Tugas Besar 2 IF2211 Strategi Algoritma

Pemanfaatan Algoritma IDS dan BFS dalam Permainan WikiRace

Disusun Oleh :

10023500 - Miftahul Jannah

13518108 - Vincent Hasiholan

13522106 - Christopher Brian

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

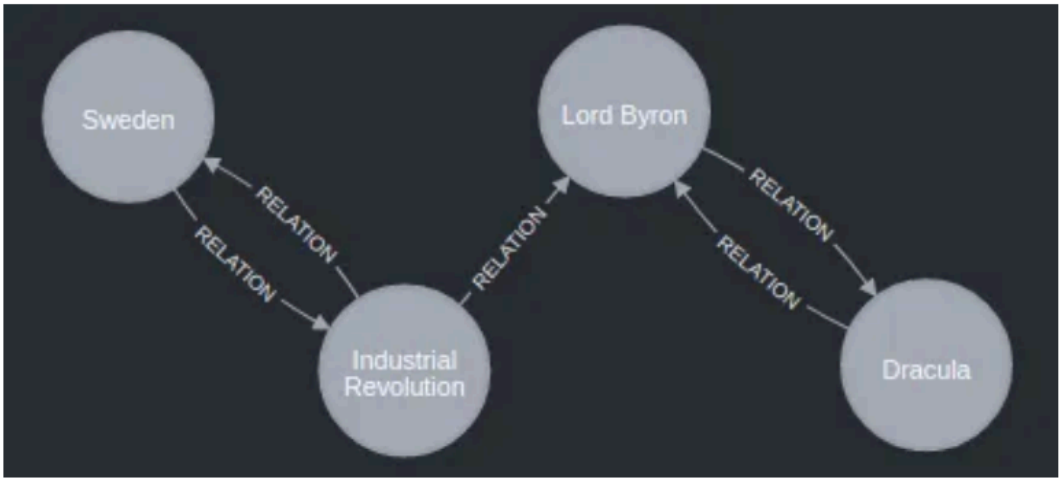
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Bab 1: Deskripsi Tugas

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

Bab 2: Landasan Teori

2.1 Penjelajahan Graf

- Tanpa informasi (uninformed/blind search)
    - Tidak ada informasi tambahan yang disediakan
    - Contoh: DFS, BFS, Depth Limited Search, Iterative Deepening Search, Uniform Cost Search
- Dengan informasi (informed Search)
    - Pencarian berbasis heuristik
    - Mengetahui non-goal state yang “lebih menjanjikan” daripada yang lain
    - Contoh: Best First Search, A*

Dalam proses pencarian solusi, terdapat dua pendekatan:

1. Graf statis: graf yang sudah terbentuk sebelum proses pencarian dilakukan - graf direpresentasikan sebagai struktur data
2. Graf dinamis: graf yang terbentuk saat proses pencarian dilakukan - graf tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi

2.2 Iterative Deepening Search (IDS)

Konsep IDS menggabungkan keuntungan dari dua metode pencarian lainnya, yaitu Breadth-First Search (BFS) dan Depth-First Search (DFS).
IDS bekerja dengan melakukan pencarian secara DFS dengan batasan kedalaman tertentu, dan jika solusi tidak ditemukan, meningkatkan batasan kedalaman tersebut dan melakukan pencarian ulang. Proses ini terus diulang dengan meningkatkan batasan kedalaman secara iteratif hingga solusi ditemukan. IDS memiliki kompleksitas waktu yang serupa dengan DFS tetapi dapat menemukan solusi optimal seperti pada BFS, tanpa membutuhkan banyak memori seperti pada BFS.

- IDS: melakukan serangkaian DFS, dengan peningkatan nilai kedalaman-cutoff, sampai solusi ditemukan

