

Tugas Besar 2 IF2211 Strategi Algoritma
Semester II tahun 2022/2023
Pengaplikasian Algoritma BFS dan IDS dalam Menyelesaikan Persoalan
WikiRace



Oleh:

Maulana Muhammad Susetyo (13522127)

Yosef Rafael Joshua (13522133)

Samy Muhammad Haikal (13522151)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

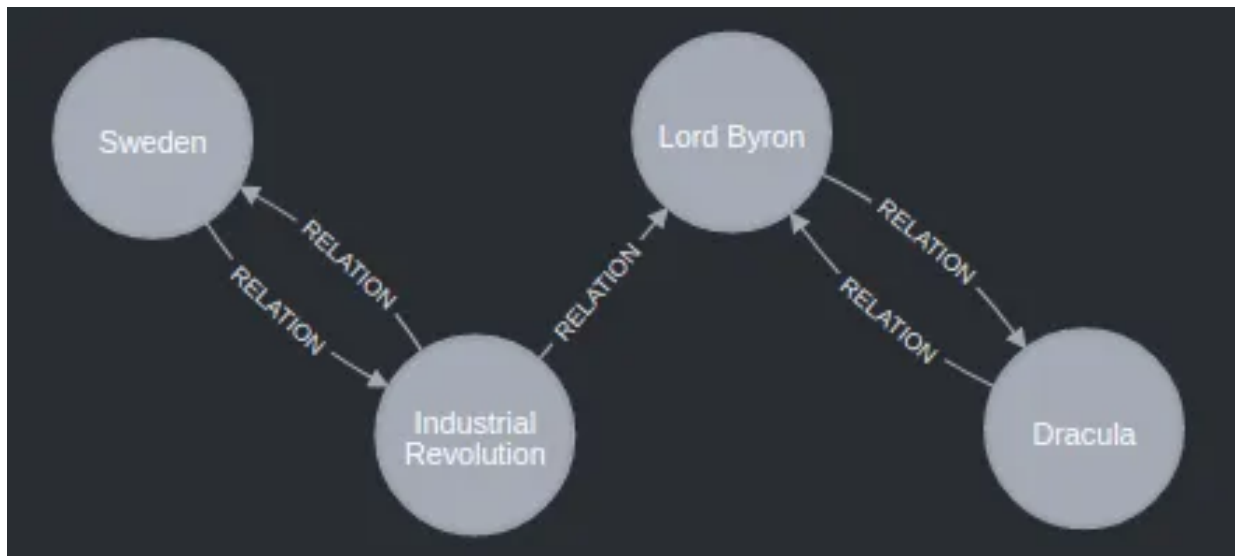
DAFTAR ISI

DAFTAR ISI	2
BAB I	3
DESKRIPSI TUGAS	3
Spesifikasi Tugas Besar 2	4
BAB II	5
LANDASAN TEORI	5
2.1 Graph Traversal	5
2.2 Algoritma Breadth-First Search	5
2.3 Algoritma Iterative-Deepening Search	5
2.4 Web-App Development	5
BAB III	6
ANALISIS PEMECAHAN MASALAH	6
3.1 Langkah-langkah pemecahan masalah	6
3.2 Mapping Persoalan Menjadi Elemen BFS dan IDS	6
3.3 Fitur Fungsional dan Aplikasi Web	6
3.4 Contoh Ilustrasi Kasus	6
BAB IV	7
IMPLEMENTASI DAN PENGUJIAN	7
BAB V	8
Kesimpulan dan Saran	8
LAMPIRAN	9
DAFTAR PUSTAKA	10

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. Ilustrasi Graf WikiRace

(Sumber: https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWYbZslicJCWQ.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.
- Program harus mengandung komentar yang jelas serta mudah dibaca.

