

Tugas Besar 2 IF 2123 Strategi Algoritma

Semester II Tahun 2023/2024

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Kelompok StressRace :

- 1. Francesco Michael Kusuma 13522038**
- 2. Marvel Pangondian 13522075**
- 3. Dimas Bagoes Hendrianto 13522112**

PROGRAM STUDI TEKNIK INFORMATIKA

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT
TEKNOLOGI BANDUNG 2024**

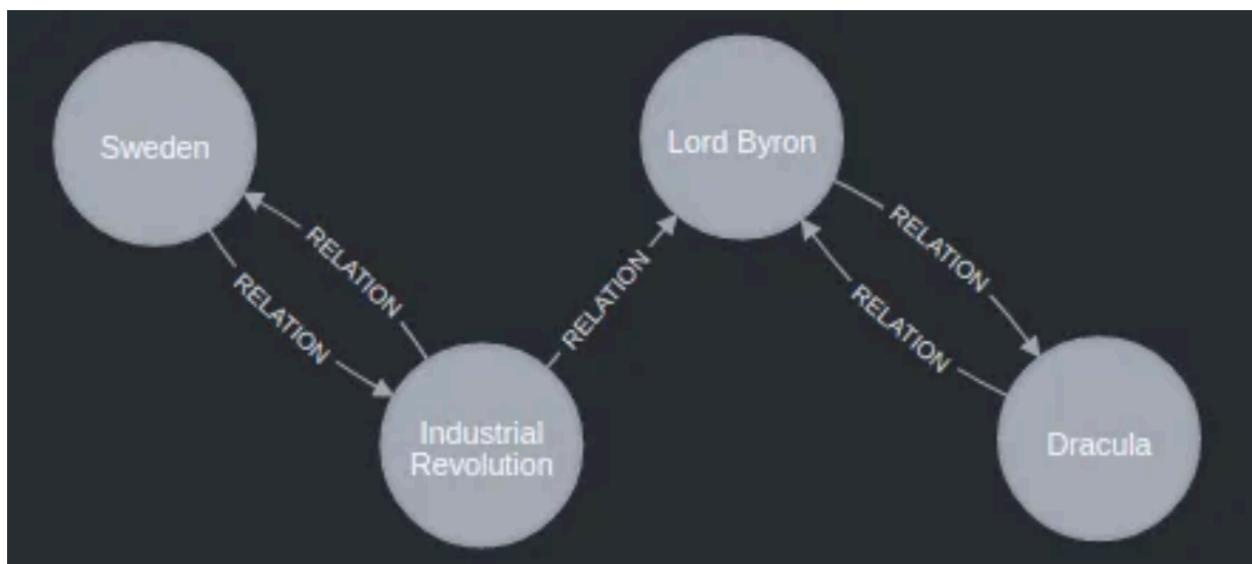
DAFTAR ISI

DAFTAR ISI	1
BAB 1	2
BAB 2	5
2.1 Graf	5
2.2 IDS Iterative Deepening Search	5
2.3 BFS Pencarian Melebar	6
2.4 Website	7
BAB 3	9
3.1 Pemecahan Masalah	9
3.2 Pemetaan Masalah Menjadi Elemen dalam Algoritma BFS dan IDS	10
3.3 Fitur	12
3.4 Ilustrasi Kasus	13
BAB 4	15
4.1 Spesifikasi Teknis	15
4.2 Penggunaan Aplikasi	31
4.3 Hasil Pengujian	33
4.3.1 Pencarian rute menggunakan algoritma BFS dengan single path	33
4.3.2 Pencarian rute menggunakan algoritma BFS dengan multiple path	37
4.3.3 Pencarian rute menggunakan algoritma IDS dengan single path	41
4.3.4 Pencarian rute menggunakan algoritma IDS dengan multiple path	45
4.4 Analisis	50
BAB 5	51
5.1 Kesimpulan	51
5.2 Saran	51
5.3 Refleksi	51
Daftar Pustaka	52
Lampiran	53

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 1.1 Ilustrasi Graf WikiRace

(Sumber:https://miro.medium.com/v2/resize:fit:1400/1*jxmEbVn2FFWybZsIicJCWQ.png)

Spesifikasi Tugas Besar 2

- Buatlah program dalam bahasa Go yang mengimplementasikan algoritma IDS dan BFS untuk menyelesaikan permainan WikiRace.
- Program menerima masukan berupa jenis algoritma, judul artikel awal, dan judul artikel tujuan.
- Program memberikan keluaran berupa jumlah artikel yang diperiksa, jumlah artikel yang dilalui, rute penjelajahan artikel (dari artikel awal hingga artikel tujuan), dan waktu pencarian (dalam ms).

- Program cukup mengeluarkan salah satu rute terpendek saja (cukup satu rute saja, tidak perlu seluruh rute kecuali mengerjakan bonus).
- Program berbasis web, sehingga perlu dibuat front-end dan back-end (tidak perlu di-deploy).
- Repository front-end dan back-end diizinkan untuk dipisah maupun digabung dalam repository yang sama.
- Program wajib dapat mencari rute terpendek kurang dari 5 menit untuk setiap permainan.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus. Akan tetapi, anggota kelompok tidak boleh sama dengan anggota kelompok pada tugas-tugas Strategi Algoritma sebelumnya.
- Program harus mengandung komentar yang jelas serta mudah dibaca.
- Mahasiswa dilarang menggunakan kode program yang didapatkan dari internet (alasan menggunakan kakas seperti GitHub Copilot tidak diterima). Mahasiswa harus membuat program sendiri, diperbolehkan untuk belajar dari program yang sudah ada.
- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.
- Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.
- Setiap kelompok harap mengisi nama kelompok dan anggotanya pada pranala bit.ly/kelompoktubes2stima24, paling lambat Kamis, 4 April pukul 22.11 WIB.
- Diwajibkan untuk memilih asisten meskipun tidak melakukan asistensi, karena asisten yang dipilih akan menjadi asisten saat asistensi (opsional) dan demo tugas besar. Pemilihan asisten dapat dilakukan pada link berikut, paling lambat Kamis, 4 April pukul 22.11 WIB.
- Program disimpan dalam repository yang bernama Tubes2_NamaKelompok (bila digabung) dan Tubes2_FE/BE_NamaKelompok (bila dipisah) dengan nama kelompok sesuai dengan yang di sheets diatas. Berikut merupakan struktur dari isi repository tersebut:
 - A. Folder src berisi program yang dapat dijalankan
 - B. Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf

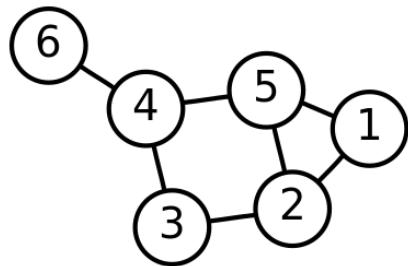
- C. README untuk tata cara penggunaan yang minimal berisi:
 - I. Penjelasan singkat algoritma IDS dan BFS yang diimplementasikan
 - II. Requirement program dan instalasi tertentu bila ada
 - III. Command atau langkah-langkah dalam meng-compile atau build program
 - IV. Author (identitas pembuat)
- Laporan dikumpulkan hari Sabtu, 27 April 2024 pada alamat Google Form berikut paling lambat pukul 23.59 :
<https://bit.ly/tubes2stima24>
- Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut:
<https://bit.ly/qnastima24>

BAB 2

Landasan Teori

2.1 Graf

Dalam matematika diskrit, khususnya teori graf, graf merupakan suatu struktur yang terdiri dari beberapa objek dan hubungan antar pasangan objek-objek tersebut. Secara sederhana, sebuah graf merupakan himpunan dari objek-objek yang dinamakan titik, simpul, atau sudut dihubungkan oleh penghubung yang dinamakan garis atau sisi. Dalam graf yang memenuhi syarat, di mana biasanya tidak berarah, sebuah garis dari titik A ke titik B dianggap sama dengan garis dari titik B ke titik A. Dalam graf berarah, garis tersebut memiliki arah. Pada dasarnya, sebuah graf digambarkan dengan bentuk diagram sebagai himpunan dari titik-titik (simpul) yang dihubungkan dengan sisi.



Gambar 2.1 Sebuah graf dengan 6 sudut dan 7 sisi

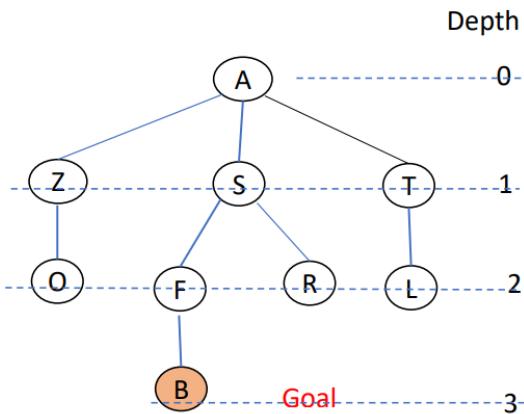
(Sumber:[https://id.wikipedia.org/wiki/Graf_\(matematika\)#/media/Berkas:6n-graf.svg](https://id.wikipedia.org/wiki/Graf_(matematika)#/media/Berkas:6n-graf.svg))

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut, sehingga secara sederhana graf didefinisikan sebagai kumpulan titik yang dihubungkan oleh garis-garis/sisi. Sedangkan definisi matematis untuk graf adalah, pasangan terurut himpunan (V, E), dimana V merupakan himpunan beranggotakan titik-titik (vertex) dan E merupakan himpunan beranggotakan sisi-sisi (edges).

2.2 IDS Iterative Deepening Search

Melakukan serangkaian DFS, dengan peningkatan nilai kedalaman-cutoff, sampai solusi ditemukan. Asumsi: simpul sebagian besar ada di level bawah, sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali. IDS memperoleh validitas yang

diinginkan dengan menerapkan batas kedalaman pada DFS, yang mengurangi kemungkinan terjebak dalam cabang yang tak terbatas atau sangat panjang. Ia melintasi setiap cabang node dari kiri ke kanan hingga mencapai kedalaman yang diperlukan. Setelah itu, IDS kembali ke node root dan menyelidiki cabang yang mirip dengan DFS.



Gambar 2.2 Pohon IDS

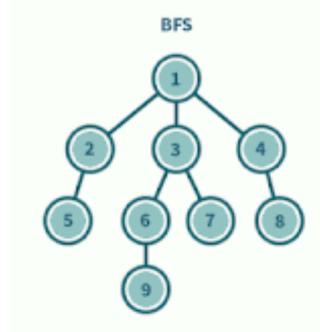
(Sumber:<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>)

2.3 BFS Pencarian Melebar

Algoritma Breadth First Search adalah algoritma pencarian melebar yang dilakukan dengan mengunjungi node pada level n terlebih dahulu sebelum mengunjungi node-node pada level n+1. Algoritma BFS berbeda dengan DFS.

Algoritma: Traversal dimulai dari simpul v

1. Kunjungi simpul v
2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul -simpul yang tadi dikunjungi, demikian seterusnya.



Gambar 2.3 Contoh BFS

(Sumber:<https://medium.com/@defytamara2610/bfs-breadth-first-search-pengertian-kekurangan-kelebihan-dan-contohnya-775a7d808fbc>)

2.4 Website

Web Development diambil dari dua kata utama yaitu web dan development. Dimana web merupakan aplikasi website, halaman, atau sumber online apapun itu yang memang bisa berfungsi dengan internet. Kemudian development merupakan membangun sesuatu dari awal. Jika diartikan secara lengkap adalah upaya dalam membangun, menciptakan dan melakukan maintaining dari aplikasi yang bekerja melalui internet seperti website. Website development jika diartikan secara detail maka akan terdapat beberapa tugas dibalik layar, seperti :

1. Server side scripting
2. Client side scripting
3. Konfigurasi dari keamanan server dan jaringan
4. Adanya Content Management System atau yang disingkat CMS

Web development juga mempunyai 3 jenis atau bagian yang saling mendukung. Ketiga jenis ini akan mempunyai jobdesk yang berbeda namun mempunyai satu tujuan yang sama yaitu menghasilkan website yang bisa berfungsi dengan baik di internet. Berikut inilah jenis-jenis dari website development antara lain :

1. Front-End Development

Front-end merupakan bagian depan website yang diakses oleh para pengunjung, jadi apa yang kamu lihat pada saat mengunjungi sebuah website, itulah yang menjadi bagian dan

tugas dari Front-end developer ini. Seorang Front-end developer memiliki tanggung jawab untuk mengurus desain website secara keseluruhan, mulai dari menu, tombol interaksi, maupun berbagai gambar lain guna meningkatkan pengalaman pengguna pada saat membuka suatu website. Biasanya, bagian developer yang satu ini akan menggunakan sejumlah bahasa pemrograman seperti HTML, CSS, hingga JavaScript. Karena bertugas untuk mengurusi sebuah website secara keseluruhan, tidak mengherankan jika front-end dituntut untuk lebih kreatif dan mempunyai skill desain yang cukup. Selain itu, bagaimana alur suatu website, interaksi, navigasi, hingga kenyamanan visual juga menjadi hal yang harus dicermati.

2. Back-End Development

Jika seorang front-end berurusan dengan tampilan depan suatu website, maka back-end berada di balik layar, yakni berurusan dengan hal-hal yang tidak terlihat oleh pengguna website. Misalnya pengurusan server, penerimaan data, hingga perbaikan fungsi sistem. Back-end biasa memakai sejumlah tools untuk berbagai keperluan pembuatan menjalankan coding-coding tertentu. Misalnya PHP, Python, atau Java untuk membangun aplikasi atau MySQL, Oracle hingga server SQL untuk mencari, menyimpan, atau mengubah berbagai data guna disajikan ke dalam coding front-end. Tools lainnya juga tidak kalah penting seperti framework, control software, hingga Linux untuk berbagai pengembangan.

