TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2023/2024

“Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace”

OLEH:

Irfan Sidiq Permana	13522007
Bastian H Suryapratama	13522034
Diero Arga Purnama	13522056

PROGRAM STUDI TEKNIK INFORMATIKA

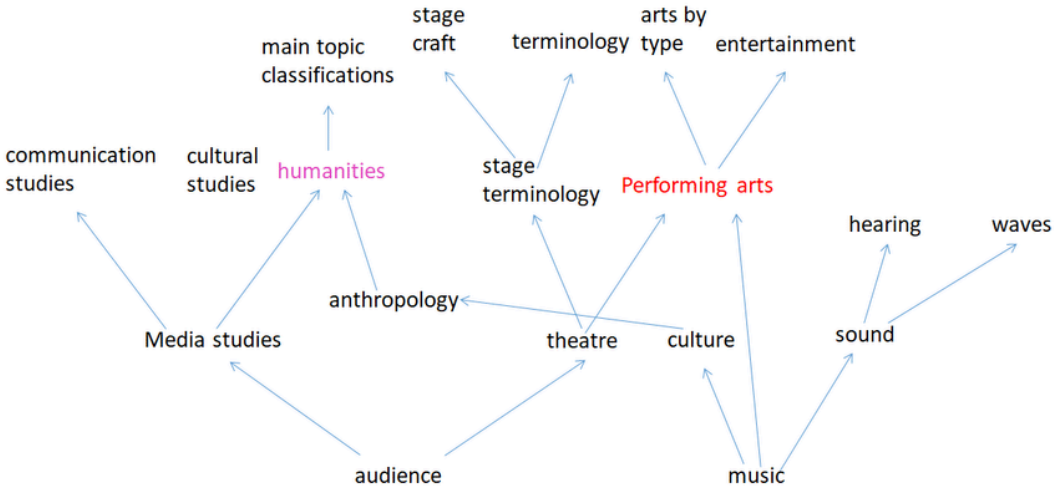
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

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.



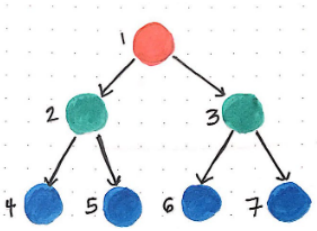
Gambar 1.1 Contoh Graf Wikipedia

Pada Tugas Besar 2 ini, kelompok kami merancang program untuk menerapkan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) untuk menemukan jalur terpendek dari satu artikel wikipedia menuju satu artikel wikipedia lain. Program dibuat menggunakan bahasa Go untuk algoritma backend, sedangkan untuk pembuatan website menggunakan *vanilla* HTML, CSS, dan Javascript. Program menerima input berupa: 1) Judul artikel awal dan judul artikel tujuan, 2) Pilihan algoritma (BFS atau IDS), dan 3) Pilihan pencarian solusi *single path* atau *multiple path*.

BAB II
LANDASAN TEORI

2.1. Algoritma *Breadth-First Search* (BFS)

Algoritma *breadth-first search* (BFS) adalah strategi pencarian dalam graf yang melakukan pencarian dengan cara mengunjungi simpul-simpul dalam graf secara melebar. Pencarian akan dimulai dari simpul awal dan dilanjutkan dengan mengunjungi semua tetangga dari simpul tersebut sebelum melanjutkan pencarian ke simpul-simpul yang memiliki kedalaman yang lebih tinggi.

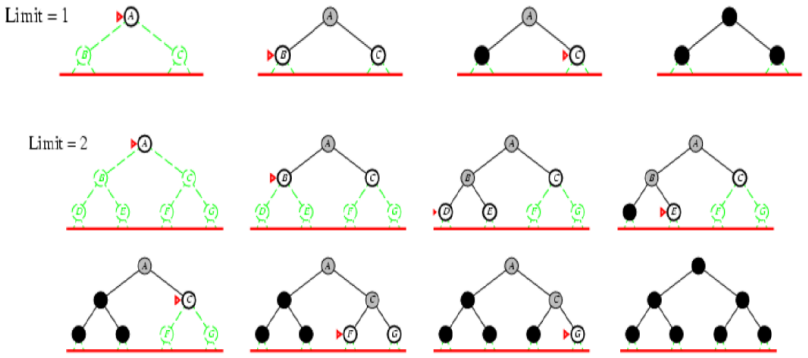


Gambar 2.1 Contoh Alur Algoritma Breadth-First Search (BFS)

Kelebihan utama dari algoritma Breadth-First Search (BFS) adalah kemampuannya untuk menemukan solusi optimal dengan jumlah langkah minimum dalam graf yang tidak memiliki bobot pada setiap edge atau memiliki bobot yang sama pada setiap edge. BFS juga menjamin menemukan solusi terpendek pertama kali ditemukan, sehingga cocok digunakan dalam aplikasi di mana solusi terpendek diperlukan. Selain itu, BFS juga dapat digunakan untuk menghasilkan jalur terpendek antara dua simpul dalam graf. Meskipun BFS membutuhkan memori tambahan untuk menyimpan simpul yang telah dieksplorasi (kebutuhan memori meningkat secara eksponensial dibandingkan DFS), algoritma ini relatif mudah diimplementasikan dan dapat diterapkan dalam berbagai masalah pencarian graf.

2.2. Algoritma *Iterative Deepening Search* (IDS)

Algoritma *iterative deepening search* (IDS) adalah strategi pencarian yang menggabungkan pendekatan dari Depth-Limited Search (DLS) dan Breadth-First Search (BFS). Depth-Limited Search (DLS) sendiri ialah algoritma Depth-First Search (DFS) yang pencariannya dibatasi hanya sampai kedalaman tertentu, misalnya *h,* dengan $h \geq 0$. IDS bekerja dengan cara melakukan pencarian secara berulang-ulang dengan meningkatkan batasan kedalaman pencarian pada setiap iterasi, yaitu dengan memanggil algoritma DLS berulang kali dengan batas kedalaman *h* yang meningkat hingga ditemukan solusi target.



Gambar 2.2 Contoh Alur Algoritma Iterative Deepening Search (IDS)

Kelebihan dari pendekatan algoritma IDS ini yaitu algoritma ini menggabungkan semua keuntungan dari pendekatan DLS dan BFS. Algoritma IDS ini memenuhi sifat *completeness* dan *optimality*, dimana algoritma IDS mencari semua kemungkinan solusi sehingga pasti menemukan solusi (bila memang ada) dan selalu mendapatkan solusi yang paling optimal (jika jumlah langkah = biaya). IDS memiliki kompleksitas waktu yang sama dengan BFS, yaitu $O(b^d)$, serta memiliki kompleksitas waktu yang sama dengan DFS, yaitu $O(bd)$, dengan *b* menyatakan maksimum percabangan yang mungkin dan *d* menyatakan kedalaman dari solusi terbaik. Maka dari itu, kita dapat mengira bahwa IDS selalu lebih baik dari BFS dan DFS, karena IDS memiliki keuntungan dari keduanya dilihat dari sisi kompleksitas waktu maupun ruang. Namun pada nyatanya, karena IDS selalu mengulangi pencarian kembali ke kedalaman 0 tiap kali ingin menambahkan batas kedalaman (demi mempertahankan kompleksitas ruang $O(bd)$ seperti DFS), maka sejatinya IDS selalu lebih lambat dari BFS yang tidak perlu kembali ke kedalaman sebelumnya.

2.3. Deskripsi Hasil Website

Website yang dibangun memiliki tampilan yang sederhana. Halaman *website* bagian atas adalah tempat *user* memasukkan *input-input* yang diinginkan. Terdapat beberapa *input* yang perlu dimasukkan, yaitu judul artikel awal, judul artikel tujuan, algoritma yang digunakan dalam pencarian, serta banyaknya *path* yang ingin dicari. Halaman *website* bagian bawah adalah tempat untuk menampilkan hasil pencarian. Beberapa *output* yang dihasilkan adalah jumlah artikel yang diperiksa, jumlah artikel yang dilalui, waktu pencarian, serta rute-rute penjelajahan artikel.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Penyelesaian Masalah

Masalah yang ingin diselesaikan oleh program adalah mencari rute terpendek dari satu artikel wikipedia menuju suatu artikel wikipedia lain. Sedangkan tahapan penyelesaian dari program ini adalah dengan menelusuri graf dinamis (graf yang terbentuk saat proses pencarian dilakukan), dimana yang menjadi simpul pada graf adalah laman wikipedia, sedangkan sisi yang menghubungkan satu simpul ke simpul lain, misalnya simpul A ke simpul B, menyatakan bahwa suatu laman wikipedia A memiliki pranala yang langsung mengarahkan ke laman wikipedia B. Graf dinamis yang terbentuk disini adalah graf berarah, karena bila laman wikipedia A memiliki pranala menuju laman wikipedia B, belum tentu laman B juga memiliki pranala menuju laman A.

