LAPORAN TUGAS BESAR II

IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA IDS DAN BFS DALAM PERMAINAN WIKIRACE

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 2 (dua) Tahun Akademik 2023/2024.

Oleh :

Muhammad Fiqri	10023519
Naufal Baldemar Ardanni	13521154
Hayya Zuhailii Kinasih	13522102

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

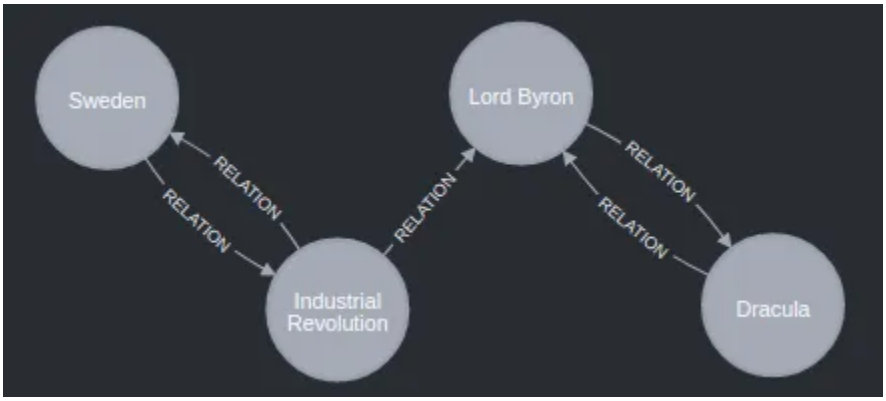
BANDUNG

2024

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

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



BAB II
LANDASAN TEORI

Dalam Tugas Besar 2 Strategi Algoritma di Institut Teknologi Bandung, kami ditugaskan untuk mengembangkan sebuah solusi yang memanfaatkan algoritma pencarian dalam struktur data graf untuk menyelesaikan permainan WikiRace. WikiRace adalah sebuah aplikasi praktis yang menantang pemain untuk berpindah dari satu artikel Wikipedia ke artikel lain dengan jumlah klik yang paling sedikit atau dalam waktu yang minimal. Algoritma yang digunakan dalam proyek ini adalah Breadth-First Search (BFS) dan Iterative Deepening Search (IDS), yang masing-masing memiliki keunikan dan keefektifan dalam menyelesaikan masalah yang ditargetkan.

2.1 Algoritma BFS (Breadth-First Search)

Breadth-First Search (BFS) adalah algoritma yang memulai pencarian dari simpul awal dan mengeksplorasi semua simpul pada kedalaman saat ini sebelum bergerak ke simpul pada kedalaman berikutnya. Dalam konteks WikiRace, BFS efektif untuk menemukan jalur terpendek karena secara sistematis memeriksa semua artikel yang bisa dijangkau dalam jumlah klik terkecil terlebih dahulu. Algoritma ini menggunakan struktur data antrian untuk mengelola simpul yang perlu dieksplorasi, memastikan bahwa semua simpul pada kedalaman tertentu sepenuhnya dieksplorasi sebelum melanjutkan ke kedalaman yang lebih dalam.

2.2 Algoritma IDS (Iterative Deepening Search)

IDS menggabungkan keefektifan dari Depth-First Search (DFS) dalam penggunaan memori yang lebih hemat dengan kemampuan BFS dalam menemukan solusi dengan jarak terpendek terlebih dahulu. Algoritma ini melakukan pencarian mendalam berulang dengan batas kedalaman yang terus meningkat. Dalam setiap iterasi, IDS berperilaku seperti DFS tetapi dengan batas kedalaman yang terkendali, dan pencarian diulang dengan meningkatkan batas tersebut hingga solusi ditemukan. Ini sangat berguna dalam aplikasi seperti WikiRace, di mana membatasi kedalaman

pencarian membantu menghindari eksplorasi yang tidak perlu pada tautan yang tidak relevan dan secara efektif menemukan rute terpendek dalam ruang pencarian yang besar.

2.3 Aplikasi Web

Pengembangan aplikasi web untuk WikiRace mencakup pembuatan antarmuka pengguna (front-end) yang memudahkan pengguna untuk memasukkan judul artikel awal dan tujuan serta menampilkan hasil pencarian rute. Bagian server (back-end) mengelola logika aplikasi, termasuk implementasi algoritma BFS dan IDS, dan pengolahan data dari Wikipedia. Aplikasi ini tidak harus dideploy secara online tetapi harus mudah digunakan dan diakses melalui browser lokal, memastikan bahwa semua interaksi pengguna dan proses algoritma dapat dijalankan secara efisien.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Langkah awal dalam memecahkan masalah permainan WikiRace adalah memahami dan mendefinisikan model data yang akan digunakan. Kami memulai dengan membangun representasi graf dari artikel-artikel Wikipedia, di mana setiap artikel diwakili sebagai simpul dan setiap hyperlink sebagai sisi. Selanjutnya, kami mengimplementasikan algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS) untuk menelusuri graf ini. Kedua algoritma ini dipilih karena efisiensinya dalam mencari jalur terpendek di dalam graf yang luas dan kompleks seperti Wikipedia. Kami juga menguji coba algoritma-algoritma ini dalam skenario yang berbeda untuk menentukan konfigurasi optimal dalam berbagai kasus.

