

LAPORAN TUGAS BESAR 2 IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN

WIKIRACE



Disusun Oleh :

Kelompok ThoriqGanteng

1. Muhammad Fatihul Irhab / 13522143
2. Muhammad Roihan / 13522152
3. Rafif Ardhinto Ichwantoro / 13522159

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

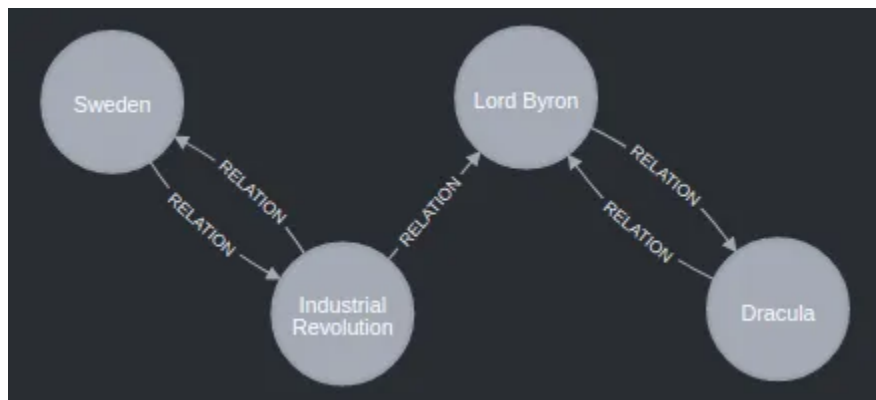
2023/2024

BAB 1

Deskripsi Tugas

1.1 Deskripsi Tugas

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



Gambar 1. Ilustrasi Graf WikiRace

(Sumber: https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZslicJCWQ.png)

1.2 Spesifikasi Program

Buatlah program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace. Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan. Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms). Program juga diminta untuk menampilkan rute terpendek yang didapatkan. Program berbasis web, sehingga perlu dibuat front-end dan back-end (tidak perlu di-deploy).

BAB 2

Landasan Teori

2.1 Dasar Teori

2.1.1 Penjelajahan Graf

Penjelajahan graf adalah suatu metode atau algoritma yang digunakan untuk secara sistematis mengunjungi setiap simpul dan sisi dari sebuah graf. Dalam konteks ini, graf merupakan struktur data yang terdiri dari simpul (nodes) yang saling terhubung oleh sisi (edges) yang menunjukkan relasi antar simpul. Tujuan dari penjelajahan graf adalah untuk mengakses atau memanipulasi setiap simpul dalam graf dengan cara yang efisien dan terorganisir, tanpa mengulang kunjungan ke simpul yang sama atau melewati simpul yang belum dikunjungi. Hal ini dapat dilakukan dengan dua metode utama, yaitu Breadth-First Search (BFS) dan Depth-First Search (DFS) maupun Iterative-Deepening Search (IDS).

2.1.2 IDS

IDS (Iterative Deepening Search) adalah sebuah strategi pencarian yang digunakan dalam algoritma traversal, khususnya pada graf, untuk mencari solusi atau jalur yang optimal dengan membatasi kedalaman pencarian. IDS merupakan gabungan dari dua teknik pencarian, yaitu Depth-First Search (DFS) dan pencarian berulang (iterative). Proses IDS dimulai dengan melakukan DFS pada graf dengan kedalaman pencarian yang bertambah secara bertahap. Pada setiap iterasi, IDS akan mulai pencarian dari simpul awal (start node) dengan kedalaman pencarian tertentu. Jika solusi tidak ditemukan pada kedalaman tersebut, maka kedalaman pencarian akan ditingkatkan dan proses pencarian akan diulang.

2.1.3 BFS

BFS (Breadth-First Search) adalah salah satu algoritma traversal yang digunakan dalam graf untuk mengunjungi setiap simpul secara sistematis, dimulai dari simpul awal (start node) dan menyebar secara merata ke simpul-simpul yang berada pada tingkat (level) yang sama sebelum melanjutkan ke tingkat berikutnya. Proses BFS dimulai dengan mengunjungi simpul awal, kemudian mengunjungi semua simpul yang terhubung langsung dengan simpul awal (simpul tetangga). Setelah itu, BFS mengunjungi semua simpul yang terhubung langsung dengan simpul-simpul yang telah dikunjungi sebelumnya, dan begitu seterusnya. Dalam proses ini, simpul-simpul yang berada pada level yang sama akan dieksplorasi terlebih dahulu sebelum mengeksplorasi simpul-simpul yang berada pada level yang lebih dalam.

