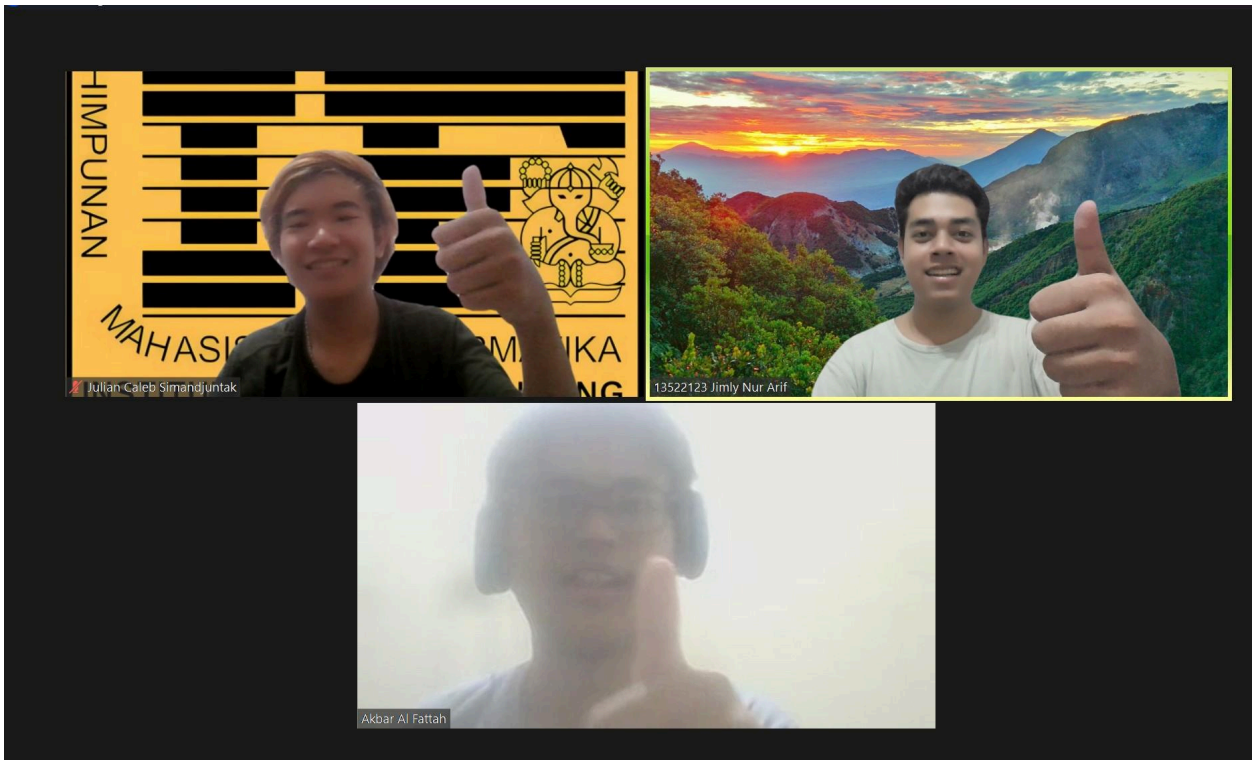
Laporan Tugas Besar 2 IF2211 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Kelompok "Kami Mau C# dan Unity":
Akbar Al Fattah - 13522036
Julian Caleb Simandjuntak - 13522099
Jimly Nur Arif - 13522123

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

IF2211 - Strategi Algoritma 2024

Bab 1
Deskripsi Tugas

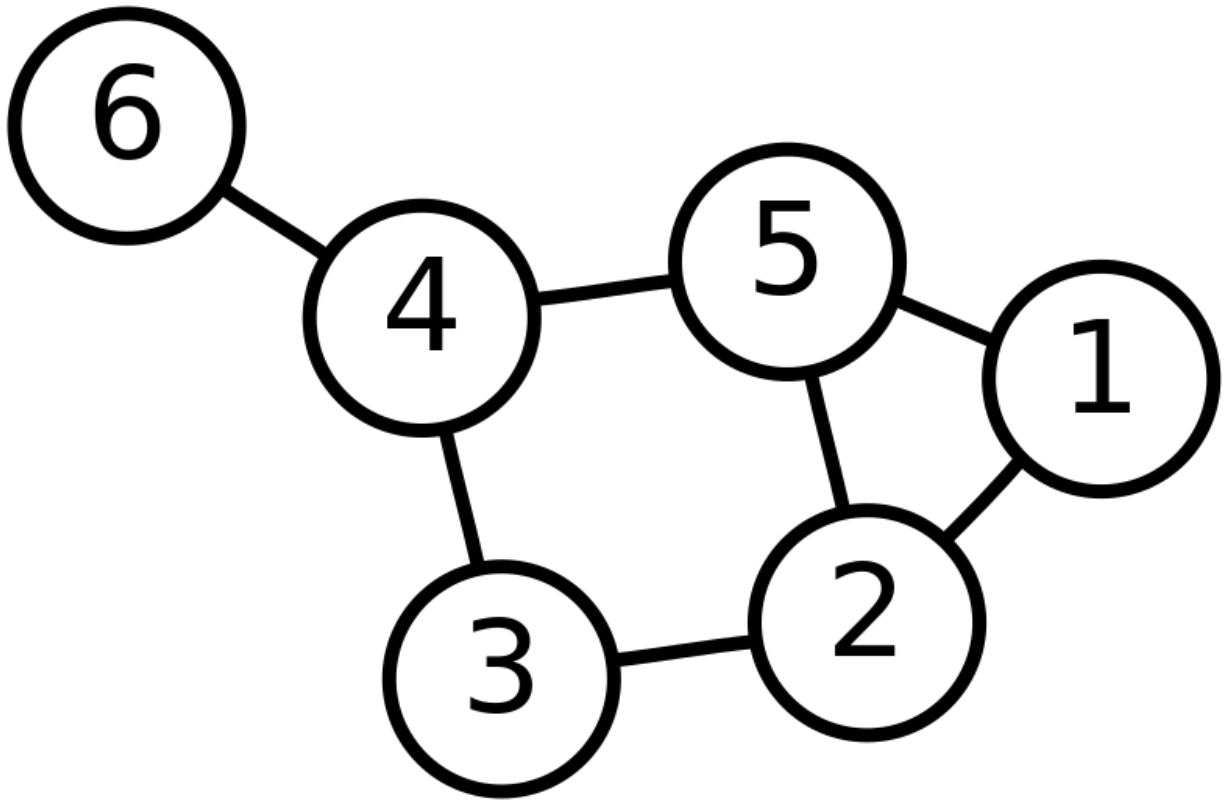
WikiRace atau Wiki Game adalah permainan yang melibatkan Wikipedia, sebuah ensiklopedia daring gratis yang dikelola oleh berbagai relawan di dunia, dimana pemain mulai pada suatu artikel Wikipedia dan harus menelusuri artikel-artikel lain pada Wikipedia (dengan mengeklik tautan di dalam setiap artikel)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