Langkah penyelesaian masalah dengan menggunakan algoritma IDS (*Iterative Deepening Search*) adalah sebagai berikut:

1) Buat dua *channel* yang berbeda, yaitu channel *resultPaths* dan channel *done*. Channel *resultPaths* digunakan untuk menyimpan semua solusi *path* yang ditemukan, sedangkan channel *done* digunakan bagi fungsi IDS untuk menunggu hingga semua *goroutine* selesai dieksekusi demi mendapatkan semua solusi pada kedalaman tertentu.
2) Buat sebuah fungsi yang akan dijalankan sebagai *goroutine*, misalnya fungsi *f*. Fungsi *f* inilah yang menjadi inti dari algoritma IDS, yang bertugas memanggil fungsi DLS demi mendapatkan solusi.
3) Di dalam fungsi *f* ini, inisialisasi nilai *maxDepth* = 0 lalu panggil fungsi DLS dengan kedalaman maksimumnya adalah *maxDepth*. Fungsi DLS akan mencari solusi, lalu memasukkan setiap solusi yang ditemukannya ke channel *resultPaths*. Bila setelah fungsi DLS selesai dieksekusi ditemukan solusi, maka *goroutine* mengirimkan *signal* ke channel *done* untuk menandai bahwa semua solusi telah ditemukan pada *depth* tertentu. Bila masih belum, maka *maxDepth* di-*increment* dengan 1 lalu memanggil kembali fungsi DLS hingga ditemukan solusi.
4) Di luar fungsi *f* yang dijalankan sebagai *goroutine*, fungsi IDS akan menunggu hingga *channel* terisi. Bila yang dicari hanya satu *path* (*single solution*), maka IDS hanya menunggu DLS mengirimkan satu saja hasil *path* solusi ke *resultPath* lalu langsung

me-*return path* tersebut. Bila yang dicari adalah beberapa path (*multiple solution*), maka IDS menunggu channel *done* terisi terlebih dahulu kemudian me-*return* semua *path* yang ada di dalam *resultPath*.

Langkah penyelesaian masalah dengan menggunakan algoritma BFS (*Breadth-First Search*) adalah sebagai berikut:

1) Pencarian akan dimulai dari artikel pertama, semua pranala yang menuju ke artikel lain dalam *main body* artikel tersebut akan diambil dan disimpan dalam sebuah *channel* queue, jika terdapat pranala artikel yang dituju dalam artikel tersebut, akan disimpan dalam *channel* resultTree.
2) Setelah pencarian telah selesai dilakukan pada artikel awal, pranala artikel akan disimpan dalam *hash map* visited untuk menandai bahwa artikel tersebut telah dikunjungi.
3) Proses ini akan diulangi untuk semua pranala yang terdapat dalam queue yang belum dikunjungi sampai ditemukan solusi. Untuk mempercepat proses pencarian, digunakan *goroutine* untuk menjalankan 100 fungsi pencarian dalam waktu yang bersamaan.
4) Untuk pencarian *single solution*, pencarian akan dihentikan ketika ditemukan satu rute menuju artikel tujuan. Setelah rute ditemukan, boolean *found* akan di-set menjadi *true* untuk memberi sinyal kepada fungsi untuk menghentikan pencarian.
5) Untuk pencarian *multiple solution*, pencarian akan dihentikan ketika terdapat artikel yang dicari yang memiliki kedalaman berbeda dengan solusi yang sudah ada, nilai boolean *doneBFS* akan di-set menjadi *true* untuk memberi sinyal kepada fungsi untuk menghentikan pencarian.
6) Jika rute yang dicari telah ditemukan, fungsi akan mengembalikan jumlah artikel yang diperiksa, jumlah artikel yang dilalui, dan rute pencarian.

Sebelum memanggil fungsi IDS ataupun BFS, program akan mengecek terlebih dahulu apakah masukan judul A dan judul B sama. Bila sama, maka website akan langsung menunjukkan rute dari A ke dirinya sendiri tanpa perlu memanggil algoritma IDS maupun BFS.

3.2 Pemetaan Masalah menjadi Elemen-elemen Algoritma

Dalam permainan *Wikirace*, permasalahan-permasalahan yang ada adalah sebagai berikut:

1) Artikel, merupakan entri teks yang terdapat dalam website https://en.wikipedia.com.
2) Artikel awal, merupakan artikel tempat dimulainya pencarian.
3) Artikel akhir, merupakan artikel yang dicari.
4) Rute, merupakan rute yang diambil untuk mencapai artikel akhir dari artikel awal.

Algoritma pencarian *Breadth-First Search* (BFS) memiliki elemen-elemen berikut:

1) Simpul, entitas dasar yang membentuk sebuah graf.
2) Graf, kumpulan simpul-simpul yang berhubungan, tempat dilakukannya pencarian.
3) Antrian (*queue*), merupakan himpunan yang berisi simpul-simpul yang akan dikunjungi.
4) Dikunjungi (*visited*), merupakan himpunan yang berisi simpul-simpul yang telah dikunjungi.

Sedangkan algoritma pencarian *Iterative Deepening Search* (IDS) memiliki elemen-elemen berikut:

1) Simpul, entitas dasar yang membentuk sebuah graf.
2) Graf, kumpulan simpul-simpul yang berhubungan, tempat dilakukannya pencarian.
3) Dikunjungi (*visited*), merupakan himpunan yang berisi simpul-simpul yang telah dikunjungi.

Sehingga dapat dipetakan permasalahan-permasalahan permainan *Wikirace* menjadi elemen-elemen algoritma BFS dan IDS sebagai berikut:

No	Permasalahan	Elemen BFS	Elemen IDS
1.	Artikel	Dipetakan sebagai simpul, yang akan disimpan dalam Graf, *queue*, dan *visited*.	Dipetakan sebagai simpul, yang akan disimpan dalam Graf dan *visited*.
2.	Artikel awal	Digunakan sebagai tempat awal pencarian, disimpan sebagai simpul awal dari Graf.	Digunakan sebagai tempat awal pencarian, disimpan sebagai simpul awal dari Graf.

3.	Artikel akhir	Digunakan untuk menandai bahwa pencarian telah berakhir, disimpan sebagai simpul akhir dari Graf.	Digunakan untuk menandai bahwa pencarian telah berakhir.
4.	Rute	Setiap simpul menyimpan rute yang telah diambilnya dalam variabel *prev*.	Setiap fungsi DLS menyimpan rute yang telah diambilnya sejauh ini dalam variabel *path*.

3.3 Fitur Fungsional & Arsitektur Website

Website yang kami buat menawarkan beberapa fungsionalitas, yaitu:

1. Web dapat mencari rute terpendek dari suatu halaman wikipedia ke halaman wikipedia lainnya dengan algoritma **BFS** serta menampilkan **salah satu** rute yang ditemukan.
2. Web dapat mencari rute terpendek dari suatu halaman wikipedia ke halaman wikipedia lainnya dengan algoritma **BFS** serta menampilkan **semua** rute yang ditemukan.
3. Web dapat mencari rute terpendek dari suatu halaman wikipedia ke halaman wikipedia lainnya dengan algoritma **IDS** serta menampilkan **salah satu** rute yang ditemukan.
4. Web dapat mencari rute terpendek dari suatu halaman wikipedia ke halaman wikipedia lainnya dengan algoritma **IDS** serta menampilkan **semua** rute yang ditemukan.

Arsitektur web tersebut terdiri atas 2 bagian utama, yaitu *frontend* dan *backend*. Untuk *frontend*, kami menggunakan HTML, CSS, serta Javascript. Kami tidak menggunakan *library* atau *framework* lain selain fitur-fitur bawaan yang disediakan ketiga bahasa tersebut. Untuk backend, kami menggunakan bahasa Go/Golang. Website tersebut hanya dijalankan secara *local* menggunakan fitur *http server* yang disediakan oleh Go.

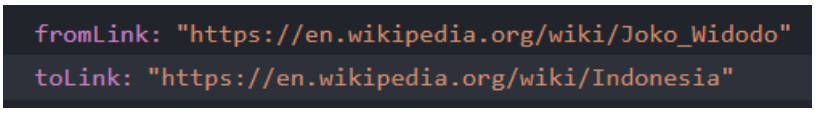
3.4 Contoh Ilustrasi Kasus

Sebagai contoh ilustrasi kasus, misalkan akan dicari rute terpendek dari laman Joko Widodo menuju laman Indonesia.

Gambar 3.4.1 Contoh Ilustrasi Kasus dari Joko Widodo menuju Indonesia

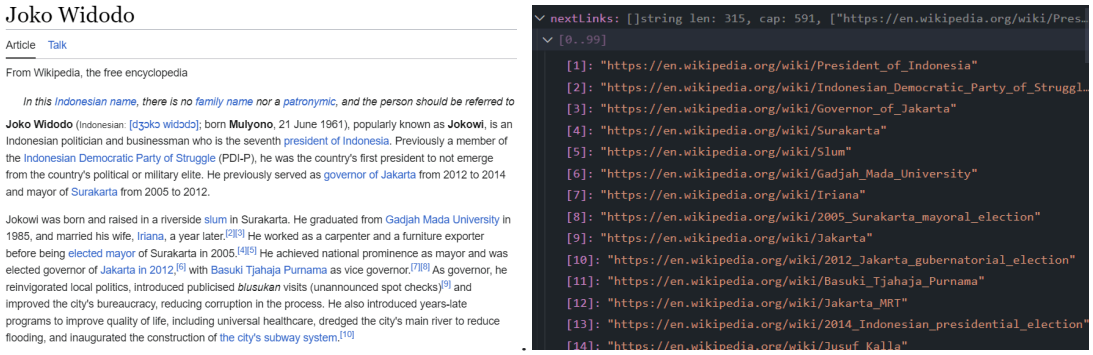
Maka alur dari keberjalanan program adalah sebagai berikut:

1) Program mengkonversi input judul laman menjadi pranala laman asal dan pranala laman tujuan.



