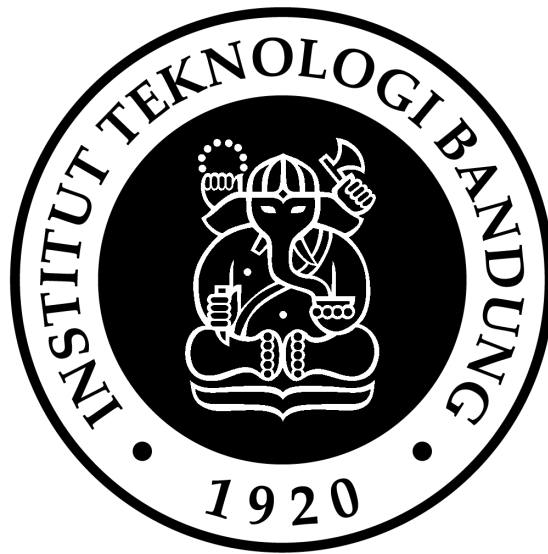


**LAPORAN TUGAS KECIL 3**

**IF2211 STRATEGI ALGORITMA**



Disusun oleh:

Muhammad Naufal Aulia (13522074)

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

**2024**

## DAFTAR ISI

|   |           |
|---|-----------|
| <b>DAFTAR ISI.....</b>  | <b>1</b>  |
| <b>BAB I.....</b>   | <b>2</b>  |
| <b>DESKRIPSI MASALAH.....</b>   | <b>2</b>  |
| 1.1 Word Ladder.....  | 2         |
| 1.2 Algoritma Uniform Cost Search (UCS).....  | 2         |
| 1.3 Algoritma Greedy Best-First Search.....   | 3         |
| 1.4 Algoritma A*.....   | 3         |
| 1.5 Algoritma Penyelesaian Masalah Word Ladder dengan Pendekatan UCS, Greedy Best-First Search, dan A*..... | 4         |
| 1.5.1 Pendekatan UCS.....   | 4         |
| 1.5.2 Pendekatan Greedy Best-First Search.....  | 5         |
| 1.5.3 Pendekatan A*.....  | 6         |
| 1.6 Analisis Implementasi Algoritma UCS, Greedy Best First Search, dan A*.....                              | 7         |
| <b>BAB II.....</b>  | <b>9</b>  |
| 2.1 Implementasi Program.....   | 9         |
| 2.2 Penjelasan Implementasi Bonus.....  | 10        |
| 2.3 Source Code.....  | 12        |
| <b>BAB III.....</b>   | <b>19</b> |
| 3.1 Tampilan Program (GUI).....   | 19        |
| 3.2 Hasil Pengujian.....  | 22        |
| 3.3 Analisis Perbandingan Solusi dari Hasil Pengujian.....  | 45        |
| <b>BAB IV.....</b>  | <b>48</b> |
| 4.1 Link Repository.....  | 48        |
| 4.2 Checklist.....  | 48        |
| <b>REFERENSI.....</b>   | <b>49</b> |

# BAB I

## DESKRIPSI MASALAH

### 1.1 Word Ladder

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, 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

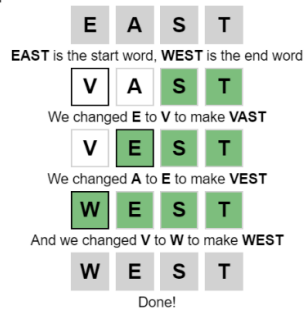
This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

##### 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



### 1.2 Algoritma Uniform Cost Search (UCS)

Uniform Cost Search (UCS) merupakan algoritma pencarian graf yang digunakan untuk mencari jalur terpendek dalam suatu graf dimana setiap simpul pada graf tersebut memiliki bobot yang berbeda. Fokus algoritma UCS terletak pada *cost* atau biaya terendah dalam jalur yang hendak dicapainya, diukur dari jumlah bobot simpul yang dilalui dari simpul awal ke simpul tujuan. Fungsi  $g(n)$  dalam UCS adalah biaya yang ditempuh dari simpul awal ke simpul  $n$ . UCS memprioritaskan

eksplorasi simpul berdasarkan *cost* terkecil dari simpul awal ke simpul tersebut sehingga dapat dipastikan bahwa jalur yang ditemukan adalah jalur terpendek.