BAB II

LANDASAN TEORI

2.1 Graph Traversal

Graph traversal adalah proses mengunjungi semua simpul/sudut dan edge/garis dari sebuah graph yang mengandung banyak simpul dan edge secara sistematis. Tujuannya adalah untuk mengakses atau memanipulasi setiap simpul dalam grafik dengan cara yang efisien dan sistematis. Pada Tugas ini ada dua Algoritma graph traversal yang digunakan: Breadth-First Search (BFS) dan Iterative-Deepening Search (IDS). Pada BFS, semua simpul di level yang sama dikunjungi sebelum melanjutkan ke level berikutnya sedangkan IDS akan melakukan BFS dengan sebuah batas kedalaman yang ditingkatkan secara bertahap.

2.2 Algoritma Breadth-First Search

Algoritma BFS (Breadth-First Search) adalah algoritma pencarian yang digunakan untuk menjelajahi atau traversing graf atau pohon secara melebar dari simpul awal yang diberikan. Algoritma ini mengunjungi semua simpul pada tingkat yang sama terlebih dahulu sebelum melanjutkan ke tingkat berikutnya. Secara umum, algoritma ini dimulai dengan mengunjungi simpul paling awal, lalu menjelajahi simpul yang bertetangga langsung dengan simpul awal tersebut, lalu simpul yang bertetangga dengan simpul tersebut, dan seterusnya hingga mencapai tujuan atau seluruh simpul dalam graf telah dikunjungi.

Algoritma BFS memastikan bahwa simpul-simpul yang lebih dekat dengan simpul awal akan dieksplorasi terlebih dahulu, sehingga algoritma ini sering digunakan untuk mencari jalur terpendek atau solusi dengan biaya minimal dalam masalah yang memiliki struktur graf.

2.3 Algoritma Iterative-Deepening Search

IDS (Iterative Deepening Search) adalah algoritma pencarian yang menggabungkan kelebihan dari BFS (Breadth-First Search) dan DFS (Depth-First Search). IDS melakukan pencarian secara DFS dengan membatasi kedalaman maksimum yang diizinkan untuk setiap iterasinya. Pada setiap iterasi, IDS meningkatkan batasan kedalaman pencarian hingga menemukan solusi yang diinginkan atau hingga mencapai kedalaman maksimum yang ditentukan.

Berikut adalah langkah-langkah umum algoritma IDS:

1. Mulai dengan kedalaman pencarian 0.
2. Lakukan pencarian DFS hingga kedalaman maksimum yang ditentukan.
3. Jika solusi ditemukan, algoritma berhenti dan mengembalikan solusi.
4. Jika solusi tidak ditemukan pada kedalaman maksimum, ulangi langkah 2 dengan meningkatkan kedalaman maksimum pencarian.

5. Ulangi langkah 3 dan 4 hingga solusi ditemukan atau semua simpul telah dieksplorasi.

Keuntungan utama dari IDS adalah bahwa ia memiliki kompleksitas waktu yang sama dengan DFS dan BFS, yaitu $O(b^d)$, di mana b adalah faktor percabangan rata-rata dan d adalah kedalaman solusi terdalam. IDS juga tidak memerlukan penyimpanan tambahan seperti BFS, karena ia hanya menyimpan simpul yang sedang dijelajahi pada setiap kedalaman tertentu. Namun, IDS masih mungkin mengulangi pencarian yang sama pada kedalaman yang lebih dalam, meskipun sudah pernah dieksplorasi pada kedalaman yang lebih dangkal.

2.4 Web-App Development

Pengembangan aplikasi web (Web App development) adalah proses menciptakan aplikasi yang berjalan di web, yang dapat diakses melalui browser web. Ini melibatkan penggunaan berbagai teknologi web seperti HTML, CSS, dan JavaScript untuk membuat antarmuka pengguna yang interaktif dan menarik. Selain itu, pengembang web juga menggunakan berbagai kerangka kerja (framework) dan alat pengembangan lainnya untuk mempercepat dan menyederhanakan proses pengembangan.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah pemecahan masalah

3.1.1 BFS

Untuk bagian BFS, program akan menerima input berupa URL wikipedia awal dan URL wikipedia tujuan. Program akan mencari semua tautan menuju halaman wikipedia lain dari laman wikipedia awal dan dimasukkan kedalam antrian. Jika URL tujuan belum ditemukan, maka akan dilakukan pencarian tautan lagi pada antrian. Hal ini dilakukan berulang kali sampai ditemukan sebuah jalur dari laman wikipedia awal menuju laman wikipedia tujuan.

