

## **LAPORAN TUGAS KECIL 3**

### **IF2211 - Strategi Algoritma**

**“Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS, Greedy**

**Best First Search, dan A\*”**



#### **Dosen:**

Ir. Rila Mandala, M.Eng., Ph.D.

Monterico Adrian, S.T., M.T.

#### **Oleh:**

Ahmad Thoriq Saputra (13522141)

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**SEMESTER II TAHUN 2023/2024**

# DAFTAR ISI

|  |           |
|--|-----------|
| <b>DAFTAR ISI</b>  | <b>1</b>  |
| <b>BAB I</b>   |           |
| <b>DESKRIPSI TUGAS</b>   | <b>3</b>  |
| <b>BAB II</b>  |           |
| <b>LANDASAN TEORI</b>  | <b>5</b>  |
| 1. Algoritma Uniform Cost Search   | 5         |
| 2. Algoritma Greedy Best First Search  | 6         |
| 3. Algoritma A*  | 7         |
| <b>BAB III</b>   |           |
| <b>ANALISIS DAN IMPLEMENTASI DALAM ALGORITMA UCS, GREEDY BEST FIRST SEARCH, DAN A*</b>         | <b>9</b>  |
| 1. Implementasi Algoritma  | 9         |
| a. Uniform Cost Algorithm  | 9         |
| b.   | 10        |
| c. Greedy Best First Search  | 10        |
| d. A*  | 11        |
| 2. Definisi f(n) dan g(n)  | 12        |
| Uniform Cost Algorithm:  | 12        |
| Greedy Best First Search:  | 12        |
| A*:  | 12        |
| 3. Keterkaitan BFS dan UCS Pada Word Ladder Solver   | 13        |
| 4. Perbandingan Efisiensi Algoritma A* Dengan Algoritma UCS                                    | 13        |
| 5. Pertimbangan Keoptimalan Algoritma Greedy Best-First Search dalam Penyelesaian Word Ladder. | 13        |
| <b>BAB IV</b>  |           |
| <b>IMPLEMENTASI DAN PENGUJIAN</b>  | <b>15</b> |
| 1. Implementasi Pada Bahasa Java   | 15        |
| a. Class Function  | 15        |
| b. Class Ladder  | 16        |
| c. Class Solver  | 17        |
| • Metode ucsSolver   | 18        |
| • Metode gbfsSolver  | 20        |
| • Metode astarSolver   | 22        |
| 2. Pengujian   | 23        |
| a. Hit → Cog   | 23        |
| b. Pork → Lord   | 24        |
| c. Look → Horn   | 25        |
| d. Leans → Think   | 27        |
| e. Person → Ballon   | 28        |

|   |           |
|---|-----------|
| f. Junkies → Changed  | 30        |
| <b>BAB V</b>  |           |
| <b>ANALISIS HASIL</b>   | <b>32</b> |
| 1. Optimalitas Solusi:  | 32        |
| 2. Waktu Eksekusi:  | 32        |
| 3. Memori yang Dibutuhkan:  | 32        |
| <b>BAB VI</b>   |           |
| <b>IMPLEMENTASI BONUS</b>   | <b>34</b> |
| 1. Component input  | 34        |
| 2. Component hasil  | 37        |
| 3. Component about  | 38        |
| <b>BAB VII</b>  |           |
| <b>KESIMPULAN</b>   | <b>40</b> |
| <b>LAMPIRAN</b>   | <b>41</b> |
| Tautan Repository : <a href="https://github.com/thoriqsaputra/Tucil3_13522141">https://github.com/thoriqsaputra/Tucil3_13522141</a> | 41        |
| <b>DAFTAR PUSTAKA</b>   | <b>42</b> |

# BAB I

## DESKRIPSI TUGAS

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragraphs, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.

### How To Play

×

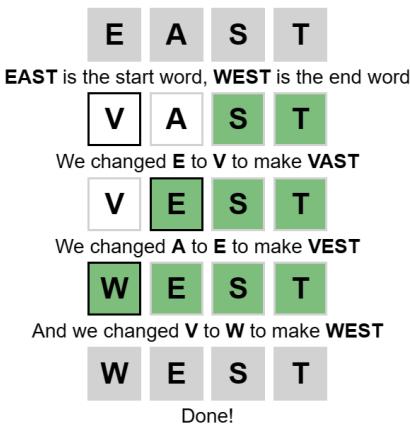
This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

#### Rules

Weave your way from the start word to the end word.

Each word you enter **can only change 1 letter** from the word above it.

#### Example



Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder

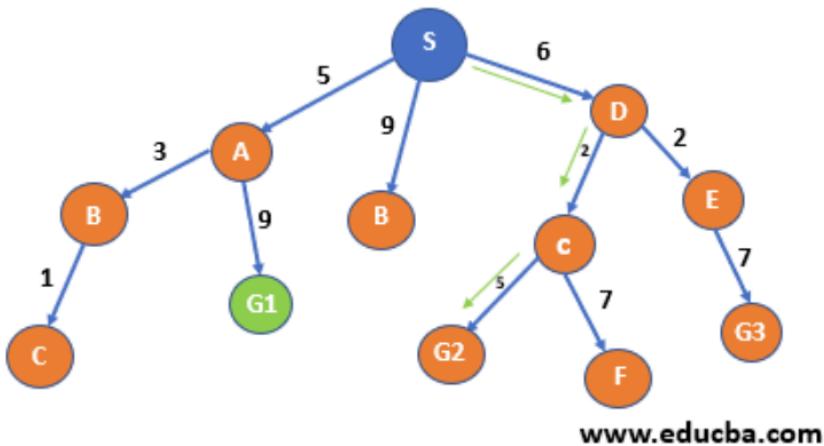
(Sumber: <https://wordwormdormdork.com/>)

Permainannya cukup sederhana bukan? Jika belum paham dengan peraturan permainannya, cobalah untuk memainkan permainannya pada link sumber di atas. Jika sudah paham dengan permainannya, sekarang adalah waktunya kalian untuk membuat sebuah solver permainan tersebut dengan harapan kita dapat menemukan solusi paling optimal untuk menyelesaikan permainan Word Ladder ini.

## BAB II

# LANDASAN TEORI

### 1. Algoritma *Uniform Cost Search*



Gambar 2. Ilustrasi Algoritma UCS

(Sumber: [Uniform Cost Search | Algorithm of Uniform Cost Search \(educba.com\)](#))

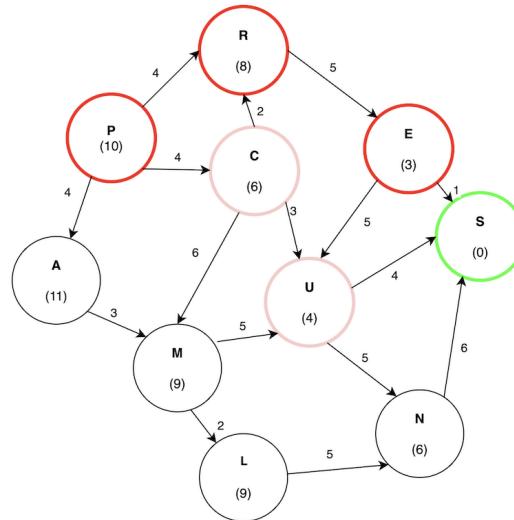
Uniform Cost Search (UCS) adalah algoritma pencarian tak terinformasi yang digunakan untuk menemukan jalur dengan biaya akumulasi terendah dalam sebuah graf berbobot. Algoritma ini beroperasi dengan cara yang sistematis dan tidak mempertimbangkan informasi tambahan tentang status node atau ruang pencarian. UCS berfokus pada pencarian jalur yang memiliki biaya total terendah dari node awal menuju salah satu node tujuan.

Salah satu keunggulan utama dari UCS adalah kemampuannya untuk menemukan jalur dengan biaya akumulasi terendah di dalam graf berbobot, di mana setiap edge memiliki biaya yang berbeda dari node awal hingga node tujuan. Dalam setiap langkah, UCS mempertimbangkan jalur dengan biaya terendah untuk dieksplorasi, sehingga sering dianggap sebagai solusi optimal.

Meskipun memiliki beberapa keunggulan, UCS juga memiliki beberapa kelemahan. Salah satunya adalah perlunya mempertahankan daftar terbuka (open list) yang terurut, karena prioritas dalam antrian prioritas harus dipertahankan. Selain itu, UCS membutuhkan penyimpanan yang besar secara eksponensial, terutama ketika digunakan dalam ruang pencarian yang besar. Selain itu, ada juga risiko bahwa algoritma ini dapat

terjebak dalam perulangan tak terbatas jika tidak diimplementasikan dengan benar. Oleh karena itu, perlu perhatian khusus dalam mengimplementasikan UCS untuk menghindari masalah seperti itu.