2.2 Aplikasi Web

Aplikasi web adalah program komputer yang dirancang untuk dijalankan di dalam web browser. Aplikasi web memungkinkan pengguna untuk melakukan berbagai tugas dan interaksi secara online, mulai dari pengolahan data, komunikasi, hiburan, hingga transaksi bisnis. Aplikasi web beroperasi di atas arsitektur client-server, di mana server menyediakan konten dan layanan, sementara browser digunakan oleh pengguna untuk mengakses dan berinteraksi dengan aplikasi tersebut. Aplikasi web terdiri dari beberapa bagian yaitu: HTML digunakan untuk membuat struktur dasar halaman web dengan menambahkan teks, gambar, dan tautan. CSS digunakan untuk mengatur tampilan halaman seperti warna, ukuran, dan layout agar terlihat menarik. JavaScript digunakan untuk menambahkan interaktivitas seperti validasi formulir, animasi, dan komunikasi dengan server untuk mengambil atau mengirim data. Salah satu framework yang dapat digunakan adalah tailwind CSS, tailwind CSS adalah sebuah framework CSS yang dirancang untuk mempercepat dan menyederhanakan proses styling dalam pengembangan aplikasi web. Berbeda dengan framework CSS tradisional yang menggunakan pendekatan pre-defined classes dengan gaya yang sudah ditentukan sebelumnya, Tailwind menggunakan pendekatan utility-first, di mana pengembang dapat langsung menerapkan gaya dan properti CSS dengan menggunakan kelas-kelas yang telah ditentukan sebelumnya. Dalam backend dari aplikasi web, kita dapat menggunakan salah satu bahasa yaitu Go, bahasa ini memiliki kemampuan concurrency yang kuat dengan goroutines dan channels, memungkinkan backend untuk menangani banyak tugas secara bersamaan dengan performa yang optimal. Selain itu, Go dilengkapi dengan standard library yang kaya fitur, seperti net/http untuk membuat server HTTP, database/sql untuk interaksi dengan database, dan encoding/json untuk manipulasi data JSON. Pengembang juga dapat dengan mudah membuat server HTTP menggunakan paket net/http dengan fitur routing, middleware, dan handling request/response yang lengkap.

BAB 3

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Dalam menyelesaikan permasalahan ini, telah didekomposisi beberapa permasalahan sebagai langkah - langkah berikut

1. Memahami dan mempelajari struktur data yang akan digunakan untuk implementasi penyelesaian persoalan.
2. Memahami konsep dari algoritma traversal BFS dan IDS.
3. Mempelajari *syntax* bahasa pemrograman go.
4. Membuat web untuk permainan wikirace.
5. Memetakan setiap input menjadi elemen - elemen pada BFS dan IDS.
6. Mengimplementasikan algoritma BFS dan IDS pada pencarian rute pada peta persoalan
7. Menghubungkan seluruh program dengan web yang sudah dibuat.

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

3.2.1 IDS

Pada kasus IDS, kami menggunakan struktur data buatan yang berperilaku seperti node pada graf. Berikut elemen - elemen yang ada pada struktur data tersebut.

Tabel 3.2.1 Elemen pada IDS

PageURL	URL dari halaman wikipedia
Children	Node link - link yang berada pada PageURL
Parent	Node link asal dari PageURL

3.2.2 BFS

Mapping elemen-elemen dari algoritma yang akan dibuat berdasarkan pemecahan masalah tersebut adalah :

Tabel 3.2.2 Elemen pada BFS

queueLinks	Berisi link-link yang akan dikunjungi
visitedLinks	Berisi link-link yang telah dikunjungi
graph	Berisi link-link yang berperan sebagai simpul dalam graf
start	Berisi link awal

final	Berisi link final/akhir
-------	-------------------------

3.3 Fitur Fungsional Dan Arsitektur Aplikasi Web Yang Dibangun

3.3.1 Arsitektur Aplikasi Web

```

.
├── query/
│   ├── BFS.go
│   └── IDS.go
├── web/
│   ├── public
│   └── src/
│       ├── assets/
│       │   └── thoriq.gif
│       ├── components/
│       │   ├── ResultBox.js
│       │   ├── searchBar.css
│       │   ├── searchBar.jsx
│       │   ├── thoriq.gif
│       │   ├── toggle.jsx
│       │   └── toggle2.js
│       ├── App.css
│       ├── App.js
│       ├── App.test.js
│       ├── index.css
│       ├── index.jsx
│       ├── reportWebVitals.js
│       ├── setupProxy.js
│       ├── setupTests.js
│       └── toggle.js
│   ├── .gitignore
│   ├── README.md
│   ├── package-lock.json
│   ├── package.json
│   └── tailwind.config.js
├── go.mod
├── go.sum
├── main.go
└── package.json

```

3.3.2 Fungsi Fungsionalitas

Berikut beberapa fungsionalitas yang kami buat.

1. Pemilihan Algoritma Pencarian

Pengguna dapat memilih algoritma pencarian yaitu BFS atau IDS.

2. Pemilihan Cara Pencarian

Pengguna dapat memilih cara pencarian yaitu first dan all. First akan menghasilkan rute yang pertama kali didapatkan. Sedangkan all akan menghasilkan seluruh rute terpendek.

3. Pencarian Path

Pengguna dapat melakukan pencarian rute dari suatu artikel ke artikel lain.

Pertama - tama pengguna memilih algoritma pencarian (BFS/IDS). Selanjutnya pengguna memilih cara pencarian (first / all). Setelah itu pengguna memasukkan judul artikel awal dan judul artikel tujuan. Untuk memulai pencarian pengguna harus mengklik tombol submit.

