

Laporan Tugas Besar 2
IF2211 Strategi Algoritma
Semester II Tahun 2023/2024

Pemanfaatan Algoritma *Breadth First Search* dan *Iterative Deepening Search* dalam Permainan *WikiRace*



Disusun oleh:

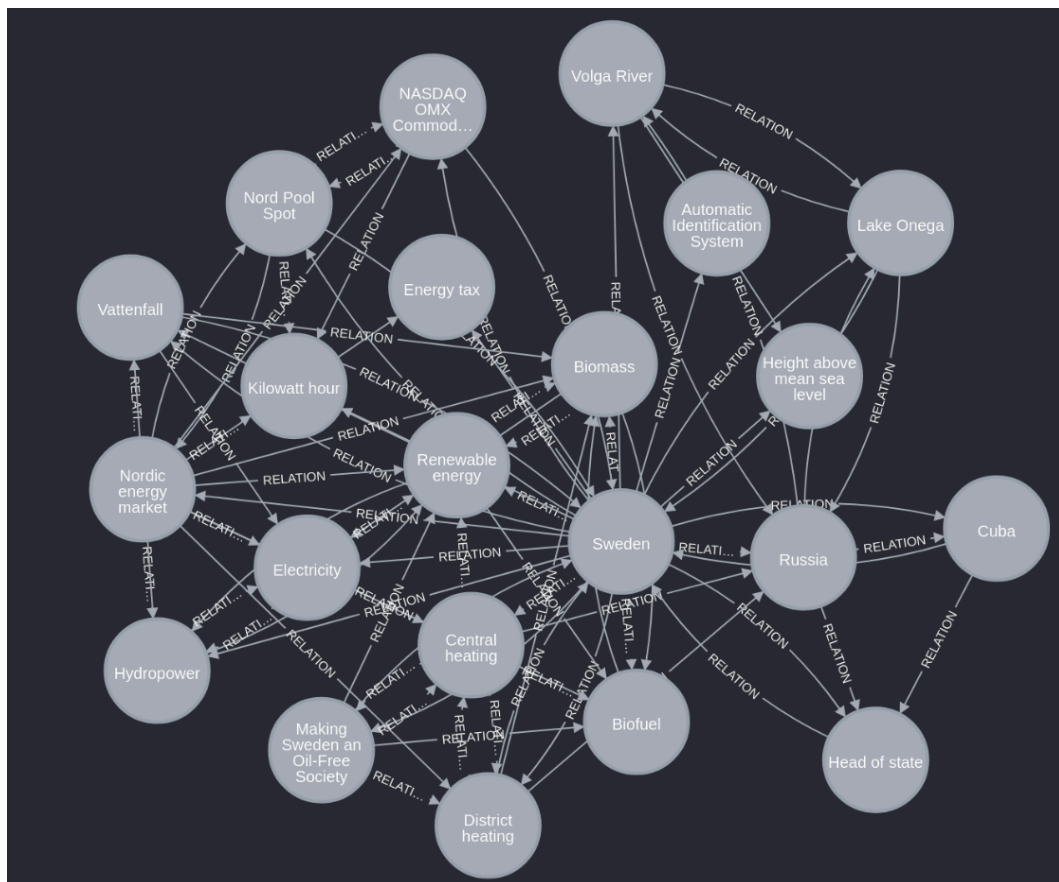
Haikal Assyauqi	13522052
Benjamin Sihombing	13522054
Muhammad Atpur Rafif	13522086

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
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.



Gambar 1. Ilustrasi WikiRace

BAB 2

LANDASAN TEORI

1. Definisi Traversal Graf

Traversal graf adalah sebuah proses untuk mengunjungi semua simpul di dalam sebuah graf, dan urutan pengunjungan akan mengklasifikasikan tipe traversal graf. Sebuah spesialisasi dari traversal graf adalah traversal pohon, karena struktur data pohon merupakan sebuah spesialisasi dari struktur data graf, yang tidak memiliki sirkuit. Dalam traversal graf, terkadang terdapat kondisi dimana sebuah simpul dikunjungi lebih dari sekali, dan semakin banyak simpul yang ada di dalam graf, akan semakin banyak pula redundansi yang akan terlihat. Dalam pencarian rute, terdapat beberapa cara yang dapat digunakan agar graf yang sudah dilewati oleh rute yang sama tidak dikunjungi kembali, salah satu caranya yang kelompok kami terapkan adalah dengan string matching dari arah rute. Dalam traversal graf, graf dapat merepresentasikan persoalan yang ingin dipecahkan, dan traversal graf adalah proses pencarian solusinya. Aplikasi dari traversal dapat bermacam-macam, beberapa contohnya adalah BFS ataupun IDS.

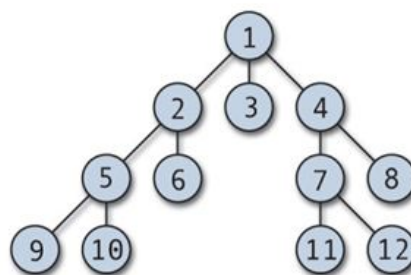
2. Algoritma *Breadth Depth Search*

Algoritma breadth-first search (BFS) adalah salah satu algoritma traversal pencarian pada graf. Algoritma ini mengunjungi simpul-simpul pada graf secara “melebar”.

Misalkan ada sebuah simpul v dari graf G . Akan dilakukan traversal graf G dengan algoritma BFS dimulai dari simpul v . Skema algoritmanya adalah sebagai berikut:

1. Kunjungi simpul v .
2. Kunjungi seluruh simpul yang bertetangga dengan simpul v .
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang sudah dikunjungi sebelumnya.
4. Lakukan langkah yang sama hingga seluruh simpul pada graf G telah dikunjungi

Agar lebih mudah dipahami, pada graf di bawah ini, urutan *node* yang dikunjungi jika menggunakan BFS yaitu 1,2,3,4,5,6,7,8,9,10,11,12.



Gambar 2. Graf

Berikut *pseudocode* dari algoritma *Breadth First Search*

```
{ Melakukan traversal graf dengan algoritma pencarian BFS. Masukan: v
adalah simpul awal traversal Luaran: Seluruh simpul graf dijelajahi
dengan algoritma pencarian BFS }
```

Deklarasi

```
w : simpul
q : antrian
```

```
procedure BuatAntrian(input/output q : antrian) { Membuat antrian
kosong }
procedure MasukAntrian(input/output q : antrian, input v :
simpul) { Memasukkan simpul v ke antrian q pada posisi belakang }
procedure HapusAntrian(input/output q : antrian, output v :
simpul) { Menghapus v dari kepala antrian q }
function AntrianKosong(input q : antrian) → boolean { Mengecek
apakah antrian q kosong }
```

Algoritma

```
BuatAntrian(q)
write(v)
Kunjungi(v) <- true
MasukAntrian(q,v) { Kunjungi semua simpul graf selama antrian
belum kosong }
while not AntrianKosong(q) do
    HapusAntrian(q,v)
    for (tiap simpul w yang bertetangga dengan simpul v) do
        if not dikunjungi(w) then
            Kunjungi(w)
            MasukAntrian(q,w)
        endif
    endfor
endwhile
{ AntrianKosong(q) }
```

3. Algoritma *Iterative Deepening Search*

Iterative Deepening Search merupakan pengembangan DFS yang memiliki nilai kedalaman *cutoff* hingga solusi ditemukan, dengan adanya nilai kedalaman ini, program memiliki batas hingga kedalaman tertentu

Berikut pseudocode dari IDS:

```
Depth <- 0
Iterate
    result <- DLS(problem,depth)
stop: result ≠ cutoff
    depth depth+1
-> result
```

4. WikiRace

WikiRace atau WikiRacing merupakan permainan yang bertemakan Wikipedia yang bertujuan untuk menghubungkan 2 topik yang tidak saling berhubungan menggunakan hyperlink yang tersedia pada laman Wikipedia, hal yang

dipertandingkan seperti *Wikispeedia* (WikiRacing yang mengandalkan kecepatan), *Wikimaze* (WikiRacing yang menghitung banyak langkah dari *start* menuju *finish*)

5. Golang

Pada tugas besar kali ini, bahasa yang digunakan yaitu Golang, Golang atau Go merupakan bahasa pemrograman yang dikembangkan oleh Google, Golang merupakan bahasa pemrograman open source yang dirancang untuk mempercepat pengembangan perangkat lunak dengan fokus pada kejelasan, kecepatan, dan efisiensi.

Keunggulan bahasa Golang dibanding bahasa lain:

1. Kode yang cukup simpel
2. *Powerful* dibanding kode lain
3. Bisa digunakan untuk *multi-core* processors
4. Mudah dipelajari
5. Mudah *dimaintenance*

BAB 3

ANALISIS PEMECAHAN MASALAH

A. Langkah-Langkah Pemecahan Masalah

Pada aplikasi, masalah yang ingin diselesaikan adalah mencari rute terpendek dari suatu laman Wikipedia ke suatu laman Wikipedia lainnya. Untuk mencari rute tersebut, aplikasi melakukan algoritma pencarian. Algoritma yang digunakan pada aplikasi ini adalah algoritma IDS dan BFS. Berikut ini adalah langkah-langkah pemecahan masalah pencarian tersebut:

1. Aplikasi akan menerima laman awal dan laman target dari masukan pengguna.
2. Laman awal akan menjadi *node* awal pada algoritma pencarian.
3. Laman saat ini akan dicek apakah merupakan laman target.
4. Jika iya, proses pencarian akan berhenti dan aplikasi akan menampilkan rute yang dilalui selama pencarian laman target.
5. Jika tidak, *scraping* data akan dilakukan dari laman tersebut untuk mencari tautan laman Wikipedia lainnya.
6. Tautan-tautan laman Wikipedia tersebut akan disimpan ke dalam kumpulan simpul baru.
7. Langkah kedua akan dilakukan ulang oleh setiap laman yang ada di kumpulan node baru.

B. Pemetaan Masalah Menjadi Elemen-Elemen IDS dan BFS

Pada saat pencarian IDS dan BFS, aplikasi tidak mengetahui berapa total simpul yang akan ditelusuri. Simpul baru akan muncul terus-menerus jika laman target belum ditemukan. Oleh karena itu, pencarian yang dilakukan representasikan permasalahan pencarian pada graf dinamis. Berikut ini pemetaan masalah dalam representasi pencarian pada graf dinamis:

- Operator: cek laman (laman akan dicek apakah merupakan laman target)
- Akar: laman awal
- Simpul: laman-laman yang bukan laman target
- Daun: laman yang merupakan laman target
- Ruang solusi: kumpulan laman target (namun pada permasalahan ini hanya terdapat 1 laman target)
- Ruang status: seluruh laman yang telah ditelusuri.