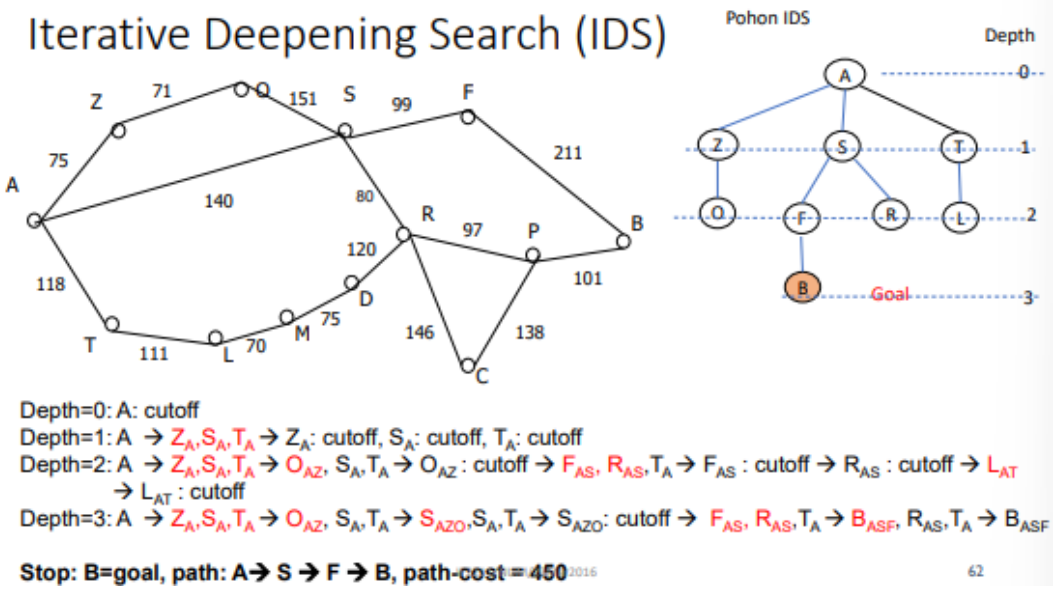
- Asumsi: simpul sebagian besar ada di level bawah, sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali

```
Depth ← 0
Iterate
 result← DLS(problem,depth)
stop: result ≠ cutoff
 depth← depth+1
→ result
```

Properti dari IDS

- Completeness?
  - Ya, jika b terbatas
- Optimality?
  - Ya, jika langkah = biaya

- Kompleksitas waktu:
  - $O(b^d)$
- Kompleksitas ruang:
  - $O(b^d)$



2.3 Breadth-First Search (BFS)

Breadth First Search (BFS) adalah **algoritma traversal grafik** dasar. Ini melibatkan mengunjungi semua node yang terhubung pada grafik secara tingkat demi tingkat. Pada artikel ini, kita akan melihat konsep **BFS** dan bagaimana penerapannya pada grafik secara efektif

Breadth First Search (BFS) sering disebut pencarian Melebar.

Algoritma ini secara sistematis menjelajahi atau mencari semua simpul pada kedalaman yang sama sebelum melanjutkan ke simpul-simpul pada kedalaman berikutnya dalam graf.

Breadth First Search (BFS) untuk Algoritma Grafik:

Mari kita bahas algoritma BFS:

1. **Inisialisasi:** Masukkan node awal ke dalam antrian dan tandai sebagai telah dikunjungi.
2. **Eksplorasi:** Saat antrian tidak kosong:
    - Keluarkan node dari antrian dan kunjungi node tersebut (misalnya, cetak nilainya).
    - Untuk setiap tetangga yang belum dikunjungi dari node yang di-dequeued:
        - Antri tetangga ke dalam antrian.
        - Tandai tetangga sebagai telah dikunjungi.
3. **Penghentian:** Ulangi langkah 2 hingga antrian kosong.

Algoritma ini memastikan bahwa semua node dalam grafik dikunjungi secara luas, dimulai dari node awal.

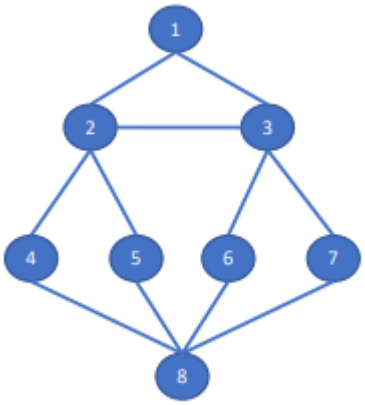
• Traversal dimulai dari simpul v.