Tugas Besar 2 IF2211 Strategi Algoritma meminta mahasiswa untuk membuat sebuah program berbasis web menyerupai WikiRace yang menggunakan Breadth First Search (BFS) dan Iterative Deepening Search (IDS) untuk mencari path Wikipage dari satu Wikipage ke Wikipage tujuan dengan menggunakan bahasa Go (FrontEnd dibebaskan). Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan, 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).

Bab 2
Landasan Teori

Graf

Graf merupakan representasi visual yang terdiri dari titik-titik yang dihubungkan oleh garis atau sisi, yang sering digunakan untuk mewakili hubungan antar objek dalam berbagai domain, seperti dalam jaringan komputer, analisis sosial, dan algoritma grafik. Sebuah graf G adalah sebuah pasangan G = (V,E) dengan V adalah sebuah himpunan tak kosong beranggotakan titik atau simpul dan E adalah sebuah himpunan beranggotakan sisi, yakni pasangan titik. Misalkan v dan u adalah titik pada graf, sisi yang menghubungkan u dan v biasa ditulis sebagai uv. Jika sisi uv adalah anggota himpunan E, sisi u dan v disebut bertetangga. Untuk suatu titik pada graf, lingkungan titik tersebut adalah himpunan seluruh titik yang bertetangga dengannya. Derajat dari suatu titik adalah banyak sisi yang terkait dengan titik tersebut.



BFS, DFS, dan IDS

Graf dapat digunakan sebagai representasi persoalan, dengan traversal graf artinya melakukan pencarian solusi persoalan yang direpresentasikan dengan graf. Persoalan dalam graf dapat diselesaikan dengan beberapa algoritma, seperti BFS, DFS, dan IDS.

Breadth First Search (BFS) adalah algoritma traversal graf yang mengeksplorasi semua simpul dalam graf pada satu kedalaman sebelum berpindah ke simpul pada tingkat kedalaman berikutnya. BFS bekerja dengan cara mengunjungi sebuah simpul, mengunjungi semua simpul yang bertetangga dengan simpul terlebih dahulu, baru mengunjungi simpul yang belum dikunjungi, bertetangga dengan simpul-simpul yang tadi dikunjungi, dan seterusnya.

Depth First Search (DFS) adalah algoritma traversal graf yang mengeksplorasi sebuah simpul dalam graf dan menjelajah sejauh mungkin di sepanjang setiap cabang sebelum melakukan backtracking. DFS bekerja dengan mengunjungi sebuah simpul, mengunjungi sebuah simpul yang bertetangga dengan simpul tersebut, diulangi hingga semua telah dikunjungi, melakukan backtrack hingga ada tetangga simpul yang belum dikunjungi, mengunjungi tetangga simpul tersebut, dan dilakukan hingga semua simpul dalam graf telah dikunjungi. Iterative Deepening Search (IDS) adalah strategi pencarian ruang/grafik keadaan di mana versi depth-first search yang dibatasi kedalamannya dijalankan berulang kali dengan batas kedalaman yang meningkat hingga tujuannya ditemukan.

Deskripsi Web

Web yang dibuat akan dibagi menjadi Frontend dan Backend. Frontend akan dibuat dengan menggunakan HTML, CSS, dan JavaScript sedangkan Backend akan dibuat dengan Go (Golang). Web akan di-run dengan menjalankan file Go pada Backend, pengguna dapat memasukkan judul Wikipage sesuai dengan deskripsi tugas, diberikan kepada Backend untuk diproses dengan menggunakan BFS atau IDS untuk web scraping, dan mengembalikan hasil pathnya kembali ke Frontend.