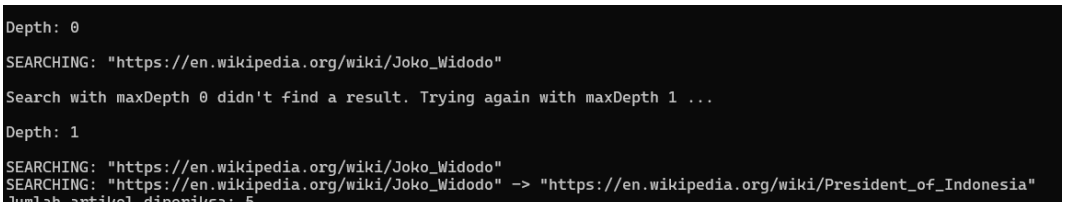
Gambar 3.4.2 Hasil Konversi Link Asal dan Link Tujuan

2) Program mengunjungi laman Joko Widodo, lalu melakukan *scraping* untuk mengambil semua pranala yang ada di dalam konten dari laman tersebut. Yang dimaksud pranala di dalam konten laman disini adalah pranala yang terdapat di dalam *div mw-content-text*, sehingga link yang tidak terlihat pada laman tidak ikut diperhitungkan.



Gambar 3.4.3 Program melakukan scraping pada laman Joko Widodo

3) Hasil dari *scraping* disimpan dalam struktur data *Tree*, dengan semua pranala yang ada di laman Joko Widodo sebagai *child node* dari laman Joko Widodo.
4) Program mengecek apakah pranala tujuan telah didapatkan pada hasil scraping laman tersebut. Karena pranala tujuan (https://en.wikipedia.org/wiki/Indonesia) tidak terdapat dalam hasil scraping laman Joko Widodo, maka algoritma tetap mencari dengan menambah kedalaman dari kedalaman 0 menuju kedalaman 1.
    - Bila menggunakan algoritma IDS, maka pencarian dimulai lagi dari kedalaman 0, yaitu dari laman Joko Widodo menuju semua *child node* nya.



Gambar 3.4.3 Algoritma IDS yang mengulang pencarian dari depth 0

- Bila menggunakan algoritma BFS, maka hasil scraping tersebut dimasukkan ke dalam queue untuk langsung diproses tanpa harus kembali ke kedalaman sebelumnya.

Gambar 3.4.4 Algoritma BFS yang tidak mengulang pencarian dari depth 0

5) Pada kedalaman 1, program akan menemukan link tujuan dari salah satu *child node* laman Joko Widodo. Program kemudian menyimpan rute yang telah dilalui menuju link tujuan tersebut.
    - Bila memilih hasil single path, maka algoritma berhenti dan program akan langsung mengirimkan hasil rute ini menuju website.
    - Bila memilih hasil multiple path, maka algoritma tetap berjalan hingga ditemukan semua rute solusi pada kedalaman yang sama.
6) Rute solusi yang telah didapatkan ditampilkan pada website.

Gambar 3.4.5 Tampilan hasil rute solusi untuk *single path* dan *multiple path*

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

Secara keseluruhan program, program terdiri dari 5 file .go, yaitu:

4.1.1 bfs.go, yang terdiri atas struktur data berikut:

No.	Struktur Data	Fungsi
1	InitTree(currentTree *Tree, c chan Tree, singleSolution bool)	Mengambil semua pranala dalam suatu artikel dan membuat sebuah Tree dari tiap-tiap pranala tersebut. Serta mengembalikan *true* jika pranala belum pernah dikunjungi sebelumnya.
2	BFS(from string, to string, singleSolution bool)	Melakukan pencarian artikel dengan judul *to* dari artikel berjudul *from* menggunakan algoritma BFS, mengembalikan jumlah artikel yang diperiksa, jumlah artikel yang dilalui, dan rute yang diambil dari artikel awal menuju artikel akhir.
3	isEqualSlice(s1, s2 []string)	Mengembalikan *true* jika dua buah slice merupakan slice yang sama.
4	isInSlice(s1 [][]string, s2 []string)	Mengembalikan *true* jika s2 terdapat di dalam s1.

bfs.go

```go
func initTree(currentTree *Tree, c chan Tree, singleSolution bool) bool {
    if getFound() {
        return false
    }

    var links []string

    // Scraping link & judul
    linkScraping(currentTree.link, &links, &currentTree.judul)
    // Cek apakah halaman sudah pernah di-visit
    if visitedAsync.keyExists(currentTree.link) {
        return false
    }

    // Masukkan semua link yang di-scrape ke dalam nextArr
    for _, link := range links {
        newTree := Tree{
            prev: append(currentTree.prev, currentTree.link),
            link:    link,
            nextArr: []*Tree{},
            depth:   currentTree.depth + 1,
        }

        // Jika link sama dengan link yang di cari, return
        if link == linkTujuan {
            newTree.judul = LinkTojudul(linkTujuan)
            if singleSolution {
                if !getFound() {
                    setFound(true)

                    select {
                    case c <- newTree:
                    default:
                    }
                }
            } else {
                if len(c) < 1 {
                    if getPJ() == -99  {
                        setPJ(newTree.depth)
                    }
                    select {
                    case c <- newTree:
                    default:
                    }
                } else {
                    if getPJ() == newTree.depth {
                        select {
                        case c <- newTree:
                        default:
                        }
                    }
                }
            }

            return false
        }

        currentTree.nextArr = append(currentTree.nextArr, &newTree)
    }
    return true
```

```go
func BFS(from string, to string, singleSolution bool) (int, int, [][]string) {
    // Return jika judul asal dan tujuan sama
    if (from == to) {
        return 0, 0, [][]string{{from}}
    }

    // Inisialisasi
    var wg sync.WaitGroup

    setFound(false)
    setDone(false)
    panjangRute = -99
    linkAsal = judulToLink(from)
    linkTujuan = judulToLink(to)

    visitedAsync = newStringBoolMap()

    a := Tree{
        prev: []string{},
        link:   linkAsal,
        nextArr: []*Tree{},
        depth: 0,
    }

    queueA := make(chan *Tree, 10000000)
    resultTree := make(chan Tree, 10000)

    initTree(&a, resultTree, singleSolution)
    for _, val := range a.nextArr {
        queueA <- val
    }

    visitedAsync.set(linkAsal)
```

```go
    // Start
    setCnt(0)
    for {
        wg.Add(100)
        for i := 0; i < 100; i++ {
            go func() {
                defer wg.Done()

                currentTree, isOpen := <- queueA
                if (!isOpen) {
                    return
                }

                // Return jika sudah di visit / sudah ketemu
                isNotVisited := initTree(currentTree, resultTree, singleSolution)
                if !isNotVisited {
                    return
                } else {
                    visitedAsync.set(currentTree.link)
                    incCnt()

                    if singleSolution {
                        if getFound() {
                            return
                        }
                    } else {
                        if getDone() {
                            return
                        }

                        if getPJ() != -99 {
                            if currentTree.depth >= getPJ() {
                                setDone(true)
                                return
                            }
                        }
```

```
            // Print
            fmt.Print("Searching: ")
            for _, val := range currentTree.prev {
                fmt.Print(val)
                fmt.Print(" -> ")
            }
            fmt.Println(currentTree.link, len(queueA), len(visitedAsync.Map))

            // Set found ke true kalo judul sama dg to
            if (currentTree.judul == to) {
                if singleSolution {
                    if !getFound() {
                        setFound(true)
                        select {
                        case resultTree <- *currentTree:
                        default:
                        }
                    }
                } else {
                    if len(resultTree) == 0 {
                        if getPJ() == -99 {
                            fmt.Println("setPJ", currentTree.link, currentTree.depth)
                            setPJ(currentTree.depth)
                        }

                        select {
                        case resultTree <- *currentTree:
                        default:
                        }
                    } else {
                        if currentTree.depth == getPJ() {
                            select {
                            case resultTree <- *currentTree:
                            default:
                            }
                        } else {
                            }
                        } else {
                            setDone(true)
                        }
                    }
                }
                return
            }

            // Masukkan link dalam halaman kedalam queue, asumsi queue tidak akan penuh
            for _, val := range currentTree.nextArr {
                if !visitedAsync.keyExists(val.link) {
                    select {
                    case queueA <- val:
                    default:
                        fmt.Println("Channel penuh")
                    }
                }
            }

        } ()
    }

    wg.Wait()

    if singleSolution {
        if getFound() {
            break
        }
    } else {
        if getDone() {
            break
        }
    }
}
```

```
    // Output
    if singleSolution {
        resT := <- resultTree
        path := append(resT.prev, resT.link)
        return cntAsync, resT.depth, [][]string{path}
    } else {
        allPath := [][]string{}
        lenC := len(resultTree)

        for i := 0; i < lenC; i++ {
            val := <-resultTree
            path := append(val.prev, val.link)
            if !isInSlice(allPath, path) {
                allPath = append(allPath, path)
            }
        }

        return cntAsync, getPJ(), allPath
    }
}

func isEqualSlice(s1, s2 []string) bool {
    if len(s1) != len(s2) {
        return false
    }

    for i, val := range s1 {
        if val != s2[i] {
            return false
        }
    }
    return true
}
```

```
func isInSlice(s1 [][]string, s2 []string) bool {
    if len(s1) == 0 {
        return false
    }

    for _, val := range s1 {
        if isEqualSlice(val, s2) {
            return true
        }
    }
    return false
}
```