## 2. Algoritma *Greedy Best First Search*



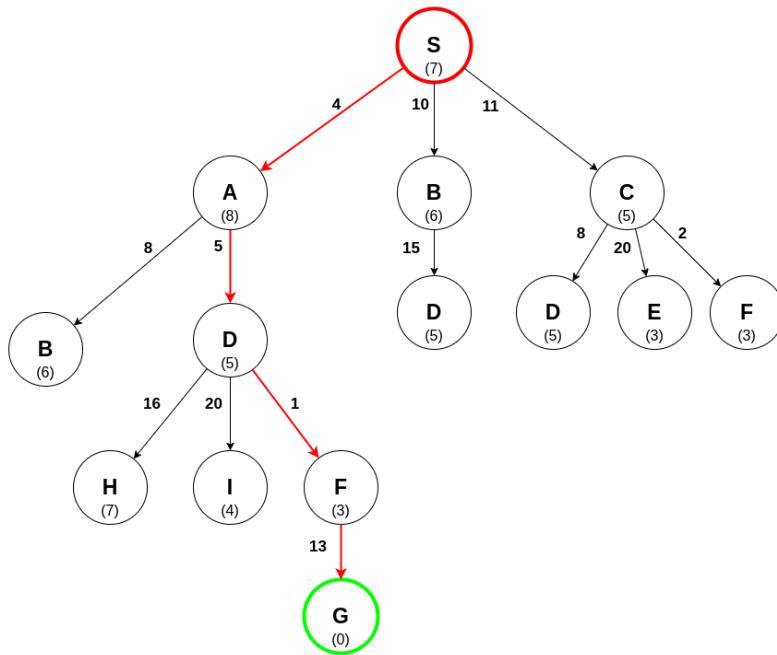
Gambar 3. Ilustrasi Algoritma GBFS  
(Sumber: [AI | Search Algorithms | Greedy Best-First Search | Codecademy](#))

Greedy Best-First Search (GBFS) adalah algoritma pencarian terinformasi di mana fungsi evaluasi sama dengan fungsi heuristik, tanpa memperhatikan bobot edge dalam graf berbobot karena hanya nilai heuristik yang dipertimbangkan. GBFS bertujuan untuk mencari node tujuan dengan cara memperluas node yang paling dekat dengan tujuan sesuai dengan nilai heuristik.

Pendekatan ini mengasumsikan bahwa mencari jalur yang paling dekat dengan tujuan akan lebih cepat menuju solusi. Namun, solusi dari GBFS mungkin tidak optimal karena jalur yang lebih pendek mungkin ada.

Dalam algoritma ini, biaya pencarian minimal karena solusi ditemukan tanpa memperluas node yang tidak berada pada jalur solusi. Algoritma ini minimal, namun tidak lengkap, karena dapat mengarah ke jalan buntu. Dinamakan "Greedy" karena pada setiap langkahnya mencoba untuk mendekati tujuan sebanyak mungkin.

### 3. Algoritma A\*



Gambar 4. Ilustrasi Algoritma A\*  
(Sumber: [AI | Search Algorithms | A\\* Search | Codecademy](#))

A\* Search merupakan algoritma pencarian terbaik berbasis informasi yang secara efisien menentukan jalur dengan biaya terendah antara dua node dalam sebuah graf berbobot berarah dengan bobot edge non-negatif. Algoritma ini merupakan varian dari algoritma Dijkstra. Perbedaan kecil terletak pada penggunaan fungsi evaluasi untuk menentukan node mana yang akan dieksplorasi selanjutnya.

Dalam A\* Search, evaluasi fungsi mencakup dua komponen utama:  $g(n)$  yang merupakan biaya sejauh ini dari node awal ke node tertentu, dan  $h(n)$  yang merupakan estimasi biaya dari node tertentu ke node tujuan. Fungsi evaluasi  $f(n)$  adalah jumlah dari kedua komponen tersebut,  $f(n) = g(n) + h(n)$ . Algoritma ini secara heuristik menggabungkan biaya aktual dari node yang telah dieksplorasi ( $g(n)$ ) dengan estimasi biaya ke node tujuan yang belum diketahui ( $h(n)$ ). Dengan demikian, A\* Search dapat menemukan jalur dengan biaya terendah secara efisien, karena secara cerdas memilih node yang memiliki perkiraan biaya terendah menuju tujuan.

Salah satu keuntungan utama dari A\* Search adalah kemampuannya untuk menghasilkan solusi optimal, asalkan fungsi heuristik  $h(n)$  memenuhi syarat tertentu, seperti admissibility (nilai  $h(n)$  tidak pernah melebihi biaya sebenarnya untuk mencapai tujuan) dan konsistensi (nilai  $h(n)$  tidak lebih besar dari perbedaan biaya antara dua node yang berdekatan). Namun, perlu

diperhatikan bahwa keefektifan A\* Search sangat tergantung pada kualitas fungsi heuristik yang digunakan.

# BAB III

## ANALISIS DAN IMPLEMENTASI

### ALGORITMA UCS, GREEDY BEST FIRST SEARCH, DAN A\*

#### 1. Implementasi Algoritma

Ketiga algoritma tersebut menggunakan sebuah kelas node yang berfungsi sebagai wadah untuk menyimpan informasi tentang kata, node induk, dan biaya suatu node. Kelas ini dirancang untuk mengatur jalur kata serta biaya dari node induk. Dalam konteks ini, node menjadi entitas penting yang membantu dalam menyusun dan melacak jalur yang diambil oleh algoritma. Dengan menggunakan kelas node, setiap simpul dalam algoritma dapat menyimpan informasi yang diperlukan, seperti kata yang diasosiasikan dengan simpul tersebut, node orang tua yang mengarah ke simpul saat ini, dan biaya yang diperlukan untuk mencapai simpul tersebut dari simpul orang tua. Penggunaan kelas node ini mempermudah manipulasi dan pengaturan data dalam algoritma, sehingga memungkinkan algoritma untuk bekerja secara efisien dalam mencari solusi yang optimal atau memenuhi kriteria tertentu sesuai dengan kebutuhan aplikasinya.

##### a. *Uniform Cost Algorithm*

Setelah mendeklarasikan priority queue, langkah pertama adalah membuat sebuah node baru dengan kata awal (startWord), parent null, dan cost 0. Priority queue digunakan agar proses mendapatkan node dengan nilai terkecil dapat dilakukan dengan mudah.

Selanjutnya, diperlukan deklarasi sebuah hash set yang diberi nama visited untuk menyimpan kata-kata yang telah dilewati. Hal ini penting untuk mencegah pengulangan kata yang sama dalam jalur pencarian.

Setelah deklarasi hash set dan priority queue, langkah selanjutnya adalah memasuki sebuah loop while dengan kondisi queue tidak kosong. Jika queue kosong, artinya jalur menuju endWord tidak dapat ditemukan.

Dalam setiap iterasi loop while, sebuah node diambil dari queue menggunakan fungsi poll(). Jika node tersebut sama dengan endWord, maka jalur dari startWord menuju endWord dapat dikembalikan.

Namun, jika node tersebut tidak sama dengan endWord, langkah selanjutnya adalah melakukan penggantian setiap karakter dalam kata pada node dengan karakter lainnya. Strategi ini lebih efisien daripada mencari kata dengan hanya satu perbedaan karakter dan mencocokkannya dengan kata-kata dalam kamus.

Jika hasil dari penggantian karakter adalah sebuah kata yang valid dalam kamus bahasa Inggris dan belum pernah dilewati sebelumnya (visited), maka akan dibuat node baru dengan kata tersebut dengan nilai cost dari penambahan satu dan nilai cost dari parent node. Selanjutnya, kata tersebut ditambahkan ke dalam set visited dan ditambahkan juga node pada queue.

Proses ini diulangi sampai node yang dihasilkan adalah endWord atau queue menjadi kosong.

**b.**

**c. Greedy Best First Search**

Setelah mendeklarasikan priority queue, langkah pertama adalah membuat sebuah node baru dengan kata awal (startWord), parent null, dan cost 0. Priority queue digunakan agar proses mendapatkan node dengan nilai terkecil dapat dilakukan dengan mudah.

Selanjutnya, diperlukan deklarasi sebuah hash set yang diberi nama visited untuk menyimpan kata-kata yang telah dilewati. Hal ini penting untuk mencegah pengulangan kata yang sama dalam jalur pencarian.