3.2.2 IDS

Untuk bagian DFS program akan menerima input berupa URL wikipedia awal dan URL wikipedia akhir yang akan dicari. Program ini dibuat secara rekursif untuk mencari semua anak (link wikipedia) dari URL awal. Dalam program IDS yang dibuat, akan dilakukan iterasi maximum depth untuk pencarian. Di setiap iterasi, program akan membandingkan apakah URL sudah sama dengan URL target. Jika belum maka program akan mencari semua anak (link yang ada di webpage tersebut) dan membandingkannya lagi sampai ketemu atau iterasi maximum depth selesai. Dalam implementasinya, karena program ini rekursif, maka selalu memanggil dirinya sendiri.

3.2 Mapping Persoalan Menjadi Elemen BFS dan IDS

3.2.1 BFS

Dua Elemen utama dari BFS, atau dari graf, adalah vertex dan edge. Dalam permasalahan wikirace, vertex dapat dikaitkan dengan laman wikipedia dan edge adalah tautan dari laman wikipedia yang menghubungkannya ke laman wikipedia lain. dari sini, sudah terlihat jelas pemetaan persoalan menjadi BFS. Dengan membuat Antrian berisi string berupa href laman wikipedia, kemudian mencari semua tautan dari laman wikipedia tersebut dan memasukkannya ke dalam antrian dan mengeluarkan laman awal dari antrian (deque).

3.2.2 IDS

Program IDS menggunakan struktur data array of string untuk menyimpan jalur (path), map untuk menyimpan url yang unik. Program IDS ini dibagi dua menjadi IDS (Iterative Deepening Search) dan DLS (Depth Limited Search). Di bagian IDS akan melakukan iterasi maximum depth. Di setiap iterasi tersebut akan memanggil DLS. Di bagian DLS ini array of string dipakai untuk merepresentasikan semua child node dari URL awal, dan juga semua child node dari child yang sudah ada. Di DLS jika url tidak sesuai dengan target, maka semua child dari url tersebut akan dicari dengan membuat array of string. Kemudian map digunakan di DLS untuk mengecek apakah url tersebut pernah dijumpai sebelumnya.

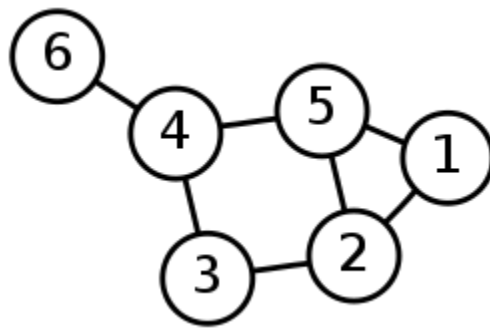
3.3 Fitur Fungsional dan Aplikasi Web

Fitur utama dari web app yang dibuat adalah dapat melakukan pencarian path dari link awal ke link tujuan. Fitur ini diimplementasikan dalam API, lalu frontend akan membuat request kepada API untuk melakukan pencarian. Jika dibedah lebih rinci lagi fitur-fitur utama web dapat dijabarkan sebagai berikut:

1. Mempunyai bagian input untuk memasukkan halaman awal dan halaman tujuan
2. Mempunyai tombol untuk melakukan pencarian baik menggunakan algoritma BFS ataupun BFS
3. Dapat menampilkan hasil dari pencarian berupa list dari halaman wikipedia yang dilewati untuk mencapai tujuan
4. Dapat menampilkan waktu pemrosesan

Adapun fitur tambahan yang diimplementasikan adalah fitur autosuggest. Fitur autosuggest akan memudahkan pengguna untuk memasukkan halaman yang sudah pasti ada laman wikipediannya.