3. Full Stack Development

Pada saat kamu mempelajari pengembangan website, tentu akan mempelajari kedua hal di atas, yakni sebagai front-end maupun back-end. Sebab keduanya masih saling berhubungan dan membutuhkan skill yang saling terkait. Oleh karenanya, dalam dunia web developer dikenal juga Full-stack developer yang harus bisa mengurusi front-end sampai back-end. Jadi, tidak hanya ahli mengembangkan bagian depan website saja, melainkan juga dengan hal-hal yang ada di dalam website tersebut. Kebanyakan Full-stack memang bekerja seperti back-end di bagian server. Namun, berbagai bahasa front-end seperti mengatur tampilan website juga perlu mereka kuasai.

BAB 3

Analisis Pemecahan Masalah

3.1 Pemecahan Masalah

Masalah yang ingin dipecahkan adalah mencari rute terpendek dari sebuah link wikipedia (*start link*) ke link wikipedia lain (*target link*) dengan melintasi pranala - pranala wikipedia dari satu laman (dimulai dari pranala *start link*) ke laman lain dan menampilkannya dalam bentuk web. Masalah tersebut dapat dipecahkan menggunakan algoritma Breadth-First-Search (BFS) atau algoritma Iterative-Deepening-Search (IDS).

Permasalahan tersebut dapat didekomposisi menjadi beberapa permasalahan sebagai berikut :

1. Memahami algoritma BFS dan IDS dan cara mengimplementasikannya dalam mencari rute terpendek antara dua laman wikipedia.
2. Memahami data struktur yang akan dipakai dalam implementasi algoritma BFS dan IDS
3. Memahami mekanisme *Web Scraping* yakni kegiatan mengambil tautan - tautan pada sebuah *web page*
4. Memahami bahasa pemrograman golang yang akan digunakan untuk mengimplementasi algoritma BFS dan IDS
5. Mengimplementasi algoritma BFS dan IDS yang sederhana yang akan digunakan sebagai *template/dasar* implementasi pencarian rute antar laman wikipedia
6. Mengembangkan algoritma BFS dan IDS untuk mencari semua rute terpendek antara dua node
7. Mengimplementasikan algoritma BFS dan IDS untuk mencari satu rute terpendek antara dua laman wikipedia
8. Mengembangkan algoritma BFS dan IDS untuk mencari semua rute terpendek antara dua laman wikipedia
9. Memahami mekanisme pengembangan sebuah laman web menggunakan Next.js yang merupakan sebuah framework dari React
10. Memahami mekanisme pembangunan sebuah laman web dengan docker

11. Mengembangkan visualisasi sebuah website menggunakan tailwind.css sebagai dasar *styling* laman web yang kami gunakan
12. Mengembangkan website menggunakan berbagai *library* yang tersedia dalam framework yang kami gunakan sebagai bagian dalam optimasi dan fitur yang kami kembangkan

Permasalahan tersebut dapat dipecah menjadi beberapa bagian :

1. Fungsi yang dapat mengambil link - link yang ada pada sebuah Wikipedia Page (*Web scraping*)
2. Fungsi yang dapat mencari sebuah rute terpendek antara dua laman wikipedia menggunakan algoritma Breadth-First-Search (BFS)
3. Fungsi yang dapat mencari semua rute terpendek antara dua laman wikipedia menggunakan algoritma Breadth-First-Search (BFS)
4. Fungsi yang dapat mencari sebuah rute terpendek antara dua laman wikipedia menggunakan algoritma Iterative-Deepening-Search (IDS).
5. Fungsi yang dapat mencari semua rute terpendek antara dua laman wikipedia menggunakan algoritma Iterative-Deepening-Search (IDS).

3.2 Pemetaan Masalah Menjadi Elemen dalam Algoritma BFS dan IDS

Untuk menyelesaikan masalah ini, penulis menganggap setiap link wikipedia sebagai sebuah node, dan link - link dalam wikipedia tersebut sebagai child yang akan diproses. Dalam implementasi BFS, penulis menggunakan prinsip *First in First Out* dan data struktur queue. Dalam implementasi IDS, penulis menggunakan prinsip *Depth Limited Search* yakni prinsip membatasi kedalaman DFS pada sebuah kedalaman i , lalu iterasi batas kedalaman tersebut sampai mendapatkan solusi terakhir.

Node	Link wikipedia
Path	Urutan tautan wikipedia untuk mencapai
Goal state	Sebuah atau semua <i>Path</i> untuk mencapai target wikipedia page

Tabel 3.2.1 Elemen pada algoritma BFS dan IDS

Berikut adalah proses pada implementasi algoritma Breadth-First-Search (BFS) :

1. Pertama akan divalidasi terlebih dahulu laman awal (*start link*) dan laman akhir (*target link*), validasi berupa apakah kedua laman merupakan laman wikipedia dan dapat diakses.
2. Setelah proses validasi, akan diinisialisasi sebuah struktur data queue dengan *start link* sebagai elemen pertama, sebuah struktur data *map* bernama *parent* untuk menyimpan parent sebuah link, dan struktur data map bernama *visited* untuk menyimpan *link - link* yang sudah dikunjungi sebelumnya.
3. Dengan prinsip *First in First Out*, dilakukan dequeue (pelepasan elemen pertama queue), elemen tersebut kemudian akan diproses.
4. Pemrosesan elemen dimulai dengan memvalidasi apakah elemen tersebut merupakan merupakan *target link*, jika iya, maka akan segera membentuk rute. Rute akan dibentuk menggunakan *map parent*, jika tidak maka elemen (*link*) akan diproses lebih lanjut.
5. Pemrosesan selanjutnya adalah *web scraping*, *link - link* yang terdapat pada web elemen tersebut akan *disrape* dan dimasukkan (enqueue). Tetapi, sebelum dimasukkan, akan diperiksa masing - masing *link* tersebut yakni apakah sudah dikunjungi sebelumnya dengan *map visited* , jika belum, maka *link* tersebut akan dimasukkan ke dalam queue dan *map parent* akan diperbarui.
6. Proses tersebut dilakukan sampai queue kosong atau sampai ditemukan sebuah rute.
7. Algoritma BFS untuk mencari semua rute terpendek memiliki proses yang sama dengan pencarian satu rute, hanya saja algoritma tetap berjalan sampai mendapatkan semua rute terpendek pada *depth* terpendek.

Berikut adalah proses pada implementasi algoritma Breadth-First-Search (BFS) :

1. Pertama akan divalidasi terlebih dahulu laman awal (*start link*) dan laman akhir (*target link*), validasi berupa apakah kedua laman merupakan laman wikipedia dan dapat diakses.
2. Setelah proses validasi, akan diinisialisasi struktur data *slice* bernama *resultPath* yang akan menyimpan hasil rute terpendek,
3. Pertama akan dimulai sebuah iterasi *depth*, dimulai dari *depth* 0 yang akan digunakan sebagai batasan kedalaman untuk algoritma *Depth Limited Search*. Iterasi ini akan dilakukan sampai mendapatkan rute terpendek.

4. Algoritma *Depth Limited Search* dimulai dengan mengecek apakah *link* yang diperiksa saat ini (*current link*) merupakan *link target*, jika iya maka akan langsung dibentuk *path/rute*, jika tidak maka dilakukan pemrosesan selanjutnya.
5. Pemrosesan selanjutnya adalah memeriksa apakah *depth current link* bernilai 0, jika iya maka rekursi berhenti, jika tidak maka *link - link* yang terdapat pada web *current link* akan *disrape*.
6. Selanjutnya akan diiterasikan setiap *urlNow* pada *link - link* yang berhasil *disrape*, setiap iterasi akan secara rekursi melaksanakan algoritma *Depth Limited Search* dengan *depth* kurang satu dari *depth* saat ini

3.3 Fitur

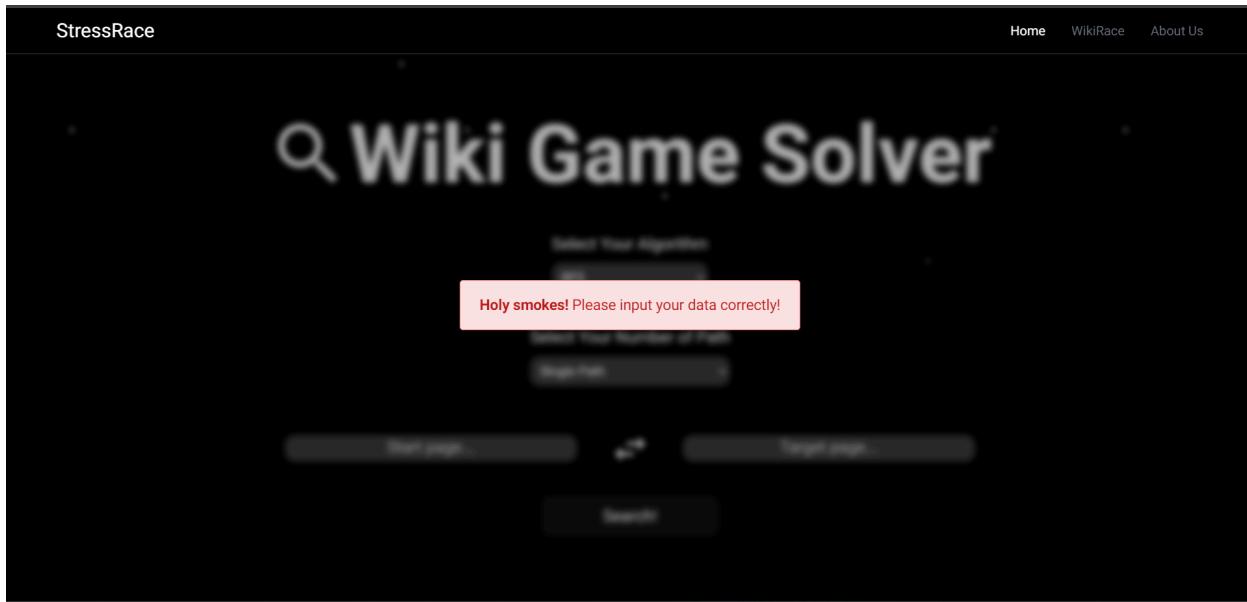
Fitur Utama :

1. Mencari sebuah jalur terpendek antara website Wikipedia menggunakan algoritma baik IDS maupun BFS
2. Mencari semua kemungkinan jalur terpendek antara website Wikipedia menggunakan algoritma baik IDS maupun BFS
3. Docker yang digunakan untuk mempermudah pengembangan dan pengujian, serta mengemas website dan algoritma menjadi satu kontainer.

Teknologi yang kami gunakan :