3.4 Contoh Ilustrasi Kasus

3.4.1 Algoritma BFS

Contoh ilustrasi yang digunakan untuk kasus kali ini adalah kita akan melakukan pencarian dari link awal yaitu Cat menuju link akhir yaitu Medan (pada contoh ilustrasi ini, tidak semua proses pencarian ditampilkan untuk menghemat tempat dan membantu proses ilustrasi lebih mudah, tetapi hal ini tidak menghilangkan hal penting dari proses ini). Mula mula kita akan memasukkan link awal ke dalam queueLinks, yang nantinya setiap queueLinks akan dilakukan proses pencarian.

Link yang telah dikunjungi	Link yang akan dilakukan pencarian
Belum ada	Cat
Cat	Species, Felidae, Near east, Mammal, Javan mongoose, United States,

Setelah itu akan dilakukan kembali pencarian berikutnya hingga terus menerus hingga mendapatkan link akhir yaitu Medan.

Link yang telah dikunjungi	Link yang akan dilakukan pencarian
Cat	Species, Felidae, Mammal, Javan mongoose, United States, Near east,

Cat, Species	Felidae, Mammal, Javan mongoose, United States, Near east,, DNA, Fossil,
Cat, Species, Felidae	Mammal, Javan mongoose, United States, Near east,, DNA, Fossil,, Predators, Claws,
Cat, Species, Felidae, Mammal	Javan mongoose, United States, Near east,, DNA, Fossil,, Predators, Claws,, Hair, Bats,
Cat, Species, Felidae, Mammal, Javan mongoose	United States, Near east,, DNA, Fossil,, Predators, Claws,, Hair, Bats,, Urva, Medan ,

Karena link akhir telah ditemukan maka proses pencarian akan berhenti, algoritma akan mengembalikan path dari link awal yaitu Cat menuju link akhir yaitu Medan berupa graf, yang nantinya akan diolah oleh fungsi GetAllPaths() untuk mendapatkan path tersebut. Dengan demikian urutan pencarian menggunakan algoritma BFS adalah: Cat -> Species -> Felidae -> Mammal -> Javan mongoose.

3.4.2 Algoritma IDS

Pada kasus yang sama berikut urutan penyelesaian dengan menggunakan algoritma IDS.

Depth = 0: Cat : cutoff

Depth = 1 : Cat -> **Species, Felidae, Javan_mongoose** -> Species : cutoff, Felidae : cutoff, Javan_mongoose : cutoff

Depth = 2 : Cat -> Species, Felidae, Javan_mongoose -> **Biodiversity**, Felidae, Javan_mongoose -> Biodiversity : cutoff -> **Claw, Fur**, Javan_mongoose -> Claw : cutoff, Fur : cutoff -> **Urva**, Medan, Wildlife_trade -> Urva : cutoff -> **Medan**, Wildlife_trade -> Medan

STOP : Medan = goal , path : Cat -> Javan_mongoose -> Medan

BAB 4

Implementasi dan Pengujian

4.1 Implementasi Program

4.1.1 BFS

```
1  package query
2
3  import (
4      "fmt"
5      "io/ioutil"
6      "net/http"
7      "regexp"
8      "strings"
9      "time"
10 )
11
12 // Graph adalah tipe data yang digunakan untuk menyimpan berbagai link
13 type Graph struct {
14     adjacencyList map[string][]string
15     visited        map[string]bool
16 }
17
18 // Fungsi untuk membuat graf baru
19 func NewGraph() *Graph {
20     return &Graph{
21         adjacencyList: make(map[string][]string),
22         visited:        make(map[string]bool),
23     }
24 }
25
26 // Fungsi untuk menambahkan sisi dalam graf dimana meminta parameter parent link dan child link
27 func (g *Graph) AddEdge(src, dest string) {
28     g.adjacencyList[src] = append(g.adjacencyList[src], dest)
29 }
30
31 // Fungsi untuk mendapatkan kedalaman maksimal dalam graf
32 func (g *Graph) maxDepth(node string) int {
33     g.visited = make(map[string]bool)
34     return g.searchMax(node)
35 }
```

```

37 // Fungsi untuk mencari kedalaman maksimal graf
38 func (g *Graph) searchMax(node string) int {
39     g.visited[node] = true
40     maxDepth := 0
41     for _, neighbor := range g.adjacencyList[node] {
42         if !g.visited[neighbor] {
43             depth := g.searchMax(neighbor)
44             if depth > maxDepth {
45                 maxDepth = depth
46             }
47         }
48     }
49     return 1 + maxDepth
50 }
51
52 // Fungsi untuk mendapatkan semua path yang terhubung dari start hingga final
53 func GetAllPaths(graph *Graph, start string, final string, visited map[string]bool, path []string, allPaths [][]string) {
54     visited[start] = true
55     path = append(path, start)
56     if start == final {
57         *allPaths = append(*allPaths, append([]string{}, path...))
58     } else {
59         for _, neighbor := range graph.adjacencyList[start] {
60             if !visited[neighbor] {
61                 GetAllPaths(graph, neighbor, final, visited, path, allPaths)
62             }
63         }
64     }
65     path = path[:len(path)-1]
66     visited[start] = false
67 }
68
69 // Fungsi untuk mendapatkan path terpendek dari start hingga final dengan menggunakan algoritma BFS
70 func Bfs2(queueLinks []string, visitedLink map[string]bool, graph *Graph, start string, final string, choice bool) *Graph {
71     // Menginisiasi kedalaman terpendek dan memulai waktu pencarian
72     shortDepth := 999999
73     timeoutSeconds := 290
74     timeout := time.Duration(timeoutSeconds) * time.Second
75     startTime := time.Now()
76
77     // Melakukan pencarian berdasarkan dengan queueLinks
78     for len(queueLinks) > 0 {
79
80         // Jika waktu sudah melewati batas waktu tertentu maka akan mengembalikan hasil yang sudah ada
81         if time.Since(startTime) > timeout {
82             return graph
83         }
84
85         // Melakukan pengambilan seluruh hyperlink yang ada dalam suatu page
86         currentLink := queueLinks[0]
87         queueLinks = queueLinks[1:]
88         links2, query2, graph2, found2 := getLinks(currentLink, visitedLink, graph, final)
89         queueLinks = append(queueLinks, links2...)
90         visitedLink = query2
91         graph = graph2
92
93         // Melakukan pengecekan apakah kedalaman sekarang sudah melewati kedalaman terpendek
94         currentDepth := graph.maxDepth(start)
95         if currentDepth > shortDepth {
96             break
97         }
98
99         // Jika link final ketemu, maka kedalaman sekarang akan dijadikan sebagai kedalaman terpendek
100        if found2 == true {
101            shortDepth = graph.maxDepth(start)
102            if choice == true {
103                break // kondisi jika diminta hanya first path
104            }
105        }
106    }
107    return graph
108 }