3.4 Contoh Ilustrasi Kasus



Gambar 3.4.1 Graf (Sumber: <https://web.cecs.pdx.edu/~sheard/course/Cs163/Doc/Graphs.html>)

3.4.1 BFS

Jika dimulai dari node 1 dan ingin mencari node 4, maka urutan penelusuran graf untuk BFS adalah: 1-2-5-3-4. dimana pada node 1 akan ditambahkan node 2 dan 5 kedalam queue dan deque node 1. Kemudian hal ini dilakukan hingga node tujuan ditemukan

3.4.2 IDS

Jika dimulai dari node 1 dan ingin mencari node 4, dengan maksimum depth 2, maka urutan penelusuran graf untuk IDS adalah: 1-2-3-5-4. Namun pada kenyataannya IDS akan berhenti ketika sudah menemukan sasarannya. sedangkan jika dilakukan dengan maksimum depth 3 adalah: 1-2-3-4.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi program

a. BFS.go

```
package BFS

import (
    "fmt"
    "log"
    "net/http"
    "os"
    "strings"
    "time"

    "github.com/PuerkitoBio/goquery"
)

func Main() {
    var awal, akhir string
    var solve []string
    var err error

    fmt.Print("Input Artikel Akhir: ")
    fmt.Scanln(&akhir)

    fmt.Print("Input Artikel Mulai: ")
    fmt.Scanln(&awal)

    solve, err = BFS(awal, akhir)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Print(solve)
}

func BFS(artAwal string, artDest string) ([]string, error) {
    var temp1, temp2 string
    var solusi []string
```

```

ter, i, j, k := 0, 0, 0, 0

MapArt := scrape(artAwal)
MapArt2 := scrape(artAwal)
MapArt[artAwal] = ""
MapArt2[artAwal] = ""

start := time.Now()
for {
    fmt.Println(MapArt2[artDest])
    if _, exists := MapArt2[artDest]; exists {
        break
    }
    for key := range MapArt {
        // GET request
        if _, exists := MapArt2[artDest]; exists {
            break
        }
        fmt.Println(len(MapArt))
        fmt.Println(len(MapArt2))
        url := "https://en.wikipedia.org/wiki/" + key

        resp, err := http.Get(url)
        if err != nil {
            log.Fatal(err)
        }

        defer resp.Body.Close()

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

        // slice untuk menampung link yang sudah di scrape

        doc.Find("a").Each(func(_ int, selec *goquery.Selection) {
            href, exists := selec.Attr("href")

```

```

        if exists && strings.HasPrefix(href, "/wiki/") &&
!strings.Contains(href, ":") /* || strings.Contains(href, "Category:")
*/ { // filter link yang awalnya /wiki/ dan tidak mengandung :
        title := strings.TrimPrefix(href, "/wiki/")
        if _, ada := MapArt2[title]; !ada {
            MapArt2[title] = key
            fmt.Println(title)
            j++
        }
    }
})
i++

}
fmt.Printf("%d,%d,%d \n", i, j, k)
CopyMap(MapArt2, MapArt)
k++
if k > 8 {
    break
}
}
temp1 = artDest
solusi = append(solusi, artDest)
for ter = 1; ter < 9; ter++ {
    temp2 = MapArt[temp1]

    if temp2 == "" {
        break
    }
    solusi = append(solusi, temp2)
    temp1 = temp2
}
elapsed := time.Since(start)
fmt.Println("Elapsed time:", elapsed)
fmt.Println("Hasil:", solusi)
reverseList(solusi)
return solusi, nil
}
func scrape(str string) map[string]string {