Bab 3
Analisis Pemecahan Masalah

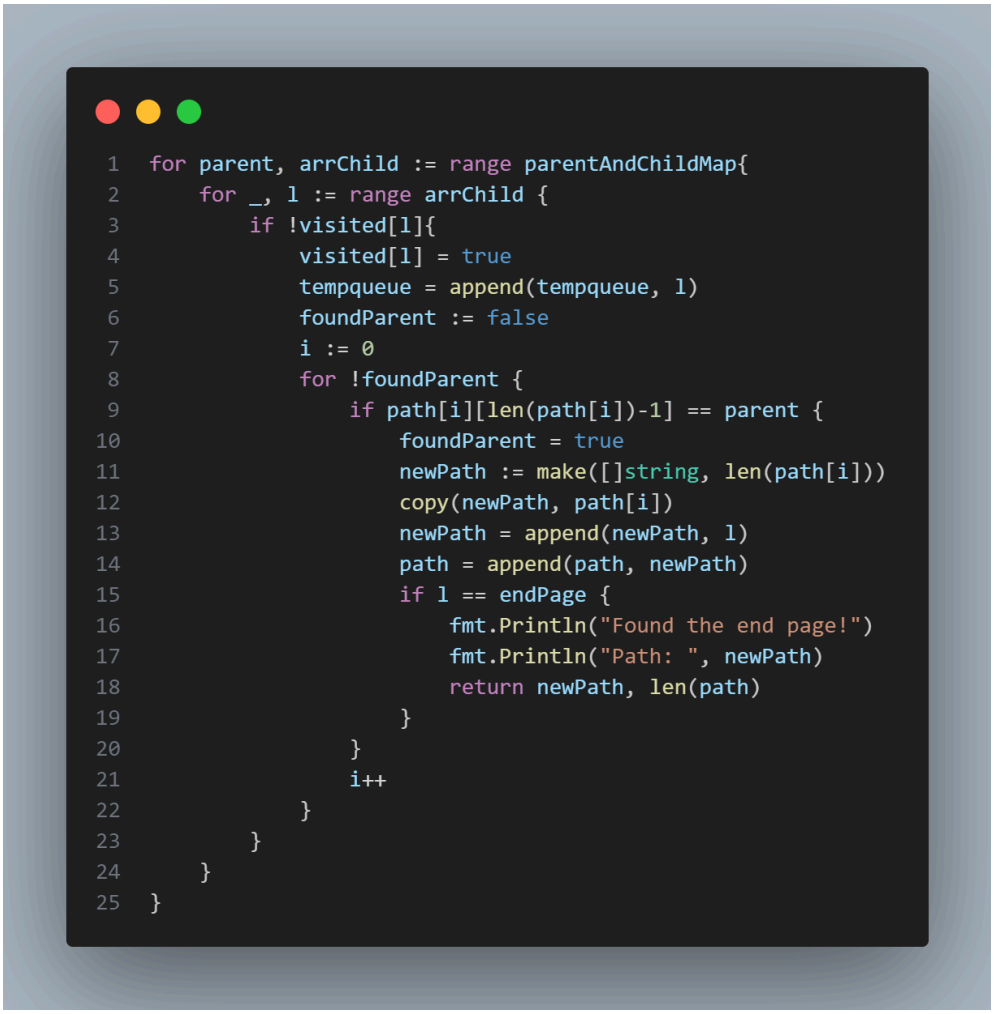
Langkah Pemecahan Masalah

Permainan WikiRace bertujuan untuk secepat mungkin dari node awal menuju node akhir. Untuk proses pencarian ini karena dapat divisualisasikan sebagai graf, maka metode pencarian seperti Breadth Search First (BFS) atau Iterative Depth Search (IDS) dapat dilakukan. Proses pencarian child dari suatu node membutuhkan proses untuk meminta data dari sebuah website. Agar program memiliki modularitas, keterbacaan, dan perawatan yang mudah maka kami menggunakan library gocolly untuk melakukan proses meminta data ke website (selanjutnya disebut fetch link). Menurut pengujian kami saat pengerjaan, kami menemukan bahwa salah satu komponen yang membutuhkan waktu yang lama pada jalannya program adalah fetch link. Oleh karena itu supaya proses ini dapat berjalan cepat, maka kami melakukan prinsip parallelism, di mana sebuah fetch link dengan fetch link lain dijalankan bersama-sama.

Pemecahan Masalah BFS

Breadth Search First atau disingkat BFS adalah algoritma pencarian yang digunakan untuk melintasi atau mencari simpul dalam struktur data grafik. Algoritma ini dimulai dari simpul awal dan menyebar ke simpul tetangga sejauh satu tingkat sebelum bergerak ke tingkat selanjutnya.
Berikut cuplikan kode BFS

```go
func BFS(startPage string, endPage string) ([]string, int) {

    path := [][]string{{startPage}}
    queue := []string{startPage}
    visited := make(map[string]bool)
    visited[startPage] = false
    if startPage == endPage {
     fmt.Println("Found the end page!")
     fmt.Println("Path: ", startPage)
     return []string{startPage}, 1
    }
    var tempqueue []string
    var wannaGetLinks []string

    fmt.Println("flag 1")
    for len(queue) >= 0 {
        fmt.Println("flag 2")
        if len(queue) == 0 {

            if len(tempqueue) == 0 {
                fmt.Println("queue dan tempqueue habis")
                return []string{}, len(path)
            }
```



```go
for parent, arrChild := range parentAndChildMap{
    for _, l := range arrChild {
        if !visited[l]{
            visited[l] = true
            tempqueue = append(tempqueue, l)
            foundParent := false
            i := 0
            for !foundParent {
                if path[i][len(path[i])-1] == parent {
                    foundParent = true
                    newPath := make([]string, len(path[i]))
                    copy(newPath, path[i])
                    newPath = append(newPath, l)
                    path = append(path, newPath)
                    if l == endPage {
                        fmt.Println("Found the end page!")
                        fmt.Println("Path: ", newPath)
                        return newPath, len(path)
                    }
                }
                i++
            }
        }
    }
}
```

Berikut adalah proses BFS sesuai fungsionalitas di program.

1. Periksa apakah start Page dan end Page sama, bila ya return path
2. bila tidak, ambil paling banyak 400 link yang ada antrian halaman yang ingin dijelajahi tiap degreenya.
3. Tambahkan link tersebut ke dalam path, periksa link hasil penjelajahan tadi apakah ada yang sama dengan end page. Simpann link hasil penjelajahan ke tempqueue
4. Periksa apakah queue sudah kosong
5. Jika queue kosong masukkan tempqueue ke queue
6. Jika queue tidak kosong, ulangi proses kedua.
7. Lakukan terus-menerus hingga halaman ditemukan.

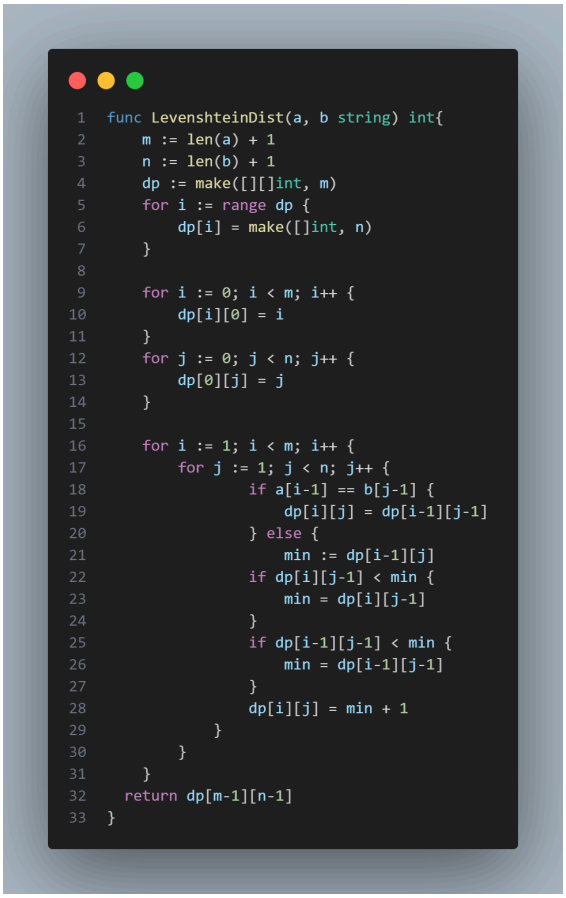
Untuk pengoptimalan proses kami menggunakan

1. fitur Remove_Redundant()

```go
func RemoveRedundant(array []string) []string {
    result := make([]string, 0)
    seen := make(map[string]int)

    for _, word := range array {
        parts := strings.Split(word, "_")
        baseWord := parts[0]

        if idx, ok := seen[baseWord]; !ok || len(word) < len(array[idx]) {
            seen[baseWord] = len(result)
            result = append(result, word)
        }
    }

    return result
}
```

yang berfungsi untuk meminimalisir node yang akan dijelajahi namun tetap memperhatikan agar tidak terjadi race condition antar degree yang berbeda. Misalnya kita ingin mencari Intel_i7_gen14 dan pada queue terdapat [Intel_i7_gen14, Intel_i7_gen14 , Intel_i7], maka kita hanya perlu mencari Intel_i7 untuk semestara, karena hampir pasti Intel_i7_gen14 terdapat dalam Intel_i7.