• Algoritma:

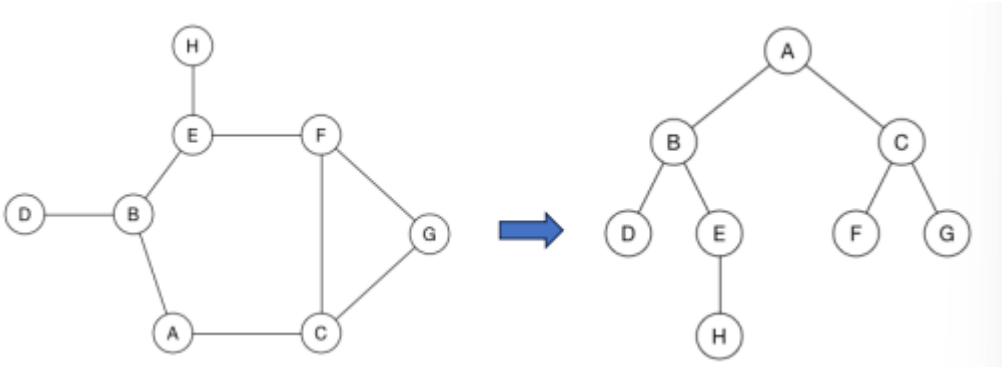
1. Kunjungi simpul v

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.



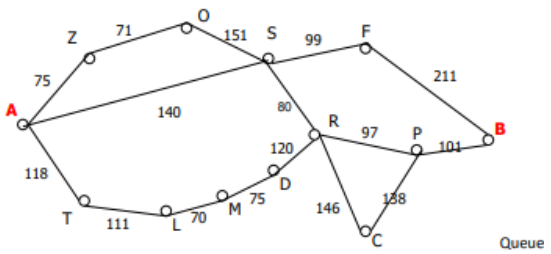
• Traversal secara BFS pada graf tersebut dapat digambarkan sebagai pohon BFS:

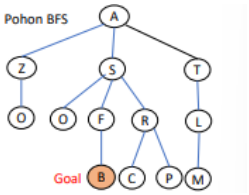


Urutan simpul-simpul yang dikunjungi secara BFS dari A → A, B, C, D, E, F, G, H

Breadth-First Search (BFS)

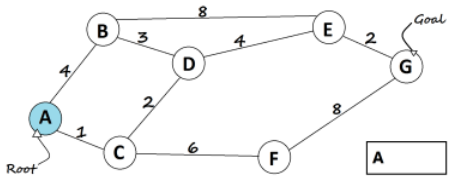
Treat agenda as a queue (FIFO)





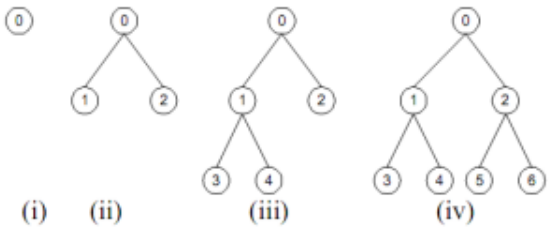
Simpul-E	Simpul Hidup
A	Z_A,S_A,T_A
Z_A	S_A,T_A,O_{AZ}
S_A	T_A,O_{AZ},O_{AS},F_{AS},R_{AS}
T_A	O_{AZ},O_{AS},F_{AS},R_{AS},L_{AT}
O_{AZ}	O_{AS},F_{AS},R_{AS},L_{AT}
O_{AS}	F_{AS},R_{AS},L_{AT}
F_{AS}	R_{AS},L_{AT},B_{ASG}
R_{AS}	L_{AT},B_{ASG},D_{ASR},C_{ASR},P_{ASR}
L_{AT}	B_{ASG},D_{ASR},C_{ASR},P_{ASR},M_{ATL}
B_{ASG}	Solusi ketemu

BFS:



Sumber:https: //medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

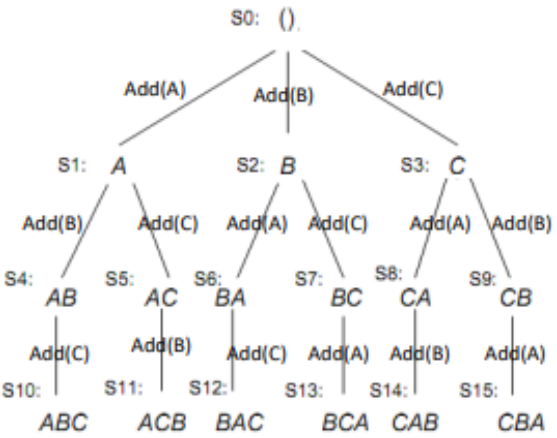
BFS untuk Pembentukan Pohon Ruang Status



• Inisialisasi dengan status awal sebagai akar, lalu tambahkan simpul anaknya, dst.