4.1.2 ids.go, yang terdiri atas struktur data berikut

No.	Struktur Data	Fungsi
1	IDS(fromTitle string, judulArtikelTujuan string, singleSolution bool) (int, int, [][]string)	Memanggil fungsi DLS untuk melakukan pencarian solusi dengan batas kedalaman yang meningkat, serta menerima hasil solusi dari fungsi DLS melalui channel
2	DLS(currentLink string, depth int, path []string, wg *sync.WaitGroup)	Melakukan pencarian artikel secara *depth-first* dengan nilai depth tertentu, dan memanggil dirinya sendiri secara rekursif sebagai *goroutine* untuk mencari solusi pada *depth* selanjutnya. Bila menemukan solusi, maka akan mengirimkan

		solusi tersebut kepada fungsi IDS melalui channel

ids.go

```
func IDS(fromTitle string, judulArtikelTujuan string, singleSolution bool) (int, int, [][]string) {
    if (fromTitle == toTitle) {
        return 0, 0, [][]string{{fromTitle}}
    }

    fromLink    = judulToLink(fromTitle)
    toLink      = judulToLink(judulArtikelTujuan)
    toTitle     = judulArtikelTujuan
    resultPaths = make(chan []string, 1000)
    done        = make(chan bool)
    cache       = newIDSTree()
    found       = false

    defer func() {
        if err := recover(); err != nil {
            fmt.Println("panic occurred:", err)
        }
    }()

    go func() {
        maxDepth := 0
        for !found {
            fmt.Println("\nDepth:", maxDepth)
            fmt.Println()

            visitedLinks = newStringBoolMap()
            var wg sync.WaitGroup
            wg.Add(1)

            DLS(fromLink, maxDepth, []string{}, &wg)

            if (!found) {
                fmt.Println("\nSearch with maxDepth", maxDepth, "didn't find a result. Trying again with maxDepth", maxDepth+1, "...")
                maxDepth++
            } else if (!singleSolution) {
                done <- true
```

```
                done <- true
            }
        }
    } ()

    // If searching for multiple paths, wait until all goroutines finishes
    if (!singleSolution) {
        <- done
    }

    var result [][]string
    result = append(result, <- resultPaths)
    close(resultPaths)
    close(done)

    for i := 0; i < len(resultPaths); i++ {
        var path = <- resultPaths
        result = append(result, path)
    }

    return len(visitedLinks.Map), len(result[0]) - 1, result
}


func DLS(currentLink string, depth int, path []string, wg *sync.WaitGroup) {
    defer func() {
        if err := recover(); err != nil {
            // do nothing
        }
    }()
    defer wg.Done()

    currentJudul := LinkTojudul(currentLink)

    if (visitedLinks.get(currentJudul)) {
```

```
if (visitedLinks.get(currentJudul)) {
    return
}
visitedLinks.set(currentJudul)

var nextLinks []string = cache.get(currentJudul)
if (len(nextLinks) == 0) {
    linkScraping(currentLink, &nextLinks, &currentJudul)
    cache.set(currentJudul, nextLinks)
}

path = append(path, currentLink)

fmt.Print("SEARCHING: \"" + path[0] + "\"")
for i := 1; i < len(path); i++ {
    fmt.Print(" -> \"" + path[i] + "\"")
}
fmt.Println()

if (slices.Contains(nextLinks, toLink)) {
    path = append(path, toLink)
    found = true
    resultPaths <- path

    return
}

if (currentJudul == toTitle) {
    found = true
    resultPaths <- path

    return
}

if (depth == 0) {
```

```
  if (depth == 0) {
      return
  }

  xthreads := 3
  var linkChannel = make(chan string, len(nextLinks))
  var currentWg sync.WaitGroup
  currentWg.Add(len(nextLinks) + xthreads)


  for i := 0; i < xthreads; i++ {
      go func() {
          for {
              nextLink, isOpen := <- linkChannel
              if (!isOpen) {
                  currentWg.Done()
                  return
              }
              DLS(nextLink, depth - 1, path, &currentWg)
          }
      } ()
  }

  for _, nextLink := range nextLinks {
      linkChannel <- nextLink
  }

  close(linkChannel)
  currentWg.Wait()
}
```

4.1.3 scraper.go, yang terdiri atas struktur data berikut

No.	Struktur Data	Fungsi

1	linkScraping(link string, links *[]string, judul *string)	Mengambil judul artikel dan seluruh pranala artikel yang terdapat dalam *main body* artikel.
2	convertJudul(judul string)	Mengembalikan judul yang telah di *encode*.
3	judulToLink(judul string)	Mengembalikan link wikipedia untuk judul artikel yang dimasukkan.
4	LinkTojudul(link string)	Mengembalikan judul artikel wikipedia dari link yang diberikan.

scraper.go

```go
func getFound() bool {
    foundMutex.Lock()
    defer foundMutex.Unlock()
    return foundAsync
}

func setCnt(val int) {
    cntMutex.Lock()
    defer cntMutex.Unlock()
    cntAsync = val
}

func incCnt() {
    cntMutex.Lock()
    defer cntMutex.Unlock()
    cntAsync++
}

func newStringBoolMap() stringBoolMap {
    return stringBoolMap{map[string]bool{}, sync.RWMutex{}}
}

func (m *stringBoolMap) get(key string) bool {
    m.RLock()
    defer m.RUnlock()
    return m.Map[key]
}

func (m *stringBoolMap) set(key string) {
    m.Lock()
    defer m.Unlock()
    m.Map[key] = true
}
```

```go
/*
    Encode judul untuk dipake di url
*/
func convertJudul(judul string) string {
    encoded := url.QueryEscape(judul)
    encoded = strings.ReplaceAll(encoded, "+", "_")
    return encoded
}

/*
    Ubah judul menjadi link en.wikipedia.org
*/
func judulToLink(judul string) string {
    encoded := convertJudul(judul)
    return "https://en.wikipedia.org/wiki/" + encoded
}

/*
    Ubah link menjadi judul artikel
*/
func LinkTojudul(link string) string {
    encoded := strings.TrimPrefix(link, "https://en.wikipedia.org/wiki/")
    encoded = strings.ReplaceAll(encoded, "_", "+")
    decoded, err := url.QueryUnescape(encoded)
    if err == nil {
        return decoded
    } else {
        // harusnya ga pernah gagal
        return "GAGAL DECODE"
    }
}
```

4.1.4 structs.go, yang terdiri atas struktur data berikut:

No.	Struktur Data	Fungsi
1	Tree	Berisi rute, judul artikel, link artikel, pointer menuju tree berikutnya, dan kedalaman.
2	stringBoolMap	Berisi map dengan key berupa judul, dan value berupa boolean yang menyatakan judul tersebut sudah dikunjungi atau belum.
3	IDSTree	Berisi map dengan key berupa judul, dan value berupa array string yang merupakan child dari judul tersebut.
4	setPJ(val int)	Mengubah nilai global variabel panjangRute menjadi val jika panjangRute adalah -99.

5	getPJ()	Mengembalikan nilai global variabel panjangRute
6	setDone(val bool)	Mengubah nilai global variabel doneBFS menjadi val.
7	getDone()	Mengembalikan nilai global variabel doneBFS.
8	setFound(val bool)	Mengubah nilai global variabel foundAsync menjadi val.
9	getFound()	Mengembalikan nilai global variabel foundAsync.
10	setCnt()	Mengubah nilai global variabel cntAsync menjadi val.
11	incCnt()	Menambahkan nilai global variabel cntAsync dengan satu.
12	newStringBoolMap()	Mengembalikan struct stringBoolMap kosong.
13	(m *stringBoolMap) get(key string)	Mengembalikan nilai pada stringBoolMap dengan key *key*.
14	(m *stringBoolMap) set(key string)	Menambahkan entri baru dalam stringBoolMap dengan key *key*.
15	(m *stringBoolMap) keyExists(key string)	Mengembalikan true jika terdapat entri pada stringBoolMap dengan key tersebut.
16	newIDSTree()	Mengembalikan struct IDSTree kosong.
17	(m *IDSTree) get(key string)	Mengembalikan nilai pada IDSTree dengan key *key.*
18	(m *IDSTree) set(key string, value []string)	Menambahkan entri baru dalam IDSTree dengan key *key*.