2. Fungsi pengoptimalan berikutnya adalah Levenshtein distance



```go
func LevenshteinDist(a, b string) int{
    m := len(a) + 1
    n := len(b) + 1
    dp := make([][]int, m)
    for i := range dp {
        dp[i] = make([]int, n)
    }

    for i := 0; i < m; i++ {
        dp[i][0] = i
    }
    for j := 0; j < n; j++ {
        dp[0][j] = j
    }

    for i := 1; i < m; i++ {
        for j := 1; j < n; j++ {
                if a[i-1] == b[j-1] {
                    dp[i][j] = dp[i-1][j-1]
                } else {
                    min := dp[i-1][j]
                if dp[i][j-1] < min {
                    min = dp[i][j-1]
                }
                if dp[i-1][j-1] < min {
                    min = dp[i-1][j-1]
                }
                dp[i][j] = min + 1
            }
        }
    }
  return dp[m-1][n-1]
}
```

Levenshtein distance mengurutkan queue dengan kemiripan kata yang paling tinggi akan didahulukan.

3. Fungsi pengoptimalan berikutnya adalah penggunaan parallelism

```go
func GetLinksMap(juduls []string) map[string][]string {
    results := SafeLinksMap{}

    c := colly.NewCollector(
        colly.MaxDepth(1),
    )
    c.Limit(&colly.LimitRule{DomainGlob: "*", Parallelism: 4000})

    isExist := SafeLinksMap{}

    q, _ := queue.New(
        22,
        &queue.InMemoryQueueStorage{MaxSize: 1000000},
    )

    for _, judul := range juduls {
        q.AddURL("https://en.wikipedia.org/wiki/" + judul)
        results.StoreLinks(judul, make([]string, 0)) // Initialize with empty slice
    }

    c.OnHTML("a[href]", func(e *colly.HTMLElement) {
        link := e.Attr("href")
        if strings.HasPrefix(link, "/wiki/") && !strings.Contains(link, ":") && link != "/wiki/Main_Page" {
            if _, ok := isExist.Load(link); !ok {
```

Parallelism digunakan untuk melakukan fetch link secara serentak sehingga tidak ada proses tunggu-menunggu fetch link yang satu dengan yang lain. Kompleksitas waktu untuk BFS adalah O(V+E) dengan V adalah banyak vertex dan E adalah banyak edge dengan pengurangan waktu dari optimasi yang terlalu sulit untuk dihitung karena banyak kasus.

Pemecahan Masalah IDS

Iterative Deepening Search (IDS) atau disebut juga Iterative Deepening Depth First Search (IDDFS) adalah algoritma pencarian pada graf. IDS pada dasarnya adalah DLS (Depth Limited Search) yang kedalamannya diiterasi secara terus menerus, dan DLS adalah DFS (Depth-First Search) yang kedalamannya dibatasi.

Untuk tugas besar ini, berikut adalah algoritma IDS **naif/lempang/dasar** yang digunakan:

1. Periksa apakah *startPage* dan *endPage* sama, jika sama, pathnya adalah *startPage* atau *endPage* dan **hentikan pencarian lebih lanjut**
2. Jika tidak, jalankan fungsi DLS dari kedalaman 1 dan kedalaman terus ditambah sampai *path* ditemukan.

Sedangkan untuk algoritma DLS **naif/lempang/dasar** adalah

1. Jika halaman yang dikunjungi saat ini sama dengan halaman yang dicari, artinya kita sudah menemukan jawabannya dan halaman yang dikunjungi dimasukkan ke path
2. Jika halaman yang dikunjungi saat ini berbeda dengan halaman yang dicari DAN sudah melebihi batas kedalaman yang ditentukan (batas <=0), artinya kita tidak menemukan jawabannya
3. Cari semua link wikipedia yang terdapat di dalam halaman yang sedang dikunjungi.
4. Iterasi semua link yang didapatkan dari langkah 3:
    a. Cari subpath secara rekursif dengan titik awal pencarian adalah link yang sedang diiterasi dan batas kedalaman dikurangi 1
    b. Jika subpath ditemukan, kembalikan gabungan dari subpath yang didapatkan dan halaman yang dikunjungi saat ini (DLS berhenti di sini)
    c. Backtrack semua perubahan jika ada
5. Kembalikan path kosong karena jawabannya tidak ditemukan dari semua kemungkinan di atas

Agar IDS memiliki performa yang lebih optimal, berikut teknik pengoptimalan yang digunakan untuk algoritma IDS:

1. Multithreading secara terfragmentasi

   Multithreading pada IDS di sini maksudnya adalah IDS akan menjalankan 2 buah thread yang mengeksekusi fungsi IDS secara bersamaan. Alasan hanya ada 2 buah thread yang dijalankan secara bersamaan karena jika 3 buah thread IDS atau lebih dijalankan secara bersamaan, maka akurasi jawaban akan menurun bahkan jawabannya tidak selalu menampilkan path dengan panjang terpendek.

   Banyak thread maksimum yang akan dijalankan sekaligus nilai kedalaman maksimum pada algoritma IDS adalah 8 dan dimulai dari nilai kedalaman 1. Alasannya adalah rata-rata hasil dari WikiRace menghasilkan path yang panjangnya 2-4 link, dan ada beberapa outlier yang menghasilkan 7-8 degree.