1. Go (Gin)
2. Docker
3. React JS
4. Next JS
5. Tailwind CSS

3.4 Ilustrasi Kasus

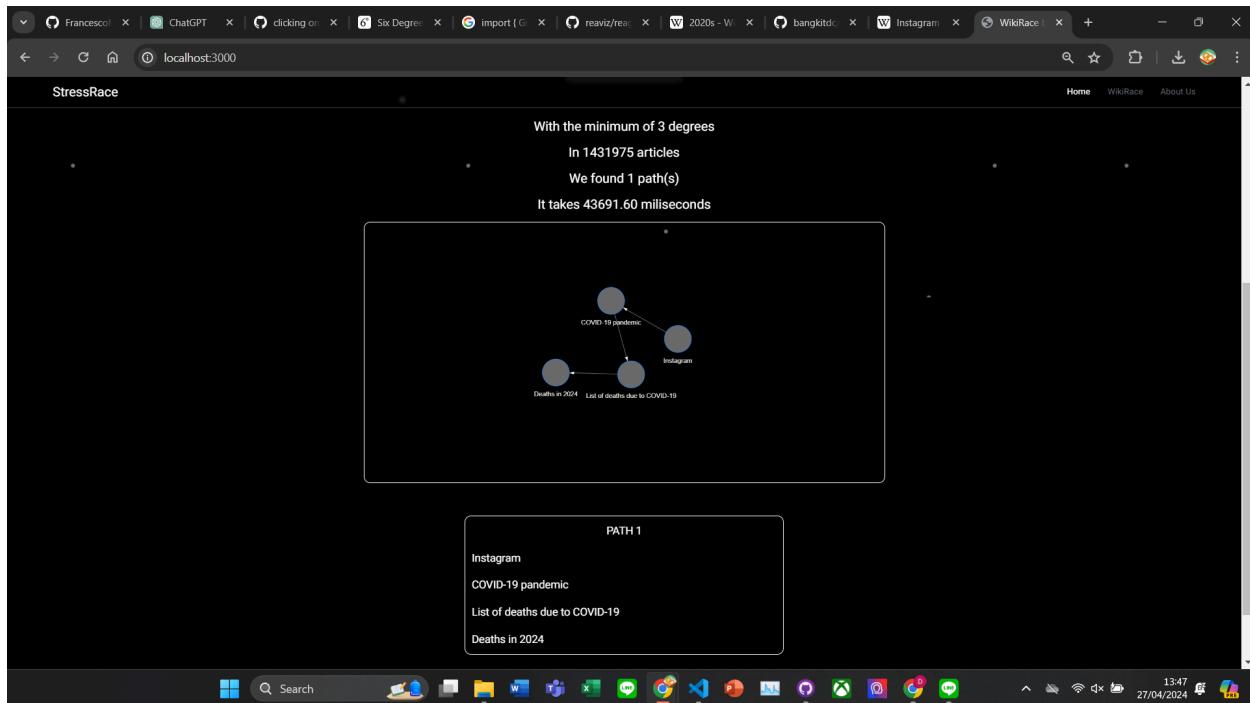


Gambar 3.4.1 Contoh Tampilan Ketika Tidak Ada Input



Gambar 3.4.2 Contoh Tampilan Ketika Sedang Memproses

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



Gambar 3.4.3 Contoh Tampilan Hasil

BAB 4

Implementasi dan Pengujian

4.1 Spesifikasi Teknis

BFS.go

```
package algorithm

import (
    "fmt"
    "scraper "server/pkg/scrapers"
    "strings"
    "sync"
    "sync/atomic"
)

// Basic node structure that BfsGoRoutine and BfsGoRoutineAllPaths
type node struct {
    url    string
    depth int32
}

// basic function to append node
func appendNode(list []node, newNode node) []node {
    return append(list, newNode)
}

// this is the most simple version of BFS, all BFS versions will be based
on this
func Bfs(start string, end string) ([]string, error) {
    start = strings.TrimSpace(start)
    end = strings.TrimSpace(end)

    if !scraper.IsWikiPageUrlExists(&start) { // Check if start url exists
        return nil, fmt.Errorf("start page does not exist")
    }
```

```
if !scraper.IsWikiPageUrlExists(&end) { // Check if end/targer url exists
    return nil, fmt.Errorf("end page does not exist")
}

queue := []string{start} // FIFO for BFS
visited := make(map[string]bool) // Visited map to avoid cycles
visited[start] = true // mark the first as true
parent := make(map[string]string) // Make parent map to keep track of parent nodes

var mutex1Queue sync.Mutex // mutex1 used when making path, technically it is not used in this BFS func but need in the makePath function

for len(queue) > 0 { // iterate until queue is empty or path is found
    curr := queue[0] // curr node , using FIFO
    queue = queue[1:] // dequeue

    if curr == end { // return path if curr url is target
        return makePath(parent, start, end, &mutex1Queue), nil
    }

    var allUrl = scraper.GetScrapeLinks(curr) // find all child nodes
    if allUrl == nil {
        continue // skip if there isn't any child
    }

    for _, linkTemp := range allUrl { // iterate over every child
        if !visited[linkTemp] {
            visited[linkTemp] = true
            parent[linkTemp] = curr
            queue = append(queue, linkTemp) // enqueue
            if linkTemp == end { // check if node is target, avoid unnecessary process later on
                return makePath(parent, start, end, &mutex1Queue), nil
            }
        }
    }
}
```

```
        return nil, nil // Return nil if end is not reachable
    }

// BfsGoRoutine is the same as Bfs, but it uses go routine
func BfsGoRoutine(start string, end string) ([]string, map[string]bool,
error) {
    start = strings.TrimSpace(start)
    end = strings.TrimSpace(end)

    if !scraper.IsWikiPageUrlExists(&start) {
        return nil, nil, fmt.Errorf("start page does not exist")
    }
    if !scraper.IsWikiPageUrlExists(&end) {
        return nil, nil, fmt.Errorf("end page does not exist")
    }
    limiter := make(chan struct{}, 100) // set limiter of go routines
    queue := []string{start}
    visited := make(map[string]bool)
    visited[start] = true
    parent := make(map[string]string)
    var found int32 = 0 // Use atomic int32 for the found flag
    var mutex1Queue sync.Mutex

    for atomic.LoadInt32(&found) == 0 { // iterate until found a solution
        limiter <- struct{}{} // insert limiter

        go func() {
            defer func() { <-limiter }() // release limiter when process
is finished
            var curr string

            mutex1Queue.Lock()
            if len(queue) > 0 {
                curr = queue[0]
                queue = queue[1:]
            }
            mutex1Queue.Unlock()
        }()
    }
}
```

```
        if curr == end { // Set 'found' to 1 if a solution is
discovered.

            atomic.StoreInt32(&found, 1)
            return
        }

        var allUrl = scraper.GetScrapeLinksConcurrent(curr)
        if allUrl == nil {
            return
        }

        mutex1Queue.Lock()
        for _, linkTemp := range allUrl {

            if !visited[linkTemp] {
                visited[linkTemp] = true
                parent[linkTemp] = curr
                queue = append(queue, linkTemp)
                if linkTemp == end {
                    atomic.StoreInt32(&found, 1) // Set 'found' to 1
if a solution is discovered.
                    break
                }
            }
        }
        mutex1Queue.Unlock()

    }()
}

if atomic.LoadInt32(&found) == 0 {
    return nil, visited, fmt.Errorf("path cannot be found") // Return
nil if end is not reachable
}

return makePath(parent, start, end, &mutex1Queue), visited, nil
}
```

```
// this is the same as BFS but uses node to keep track of current depth
// it will search for all shortest path in the shortest depth possible
func BfsMultPath(start string, end string) ([] []string, map[string]bool,
error) {
    start = strings.TrimSpace(start)
    end = strings.TrimSpace(end)

    if !scraper.IsWikiPageUrlExists(&start) {
        return nil, nil, fmt.Errorf("start page does not exist")
    }
    if !scraper.IsWikiPageUrlExists(&end) {
        return nil, nil, fmt.Errorf("end page does not exist")
    }
    startNode := node{ // initialize startNode
        url:    start,
        depth: 0,
    }
    var queue []node
    queue = appendNode(queue, startNode)
    visited := make(map[string]bool)
    visited[start] = true
    parent := make(map[string][]string)
    // parents := make(map[string][]string)
    var maxDepth int32 = 999 // records the depth of the shortest solution

    for len(queue) > 0 {
        curr := queue[0]
        queue = queue[1:]

        if curr.depth > maxDepth { // stops when the current depth is
bigger then the shortest depth solution
            break
        }

        if curr.url == end && curr.depth <= maxDepth {
            maxDepth = curr.depth
            if curr.depth == 0 || curr.depth == 1 {
```

```
        break
    }

}

var allUrl = scraper.GetScrapeLinks(curr.url)
if allUrl == nil {
    continue // continue when current node is a leaf / no children
}
currDepth := curr.depth
for _, linkTemp := range allUrl {
    if !visited[linkTemp] {
        visited[linkTemp] = true
        parent[linkTemp] = append(parent[linkTemp], curr.url)
        newNode := node{ // initialize node with depth + 1 of
parent node
            url: linkTemp,
            depth: currDepth + 1,
        }
        queue = appendNode(queue, newNode) // enqueue
    }
}
}

return makePathAll(parent, start, end), visited, nil
}

func BfsAllPathGoRoutine(start string, end string) ([][][]string,
map[string]bool, error) {
    start = strings.TrimSpace(start)
    end = strings.TrimSpace(end)

    if !scraper.IsWikiPageUrlExists(&start) {
        return nil, nil, fmt.Errorf("start page does not exist")
    }
    if !scraper.IsWikiPageUrlExists(&end) {
        return nil, nil, fmt.Errorf("end page does not exist")
    }
}
```

```
}

limiter := make(chan struct{}, 200)
startNode := node{
    url:    start,
    depth: 0,
}

visited := make(map[string]bool)
visited[start] = true
parent := make(map[string][]string)
var found int32 = 0 // Use atomic int32 for the found flag
var mutex1 sync.Mutex
var wg sync.WaitGroup
var maxDepth int32 = 10
queue := []node{startNode}

for atomic.LoadInt32(&found) == 0 {
    newQueue := []node{}

    for _, value := range queue {
        limiter <- struct{}{}
        wg.Add(1) // using wait group to make sure all go routines has
finished

        go func(curr node) {
            defer func() { <-limiter }()
            defer wg.Done()

            if curr.url == end && curr.depth <=
atomic.LoadInt32(&maxDepth) { // when current url is the target, update
maxDepth
                atomic.SwapInt32(&maxDepth, curr.depth)
                return
            }
        }
    }
}
```

```
        if curr.depth > atomic.LoadInt32(&maxDepth) { // when
current depth is more than the maxDepth, make found = 1 as a flag to not
process nodes anymore
            atomic.StoreInt32(&found, 1)
            return
        }

var allUrl = scraper.GetScrapeLinksConcurrent(curr.url)
if allUrl == nil {
    return
}

mutex1.Lock()
for _, linkTemp := range allUrl {

    if !visited[linkTemp] {
        visited[linkTemp] = true
        parent[linkTemp] = append(parent[linkTemp],
curr.url) // update parent map
        if linkTemp == end && curr.depth <=
atomic.LoadInt32(&maxDepth) {
            atomic.SwapInt32(&maxDepth, curr.depth+1)
            atomic.StoreInt32(&found, 1)
            visited[linkTemp] = false
            break
        } else {
            newQueue = append(newQueue, node{ // enqueue
node, node has depth + 1 of parent
                url: linkTemp,
                depth: curr.depth + 1,
            })
        }
    }
}
mutex1.Unlock()
```

```
    } (value)

}

wg.Wait()
queue = newQueue

}

if atomic.LoadInt32(&found) == 0 {
    return nil, visited, fmt.Errorf("path cannot be found") // Return
nil if end is not reachable
}

return makePathAll(parent, start, end), visited, nil
}

// func makePath receives parent map, start url, and end url to create a
// path from end to start based on parent map
func makePath(parent map[string]string, start string, end string,
mutexQueue *sync.Mutex) []string {
    path := []string{}
    curr := end
    for curr != start { // iterate until curr is start url
        path = append([]string{curr}, path...)
        mutexQueue.Lock()
        curr = parent[curr]
        mutexQueue.Unlock()
    }
    path = append([]string{start}, path...)
    return path
}

// func makePathAll receives parent map, start url, and end url to create all
// paths from start url to end url
func makePathAll(parent map[string][]string, start string, end string)
[][]string {
    if start == end { // base case, start is end
```

```
        return [] [] string{{start}}
```

```
}
```

```
var paths [] [] string // initialize paths
```

```
if _, exists := parent[end]; !exists { // check if end has a parent
```

```
    return nil
```

```
}
```

```
for _, parentUrl := range parent[end] {
```

```
    parentPaths := makePathAll(parent, start, parentUrl) //recursively
```

```
call itself to find all paths from start url to parent url
```

```
    for _, path := range parentPaths {
```

```
        paths = append(paths, append(path, end)) // append every
```

```
parent paths with end url and insert it as new paths
```

```
    }
```

```
}
```

```
return paths
```

```
}
```

```
IDS.go
```

```
package algorithm
```

```
import (
```

```
    "fmt"
```

```
    "scraper" "server/pkg/scraper"
```

```
    "strings"
```

```
    "sync"
```

```
)
```

```
var LockPaths sync.Mutex
```

```
// go routine version of IDS will be based on this Ids
```

```
// func Ids receives startPage and endPage URL, and returns the list of all paths
```

```
using IDS algorithm
```

```
func Ids(startPage string, endPage string, maxDepth int) ([] [] string, int) {
```

```
startPage = strings.TrimSpace(startPage)
endPage = strings.TrimSpace(endPage)
var paths [][]string          // store possible paths from start to goal page
var shortestDepth int = -1 // depth of shortest path

for depth := 0; depth <= maxDepth && shortestDepth == -1; depth++ {
    var currPathsDepth [][]string                      // store possible
paths temporary
    var visited map[string]bool = make(map[string]bool) // store nodes/links
that has been visited to avoid cycles
    dls(startPage, endPage, depth, visited, nil, &currPathsDepth)
    // if possible path has been found
    if len(currPathsDepth) > 0 {
        shortestDepth = depth
        paths = append(paths, currPathsDepth...)
    }
}

return paths, shortestDepth
}

// func dls will perform dfs with a depth constraint
// dls will perform dfs recursively
func dls(currUrl string, endPage string, currDepth int, visited map[string]bool,
currPath []string, currPathDepth *[][][]string) {
    if currDepth == 0 && currUrl == endPage { // if the currUrl is the target
Url, then update currentPathDepth
        path := make([]string, len(currPath)+1)
        copy(path, append(currPath, currUrl))
        *currPathDepth = append(*currPathDepth, path)
        return
    }

    if currDepth <= 0 { // if current depth is at the bottom of dfs, then stop
recursion
        return
    }
}
```

```
var allUrl = scraper.GetScrapeLinks(currUrl) // getting all possible child of currUrl
if allUrl == nil { // if no more child (url is a leaf) then return
    return
}
visited[currUrl] = true // mark current node/url as already visited
currPath = append(currPath, currUrl) // update current path

for _, url := range allUrl { // iterate over every child
    if !visited[url] {
        dls(url, endPage, currDepth-1, visited, currPath, currPathDepth) // process left most child first
    }
}
visited[currUrl] = false // backtrack

}

// func IdsFirstPath is IDS with go routine implementation and only gives one solution
// in func IdsFirstPath, there is no visited map to keep track of all nodes that has been visited
// because DLS basically DFS but with a maxDepth, therefore Infinite Loops in Cycles is impossible
// reference : https://youtu.be/Y85ECK_H3h4?si=dm3PtHyv16rDHC38
// The reason that IdsFirstPath does not have visited map is to reduce the amount of mutex used when using go routine
// IdsFirstPath is the same as IDS, but it only returns one path and it also returns the amount of articles that has been processed

var Found int32

func IdsFirstPath(startUrl string, endUrl string, maxDepth int) ([]string, int, error) {
    Found = 0
    startUrl = strings.TrimSpace(startUrl)
```

```
endUrl = strings.TrimSpace(endUrl)
var resultPath []string
if !scraper.IsWikiPageUrlExists(&startUrl) {
    return nil, 0, fmt.Errorf("start page doesn't exist")
}
if !scraper.IsWikiPageUrlExists(&endUrl) {
    return nil, 0, fmt.Errorf("end page doesn't exist")
}

for depth := 0; depth <= maxDepth; depth++ {
    dlsFirstPath(startUrl, endUrl, depth, &resultPath, []string{})
    if len(resultPath) > 0 {
        return resultPath, len(scraper.Unique), nil
    }
}
return nil, len(scraper.Unique), fmt.Errorf("path cannot be found at depth
10")
}

// dlsFirstPath functions similarly to dls, but it utilizes goroutines and
terminates as soon as a path is found
func dlsFirstPath(currUrl string, endUrl string, depth int, resultPaths
*[]string, currPath []string) {
    // fmt.Println(currUrl)
    // fmt.Println(currUrl)
    LockPaths.Lock()
    newPath := append(currPath, currUrl)

    if len(*resultPaths) > 0 {
        LockPaths.Unlock()
        return
    } else if currUrl == endUrl {
        *resultPaths = append(*resultPaths, newPath...)
        LockPaths.Unlock()
        return
    }
    LockPaths.Unlock()

    if depth <= 0 {
```

```
        return
    }

    allUrlNow := scraper.GetScrapeLinksConcurrent(currUrl)
    if allUrlNow == nil {
        return
    }

    var relevantLen int
    if len(allUrlNow) < 19 {
        relevantLen = len(allUrlNow) - 3
        if relevantLen < 0 {
            relevantLen = 1
        }
    } else {
        relevantLen = 20
    }
    limiter := make(chan struct{}, relevantLen)
    for _, urlNow := range allUrlNow {
        limiter <- struct{}{}
        go func(nextUrl string) {
            dlsFirstPath(nextUrl, endUrl, depth-1, resultPaths, newPath)
            <-limiter
        }(urlNow)
    }
}

// Func IdsAllPath is the same as Ids, but it uses go routines
// similar to IdsFirstPath, IdsAlPathh doesn't have visited map to reduce mutex
func IdsAllPath(startUrl string, endUrl string, maxDepth int) ([][][]string, int, error) {
    startUrl = strings.TrimSpace(startUrl)
    endUrl = strings.TrimSpace(endUrl)
    var allPaths [][]string
    if !scraper.IsWikiPageUrlExists(&startUrl) {
        return nil, 0, fmt.Errorf("start page doesn't exist")
    }
    if !scraper.IsWikiPageUrlExists(&endUrl) {
        return nil, 0, fmt.Errorf("end page doesn't exist")
```

```
}

for depth := 0; depth <= maxDepth; depth++ {
    dlsAllPath(startUrl, endUrl, depth, &allPaths, []string{})
    if len(allPaths) > 0 {
        return allPaths, len(scraping.Unique), nil
    }
}
return nil, len(scraping.Unique), nil
}

// Func dlsAllPath is the same as dls, but it uses go routines
// similar to IdsFirstPath, IdsAlPathh doesn't have visited map to reduce mutex
func dlsAllPath(currUrl string, endUrl string, depth int, paths *[][]string,
currPath []string) {

    if currUrl == endUrl {
        LockPaths.Lock()
        newPath := append(currPath, currUrl)
        *paths = append(*paths, newPath)
        LockPaths.Unlock()
        return
    }

    if depth <= 0 {
        return
    }
    newPath := append(currPath, currUrl)
    allUrlNow := scraping.GetScrapeLinksConcurrent(currUrl)
    if allUrlNow == nil {
        return
    }
    var relevantLen int
    if len(allUrlNow) < 15 {
        relevantLen = len(allUrlNow) - 3
        if relevantLen < 0 {
            relevantLen = 1
        }
    } else {
```

```
relevantLen = 15
}

limiter := make(chan struct{}, relevantLen)

for _, urlNow := range allUrlNow {
    limiter <- struct{}(){}
    go func(nextUrl string) {
        dlsAllPath(nextUrl, endUrl, depth-1, paths, newPath)
        <-limiter
    }(urlNow)
}
}
```