3.2 Pemetaan Masalah

Seperti yang telah dijabarkan di atas, algoritma BFS dan IDS digunakan untuk menelusuri simpul-simpul dalam graf. Dalam kasus *Wikirace*, graf yang dimaksud terdiri atas artikel sebagai simpul dan hyperlink sebagai sisi yang berarah. Graf juga harus dibangun selagi melakukan penelusuran, maka pencarian dilakukan dengan graf dinamis.

Untuk algoritma pencarian dengan graf dinamis, terdapat operator untuk menambahkan simpul pada graf, simpul yang berupa status persoalan, daun yang berupa status solusi, ruang solusi, dan ruang status. Penambahan simpul pada graf dilakukan dengan membaca *hyperlink* yang terdapat pada simpul persoalan, kemudian dihubungkan melalui struktur data map yang berisi informasi mengenai *parent* dari simpul yang bersangkutan.

Pada algoritma BFS, simpul-simpul yang didapatkan dimasukkan ke dalam *queue* untuk diproses, dan ketika ditemukan solusi yang tepat, pemrosesan simpul dalam queue dihentikan. Pada algoritma IDS, yang merupakan DFS bertahap, setiap mendapat simpul baru, simpul dimasukkan ke dalam stack. Bila belum mencapai batas kedalaman iterasi tertentu, simpul tersebut akan diproses. Bila sudah, simpul akan dikeluarkan dari stack jika bukan merupakan solusi.

3.3 Fitur Fungsional pada Aplikasi Web

No.	Fitur
F01	Aplikasi dapat menerima input berupa judul artikel awal, judul artikel tujuan, dan jenis algoritma yang ingin dipakai.
F02	Aplikasi dapat memperlihatkan hasil dari pemrosesan input, yaitu jumlah artikel yang dilewati, jarak artikel awal ke artikel tujuan, rute untuk menuju ke artikel tujuan, dan waktu yang dibutuhkan dalam millisecond.
F03	Aplikasi memperlihatkan rute terdekat untuk sampai ke artikel tujuan.
F04	Aplikasi memberikan pesan error jika artikel tujuan yang dimasukkan tidak ada.
F05	Aplikasi dapat menampilkan sugesti *autocomplete* ketika pengguna sedang menulis artikel yang ingin digunakan.

3.4 Ilustrasi Kasus

Misalkan artikel yang ingin digunakan sebagai titik awal adalah artikel “Wikiracing”, dan artikel yang ingin dituju adalah “Disneyland”. Jika menggunakan algoritma BFS, program akan memasukkan “Wikiracing” ke dalam *queue.* Kemudian, untuk setiap artikel pada *queue* selama *queue* tidak kosong, artikel tersebut akan di-*scrape* untuk mendapatkan artikel-artikel selanjutnya. Artikel-artikel yang baru dimasukkan ke dalam *queue*, untuk diproses juga nantinya. Pada saat yang sama, setiap kali artikel diproses, artikel tersebut ditandai bahwa sudah dikunjungi, sehingga artikel tersebut tidak akan dikunjungi lagi. Sebagai contoh, pada artikel “Wikiracing”, terdapat tautan ke artikel “Crowdsourcing”. Artikel tersebut akan ditambahkan ke dalam queue dan juga graf. Ketika program sedang mengambil tautan dari artikel “Gamification”, program akan mendapatkan tautan ke “Crowdsourcing”. Karena sudah terdapat dalam graf, tautan tersebut dibiarkan. Pada artikel “Gamification” juga ditemukan tautan menuju “Disneyland”, namun program tidak langsung berhenti. Program akan berhenti saat giliran artikel “Disneyland” diproses dari *queue*.

Konsep yang serupa berlaku pada algoritma IDS, hanya saja urutan penelesurannya berbeda. Algoritma IDS akan menelusuri satu artikel, kemudian menelusuri artikel yang terkandung di dalamnya, seterusnya hingga sampai ke batas kedalaman iterasi tersebut. Kemudian algoritma akan melakukan *backtrack* untuk menelusuri cabang yang lain.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur Data

Struktur data yang digunakan pada algoritma BFS dan IDS adalah sebagai berikut:

1. Struct

   Struct yang dibuat berupa struct Article yang memiliki atribut Title bertipe string dan URL yang juga bertipe string. Struct tersebut berfungsi untuk menampung data judul dan tautan ke suatu halaman Wikipedia. Selain itu, juga dibuat struct Result yang memiliki atribut Articles bertipe list of Article, ArticleDistance bertipe integer, ArticlesVisited bertipe integer, dan TimeElapsed bertipe string. Struct tersebut berfungsi menampung data yang akan ditampilkan pada aplikasi.