```

```

106 // Fungsi untuk mendapatkan berbagai hyperlink yang ada di suatu page
107 func getLinks(html string, visitedLink map[string]bool, graph *Graph, final string) ([]string, map[string]bool, *Graph, bool) {
108     client := &http.Client{
109         Timeout: 30 * time.Second,
110     }
111     resp, err := client.Get(html)
112     if err != nil {
113         fmt.Println("Error fetching the URL:", err)
114     }
115     defer resp.Body.Close()
116     body, err := ioutil.ReadAll(resp.Body)
117     if err != nil {
118         fmt.Println("Error reading response body:", err)
119     }
120
121     re := regexp.MustCompile(`<a[>]*href="([^"]*)"[>]*`)
122     matches := re.FindAllStringSubmatch(string(body), -1)
123     uniqueLinks := make(map[string]bool)
124     for _, match := range matches {
125         if len(match) > 1 {
126             link := match[1]
127             if validLink(link) && !uniqueLinks[link] && !visitedLink[link] {
128                 uniqueLinks[link] = true
129             }
130         }
131     }
132     var newLinks []string
133     var linkFound []string
134     for link := range uniqueLinks {
135         link = "https://en.wikipedia.org" + link
136         if link == final {
137             linkFound = append(linkFound, link)
138             visitedLink[link] = true
139             graph.AddEdge(html, link)
140             return linkFound, visitedLink, graph, true
141         }

```

```

143
144     if visitedLink[link] == false {
145         newLinks = append(newLinks, link)
146         visitedLink[link] = true
147         graph.AddEdge(html, link)
148     }
149 }
150 return newLinks, visitedLink, graph, false
151 }
152
153 // Fungsi untuk mengecek apakah link tersebut termasuk link yang valid atau tidak
154 func validLink(link string) bool {
155     matched, _ := regexp.MatchString(`^/wiki/[A-Z][^(),:;%#]*$`, link)
156     return matched && !strings.Contains(link, ".")
157 }
158

```

4.1.2 IDS

```
package query
```

```
import (
    "fmt"
    "io"
    "net/http"

```

```

    "strings"
    "sync"
    "time"

    "github.com/PuerkitoBio/goquery"
)

// Node adalah tipe data untuk node dalam struktur data graph
type Node struct {
    PageURL string // URL dari halaman Wikipedia
    Children []*Node // Anak-anak dari node
    Parent *Node // Parent dari node
}

// Fungsi untuk mendapatkan path menggunakan algoritma IDS
func GetPathIDS(start, goal string, allPaths *[][]string, method string){
    startURL := "https://en.wikipedia.org/wiki/" + start
    goalURL := "https://en.wikipedia.org/wiki/" + goal
    fmt.Println("Start URL:", startURL)
    root := &Node{PageURL: startURL} // Membuat node root dengan startURL
    sebagai PageURL

    paths, found := IDS(startURL, goalURL, root, method)

    if !found{
        fmt.Println("shortest path not found") // Kondisi jika path tidak
        ditemukan
    }
    if(method == "FIRST"){
        getFirstPathIDS(paths, allPaths)
    }else if(method == "ALL"){
        getAllPathIDS(paths, allPaths)
    }
}

// Fungsi untuk mencetak seluruh path
func PrintAllPathIDS(paths [][]string){
    fmt.Println("Number of Paths:", len(paths))
    for i := range paths{
        fmt.Println("Path ke : ", i+1)
        for j := range paths[i] {
            fmt.Println(paths[i][j]) // Mencetak URL dari setiap node dalam
            jalur terpendek yang ditemukan
        }
    }
}