4.2 Penggunaan Aplikasi



Gambar 4.2.1 Page Home

Garis besar penggunaan website kami adalah sebagai berikut :

1. Pilih terlebih dahulu jenis algoritma yang anda ingin gunakan



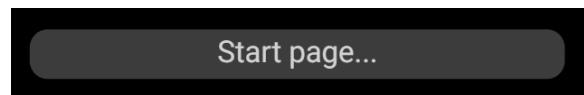
Gambar 4.2.2 Tombol Pilihan Algoritma

2. Tentukan jumlah path yang anda ingin hasilkan pada box seperti dibawah ini



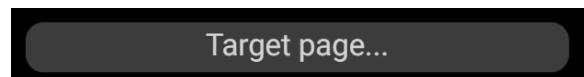
Gambar 4.2.3 Tombol Pilihan Jumlah Hasil

3. Masukkan judul laman awal pencarian pada box seperti dibawah ini



Gambar 4.2.3 Input Start Page

4. Masukkan judul laman akhir pencarian pada box seperti dibawah ini



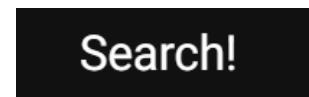
Gambar 4.2.4 Input Target Page

5. Anda dapat menukar judul laman awal dengan judul laman akhir dengan tombol dibawah

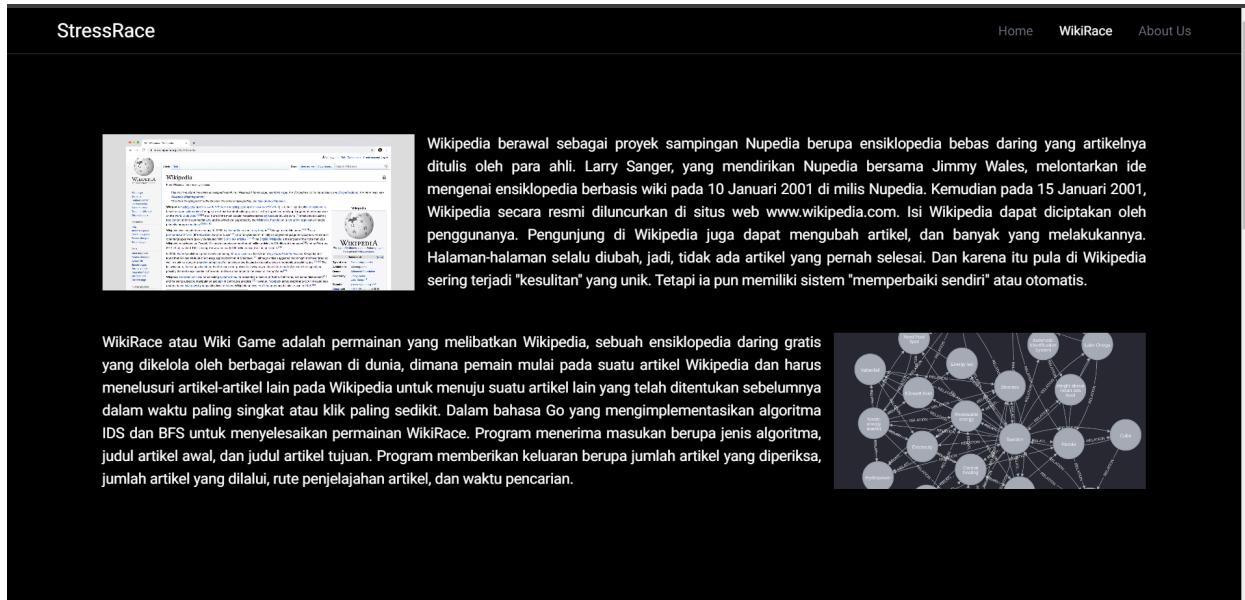


Gambar 4.2.5 Tombol Swap Input

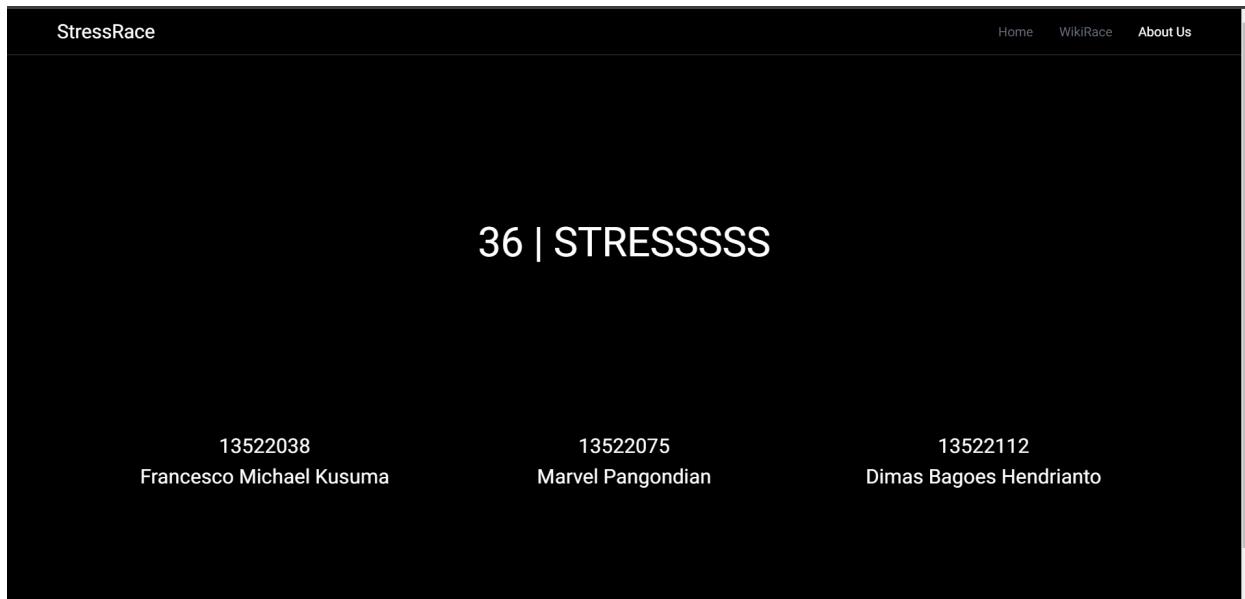
6. Mulai pencarian dengan menekan tombol Search! seperti dibawah



Gambar 4.2.6 Tombol Mulai Pencarian



Gambar 4.2.7 Page Wikirace



Gambar 4.2.7 Page About Us

4.3 Hasil Pengujian

4.3.1 Pencarian rute menggunakan algoritma BFS dengan single path

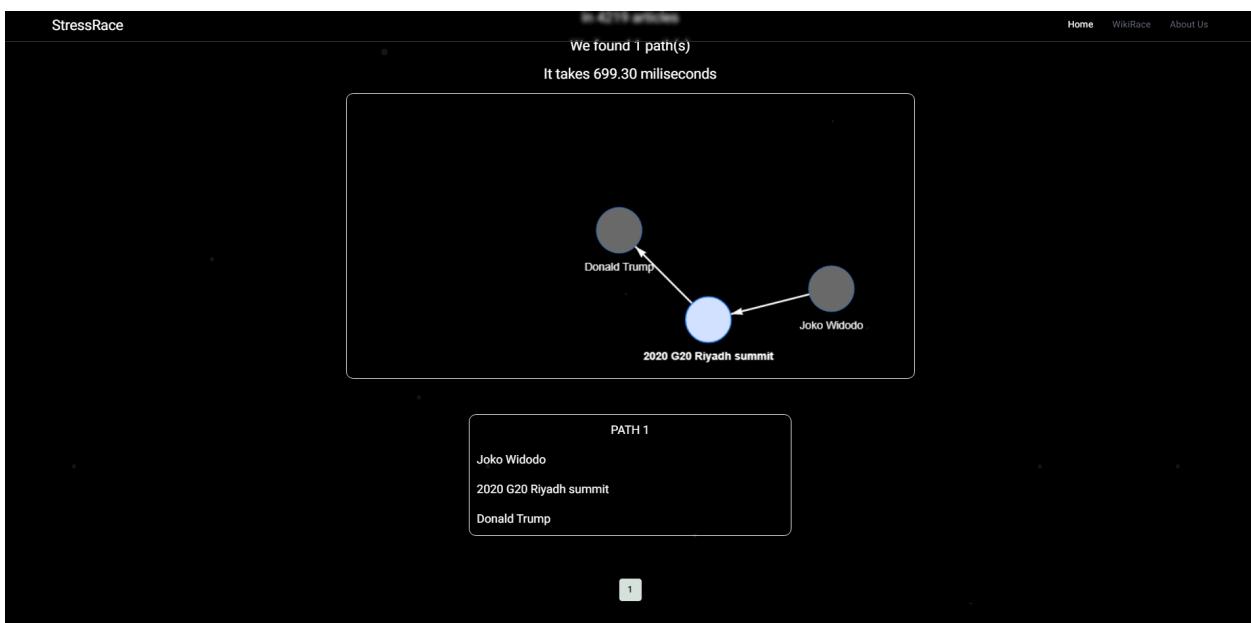
- Pencarian : Joko Widodo → Donald Trump