2. Map

   Program ini menggunakan map untuk beberapa kepentingan. Pertama, map redirectMap bertipe [string]bool berisi semua tautan yang akan teralihkan ke halaman Wikipedia yang dituju. Struktur data tersebut dipilih untuk memudahkan penemuan halaman tujuan. Kemudian, map juga digunakan untuk mencatat tautan asal suatu tautan, alias *parent* suatu tautan. Terakhir, map digunakan pada algoritma IDS untuk mencatat kedalaman suatu halaman dari halaman awal.

3. List

   Program ini menggunakan list of string sebagai queue pada algoritma BFS dan list of string sebagai stack pada algoritma IDS. Selain itu, list juga digunakan pada struct Result untuk menyimpan rute menuju halaman tujuan.

4.1.2 Fungsi dan Prosedur

1. Prosedur main

```
// file main.go
func main() {
    mux := http.NewServeMux()
    fmt.Println("http://localhost:8080/")
    mux.Handle("/css/", http.StripPrefix("/css/",
http.FileServer(http.Dir("view/stylesheet"))))
```

```
    mux.HandleFunc("/", mainHandler)
    mux.HandleFunc("/search", searchHandler)
    http.ListenAndServe(":8080", mux)
}
```

2. Prosedur mainHandler

```
// file main.go
func mainHandler(w http.ResponseWriter, r
*http.Request) {
    var result Result
    result.ArticleDistance = -2
    t.Execute(w, result)
}
```

3. Prosedur searchHandler

```
// file main.go
func searchHandler(w http.ResponseWriter, r
*http.Request) {
    u, err := url.Parse(r.URL.String())
    if err != nil {
        fmt.Println(err)
        return
    }

    queries := u.Query()
    sourceTitle := queries.Get("source")
    destTitle := queries.Get("dest")
    algorithm := queries.Get("algorithm")

    var source Article
    tempURL := urlBuilder(sourceTitle)
    source.Title = getTitle(tempURL)
    source.URL = urlBuilder(source.Title)

    var dest Article
    tempURL = urlBuilder(destTitle)
    dest.Title = getTitle(tempURL)
```

```
    dest.URL = urlBuilder(dest.Title)

    var result Result

    go func() {
        time.Sleep(5 * time.Minute)
        result.ArticleDistance = -1
        t.Execute(w, result)
    } ()

    if dest.Title == source.Title {
        start := time.Now()
        result.ArticleDistance = 0
        result.ArticlesVisited = 1
        result.Articles = append(result.Articles, dest)
        result.TimeElapsed = time.Since(start).String()
    } else {
        if algorithm == "bfs" {
            result = BFSSearch(source, dest)
        }
        if algorithm == "ids" {
            result = IDSSearch(source, dest)
        }
    }

    t.Execute(w, result)
}
```

4. Fungsi adjustTitle

```
// file util.go
// trim spaces and add underscore
func adjustTitle(str string) string {
    return strings.ReplaceAll(strings.TrimSpace(str), "
", "_")
}
```

5. Fungsi urlBuilder

```go
// file util.go
// return url to wikipedia page
func urlBuilder(title string) string {
    base := "https://en.wikipedia.org/wiki/"
    title = adjustTitle(title)
    return base + title
}
```

6. Fungsi isAcceptable

```go
// file util.go
// check if given link is acceptable, ie. leads to a
wikipedia article
func isAcceptable(url string) (string, bool) {
    if !strings.HasPrefix(url, "/wiki/") ||
strings.Contains(url, ":") {
        return "https://en.wikipedia.org" + url, false
    }
    return "https://en.wikipedia.org" + url, true
}
```

7. Fungsi getPath

```go
// file util.go
// get path from article to the root article
func getPath(str string, parent map[string]string)
[]string {
    var path []string
    for parent[str] != "none" {
        path = append([]string{str}, path...)
        str = parent[str]
    }
    return path
}
```

8. Fungsi getTitle

```go
// file util.go
// scrape url to get its title
```

```
func getTitle(link string) string {
    resp, err := http.Get(link)
    if err != nil {
        fmt.Println(err)
        return ""
    }
    defer resp.Body.Close()

    doc, err :=
goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        fmt.Println(err)
        return ""
    }
    title := doc.Find("h1#firstHeading").Text()
    return title
}
```

9. Fungsi buildWhatLinksHereURL

```
// file util.go
// return link to access redirect pages
func buildWhatLinksHereURL(title string) string {
    return
"https://en.wikipedia.org/wiki/Special:WhatLinksHere?ta
rget=" + adjustTitle(title) +
"&namespace=0&limit=500&hidetrans=1&hidelinks=1"
}
```