C. Fitur Fungsional dan Arsitektur dari *Website*

Tujuan dari aplikasi ini adalah untuk mengecek rute dari suatu laman Wikipedia ke laman Wikipedia lainnya. Jadi, fitur-fitur yang tersedia di *website* pun sedikit dan ringkas. Website ini hanya mengandung 2 fitur fungsional. Berikut ini fitur-fitur tersebut:

1. Kontainer Masukan

Pada *website* ini terdapat 2 buah kontainer masukan. Kontainer pertama untuk menerima laman awal dan kontainer kedua untuk menerima laman target.

2. Tombol Simpul

Simpul bisa ditekan untuk dilihat rute dari awal hingga ke simpul tersebut. Waktu ditemukannya simpul juga akan ditampilkan.

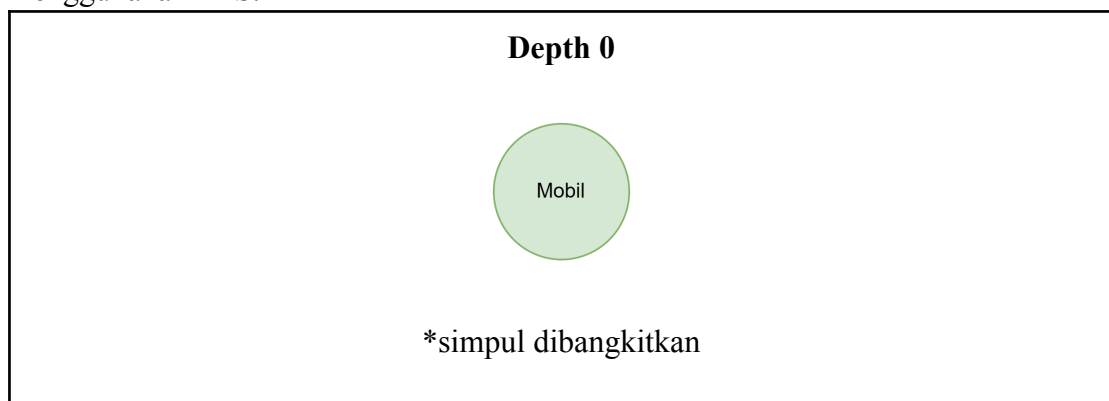
3. Pencarian

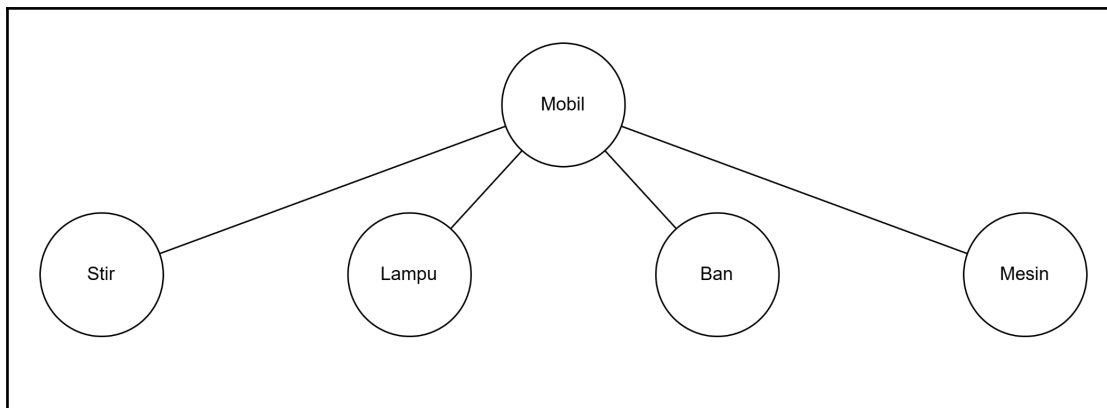
Karena aplikasi ini dapat mencari solusi dengan metode BFS atau IDS. *Website* menyediakan *drop-down* list untuk memilih jenis pencarian yang digunakan. Tombol *searching* juga disediakan untuk memulai pencarian.

Arsitektur yang digunakan pada program ini sangatlah simpel, hanya menggunakan html, css, dan js saja. Penulis program ingin mencoba kembali ke awal dan asal, tanpa direpotkan dengan melakukan bundling dan segala halnya. Karena arsitektur yang sangat simpel ini, maka seluruh *website* dapat dilakukan *embed* ke binary dari programnya. Sehingga kita hanya perlu mengirim file binary, tanpa mengirim folder *statik* untuk menjalankan *website*. Hal ini sangat mempermudah *deploy* menggunakan *docker*. Kemudian pada visualisasi data menggunakan *graph*, digunakan library yang bernama d3.js.

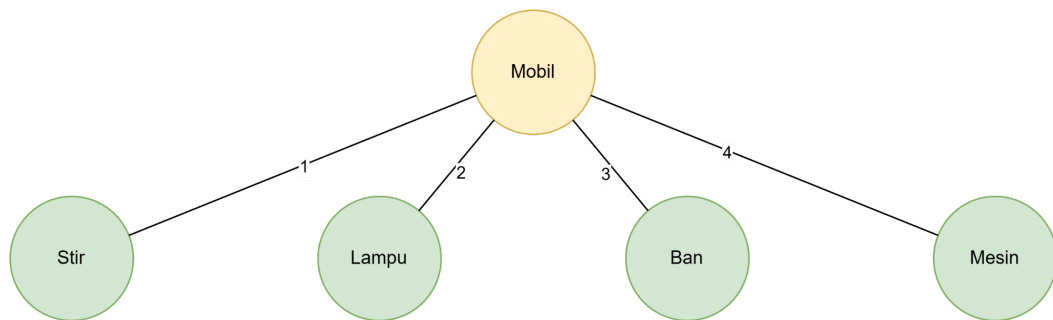
D. Ilustrasi Kasus

Misalnya pengguna ingin mencari rute dari laman “Mobil” ke laman “kayu”. Ilustrasi kasus terinspirasi dari laman Car, Steering_wheel, Vehicle, Vehicle_horn, Electric_light, Electronic_component, Light, Internal_combustion_engine, Combustion, Fuel, Wheel, Wood, dan Synthetic_rubber. Berikut ini proses pencarian menggunakan BFS.