Setelah deklarasi hash set dan priority queue, langkah selanjutnya adalah memasuki sebuah loop while dengan kondisi queue tidak kosong. Jika queue kosong, artinya jalur menuju endWord tidak dapat ditemukan.

Dalam setiap iterasi loop while, sebuah node diambil dari queue menggunakan fungsi poll(). Jika node tersebut sama dengan endWord, maka jalur dari startWord menuju endWord dapat dikembalikan. Untuk mengatasi program melakukan backtrack pada algoritma ini, isi dari queue dihapuskan.

Namun, jika node tersebut tidak sama dengan endWord, langkah selanjutnya adalah melakukan penggantian setiap karakter dalam kata pada node dengan karakter lainnya. Strategi ini lebih efisien daripada mencari kata dengan hanya satu perbedaan karakter dan mencocokkannya dengan kata-kata dalam kamus.

Jika hasil dari penggantian karakter adalah sebuah kata yang valid dalam kamus bahasa Inggris dan belum pernah dilewati sebelumnya (visited), maka akan dibuat node dengan kata tersebut dan nilai cost dari penambahan satu dan nilai heuristic. Terdapat fungsi heuristic yang akan menghasilkan nilai yang lebih besar jika kata pada current node jauh daripada endWord. Selain menambahkan nilai cost dari parent node, akan ditambahkan nilai heuristic node tersebut. Terdapat fungsi heuristic yang akan menghasilkan nilai yang lebih besar jika kata pada current node jauh daripada endWord. Selanjutnya, kata tersebut ditambahkan ke dalam set visited dan ditambahkan juga node pada queue.

Proses ini diulangi sampai node yang dihasilkan adalah endWord atau queue menjadi kosong.

#### d. A\*

Setelah mendeklarasikan priority queue, langkah pertama adalah membuat sebuah node baru dengan kata awal (startWord), parent null, dan cost 0. Priority queue digunakan agar proses mendapatkan node dengan nilai terkecil dapat dilakukan dengan mudah.

Selanjutnya, diperlukan deklarasi sebuah hash set yang diberi nama visited untuk menyimpan kata-kata yang telah dilewati. Hal ini penting untuk mencegah pengulangan kata yang sama dalam jalur pencarian.

Setelah deklarasi hash set dan priority queue, langkah selanjutnya adalah memasuki sebuah loop while dengan kondisi queue tidak kosong. Jika queue kosong, artinya jalur menuju endWord tidak dapat ditemukan.

Dalam setiap iterasi loop while, sebuah node diambil dari queue menggunakan fungsi poll(). Jika node tersebut sama dengan endWord, maka jalur dari startWord menuju endWord dapat dikembalikan.

Namun, jika node tersebut tidak sama dengan endWord, langkah selanjutnya adalah melakukan penggantian setiap karakter dalam kata pada node dengan karakter lainnya. Strategi ini lebih efisien daripada mencari kata dengan hanya satu perbedaan karakter dan mencocokkannya dengan kata-kata dalam kamus.

Jika hasil dari penggantian karakter adalah sebuah kata yang valid dalam kamus bahasa Inggris dan belum pernah dilewati sebelumnya (visited), maka akan dibuat node

baru dengan kata tersebut dan nilai cost dari penambahan satu dan nilai cost dari parent node. Selain menambahkan nilai cost dari parent node, akan ditambahkan nilai heuristic node tersebut. Terdapat fungsi heuristic yang akan menghasilkan nilai yang lebih besar jika kata pada current node jauh daripada endWord. Selanjutnya, kata tersebut ditambahkan ke dalam set visited dan ditambahkan juga node pada queue.

Proses ini diulangi sampai node yang dihasilkan adalah endWord atau queue menjadi kosong.

## 2. Definisi f(n) dan g(n)

### Uniform Cost Algorithm:

- $f(n)$ : Tidak digunakan dalam UCS. UCS hanya memperhatikan biaya aktual ( $g(n)$ ) untuk mencapai simpul n. Jadi,  $f(n)$  dapat dianggap tidak relevan dalam konteks UCS.
- $g(n)$ : Nilai biaya aktual atau estimasi untuk mencapai simpul n dari titik awal.

### Greedy Best First Search:

- $f(n)$ : Tidak digunakan dalam GBFS. GBFS hanya memperhatikan nilai heuristik ( $h(n)$ ) sebagai tujuan utama dalam pemilihan simpul berikutnya. Sehingga,  $f(n)$  juga dianggap tidak relevan dalam GBFS.
- $g(n)$ : Tidak digunakan dalam GBFS. GBFS tidak mempertimbangkan estimasi biaya ( $g(n)$ ) untuk mencapai simpul n.
- $h(n)$ : Nilai heuristik yang mengestimasi biaya yang tersisa untuk mencapai tujuan dari simpul n. Digunakan sebagai satu-satunya faktor dalam pemilihan simpul berikutnya

### A\*:

- $f(n)$ : Nilai total dari simpul n. Biasanya didefinisikan sebagai jumlah estimasi mencapai simpul n ( $g(n)$ ) dan estimasi biaya yang tersisa untuk mencapai tujuan ( $h(n)$ ). Formalnya,  $f(n) = g(n) + h(n)$ .
- $g(n)$ : Nilai biaya aktual atau estimasi untuk mencapai simpul n dari titik awal.
- $h(n)$ : Nilai heuristik yang mengestimasi biaya yang tersisa untuk mencapai tujuan dari simpul n.

### **3. Keterkaitan BFS dan UCS Pada *Word Ladder Solver***

Pada kasus Word Ladder, meskipun algoritma UCS (Uniform Cost Search) dan BFS (Breadth-First Search) memiliki beberapa kesamaan dalam pendekatan pencarian, mereka tidak akan secara konsisten menghasilkan urutan node yang sama dan path yang identik. Meskipun keduanya mempertimbangkan simpul-simpul pada kedalaman yang sama terlebih dahulu, strategi dasar yang berbeda menyebabkan hasil yang berbeda.

Walaupun dalam beberapa kasus keduanya mungkin menghasilkan solusi yang sama, namun secara umum, mereka tidak selalu dapat diandalkan untuk memberikan urutan node dan path yang sama dalam pencarian Word Ladder.

### **4. Perbandingan Efisiensi Algoritma A\* Dengan Algoritma UCS**

Algoritma A\* biasanya lebih efisien daripada algoritma UCS pada kasus Word Ladder. Ini dikarenakan A\* menggunakan heuristik ( $h(n)$ ) untuk memberikan perkiraan biaya yang tersisa untuk mencapai tujuan dari setiap simpul, sedangkan UCS hanya mempertimbangkan biaya sejauh ini ( $g(n)$ ).

Dalam Word Ladders, di mana setiap langkah hanya memerlukan satu perubahan karakter, heuristik yang sederhana seperti jumlah perbedaan huruf antara kata saat ini dan kata tujuan seringkali cukup efektif. Dengan memanfaatkan heuristik, A\* mampu mengevaluasi jalur menuju tujuan secara lebih baik, memungkinkannya mengeksplorasi lebih banyak jalur yang berpotensi mengarah ke solusi dengan lebih cepat daripada UCS.

### **5. Pertimbangan Keoptimalan Algoritma Greedy Best-First Search dalam Penyelesaian Word Ladder.**

Algoritma *Greedy Best-First Search* (GBFS) tidak menjamin solusi optimal untuk persoalan Word Ladder. Hal ini dikarenakan GBFS hanya memperhatikan nilai heuristik ( $h(n)$ ) dalam pemilihan simpul berikutnya tanpa mempertimbangkan biaya sejauh ini ( $g(n)$ ) untuk mencapai simpul tersebut. Sebagai hasilnya, GBFS cenderung menuju simpul yang dianggap paling dekat dengan tujuan berdasarkan nilai heuristiknya tanpa memperhitungkan biaya yang sudah ditempuh, yang bisa saja mengakibatkan jalur yang ditemukan tidak optimal.

Dalam konteks Word Ladder, di mana tujuan adalah untuk mengubah satu kata menjadi kata lainnya dengan mengubah satu karakter pada setiap langkahnya, GBFS dapat menemukan solusi, tetapi solusi yang ditemukan mungkin tidak optimal karena algoritma tidak mempertimbangkan biaya sejauh ini ( $g(n)$ ). Ini bisa menyebabkan algoritma GBFS terjebak dalam mencari jalur yang terlalu jauh dari solusi optimal, terutama jika ada jalur yang lebih pendek tetapi memiliki nilai heuristik yang lebih rendah.