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Kedalaman : 2

Jumlah artikel yang ditelusuri : 4219 artikel

Durasi : 699,30 ms



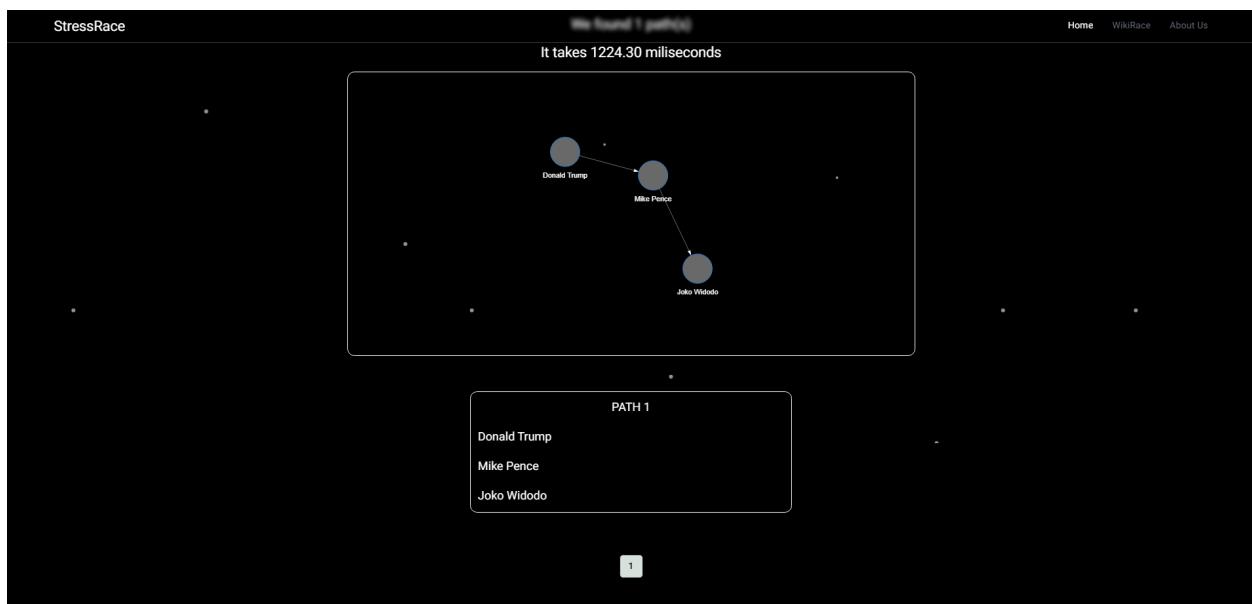
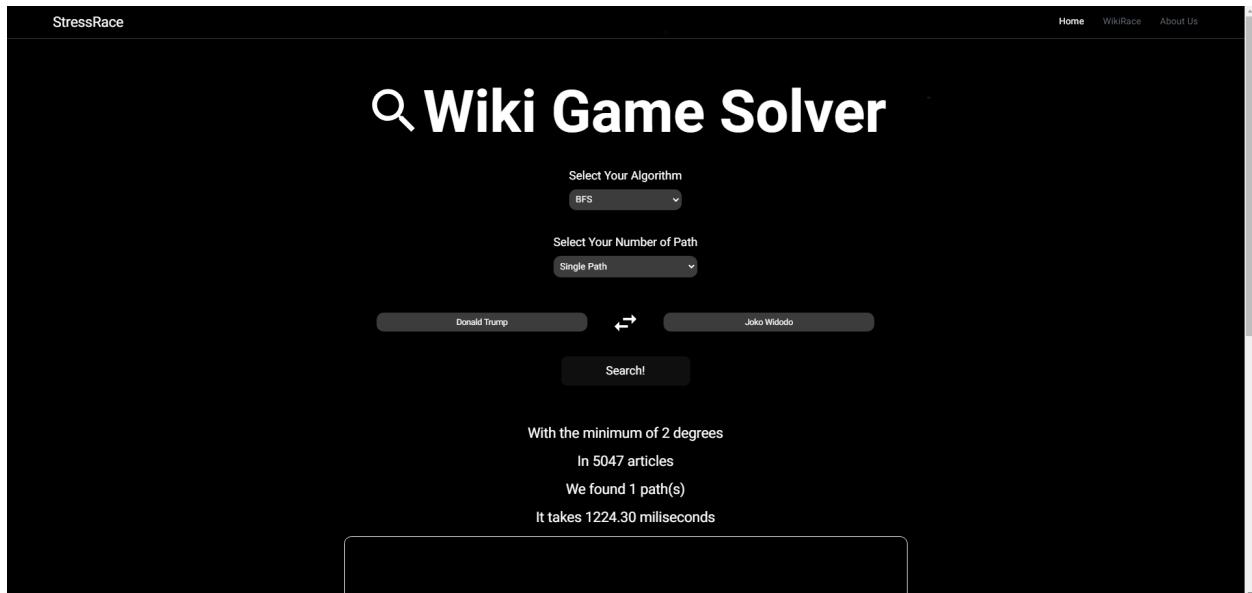
- b. Pencarian : Donald Trump → Joko Widodo

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Kedalaman : 2

Jumlah artikel yang ditelusuri : 5047 artikel

Durasi : 1224,5 ms



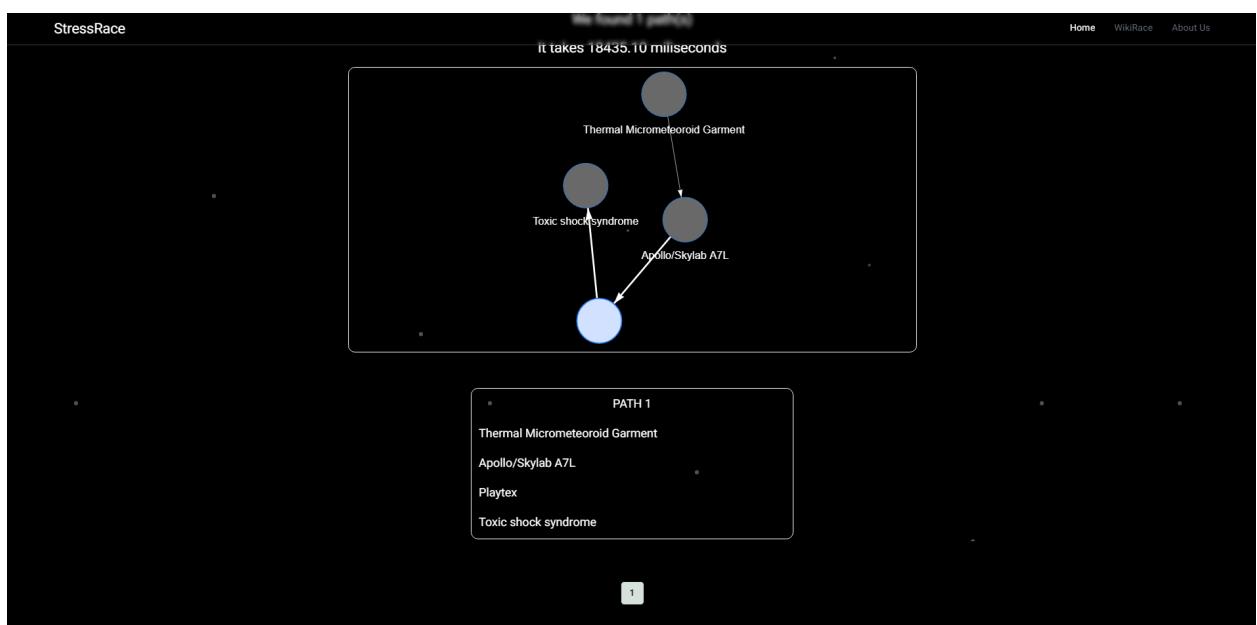
- c. Pencarian : Thermal Micrometeoroid Garment → Toxic shock syndrome

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Kedalaman : 3

Jumlah artikel yang ditelusuri : 215022 artikel

Durasi : 18435,10 ms

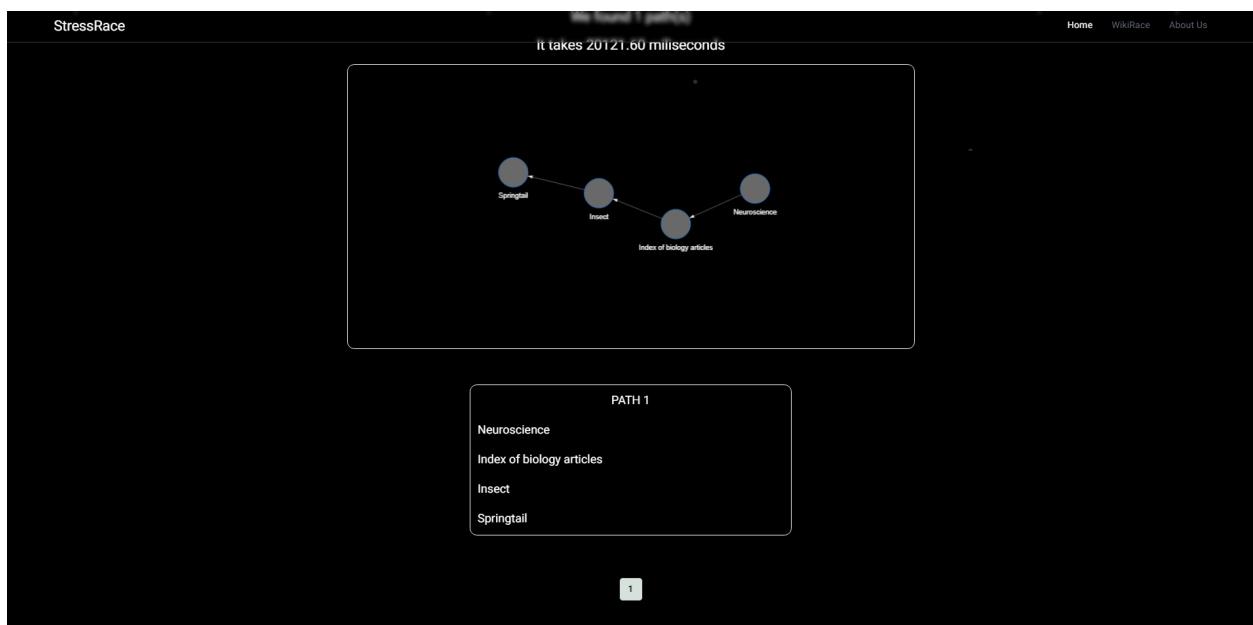


d. Pencarian : Neuroscience → Springtail

Kedalaman : 3

Jumlah artikel yang ditelusuri : 201054 artikel

Durasi : 20121,60 ms



4.3.2 Pencarian rute menggunakan algoritma BFS dengan multiple path

- Pencarian : Joko Widodo → Donald Trump

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Kedalaman : 2

Jumlah artikel yang ditelusuri : 162347 artikel

Jumlah rute : 72 rute

Durasi : 7874.00 ms

StressRace Home WikiRace About Us

Wiki Game Solver

Select Your Algorithm BFS

Select Your Number of Path Multiple Path

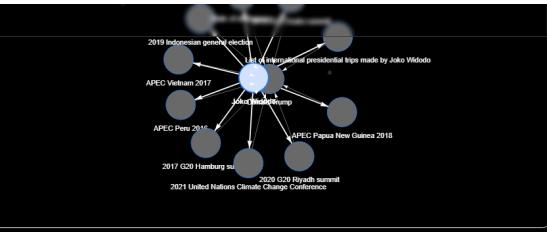
Joko Widodo ↔ Donald Trump

Search!

With the minimum of 2 degrees
In 162347 articles
We found 72 path(s)
It takes 7874.00 milliseconds



StressRace Home WikiRace About Us



PATH 1	PATH 2	PATH 3
Joko Widodo 2020 G20 Riyadh summit Donald Trump	Joko Widodo APEC Vietnam 2017 Donald Trump	Joko Widodo APEC Peru 2016 Donald Trump
PATH 4	PATH 5	PATH 6
Joko Widodo APEC Papua New Guinea 2018 Donald Trump	Joko Widodo 2019 G20 Osaka summit Donald Trump	Joko Widodo 2017 G20 Hamburg summit Donald Trump

1 2 3 4 5 ... 12

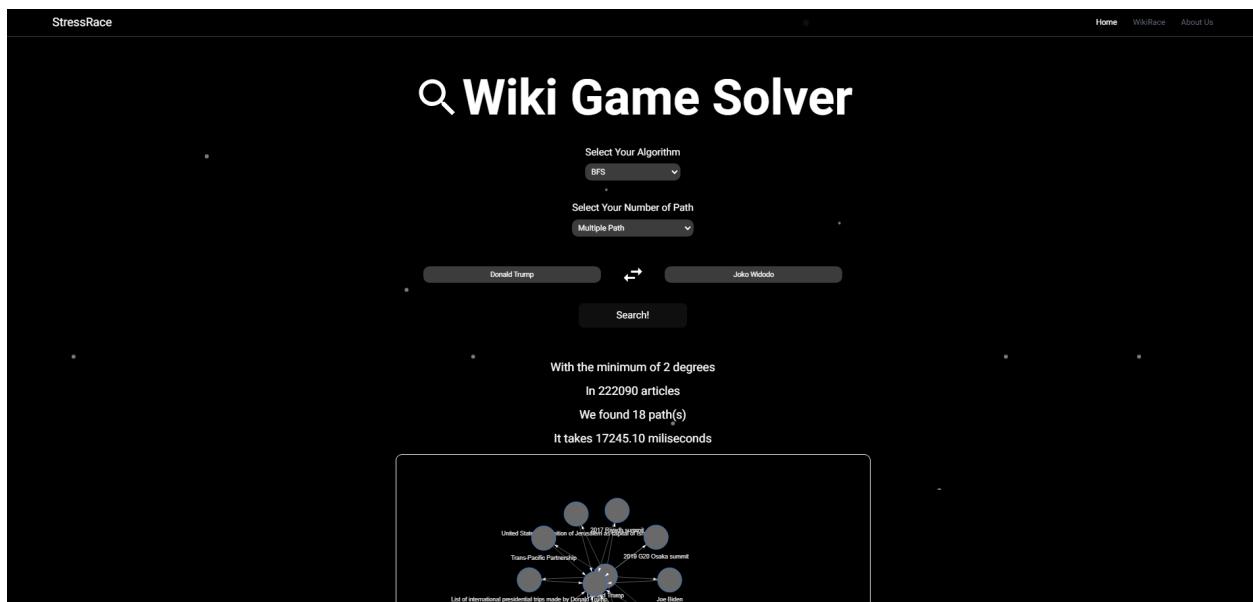
b. Pencarian : Donald Trump → Joko Widodo