   Arti dari terfragmentasi adalah 8 thread ini akan dibagi menjadi 4 buah sesi dengan 1 buah sesi diisi oleh 2 buah thread IDS. Alasan dilakukannya

fragmentasi pada multithreading IDS adalah untuk meningkatkan akurasi jawaban terpendek saat melakukan multithreading. Semakin sedikit sesi/banyak thread yang dijalankan secara bersamaan, jawabannya semakin tidak akurat karena banyak link yang didapatkan di kedalaman tertentu pasti tidak sama sehingga ada kemungkinan thread dengan kedalaman maksimum yang lebih tinggi selesai terlebih dahulu daripada thread dengan kedalaman maksimum yang lebih kecil. Selain itu, jika path sudah ditemukan di salah satu sesi, maka IDS secara keseluruhan dapat dihentikan karena path yang didapat adalah path terpendek.

2. Pengurutan array string link berdasarkan kemiripan dengan Levenshtein Distance

   Sama seperti pada BFS, Levenshtein Distance juga digunakan untuk mengurutkan daftar link pada halaman wikipedia tertentu berdasarkan kemiripan dengan link tujuan. Alasan dilakukan pengurutan ini adalah untuk mengurangi waktu IDS dan meningkatkan akurasi jawaban.

3. Penghentian semua thread secara serentak setelah path ditemukan

   Pada bagian 1 optimasi IDS, sudah dijelaskan bahwa program akan membuat 8 buah thread yang terbagi ke 4 buah sesi. Kemudian, jika path sudah ditemukan, semua thread yang berjalan akan dihentikan untuk menghemat waktu.

Berikut adalah cuplikan kode untuk IDS.

```go
func IDS(startPage string, endPage string) ([]string,int) {
    if startPage == endPage { //cek apakah awal dan akhirnya sama
        return []string{startPage},1       You, 34 minutes ago • feat: count number of visited links (for IDS)
    }
    links := scraper.GetLinksArr(startPage)
    if len(links) == 0 { //handling error: halaman tidak ada di wikipedia
        return []string{},0
    }
    path := []string{}
    temp:= 0
    total := 0
    for iteration := 1; iteration <= 8; iteration += 2 { //iterasi fragmentasi IDS multithreading (fragmentasi:
        fmt.Println("Fragmen dari: ",iteration," sampai: ",iteration+1)
        path,temp= IDSFragment(startPage, endPage, iteration, iteration+1) //cari path dari kedalaman iteration
        total += temp //jumlahkan link yang dikunjungi
        if path != nil { //kalau sudah ketemu path dari fragmen, jangan dilanjutkan IDSnya
            break
        }
    }

    return path,total //kembalikan path dan total link yang dikunjungi
}
```

```go
func DLS(src string, target string, limit int, visited map[string]bool, stopExplore chan bool, visCount* int) ([]string, bool) {
    visited[src] = true //tandai sudah pernah divisit
    *visCount++ //tambah jumlah halaman yang dikunjungi dengan 1 (halaman saat ini)
    if src == target { //kalau halaman yang divisit sekarang sama dengan halaman yang dicari
        ret := []string{src} //masukin ke path
        return ret, true    //artinya sudah ketemu pathnya
    }

    if limit <= 0 { //kalau limitnya sudah habis DAN src dan targetnya beda
        return nil, false //tidak ketemu
    }

    links := scraper.GetLinksArr(src) //dapatkan semua link yang ada di halaman yang sedang dikunjungi
    links = scraper.SortStringsBySim(target,links) //urutkan semua link berdasarkan kemiripan dengan target
    fmt.Println(target) //debug
    fmt.Println(links)
    for _, nextLink := range links {         //iterasi ke semua link yang ada di halaman yang sedang dikunjungi
        if visited[nextLink] { //lewati link yang sudah dikunjungi
            continue
        }
        visited[nextLink] = true //tandai link selanjutnya dengan true
        select {
        case explored := <-stopExplore: //jika ada sinyal untuk menghentikan algoritma IDS
            if explored {
                return nil, true //hentikan semua algoritma IDS
            }
        default:
            subPath, found := DLS(nextLink, target, limit-1, visited, stopExplore, visCount) //kunjungi node selanjutnya dan kura
            if found { //kalau ketemu
                return append([]string{src}, subPath...), true //tambahkan nama halaman yang sedang dikunjungi sekarang ke subpat
            }
        }

        delete(visited, nextLink) //Backtrack visited
    }
    return nil, false //tidak ketemu pathnya
}
```

```go
func IDSFragment(startPage string, endPage string, startIdx int, endIdx int) ([]string,int) { //thread IDS terfragmentasi
    var wg sync.WaitGroup
    wg.Add(2)

    ch := make(chan []string, 2)
    stopExplore := make(chan bool, 1)
    total:=0

    for iteration := startIdx; iteration <= endIdx; iteration++ { //tambah kedalaman terus dari startIDX sampai endIdx sampai ketemu pathnya
        initVal :=0
        go func(d int) { //buat goroutine
            defer wg.Done()
            path, found:= DLS(startPage, endPage, d, map[string]bool{}, stopExplore, &initVal) //dapatkan path dan apakah berhasil ditemukan
            if found { // jika ditemukan
                ch <- path //masukkan path ke channel
                stopExplore <- true //hentikan semua goroutine DLS
                total += initVal //jumlahkan banyak link yang divisit
                return
            }
            ch <- nil
        }(iteration)
    }

    wg.Wait()

    for i := 0; i < 2; i++ {
        path := <-ch

        if path != nil { //jika path tidak kosong
            if path[0] == startPage && path[len(path)-1] == endPage { //periksa jika link pertama adalah startPage dan link terakhir adalah endPage
                return path, total //return path yang udah ketemu dan banyak link total yang dikunjungi
            }
        }
    }
    return nil,total //return path yang tidak ditemukan
}
```

Kompleksitas waktu untuk IDS adalah O(b^d) dengan b adalah branching factor dan d adalah kedalaman tertinggi, dengan beberapa pengurangan waktu setelah optimasi dilakukan.

Fitur Fungsionalitas dan Arsitektur Website

Website mempunyai beberapa fungsionalitas yang mendukung Website berjalan, selain daripada IDS dan BFS, yaitu di Backend, Go digunakan untuk menyalakan Website itu sendiri sekaligus menjalankan IDS atau BFS sesuai kebutuhan.