• Semua simpul pada level d dibangkitkan terlebih dahulu sebelum simpul-simpul pada level d+1

BFS untuk Permutasi

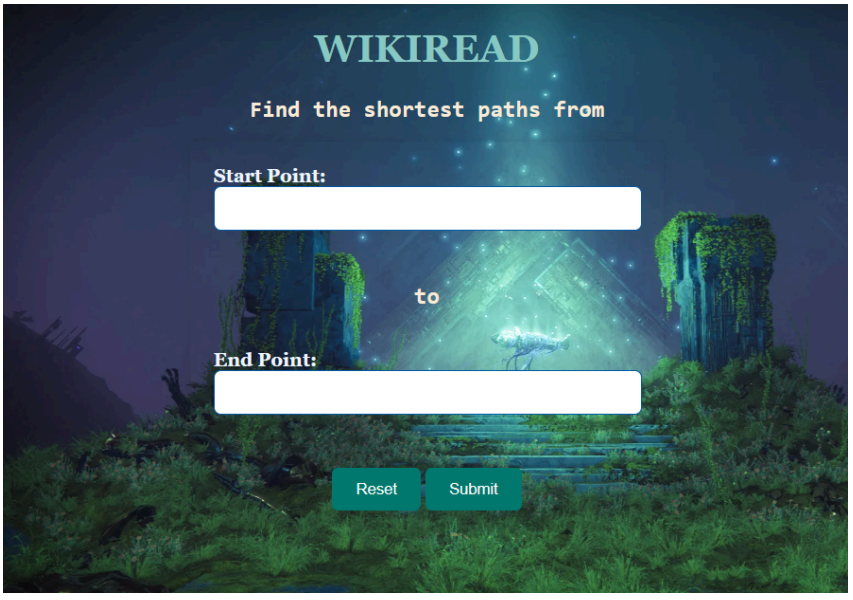


Properti dari BFS

- Completeness?
    - Ya (selama nilai b terbatas)
- Optimality?
    - Ya, jika langkah = biaya
- Kompleksitas waktu:
    - 1+b+b 2+b 3+…+b^d = O(b^d)
- Kompleksitas ruang:
    - O(b^d)
- Kurang baik dalam kompleksitas ruang

2.4 Aplikasi Web

Aplikasi web ini menggunakan HTML dan CSS, berikut tampilan websitenya



Aplikasi ini bernama WIKIREAD dimana ada ‘Start Point’ untuk titik awal dan ‘End Point’ untuk titik akhirnya. Terdapat juga tombol ‘Reset’ dan ‘Submit’.

mainpage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
       <meta  name="viewport"  content="width=device-width,
   initial-scale=1.0">
    <title>Wikiread - Tugas Besar 2 Stima</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>WIKIREAD</h1>
    <div class="firstParagraph">
        <h2>Find the shortest paths from</h2>
    </div>
    <div class="mainprogram">
        <form action="/submit" method="GET">
            <div class="StartPoint">
                <label for="start">Start Point: </label>
                            <input  type="text"  id="start"
   name="start">
            </div>
            <br>
            <h2>to</h2>
            <br>
            <div class="EndPoint">
                <label for="end">End Point: </label>
                <input type="text" id="end" name="end">
            </div>
            <br>
            <br>
            <div class="option">
                <input type="reset">
                <input type="submit">
            </div>
```

```
            <br>
            <br>
            <br>
        </form>
    </div>

    {{if .Paths}}
    <div id="results">
        <h2>Shortest Paths BFS:</h2>
        {{range .Paths}}
        <ul>
            {{range .}}
            <li>{{.}}</li>
            {{end}}
        </ul>
        {{end}}
    </div>
    {{end}}

    {{if .Paths2}}
    <div id="results2">
        <h2>Shortest Paths IDS:</h2>
        {{range .Paths2}}
        <ul>
            {{range .}}
            <li>{{.}}</li>
            {{end}}
        </ul>
        {{end}}
    </div>
    {{end}}

    <div id="additionalInfo">
        <h2>Additional Information:</h2>
        {{if .TotalPagesVisited}}
                        <p>Total   Pages   Visited   (BFS):
{{.TotalPagesVisited}}</p>
        {{end}}
        {{if .ShortestPathDepth}}
```

```
                    <p>Shortest Path Depth (BFS):
    {{.ShortestPathDepth}}</p>
        {{end}}
        {{if .SearchTimeInSeconds}}
                    <p>Search Time (BFS) (seconds):
    {{.SearchTimeInSeconds}}</p>
        {{end}}
        {{if .TotalPagesVisited2}}
                    <p>Total Pages Visited (IDS):
    {{.TotalPagesVisited2}}</p>
        {{end}}
        {{if .ShortestPathDepth2}}
                    <p>Shortest Path Depth (IDS):
    {{.ShortestPathDepth2}}</p>
        {{end}}
        {{if .SearchTimeInSeconds2}}
                    <p>Search Time (IDS) (seconds):
    {{.SearchTimeInSeconds2}}</p>
        {{end}}
     </div>
</body>
</html>
```

style.css