```

```

    }
}

// Fungsi untuk mendapatkan seluruh path
func getAllPathIDS(paths [][]*Node, allPaths *[][]string){
    for i := range paths{
        path := []string{}
        for _, node := range paths[i] {
            prefix := "https://en.wikipedia.org/wiki/"
            cleanURL := strings.TrimPrefix(node.PageURL, prefix)
            path = append(path, cleanURL)
        }
        *allPaths = append(*allPaths, path)
    }
}

// Fungsi untuk mendapatkan path pertama
func getFirstPathIDS(paths [][]*Node, allPaths *[][]string){
    path := []string{}
    for _, node := range paths[0] {
        prefix := "https://en.wikipedia.org/wiki/"
        cleanURL := strings.TrimPrefix(node.PageURL, prefix)
        path = append(path, cleanURL)
    }
    *allPaths = append(*allPaths, path)
}

// Fungsi untuk mendapatkan path dari node leaf
func getPath(leaf *Node) []*Node {
    path := []*Node{leaf} // Mulai dengan node leaf sebagai bagian dari jalur
    current := leaf

    for current.Parent != nil { // Selama node saat ini memiliki parent
        parent := current.Parent // Dapatkan parent dari node saat ini
        path = append([]*Node{parent}, path...) // Masukkan parent ke dalam
        // jalur di depan
        current = parent // Pindah ke parent node
    }

    return path
}

var wg sync.WaitGroup

```

```

var pathsAns = [][]*Node{}
var cnt int

var tOutSeconds int
var tOut time.Duration
var startT time.Time

// Fungsi DLS
func DLS(limit int,goalURL string,mxLimit int, parent *Node,method string) {
    defer wg.Done()
    cnt++ // Mengitung jumlah link yang di cek
    fmt.Println("cnt : ",cnt)

    if (method == "FIRST"){
        if len(pathsAns) != 0 { // Jika method adalah FIRST dan path
            ditemukan, keluar dari fungsi
            return
        }
    }

    if time.Since(startT) > tOut {
        return // Jika waktu habis keluar dari fungsi
    }

    if parent.PageURL == goalURL {
        pathsAns = append(pathsAns, getPath(parent)) // Jika goalURL ditemukan
        tambahkan ke pathsAns
        if (method == "FIRST"){
            return
        }
    }

    if(limit <=0){
        return // Jika sudah melebihi depth keluar dari fungsi
    }

    links, err := GetLinks(parent.PageURL) // Ambil seluruh link dari page url
    if err != nil {
        fmt.Println("Error fetching links:", err)
        return
    }
    var mxGo int
    if(mxLimit <=2){
        mxGo = 50 // Depth <= 2 maksimal goroutine adalah 50
    }else if (mxLimit == 3){

```

```

    mxGo = 25 // Depth = 3 maksimal goroutine adalah 25
} else if (mxLimit == 4){
    mxGo = 10 // Depth = 4 maksimal goroutine adalah 10
} else{
    mxGo = 5 // Depth > 4 maksimal goroutine adalah 50
}
goCnt := 0 // counter untuk menghitung banyak goroutine yang sedang
berjalan

for _, link := range links { // Iterasi seluruh link yang didapatkan
    child := &Node{PageURL: link, Parent: parent}
    parent.Children = append(parent.Children, child) // Tambahkan Node link
    ke parent
    currentLimit := limit-1
    wg.Add(1)

    var wg2 sync.WaitGroup
    wg2.Add(1)
    goCnt++

    go func(){
        defer wg2.Done()
        DLS(currentLimit, goalURL, mxLimit, child, method) // Panggil DLS
        untuk kedalaman selanjutnya
    }()
    if goCnt >= mxGo{
        wg2.Wait()
        goCnt = 0;
    }
    if (method == "FIRST"){
        if len(pathsAns) != 0 {
            return
        }
    }
}
}

// Fungsi IDS
func IDS(startURL, goalURL string, parent *Node, method string) ([][]*Node,
bool) {
    tOutSeconds = 290 // maksimum time out 290 detik
    pathsAns = [][]*Node{}
    cnt = 0
    tOut = time.Duration(tOutSeconds) * time.Second
    startT = time.Now()