structs.go

```go
// Struct
type Tree struct {
    prev    []string
    judul   string
    link    string
    nextArr []*Tree
    depth   int
}

type stringBoolMap struct {
    Map map[string]bool
    sync.RWMutex
}

type IDSTree struct {
    Map map[string][]string
    sync.RWMutex
}
```

```go
// Method
func setPJ(val int) {
    pjMutex.Lock()
    defer pjMutex.Unlock()
    if panjangRute == -99 {
        panjangRute = val
    }
}

func getPJ() int {
    pjMutex.Lock()
    defer pjMutex.Unlock()
    return panjangRute
}

func setDone(val bool) {
    doneMutex.Lock()
    defer doneMutex.Unlock()
    doneBFS = val
}

func getDone() bool {
    doneMutex.Lock()
    defer doneMutex.Unlock()
    return doneBFS
}


func setFound(val bool) {
    foundMutex.Lock()
    defer foundMutex.Unlock()
    foundAsync = val
}
```

```go
func getFound() bool {
    foundMutex.Lock()
    defer foundMutex.Unlock()
    return foundAsync
}

func setCnt(val int) {
    cntMutex.Lock()
    defer cntMutex.Unlock()
    cntAsync = val
}

func incCnt() {
    cntMutex.Lock()
    defer cntMutex.Unlock()
    cntAsync++
}

func newStringBoolMap() stringBoolMap {
    return stringBoolMap{map[string]bool{}, sync.RWMutex{}}
}

func (m *stringBoolMap) get(key string) bool {
    m.RLock()
    defer m.RUnlock()
    return m.Map[key]
}

func (m *stringBoolMap) set(key string) {
    m.Lock()
    defer m.Unlock()
    m.Map[key] = true
}
```

```go
func (m *stringBoolMap) keyExists(key string) bool {
    m.RLock()
    defer m.RUnlock()

    _, ok := m.Map[key]
    return ok
}

func newIDSTree() IDSTree {
    return IDSTree{map[string][]string{}, sync.RWMutex{}}
}

func (m *IDSTree) get(key string) []string {
    m.RLock()
    defer m.RUnlock()
    return m.Map[key]
}

func (m *IDSTree) set(key string, value []string) {
    m.Lock()
    defer m.Unlock()
    m.Map[key] = value
}
```

4.1.5 main.go, yang terdiri atas struktur data berikut:

No.	Struktur Data	Fungsi
1	processHandler(w http.ResponseWriter, r *http.Request)	Menangani http request, memanggil fungsi pencarian, serta menangani response dari backend ke frontend
2	main()	Menjalankan *flow* utama program serta menjalankan local web server

main.go

```go
func processHandler(w http.ResponseWriter, r *http.Request) {
    // Set CORS headers
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS")
    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    // Handle preflight request
    if r.Method == http.MethodOptions {
        w.WriteHeader(http.StatusNoContent)
        return
    }

    // Parse JSON data from request body
    var requestData map[string]string
    if err := json.NewDecoder(r.Body).Decode(&requestData); err != nil {
        http.Error(w, "Failed to parse request body", http.StatusBadRequest)
        return
    }

    // Get all components of the received data
    fmt.Println(requestData) // testing
    startPage := requestData["start-page"]
    endPage := requestData["end-page"]
    algorithm := requestData["algorithm"]
    numberOfPath := requestData["number-of-path"]

    // Find path

    var (
        jumlahArtikelDiperiksa int
        jumlahArtikelDilalui   int
        routes                 [][]string
    )

    startTime := time.Now()
```

```go
startTime := time.Now()

if algorithm == "bfs" {
    if numberOfPath == "single" {
        fmt.Println(startPage, endPage, "BFS", "Single") // testing
        // single bfs
        jumlahArtikelDiperiksa, jumlahArtikelDilalui, routes = scraper.BFS(startPage, endPage, true)
    } else {
        fmt.Println(startPage, endPage, "BFS", "Multiple") // testing
        // multiple bfs
        jumlahArtikelDiperiksa, jumlahArtikelDilalui, routes = scraper.BFS(startPage, endPage, false)
    }
} else {
    if numberOfPath == "single" {
        fmt.Println(startPage, endPage, "IDS", "Single") // testing
        // single ids
        jumlahArtikelDiperiksa, jumlahArtikelDilalui, routes = scraper.IDS(startPage, endPage, true)
    } else {
        fmt.Println(startPage, endPage, "IDS", "Multiple") // testing
        jumlahArtikelDiperiksa, jumlahArtikelDilalui, routes = scraper.IDS(startPage, endPage, false)
    }
}

endTime := time.Now()
searchDuration := endTime.Sub(startTime).Milliseconds()

fmt.Println("Jumlah artikel diperiksa:", jumlahArtikelDiperiksa) // testing
fmt.Println("Jumlah artikel dilalui:", jumlahArtikelDilalui)     // testing
fmt.Println("Routes:")                                           // testing
fmt.Println(routes)                                              // testing
fmt.Println("Search duration:", searchDuration, "ms")            // testing
```

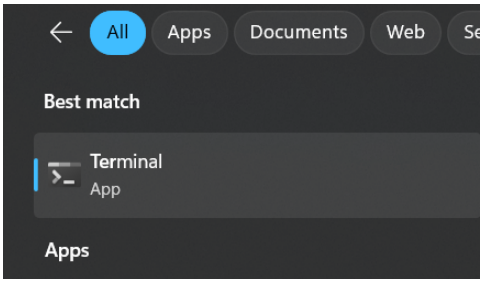
```
    // Convert routes to include titles
    routesWithTitle := make([][][]string, len(routes))
    for i, route := range routes {
        routesWithTitle[i] = make([][]string, len(route))
        for j, link := range route {
            routesWithTitle[i][j] = make([]string, 2)
            routesWithTitle[i][j][0] = link
            routesWithTitle[i][j][1] = scraper.LinkTojudul(link)
        }
    }

    fmt.Println("routesWithTitle:")
    for _, route := range routesWithTitle {
        fmt.Println(route)
    }

    // Send response
    response := map[string]any{
        "jumlahArtikelDiperiksa": jumlahArtikelDiperiksa,
        "jumlahArtikelDilalui":   jumlahArtikelDilalui,
        "routes":                 routesWithTitle,
        "searchDuration":         searchDuration,
    }

    // Encode response into JSON and send it
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(response)
}
```

```
func main() {
    // Get the path of the frontend directory from the backend directory
    frontendDir := filepath.Join("..", "frontend")

    // Serve static files from the frontend directory
    http.Handle("/", http.FileServer(http.Dir(frontendDir)))

    // Handle data endpoint
    http.HandleFunc("/data", processHandler)

    // Start HTTP server
    fmt.Println("Server running on localhost:8080")
    http.ListenAndServe(":8080", nil)
}
```

4.2 Tata Cara Penggunaan Program

Program dijalankan dengan cara sebagai berikut:

1. Buka terminal



Gambar 4.2.1 Buka terminal

2. Masuk ke dalam folder src/*backend* dalam terminal

```
C:\Users\DIERO PURNAMA\Documents\CODING\GO\Tubes2_wikiwikiwiki-1\src>cd backend

C:\Users\DIERO PURNAMA\Documents\CODING\GO\Tubes2_wikiwikiwiki-1\src\backend>
```

Gambar 4.2.2 Terminal buka backend

3. Jalankan perintah “go run .”

```
C:\Users\DIERO PURNAMA\Documents\CODING\GO\Tubes2_wikiwikiwiki-1\src\backend>go run .
Server running on localhost:8080
```

Gambar 4.2.3 Terminal go run

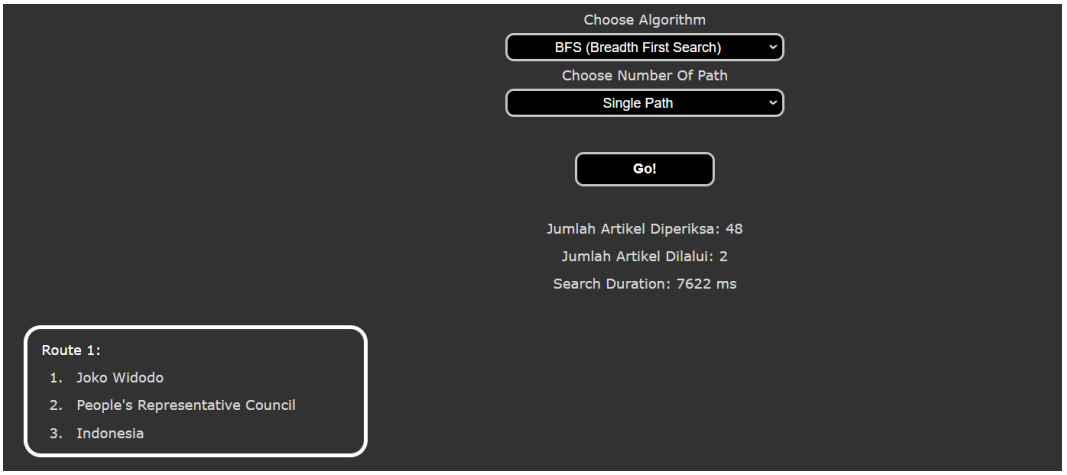
4. Buka “localhost:8080” dalam browser

Gambar 4.2.4 localhost:8080

5. Masukkan judul artikel awal dan judul artikel tujuan serta algoritma yang ingin dipakai.

Gambar 4.2.5 Input

6. Website akan menampilkan jumlah artikel yang dilalui, jumlah artikel yang diperiksa, durasi pencarian, serta rute yang diambil.



