menit.
6. Program dapat dijalankan menggunakan *Docker* baik untuk frontend maupun backend.

Fitur ini dapat digunakan pada program kami yang sudah diciptakan sedemikian rupa dengan memanfaatkan tampilan yang *user-friendly* dan menarik untuk dapat mempermudah interaksi antara *user* dengan aplikasi.

Dengan demikian, kelompok menyimpulkan bahwa dengan mengerjakan Tugas Besar 2 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024 ini, dapat diketahui bahwa untuk menyelesaikan suatu masalah yang mungkin ditemukan dalam kehidupan sehari-hari, dalam hal ini misalnya, melakukan penelusuran terhadap artikel-artikel 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, dapat diimplementasikan program *WikiRace Solver* sebagai bentuk penerapan dari konsep algoritma BFS dan IDS yang telah dipelajari pada kuliah IF2211.

5.2. Saran dan Refleksi

Tugas Besar 2 IF2211 Strategi Algoritma Semester 2 Tahun 2023/2024 menjadi salah satu proses pembelajaran bagi kelompok dalam menerapkan ilmu-ilmu yang diajarkan pada kuliah maupun melakukan eksplorasi materi secara mandiri. Berikut ini merupakan sejumlah saran dari kelompok untuk pihak-pihak yang ingin melakukan atau mengerjakan hal serupa.

1. Program yang diminta adalah program dengan menggunakan bahasa pemrograman Go, yakni salah satu bahasa pemrograman yang belum dikuasai secara menyeluruh oleh ketiga anggota kelompok yang terlibat dalam penggerjaan tugas besar ini. Dengan demikian, kelompok merekomendasikan agar disediakan

waktu yang cukup untuk melakukan eksplorasi terkait bahasa pemrograman yang digunakan sebelum mengimplementasikannya ke dalam sebuah program. Hal ini akan meningkatkan efektivitas kerja tim dalam pembuatan suatu program. Di samping itu, perlu dipertimbangkan pula waktu yang dimiliki untuk melakukan eksplorasi terhadap suatu bahasa pemrograman atau *framework* tertentu sehingga tidak membebani *programmer* dalam penggerjaan proyek dengan jangka waktu yang singkat. Gunakan kemampuan berpikir serta bekerja yang elaboratif dan koordinatif di antara seluruh anggota kelompok yang terlibat.

2. Modularitas menjadi hal yang krusial dalam menciptakan suatu program secara efektif dan efisien. Dalam jangka waktu yang singkat, pemrograman secara modular dapat membantu *programmer* untuk memudahkan proses pencarian kesalahan/*error* serta *debugging*. Pada dasarnya, memrogram secara modular berarti memecah-mecah program menjadi modul-modul kecil di mana masing-masing modul berinteraksi melalui antarmuka modul. Masalah yang awalnya kompleks dapat dibagi menjadi bagian-bagian kecil yang lebih sederhana dan dapat diselesaikan dalam lingkup yang lebih kecil. Alhasil, apabila terdapat *error/bug* pada program, kesalahan dapat dengan mudah ditemukan karena alur logika yang jelas serta dapat dilokalisasi dalam satu modul. Lebih dari pada itu, modifikasi program dapat dilakukan tanpa mengganggu *body* program secara keseluruhan. Oleh karena itu, kelompok sangat menyarankan untuk melakukan pemrograman secara modular dalam mengimplementasikan algoritma BFS dan IDS dalam pembuatan program *WikiRace Solver* ini.
3. Penting bagi kelompok untuk memiliki strategi serta distribusi tugas yang baik. Ketika membuat program dalam sebuah tim, kesamaan cara menulis kode serta kemampuan untuk menulis komentar menjadi hal yang sangat penting. Hal ini diperlukan agar memudahkan anggota kelompok dalam menyatukan dan melanjutkan sebuah program. Kemampuan tersebut tentunya didukung juga dengan adanya *version control system* yang baik yang dapat digunakan oleh *programmer* dalam membuat sebuah program secara bersama-sama. Untuk itu, kami sangat menyarankan ‘GitHub’ untuk digunakan sebagai *version control*

system dalam penggerjaan tugas-tugas besar pada mata kuliah IF2211 ini, maupun pada pembuatan program dan penggerjaan proyek yang lainnya.

4. Kelompok menyadari bahwa pada implementasi program yang telah kami buat, masih banyak aspek yang dapat dikembangkan lebih lagi. Salah satunya ialah dengan mengoptimalkan algoritma BFS dan IDS yang digunakan agar proses penelusuran artikel-artikel yang dituju dapat berlangsung lebih cepat. Program juga dapat dikembangkan dari sisi UI/UX yang telah diimplementasikan oleh kelompok sehingga dapat meningkatkan kepuasan pengguna dari sisi *experience*. Hal ini tentu menjadi ruang untuk *programmer* agar dapat melakukan improvisasi terhadap implementasi dan pengembangan program *WikiRace Solver*, terutama dalam hal eksplorasi algoritma dan desain UI/UX. Selain itu, kelompok juga merekomendasikan untuk menggunakan sebuah IDE buatan *Microsoft* yang bernama *Visual Studio* untuk mempermudah proses pengembangan dengan bahasa Go.

LAMPIRAN

Link *repository* GitHub:

https://github.com/ZachS17/Tubes2_GasTubes

Link *video*:

<https://drive.google.com/file/d/1lyifHGZeM-cIhSp7y54zkE5Ux6e0bXty/view?usp=sharing>

DAFTAR PUSTAKA

Munir, Rinaldi. (2024). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1).

Institut Teknologi Bandung.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>. Diakses pada 20 April 2024.

Munir, Rinaldi. (2021). Breadth First Search (BFS) dan Depth First Search (DFS)(Bagian 2).

Institut Teknologi Bandung.

<http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>. Diakses pada 20 April 2024.