10. Fungsi getRedirects

```
// file util.go
// get redirects
func getRedirects(link string, redirectMap
*map[string]bool) {
    resp, err := http.Get(link)
    if err != nil {
        fmt.Println(err)
        return
```

```
    }
    defer resp.Body.Close()

    doc, err :=
goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        fmt.Println(err)
        return
    }
    content :=
doc.Find("ul#mw-whatlinkshere-list").Find("a.mw-redirec
t")
    content.Each(func(index int, item
*goquery.Selection) {
        title := item.Text();
        if title != "edit" {
            (*redirectMap)[urlBuilder(title)] = true
        }
    })
    next, exist :=
doc.Find("div.mw-pager-navigation-bar").Find("a.mw-next
link").Attr("href")
    if exist {
        getRedirects("https://en.wikipedia.org" + next,
redirectMap)
    }
}
```

11. Fungsi BFSSearch

```
// file bfs.go
func BFSSearch(source, dest Article) Result {
    var result Result
    var redirectMap = make(map[string]bool)
    redirectMap[dest.URL] = true
    getRedirects(buildWhatLinksHereURL(dest.Title),
&redirectMap)

    start := time.Now()
```

```
    result.Articles = append(result.Articles, source)

    path, totalVisited := bfs(source.URL, redirectMap)

    for _, p := range path {
        var article Article
        article.URL = p
        article.Title = getTitle(p)
        result.Articles = append(result.Articles,
article)
    }

    result.ArticlesVisited = totalVisited
    result.ArticleDistance = len(result.Articles) - 1
    result.TimeElapsed = time.Since(start).String()

    return result
}
```

12. Fungsi bfs

```
// file bfs.go
func bfs(source string, redirectMap map[string]bool)
([]string, int) {
    // initial declarations
    var parent = make(map[string]string)
    parent[source] = "none"
    var queue []string
    queue = append(queue, source)
    var foundLink string
    found := false
    depth := 0
    totalVisited := 1

    // search loop, ends when destination page is found
    for {
        fmt.Println(depth)
        // variables for concurrent search
        var prev = make(chan string)
```

```
        var next = make(chan string, 1000)
        var wg sync.WaitGroup

        // setup 100 threads to scrape links
        wg.Add(100)
        for i:=0; i<100; i++ {
            go getMultipleLinks(prev, next, &wg,
&parent)
        }
        go closeChan(next, &wg)

        // put queue items in channel for scraping
        go putInChan(queue, prev)

        // iterate over links received from scraping,
put in queue
        queue = nil
        fmt.Println(depth)
        for link := range next {
            totalVisited++
            if redirectMap[link] {
                found = true
                foundLink = link
                break
            }
            queue = append(queue, link)
        }
        if found {
            break
        }
        depth++
    }
    if found {
        return getPath(foundLink, parent), totalVisited
    }
    return nil, 0
}
```

13. Prosedur getMultipleLinks

```go
// file bfs.go
// scrape multiple pages
func getMultipleLinks(prev, next chan string, wg
*sync.WaitGroup, parent *map[string]string) {
    i := 0
    for item := range prev {
        i++
        getLinksBFS(next, item, parent)
    }
    (*wg).Done()
}
```

14. Prosedur putInChan

```go
// file bfs.go
// put list items in channel
func putInChan(arr []string, ch chan string) {
    i := 0
    for _, item := range arr {
        i++
        ch <- item
    }
    close(ch)
}
```

15. Prosedur closeChan

```go
// file bfs.go
// close channel once all waitgroups are done
func closeChan(ch chan string, wg *sync.WaitGroup) {
    (*wg).Wait()
    close(ch)
}
```

16. Prosedur getLinksBFS

```go
// file bfs.go
// get links from a page
func getLinksBFS(next chan string, current string,
```

```
parent *map[string]string) {
    resp, err := http.Get(current)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer resp.Body.Close()

    doc, err :=
goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        fmt.Println(err)
        return
    }
    content := doc.Find("div#bodyContent").Find("a")
    content.Each(func(index int, item
*goquery.Selection) {
        link, _ := item.Attr("href")
        link, valid := isAcceptable(link)
        if !valid {
            return
        }

        mut.Lock()
        if (*parent)[link] == "" {
            (*parent)[link] = current
            next <- link
        }
        mut.Unlock()
    })
}
```

17. Fungsi IDSSearch

```
// file ids.go
func IDSSearch(source, dest Article) Result {
    var result Result
    var redirectMap = make(map[string]bool)
    redirectMap[dest.URL] = true
```

```
    getRedirects(buildWhatLinksHereURL(dest.Title),
&redirectMap)

    start := time.Now()
    result.Articles = append(result.Articles, source)

    path, totalVisited := iterate(source.URL,
redirectMap, 0)

    for _, p := range path {
        var article Article
        article.URL = p
        article.Title = getTitle(p)
        result.Articles = append(result.Articles,
article)
    }

    result.ArticlesVisited = totalVisited
    result.ArticleDistance = len(result.Articles) - 1
    result.TimeElapsed = time.Since(start).String()

    return result
}
```