```
body {
    font-family:Georgia, 'Times New Roman', monospace;
    background-color: lightblue;
    margin: 0;
    padding: 0;
}

h1 {
    text-align: center;
    color: #88cbc6;
}

.firstParagraph {
```

```
    text-align: center;
    margin-top: auto;
    margin-bottom: auto;
    font-family:monospace;
    color:antiquewhite;
}

.mainprogram {
    width: 50%;
    margin: 0 auto;
    background-color: transparent;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    color: aliceblue;
}

.result {
        color: white; /* Add this rule to change the
   background color of the result text */
    padding: 10px;
    border-radius: 5px;
    margin-top: 20px;
}

h2 {
    text-align: center;
    font-family: monospace;
    color:antiquewhite;

}
label {
    font-weight: bold;
}

input[type="text"] {
    width: calc(100% - 20px);
    padding: 10px;
    margin-bottom: 10px;
```

```
    border: 1px solid #035AA6;
    border-radius: 6px;
    color: #000000;

}

input[type="submit"],
input[type="reset"] {
    padding: 10px 20px;
    background-color: #01796F;
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

input[type="submit"]:hover,
input[type="reset"]:hover {
    background-color: #01796F;
}

.option {
    text-align: center;
}
```

Bab 3: Analisis Pemecahan Masalah

3.1 Langkah-langkah pemecahan masalah

1. Aplikasi mengambil value dari form untuk diolah titik mulai pencarian dan titik akhir dari pencarian.
2. Ketika value sudah didapatkan, aplikasi melakukan *crawling* atau bergerak merombak seluruh atribut HTML yang dapat memindah ke website lain.
3. Semua atribut ini disimpan dan digunakan sebagai link perpindahan menuju titik terakhir.
4. Ulangi langkah dua dan tiga hingga pencarian sudah sampai di titik terakhir yang diinginkan.

3.2 Proses pemetaan masalah menjadi elemen-elemen algoritma IDS dan BFS

Masalah yang diangkat dari wikirace ini adalah untuk mencari seluruh *hyperlink* yang ada di dalam satu website sehingga *hyperlink* itu dapat digunakan sebagai elemen pencarian. Aplikasi harus dapat memisahkan seluruh atribut yang tidak memiliki *hyperlink* dengan yang memiliki *hyperlink* dan menyimpannya ke sebuah list yang digunakan aplikasi untuk berpindah ke website tersebut hingga ditemukan website yang merupakan tujuan yang diinginkan.

Dalam Tugas Besar ini, algoritma IDS yang kamu gunakan berupa Depth Limited Search (DLS) secara iteratif dengan depth limit yang meningkat di setiap iterasi. DLS akan mencari semua kemungkinan path dari page yang sedang dicek ke page yang bisa dicapai menggunakan link secara rekursif. Jika target page ditemukan, maka path ke page tersebut akan dikembalikan. Jika tidak, depth limit akan ditingkatkan pada iterasi berikutnya, dan pencarian akan terus dilakukan hingga target page ditemukan pada path atau depth limit maksimum dicapai, yang kami atur menjadi maksimal 10 degree. Algoritma ini menggabunkan completeness

dari Depth First Search (DFS) dan efisiensi dari Breadth First Search (BFS).