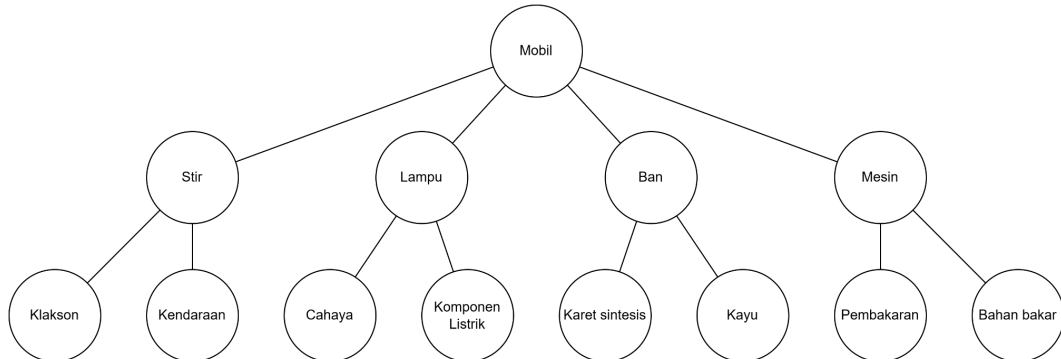




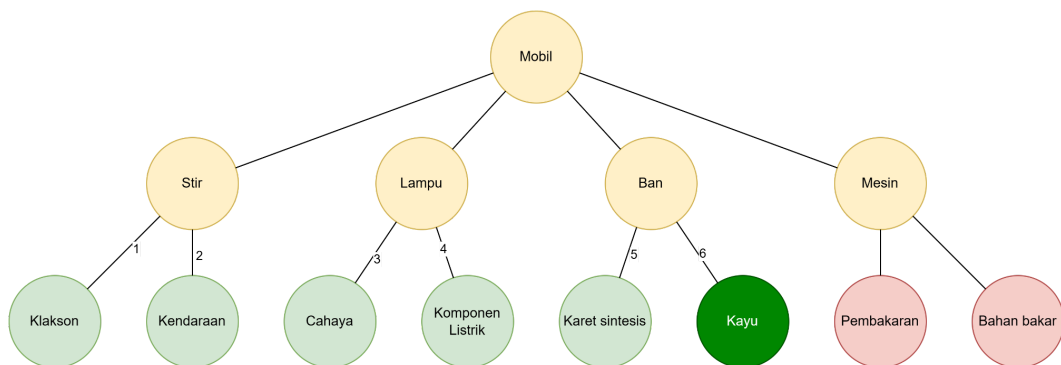
Depth 1



*simpul dibangkitkan



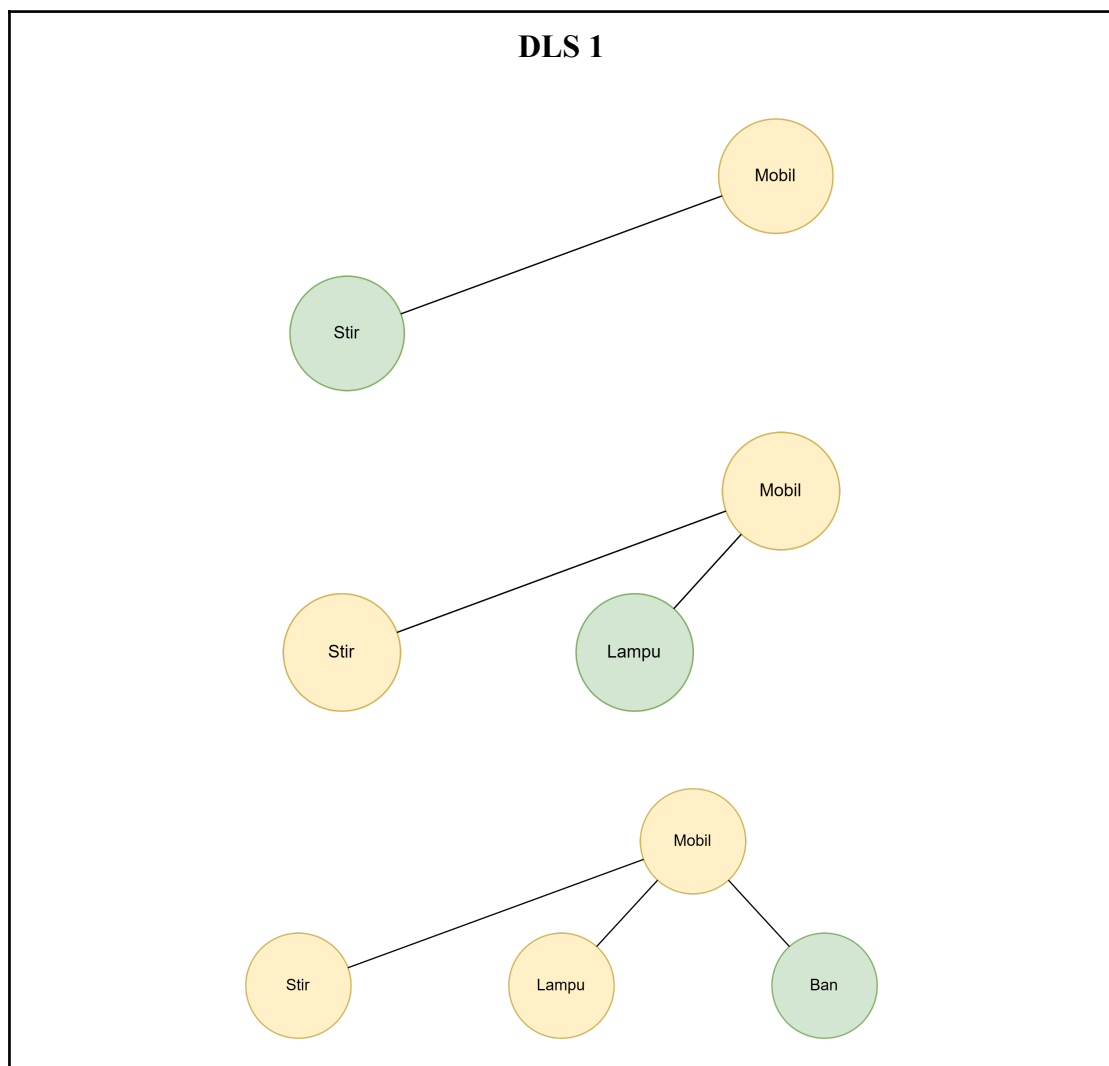
Depth 2

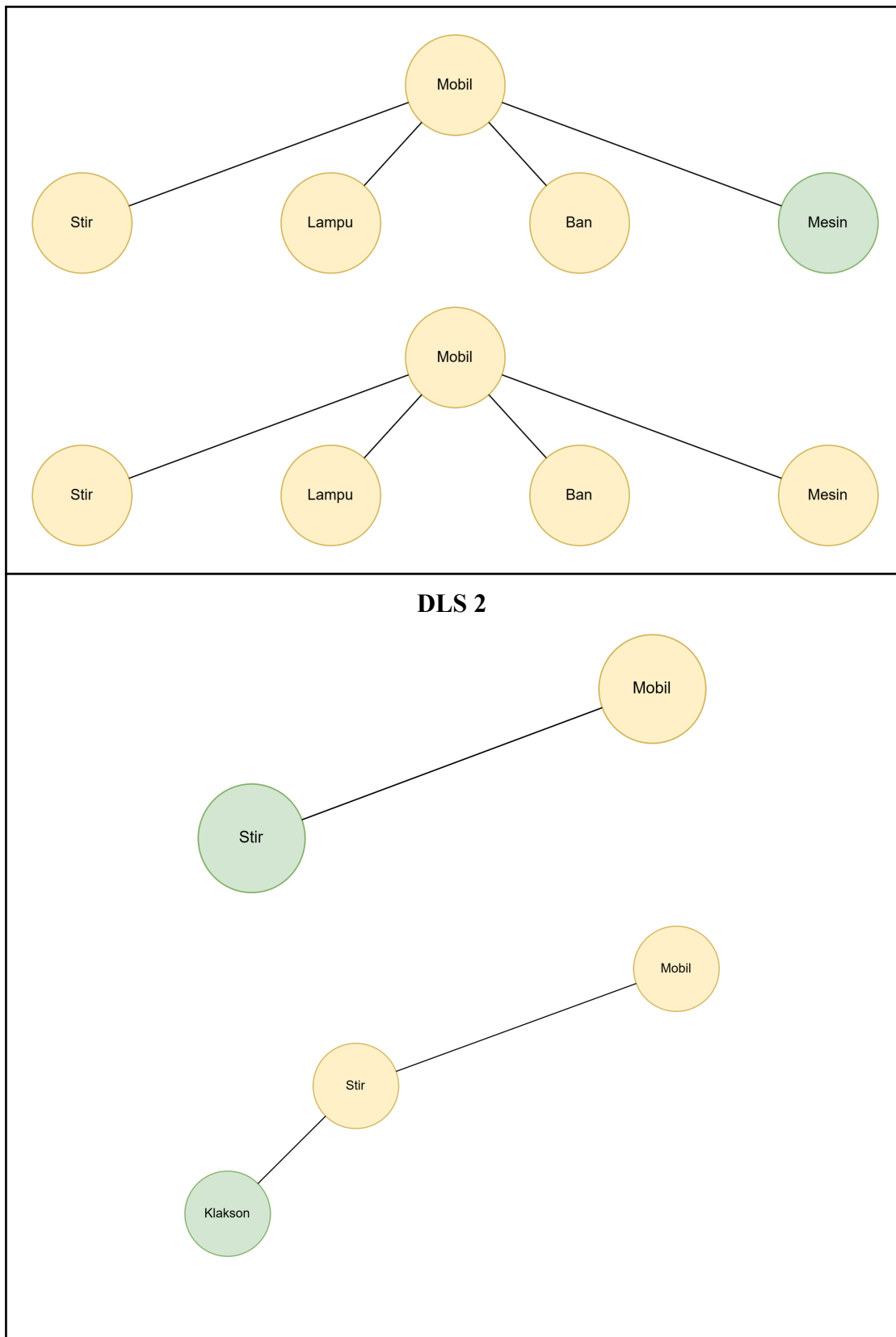


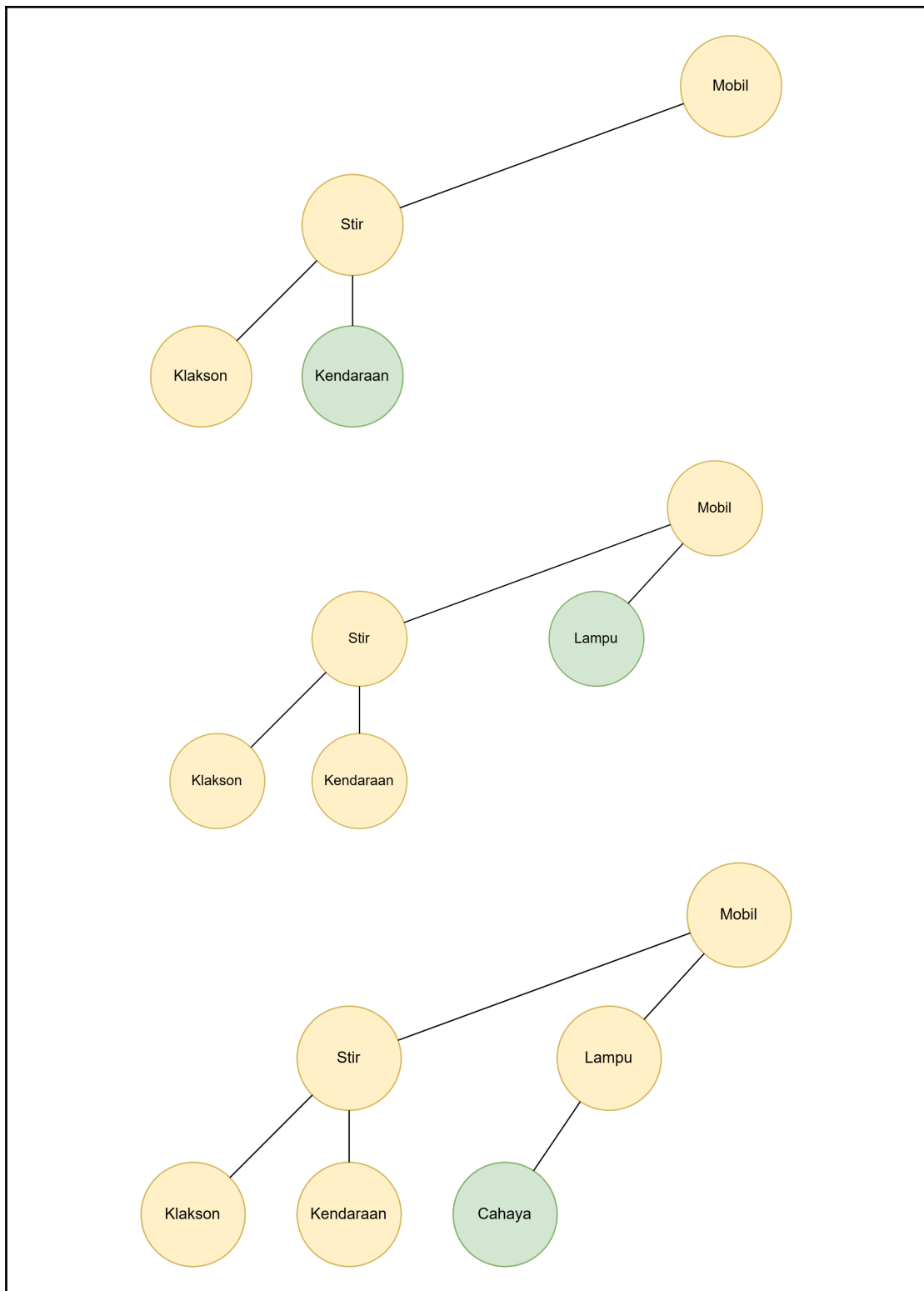
Gambar 3. Ilustrasi BFS

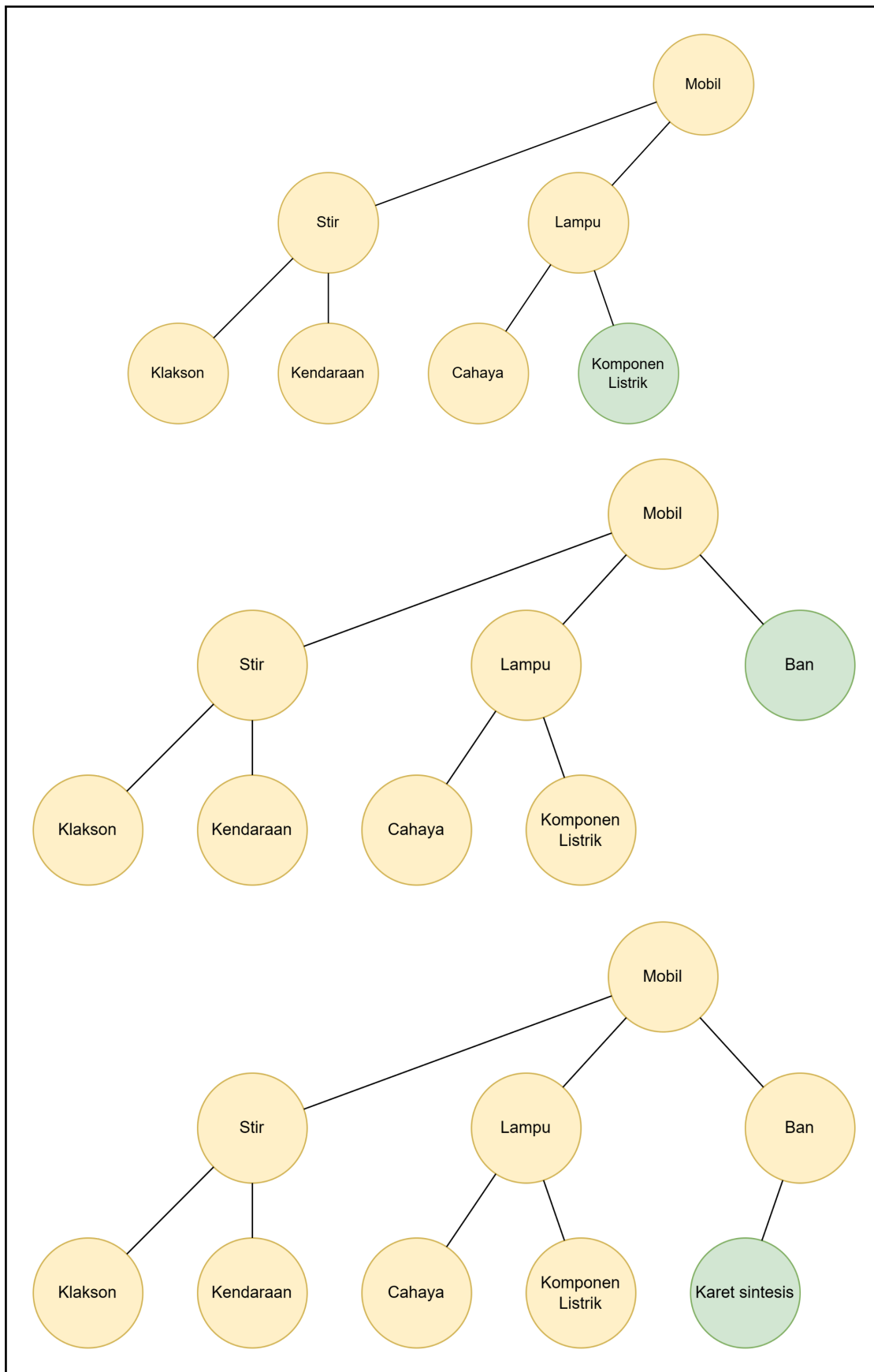
Pada proses BFS kasus di atas, pertama, simpul akar akan dicek apakah simpul merupakan. Simpul akar bukan solusi, maka simpul-simpul baru akan dibangkitkan dengan menggunakan *scraping* pada laman Mobil. Stir, Lampu, Ban, dan Mesin akan menjadi simpul di *depth* 1. Setiap simpul pada *depth* 1 akan dicek apakah simpul merupakan. Ternyata, pada *depth* 1 belum ditemukan solusi. Karena solusi masih belum ditemukan, simpul kembali dari seluruh simpul di *depth* 1. Sekarang, pada *depth* 2 terdapat simpul Klakson, Kendaraan, Cahaya, Komponen Listrik, Karet Sintesis, Kayu, Pembakaran, dan Bahan Bakar. Proses pencarian dilanjutkan ke *depth* 2. Setiap simpul akan dicek kembali. Pada simpul ke 6 *depth* 2, ternyata telah ditemukan laman target. Proses pencarian akan diberhentikan. Simpul Pembakaran dan Bahan Bakar tidak akan dikunjungi (dicek).

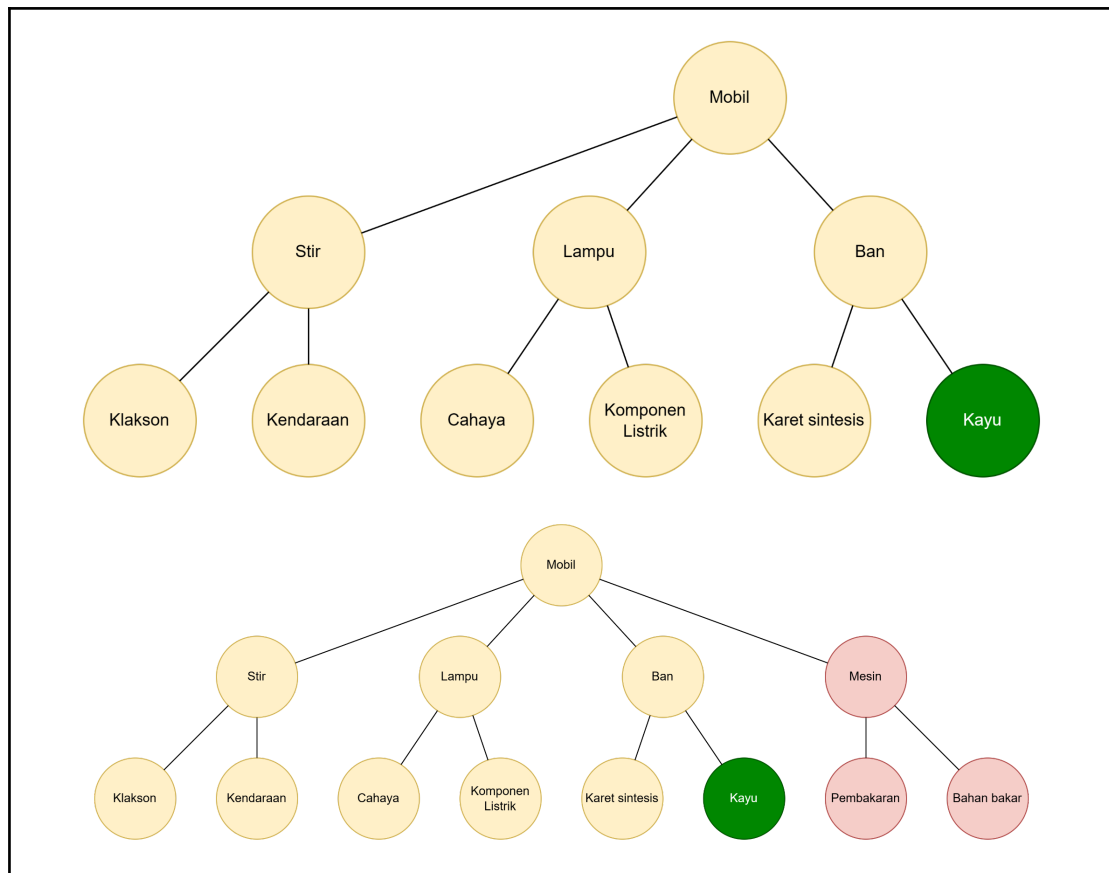
Pada algoritma IDS, pencarian akan melakukan iterasi algoritma DLS. Berikut ini proses pencarian menggunakan IDS.











Gambar 4. Ilustrasi IDS

Pada pencarian solusi, dilakukan DLS sebanyak 2 kali. DLS adalah algoritma yang mirip dengan DFS namun *depth*-nya dibatasi. Pada DLS 1, simpul Stir, Lampu, Ban, dan Mesin diperiksa (dicek). Namun, solusi tidak ditemukan. Iterasi dilanjutkan ke DLS 2. DLS 2 artinya DLS dengan *depth* limit 2. Simpul akan dicek hingga kedalaman 2. Jika sudah mencapai kedalaman 2 dan solusi tidak ditemukan, *backtracking* akan dilakukan. Hal ini bisa terlihat dari ilustrasi di atas. Setelah dilakukan pengecekan di simpul Klakson, ternyata solusi belum ditemukan. Maka, *backtracking* dilakukan ke simpul Stir dan dilanjutkan ke tetangga lainnya yaitu simpul Kendaraan.

E. Permasalahan Teknis

Pada tahap pengembangan, akan ditemukan beberapa permasalahan teknis sebagai berikut ini:

1. HTTP Request