### 1.3 Algoritma Greedy Best-First Search

Algoritma Greedy Best-First Search adalah algoritma pencarian graf yang menggunakan fungsi evaluasi  $f(n)$  untuk setiap simpul dalam graf. Fungsi evaluasi  $f(n)$  ini didasarkan pada estimasi biaya dari simpul  $n$  ke tujuan akhir (goal). Dalam Greedy Best-First Search, hanya bobot heuristik  $h(n)$  yang digunakan untuk menentukan urutan ekspansi simpul. Algoritma Greedy Best-First Search cenderung memilih simpul yang terlihat paling dekat dengan tujuan, tanpa memperhitungkan biaya total yang diperlukan untuk mencapai simpul tersebut. Karena algoritma ini hanya menggunakan nilai heuristik  $h(n)$  untuk memandu ekspansi, maka tidak dapat menjamin solusi optimal, namun algoritma ini memiliki keunggulan yakni pada kecepatan eksekusinya pada ruang pencarian yang besar.

### 1.4 Algoritma A\*

Algoritma A\* (A-star) merupakan algoritma pencarian graf yang digunakan dalam mencari jalur terpendek atau solusi optimal. Algoritma ini menggabungkan konsep dari dua metrik: biaya sejauh ini  $g(n)$  dan estimasi biaya dari simpul  $n$  ke tujuan akhir  $h(n)$ , dengan fungsi evaluasi  $f(n)$  yang didefinisikan sebagai:

$$f(n) = g(n) + h(n)$$

Di sini,  $g(n)$  adalah biaya yang telah dikeluarkan untuk mencapai simpul  $n$ , sedangkan  $h(n)$  adalah estimasi biaya dari simpul  $n$  ke tujuan akhir (heuristic). Jadi, algoritma A\* merupakan penggabungan atas evaluasi *cost* atau biaya simpul dari algoritma UCS dan evaluasi heuristik dari algoritma Greedy Best-First Search. Karena algoritma A\* menggunakan fungsi evaluasi  $f(n)$  yang menyertakan kedua informasi tersebut, algoritma ini memiliki kecenderungan untuk menemukan solusi optimal.

## **1.5 Algoritma Penyelesaian Masalah Word Ladder dengan Pendekatan UCS, Greedy Best-First Search, dan A\***

Dalam penyelesaian masalah menggunakan ketiga algoritma ini, terdapat kesamaan proses awal dan general yang digunakan yakni sebagai berikut:

1. Program akan melakukan inisialisasi dictionary yang berisi kumpulan kata-kata valid sebagai referensi dalam menyelesaikan permasalahan word ladder. Dictionary ini yang akan digunakan untuk mengekspan simpul berupa kata-kata.
2. Program akan menerima input berupa kata awal dan kata akhir.
3. Dari input tersebut, program akan membuat simpul awal yang merepresentasikan kata awal. Simpul ini direpresentasikan dalam kelas Node yang memiliki atribut berupa kata, parent, dan nilai  $f(n)$ .
4. Dalam melakukan ekspansi simpul, program akan menghasilkan kata-kata yang berbeda satu karakter dari kata yang diberikan. Kata-kata ini akan dihasilkan dengan mengganti satu karakter pada kata yang diberikan dengan karakter lain dari a sampai z, kecuali karakter asli pada posisi tersebut. Kata-kata yang dihasilkan ini akan dicek keberadaannya dalam dictionary. Kata-kata yang valid akan dijadikan tetangga dari simpul tersebut.
5. Selanjutnya untuk algoritma pemrosesan simpul kata akan dijelaskan pada bagian masing-masing algoritma.

### **1.5.1 Pendekatan UCS**

Pertama digunakan pendekatan UCS (Uniform Cost Search) dengan langkah deskriptif sebagai berikut:

1. Inisialisasi simpulHidup sebagai priority queue yang akan menyimpan node yang akan diekspan. Pada priority queue untuk UCS, node akan diurutkan berdasarkan prioritas nilai  $f(n) = g(n)$ , yakni nilai cost dari node tersebut. Semakin rendah costnya, semakin tinggi prioritasnya untuk diperiksa (berada lebih di depan dalam priority queue).
2. Priority queue simpulHidup ini akan diinisialisasi dengan simpul awal, yakni simpul yang merepresentasikan kata awal dengan nilai cost 0.
3. Lakukan loop selama simpulHidup tidak kosong.
4. Ambil simpul dengan nilai  $f(n)$  terkecil dari simpulHidup (dequeue).
5. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, maka pencarian selesai. Rekonstruksi path dari simpul akhir ke simpul awal, lalu kembalikan path tersebut.