```

```

    for depth := 0; depth <= 6; depth++ { // Iterasi depth mulai dari 0
sampai 6
        if time.Since(startT) > tOut {
            if len(pathsAns) != 0 {
                return pathsAns,true // Jika waktu sudah habis kembalikan
pathsAns
            }else{
                return pathsAns,false
            }
        }
        fmt.Println("Depth:", depth)

        wg.Add(1)
        cnt ++
        DLS(depth,goalURL,depth,parent,method) // Panggil DLS dengan parent
adalah root
        wg.Wait()
        if len(pathsAns) != 0 {

            return pathsAns,true
        }
    }
    return pathsAns,false
}

var (
    cache    = make(map[string][]string)
    cacheMux = sync.RWMutex{}
)

// getLinks mengambil tautan-tautan dari halaman Wikipedia
func GetLinks(pageURL string) ([]string, error) {
    cacheMux.RLock()
    if doc, found := cache[pageURL]; found {
        cacheMux.RUnlock() // Cek apakah pageURL telah tersimpan di cache
        return doc, nil
    }
    cacheMux.RUnlock()

    // Membuat HTTP request untuk halaman URL yang diberikan
    client := &http.Client{}
    req, err := http.NewRequest("GET", pageURL, nil)
    if err != nil {
        return nil, err
    }

```



```

}
req.Header.Set("User-Agent", "Mozilla/5.0")

// Mengirimkan request dan menerima response
resp, err := client.Do(req)
if err != nil {
    return nil, err
}
defer resp.Body.Close()

// Memeriksa status response
if resp.StatusCode != http.StatusOK {
    return nil, fmt.Errorf("failed to fetch page: %s", resp.Status)
}

// Membaca isi dari response body
bodyBytes, err := io.ReadAll(resp.Body)
if err != nil {
    return nil, err
}

// Membuat dokumen HTML dari isi body
doc, err :=
goquery.NewDocumentFromReader(strings.NewReader(string(bodyBytes)))
if err != nil {
    return nil, err
}

// Menemukan dan mengekstrak tautan-tautan yang valid dalam dokumen
links := []string{}
doc.Find("a").Each(func(_ int, s *goquery.Selection) {
    link, exists := s.Attr("href")
    if exists && strings.HasPrefix(link, "/wiki/") &&
!strings.Contains(link, "Main_Page") && !strings.Contains(link, ":") {
        link = "https://en.wikipedia.org" + link
        links = append(links, link)
    }
})

// Menyimpan hasil tautan di cache
cacheMux.Lock()
cache[pageURL] = links
cacheMux.Unlock()

return links, nil

```

```
}
```

4.2 Penjelasan Tata Cara Penggunaan Program

Berikut adalah dependencies yang dibutuhkan untuk menjalankan program :


1. NodeJs
2. React
3. Golang

Berikut adalah tata cara yang harus dilakukan untuk menggunakan program :

1. Install semua dependencies yang dibutuhkan
2. Clone github, buka terminal
3. Ganti directory ke web
4. Lalu jalankan frontend dengan cara npm start
5. Buka terminal baru
6. Ganti directory ke Tubes2_ThoriqGanteng
7. Lalu jalankan backend dengan cara go run main.go
8. Buka link web pada frontend, web siap digunakan

4.3 Hasil Pengujian

4.3.1 Algoritma BFS

Test Case	Hasil
Start : Indonesia Final : Korea Mode : First Path	 <p>The screenshot shows the 'Wiki Race' application interface. At the top, there's a blue header with the title 'Wiki Race'. Below it, there are three tabs: 'BFS', 'DFS', and 'A*'. The 'BFS' tab is selected. Under the tabs, there are two input fields: 'From:' with the value 'Indonesia' and 'To:' with the value 'Korea'. Below these fields is a blue button labeled 'Jalankan'. The results are displayed in a blue box with the following information: 'Banyak Path:' followed by '1', 'Waktu Pencarian:' followed by '1 detik', 'Banyaknya link:' followed by '2.440', and 'Jumlah siklus:' followed by '3'. At the bottom of the results box, it says 'Indonesia - Muka, Jaland - Korea'.</p>

Start : Indonesia
Final : Korea
Mode : All Paths

Wiki Race

BFSIDS

FIRSTALL

From :

Indonesia

To :

Korea

Submit

Banyak Path:

26

Waktu Pencarian :

227 detik

Banyaknya link :

153161

Artikel dilalui :

3

Indonesia Buddhaen Korea
Indonesia Indonesian_National_Revolution Korea

Start : Bruno Fernandes
Final : Italy
Mode : First Path

Wiki Race

BFSIDS

FIRSTALL

From :

Bruno Fernandes

To :

Italy

Submit

Banyak Path:

1

Waktu Pencarian :

0 detik

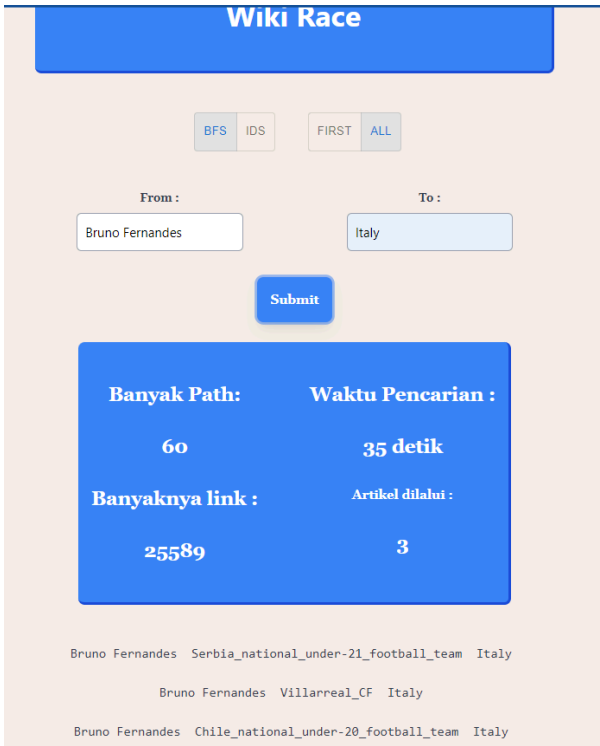
Banyaknya link :