Sebuah HTTP *request* oleh sebuah *client* hanya mengirimkan sebuah permintaan dan mendapatkan sebuah respons dari *server*. Pada aplikasi yang membutuhkan waktu yang sangat lama, seperti aplikasi *wikiracer* ini, baiknya informasi mengenai progress aplikasi ditampilkan kepada pengguna. Apabila kita hanya menampilkan *loading bar* saja, maka pengguna dapat berpikir bahwa aplikasi sedang rusak karena membutuhkan waktu yang lama.

Seperti yang disebutkan sebelumnya, sebuah HTTP *request* saja tidak cukup untuk menampilkan informasi mengenai progress aplikasi. Sebenarnya kita bisa mengimplementasikan *polling*, yaitu melakukan *request* dalam interval tertentu untuk mendapatkan progress dari aplikasi. Namun sudah terdapat teknologi yang dinamakan *websocket* untuk melakukan komunikasi dua arah.

Saat pengguna melakukan sebuah *request*, akan terdapat sebuah *prosedur* untuk mengubah protokol menjadi *websocket*. *Server* akan membuat dua buah *thread* atau *goroutine*. Pertama digunakan untuk menerima pesan dari *client*, sedangkan *goroutine* kedua digunakan untuk mengirimkan pesan menuju *client*. Apabila kita melakukan kedua proses tersebut dalam sebuah *thread*, maka akan terjadi *blocking*. Karena kita tidak tahu apabila *client* atau *server* terlebih dahulu yang akan mengirim pesan. Komunikasi antar *goroutine* dilakukan menggunakan *channel*.

Metode pencarian yang digunakan dilakukan di *goroutine* yang berbeda agar tidak melakukan *blocking* pada *websocket thread*. Terdapat sebuah *channel* yang akan menjadi parameter fungsi pencarian. Apabila terdapat informasi baru, maka fungsi pencarian tersebut akan mengirimkan pesan melewati *channel* pada parameter, yang nantinya akan diteruskan menuju *client*. Sehingga *client* akan melihat informasi terbaru mengenai progress aplikasi. Terlebih lagi ketika mencari banyak jalur, akan membutuhkan waktu lebih lama lagi.

2. Data Race

Salah satu permasalahan dan penyebab *bug* yang sering terjadi pada aplikasi yang menggunakan *thread* lebih dari satu adalah *data race*. Hal ini terjadi ketika urutan selesainya *thread* mempengaruhi hasil akhir dari program. Terlebih lagi pada aplikasi ini membutuhkan *fetch* ke laman Wikipedia. Selain itu, perbedaan dari traversal *graph* adalah urutan ekspansi untuk simpul selanjutnya, dengan adanya *data race*, maka urutan simpul ekspansi tidak bisa diprediksi.