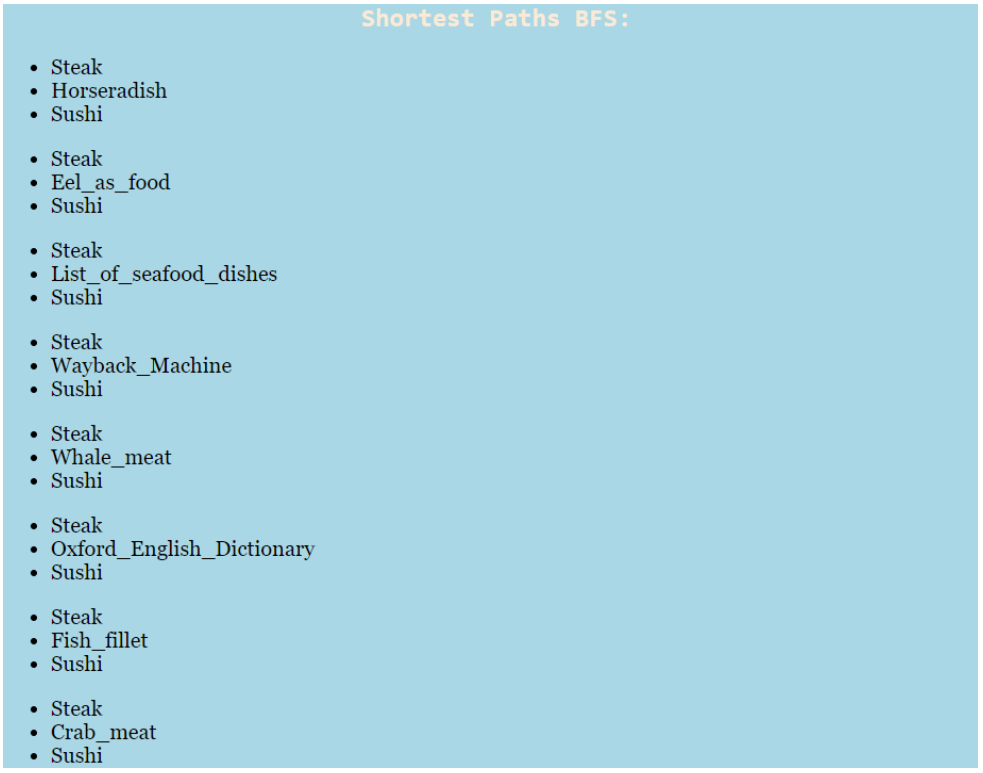
Kami juga menggunakan algoritma BFS sebagai pembanding. Algoritma yang kami gunakan berupa BFS direksional, sehingga proses pencarian dimulai dari source page dan juga target page, dengan tujuan untuk mencari page di tengah-tengah yang bisa dicapai dari kedua page tersebut. Terdapat dua daftar page, yaitu unvisitedForward, yang berupa page yang belum dikunjungi pada arah maju (dari source page menuju target page), dan unvisitedBackward (dari arah sebaliknya). Algoritma ini akan mengembangkan scope pencarian secara iteratif dari kedua arah sampai menemukan page yang bisa diakses dari source dan target page. Kemudian, algoritma akan mencari semua path yang mungkin dari source page ke target page melalui page yang bisa diakses dari kedua arah tersebut.

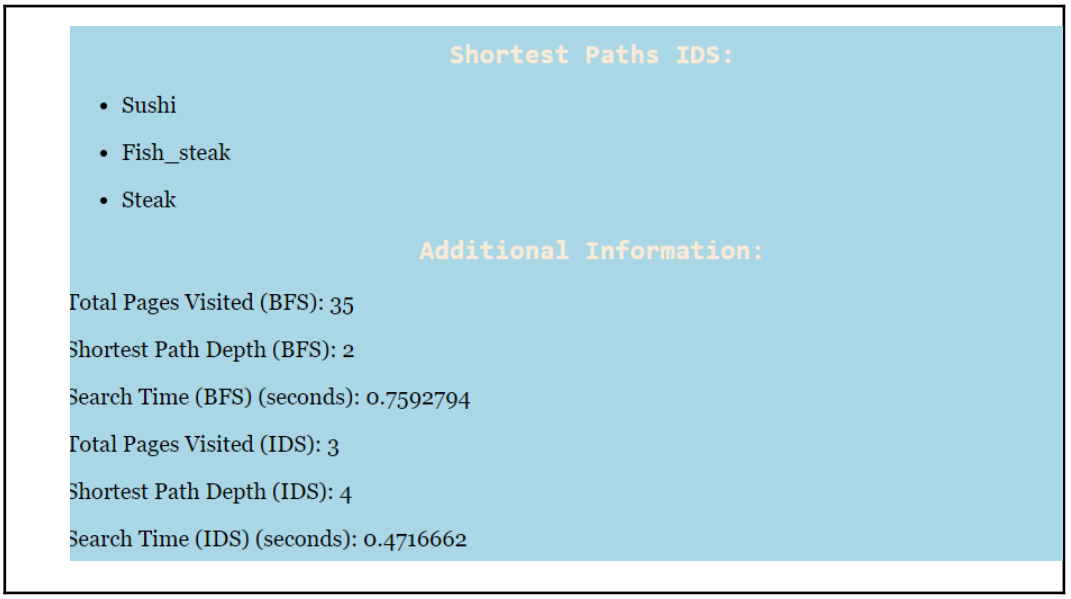
3.3 Fitur fungsional dan arsitektur aplikasi web yang dibangun

Website kami akan menerima input berupa judul artikel asal dan artikel tujuan, dan akan secara otomatis mengkonversinya menjadi url wikipedia yang valid sebagai source page dan target page. Setelah tombol submit diklik, website akan menampilkan path terpendek dari source page ke target page, jumlah page yang dilalui, jumlah page yang dilangkahi, dan total waktu yang diperlukan untuk mendapatkan path terpendek.

3.4 Contoh ilustrasi kasus

Misalkan seorang pengguna hendak melakukan pencarian dari Steak ke Sushi. Kedua nilai input ini diproses oleh program dibelakang website untuk mendapatkan link - link yang mengarahkan ke page lain dari kedua page. Ketika sudah ditemukan path terpendek dari page asal ke page tujuan, maka program berhenti dan menampilkan hasil dari pencarian tersebut.





Bab 4: Implementasi dan Pengujian

4.1 Spesifikasi teknis program

Sebelum menjalankan program, periksa dulu apakah semua requirement di bawah ini sudah terpenuhi

1. Melakukan pengecekan apakah PC/laptop sudah terinstall go v1.22.2 atau ke atas di terminal
2. Terdapat fungsi main yang berisikan logika - logika yang aplikasi jalankan.
3. Fungsi BFS dan IDS ditaruh di file pathfinding.go untuk membedakan algoritma yang hendak digunakan.

Selanjutnya, akan dibahas mengenai struktur data dan fungsi yang dibangun untuk menyelesaikan Tugas Besar ini. Secara garis besar, terdapat 2 struktur data utama yang digunakan, yaitu maps/dictionaries yaitu slices/arrays. Maps digunakan untuk merepresentasikan graf dalam bentuk adjacency matrix untuk menyimpan informasi page apa saja yang terhubung melalui link. Maps ini juga menyimpan data page mana saja yang sudah dikunjungi dan yang belum. Slices/arrays digunakan untuk menyimpan paths dan links pada pemrosesan algoritma BFS dan IDS.