1232

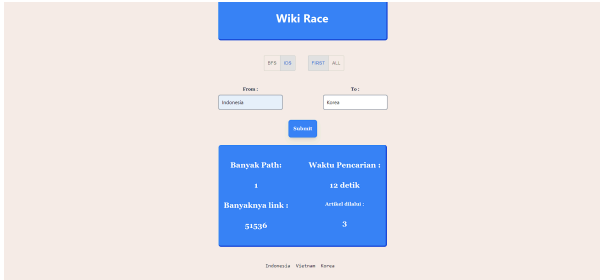
Artikel dilalui :

3

Bruno_Fernandes Henrik_Larsson Italy

<p>Start : Bruno Fernandes</p> <p>Final : Italy</p> <p>Mode : All Paths</p>	 <p>The screenshot shows the 'Wiki Race' application interface. At the top, there's a blue header with the title 'Wiki Race'. Below it, there are two sets of buttons: 'BFS' and 'IDS' (both disabled), and 'FIRST' and 'ALL' (both active). The 'From' field contains 'Bruno Fernandes' and the 'To' field contains 'Italy'. A 'Submit' button is below these fields. A large blue box displays the search results: 'Banyak Path: 60', 'Waktu Pencarian : 35 detik', 'Banyaknya link : 25589', and 'Artikel dilalui : 3'. At the bottom, there are three paths listed: 'Bruno Fernandes Serbia_national_under-21_football_team Italy', 'Bruno Fernandes Villarreal_CF Italy', and 'Bruno Fernandes Chile_national_under-20_football_team Italy'.</p>
-----------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.3.2 Algoritma IDS

Test Case	Hasil
<p>Start : Indonesia</p> <p>Final : Korea</p> <p>Mode : First Path</p>	 <p>The screenshot shows the 'Wiki Race' application interface. At the top, there's a blue header with the title 'Wiki Race'. Below it, there are two sets of buttons: 'BFS' and 'IDS' (both disabled), and 'FIRST' and 'ALL' (both active). The 'From' field contains 'Indonesia' and the 'To' field contains 'Korea'. A 'Submit' button is below these fields. A large blue box displays the search results: 'Banyak Path: 1', 'Waktu Pencarian : 12 detik', 'Banyaknya link : 35336', and 'Artikel dilalui : 3'. At the bottom, there is one path listed: 'Indonesia Vietnam Korea'.</p>

Start : Indonesia
Final : Korea
Mode : All Paths

Wiki Race

BFS IDS FIRST ALL

From : Indonesia To : Korea

Submit

Banyak Path:	Waktu Pencarian :
26	97 detik
Banyaknya link :	Artikel dilalui :
582937	3

Indonesia Vietnam Korea

Indonesia Southeast_Asia Korea

Start : Bruno Fernandes
Final : Italy
Mode : First Path

Wiki Race

BFS IDS FIRST ALL

From : Bruno Fernandes To : Italy

Submit

Banyak Path:	Waktu Pencarian :
1	0 detik
Banyaknya link :	Artikel dilalui :
2425	3

Bruno_Fernandes Novara_Calcio Italy

Start : Bruno Fernandes
Final : Italy
Mode : All Paths

Wiki Race

BFSIDS

FIRSTALL

From :

Bruno Fernandes

To :

Italy

Submit

Banyak Path:

115

Waktu Pencarian :

55 detik

Banyaknya link :

198254

Artikel dilalui :