Jadi, walaupun GBFS dapat menemukan solusi untuk Word Ladder, tidak ada jaminan bahwa solusi yang ditemukan akan optimal.

# BAB IV

## IMPLEMENTASI DAN PENGUJIAN

### 1. Implementasi Pada Bahasa Java

#### a. Class Function

```
● ● ●

public class function {
    // kamus kata yang valid
    private static final String DICTIONARY_FILE = "/engDic.txt";

    public static Set<String> getValidWords() {
        // Menyimpan kata-kata yang valid pada set
        Set<String> validWords = new HashSet<>();
        // Membaca file kamus kata
        // Menggunakan getResourceAsStream untuk membaca file dari resources
        try (InputStream inputStream = function.class.getResourceAsStream(DICTIONARY_FILE);
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream))) {
            String line;
            // Membaca file baris per baris yang kemudian dimasukkan ke dalam set
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split("\n");
                if (parts.length > 0) {
                    validWords.add(parts[0].toLowerCase().trim());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Gambar 5. Snippet Class Function

Kode ini digunakan untuk mengambil dan menyimpan kumpulan kata-kata bahasa Inggris yang valid dari sebuah file kamus, yang kemungkinan besar akan digunakan sebagai referensi dalam menyelesaikan Word Ladder Problem. Dengan kumpulan kata-kata yang valid ini, kita dapat memeriksa apakah suatu kata termasuk dalam kamus kata yang diizinkan, serta menemukan langkah-langkah yang diperlukan untuk mengubah satu kata menjadi kata lainnya dalam Word Ladder. Dengan kata lain, kumpulan kata-kata ini berperan penting dalam menyediakan referensi yang diperlukan untuk menyelesaikan permasalahan Word Ladder dengan efisien.

## b. Class Ladder

```
● ● ●

public class Ladder {
    private List<String> ladder;
    private int visitedWords;

    public Ladder(List<String> ladder, int visitedWords)
    {
        this.ladder = ladder;
        this.visitedWords = visitedWords;
    }

    public List<String> getLadder() {
        return ladder;
    }

    public int getVisitedWords() {
        return visitedWords;
    }
}
```

Gambar 6. Snippet Class Ladder

Kelas Ladder berperan sebagai wadah untuk menyimpan hasil dari metode algoritma UCS, A\*, dan GBFS dalam menyelesaikan Word Ladder Problem. Dengan mengandung atribut ladder, yang merupakan daftar kata-kata dalam jalur dari kata awal hingga kata tujuan, serta atribut visitedWords yang menyimpan jumlah kata yang telah dikunjungi dalam proses pencarian, kelas ini menyediakan struktur yang efisien untuk merepresentasikan solusi dari algoritma-algoritma tersebut.

### c. Class Solver

```
public class Solver {  
  
    private static class Node implements Comparable<Node> {  
        String word;  
        Node parent;  
        int cost;  
  
        public Node(String word, Node parent, int cost) {  
            this.word = word;  
            this.parent = parent;  
            this.cost = cost;  
        }  
  
        @Override  
        public int compareTo(Node other) {  
            return Integer.compare(cost, other.cost);  
        }  
    }  
  
    static int heuristic(String word, String endWord){  
        int count = 0;  
        for (int i = 0; i < word.length(); i++) {  
            if (word.charAt(i) != endWord.charAt(i)) {  
                count++;  
            }  
        }  
        return count;  
    }  
}
```

Gambar 7. Snippet Class Solver dan Class Node

Kelas Solver merupakan kelas yang bertanggung jawab untuk menyelesaikan permasalahan Word Ladder dengan memanfaatkan algoritma A\*, GBFS, dan UCS. Kelas ini menggunakan inner class Node untuk menyimpan informasi tentang sebuah kata, node parent, dan biaya node tersebut. Selain itu, terdapat sebuah metode heuristic() yang digunakan untuk menghitung nilai heuristik dari sebuah kata terkait kata tujuan. Metode ini membandingkan setiap karakter dari kata dengan kata tujuan dan menghitung jumlah karakter yang berbeda, yang kemudian digunakan sebagai perkiraan biaya tersisa untuk mencapai kata tujuan. Dengan demikian, kelas Solver menyediakan struktur dan fungsi yang diperlukan untuk mengimplementasikan algoritma pencarian yang diperlukan untuk menyelesaikan Word Ladder Problem.

- Metode ucsSolver

```
public static Ladder ucsSolver(String startWord, String endWord, Set<String> validWords) {  
    // Priority queue untuk menyimpan node yang akan di proses  
    PriorityQueue<Node> queue = new PriorityQueue<>();  
    // Menambahkan node dengan startWord ke queue  
    queue.add(new Node(startWord, null, 0));  
  
    // Set untuk menyimpan kata yang sudah di lewati  
    Set<String> visited = new HashSet<>();  
    // Menambahkan startWord ke visited  
    visited.add(startWord);  
  
    // Loop sampai queue kosong atau ditemukan endWord  
    while (!queue.isEmpty()) {  
        // Mengambil node dengan cost terkecil  
        Node current = queue.poll();  
        // Mengambil kata dari node  
        String word = current.word;  
  
        // Jika kata sama dengan endWord  
        if (word.equals(endWord)) {  
            // Membuat list untuk menyimpan ladder  
            List<String> ladder = new ArrayList<>();  
            // Mengambil node parent dari node current  
            Node node = current;  
            // Loop sampai node null dan menambahkan kata ke ladder  
            while (node != null) {  
                ladder.add(0, node.word);  
                node = node.parent;  
            }  
            // Mengembalikan ladder berupa list kata dan jumlah kata yang di lewati  
            return new Ladder(ladder, visited.size());  
        }  
  
        // Proses perubahan kata  
        for (int i = 0; i < word.length(); i++) {  
            // Mengubah kata dengan mengganti karakter ke-i dengan a-z  
            char[] wordArray = word.toCharArray();  
            for (char c = 'a'; c <= 'z'; c++) {  
                wordArray[i] = c;  
                String nextWord = new String(wordArray);  
                // Jika kata valid dan belum di lewati maka tambahkan ke queue dengan  
                parent cost + 1  
                // Tambahkan juga ke visited  
                if (validWords.contains(nextWord) && !visited.contains(nextWord)) {  
                    int cost = current.cost + 1;  
                    queue.add(new Node(nextWord, current, cost));  
                    visited.add(nextWord);  
                }  
            }  
        }  
    }  
    // Jika tidak ditemukan ladder maka kembalikan ladder kosong  
    return new Ladder(Collections.emptyList(), visited.size());  
}
```

#### Gambar 8. snippet metode ucsSolver

Metode ucsSolver adalah sebuah fungsi yang bertugas untuk menghasilkan solusi dalam Word Ladder menggunakan algoritma UCS (Uniform Cost Search). Algoritma ini bekerja dengan mengeksplorasi simpul-simpul secara bertahap, dimulai dari simpul awal, dan memilih simpul dengan biaya terendah untuk dieksplorasi berikutnya. Fungsi ini menerima argumen berupa kata awal, kata akhir, serta himpunan kata-kata yang valid. Dengan memanfaatkan struktur data Priority Queue untuk mengatur simpul-simpul yang akan dieksplorasi, serta struktur kelas Node untuk merepresentasikan setiap simpul dengan informasi tentang kata, node induk, dan biaya, fungsi ini memungkinkan pencarian jalur yang optimal dalam Word Ladder. Proses pencarian dilakukan dengan mengevaluasi setiap kemungkinan perubahan satu karakter pada setiap kata dalam himpunan valid, dan memilih jalur dengan biaya terendah yang memenuhi syarat untuk mencapai kata akhir.

- Metode gbfsSolver

```
public static Ladder gbfsSolver(String startWord, String endWord, Set<String> validWords) {  
    // Priority queue untuk menyimpan node yang akan di proses  
    PriorityQueue<Node> queue = new PriorityQueue<>();  
    // Set untuk menyimpan kata yang sudah di lewati  
    Set<String> visited = new HashSet<>();  
    // Menambahkan node dengan startWord ke queue  
    queue.add(new Node(startWord, null, 0));  
    visited.add(startWord);  
  
    Node currentNode = null;  
  
    // Loop sampai queue kosong atau ditemukan endWord  
    while (!queue.isEmpty()) {  
        // Mengambil node dengan cost terkecil  
        currentNode = queue.poll();  
        String word = currentNode.word;  
        // queue di clear untuk tidak mencegah backtracking  
        queue.clear();  
        // Jika kata sama dengan endWord  
        if (word.equals(endWord)) {  
            // Membuat list untuk menyimpan ladder  
            List<String> ladder = new ArrayList<>();  
            Node node = currentNode;  
            // Loop sampai node null dan menambahkan kata ke ladder  
            while (node != null) {  
                ladder.add(0, node.word);  
                node = node.parent;  
            }  
            // Mengembalikan ladder berupa list kata dan jumlah kata yang di lewati  
            return new Ladder(ladder, visited.size());  
        }  
  