6. Jika simpul saat ini bukan simpul tujuan, maka ekspansi simpul tersebut dengan menambahkan simpul tetangga yang valid ke dalam `simpulHidup`. Simpul tetangga ini akan memiliki nilai  $f(n)$  yang dihitung berdasarkan nilai  $g(n)$  dari simpul saat ini dan dipastikan tidak terdapat dalam simpul yang sudah diekspansi saat ini.
7. Ulangi langkah-langkah di atas dalam loop hingga `simpulHidup` kosong atau simpul tujuan ditemukan.
8. Jika `priority queue simpulHidup` kosong, maka tidak ditemukan solusi. Kembalikan hasil berupa list kosong.

### 1.5.2 Pendekatan Greedy Best-First Search

Pada algoritma greedy best-first search, langkahnya adalah sebagai berikut:

1. Inisialisasi `simpulHidup` sebagai `priority queue` yang akan menyimpan node yang akan diekspansi. Pada `priority queue` untuk Greedy Best First Search, node akan diurutkan berdasarkan prioritas nilai  $f(n) = h(n)$ , yakni nilai heuristik dari node tersebut. Semakin rendah heuristiknya, semakin tinggi prioritasnya untuk diperiksa (berada lebih di depan dalam `priority queue`).
2. `Priority queue simpulHidup` ini akan diinisialisasi dengan simpul awal oleh kata awal.
3. Lakukan loop selama `simpulHidup` tidak kosong.
4. Ambil simpul dengan nilai  $f(n)$  terkecil dari `simpulHidup` (dequeue).
5. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, maka pencarian selesai. Rekonstruksi path dari simpul akhir ke simpul awal, lalu kembalikan path tersebut.
6. Jika simpul saat ini bukan simpul tujuan, maka ekspansi simpul tersebut dengan menambahkan simpul tetangga yang valid ke dalam `simpulHidup`. Simpul tetangga ini akan memiliki nilai  $f(n)$  yang dihitung berdasarkan nilai heuristik dari simpul tersebut. Namun yang berbeda pada GBFS, `queue simpulHidup` akan dikosongkan dahulu sebelum menambahkan simpul tetangga yang baru. Hal ini sebagai bentuk implementasi sifat *irrevocable* atau tidak bisa membatalkan simpul yang telah dibangun saat ini, sesuai dengan referensi pada salindia perkuliahan (<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>)
7. Nilai heuristik ini dikalkulasi dengan menghitung *Hamming distance* antara kata tetangga dengan kata tujuan. Jarak *Hamming* adalah jumlah karakter yang berbeda antara dua kata.

8. Ulangi langkah-langkah di atas dalam loop hingga simpulHidup kosong atau simpul tujuan ditemukan.
9. Jika priority queue simpulHidup kosong, maka tidak ditemukan solusi. Kembalikan hasil berupa list kosong.

### 1.5.3 Pendekatan A\*

Pada algoritma A\*, langkahnya adalah sebagai berikut:

1. Inisialisasi simpulHidup sebagai priority queue yang akan menyimpan node yang akan diekspan. Pada priority queue untuk A\*, node akan diurutkan berdasarkan prioritas nilai  $f(n) = g(n) + h(n)$ , yakni nilai cost dari node tersebut ditambah dengan nilai heuristik dari node tersebut. Semakin rendah nilai  $f(n)$ -nya, semakin tinggi prioritasnya untuk diperiksa (berada lebih di depan dalam priority queue).
2. Priority queue simpulHidup ini akan diinisialisasi dengan simpul awal, yakni simpul yang merepresentasikan kata awal dengan nilai  $f(n) = g(n) + h(n) = 0$ .
3. Lakukan loop selama simpulHidup tidak kosong.
4. Ambil simpul dengan nilai  $f(n)$  terkecil dari simpulHidup (dequeue).
5. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, maka pencarian selesai. Rekonstruksi path dari simpul akhir ke simpul awal, lalu kembalikan path tersebut.
6. Jika simpul saat ini bukan simpul tujuan, maka ekspan simpul tersebut dengan menambahkan simpul tetangga yang valid ke dalam simpulHidup. Simpul tetangga ini akan memiliki nilai  $f(n)$  yang dihitung berdasarkan nilai  $g(n)$  dari simpul saat ini dan nilai  $h(n)$  dari simpul tetangga tersebut.
7. Nilai  $f(n)$  ini dihitung dengan menjumlahkan nilai  $g(n)$  yakni cost dari simpul saat ini dengan nilai  $h(n)$  yakni heuristik dari simpul tetangga tersebut. Heuristik yang digunakan adalah Hamming distance, yaitu jumlah karakter yang berbeda antara dua kata.
8. Ulangi langkah-langkah di atas dalam loop hingga simpulHidup kosong atau simpul tujuan ditemukan.
9. Jika priority queue simpulHidup kosong, maka tidak ditemukan solusi. Kembalikan hasil berupa list kosong.

## **1.6 Analisis Implementasi Algoritma UCS, Greedy Best First Search, dan A\***

Implementasi algoritma yang telah dijelaskan pada bagian sebelumnya akan dilanjutkan dengan analisis pada poin-poin di bawah ini.

### **1. Definisi $f(n)$ dan $g(n)$**

$f(n)$  merupakan fungsi evaluasi yang digunakan pada algoritma pencarian yang bergantung pada algoritmanya,  $f(n)$  dapat dihitung sebagai biaya aktual dari simpul awal ke simpul  $n$  ( $g(n)$ ), estimasi biayanya saja ( $h(n)$ ), atau gabungan keduanya. Dalam konteks permasalahan Word Ladder,  $f(n)$  adalah biaya total dari simpul awal ke simpul  $n$ . Bisa mencakup biaya aktual  $g(n)$  yakni jumlah langkah (atau perubahan karakter) yang diperlukan, juga estimasi biaya dari simpul  $n$  ke simpul tujuan yaitu banyaknya perbedaan karakter antara dua kata (Hamming Distance)

### **2. Admissibility heuristik yang digunakan pada algoritma A\***

Secara umum, heuristik akan admissible jika nilainya selalu kurang dari atau sama dengan biaya aktual untuk mencapai simpul tujuan dari simpul  $n$ . Pada penyelesaian masalah Word Ladder ini, heuristik yang digunakan pada algoritma A\* adalah Hamming distance, yaitu jumlah karakter yang berbeda antara dua kata. Hamming distance memenuhi syarat ini karena mencari jumlah karakter yang berbeda antara dua kata, yang pasti kurang dari atau sama dengan jumlah langkah yang diperlukan untuk mengubah satu kata menjadi yang lain. Sehingga, dengan menggunakan heuristik yang admissible seperti ini, A\* dapat menjamin pencarian solusi optimal dalam persoalan Word Ladder.

### **3. Algoritma UCS vs BFS pada implementasi masalah Word Ladder**

Pada permasalahan Word Ladder, setiap langkah memiliki biaya yang sama yakni 1 langkah per perubahan huruf. Dalam hal ini, algoritma UCS dan BFS sama-sama membangkitkan node dalam urutan teratur berdasarkan biaya atau kedalaman karena BFS mempertimbangkan kedalaman simpul sementara UCS mempertimbangkan biaya aktual yang dalam kasus ini juga merupakan langkah kedalaman simpul. Sehingga dapat diestimasi urutan simpul yang dibangkitkan dan path yang dihasilkan akan sama pula, dan dapat dikatakan bahwa UCS dan BFS pada permasalahan ini adalah sama.

### **4. Efisiensi algoritma**

Secara teoritis, algoritma A\* lebih efisien daripada UCS dalam kasus word ladder karena A\* menggunakan informasi heuristik (Hamming distance) untuk memandu pencarian. Dengan



mempertimbangkan biaya aktual  $g(n)$  dan estimasi biaya ke simpul tujuan  $h(n)$ , A\* dapat secara cerdas mengeksplorasi ruang pencarian dan lebih cenderung mengarah ke solusi optimal dengan jumlah langkah minimum. Jika dibandingkan dengan Greedy Best First Search, A\* masih tidak seefisien GBFS namun GBFS tidak bisa menjamin solusi yang dihasilkan adalah optimal terlepas dari efisiensinya yang tinggi.