Oleh karena itu, kami memisahkan program bagian *fetching* dan *searching* (atau *traversal*) untuk menghindari ekspansi simpul yang tidak bisa diprediksi. Bagian program yang bertanggung jawab untuk melakukan *fetching* disebut dengan *prefetcher*. Pada saat berjalannya program, fungsi *prefetcher* sama sekali tidak mengubah struktur data yang digunakan untuk pencarian. Namun hanya untuk melakukan *fetching* ke laman Wikipedia, dan mengirimkan hasilnya menggunakan *channel*. Fungsi *prefetcher* membutuhkan informasi mengenai laman yang diperlukan pada saat *searching*. Nantinya fungsi *prefetcher* akan membuat beberapa *goroutine* sesuai sampai batas yang telah ditentukan.

Fungsi *searching* akan memeriksa *map* yang memiliki *key* berupa laman, dan *value* berupa seluruh *link* yang terdapat pada laman tersebut. Misalkan pada saat ini fungsi *searching* sedang melakukan ekspansi terhadap

simpul A. Maka *map* akan diperiksa, apabila simpul tersebut sudah ada, maka fungsi *searching* dilanjutkan. Namun apabila belum ditemukan, fungsi *searching* akan melakukan *blocking*, dan menunggu pesan dari *goroutine* yang telah dibuat oleh fungsi *prefetcher*. Apabila pesan yang ditemukan merupakan hasil dari *fetching* simpul A, maka fungsi *searching* akan dilanjutkan. Namun jika belum, maka akan terus melakukan *blocking* hingga simpul A selesai dilakukan *fetching*. Seluruh pesan yang diterima dari *goroutine* tersebut disimpan pada *map*.

Prosedur yang baru dijelaskan terdengar sama saja dengan program yang tidak memanfaatkan *goroutine*. Namun perhatikan bahwa, seluruh pesan akan disimpan pada *map*. Saat melakukan *blocking* untuk menunggu A, simpul lain selesai *fetching*. Sehingga pada saat melakukan ekspansi simpul lain tersebut, program tidak perlu melakukan *blocking*. Hal ini menjamin ekspansi program terprediksi dan sesuai dengan alur definisinya. Program yang dijalankan dua kali dengan simpul awal dan akhir yang sama, akan selalu menghasilkan jalur yang dengan urutan ditemukan sama.

3. *Canonical link*

Sebuah laman pada Wikipedia dapat memiliki lebih dari satu nama yang disebut dengan *redirect*. Hal ini berarti kita tidak bisa selalu mempercayai hasil seluruh *link* yang didapat ketika melakukan *fetching*, kecuali dengan mengunjungi *link* tersebut. Pada bagian *head* di laman Wikipedia, terdapat sebuah *link tag* yang disebut dengan *canonical*. *Tag* ini memiliki atribut *href* yang berisi *link* utama yang digunakan untuk mengunjungi laman tersebut. Misalkan apabila kita mencari Hitler, maka akan diteruskan ke laman *Adolf_Hitler*.

Pada algoritma yang dijelaskan sebelumnya, apabila seluruh *link* dapat dipercaya, atau dengan kata lain tidak terdapat *redirect*. Terdapat optimisasi, yaitu ketika jalur terpendek memiliki kedalaman n , maka kita bisa berhenti melakukan pencarian pada kedalaman $n - 1$. Karena pada titik tersebut, kita telah menemukan simpul tujuan saat melakukan *fetching* dan *scrapping*. Namun dengan adanya *redirect*, optimisasi ini tidak bisa dilakukan. Karena kita harus memvalidasi seluruh *link* yang dihasilkan pada kedalaman $n - 1$ menjadi atau tidak *redirect* menuju laman tujuan.

Oleh karena itu, kita mengambil titik tengah dari keduanya, yaitu menggunakan dua cara yang dijelaskan sebelumnya. Apabila pada kedalaman $n - 1$, sebuah simpul memiliki *link* menuju laman tujuan, maka simpul tersebut tidak akan diekspansi dan dimasukkan jalur dimasukkan kedalam jalur hasil. Namun jika tidak, maka akan dilakukan validasi pada kedalaman n . Hal ini membutuhkan waktu sangat lama, sehingga pada pengetesan nanti, ketika sebuah jalur ditemukan, maka program akan dihentikan, terlepas validasi selesai dilakukan atau tidak.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

A. Struktur Data, Fungsi, dan Prosedur

1. Struktur Data

```
// type.go
type WikipediaResponse struct {
    Continue *(struct {
        Continue string
        Plcontinue string
    })
    Query struct {
        Pages map[string](struct {
            Title string
            Links [] (struct {
                Title string
            })
        })
    }
}

const (
    Log = iota + 1
    Start
    Found
    End
    Error
)

type Status uint8

type Request struct {
    Start string
    End string
    Type string
    Cancel bool
}

type Response struct {
    Status Status `json:"status"`
    Message interface{} `json:"message"`
}

// searchBFS.go
type FetchResult struct {
    From string
    Canonical string
    To []string
}

type StateBFS struct {
    Start string
    End string
    ResultPaths [][]string
    ResultDepth int
    Queue [][]string
    FetchedCount int // Optimization to start searching for unfetched
```



```

data
    FetchedData map[string][]string
    Canonical    map[string]string
    FetchChannel chan FetchResult
    Visited      map[string]bool
    Running      int
}

// searchIDS.go

type StateIDS struct {
    Start      string
    End        string
    CanonicalEnd string
    ResultPaths [][]string
    Path        []string
    PathSet     map[string]bool
    Canonical   map[string]string

    FetchedData map[string][]string
    FetchChannel chan FetchResult
    CurrentFetch []string
    NextFetch    []string

    ForceQuit bool
    ForceQuitFetch bool
    ForceQuitFetchMutex sync.Mutex

    MaxDepth int
    ResultDepth int
}

```

2. Function

```

// type.go
func (s Status) String() string {
    switch s {
    case Log:
        return "update"
    case Start:
        return "started"
    case Found:
        return "found"
    case End:
        return "finished"
    default:
        return "error"
    }
}

func (s Status) MarshalJSON() ([]byte, error) {
    return json.Marshal(s.String())
}

// links.go
func parsePage(to string) (string, bool) {
    if !strings.HasPrefix(to, WIKI) {
        return "", false
    }
}

```

```

    }

    rel, err := filepath.Rel(WIKI, to)
    if err != nil {
        return "", false
    }

    if strings.ContainsAny(rel, ":#") {
        return "", false
    }

    page, err := url.QueryUnescape(rel)
    if err != nil {
        return "", false
    }

    return page, true
}

// TODO: Filter namespace
func filterPages(links []string) Pages {
    pages := make([]string, 0)

    visited := make(map[string]bool)
    for _, to := range links {
        page, ok := parsePage(to)

        if !ok {
            continue
        }

        if visited[page] {
            continue
        }
        visited[page] = true

        pages = append(pages, page)
    }

    return pages
}

func hash(str string) string {
    hasher := sha256.New()
    hasher.Write([]byte(str))
    return base32.StdEncoding.EncodeToString(hasher.Sum(nil))
}

func fileExist(path string) bool {
    _, err := os.Stat(path)
    return err == nil
}

func createCachePath(str string) string {
    return CACHE_DIR + "/" + hash(str)
}

func readCache(page string) (string, []string, bool) {
    time.Sleep(100 * time.Millisecond)

```

```

    pagePath := createCachePath(page)
    if !fileExist(pagePath) {
        return "", nil, false
    }

    cacheByte, err := os.ReadFile(pagePath)
    cache := string(cacheByte)
    if err != nil {
        return "", nil, false
    }

    if len(cache) == 0 {
        os.Remove(pagePath)
        return "", nil, false
    }

    if cache[0] != '\n' {
        return page, strings.Split(cache, "\n"), true
    }

    canon := strings.Replace(cache, "\n", "", 1)
    canonPath := createCachePath(canon)
    if !fileExist(canonPath) {
        return "", nil, false
    }

    cacheCanonByte, err := os.ReadFile(canonPath)
    if err != nil {
        return "", nil, false
    }

    cacheCanon := string(cacheCanonByte)
    return canon, strings.Split(cacheCanon, "\n"), true
}

func getLinks(page string) (string, Pages) {
    // time.Sleep(300 * time.Millisecond)
    // P := make(map[string][]string)
    // P["Adolf_Hitler"] = []string{"B_"}
    // P["Hitler"] = []string{"B_"}
    // P["B"] = []string{"C", "Hitler", "D"}
    // P["B_"] = []string{"C", "Hitler", "D"}
    // P["C"] = []string{"D", "B_"}
    // P["D"] = []string{"E", "B", "F"}
    // P["E"] = []string{"Traffic"}
    // P["F"] = []string{"Traffic"}
    //
    // canon := page
    // if page == "Hitler" {
    //     canon = "Adolf_Hitler"
    // }
    // if page == "Traffic_" {
    //     canon = "Traffic"
    // }
    // if page == "B_" {
    //     canon = "B"
    // }
    //
    // return canon, P[page]

```

```

var canon string
var pages []string

// canon, pages, ok := readCache(page)
// if ok {
//   return canon, pages
// }

canonURL, pagesURL := scrap(WIKI + url.PathEscape(page))
pages = filterPages(pagesURL)
canon, parseOk := parsePage(canonURL)
if !parseOk {
    canon = page
}

// writeCache(page, canon, pages)

return canon, pages
}

// scrapper.go
func toAbsUrl(from, to *url.URL) url.URL {
    toStr := to.String()
    if to.IsAbs() {
        return *to
    } else {
        to.Scheme = from.Scheme
        if !strings.HasPrefix(toStr, "///") {
            to.Host = from.Host
        }
        return *to
    }
}

func scrap(urlStr string) (string, []string) {
    result := make([]string, 0)
    from, err := url.Parse(urlStr)

    canonical := ""
    if err != nil {
        log.Println("Scrapper can't parse URL " + urlStr)
        return canonical, result
    }

    // log.Println("[Scrapper] Visiting " + urlStr)
    response, err := http.Get(urlStr)
    if err != nil {
        log.Println("Scrapper can't visit URL " + urlStr)
        return canonical, result
    }
    defer response.Body.Close()

    insideMain := false
    tokenizer := html.NewTokenizer(response.Body)
    count := 0
    for {
        token := tokenizer.Next()
        if token == html.ErrorToken {
            break
        }
    }
}