```

```

MapArt := make(map[string]string)
// URL of the Wikipedia page you want to scrape
url := "https://en.wikipedia.org/wiki/" + str

// Make a GET request to the URL
resp, err := http.Get(url)
if err != nil {
    log.Fatal(err)
}
defer resp.Body.Close()

// Parse the HTML document
doc, err := goquery.NewDocumentFromReader(resp.Body)
if err != nil {
    log.Fatal(err)
}

// Create a new file to write the links
file, err := os.Create("links.txt")
if err != nil {
    log.Fatal(err)
}
defer file.Close()

// Find all links in the document
doc.Find("a").Each(func(i int, s *goquery.Selection) {
    // Get the href attribute of the link
    href, exists := s.Attr("href")
    if exists {
        // Check if the link points to another Wikipedia page
        if strings.HasPrefix(href, "/wiki/") && !strings.HasPrefix(href,
str) && (!strings.Contains(href, ":") || strings.Contains(href,
"Category:")) {
            // Print the title of the linked Wikipedia page
            title := strings.TrimPrefix(href, "/wiki/")
            MapArt[title] = str
            //fmt.Fprintf(file, "https://en.wikipedia.org/wiki/%s\n", title)
            title = strings.ReplaceAll(title, "_", " ")
            fmt.Println(title)

```

```

    }
    }
    })
    return MapArt
}

func CopyMap(map1 map[string]string, map2 map[string]string) {
    for key, value := range map1 {
        map2[key] = value
    }
}

func reverseList(listStr []string) {
    n := len(listStr)
    for i := 0; i < n/2; i++ {
        listStr[i], listStr[n-i-1] = listStr[n-i-1], listStr[i]
    }
}

```

b. IDS.go

```

package DFS

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

    // "os"

    "github.com/PuerkitoBio/goquery"
)

// fungsi untuk menghapus duplikat dari slice
func removeDuplicate(arr []*string) {

```

```

    seen := make(map[string]bool)

    // buat slice baru yang unik
    new := []string{}

    for _, element := range *arr {
        if !seen[element] {
            seen[element] = true
            new = append(new, element)
        }
    }
    // update slice yang lama
    *arr = new
}

func getLinks(aLink string) []string {
    // GET request
    response, err := http.Get(aLink)
    if err != nil {
        log.Fatal(err)
    }

    defer response.Body.Close()

    // Parse HTML document
    HTMLdoc, err := goquery.NewDocumentFromReader(response.Body)
    if err != nil {
        log.Fatal(err)
    }

    // slice untuk menampung link yang sudah di scrape
    var linkContainer []string

    HTMLdoc.Find("a").Each(func(_ int, selec *goquery.Selection) {
        link, found := selec.Attr("href")
        if found && strings.HasPrefix(link, "/wiki/") &&
!strings.Contains(link, ":") { // filter link yang awalnya /wiki/ dan
tidak mengandung :

```

```

        linkContainer = append(linkContainer,
"https://en.wikipedia.org"+link)
    }
})

// menghapus duplikat yang ada
removeDuplicate(&linkContainer)
return linkContainer
}

// bool karena ada pengecekan
// ptr ke jumlahArtikel karena akan ngeupdate tiap iterasi

func IDS(target string, current string, unique *map[string]bool,
jumlahArtikel *uint64, maxDepth int, prevPath []string) (bool, []string)
{
    for i := 0; i <= maxDepth; i++ {

        check, arr := DLS(target, current, i, unique, jumlahArtikel,
prevPath)

        for i2, j := 0, len(arr)-1; i2 < j; i2, j = i2+1, j-1 {
            arr[i2], arr[j] = arr[j], arr[i2]
        }

        // fmt.Println("iterasi", i, "path", arr)

        if check {
            return true, arr
        }
    }

    empty := make([]string, 0)
    return false, empty
}

func DLS(target string, current string, limit int, unique
*map[string]bool, jumlahArtikel *uint64, prevPath []string) (bool,
[]string) {