#### **5. Optimalitas solusi yang ditawarkan Greedy Best First Search**

Algoritma Greedy Best First Search tidak menjamin solusi optimal untuk persoalan word ladder. Meskipun Greedy Best First Search berusaha untuk mengambil langkah terbaik berdasarkan heuristik (Hamming distance), tetapi bisa terperangkap dalam jalur yang tidak optimal dan tidak menemukan solusi optimal. Heuristik yang digunakan dalam Greedy Best First Search mungkin tidak selalu akurat atau mempertimbangkan semua faktor penting, yang dapat menyebabkan keputusan yang tidak optimal. Oleh karena itu, meskipun Greedy Best First Search mungkin cepat dalam pencarian, tidak ada jaminan bahwa solusi yang ditemukan akan optimal dalam kasus word ladder.

## **BAB II**

### **IMPLEMENTASI ALGORITMA**

#### **2.1 Implementasi Program**

Implementasi algoritma UCS, Greedy Best First Search, dan A\* dalam penyelesaian word ladder penulis buat dalam bahasa pemrograman Java. Program ini terdiri dari beberapa package dan class yang dijelaskan sebagai berikut:

##### **1. Package algorithm**

Berisi class-class yang merepresentasikan algoritma penyelesaian word ladder.

###### **a. Class WordLadderSolver**

Class ini merupakan abstract class yang merepresentasikan algoritma penyelesaian word ladder. Class ini memiliki beberapa method yang digunakan dalam penyelesaian word ladder, seperti generateNeighbors, reconstructPath, calculateG, calculateH, dan solve. Method solve merupakan abstract method yang harus diimplementasikan oleh subclass.

###### **b. Class UCSSolver**

Class ini merupakan subclass dari WordLadderSolver yang merepresentasikan algoritma UCS (Uniform Cost Search) dalam penyelesaian word ladder. Class ini mengimplementasikan method solve yang merupakan algoritma UCS dengan langkah-langkah yang sudah dijelaskan sebelumnya.

###### **c. Class GreedyBestFirstSolver**

Class ini merupakan subclass dari WordLadderSolver yang merepresentasikan algoritma Greedy Best First Search dalam penyelesaian word ladder. Class ini mengimplementasikan method solve yang merupakan algoritma Greedy Best First Search dengan langkah-langkah yang sudah dijelaskan sebelumnya.

###### **d. Class AStarSolver**

Class ini merupakan subclass dari WordLadderSolver yang merepresentasikan algoritma A\* dalam penyelesaian word ladder. Class ini mengimplementasikan method solve yang merupakan algoritma A\* dengan langkah-langkah yang sudah dijelaskan sebelumnya.

##### **2. Package util**

Berisi class-class yang digunakan dalam penyelesaian word ladder.

#### a. Class Dictionary

Class ini merepresentasikan dictionary yang berisi kumpulan kata-kata valid yang digunakan dalam penyelesaian word ladder. Class ini memiliki atribut dictionary yang merupakan `Set<String>` yang digunakan untuk menyimpan kata-kata dalam dictionary. Method yang dimilikinya adalah method `isWord` yang digunakan untuk mengecek apakah sebuah kata ada dalam dictionary, serta private method `loadDictionary` yang digunakan untuk memuat kata-kata dari file ke dalam *dictionary*. Kata-kata *dictionary* yang digunakan adalah *dictionary* dari pranala berikut: <https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt>

#### b. Class Node

Class ini merepresentasikan simpul dalam penyelesaian word ladder. Class ini memiliki atribut berupa kata, parent, dan nilai  $f(n)$ . Class ini juga memiliki method `countCost` yang digunakan untuk menghitung nilai cost dari simpul, serta method `getWord`, `getParent`, dan `getFn` yang digunakan untuk mendapatkan nilai atribut dari setiap simpul.

### 3. Package gui

Berisi class-class yang digunakan untuk membuat GUI (Graphical User Interface) dari program word ladder.