Gambar 4.2.6 Output

4.3 Hasil Pengujian

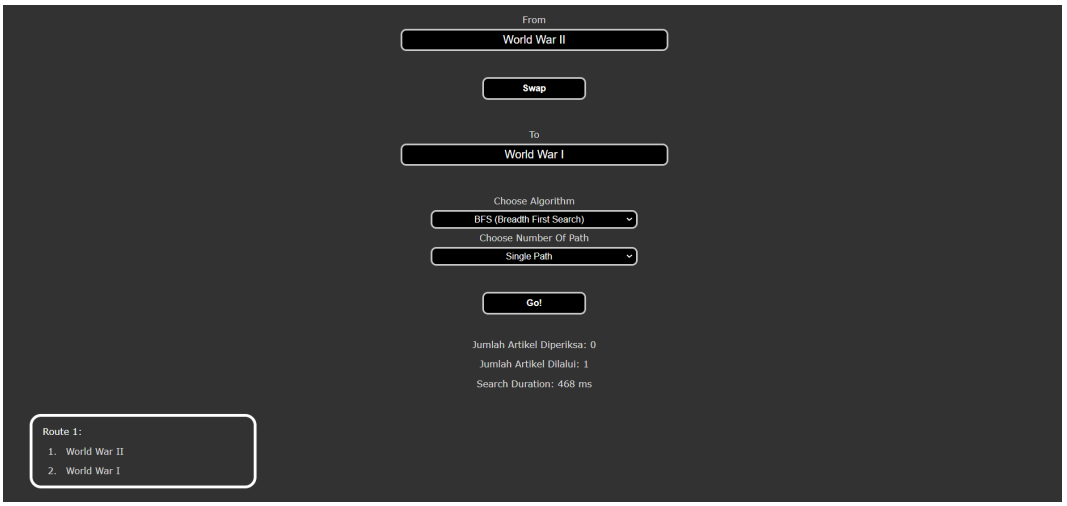
Untuk mengetes program kami, kami melakukan pengujian dengan berbagai data uji sebagai berikut:

Judul Asal	Judul Tujuan	Kedalaman (steps)

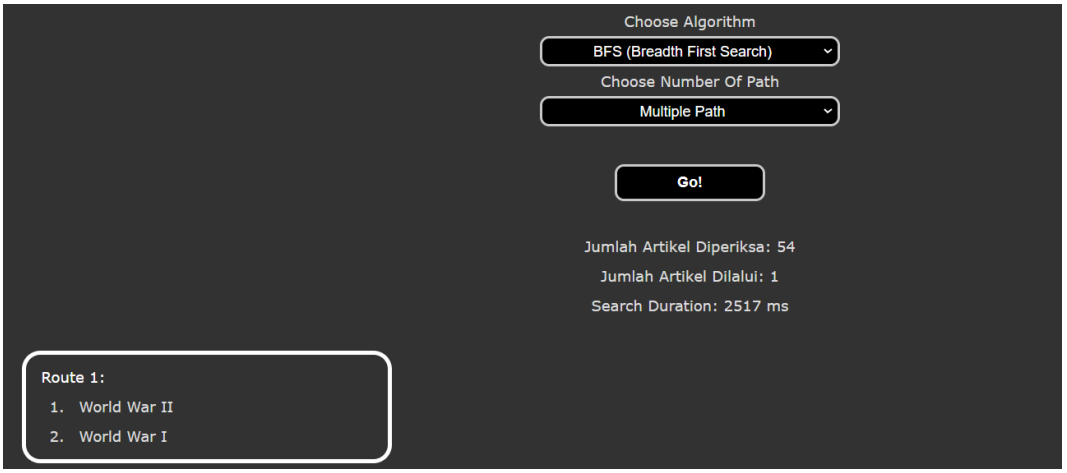
World War II	World War I	1
Joko Widodo	Indonesia	2
Paging	Vector space	3
Paging	Stardew Valley	4

a. Test Case 1

1) BFS



Gambar 4.3.1 BFS Single Path untuk Test Case 1



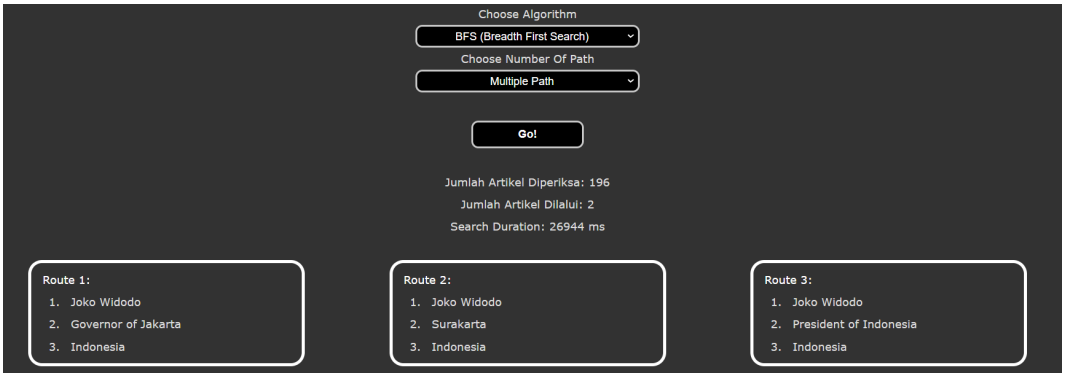
Gambar 4.3.2 BFS Multiple Path untuk Test Case 1

Gambar 4.3.3 IDS Single Path untuk Test Case 1

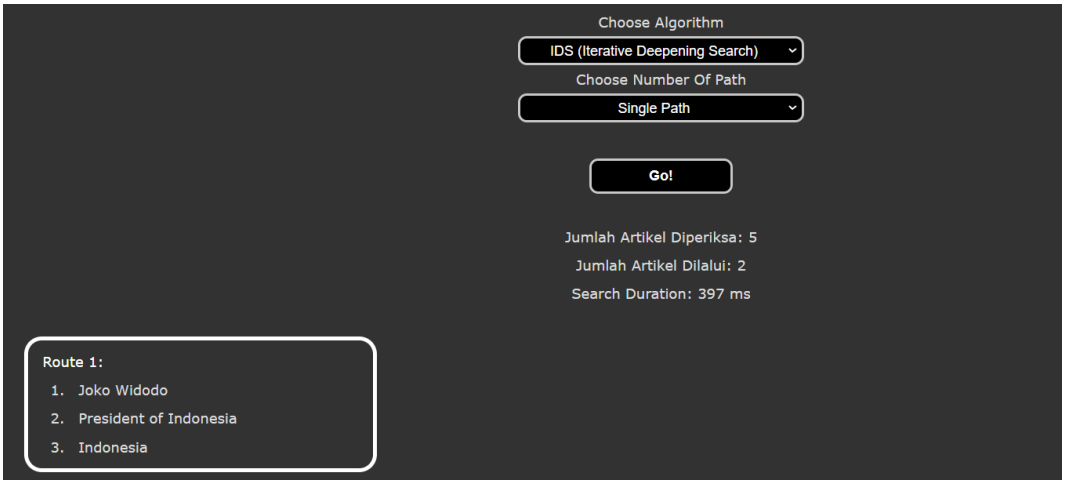
Gambar 4.3.4 IDS Multiple Path untuk Test Case 1

b. Test Case 2

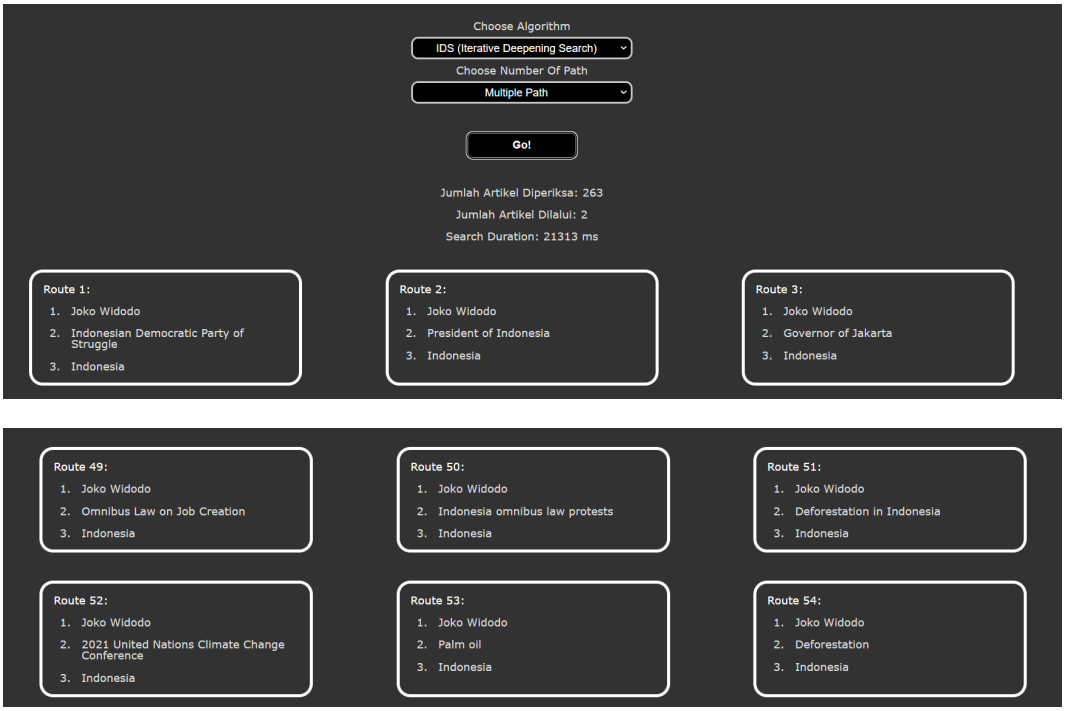
Gambar 4.3.5 BFS Single Path untuk Test Case 2