        // Proses perubahan kata  
        for (int i = 0; i < word.length(); i++) {  
            char[] wordArray = word.toCharArray();  
            for (char c = 'a'; c <= 'z'; c++) {  
                wordArray[i] = c;  
                String nextWord = new String(wordArray);  
                // Jika kata valid dan belum di lewati maka tambahkan ke queue dengan  
                // cost berupa 1 + heuristic  
                // Tambahkan juga ke visited  
                if (validWords.contains(nextWord) && visited.add(nextWord)) {  
                    int cost = heuristic(nextWord, endWord) + 1;  
                    queue.add(new Node(nextWord, currentNode, cost));  
                    visited.add(nextWord);  
                }  
            }  
        }  
    }  
    // Jika tidak ditemukan ladder maka kembalikan ladder kosong  
    return new Ladder(Collections.emptyList(), visited.size());  
}
```

Gambar 9. snippet metode gbfsSolver

Metode gbfsSolver merupakan sebuah fungsi yang bertujuan untuk menyelesaikan Word Ladder menggunakan algoritma GBFS (Greedy Best-First Search). Algoritma ini berfokus pada pemilihan simpul berikutnya berdasarkan nilai heuristik yang menunjukkan seberapa dekat simpul tersebut dengan tujuan akhir. Dalam fungsi ini, setiap kata dalam kamus kata yang valid dievaluasi berdasarkan heuristik yang dihasilkan oleh metode heuristic(). Selanjutnya, kata-kata dieksplorasi secara berurutan berdasarkan nilai heuristiknya, dengan harapan dapat menemukan jalur yang optimal menuju kata akhir. Fungsi ini menerima argumen berupa kata awal, kata akhir, serta himpunan kata-kata yang valid. Dengan menggunakan struktur data Priority Queue untuk mengatur simpul-simpul yang akan dieksplorasi, serta struktur kelas Node untuk merepresentasikan setiap simpul dengan informasi tentang kata, node parent, dan cost.

- Metode astarSolver

```
public static Ladder aStarSolver(String startWord, String endWord, Set<String> validWords) {  
    // Priority queue untuk menyimpan node yang akan di proses  
    PriorityQueue<Node> queue = new PriorityQueue<>();  
    // Set untuk menyimpan kata yang sudah di lewati  
    Set<String> visited = new HashSet<>();  
    // Menambahkan node dengan startWord ke queue  
    queue.add(new Node(startWord, null, 0));  
    visited.add(startWord);  
  
    // Loop sampai queue kosong atau ditemukan endWord  
    while (!queue.isEmpty()) {  
        // Mengambil node dengan cost terkecil  
        Node currentNode = queue.poll();  
        String word = currentNode.word;  
  
        // Jika kata sama dengan endWord  
        if (word.equals(endWord)) {  
            // Membuat list untuk menyimpan ladder  
            List<String> ladder = new ArrayList<>();  
            Node node = currentNode;  
            // Loop sampai node null dan menambahkan kata ke ladder  
            while (node != null) {  
                ladder.add(0, node.word);  
                node = node.parent;  
            }  
            // Mengembalikan ladder berupa list kata dan jumlah kata yang di lewati  
            return new Ladder(ladder, visited.size());  
        }  
  