18. Fungsi iterate

```
// file ids.go
// handle iterations of ids
func iterate(source string, redirectMap
map[string]bool, iter int) ([]string, int) {
    var stack []string
    var parent = make(map[string]string)

    stack = append(stack, source)
    depth[source] = 0
    parent[source] = "none"

    // fmt.Println(iter)
    stack, found, totalVisited := dls(stack, iter,
```

```
redirectMap, &parent)
    if found {
        return getPath(stack[0], parent), totalVisited
    } else {
        iter++
        return iterate(source, redirectMap, iter)
    }
}
```

19. Fungsi dls

```
// file ids.go
// depth limited search
func dls(stack []string, max int, redirectMap
map[string]bool, parent *map[string]string) ([]string,
bool, int) {
    totalVisited := 0

    for len(stack) > 0 {
        totalVisited++
        if redirectMap[stack[0]] {
            return stack, true, totalVisited
        }

        if depth[stack[0]] < max {
            nodes := getLinksIDS(stack[0], parent)
            stack = append(nodes, stack[1:]...)
        } else {
            stack = stack[1:]
        }
    }
    return stack, false, totalVisited
}
```

20. Fungsi getLinks IDS

```
// file ids.go
// get links
func getLinksIDS(current string, parent
```

```
*map[string]string) []string {
    var urls []string
    d := depth[current] + 1
    resp, err := http.Get(current)
    if err != nil {
        fmt.Println(err)
        return nil
    }
    defer resp.Body.Close()

    doc, err :=
goquery.NewDocumentFromReader(resp.Body)
    if err != nil {
        fmt.Println(err)
        return nil
    }

    content := doc.Find("div#bodyContent").Find("a")
    content.Each(func(index int, item
*goquery.Selection) {
        link, _ := item.Attr("href")
        link, valid := isAcceptable(link)
        if !valid {
            return
        }
        if (*parent)[link] == "" {
            (*parent)[link] = current
            depth[link] = d
            urls = append(urls, link)
        }
    })
    return urls
}
```

4.2 Tata Cara Penggunaan Program

Langkah awal penggunaan program adalah dengan meng-*clone repository* GitHub dengan membuka terminal dan jalankan perintah berikut:

```
git clone https://github.com/hayyazk/Tubes2_wiikiii.git
```

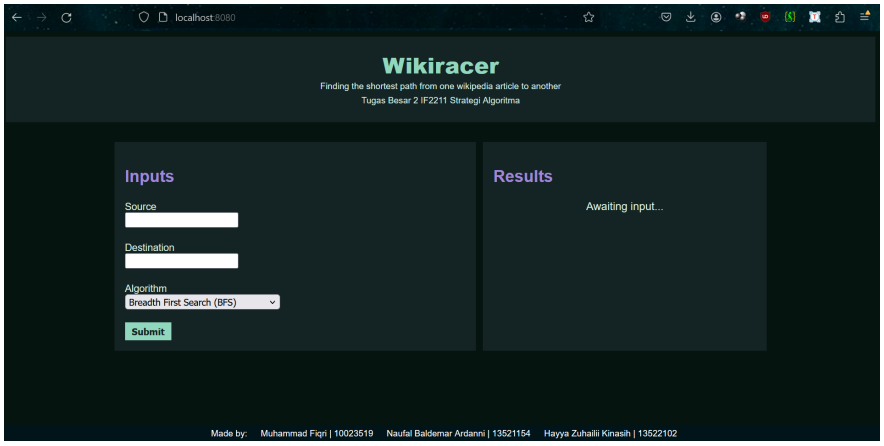
Kemudian, dalam folder src, jalankan perintah berikut:

```
go build
```

Jalankan file executable yang telah dibuat dengan perintah berikut:

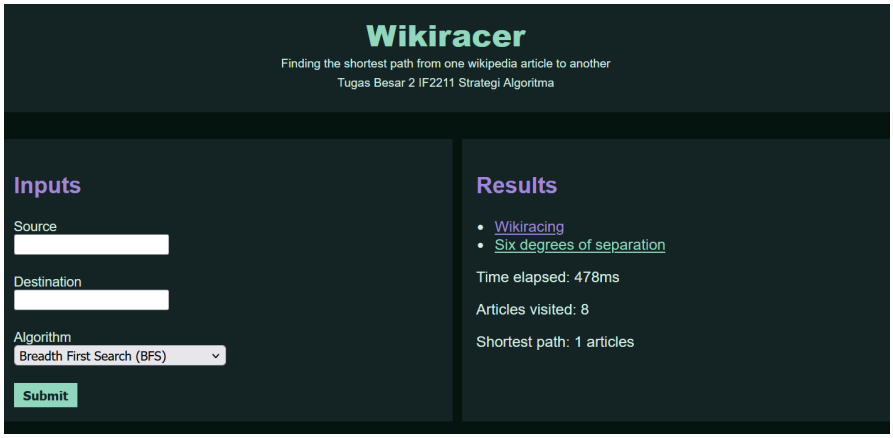
```
./Tubes2_wiikiii.exe
```

Ketik link yang terdapat pada terminal atau buka `localhost:8080` pada browser. Akan muncul tampilan seperti di bawah ini.