Kedalaman : 2

Jumlah artikel yang ditelusuri : 222090 artikel

Jumlah rute : 18 rute

Durasi : 17245,10 ms



c. Pencarian : Thermal Micrometeoroid Garment → Toxic shock syndrome

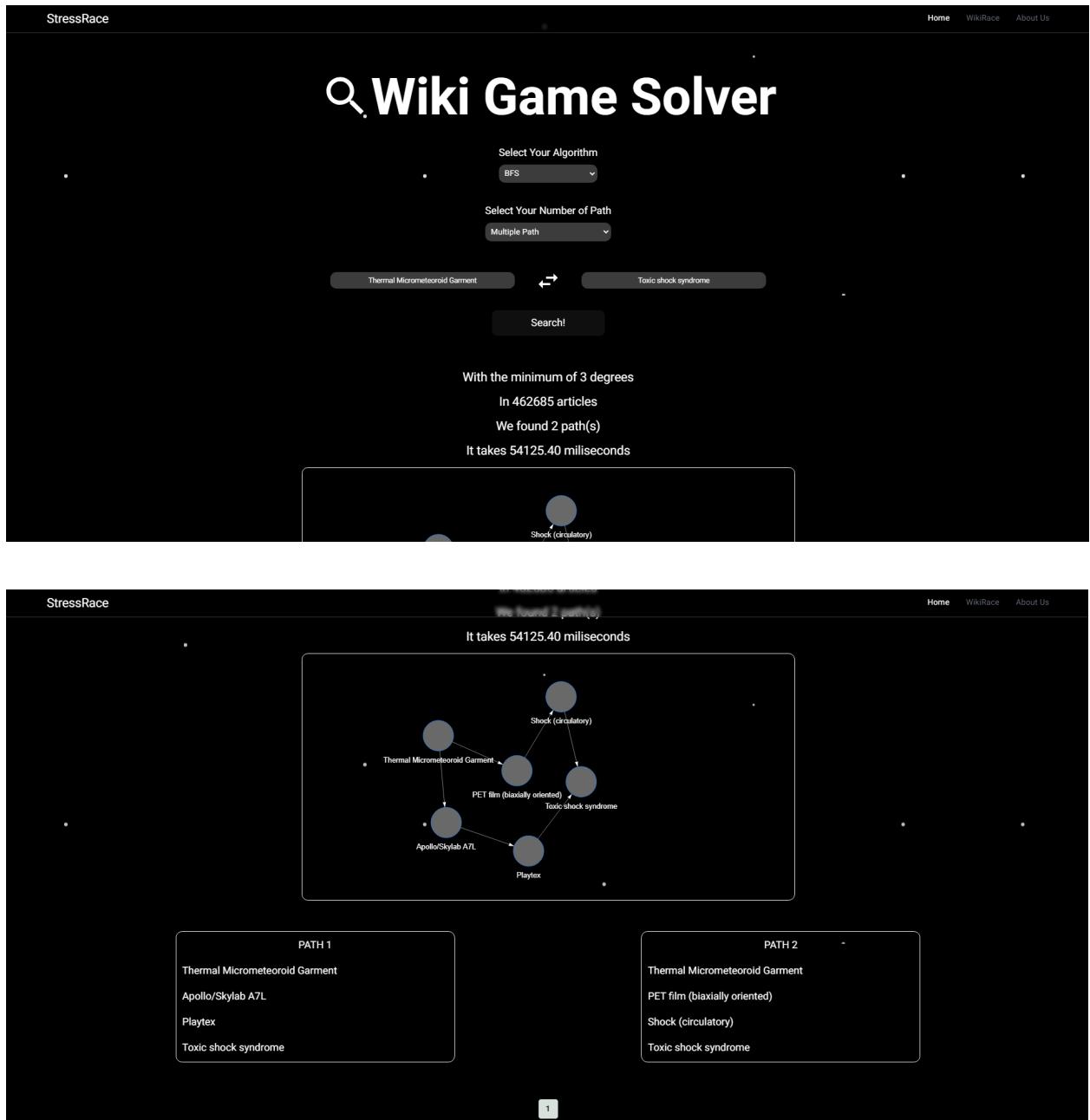
Kedalaman : 2

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Jumlah artikel yang ditelusuri : 462685 artikel

Jumlah rute : 2 rute

Durasi : 54125,40 ms



d. Pencarian : Neuroscience → Springtail

Kedalaman : 3

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Jumlah artikel yang ditelusuri : 1699971 artikel

Jumlah rute : 67 rute

Durasi : 421399,10 ms

The screenshot shows the StressRace Wiki Game Solver interface. At the top, there are dropdown menus for 'Select Your Algorithm' (set to BFS) and 'Select Your Number of Path' (set to Multiple Path). Below these are two input fields: 'Neuroscience' on the left and 'Springtail' on the right, separated by a double-headed arrow. A 'Search!' button is located below the inputs. To the right of the search area, text indicates: 'With the minimum of 3 degrees', 'In 1699971 articles', 'We found 67 path(s)', and 'It takes 421399.10 milliseconds'. Below this text is a network diagram with nodes for Arachnid, Myriapoda, Chelicera, Insect, Crustacean, and Neuroscience, connected by lines labeled 'Slow'. A central node is highlighted in blue. Below the network diagram, six paths are listed:

- PATH 1: Neuroscience, Index of biology articles, Insect, Springtail
- PATH 2: Neuroscience, Biology, Timeline of evolution, Springtail
- PATH 3: Neuroscience, Biology, Carnivore, Springtail
- PATH 4: Neuroscience, Biology, Insecta, Springtail
- PATH 5: Neuroscience, Biology, Snow, Springtail
- PATH 6: Neuroscience, Outline of biology, Myriapoda, Springtail

Pagination controls at the bottom allow for navigating through the 67 paths.

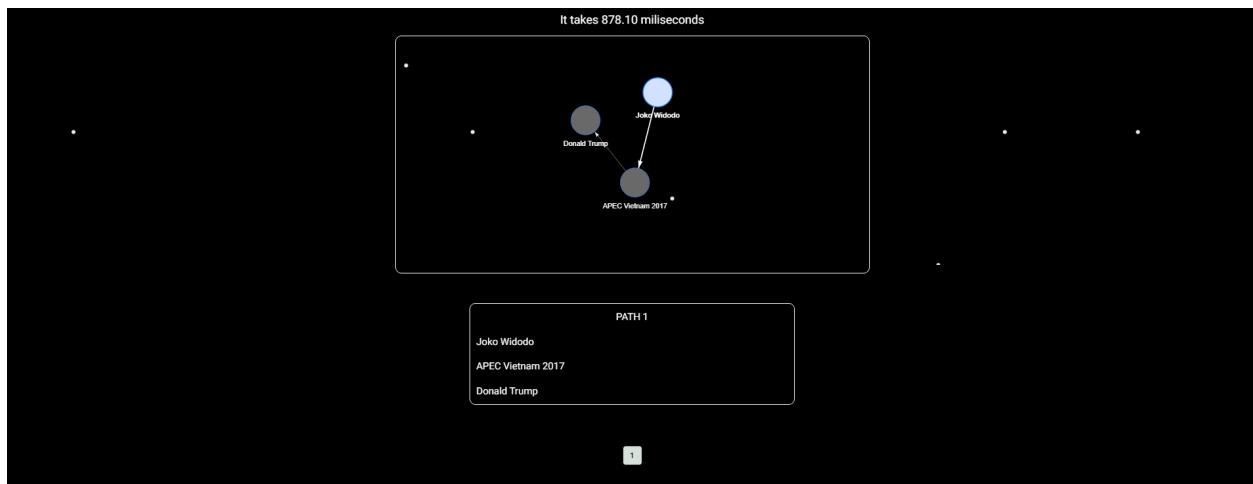
4.3.3 Pencarian rute menggunakan algoritma IDS dengan single path

- Pencarian : Joko Widodo → Donald Trump

Kedalaman : 2

Jumlah artikel yang ditelusuri : 72 artikel

Durasi : 878,10 ms



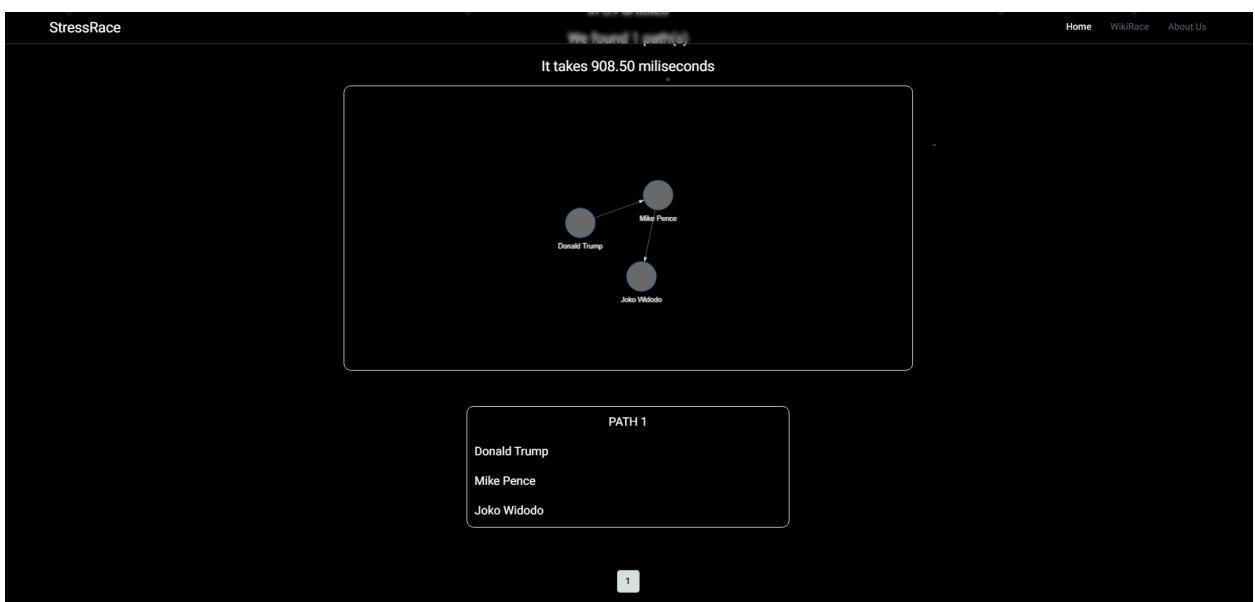
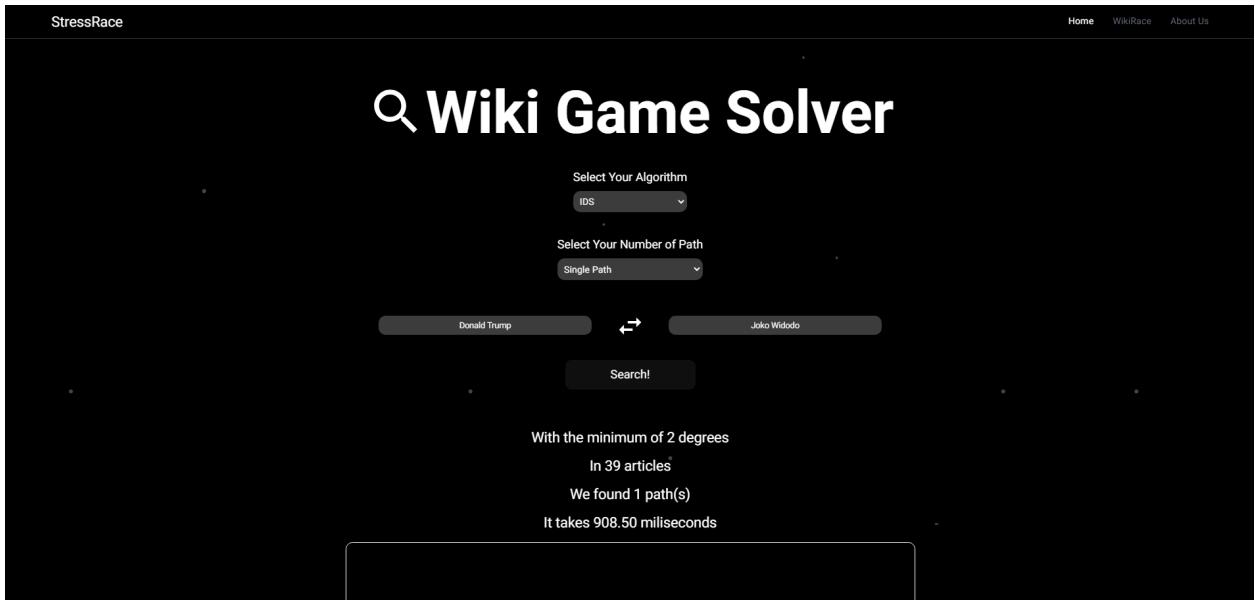
b. Pencarian : Donald Trump → Joko Widodo

Kedalaman : 2

Jumlah artikel yang ditelusuri : 39 artikel

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Durasi : 908,50 ms



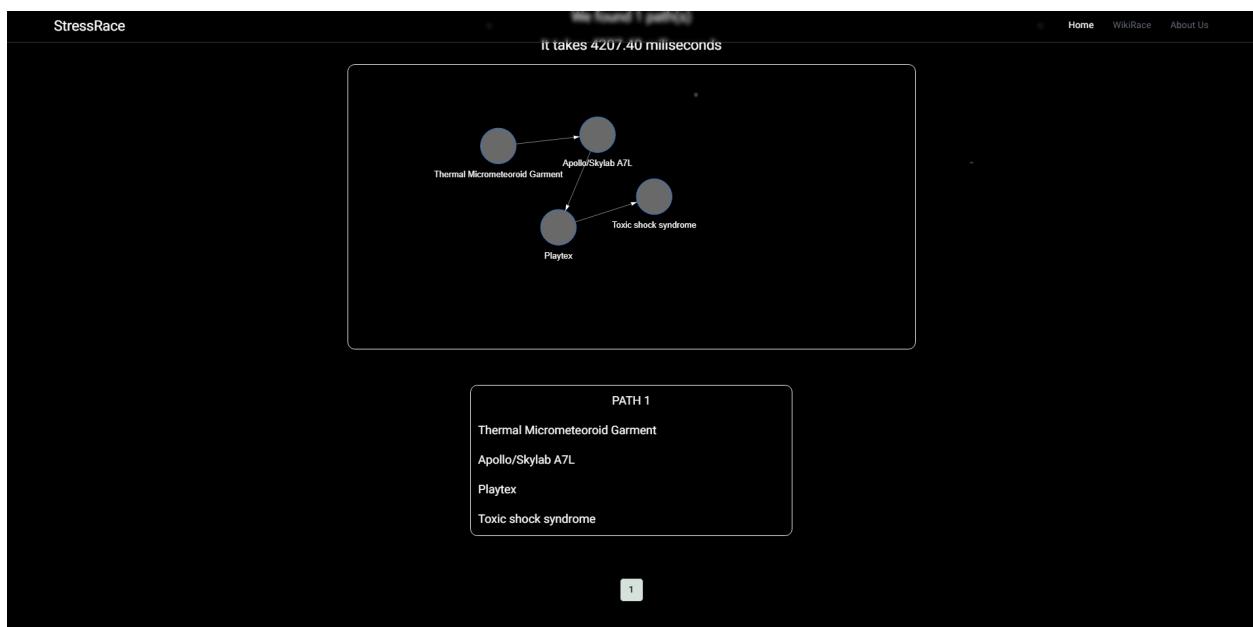
- c. Pencarian : Thermal Micrometeoroid Garment → Toxic shock syndrome