Gambar 4.3.6 BFS Multiple Path untuk Test Case 2

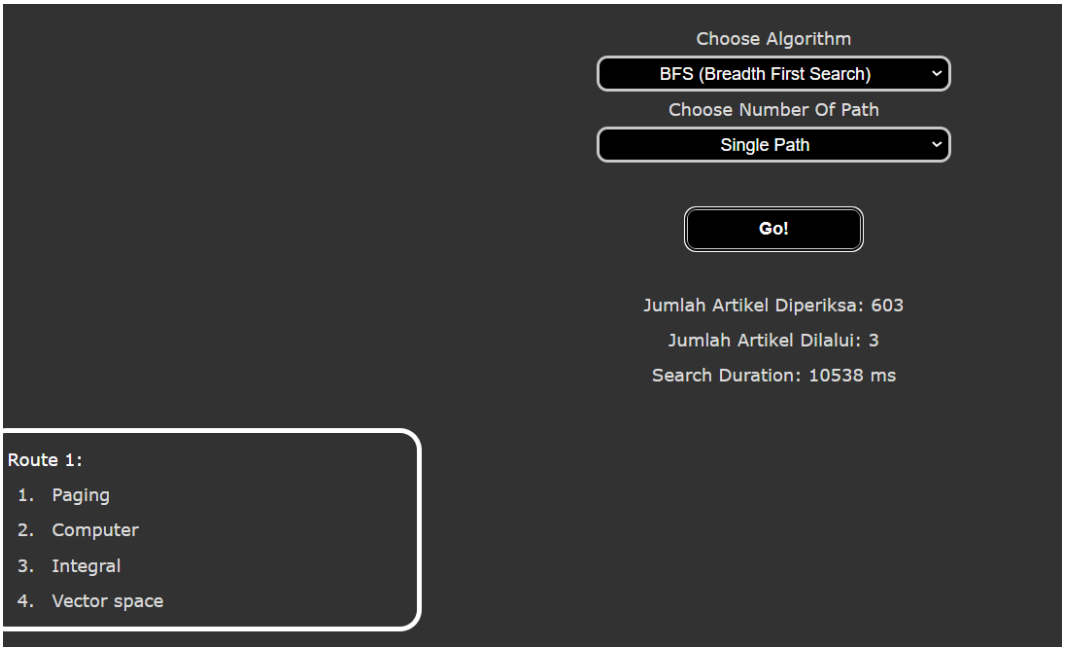


Gambar 4.3.7 IDS Single Path untuk Test Case 2

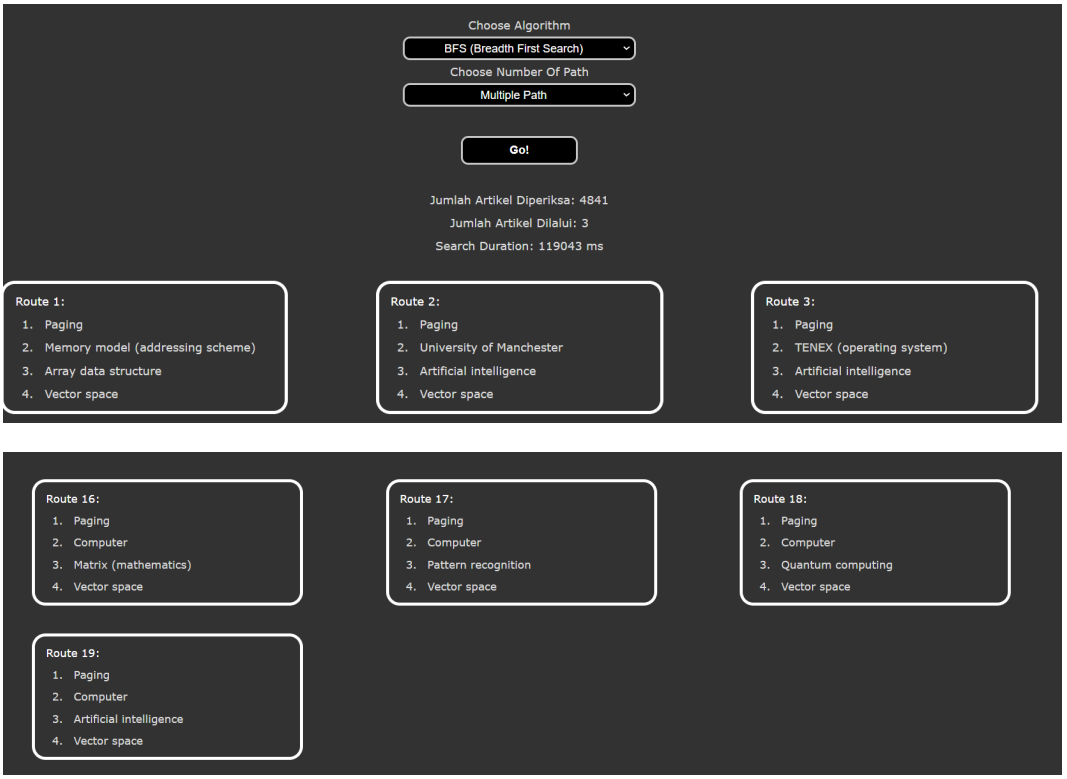


Gambar 4.3.8 IDS Multiple Path untuk Test Case 2

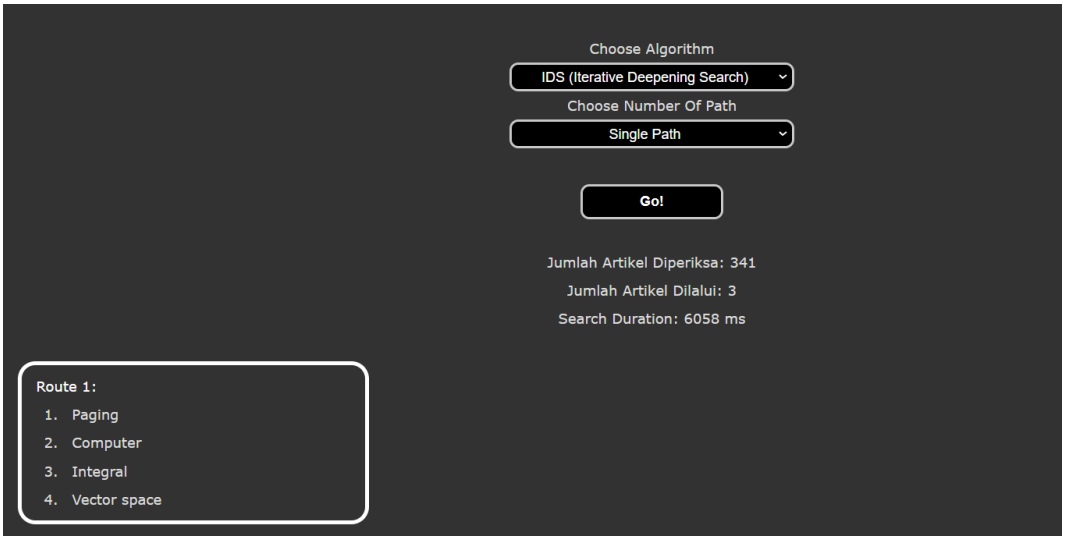
c. Test Case 3



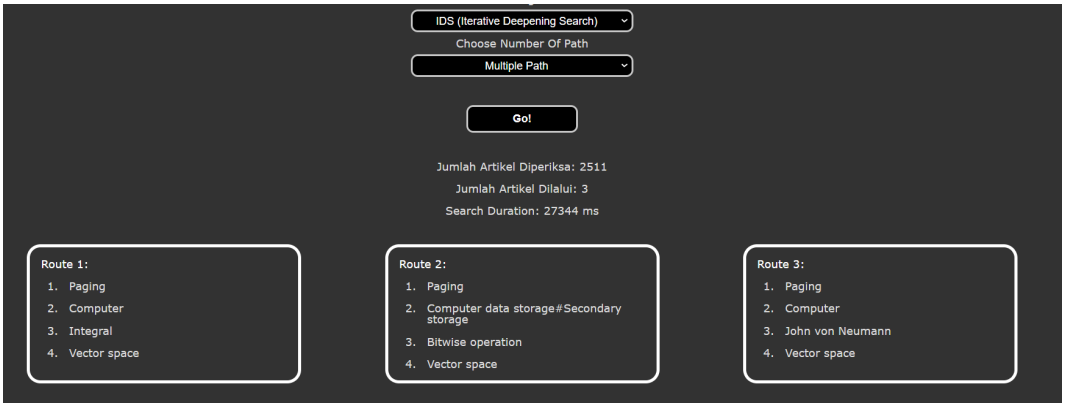
Gambar 4.3.9 BFS Single Path untuk Test Case 3