```

```

name, _ := tokenizer.TagName()
if bytes.Equal(name, []byte("main")) {
    if token == html.StartTagToken {
        insideMain = true
    } else if token == html.EndTagToken {
        insideMain = false
    }
}

if bytes.Equal(name, []byte("link")) {
    isCanonical := false
    var href []byte
    for {
        key, value, next := tokenizer.TagAttr()
        if bytes.Equal(key, []byte("rel")) &&
bytes.Equal(value, []byte("canonical")) {
            isCanonical = true
        }

        if bytes.Equal(key, []byte("href")) {
            href = value
        }

        if !next {
            break
        }
    }

    if isCanonical {
        canonical = string(href)
    }
}

if !insideMain {
    continue
}

if bytes.Equal(name, []byte("a")) {
    for {
        key, value, next := tokenizer.TagAttr()
        if bytes.Equal(key, []byte("href")) {
            str := string(value)
            to, err := url.Parse(str)
            if err != nil {
                log.Println("Scraper can't parse URL " +
str)

                continue
            }
            absTo := toAbsUrl(from, to)
            result = append(result, absTo.String())
        }

        if !next {
            break
        }
    }
    count += 1
}
}

```

```

    // log.Println("[Scrapper] Links found in "+urlStr+":",
    len(result))

    return canonical, result
}

```

3. Prosedur

```

// links.go
func writeCache(page, canon string, pages []string) {
    os.MkdirAll("cache", os.ModePerm)
    go func() {
        pagePath := createCachePath(page)
        canonPath := createCachePath(canon)

        if page != canon {
            if fileExist(pagePath) {
                return
            }
            os.WriteFile(pagePath, []byte("\n"+canon), os.ModePerm)
        }

        if fileExist(canonPath) {
            return
        }
        os.WriteFile(canonPath, []byte(strings.Join(pages, "\n")),
os.ModePerm)
    }()
}

// searchBFS.go

func (s *StateBFS) prefetch() {
    for s.FetchedCount < len(s.Queue) {
        path := s.Queue[s.FetchedCount]
        current := path[len(path)-1]

        if _, found := s.FetchedData[current]; !found {
            if s.Running >= MAX_CONCURRENT {
                if s.FetchedCount == 0 {
                    r := <-s.FetchChannel
                    s.Canonical[r.From] = r.Canonical
                    s.FetchedData[r.Canonical] = r.To

                    s.Running -= 1
                } else {
                    break
                }
            }

            s.Running += 1
            go func() {
                canonical, pages := getLinks(current)
                s.FetchChannel <- FetchResult{
                    From:      current,
                    To:          pages,
                    Canonical: canonical,
                }
            }()
            s.FetchedCount += 1
        }
    }
}

```

```

    }
}

func SearchBFS(start, end string, responseChan chan Response,
forceQuit chan bool) {
    responseChan <- Response{
        Status: Start,
        Message: "Started...",
    }

    canonicalEnd, _ := getLinks(end)

    s := StateBFS{
        Start:      start,
        End:         canonicalEnd,
        ResultPaths: make([][]string, 0),
        Queue:       make([][]string, 0),
        FetchedData: make(map[string][]string),
        FetchChannel: make(chan FetchResult),
        Canonical:   make(map[string]string),
        Visited:     make(map[string]bool),
        FetchedCount: 0,
        Running:     0,
        ResultDepth: -1,
    }

    s.Queue = append(s.Queue, []string{start})

    for {
        if len(s.Queue) == 0 {
            break
        }

        s.prefetch()
        path := s.Queue[0]
        s.Queue = s.Queue[1:]
        depth := len(path) - 1
        current := path[depth]
        s.FetchedCount -= 1

        if s.ResultDepth != -1 && depth > s.ResultDepth {
            break
        }

        if canonical, found := s.Canonical[current]; found {
            current = canonical
        }
        if s.Visited[current] {
            continue
        }
        path[depth] = current
        s.Visited[current] = true

        for {
            if _, found := s.FetchedData[current]; found {
                path[depth] = current
                s.Visited[current] = true

                statusLog := ""
                if s.ResultDepth == depth {

```

```

        statusLog = "\nValidating " + current
    } else {
        statusLog = "\nTraversed: " + strings.Join(path,
" - ")
    }

    responseChan <- Response{
        Status: Log,
        Message: "Visited article count: " +
strconv.Itoa(len(s.FetchedData)) +
        "\nDepth: " + strconv.Itoa(depth) +
        statusLog,
    }

    var result []string = nil
    if current == canonicalEnd && (s.ResultDepth == -1 ||
s.ResultDepth == depth) {
        result = path
        s.ResultDepth = depth
    }

    newPaths := make([][]string, 0)

    for _, next := range s.FetchedData[current] {
        newPath := make([]string, len(path))
        copy(newPath, path)
        newPath = append(newPath, next)

        if next == canonicalEnd && (s.ResultDepth == -1
|| s.ResultDepth == depth+1) {
            result = newPath
            s.ResultDepth = depth + 1
            break
        }

        newPaths = append(newPaths, newPath)
    }

    if result != nil {
        s.ResultPaths = append(s.ResultPaths, result)
        responseChan <- Response{
            Status: Found,
            Message: result,
        }
    } else {
        for _, newPath := range newPaths {
            s.Queue = append(s.Queue, newPath)
        }
    }

    break
}

select {
case <-forceQuit:
    return
case r := <-s.FetchChannel:
    s.Canonical[r.From] = r.Canonical
    s.FetchedData[r.Canonical] = r.To

```



```

        s.Running -= 1
        s.prefetch()

        if current == r.From {
            current = r.Canonical
        }
    }
}

responseChan <- Response{
    Status: End,
    Message: "Search finished",
}

}

// searchIDS.go

func prefetcherIDS(s *StateIDS) {
    i := 0
    running := 0
    finishedChan := make(chan bool)
    for i < len(s.CurrentFetch) {
        s.ForceQuitFetchMutex.Lock()
        if s.ForceQuitFetch {
            return
        }
        s.ForceQuitFetchMutex.Unlock()

        current := s.CurrentFetch[i]
        if running >= MAX_CONCURRENT {
            <-finishedChan
            running -= 1
        }

        running += 1
        go func() {
            canon, pages := getLinks(current)
            s.FetchChannel <- FetchResult{
                From: current,
                To: pages,
                Canonical: canon,
            }
            finishedChan <- true
        }()

        i += 1
    }

    for running > 0 {
        <-finishedChan
        running -= 1
    }
}

func traverserIDS(s *StateIDS, responseChan chan Response, forceQuit
chan bool) {
    if s.ForceQuit {
        return
    }
}

```

```

depth := len(s.Path) - 1
current := s.Path[depth]

if canonical, found := s.Canonical[current]; found {
    current = canonical
}

if depth == s.MaxDepth {
    for {
        if pages, found := s.FetchedData[current]; found {
            s.Path[depth] = current

            statusLog := ""
            if s.ResultDepth == depth {
                statusLog = "\nValidating " + current
            } else {
                statusLog = "\nTraversed: " +
strings.Join(s.Path, " - ")
            }

            responseChan <- Response{
                Status: Log,
                Message: "Visited article count: " +
strconv.Itoa(len(s.FetchedData)) +
                "\nIteration: " + strconv.Itoa(s.MaxDepth) +
                "\nDepth: " + strconv.Itoa(depth) +
                statusLog,
            }

            var result []string = nil
            if current == s.CanonicalEnd && (s.ResultDepth == -1
|| s.ResultDepth == depth) {
                result = s.Path
                s.ResultDepth = depth
            }

            if s.ResultDepth == -1 || depth <= s.ResultDepth {
                for _, next := range pages {
                    if next == s.CanonicalEnd && (s.ResultDepth
== -1 || s.ResultDepth == depth+1) {
                        path := make([]string, len(s.Path))
                        copy(path, s.Path)
                        path = append(path, next)

                        result = path
                        s.ResultDepth = depth + 1
                        continue
                    }

                    s.NextFetch = append(s.NextFetch, next)
                }
            }

            if result != nil {
                responseChan <- Response{
                    Status: Found,
                    Message: result,
                }
            }
        }
    }
}

```

```

        s.ResultPaths = append(s.ResultPaths, result)
        return
    }

    break
}

select {
case <-forceQuit:
    s.ForceQuit = true
    s.ForceQuitFetchMutex.Lock()
    s.ForceQuitFetch = true
    s.ForceQuitFetchMutex.Unlock()
    return
case r := <-s.FetchChannel:
    s.FetchedData[r.Canonical] = r.To
    s.Canonical[r.From] = r.Canonical

    if r.From == current {
        current = r.Canonical
    }
}
}
} else {
    if pages, found := s.FetchedData[current]; found {
        for _, next := range pages {
            if canonical, found := s.Canonical[next]; found {
                next = canonical
            }

            if s.PathSet[next] {
                continue
            }

            s.Path = append(s.Path, next)
            s.PathSet[next] = true
            traverserIDS(s, responseChan, forceQuit)
            delete(s.PathSet, next)
            s.Path = s.Path[:len(s.Path)-1]
        }
    } else {
        panic("Non-leaf node should be cached in previous
iteration, call this function from depth 0")
    }
}
}

func SearchIDS(start, end string, responseChan chan Response,
forceQuit chan bool) {
    responseChan <- Response{
        Status: Start,
        Message: "Started...",
    }

    s := StateIDS{
        Start:      start,
        End:         end,
        ResultPaths: make([][]string, 0),
        Path:        make([]string, 0),
        PathSet:     make(map[string]bool),
    }

```

```

        Canonical:    make(map[string]string),
        FetchedData:  make(map[string][]string),
        FetchChannel: make(chan FetchResult),
        CurrentFetch: make([]string, 0),
        NextFetch:    make([]string, 0),
        MaxDepth:     0,
        ResultDepth:  -1,
    }

    s.Path = append(s.Path, s.Start)
    s.NextFetch = append(s.NextFetch, s.Start)
    canonicalEnd, _ := getLinks(end)
    s.CanonicalEnd = canonicalEnd

    for s.ResultDepth == -1 || s.MaxDepth <= s.ResultDepth {
        s.CurrentFetch = make([]string, 0)
        for _, nextFetch := range s.NextFetch {
            if _, found := s.FetchedData[nextFetch]; !found {
                s.CurrentFetch = append(s.CurrentFetch, nextFetch)
            }
        }

        s.NextFetch = make([]string, 0)
        go prefetcherIDS(&s)
        traverserIDS(&s, responseChan, forceQuit)
        if s.ForceQuit {
            return
        }

        s.MaxDepth += 1
    }

    responseChan <- Response{
        Status: End,
        Message: "Search finished",
    }
}

// main.go

func main() {
    http.HandleFunc("/api", func(w http.ResponseWriter, r
    *http.Request) {
        conn, err := upgrader.Upgrade(w, r, nil)
        if err != nil {
            log.Println(err)
            return
        }

        write := make(chan Response)
        read := make(chan Request)
        wsQuit := make(chan bool)

        go func(conn *websocket.Conn) {
            for {
                msgType, msg, err := conn.ReadMessage()
                if err != nil {
                    log.Println(err)
                    wsQuit <- true
                    break
                }
            }
        }(conn)

        for {
            select {
                case req := <= 1
    }

    responseChan <- Response{
        Status: End,
        Message: "Search finished",
    }
}

// main.go

func main() {
    http.HandleFunc("/api", func(w http.ResponseWriter, r
    *http.Request) {
        conn, err := upgrader.Upgrade(w, r, nil)
        if err != nil {
            log.Println(err)
            return
        }

        write := make(chan Response)
        read := make(chan Request)
        wsQuit := make(chan bool)

        go func(conn *websocket.Conn) {
            for {
                msgType, msg, err := conn.ReadMessage()
                if err != nil {
                    log.Println(err)
                    wsQuit <- true
                    break
                }
            }
        }(conn)

        for {
            select {