Terdapat beberapa fungsi yang digunakan. Fungsi-fungsi getLink, getPaths, dan containsPath digunakan di fungsi pencarian menggunakan algoritma BFS dan IDS untuk pemrosesan. Fungsi getLink digunakan untuk mencari semua external links yang mungkin menghubungan page asal dan page tujuan. Fungsi ini mengembalikan daftar page dan pesan error. Fungsi getPaths digunakan untuk mencari semua path yang bisa diambil dari page tertentu ke page asal atau page tujuan. Fungsi ini mengembalikan daftar path. Fungsi containsPath digunakan untuk mengecek apakah daftar path mengandung path tertentu. Fungsi ini mengembalikan boolean.

Fungsi BFS digunakan untuk melakukan pencarian/pathfinding menggunakan algoritma BFS secara bidireksional dan mengembalikan daftar path terpendek untuk mencapai target page dari source page. Fungsi ini memiliki daftar page yang dikunjungi dan belum dikunjungi, masing-masing untuk arah maju (dari source page ke target page) dan arah mundur (sebaliknya). Dilakukan iterasi pathfinding sampai ditemukan semua path terpendek atau semua page telah dikunjungi. Setiap iterasi dilakukan pada arah dengan jumlah link terkecil di tingkat berikutnya. Pada setiap arah prosesnya kurang lebih sebagai berikut, fungsi akan mengambil link yang bisa dikunjungi dari page yang sedang dicek, tandai semua page yang belum dikunjungi sebagai telah dikunjungi, kosongkan dictionary page yang belum dikunjungi, lalu jika page tujuan tidak ada pada dictionary telah dikunjungi atau belum dikunjungi, masukkan ke belum dikunjungi. Jika page tujuan ada pada dictionary belum dikunjungi, tambahkan page asal sebagai salah satu parent nodenya. Lakukan pengecekan penyelesaian pencarian. Pencarian selesai jika salah satu page terdapat di belum dikunjungi arah maju dan juga belum dikunjungi arah mundur. Selanjutnya, cari semua path terpendek

Untuk pencarian menggunakan Iterative Deepening Search, digunakan fungsi IDS. Fungsi ini menerima parameter page, target page, depth, daftar page yang sudah dikunjungi, dan path yang akan dibuat, serta mengembalikan boolean

apakah path valid ke target page ditemukan. Fungsi ini melakukan iterasi pada setiap depth sampai kedalaman 10 degree. Untuk setiap depth, akan dicari path ke target page, jika path valid ditemukan maka path akan dikembalikan. Pada setiap iterasi, path ditemukan menggunakan bantuan fungsi DLS untuk melakukan Depth Limited Search di setiap iterasi depth. Jika limit depth dicapai dan page yang sedang dicek bukan page target, kembalikan false. Jika page yang sedang dicek merupakan page target, tambahkan dalam path, kembalikan true. Jika page yang sedang dicek sudah dikunjungi, kembalikan false. Tandai page yang sedang dicek sebagai sudah dikunjungi, ambil page lain yang terhubung melalui link. Lakukan pemanggilan secara rekursif setiap page yang terhubung, dengan depth dikurangi setiap iterasi. Jika nilai true dikembalikan, berarti page target ditemukan.

4.2 Penjelasan tata cara penggunaan program

1. Mengetik perintah go run wikiread.go pathfinding.go pada cmd atau terminal pada direktori yang memiliki file wikiread.go
2. Setelah itu, buka web browser (dalam kasus ini, google chrome), ketik localhost:9999 dan website muncul
3. Mengisi titik mulai pencarian dan titik tujuan dari pencarian