Gambar 4.3.10 BFS Multiple Path untuk Test Case 3

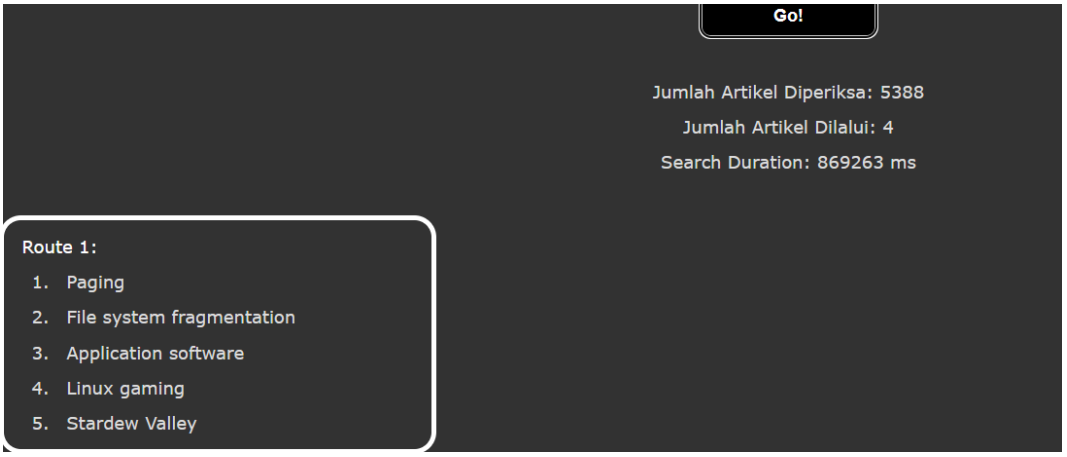


Gambar 4.3.11 IDS Single Path untuk Test Case 3

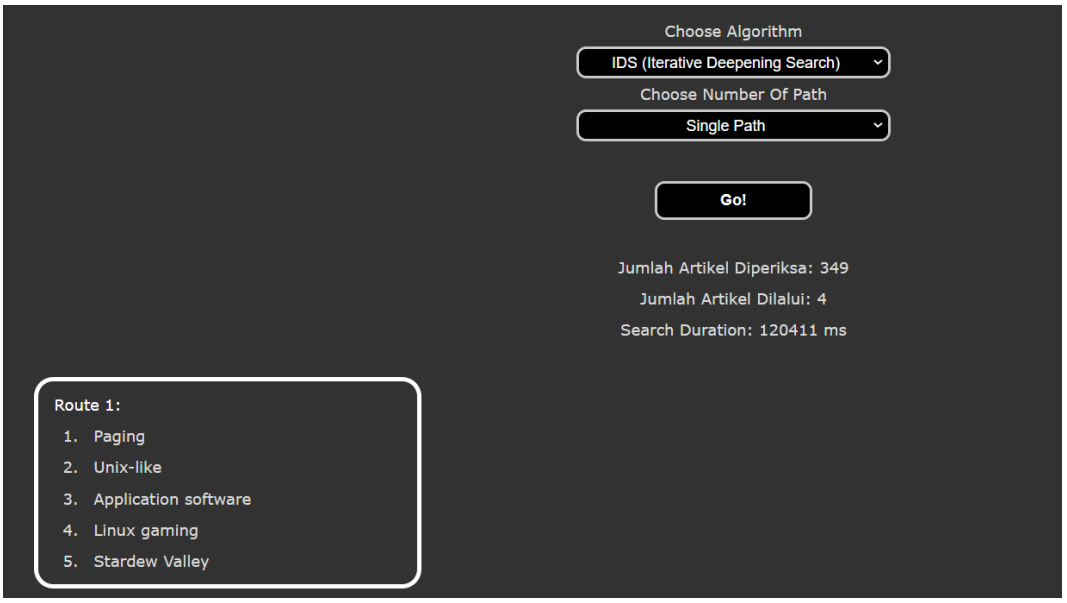


Gambar 4.3.12 IDS Multiple Path untuk Test Case 3

d. Test Case 4



Gambar 4.3.13 BFS Single Path untuk Test Case 4



Gambar 4.3.14 IDS Single Path untuk Test Case 4

4.4 Analisis Hasil Pengujian

Dari hasil pengujian diatas, diperoleh data mengenai waktu eksekusi program serta jumlah rute yang didapatkan dari masing-masing algoritma dan berdasarkan pilihan pencarian jumlah rute (*single path* atau *multiple path*). Berdasarkan data tersebut, dapat dilihat bahwa waktu eksekusi dari multiple path relatif jauh lebih lama dari waktu eksekusi single path. Hal ini sangat wajar, mengingat pada algoritma multiple path program harus mencari semua solusi yang ada pada depth tertentu, sedangkan jumlah link yang perlu dicek untuk tiap depth meningkat secara eksponensial. Kemudian, dilihat dari sisi jumlah rute yang ditemukan untuk pencarian *multiple path,* didapatkan bahwa jumlah rute yang ditemukan oleh algoritma BFS lebih banyak daripada algoritma IDS. Hal ini dikarenakan algoritma IDS tidak hanya mengecek judul heading pada laman yang dikunjungi, melainkan juga mengecek apakah link tujuan berada di dalam hasil scraping laman yang sedang dikunjungi. Karena link hasil scraping bisa saja merupakan link redirect menuju laman tujuan, maka ketika sudah ditemukan solusi pada depth saat ini, semua link redirect yang terdapat pada depth saat ini tidak akan masuk solusi algoritma IDS karena link tersebut berbeda dengan link tujuan (padahal sama-sama mengarah ke laman tujuan). Di sisi lain, BFS mengecek solusi dengan mengunjungi tiap laman untuk mengecek kesesuaian judul laman dengan judul tujuan. Pendekatan ini memang menyebabkan waktu eksekusi jauh lebih lama dari implementasi IDS diatas (karena ketambahan 1 depth), namun pendekatan ini dijamin mendapatkan semua solusi pada kedalaman tertentu, termasuk rute melalui semua link redirect yang ada pada kedalaman sebelumnya.

Selanjutnya, dilihat dari tipe algoritma yang digunakan diperoleh bahwa algoritma IDS selalu dapat mencari solusi dengan lebih cepat dari algoritma BFS. Hal ini sangat tidak wajar, mengingat secara teori IDS selalu lebih lambat dari BFS karena IDS selalu mengembalikan pencarian mulai dari kedalaman 0 setiap kali ingin menambah batas kedalaman (demi mempertahankan kompleksitas ruangnya agar sama seperti DFS). Kejanggalan ini terjadi karena algoritma BFS yang diimplementasikan mengecek apakah sudah ditemukan laman solusi dengan mengecek *heading* dari laman yang sedang di-*visit*, berbeda dengan IDS yang juga mencocokkan apakah link tujuan sudah terdapat dalam hasil scraping saat ini. Dengan begitu, untuk mencari laman yang terdapat pada kedalaman 2, BFS akan mengunjungi semua laman yang berada di kedalaman 1 dan 2, sedangkan IDS hanya mengunjungi laman yang berada di kedalaman 1 (kecuali bila semua link pada

kedalaman 1 merupakan link *redirect* yang berbeda dengan link asli tujuan). Maka dari itu, waktu eksekusi BFS meningkat dengan jauh lebih cepat dari IDS seiring bertambahnya kedalaman laman solusi.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritma *Breadth-First Search* (BFS) dan *Iterative Deepening Search* (IDS) adalah salah satu algoritma pencarian rute terpendek dari satu simpul menuju simpul lain dalam graf (jika jumlah langkah = biaya). Kedua algoritma tersebut dapat dimanfaatkan untuk mencari solusi yang optimal dalam permainan *Wikirace*. Walaupun solusi yang didapatkan dari pencarian menggunakan algoritma BFS dan IDS pasti merupakan solusi yang optimal, pencarian memakan waktu yang cukup lama. Hal ini dikarenakan jumlah link yang perlu diperiksa meningkat secara eksponensial tiap depthnya, sehingga suatu depth akan memiliki waktu eksekusi pencarian yang jauh lebih lama dari depth sebelumnya. Maka, jika algoritma BFS dan IDS ingin digunakan dalam permainan *Wikirace* dimana kecepatan pencarian merupakan hal yang sangat penting, perlu dilakukan optimalisasi lebih lanjut seperti menggunakan *goroutine* untuk mempercepat pencarian ataupun menggunakan database Wikipedia agar program tidak perlu mengirim HTTP request untuk mendapatkan link menuju laman berikutnya.

5.2. Saran

Saran-saran untuk kelompok kami agar kedepannya dapat mengerjakan tugas-tugas lain dengan lebih baik diantaranya:

1) Mempelajari lebih dalam bahasa pemrograman yang digunakan, untuk mengurangi waktu yang dihabiskan untuk *debugging* program.
2) Menggunakan waktu yang diberikan dengan lebih baik, dan meningkatkan komunikasi antar anggota dalam pengerjaan tugasnya.

5.3. Refleksi

Refleksi yang didapatkan oleh kelompok kami setelah menyelesaikan tugas besar ini adalah bahwa tugas ini memakan waktu yang jauh lebih banyak dari yang kami pikirkan pada awalnya, sehingga untuk kedepannya kami merasa bahwa perlu dilakukan lebih banyak pertemuan kelompok agar tugas dapat diselesaikan dengan lebih cepat dan menghasilkan hasil akhir yang lebih matang.

LAMPIRAN

Repository github dapat diakses melalui pranala berikut:

https://github.com/IrfanSidiq/Tubes2_wikiwikiwiki

Link video:

https://youtu.be/ntDRXQqRRQQ

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