Kedalaman : 3

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Jumlah artikel yang ditelusuri : 633 artikel

Durasi : 4207,40 ms

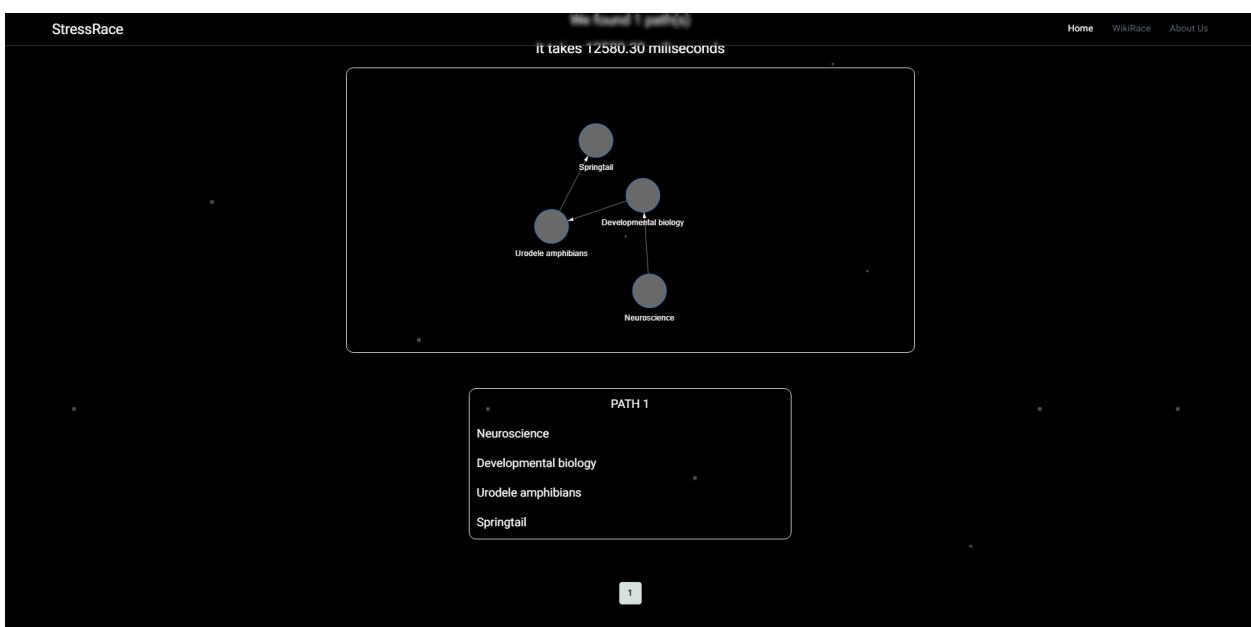
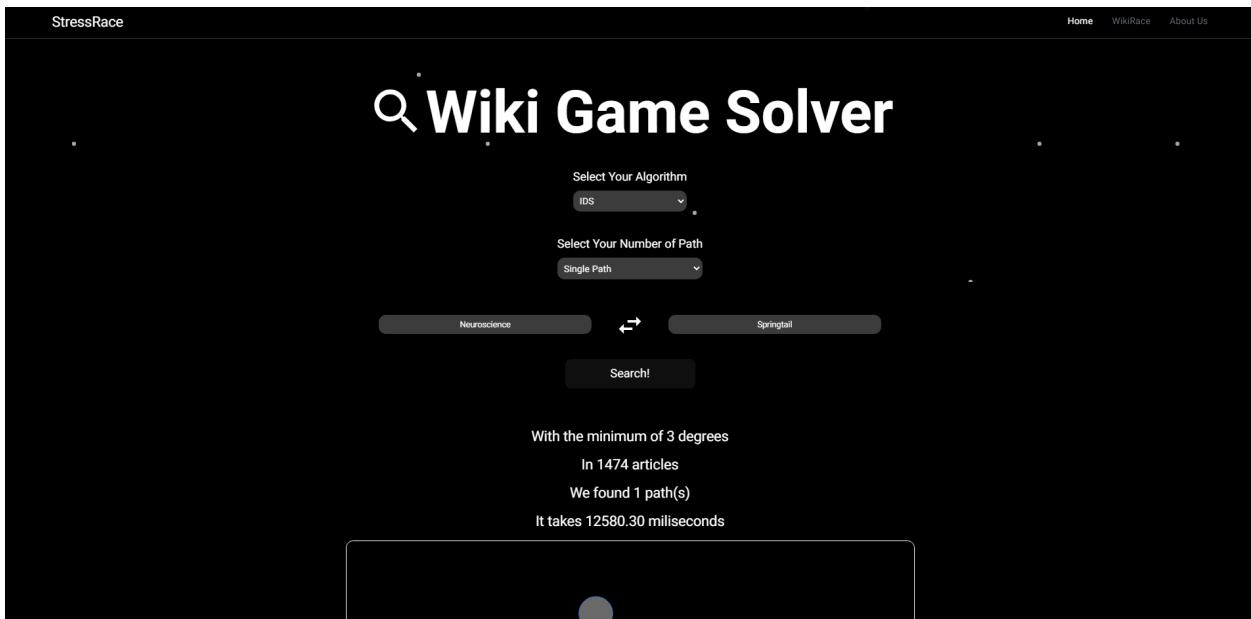


d. Pencarian : Neuroscience → Springtail

Kedalaman : 3

Jumlah artikel yang ditelusuri : 1474 artikel

Durasi : 12580,30 ms



4.3.4 Pencarian rute menggunakan algoritma IDS dengan multiple path

- Pencarian : Joko Widodo → Donald Trump

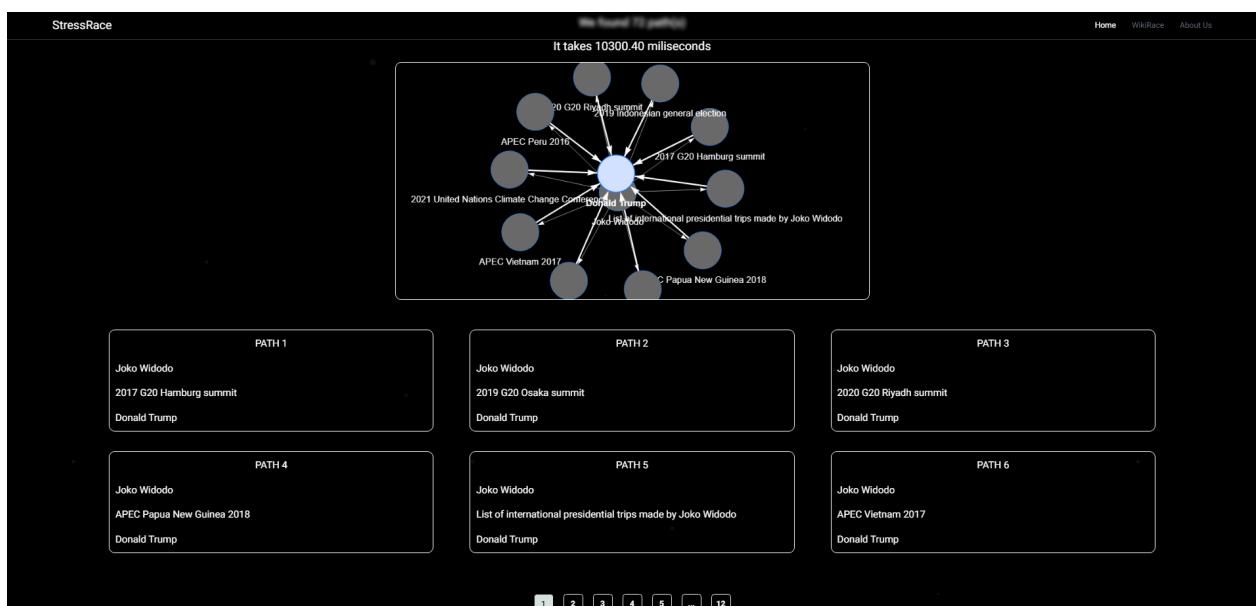
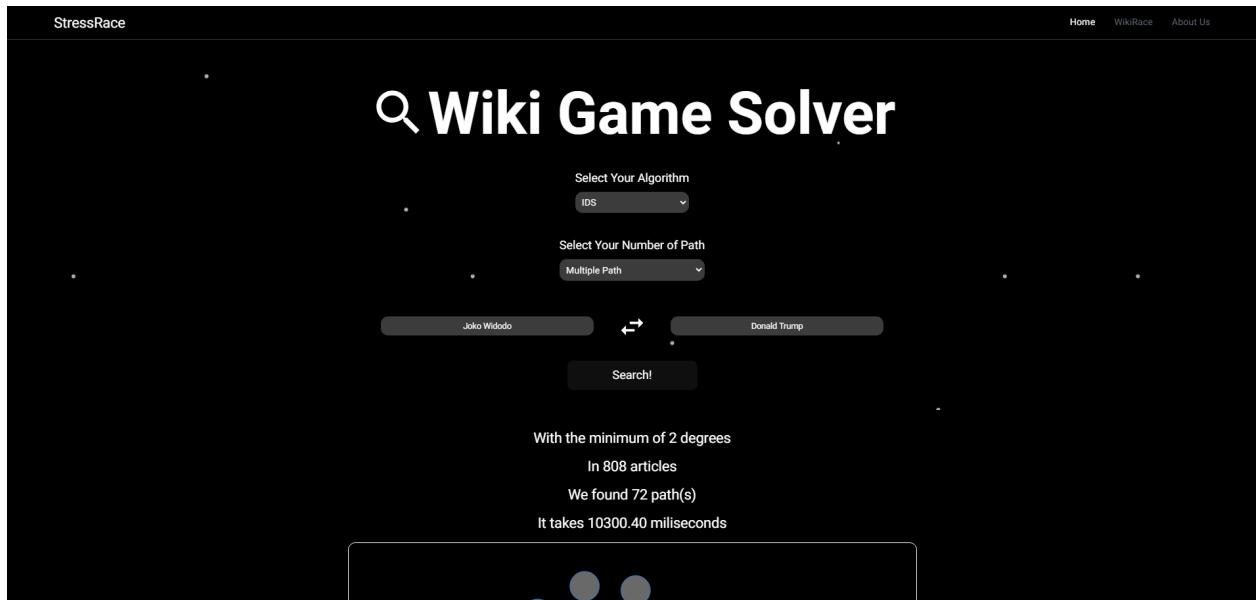
Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Kedalaman : 2

Jumlah artikel yang ditelusuri : 808 artikel

Jumlah rute : 72 rute

Durasi : 10300,40 ms



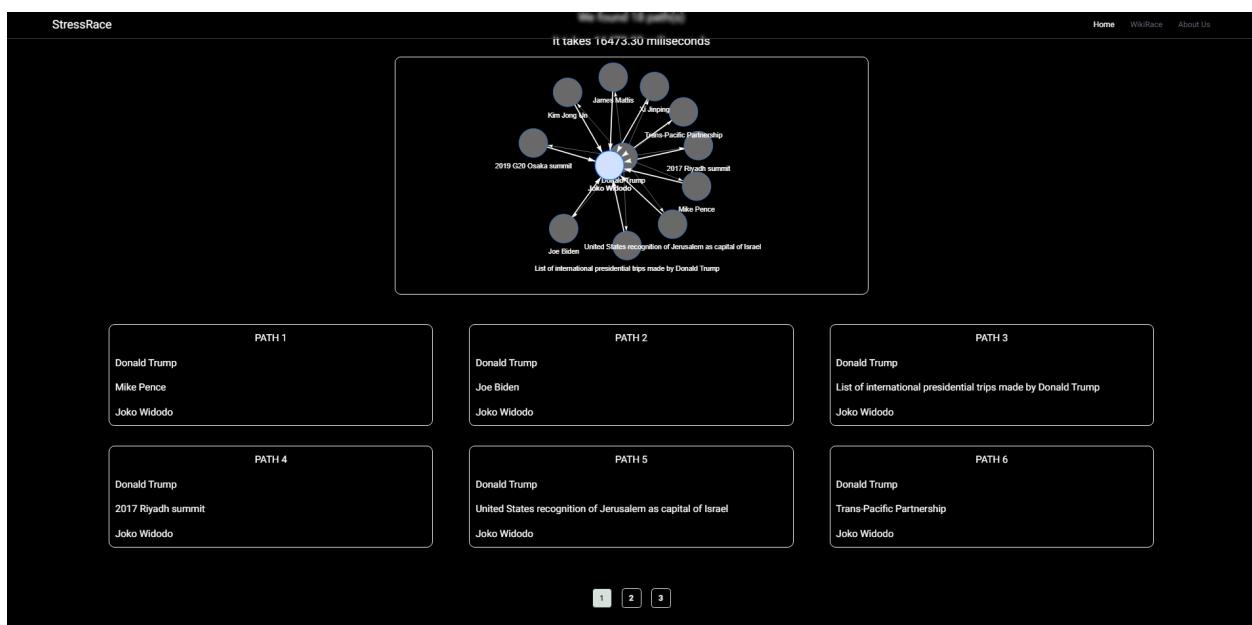
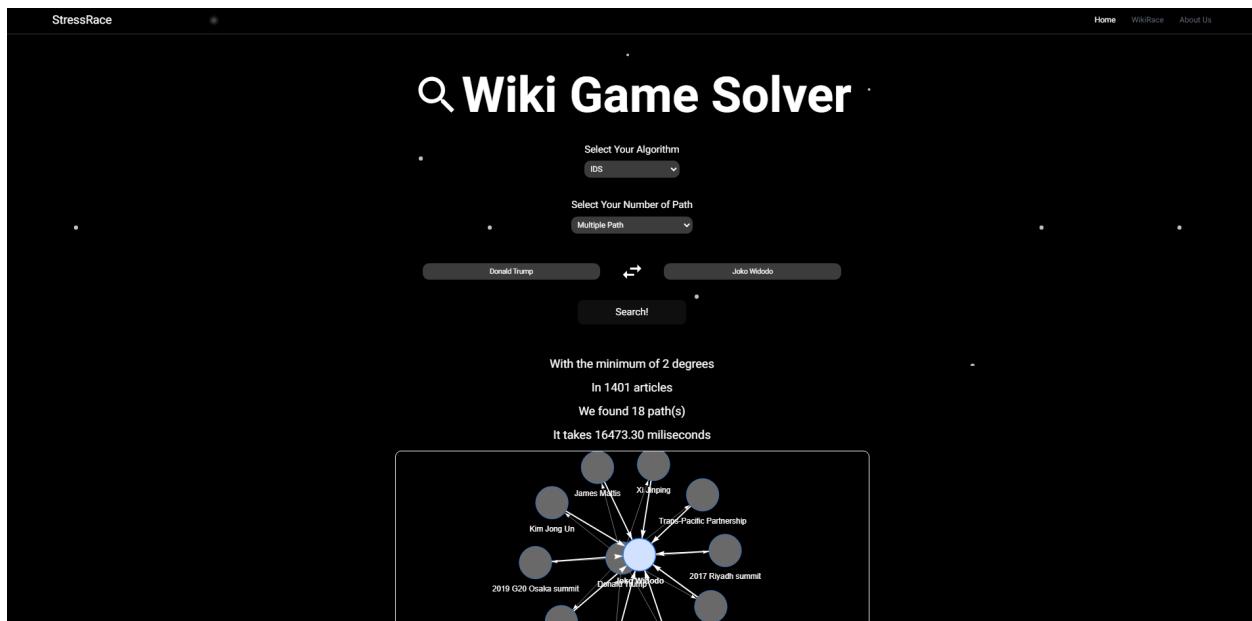
b. Pencarian : Donald Trump → Joko Widodo