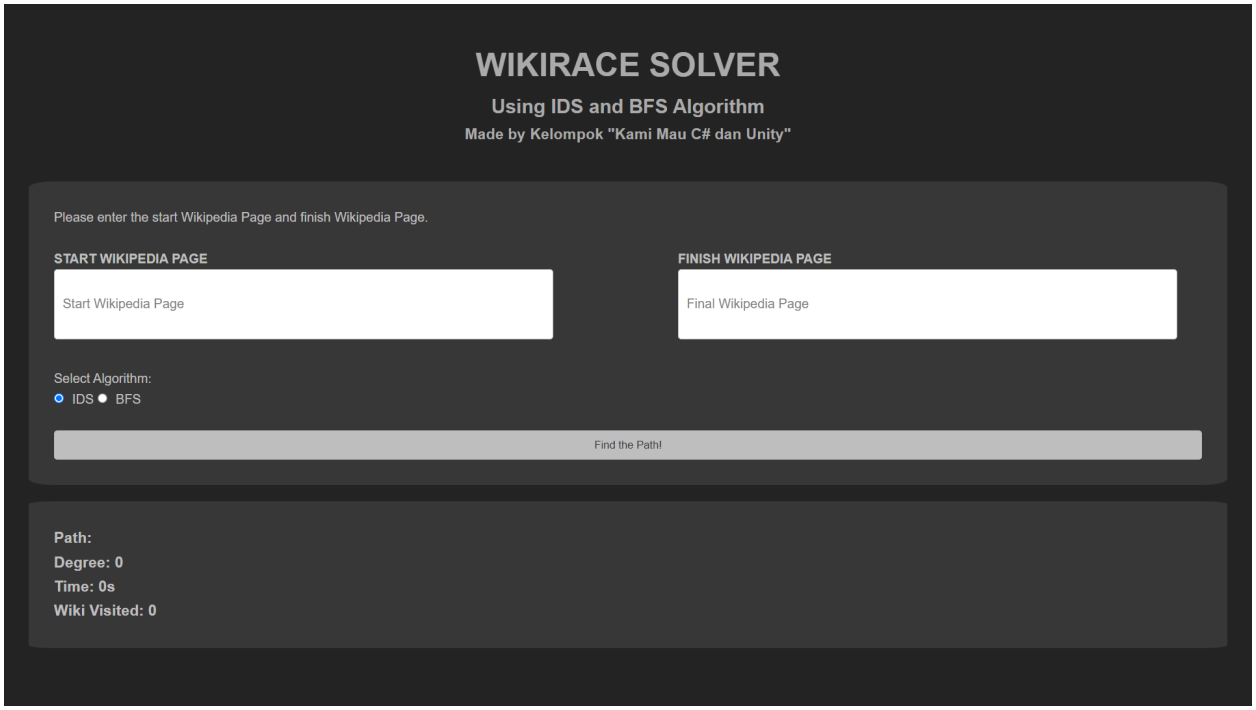
```go
func main() {
    // Membuat server untuk frontend
    // sekaligus inisialisasi awal empty array
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var paths []string

        data := Response{
            Path:     paths,
            PathLink: paths,
            Degree:   0,
            Duration: 0,
            File:     0,
        }

        tmpl, err := template.ParseFiles("../frontend/index.html")
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        err = tmpl.Execute(w, data)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }
    })
```

```
    // Proses mengambil data dari form
    http.HandleFunc("/submit", func(w http.ResponseWriter, r *http.Request) {
        // Mengecek data dari form
        err := r.ParseForm()
        if err != nil {
            http.Error(w, "Failed to parse form data", http.StatusBadRequest)
            return
        }

        // Mengambil value dan meletakkannya pada variable
        start := url.QueryEscape(strings.ReplaceAll(r.Form.Get("start"), " ", "_"))
        finish := url.QueryEscape(strings.ReplaceAll(r.Form.Get("finish"), " ", "_"))
        algorithm := r.Form.Get("algorithm")

        // Debug
        fmt.Printf("Start: %s, Finish: %s, Algorithm: %s\n", start, finish, algorithm)

        // Mencari hasil
        var path []string
        var pathLink []string
        var nFileVisited int
        startTime := time.Now()
        if algorithm == "BFS" {
            pathLink, nFileVisited = BFS.BFS(start, finish)
        } else if algorithm == "IDS" {
            pathLink, nFileVisited = IDS.IDS(start, finish)

        }
        endTime := time.Now()

        // Judul
        for _, link := range pathLink {
            decodedLink, err := url.QueryUnescape(link)
            if err != nil {
                fmt.Println("Error decoding link:", err)
                return
            }
            path = append(path, decodedLink)
        }

        // Degree
        degree := len(pathLink) - 1

        // Duration
        duration := endTime.Sub(startTime)

        // Debug
        fmt.Println(path)
        fmt.Println("Jumlah File yang dikunjungi BFS: ", nFileVisited)
        // fmt.Println("Duration:", duration)

        // Passing ke HTML
        data := Response{
            Path:     path,
            PathLink: pathLink,
            Degree:   degree,
            Duration: duration,
            File:     nFileVisited,
        }

        tmpl, err := template.ParseFiles("../frontend/index.html")
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        err = tmpl.Execute(w, data)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }
    })

    // Menyalakan server
    fmt.Println("Server is running on port 8080")
    http.ListenAndServe(":8080", nil)
}
```

Di FrontEnd juga terdapat fungsi untuk memberikan suggestion bagi pengguna dalam memberikan input Wikipage, sekaligus untuk mencegah invalid input.

```
<!-- Fungsi JavaScript untuk memberikan rekomendasi/sugesti Wikipedia Page -->
<script>
    // Function to provide Wikipedia page suggestions
    function setupSuggestions(inputField, suggestionsDiv) {
        inputField.addEventListener('input', function() {
            const searchTerm = inputField.value.trim();
            if (searchTerm === '') {
                suggestionsDiv.innerHTML = '';
                return;
            }

            // Using the Wikipedia API to search for pages related to the input
            fetch(`https://en.wikipedia.org/w/api.php?action=opensearch&format=json&search=${searchTerm}&origin=*`)
            .then(response => response.json())
            .then(data => {
                const suggestions = data[1];
                suggestionsDiv.innerHTML = '';
                // If no suggestions found
                if (suggestions.length === 0) {
                    suggestionsDiv.innerHTML = '<p>No suggestions found</p>';
                // If suggestions found
                } else {
                    // Insert them into a list
                    const ul = document.createElement('ul');
                    suggestions.forEach(suggestion => {
                        const li = document.createElement('li');
                        li.innerHTML = `
                        <p>${suggestion}</p>
                        <a href="https://en.wikipedia.org/wiki/${suggestion}" target="_blank">https://en.wikipedia.org/wiki/${suggestion}</a>`;
                        // Clickable
                        li.addEventListener('click', function() {
                            inputField.value = suggestion;
                            suggestionsDiv.innerHTML = '';
                        });
                        ul.appendChild(li);
                    });
                    suggestionsDiv.appendChild(ul);
                }
            })
            .catch(error => {
                console.error('Error fetching data:', error);
                suggestionsDiv.innerHTML = '<p>Error fetching suggestions. Please try again later.</p>';
            });
        });
    }

    // For the start input
    const startInput = document.getElementById('start');
    const startSuggestionsDiv = document.getElementById('startSuggestions');
    setupSuggestions(startInput, startSuggestionsDiv);

    // For the finish input
    const finishInput = document.getElementById('finish');
    const finishSuggestionsDiv = document.getElementById('finishSuggestions');
    setupSuggestions(finishInput, finishSuggestionsDiv);
</script>
```