```

```

// cek apakah link sudah pernah dijumpai
if !(*unique)[current] {
    (*unique)[current] = true
}
// copy slice awal, masukkan current
tempArr := make([]string, len(prevPath), len(prevPath))
copy(tempArr, prevPath)
tempArr = append(tempArr, current)

// fmt.Println("prev:", prevPath)

// fmt.Println("arr:", tempArr)

// fmt.Println("temp:", tempArr, "current", current)

// url akhir ditemukan
if target == current {
    return true, tempArr
}

// limitnya sudah maksimal. berhenti
if limit <= 0 {
    clear(tempArr)
    return false, tempArr
}

// belum ditemukan, tapi belum mencapai limit
// buat array yang isinya children (semua link wikipedia yang ada di
webpage tersebut)
// cek untuk setiap elemen array tersebut
children := getLinks(current)
for _, element := range children {

    check, arr := DLS(target, element, limit-1, unique,
jumlahArtikel, tempArr)

    if check {
        return true, arr
    }
}

```



```

    }

    empty := make([]string, 0)
    return false, empty
}

func main() {
    // URL awal dan akhir
    url := "https://en.wikipedia.org/wiki/Joko_Widodo"
    target := "https://en.wikipedia.org/wiki/Indonesia"

    // var childLinks []string = getLinks(url)
    // for _, element := range childLinks {
    //     fmt.Println(element)
    // }
    // fmt.Println(len(childLinks))

    // boolean found
    var destFound bool = false

    // jumlah artikel
    linkMap := make(map[string]bool)
    var jumlahArtikel uint64 = 0 // kembangkan lagi

    // panjang path
    var panjangPath int = 0

    // container path dari link awal sampai link akhir
    path := make([]string, 0, 10)

    // mulai timer
    timerStart := time.Now()

    for i := 0; i <= 6; i++ {
        pathContainer := make([]string, 0, 10)
        check, array := IDS(target, url, &linkMap, &jumlahArtikel, i,
pathContainer)

```

```

        // fmt.Println("array:", i, array)

        if check {
            destFound = true
            panjangPath = i
            path = make([]string, len(array))
            copy(path, array)
            break
        }
    }

    timerStop := time.Now()
    programDuration := timerStop.Sub(timerStart)

    if destFound {
        fmt.Println("found")
        fmt.Println("panjang path:", panjangPath)
        fmt.Println("jalur:")
        for _, element := range path {
            fmt.Println(element)
        }
        fmt.Println("durasi algoritma IDS:",
programDuration.Milliseconds(), "ms")
        fmt.Println("jumlah artikel:", len(linkMap))
    } else { // !destFound
        fmt.Println("not found")
    }
}

```

c.

4.2 Tata cara penggunaan program

Tata cara menjalankan program:

1. Masuk ke folder src dan jalankan backend dengan go run api.go
2. Buka index.html menggunakan browser bebas

Tata cara menggunakan :

1. Masukkan judul wikipedia awal di kolom source link
2. Masukkan judul wikipedia tujuan di kolom destination link


- Pilih metode BFS atau IDS dan tekan tombol sesuai metode

4.3 Hasil Pengujian

Test Case 1	
BFS	<div><h3>Wikipedia Pathfinder</h3><div><div>Jokowi</div><div>Tokyo</div></div><div><div>BFS</div><div>IDS</div></div><p>Found Result using BFS Algorithm with processing time 20341 ms</p><div><div>Jokowi</div><div>Khalid bin Mohammed bin Zayed Al Nahyan</div><div>Tokyo</div></div></div>
IDS	<div><h3>Wikipedia Pathfinder</h3><div><div>Jokowi</div><div>Tokyo</div></div><div><div>BFS</div><div>IDS</div></div><p>Found Result using IDS Algorithm with processing time 112711 ms</p><div><div>Jokowi</div><div>Jakarta</div><div>Tokyo</div></div></div>
Test Case 2	