4. Menekan tombol submit ketika sudah yakin apa yang mau dicari
5. Aplikasi memproses inputan yang sudah disubmit dan menunjukkan hasilnya.

4.3 Hasil pengujian

Hasil dari pengujian aplikasi adalah sebuah list dari rute perjalanan dari titik awal hingga titik akhir yang memiliki jumlah langkah yang sama. Selain itu, ditampilkan lama dari proses algoritma dari program aplikasi untuk mendapatkan jawaban yang diharapkan sesuai. Didapatkan bahwa algoritma baik BFS maupun IDS dapat bekerja baik dalam waktu yang cepat untuk degree 2 ke bawah. Sayangnya, program yang kami buat kurang teroptimasi sehingga kemungkinan memerlukan waktu yang lama untuk path yang membutuhkan lebih banyak langkah.

4.4 Analisis hasil pengujian

Analisa yang kelompok kami dapatkan adalah banyak sekali link yang dapat membuat program tidak dapat memberikan hasil yang sesuai. Perlu diperhatikan bagaimana struktur dari penulisan dalam wikipedia dan teliti dalam menentukan kata-kata apa yang tidak diperlukan sehingga program dapat berjalan lebih baik. Selain itu, perlu ditambahkan implementasi concurrency menggunakan goroutine untuk mempercepat waktu, merapikan website dan program input output, serta menambahkan lebih banyak error checking untuk memastikan website bekerja sempurna.

Bab 5: Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Antara algoritma BFS dan IDS, keduanya dapat berjalan baik tetapi hanya bekerja dalam waktu yang singkat untuk degree 2 ke bawah pada program yang kami buat. Hal ini dikarenakan program yang kami buat belum mengimplementasikan concurrency menggunakan goroutine atau package serupa.

5.2 Saran

Untuk saran, sebaiknya memikirkan dan melakukan *searching* terhadap beberapa website yang berada di dalam domain wikipedia dikarenakan ada kemungkinan isi dari artikel tidak sesuai dengan nama dari link. Selain itu, lakukan optimasi untuk mempersingkat waktu menggunakan concurrency, rapikan tampilan website, lalu tambahkan lebih banyak error checking.

5.3 Refleksi

Yang kelompok kami refleksikan dari pengerjaan tugas besar ini adalah untuk melakukan banyak percobaan untuk mencari kata - kata dalam link yang membuat program dalam aplikasi menjadi memberikan hasil yang tidak sesuai dengan harapan. Selain itu, agar dapat mempertimbangkan lebih banyak case dan meningkatkan manajemen waktu serta kerja sama.

Lampiran

Tautan *Repository GitHub* :

https://github.com/ChristopherBrian/Tubes2_TrialTanksUnity3D

Daftar Pustaka

• Materi kuliah IF3170 Inteligensi Buatan Teknik Informatika ITB, Course Website: http://kuliah.itb.ac.id → STEI → Teknik Informatika → IF3170

• Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice-Hall International, Inc, 2010, Textbook Site: http://aima.cs.berkeley.edu/ (2nd edition)

• Free online course materials | MIT OpenCourseWare Website: Site: http://ocw.mit.edu/courses/electrical-engineering-andcomputer-science/

• Free online | geeksforgeeks OpenCourseWare Website: Site: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/