Masukkan judul kedua artikel Wikipedia yang ingin ditemukan rutenya dan pilih algoritma yang ingin digunakan.

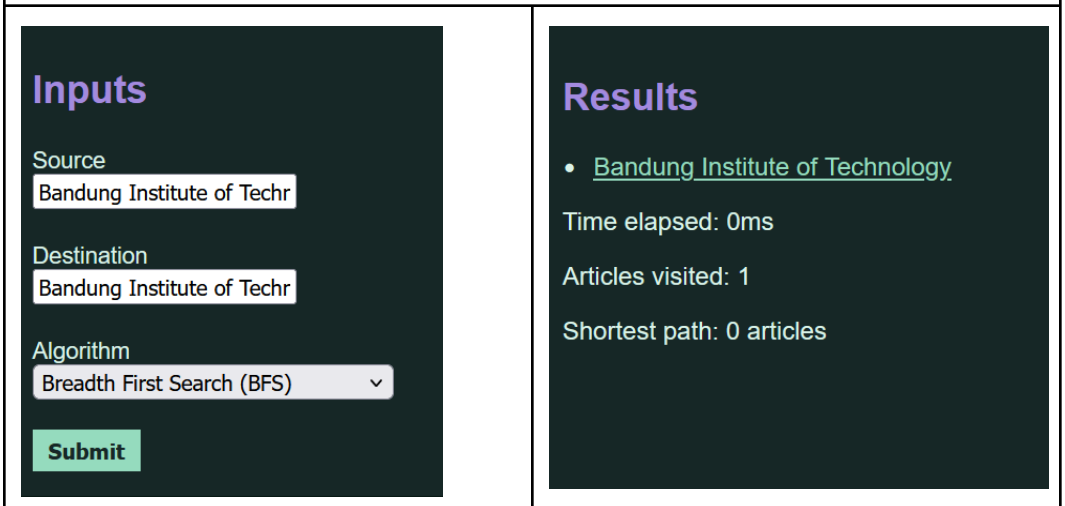
Klik tombol Submit. Input akan diproses dan hasilnya akan ditampilkan di bagian Results.