Secara arsitektural, Website terdiri atas bagian judul, form untuk meminta input, dan bagian untuk mengeluarkan output dari Backend.


WIKIRACE SOLVER
Using IDS and BFS Algorithm
Made by Kelompok "Kami Mau C# dan Unity"
Please enter the start Wikipedia Page and finish Wikipedia Page.
START WIKIPEDIA PAGE
Start Wikipedia Page
FINISH WIKIPEDIA PAGE
Final Wikipedia Page
Select Algorithm:
IDS BFS
Find the Path!
Path:
Degree: 0
Time: 0s
Wiki Visited: 0

```
<body>
    <!-- Header website -->
    <div id="header">
        <h1>WIKIRACE SOLVER</h1>
        <h2>Using IDS and BFS Algorithm</h2>
        <h3>Made by Kelompok "Kami Mau C# dan Unity"</h3>
    </div>
    <!-- Form untuk website -->
    <div id="form">
        <form action="/submit" method="GET">
            <!-- Submission start, finish, dan tipe algorithm akan diberikan kepada Go file di backend -->
            <p>Please enter the start Wikipedia Page and finish Wikipedia Page.</p>
            <br />
            <div id="input-container">
                <div id="suggestions">
                    <p><strong>START WIKIPEDIA PAGE</strong></p>
                    <input type="text" id="start" name="start" placeholder="Start Wikipedia Page" required>
                    <div id="startSuggestions"></div>
                </div>
                <div id="suggestions">
                    <p><strong>FINISH WIKIPEDIA PAGE</strong></p>
                    <input type="text" id="finish" name="finish" placeholder="Final Wikipedia Page" required>
                    <div id="finishSuggestions"></div>
                </div>
            </div>
            <br />
            <p>Select Algorithm:</p>
            <input type="radio" id="ids" name="algorithm" value="IDS" checked>
            <label for="ids">IDS</label>
            <input type="radio" id="bfs" name="algorithm" value="BFS">
            <label for="bfs">BFS</label>
            <br />
            <br />
            <button type="submit">Find the Path!</button>
        </form>
    </div>
    <div id="result">
        <h3>Path:</h3>
        <ol>
            {{range $i, $element := .Path}}
                <li>
                    <a href="https://en.wikipedia.org/wiki/{{index $.PathLink $i}}">{{index $.Path $i}}</a>
                </li>
            {{end}}
        </ol>
        <h3>Degree: {{.Degree}}</h3>
        <h3>Time: {{.Duration}}</h3>
        <h3>Wiki Visited: {{.File}}</h3>
    </div>
```

Bab 4
Implementasi dan Pengujian

Tata Cara Penggunaan

1. Lakukan git clone pada repository (atau git pull jika sudah clone)

```
git clone
https://github.com/DeltDev/Tubes2_Kami-Mau-Csharp-dan-Unity.git
```

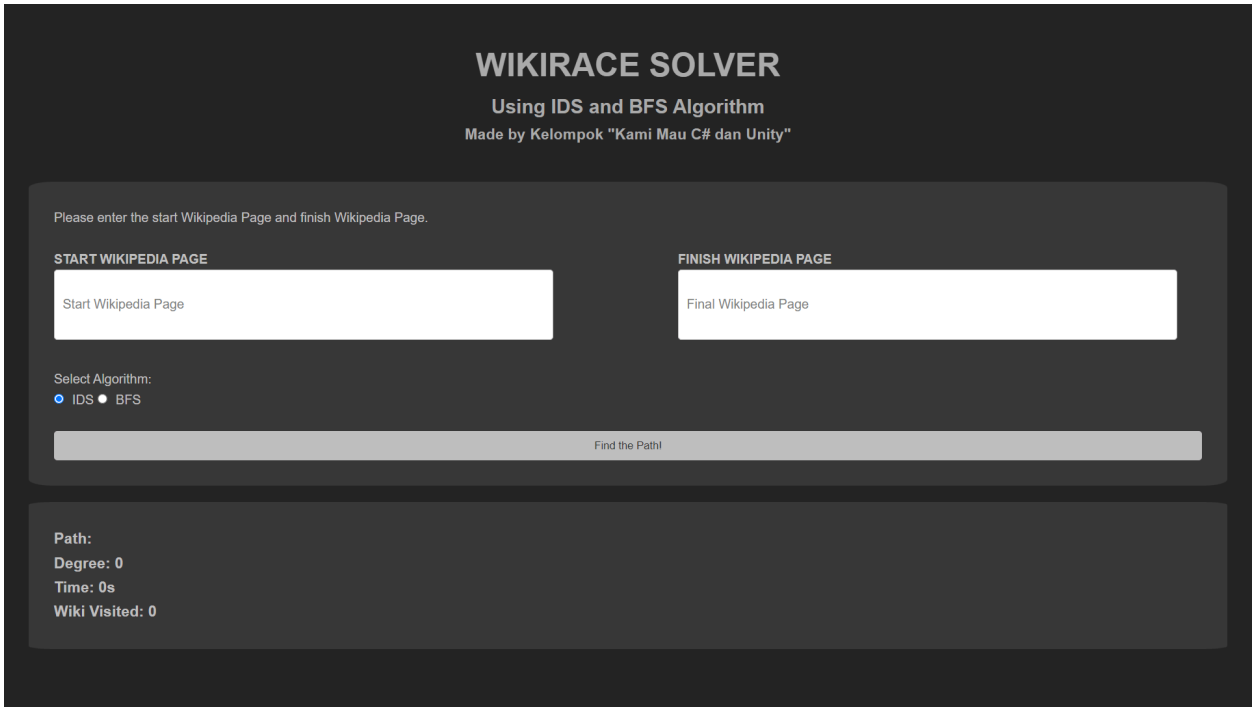
2. Masuk ke folder backend

```
cd src
cd backend
```