        // Proses perubahan kata  
        for (int i = 0; i < word.length(); i++) {  
            char[] wordArray = word.toCharArray();  
            // Mengubah kata dengan mengganti karakter ke-i dengan a-z  
            for (char c = 'a'; c <= 'z'; c++) {  
                wordArray[i] = c;  
                String nextWord = new String(wordArray);  
                // Jika kata valid dan belum di lewati maka tambahkan ke queue dengan  
                // cost berupa cost parent + 1 + heuristic  
                if (validWords.contains(nextWord) && visited.add(nextWord)) {  
                    int cost = currentNode.cost + 1 + heuristic(nextWord, endWord);  
                    queue.add(new Node(nextWord, currentNode, cost));  
                    visited.add(nextWord);  
                }  
            }  
        }  
    }  
    // Jika tidak ditemukan ladder maka kembalikan ladder kosong  
    return new Ladder(Collections.emptyList(), visited.size());  
}
```

Gambar 10. snippet metode astarSolver

Metode astarSolver adalah fungsi yang menyelesaikan Word Ladder dengan algoritma A\*. Algoritma ini kombinasikan pencarian biaya (uniform cost search) dengan pencarian

heuristik yang memperkirakan jarak ke tujuan. Setiap kata dievaluasi berdasarkan heuristik dari heuristic(), yang memperkirakan biaya tersisa. Kata-kata dieksplorasi berdasarkan nilai total biaya dan heuristiknya. Dengan menggunakan Priority Queue untuk mengatur simpul-simpul dan kelas Node untuk merepresentasikan simpul dengan informasi kata, node parent, dan cost, astarSolver memberikan solusi efisien untuk Word Ladder.

## 2. Pengujian

### a. Hit → Cog

StartWord = Hit

EndWord = Cog

| Algoritma | Hasil   |
|-----------|---|
| UCS       | <p><b>Ladder has been found!</b></p> <p><b>Took 5.5816 milliseconds</b></p> <p><b>848 words visited</b></p> |
| GBFS      | <p><b>Ladder has been found!</b></p> <p><b>Took 0.105 milliseconds</b></p> <p><b>62 words visited</b></p>   |

|    |  |
|----|--|
| A* | <p><b>Ladder has been found!</b></p> <p><b>Took 0.462 milliseconds</b></p> <p><b>338 words visited</b></p> |
|----|--|

### b. Pork → Lord

StartWord = Pork  
EndWord = Lord

| Algoritma | Hasil  |
|-----------|--|
| UCS       | <p><b>Ladder has been found!</b></p> <p><b>Took 6.9168 milliseconds</b></p> <p><b>1870 words visited</b></p> |

|      |   |
|------|---|
| GBFS | <p><b>Ladder has been found!</b></p> <p><b>Took 0.128 milliseconds</b></p> <p><b>29 words visited</b></p>   |
| A*   | <p><b>Ladder has been found!</b></p> <p><b>Took 0.4403 milliseconds</b></p> <p><b>269 words visited</b></p> |

### c. Look → Horn

StartWord = Look

EndWord = Horn

| Algoritma | Hasil |
|-----------|-------|
|           |       |

UCS

**Ladder has been found!**

**Took 4.2354 milliseconds**

**1340 words visited**

1 look 2 loon 3 lorn 4 horn

GBFS

**Ladder has been found!**

**Took 0.2471 milliseconds**

**84 words visited**

1 look 2 hook 3 hock 4 bock 5 bonk 6 conk  
7 cork 8 corn 9 horn

A\*

**Ladder has been found!**

**Took 0.3528 milliseconds**

**168 words visited**

1 look 2 loon 3 lorn 4 horn

d. Leans → Think

StartWord = Hit

EndWord = Cog

| Algoritma | Hasil  |
|-----------|--|
| UCS       | <p><b>Ladder has been found!</b></p> <p><b>Took 5.5816 milliseconds</b></p> <p><b>848 words visited</b></p> <p>1 hit 2 hot 3 cot 4 cog</p> |

GBFS

**Ladder has been found!**

**Took 0.105 milliseconds**

**62 words visited**



1 hit 2 hot 3 cot 4 cog

A\*

**Ladder has been found!**

**Took 0.462 milliseconds**

**338 words visited**



1 hit 2 hot 3 hog 4 cog

e. Person → Ballon

StartWord = Person

EndWord = Ballon

|           |       |
|-----------|-------|
| Algoritma | Hasil |
|-----------|-------|

UCS

**Ladder has been found!**

**Took 83.7829 milliseconds**

**5482 words visited**

|    |        |    |        |   |        |    |        |    |        |    |        |
|----|--------|----|--------|---|--------|----|--------|----|--------|----|--------|
| 1  | person | 2  | parson | 3 | parton | 4  | carton | 5  | canton | 6  | cantor |
| 7  | canter | 8  | banter | 9 | bander | 10 | balder | 11 | baller | 12 | ballet |
| 13 | ballot | 14 | ballon |   |        |    |        |    |        |    |        |

GBFS

):

**Took 0.226 milliseconds**

**7 words visited**

**No ladder found**



|    |   |
|----|---|
| A* | <p><b>Ladder has been found!</b></p> <p><b>Took 12.4507 milliseconds</b></p> <p><b>1885 words visited</b></p> |