BFS	<div><h3>Wikipedia Pathfinder</h3><div><div>Morocco</div><div>Tokyo</div></div><div><div>BFS</div><div>IDS</div></div><p>Found Result using BFS Algorithm with processing time 50476 ms</p><div><div>Morocco</div><div>400 metres hurdles</div><div>Tokyo</div></div></div>
IDS	<div><h3>Wikipedia Pathfinder</h3><div><div>Morocco</div><div>Tokyo</div></div><div><div>BFS</div><div>IDS</div></div><p>Found Result using IDS Algorithm with processing time 5399 ms</p><div><div>Morocco</div><div>Urban area</div><div>Tokyo</div></div></div>
Test Case 3	
BFS	<div><h3>Wikipedia Pathfinder</h3><div><div>Morocco</div><div>Indonesia</div></div><div><div>BFS</div><div>IDS</div></div><p>Found Result using BFS Algorithm with processing time 2989 ms</p><div><div>Morocco</div><div>Indonesia</div></div></div>

IDS	<div data-bbox="350 210 1421 703"> <h3>Wikipedia Pathfinder</h3> <div> <input type="text" value="Morocco"/> <input type="text" value="Indonesia"/> </div> <div> <input type="button" value="BFS"/> <input type="button" value="IDS"/> </div> <p>Found Result using IDS Algorithm with processing time 159 ms</p> <div> <div>Morocco</div> <div>Indonesia</div> </div> </div>
Test Case 4	
BFS	<div data-bbox="350 802 1421 1295"> <h3>Wikipedia Pathfinder</h3> <div> <input type="text" value="Morocco"/> <input type="text" value="Nuremberg trials"/> </div> <div> <input type="button" value="BFS"/> <input type="button" value="IDS"/> </div> <p>Found Result using BFS Algorithm with processing time 12371 ms</p> <div> <div>Morocco</div> <div>Manchukuo</div> <div>Nuremberg trials</div> </div> </div>
IDS	<div data-bbox="350 1327 1421 1820"> <h3>Wikipedia Pathfinder</h3> <div> <input type="text" value="Morocco"/> <input type="text" value="Nuremberg trials"/> </div> <div> <input type="button" value="BFS"/> <input type="button" value="IDS"/> </div> <p>Found Result using IDS Algorithm with processing time 12152 ms</p> <div> <div>Morocco</div> <div>Governments-in-exile</div> <div>Nuremberg trials</div> </div> </div>

Test Case 5	
BFS	 <p>The screenshot shows the 'Wikipedia Pathfinder' interface. It has two input fields: 'Yogyakarta' and 'Nuremberg trials'. Below these are two buttons: 'BFS' and 'IDS'. A message states: 'Found Result using BFS Algorithm with processing time 30964 ms'. Below the message is a list of results: 'Yogyakarta', 'Empire of Japan', and 'Nuremberg trials'.</p>
IDS	

4.3 Analisis Algoritma

Secara teori metode pencarian dengan IDS seharusnya dapat menemukan hasil lebih cepat daripada metode pencarian dengan BFS. Hal tersebut karena untuk algoritma IDS tidak harus mencari semua node sedangkan BFS harus mencari semua node meskipun sudah ditemukan node yang ingin dicari. Untuk space complexity, IDS juga lebih efisien karena tidak harus menyimpan banyak data seperti BFS. Namun dari hasil pengujian yang telah kami lakukan, kami menemukan bahwa di beberapa kasus, program BFS yang kami buat lebih cepat dibandingkan dengan program IDS. Hal ini mungkin terjadi karena implementasi scraping yang berbeda dan kedua program belum dioptimasi secara maksimal. Untuk depth yang kecil BFS bisa lebih cepat karena jumlah artikel yang disimpan di dalam antrian masih terhitung dikit sehingga pencarian lebih cepat.

BAB V

Kesimpulan dan Saran

Kesimpulan

Dengan ini kami dapat menyimpulkan bahwa kami telah berhasil untuk membuat penyelesaian wikirace dengan metode IDS dan BFS. Metode IDS lebih efisien dan cepat untuk jumlah pengecekan yang banyak dan dengan depth yang lebih dalam. Sedangkan metode BFS bisa lebih cepat jika depth yang dicari tidak terlalu banyak.

Saran

1. Program kurang teroptimasi karena bottleneck scrapping
2. Perlu lebih banyak error handling

LAMPIRAN

Link Repository Github: https://github.com/rendangmunir/Tubes2_digendong-maul

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>