#### a. Class MainGUI

Class ini merupakan class utama yang digunakan untuk membuat GUI dari program word ladder. Class ini memiliki atribut berupa `JFrame`, `JPanel`, `TextField`, `Button`, dan `TextArea` yang digunakan untuk membuat tampilan GUI. Class ini juga memiliki method `initGUI` yang digunakan untuk menginisialisasi komponen-komponen GUI, serta method `actionPerformed` yang digunakan untuk menangani event dari button yang ada di GUI. Untuk menjalankan solver, terdapat method `runSolver` sebagai method utama untuk memanggil algoritma penyelesaian word ladder dan mengelola input serta outputnya.

## 2.2 Penjelasan Implementasi Bonus

Implementasi bonus GUI ini memanfaatkan Java Swing untuk membuat antarmuka pengguna grafis (GUI) untuk program Word Ladder Solver. Dalam implementasinya, dibuat package terpisah yakni package `gui` yang telah dijelaskan di bagian sebelumnya. Berikut adalah penjelasan mengenai komponen-komponen dan fungsionalitas yang disediakan oleh GUI:

1. JTextField (startWordField dan endWordField):  
Komponen ini digunakan untuk memasukkan kata awal dan kata akhir dalam permainan Word Ladder. Input ini kemudian diterima untuk diproses pada pencarian solusi.
2. JComboBox (algorithmComboBox):  
Komponen ini digunakan untuk memilih algoritma pencarian yang akan digunakan, yaitu UCS, Greedy Best First Search, atau A\* dalam bentuk *dropdown option*. Pengguna dapat memilih salah satu algoritma untuk menyelesaikan permainan Word Ladder.
3. JButton (solveButton):  
Tombol ini digunakan untuk memulai proses pencarian solusi setelah pengguna memasukkan kata awal, kata akhir, dan memilih algoritma. Ketika tombol ini ditekan, program akan menjalankan solver sesuai dengan konfigurasi yang diberikan oleh pengguna.
4. JTextArea (resultArea):  
Area ini digunakan untuk menampilkan solusi dari permainan Word Ladder. Jika solusi ditemukan, rute kata-kata dalam solusi akan ditampilkan di sini. Jika tidak ditemukan solusi, pesan "No path found" akan ditampilkan.
5. JLabel (stepsLabel, visitedNodesLabel, dan executionTimeLabel):  
Label-label ini digunakan untuk menampilkan informasi tambahan terkait dengan proses pencarian solusi, berikut rinciannya:
  - stepsLabel: Menampilkan jumlah langkah dalam solusi.
  - visitedNodesLabel: Menampilkan jumlah simpul yang dikunjungi selama pencarian solusi.
  - executionTimeLabel: Menampilkan waktu eksekusi yang diperlukan untuk menemukan solusi.
6. Background Image:  
Komponen ImagePanel digunakan untuk menampilkan gambar latar belakang pada GUI untuk menambah kesan modern.
7. ActionListener solveButton:  
Ketika tombol "SOLVE" ditekan, program akan menjalankan method runSolver dan memulai proses sesuai dengan algoritma yang dipilih oleh pengguna dan menampilkan solusi serta informasi terkait di area yang sesuai.
8. Validasi Input:  
Terdapat beberapa validasi input yang dilakukan untuk memastikan bahwa kata awal dan akhir yang dimasukkan oleh pengguna valid, serta memastikan bahwa panjang kata awal dan akhir sama.

## 2.3 Source Code

### WordLadderSolver.java

```
public abstract class WordLadderSolver {
    protected Dictionary dictionary;
    protected String startWord;
    protected String endWord;
    protected int visitedNodes;

    // Constructor untuk WordLadderSolver
    public WordLadderSolver(Dictionary dictionary, String startWord, String endWord) {
        this.dictionary = dictionary;
        this.startWord = startWord.toLowerCase();
        this.endWord = endWord.toLowerCase();
    }

    // Method untuk menghasilkan kata-kata yang berbeda satu karakter dari kata yang
    // diberikan
    protected List<String> generateNeighbors(String word) {
        List<String> neighbors = new ArrayList<>();
        char[] wordArray = word.toCharArray();
        for (int i = 0; i < word.length(); i++) {
            char originalChar = wordArray[i];
            for (char c = 'a'; c <= 'z'; c++) {
                if (c != originalChar) {
                    wordArray[i] = c;
                    String neighbor = new String(wordArray);
                    if (dictionary.isWord(neighbor)) {
                        neighbors.add(neighbor);
                    }
                }
            }
            wordArray[i] = originalChar;
        }
        return neighbors;
    }

    // Method untuk mendapatkan jumlah node yang dikunjungi
    public int getVisitedNodes() {
        return visitedNodes;
    }

    // Method untuk merekonstruksi path dari node akhir ke node awal
    public List<String> reconstructPath(Node endNode) {
```

```

        List<String> path = new ArrayList<>();
        Node currentNode = endNode;
        while (currentNode != null) {
            path.add(0, currentNode.getWord().toUpperCase()); // Add the word to the
beginning of the path
            currentNode = currentNode.getParent();
        }
        return path;
    }

    // Method untuk menghitung nilai g(n) dari node
    public int calculateG(Node node) {
        return node.countCost();
    }

    // Method untuk menghitung nilai h(n) dari node, menggunakan heuristik Hamming distance
    public int calculateH(String word) {
        int distance = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) != endWord.charAt(i)) {
                distance++;
            }
        }
        return distance;
    }

    // Abstract method algoritma penyelesaian
    public abstract List<String> solve();
}