```

```

    }

    if msgType == websocket.TextMessage {
        var request Request
        json.Unmarshal(msg, &request)

        if request.Cancel {
            read <- request
            continue
        }

        if len(request.Start) == 0 || len(request.End) ==
0 {
            write <- Response{
                Status: Error,
                Message: `Empty "start" or "end" of
field`,
            }
            continue
        }

        if request.Type != "BFS" && request.Type != "IDS"
{
            write <- Response{
                Status: Error,
                Message: "Invalid method",
            }
            continue
        }

        read <- request
    }
} (conn)

go func(conn *websocket.Conn) {
    for {
        select {
            case <-wsQuit:
                break
            case msg := <-write:
                conn.WriteJSON(msg)
        }
    }
} (conn)

running := false
finished := make(chan bool)
runQuit := make(chan bool)
for {
    select {
        case <-finished:
            running = false
        case <-wsQuit:
            log.Println("End")
            runQuit <- true
            break
        case req := <-read:
            if running {
                if req.Cancel {

```

```

        runQuit <- true
    } else {
        write <- Response{
            Status: Error,
            Message: "Program still running",
        }
    }
    continue
}

running = true
go func() {
    var fn TraverseFunction
    if req.Type == "BFS" {
        fn = SearchBFS
    } else if req.Type == "IDS" {
        fn = SearchIDS
    } else {
        log.Panic("Invalid method")
    }
    fn(req.Start, req.End, write, runQuit)
    finished <- true
}()
}

// content, _ := fs.Sub(static, "static")
http.Handle("/", http.FileServer(http.Dir("static")))
// http.Handle("/", http.FileServer(http.FS(content)))
log.Println("Listening on port 3000")
err := http.ListenAndServe(":3000", nil)
if err != nil {
    panic(err)
}
}

```

B. Tata Cara Menggunakan Aplikasi

Untuk menjalankan program ini, ada beberapa aplikasi (*software*) yang diperlukan. Komputer yang menjalankan aplikasi harus sudah terinstall aplikasi peramban dan bahasa pemrograman Go. Berikut tata cara menggunakan aplikasi ini:

1. Lakukan clone repository [gogogo](#) ini.
2. Buka terminal atau command prompt.
3. Ganti directory ke Tubes2_gogogo.
4. Masukkan command

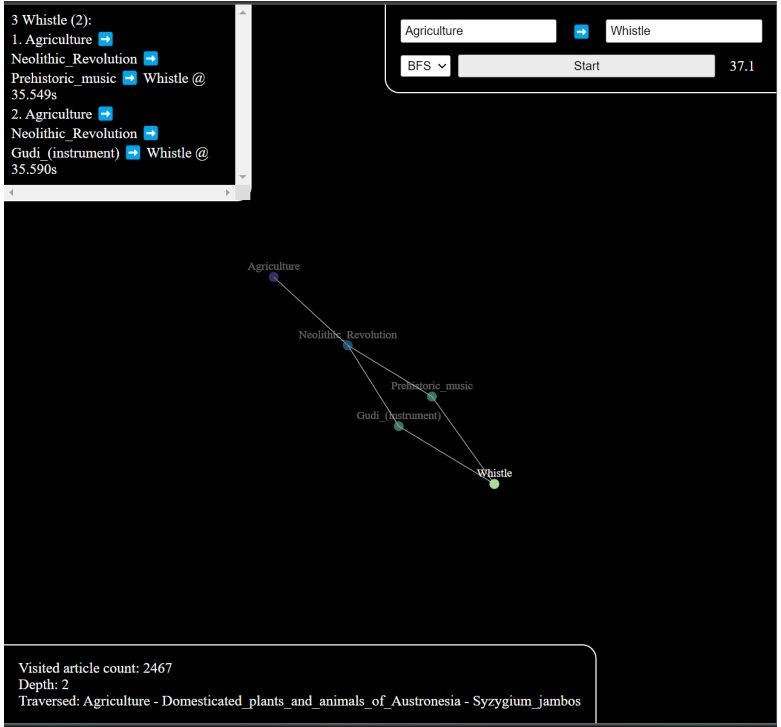
```
docker compose up
```

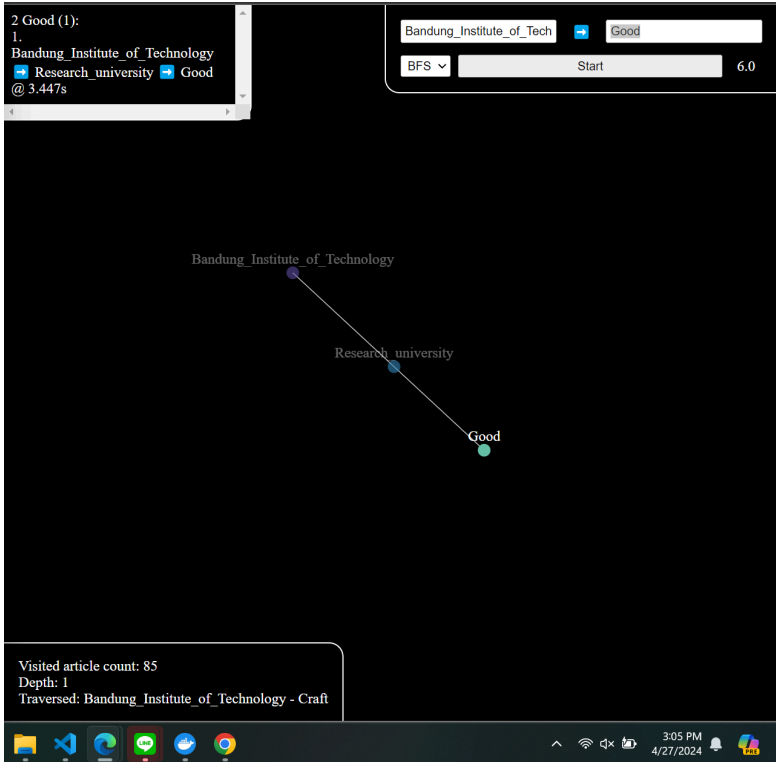
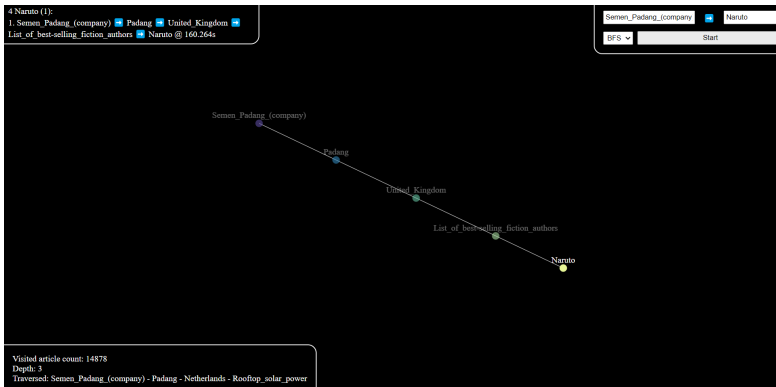
5. Bukan peramban (disarankan Google Chrome atau Firefox)
6. Masukkan tautan <http://localhost:3000/>.

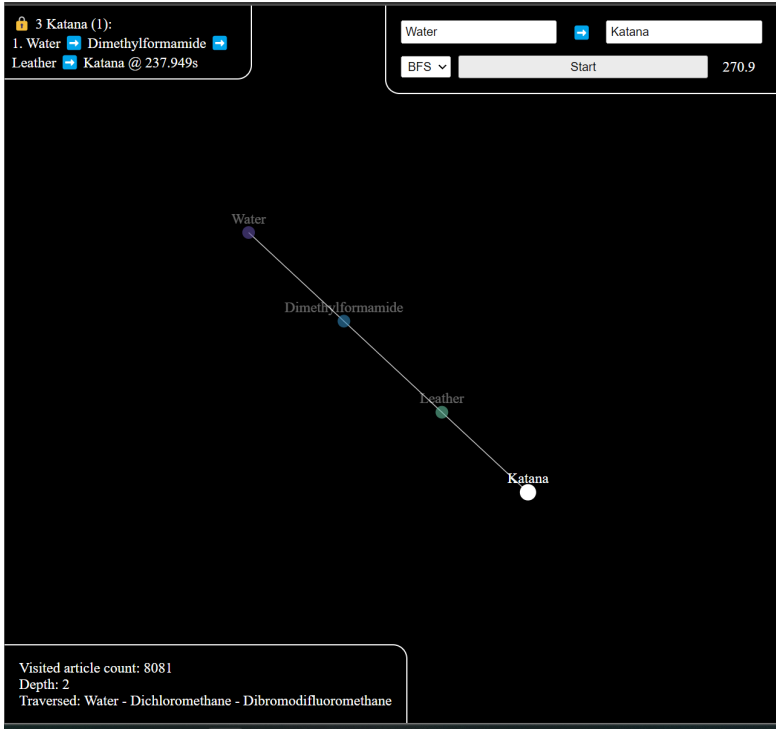
7. Masukkan laman awal dan laman akhir.
8. Pilih metode pencarian IDS atau BFS.
9. Tekan tombol *Search*.
10. Berhentikan *docker* dengan menekan Ctrl+C.