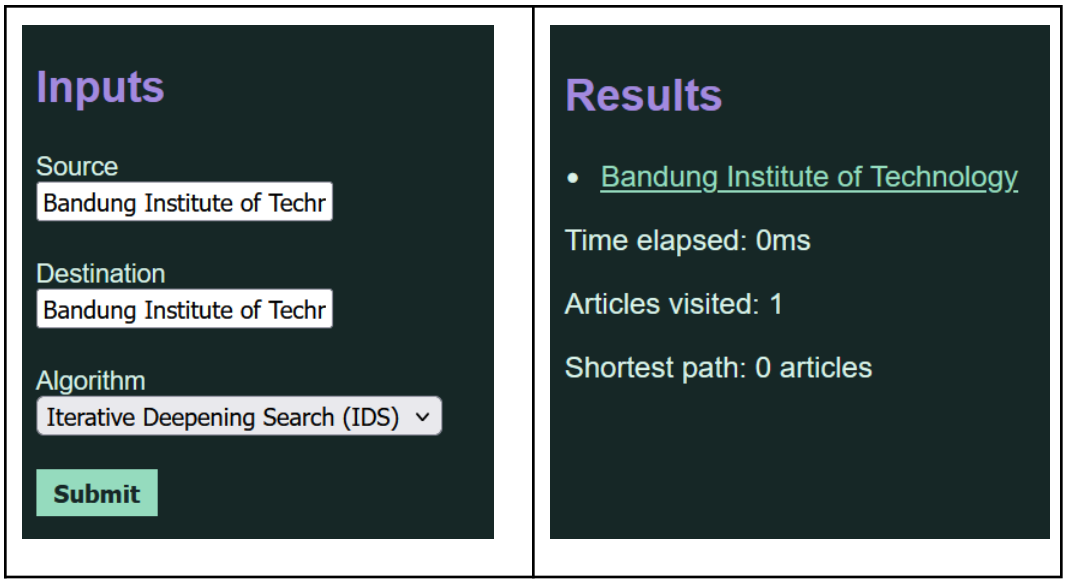
4.3 Hasil Pengujian

1. Pengujian jarak artikel 0

BFS



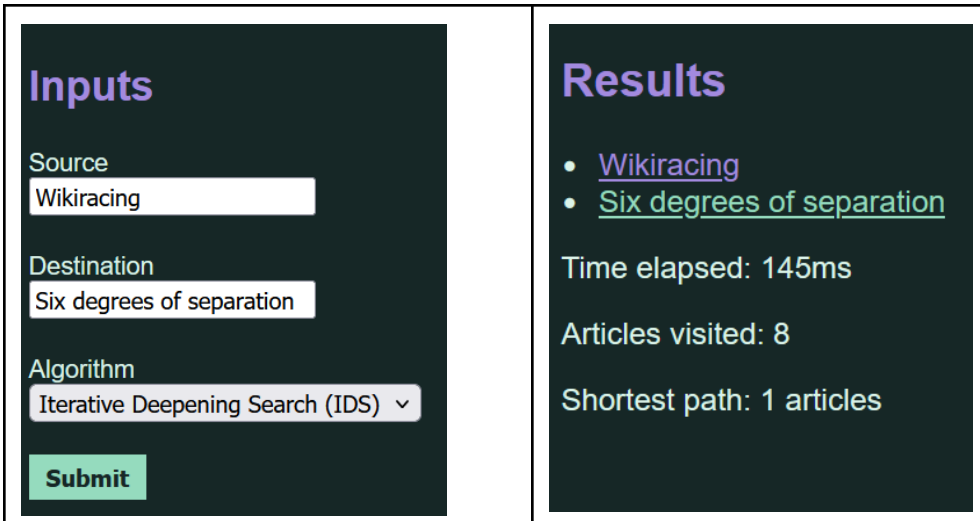
IDS



2. Pengujian jarak artikel 1

BFS

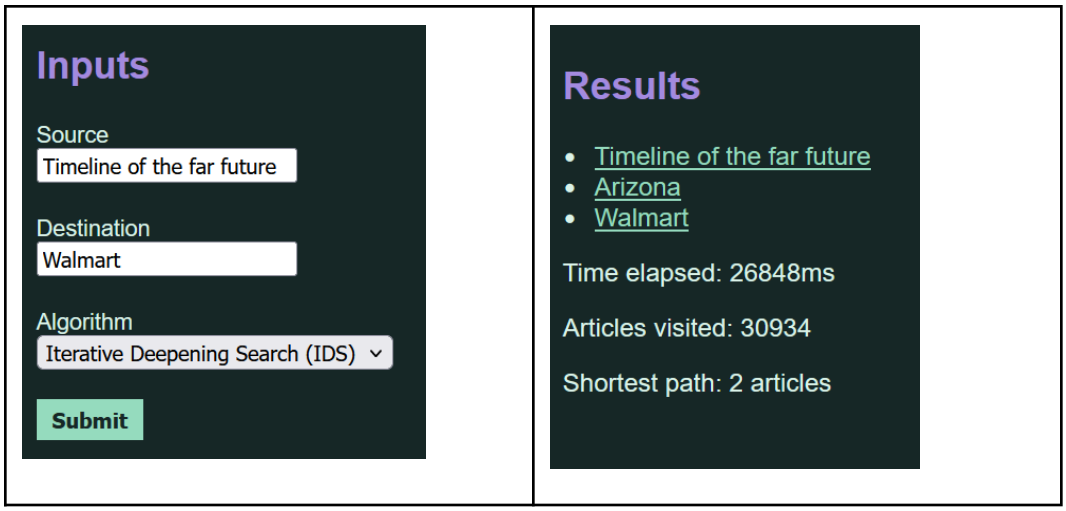
IDS



3. Pengujian jarak artikel 2

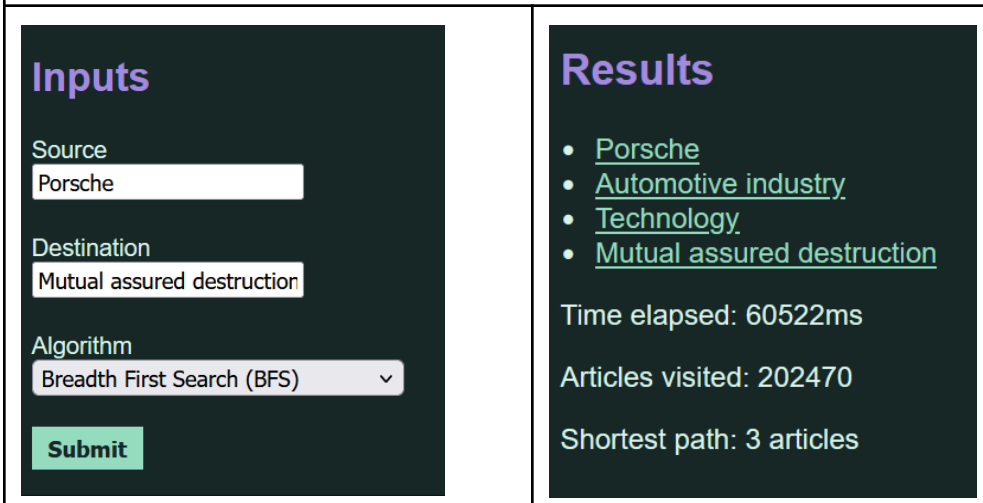
BFS

IDS

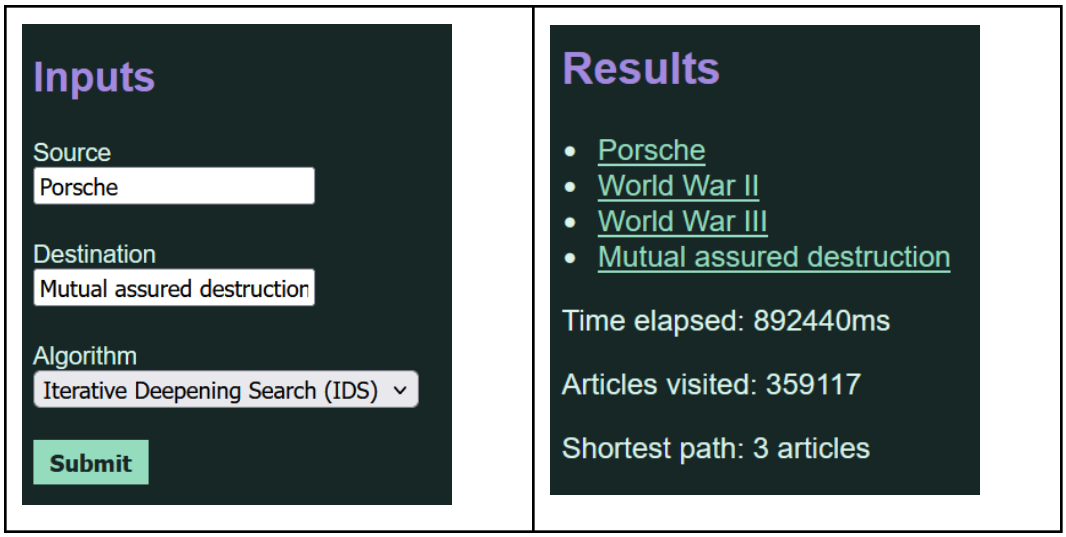


4. Pengujian jarak artikel 3

BFS



IDS



4.4 Analisis Hasil Pengujian

Berdasarkan beberapa pengujian yang dilakukan, algoritma IDS cenderung lebih cepat daripada BFS dalam menemukan solusi jika kedalamannya rendah. Semakin dalam solusinya, waktu yang dibutuhkan algoritma IDS juga semakin meningkat. Hal tersebut disebabkan oleh perlunya algoritma IDS untuk mengulangi pencarian dari artikel awal setiap iterasi. Kami berupaya untuk mengimplementasikan cache, ataupun metode lain, untuk mengurangi waktu yang termakan dalam setiap iterasi, namun program yang ditulis tidak dapat berjalan dengan baik. Berdasarkan kompleksitas waktu kedua algoritma tersebut, keduanya memiliki kompleksitas yang sama, yaitu O(b^d). Namun, dalam implementasi, algoritma BFS bisa menjadi lebih cepat daripada IDS karena pada BFS kami memanfaatkan fitur *goroutine* pada bahasa pemrograman Go. Karena algoritma BFS dapat melakukan *scraping* pada lebih banyak artikel pada satu waktu, algoritma tersebut menjadi lebih cepat.