Kedalaman : 2

Jumlah artikel yang ditelusuri : 1401 artikel

Jumlah rute : 18 rute

Durasi : 16473,30 ms



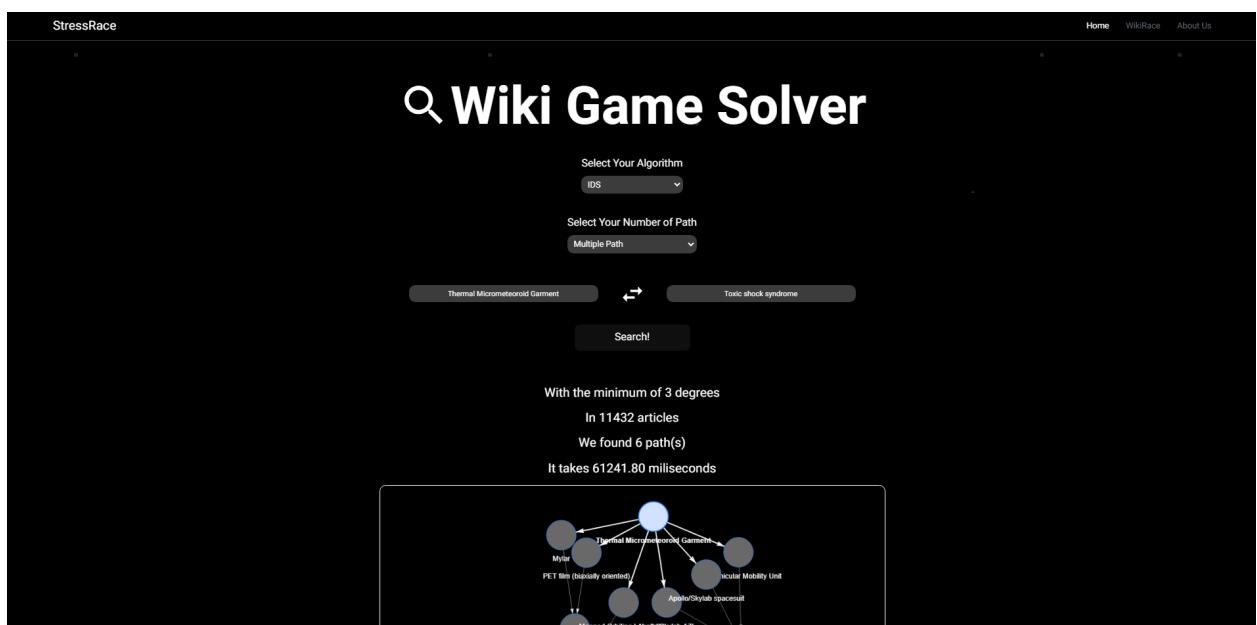
c. Pencarian : Thermal Micrometeoroid Garment → Toxic shock syndrome

Kedalaman : 3

Jumlah artikel yang ditelusuri : 11432 artikel

Jumlah rute : 6 rute

Durasi : 61241,80 ms



Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



d. Pencarian : Neuroscience → Springtail

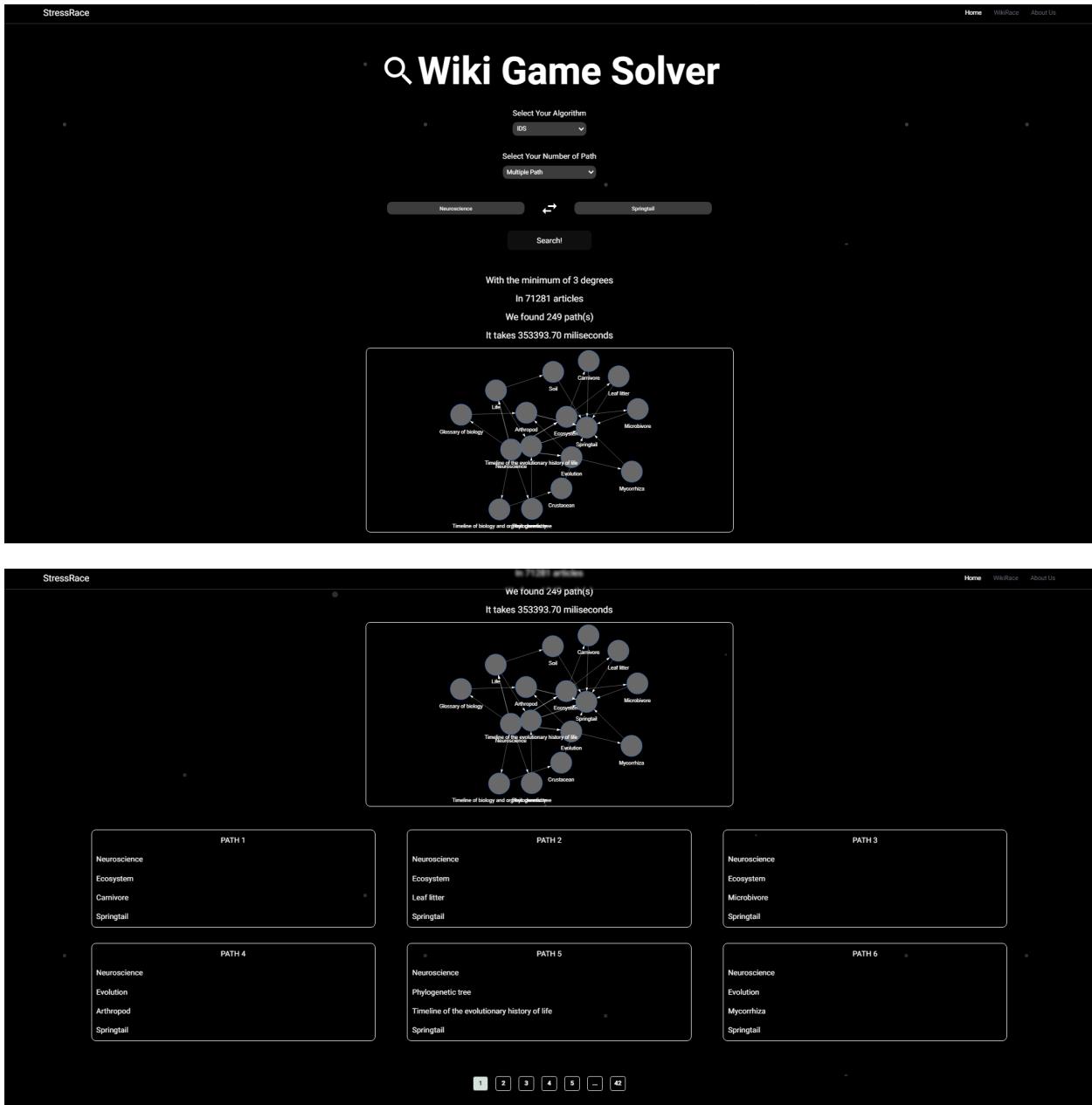
Kedalaman : 3

Jumlah artikel yang ditelusuri : 71281 artikel

Jumlah rute : 249 rute

Durasi : 353393,70 ms

Tugas Besar 2 IF 2123 Strategi Algoritma
Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace



4.4 Analisis

	Joko Widodo → Donald Trump Kedalaman = 2	Donald Trump → Joko Widodo Kedalaman = 2	Thermal Micrometeoroid Garment → Toxic shock syndrome	Neuroscience → Springtail Kedalaman = 3
--	--	--	---	---

			Kedalaman = 3	
BFS dengan single path	699,30 ms	1224,5 ms	18435,10 ms	20121,60 ms
BFS dengan multiple path	7874,00 ms	17245,10 ms	54125,40 ms	421399,10 ms
IDS dengan single path	878,10 ms	908,50 ms	4207,40 ms	4207,40 ms
IDS dengan multiple path	10300,40 ms	16473,30 ms	61241,80 ms	353393,70 ms

Tabel 4.1 Hasil Pengujian

Dari hasil pengujian diperoleh hasil sebagaimana tertera pada tabel 4.1 Hasil Pengujian. Dengan melihat hasil kedalaman pencarian sebagai parameter pembanding diperoleh kesimpulan sebagai berikut. Dalam konteks permainan WikiRace, di mana ruang pencarian terbatas pada artikel Wikipedia, BFS mungkin menjadi pilihan yang lebih baik jika memori cukup tersedia dan pencarian harus optimal. Hal ini bisa terlihat dari pencarian Jokowi dan Donald Trump dimana kedalaman yang diperoleh yaitu 2, BFS secara signifikan lebih cepat apabila dibandingkan dengan IDS. Namun, jika memori terbatas atau pencarian dapat dipertahankan dalam batasan kedalaman tertentu, IDS bisa menjadi pilihan yang lebih baik karena lebih hemat memori. Dapat dilihat juga bahwa dalam mencari solusi pada kedalaman yang tinggi, terlihat bahwa algoritma IDS lebih unggul, hal ini disebabkan oleh penggunaan memori oleh BFS yang meningkat secara eksponensial semakin lebar suatu graf. Waktu pencarian solusi juga dipengaruhi oleh faktor eksternal yakni koneksi jaringan, koneksi jaringan yang kuat, berdasarkan percobaan penulis, membantu mempercepat pencarian solusi, hal tersebut dikarenakan *Web Scraping* yang secara prinsip membutuhkan koneksi internet.

BAB 5

Penutup

5.1 Kesimpulan

Tugas pembuatan website Wikirace dengan pemanfaatan Algoritma IDS dan BFS merupakan suatu tantangan yang menarik dan relevan dalam mata kuliah Strategi Algoritma. Penerapan algoritma IDS dan BFS tersebut memberikan kontribusi signifikan terhadap peningkatan akurasi dan keefektifan sistem dalam mencari hubungan antar satu website wikipedia dengan website wikipedia lainnya. Penggunaan metode dan teknik yang tepat dalam ekstraksi fitur, representasi data, dan pengindeksan memberikan kontribusi yang sangat signifikan terhadap hasil yang optimal.

5.2 Saran

Untuk selanjutnya manajemen waktu harus dapat dijaga dengan baik karena pengeraaan masih serba dadakan sehingga kurang tereksekusi maksimal. Selain itu pembagian kerja juga harus lebih merata lagi karena beban kerja yang sekarang masih tidak seimbang sehingga tidak tereksekusi secara efisien. Dalam hal komunikasi juga masih kurang optimal sehingga terdapat miskomunikasi yang menghambat pengeraaan. Penggunaan bahasa pemrograman yang relatif baru untuk tugas besar kali ini juga menjadi tantangan tersendiri sehingga membuat kami berlatih lebih lagi.

5.3 Refleksi

Selama pengeraaan tugas besar ini dirasa masih kurang dalam hal manajemen waktu karena terdapat banyak tugas besar lain dan pengalokasian waktu untuk tugas besar ini masih kurang sehingga tidak terselesaikan dengan optimal. Selain itu pembagian beban kerja yang kurang seimbang dan terdapat pula masalah komunikasi dalam kelompok.

Daftar Pustaka

Munir, Rinaldi. 2024, “[Breadth First Search \(BFS\) dan Depth First Search \(DFS\) \(Bagian 1\)](#)”
Diakses 20 April 2024, dari

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

Munir, Rinaldi. 2024, “[Breadth First Search \(BFS\) dan Depth First Search \(DFS\) \(Bagian 2\)](#)”
Diakses 20 April 2024, dari

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

<https://www.sixdegreesofwikipedia.com/>

<https://github.com/PuerkitoBio/goquery>

Lampiran

Github : https://github.com/FrancescoMichael/Tubes2_StressRace.git

Video : https://youtu.be/gW_4sZ4kHMg