3. Jalankan file main

```
go run .
```

4. Buka localhost:8080 pada browser
5. Akan muncul tampilan UI sebagai berikut



6. Masukkan WikiPage awal ke bagian Start Wikipedia Page dan tujuan ke Finish Wikipedia Page
7. Pilih algoritma yang ingin digunakan, IDS atau BFS
8. Tekan tombol "Find The Path!"

9. Tunggu hasil dari algoritma IDS atau BFS

Pengujian BFS

Path:
1. YAFM
2. Frequency
3. Psychoacoustics
4. Computer_games
5. Computer_Space

Degree: 4

Time: 49.2323448s

Wiki Visited: 403258

Path:
1. Harvard_University
2. Massachusetts_Institute_of_Technology

Degree: 1

Time: 69.3723ms

Path:
1. Lenovo
2. Panasonic
3. Indonesia

Degree: 2

Time: 13.662907s

Wiki Visited: 2232

Path:
1. Nazi_Germany
2. Empire
3. Japan

Degree: 2

Time: 21.5667141s

Wiki Visited: 141391

Path:

1. YAFM
2. Radio_station
3. Audience

Degree: 2

Time: 128.6677ms

Wiki Visited: 467

Path:

1. YAFM
2. Frequency
3. Transducer
4. Transmitter

Degree: 3

Time: 6.7683979s

Wiki Visited: 28660

Path:

1. Intel_Core
2. Workstation
3. Computer_network

Degree: 2

Time: 4.6231831s

Wiki Visited: 16939

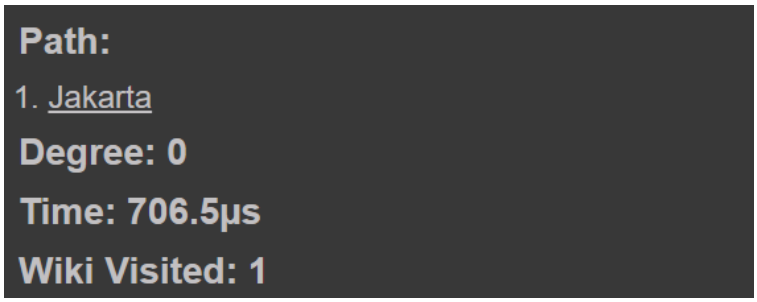
Path:

1. Taman_Impian_Jaya_Ancol
2. Ali_Sadikin

Degree: 1

Time: 992.4219ms

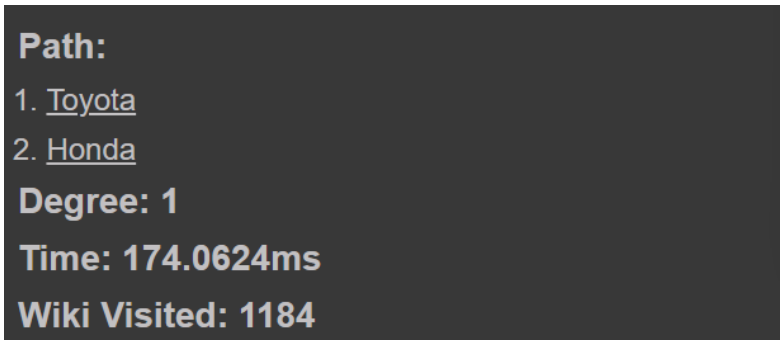
Wiki Visited: 22



Analisis Hasil Pengujian BFS

Berdasarkan pengujian, dapat dianalisis bahwa kompleksitas dapat meningkat secara eksponensial. Pada kasus best case pencarian pada degree yang lebih rendah bisa jadi lebih lama, tergantung jumlah link pada awal dan jumlah link berikutnya yang di scrape. Dan hasil pengujian di backend, ditemukan faktor utamanya adalah fetch link yang membutuhkan waktu yang banyak.

Pengujian IDS



Path:

1. Lenovo
2. Intel

Degree: 1

Time: 177.0764ms

Wiki Visited: 3

Path:

1. Indonesia
2. Jakarta

Degree: 1

Time: 271.4199ms

Wiki Visited: 3

Path:

1. Harvard_University
2. Massachusetts_Institute_of_Technology

Degree: 1

Time: 1.9177123s

Berdasarkan hasil pencarian menggunakan algoritma IDS, dapat dianalisis bahwa ada beberapa kasus di mana IDS jauh lebih cepat menemukan solusi daripada BFS karena IDS mengutamakan untuk mencari ke arah mendalam terlebih dahulu. Namun, ada juga kasus di mana IDS berjalan jauh lebih lambat daripada BFS.

Bab 5
Kesimpulan, Saran, dan Refleksi

Kesimpulan yang didapatkan dari tugas besar 2 ini adalah BFS dan IDS memiliki perbedaan yang cukup jauh untuk mencari path terpendek. IDS bekerja secara mendalam terlebih dahulu, sedangkan BFS bekerja secara melebar terlebih dahulu. Akibatnya, ada kemungkinan IDS bekerja jauh lebih cepat daripada BFS dan sebaliknya. Fetch link merupakan proses yang paling membutuhkan waktu. Dengan prinsip parallelism, fetch link dapat dipangkas waktunya. Proses pengeliminisian simpul sangat penting, karena seperti pisau bermata dua, bisa membuat lebih kencang, tapi bisa jadi menghilangkan informasi yang sedang dicari.

Refleksi untuk kami adalah membaca dokumentasi sangat penting, sehingga pemahaman untuk menggunakan library maupun menggunakan bahasa pemrograman akan lebih terstruktur.

Lampiran

Link Github: https://github.com/DeltDev/Tubes2_Kami-Mau-Csharp-dan-Unity
Link Video: Video Tubes 2 Stima
Passcode Video: AM&p+3r2

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
go-colly.org/docs/examples/basic/