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Tugas besar 2 IF2211 Strategi Algoritma memiliki objektif untuk membuat sebuah aplikasi web yang dapat memanfaatkan *web scraping* beserta algoritma Breadth-First Search (BFS) dan Iterative Deepening Search (IDS) untuk menemukan rute terpendek untuk berangkat dari satu artikel Wikipedia ke artikel Wikipedia yang lain.

5.2 Saran

Untuk kedepannya, saran yang dapat diambil adalah sebagai berikut:

1. Cari informasi lebih dalam terkait optimisasi yang dapat dilakukan agar program berjalan dengan efisien.
2. Pelajari kembali konsep *concurrency* dan pengimplementasiannya.
3. Eksplorasi metode-metode penyimpanan data dengan jumlah yang besar.

LAMPIRAN

Repository : https://github.com/hayyazk/Tubes2_wiikiii/

DAFTAR PUSTAKA

Spaceface Dev. WikiRace Solver dengan Single Solution. Diambil dari https://wiki.spaceface.dev/

Six Degrees of Wikipedia. WikiRace Solver dengan Multiple Solutions. Diambil dari https://www.sixdegreesofwikipedia.com/

The Wikipedia Game. Salah Satu Situs Permainan WikiRace. Diambil dari https://www.thewikipediagame.com/

Baweja, P. How I Built WikiRacer. Medium. Diambil dari https://medium.com/@parulbaweja8/how-i-built-wikiracer-f493993fbdd

PuerkitoBio. GoQuery. GitHub. Diambil dari https://github.com/PuerkitoBio/goquery