|----|---|

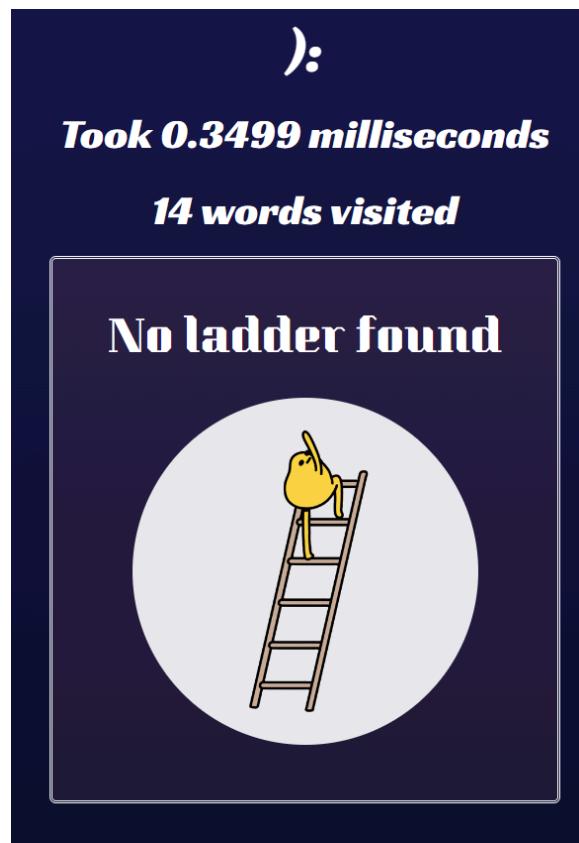
#### f. Junkies → Changed

StartWord = Junkies

EndWord = Changed

| Algoritma | Hasil   |
|-----------|---|
| UCS       | <p><b>Ladder has been found!</b></p> <p><b>Took 90.6387 milliseconds</b></p> <p><b>4572 words visited</b></p> |

GBFS



A\*



## **BAB V**

# **ANALISIS HASIL**

Merujuk pada hasil pengujian pada subbab Look → Horn didapatkan hasil sebagai berikut:

### 1. Optimalitas Solusi:

- UCS dan A\* menunjukkan solusi dengan panjang path yang sama, yaitu 4, yang menunjukkan bahwa keduanya mampu menemukan solusi optimal dalam kasus ini.
- Namun, GBFS menghasilkan solusi dengan panjang path yang lebih besar, yaitu 9. Hal ini menunjukkan bahwa GBFS tidak menjamin solusi optimal dalam beberapa kasus, terutama jika heuristik yang digunakan tidak memperkirakan biaya dengan akurat. Dalam beberapa kasus, GBFS juga tidak dapat menghasilkan solusi sama sekali, yang dapat terjadi jika heuristik yang digunakan tidak memperkirakan biaya dengan akurat atau jika algoritma terjebak dalam keadaan lokal yang tidak mengarah ke solusi.

### 2. Waktu Eksekusi:

- UCS memiliki waktu eksekusi sebesar 4.23 ms, yang mungkin disebabkan oleh pencarian node secara bertahap dengan mempertimbangkan biaya yang terus bertambah.
- GBFS memiliki waktu eksekusi tercepat, hanya 0.2481 ms. Hal ini karena GBFS hanya mempertimbangkan nilai heuristik dalam pemilihan simpul berikutnya, tanpa memperhitungkan biaya sejauh ini.
- A\* memiliki waktu eksekusi yang sedikit lebih lambat dibandingkan GBFS, yaitu 0.3528 ms. Meskipun demikian, A\* menggabungkan kedua pendekatan UCS dan GBFS, sehingga biasanya lebih efisien dalam mencari solusi optimal.

### 3. Memori yang Dibutuhkan:

- UCS membutuhkan lebih banyak memori karena perlu menyimpan informasi tentang setiap node yang dieksplorasi, termasuk biaya sejauh ini.
- GBFS membutuhkan lebih sedikit memori karena hanya perlu menyimpan informasi tentang simpul-simpul yang belum dieksplorasi.

- A\* juga membutuhkan memori yang cukup besar karena perlu menyimpan informasi tentang setiap simpul yang dieksplorasi, serta nilai heuristik dari masing-masing simpul.

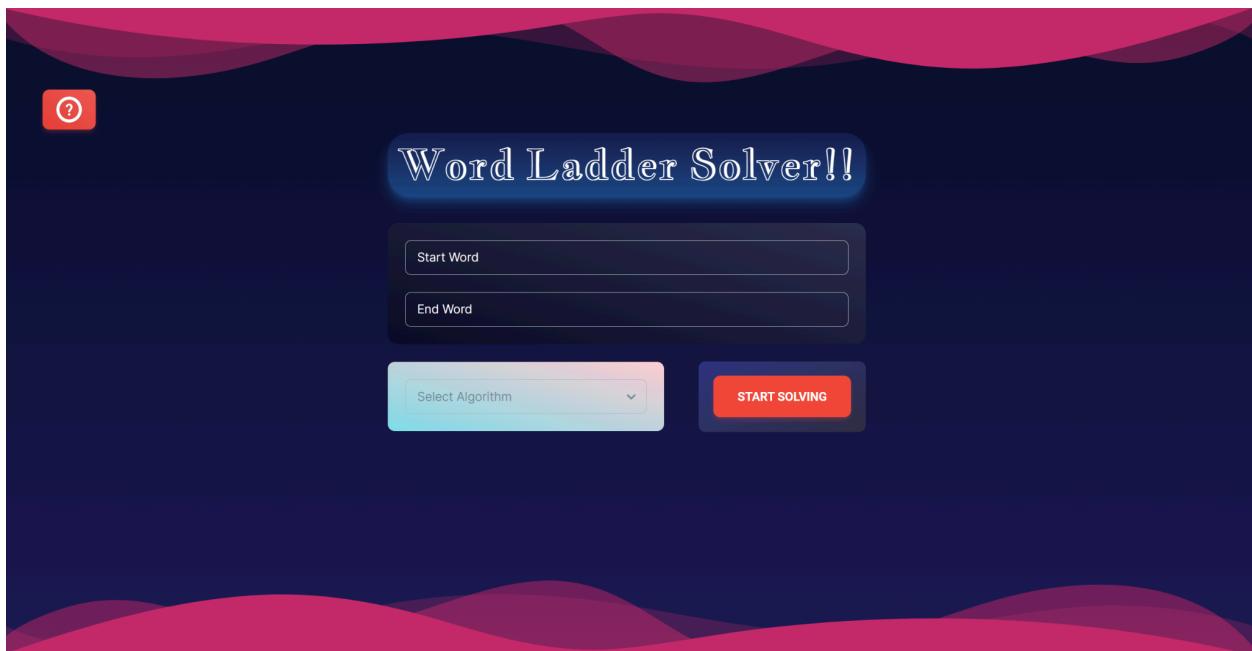
Dengan demikian, dari segi waktu eksekusi, GBFS merupakan pilihan yang paling cepat dalam kasus ini, meskipun tidak menjamin solusi optimal. Sementara itu, A\* memberikan keseimbangan antara waktu eksekusi dan optimalitas solusi, sementara UCS lebih lambat dan membutuhkan lebih banyak memori karena pendekatannya yang lebih sistematis.

## **BAB VI**

## **IMPLEMENTASI BONUS**

Pada tugas kecil ini, dibangun sebuah antarmuka pengguna (user interface) dalam pemecahan permasalahan Word Ladder menggunakan aplikasi berbasis web. Framework yang dipilih untuk mengimplementasikan website ini adalah Next.js, dengan dukungan dari Tailwind CSS untuk desainnya. Untuk menyambungkan program solver yang ditulis dalam bahasa Java dengan frontend, digunakan Spring Boot sebagai backend. Website yang dibangun memiliki tampilan yang sederhana, terdiri dari area input untuk pengguna memasukkan kata awal dan kata akhir, serta popup yang berisi informasi tentang program dan cara penggunaannya. Selain itu, juga disediakan halaman hasil yang menampilkan solusi dari permasalahan Word Ladder yang dipecahkan. Keseluruhan antarmuka dibuat dengan tujuan memberikan pengalaman pengguna yang baik dan memudahkan pengguna dalam memahami serta menggunakan program solver tersebut.

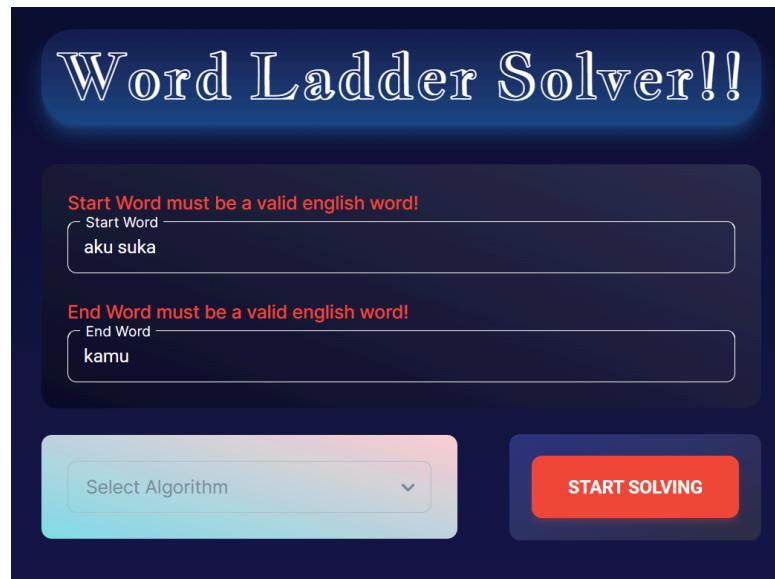
## 1. Component input



Gambar 11. Screenshot Halaman Input

Berikut adalah tampilan bagian input pada antarmuka pengguna. Di sini, pengguna dapat memasukkan kata awal (startWord) dan kata akhir (endWord), serta memilih algoritma yang diinginkan untuk menyelesaikan Word Ladder. Terdapat beberapa penanganan kasus yang dilakukan pada input, seperti ketika kata yang dimasukkan bukan merupakan kata bahasa Inggris yang valid, atau saat panjang dari startWord dan endWord tidak sama. Selain itu, juga terdapat

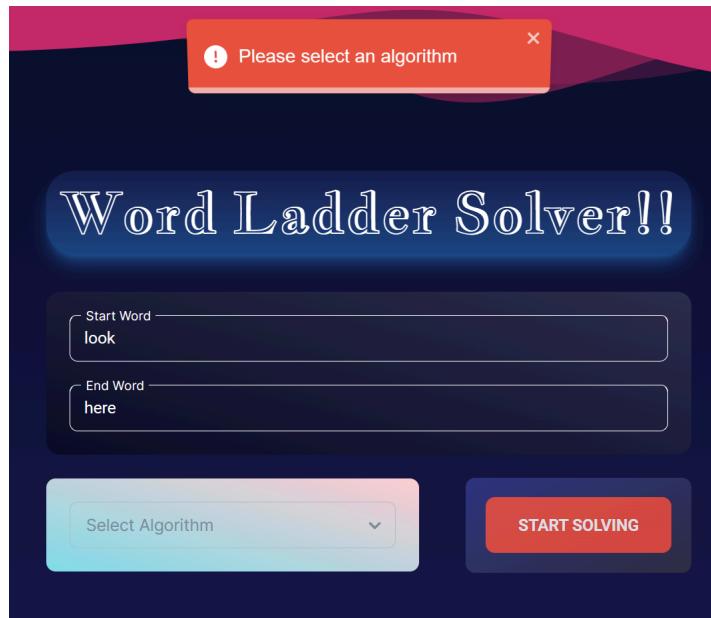
penanganan untuk kondisi saat pengguna mencoba memulai pencarian sebelum memasukkan input atau ketika input tidak memenuhi kriteria. Penanganan kasus-kasus ini membantu memastikan bahwa input yang diberikan sesuai dengan yang diharapkan dan membantu pengguna dalam menggunakan antarmuka dengan lancar.



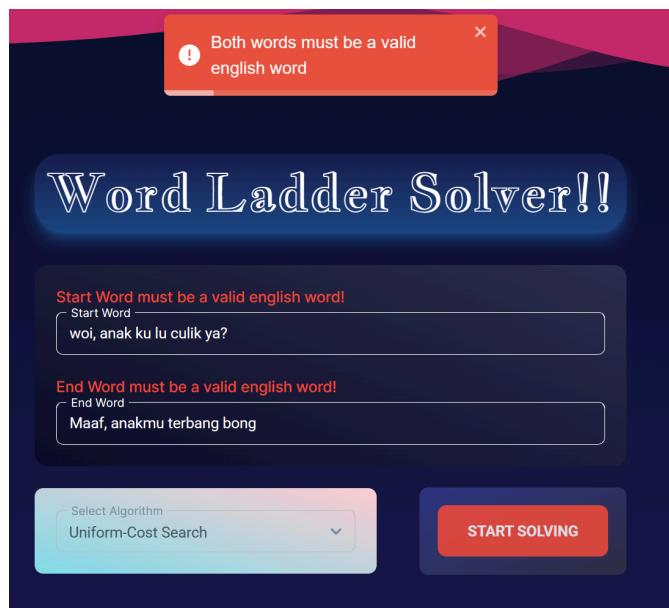