```

## UCSSolver.java

```

public class UCSSolver extends WordLadderSolver {
    private PriorityQueue<Node> simpulHidup;

    // Constructor untuk UCSSolver
    public UCSSolver(Dictionary dictionary, String startWord, String endWord) {
        super(dictionary, startWord, endWord);
    }

    @Override
    public List<String> solve() {
        // Jika kata bukan kata valid, kembalikan list berisi "Invalid"
        if (!dictionary.isWord(startWord) || !dictionary.isWord(endWord)) {

```

```

        return new ArrayList<> (Collections.singletonList("Invalid"));
    }

    // Priority queue untuk menyimpan node yang akan diekspan, diurutkan berdasarkan
    nilai fn
    simpulHidup = new PriorityQueue<> (Comparator.comparingInt (Node::getFn));
    // Set untuk melacak node yang sudah dikunjungi
    Set<String> explored = new HashSet<> ();

    Node startNode = new Node (startWord, null, 0);
    simpulHidup.add (startNode);

    while (!simpulHidup.isEmpty()) {
        Node currentNode = simpulHidup.poll();
        String currentWord = currentNode.getWord();

        visitedNodes++;

        // Jika node saat ini adalah node tujuan, pencarian selesai
        if (currentWord.equals (endWord)) {
            return reconstructPath (currentNode);
        }

        explored.add (currentWord);

        // Generate tetangga dari kata saat ini
        List<String> neighbors = generateNeighbors (currentWord);
        for (String neighbor : neighbors) {
            if (!explored.contains (neighbor)) {
                int fn = calculateG (currentNode); // skor f(n) adalah g(n) pada UCS
                Node neighborNode = new Node (neighbor, currentNode, fn);
                simpulHidup.add (neighborNode);
            }
        }
    }

    // Jika simpulHidup kosong dan tidak ditemukan solusi, kembalikan list kosong
    return new ArrayList<> ();
}

```

### GreedyBestFirstSolver.java

```

public class GreedyBestFirstSolver extends WordLadderSolver {
    private PriorityQueue<Node> simpulHidup;

```

```

// Constructor untuk GreedyBestFirstSolver
public GreedyBestFirstSolver(Dictionary dictionary, String startWord, String endWord) {
    super(dictionary, startWord, endWord);
}

@Override
public List<String> solve() {
    // Jika kata bukan kata valid, kembalikan list berisi "Invalid"
    if (!dictionary.isWord(startWord) || !dictionary.isWord(endWord)) {
        return new ArrayList<>(Collections.singletonList("Invalid"));
    }

    // Priority queue untuk menyimpan node yang akan diekspan, diurutkan berdasarkan
    nilai fn
    simpulHidup = new PriorityQueue<>(Comparator.comparingInt(Node::getFn));
    // Set untuk melacak node yang sudah dikunjungi
    Set<String> explored = new HashSet<>();

    Node startNode = new Node(startWord, null, 0);
    simpulHidup.add(startNode);

    while (!simpulHidup.isEmpty()) {
        Node currentNode = simpulHidup.poll();
        String currentWord = currentNode.getWord();

        visitedNodes++;

        // Jika node saat ini adalah node tujuan, pencarian selesai
        if (currentWord.equals(endWord)) {
            return reconstructPath(currentNode);
        }

        explored.add(currentWord);

        // Generate tetangga dari kata saat ini
        List<String> neighbors = generateNeighbors(currentWord);
        for (String neighbor : neighbors) {
            if (!explored.contains(neighbor)) {
                int fn = calculateH(neighbor); // skor f(n) adalah h(n) pada Greedy Best
                First Search

                Node neighborNode = new Node(neighbor, currentNode, fn);
                simpulHidup.add(neighborNode);
            }
        }
    }
}