3

Bruno Fernandes Portugal_national_under-21_football_team Italy

Bruno Fernandes Manchester_United_F.C. Italy

Bruno Fernandes Udinese_Calcio Italy

Bruno Fernandes Portugal_Olympic_football_team Italy

Bruno Fernandes U.C._Sampdoria Italy

Bruno Fernandes UEFA_Nations_League Italy

Bruno Fernandes Portugal_national_under-23_football_team Italy

Bruno Fernandes Serie_B Italy

Bruno Fernandes Portuguese_name Italy

Bruno Fernandes Serie_A Italy

Bruno Fernandes FC_Porto Italy

Bruno Fernandes Cagliari_Calcio Italy

Bruno Fernandes 2016%E2%80%93Serie_A Italy

Bruno Fernandes Vit%C3%B3ria_S.C. Italy

Bruno Fernandes Porto Italy

Bruno Fernandes 2017%E2%80%93UEFA_Champions_League Italy

Bruno Fernandes 2017_Uefa_European_Under-21_Championship Italy

Bruno Fernandes Primeira_Liga Italy

Bruno Fernandes Est%C3%A1dio_Jos%C3%A9_Alvareda Italy

Bruno Fernandes 2018%E2%80%93Sporting_CP_season Italy

Bruno Fernandes Brighton_%26_Hove_Albion_F.C. Italy

4.4 Analisis Hasil Pengujian

Pada hasil pengujian yang telah dilakukan, algoritma BFS dan IDS sudah menjalankan tugasnya dengan baik dan memberikan hasil yang sesuai ketika diminta untuk mencari path pada permainan wikirace. Algoritma BFS memiliki kompleksitas waktu $O(b^d)$ dimana b adalah percabangan maksimum dan d adalah kedalaman dari simpul tujuan. Sedangkan algoritma IDS memiliki kompleksitas waktu yang lebih buruk dari BFS karena IDS melakukan pencarian ulang dari awal pada setiap kedalaman yang ditingkatkan. Untuk setiap kedalaman algoritma ini

memiliki kompleksitas waktu $O(b^l)$ dimana b adalah percabangan maksimum dan l adalah batas kedalaman. Dalam perhitungan kompleksitas seharusnya BFS lebih cepat daripada DFS. Hal terkadang tidak terbukti dalam algoritma kamu karena pengaruh dari optimasi masing-masing algoritma, hal ini dapat dilihat dari banyak link yang diakses, dimana IDS lebih sedikit mengakses link dibandingkan BFS, dan kecepatan internet saat mengakses hyperlink yang ada di suatu page tersebut.

Untuk memenuhi syarat bonus, kami memastikan bahwa seluruh jalur terpendek yang menghubungkan URL awal dan URL tujuan dapat ditemukan. Hal ini menyebabkan program berjalan lebih lama karena setelah menemukan suatu jalur, program tidak langsung berhenti, tetapi melanjutkan pencarian untuk jalur-jalur berikutnya.

BAB 5

Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Dari tugas ini, kami berhasil membuat sebuah web yang dapat digunakan untuk permainan WikiRace dengan menggunakan algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS). Program ini memiliki fitur untuk menerima input berupa jenis algoritma (BFS atau IDS), judul artikel awal, dan judul artikel tujuan. Kemudian, program akan melakukan pencarian jalur terpendek atau jalur dengan jumlah langkah terkecil antara artikel awal dan artikel tujuan di Wikipedia.

Dari pengerjaan tugas besar ini, kami mendapatkan beberapa kesimpulan, yaitu sebagai berikut.

1. Algoritma BFS memakan lebih banyak memori karena menyimpan semua node pada graf sedangkan IDS hanya menyimpan informasi pada suatu kedalaman dalam suatu waktu.
2. Secara umum, BFS menghasilkan jalur yang lebih cepat dibandingkan IDS. Hal ini disebabkan karena IDS melakukan pencarian ulang dari awal pada setiap kedalaman yang ditingkatkan sedangkan BFS tidak.

5.2 Saran

Saran yang dapat dijadikan sebagai pembelajaran untuk tugas besar kedepannya:

1. Lebih komunikatif antar anggota, agar tidak terjadi kesalahpahaman dalam pemahaman konsep mengenai tugas besar.
2. Mendesain terlebih dahulu struktur data dan algoritma yang akan digunakan, agar tidak terjadi pengulangan struktur data atau algoritma yang memiliki fungsi yang sama.
3. Memperbaiki tampilan yang lebih baik, agar dapat lebih dinikmati bagi pengguna yang menggunakan program.
4. Mengoptimalkan pencarian baik menggunakan algoritma BFS maupun IDS, karena algoritma yang telah dibuat saat ini masih kurang optimal.
5. Menggunakan waktu pengerjaan tugas besar dengan lebih baik.

5.3 Refleksi

Dalam pengerjaan tugas besar ini, terdapat beberapa kendala seperti penggunaan bahasa pemrograman golang yang belum pernah diajarkan dalam kelas dan pembuatan interface berbasis web yang belum pernah diajarkan juga dalam kelas. Kami menyadari algoritma pencarian baik BFS dan IDS yang kami buat bukan merupakan algoritma yang paling optimal dan efisien, hal ini merupakan celah terbesar yang harus dapat ditingkatkan kedepannya. Namun, hal ini memberikan banyak pengalaman dan pengetahuan bagi kami, oleh karena itu kami berusaha untuk melakukan eksplorasi dari banyak sumber dan mendalaminya sehingga akhirnya dapat menyelesaikan tugas ini dengan lancar.

Lampiran

Repository : https://github.com/ozarabal/Tubes2_ThoriqGanteng

Daftar Pustaka

- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>
- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
- <https://wiki.spaceface.dev/>
- <https://www.sixdegreesofwikipedia.com/>
- <https://www.thewikipediagame.com/>
- <https://medium.com/@parulbaweja8/how-i-built-wikiracer-f493993fbdd>
- <https://github.com/PuerkitoBio/goquery>