Gambar 12. Penanganan Kata Tidak Valid



Gambar 13. Notifikasi Panjang Kata Tidak Sama



Gambar 13. Notifikasi Algoritma Belum Diisi

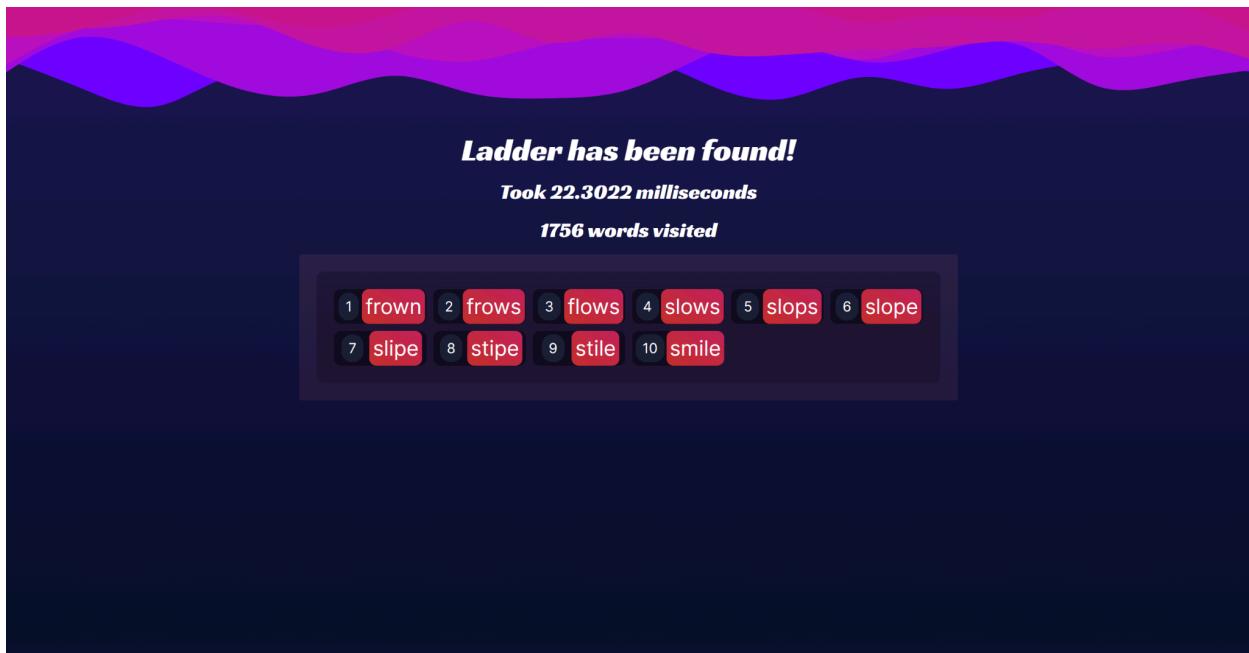


Gambar 14. Notifikasi Kata Harus Valid

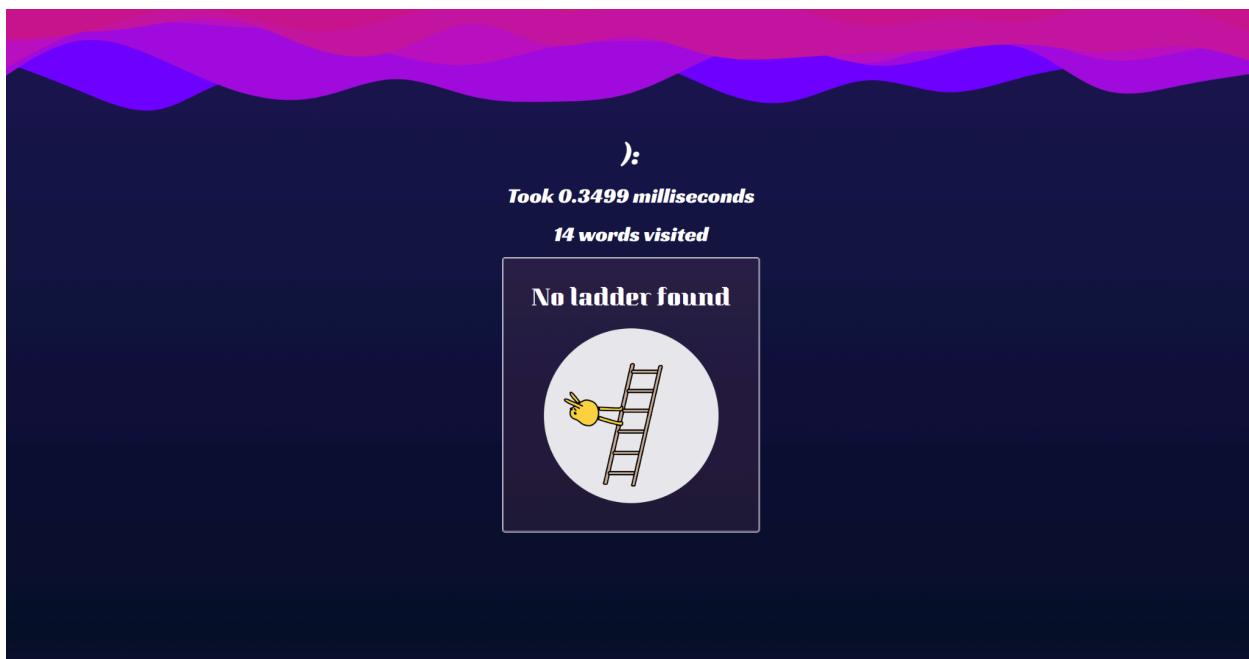
## 2. Component hasil

Pada bagian hasil, akan ditampilkan informasi mengenai lama eksekusi program, jumlah kata-kata atau node yang dikunjungi selama proses pencarian, serta jalur dari startWord hingga endWord. Jika program berhasil menemukan solusi, maka jalur tersebut akan ditampilkan secara jelas. Namun, jika program mengembalikan sebuah list kosong, yang menandakan bahwa solusi tidak ditemukan, maka pada halaman hasil akan ditampilkan pesan "no ladder found". Hal ini

memastikan bahwa pengguna mendapatkan informasi yang jelas mengenai hasil dari proses pencarian Word Ladder, sehingga dapat memahami apakah solusi berhasil ditemukan atau tidak.



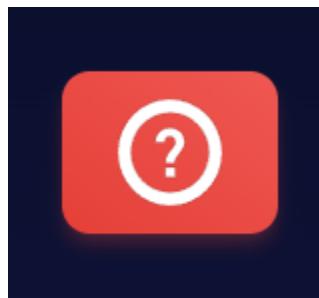
Gambar 15. Pencarian Berhasil



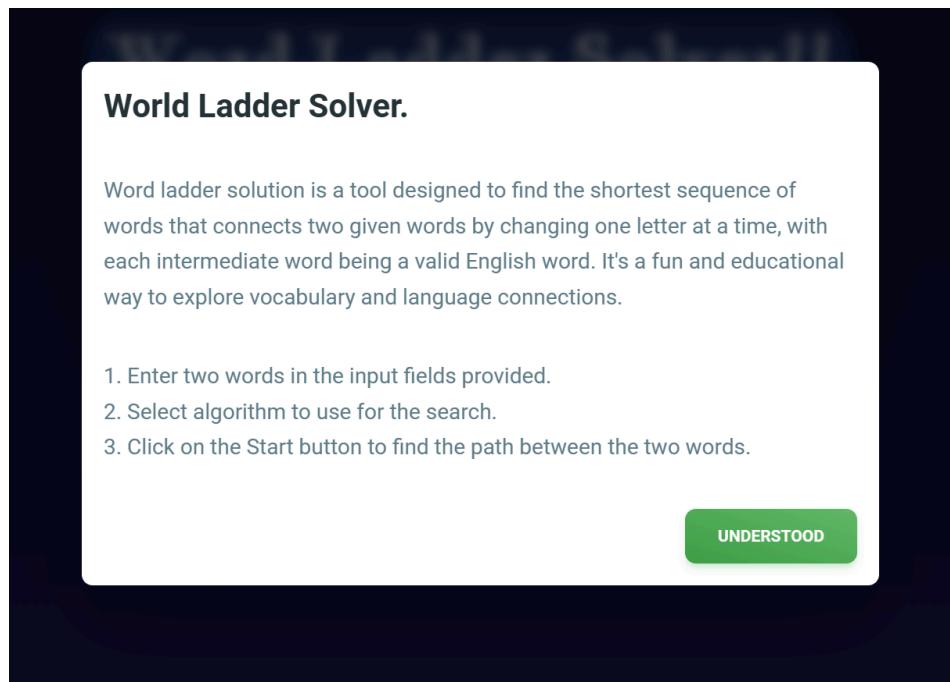
Gambar 16. Pencarian Tidak Ditemukan

### 3. Component about

Pada halaman utama, terdapat tombol berbentuk tanda tanya (?) yang bertindak sebagai pemicu untuk menampilkan sebuah popup. Popup ini berisi informasi mengenai program, termasuk penjelasan tentang tujuan dan fungsionalitas website, serta panduan cara penggunaannya. Dengan menekan tombol ini, pengguna dapat dengan mudah memahami konsep dan penggunaan aplikasi, sehingga memperjelas pengalaman pengguna dan meningkatkan kemudahan dalam menggunakan website ini.



Gambar 17. Button About



Gambar 18. About Popup

## **BAB VII**

## **KESIMPULAN**

Dari pengujian yang dilakukan pada tugas kecil ini, dapat disimpulkan bahwa algoritma pencarian yang paling optimal untuk pemecahan masalah trivia Word Ladder adalah A\*. Dibandingkan dengan UCS dan GBFS, A\* menunjukkan kinerja yang seimbang, dengan waktu eksekusi yang berada di tengah-tengah antara UCS dan GBFS, sementara mampu menghasilkan path yang optimal dan singkat. Sementara itu, UCS menunjukkan waktu eksekusi yang paling lama, namun menghasilkan path yang sama seperti A\*. Di sisi lain, GBFS menunjukkan waktu eksekusi yang paling cepat, tetapi terdapat beberapa kasus di mana algoritma ini gagal menghasilkan solusi. Hal ini disebabkan oleh kecenderungan GBFS untuk terjebak pada node tertentu tanpa kemampuan untuk melakukan backtracking.

Dengan demikian, meskipun GBFS memberikan kinerja yang cepat dalam beberapa kasus, namun tidak menjamin solusi optimal dan rentan terhadap kegagalan dalam menemukan solusi. UCS, meskipun konsisten dalam waktu eksekusi, menghasilkan solusi yang optimal seperti yang dilakukan oleh A\*. Oleh karena itu, A\* menjadi pilihan yang lebih diunggulkan karena mampu memberikan keseimbangan antara waktu eksekusi dan kualitas solusi dalam pemecahan masalah Word Ladder.

# LAMPIRAN

Tautan Repository : [https://github.com/thoriqsaputra/Tucil3\\_13522141](https://github.com/thoriqsaputra/Tucil3_13522141)

| Poin  | Ya                                  | Tidak                    |
|---|-------------------------------------|--------------------------|
| 1. Program berhasil dijalankan.   | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 2. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS                      | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 3. Solusi yang diberikan pada algoritma UCS optimal   | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 4. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 5. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma A*                       | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 6. Solusi yang diberikan pada algoritma A* optimal  | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 7. [Bonus]: Program memiliki tampilan GUI   | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

## **DAFTAR PUSTAKA**

Jagga, Savi. "Uniform Cost Search." 28 Februari 2023. [Online]. Tersedia: <https://www.educba.com/uniform-cost-search/>. [Diakses pada 5 Mei 2024, pukul 16:00].

sahilkumar19. "Greedy Best-First Search." 22 April 2023. [Online]. Tersedia: <https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search>. [Diakses pada 5 Mei 2024, pukul 19:00].

TECHBREAUX. "A\* Search." 12 April 2023. [Online]. Tersedia: <https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search>. [Diakses pada 5 Mei 2024, pukul 19:00].