C. Hasil Pengujian

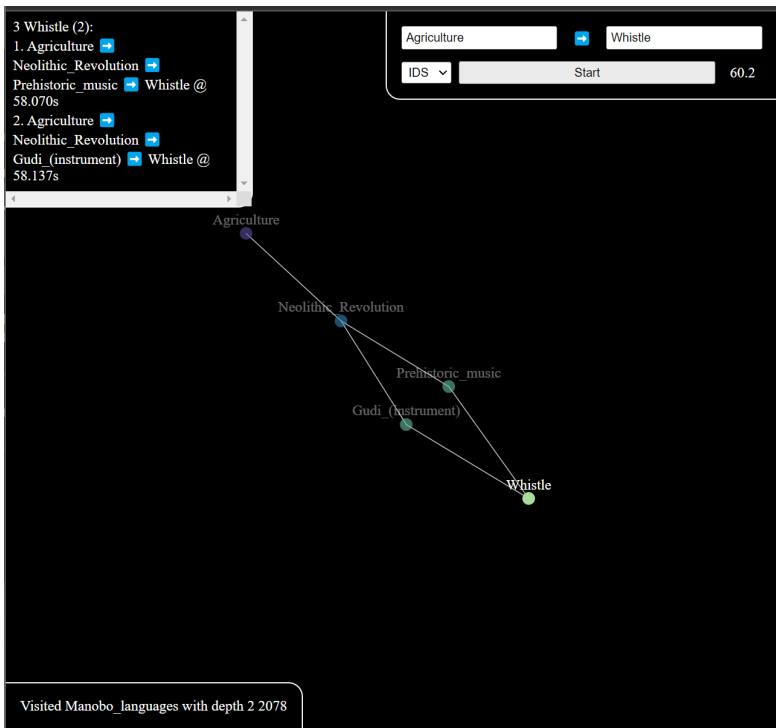
Tabel Pengujian BFS

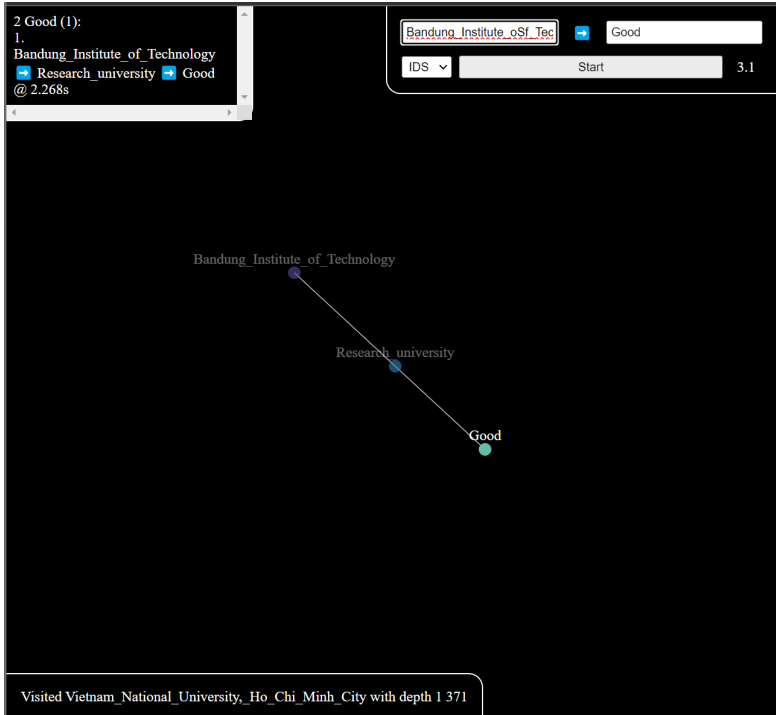

No	Input	Hasil
1	Agriculture ke Whistle	 <p>waktu: 35.549 s</p>

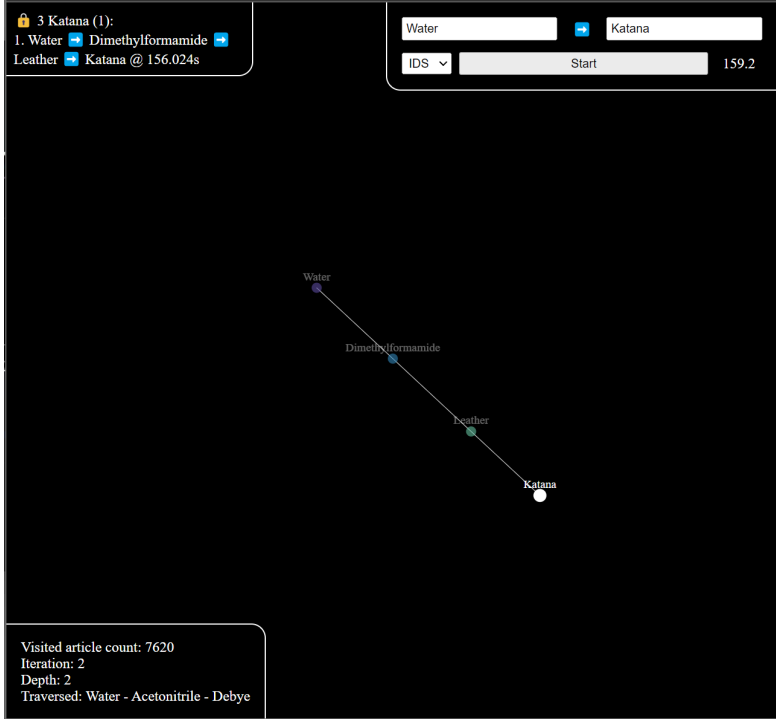
2	Bandung_Institute_of_Technology ke Good	 <p>2 Good (1): 1. Bandung_Institute_of_Technology Research_university -> Good @ 3.447s</p> <p>Bandung_Institute_of_Technology Research_university Good</p> <p>Visited article count: 85 Depth: 1 Traversed: Bandung_Institute_of_Technology - Craft</p> <p>waktu: 3.447s</p>
3	Semen_Padang (Company) ke Naruto	 <p>4 Naruto (1): 1. Semen_Padang (company) -> Padang -> United Kingdom -> List_of_best-selling_fiction_authors -> Naruto @ 160.264s</p> <p>Semen_Padang (company) Padang United Kingdom List_of_best-selling_fiction_authors Naruto</p> <p>Visited article count: 14878 Depth: 5 Traversed: Semen_Padang (company) - Padang - Netherlands - Rooftop_solar_power</p> <p>waktu: 160.264 s</p>

4	Water ke Katana	 <p>3 Katana (1): 1. Water → Dimethylformamide Leather → Katana @ 237.949s</p> <p>Water → Dimethylformamide → Leather → Katana</p> <p>Visited article count: 8081 Depth: 2 Traversed: Water - Dichloromethane - Dibromodifluoromethane</p> <p>waktu: 237.949 s</p>
---	-----------------	--

Tabel Pengujian IDS

No	Input	Hasil
1	Agriculture ke Whistle	 <p>3 Whistle (2): 1. Agriculture → Neolithic Revolution → Prehistoric music → Whistle @ 58.070s 2. Agriculture → Neolithic Revolution → Gudi (instrument) → Whistle @ 58.137s</p> <p>Agriculture → Neolithic Revolution → Prehistoric music → Gudi (instrument) → Whistle</p> <p>Visited Manobo_languages with depth 2 2078</p> <p>waktu: 58.070 s</p>

2	Bandung_Institute_of_Technology ke Good	 <p>2 Good (1): 1. Bandung_Institute_of_Technology @ 2.268s</p> <p>Bandung_Institute_of_Technology Research_university Good</p> <p>Visited Vietnam_National_University, Ho_Chi_Minh_City with depth 1 371</p> <p>waktu: 2.268 s</p>
3	Semen_Padang (Company) ke Naruto	 <p>4 Naruto (1): 1. Semen_Padang (company) @ 219.672s</p> <p>Semen_Padang (company) Padang United_Kingdom List_of_best-selling_fiction_authors Naruto</p> <p>Visited article count: 15462 Iteration: 3 Depth: 3 Traversed: Semen_Padang (company) - Padang - Japanese_occupation_of_the_Dutch_East_Indies - Demographics_of_East_Timor</p> <p>waktu: 219.672 s</p>

4	Water ke Katana	 <p>3 Katana (1): 1. Water → Dimethylformamide Leather → Katana @ 156.024s</p> <p>Water → Dimethylformamide → Leather → Katana</p> <p>Visited article count: 7620 Iteration: 2 Depth: 2 Traversed: Water - Acetonitrile - Debye</p> <p>waktu: 156.024 s</p>
---	-----------------	--

D. Analisis Pengujian

Dari hasil pengujian dapat dilihat bahwa kedua algoritma bisa menemukan rute terpendek dari suatu laman ke laman yang lain. Waktu proses kedua algoritma juga hampir sama yaitu 437.209 sekon (BFS) dan 436.034 sekon (IDS). Hal ini terjadi karena sebenarnya kedua algoritma ini memiliki kemiripan. Algoritma IDS sendiri adalah algoritma yang mengkombinasikan algoritma BFS dan DFS. Berdasarkan *time complexity* juga kedua algoritma memiliki kompleksitas yang sama yaitu $O(b^d)$.

BAB 5

KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Aplikasi *website* ini berhasil menyelesaikan masalah pencarian rute terpendek suatu laman Wikipedia dari laman wikipedia yang lainnya. Aplikasi juga bisa menjalankan pencarian baik dengan menggunakan algoritma BFS ataupun menggunakan algoritma IDS. Aplikasi pun berhasil melakukan pencarian rute dengan banyak solusi. Dari hasil pengujian aplikasi ini, ditemukan bahwa kedua algoritma memiliki waktu pencarian yang mirip. Hal ini terjadi karena algoritma BFS dan algoritma IDS memiliki kemiripan dalam langkah-langkahnya.

B. Saran

Hal yang bisa dilakukan untuk dapat meningkatkan hasil ataupun kinerja dari aplikasi ini:

1. Memiliki koneksi internet yang stabil dan cepat. Koneksi internet akan sangat mempengaruhi kinerja dan hasil. Internet yang lambat akan menghambat proses pencarian sehingga kinerja aplikasi tidak efektif.
2. Mempersiapkan waktu yang banyak. Proses pencarian bisa memakan waktu yang cukup lama. Hal ini bisa terjadi karena proses banyak pencarian yang dilakukan ataupun karena jaringan yang kurang baik. Untuk bisa mendapatkan hasil perlu disediakan waktu apalagi jika sedang melakukan pengujian.

LAMPIRAN

A. Tautan *Repository* GitHub

https://github.com/atpur-rafif/Tubes2_gogogo

DAFTAR PUSTAKA

<https://www.elpl.org/blogs/post/wiki-racing-a-fun-educational-game-for-all-ages/>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>

[https://www.sixdegreesofwikipedia.com/?source=Semen%20Padang%20\(company\)&target=Naruto](https://www.sixdegreesofwikipedia.com/?source=Semen%20Padang%20(company)&target=Naruto)