```



```

        // Jika simpulHidup kosong dan tidak ditemukan solusi, kembalikan list kosong
        return new ArrayList<>();
    }
}

```

## AStarSolver.java

```

public class AStarSolver extends WordLadderSolver {
    private PriorityQueue<Node> simpulHidup;

    // Constructor untuk AStarSolver
    public AStarSolver(Dictionary dictionary, String startWord, String endWord) {
        super(dictionary, startWord, endWord);
    }

    @Override
    public List<String> solve() {
        // Jika kata bukan kata valid, kembalikan list berisi "Invalid"
        if (!dictionary.isWord(startWord) || !dictionary.isWord(endWord)) {
            return new ArrayList<>(Collections.singletonList("Invalid"));
        }

        // Priority queue untuk menyimpan node yang akan diekspan, diurutkan berdasarkan
        // nilai fn
        simpulHidup = new PriorityQueue<>(Comparator.comparingInt(Node::getFn));
        // Set untuk melacak node yang sudah dikunjungi
        Set<String> explored = new HashSet<>();

        Node startNode = new Node(startWord, null, 0);
        simpulHidup.add(startNode);

        while (!simpulHidup.isEmpty()) {
            Node currentNode = simpulHidup.poll();
            String currentWord = currentNode.getWord();

            visitedNodes++;

            // Jika node saat ini adalah node tujuan, pencarian selesai
            if (currentWord.equals(endWord)) {
                return reconstructPath(currentNode);
            }

            explored.add(currentWord);
        }
    }
}

```

```

        // Generate tetangga dari kata saat ini
        List<String> neighbors = generateNeighbors(currentWord);
        for (String neighbor : neighbors) {
            if (!explored.contains(neighbor)) {
                int gn = calculateG(currentNode);
                int hn = calculateH(neighbor);
                int fn = gn + hn; // skor f(n) adalah g(n) + h(n) pada A*
                Node neighborNode = new Node(neighbor, currentNode, fn);
                simpulHidup.add(neighborNode);
            }
        }
    }

    // Jika simpulHidup kosong dan tidak ditemukan solusi, kembalikan list kosong
    return new ArrayList<>();
}
}

```

## Dictionary.java

```

public class Dictionary {
    private Set<String> dictionary;

    // Constructor untuk Dictionary
    public Dictionary(String filePath) {
        dictionary = new HashSet<>();
        loadDictionary(filePath);
    }

    // Memuat kata-kata dari file ke dalam dictionary
    private void loadDictionary(String filePath) {
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String word;
            while ((word = br.readLine()) != null) {
                dictionary.add(word.toUpperCase());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Method untuk mengecek apakah kata ada di dalam dictionary
    public boolean isWord(String word) {
        return dictionary.contains(word.toUpperCase());
    }
}

```

## Node.java

```
public class Node {
    private String word;
    private Node parent;
    private int fn;

    public Node(String word, Node parent, int fn) {
        this.word = word;
        this.parent = parent;
        this.fn = fn;
    }

    public String getWord() {
        return word;
    }

    public Node getParent() {
        return parent;
    }

    public int getFn() {
        return fn;
    }

    public void setFn(int fn) {
        this.fn = fn;
    }

    public int countCost() {
        int cost = 0;
        Node current = this;
        while (current.getParent() != null) {
            cost++;
            current = current.getParent();
        }
        return cost;
    }
}
```

Untuk *source code* GUI karena terlalu panjang untuk ditampilkan, maka dapat langsung dilihat melalui pranala github repository pada bab iv di bawah dokumen ini.

## BAB III

### HASIL PENGUJIAN DAN ANALISIS

#### 3.1 Tampilan Program (GUI)

##### Halaman Awal

**WORD LEDENG**  
word ladder solver

START WORD:

END WORD:

ALGORITHM:

UCS

UCS

Greedy Best First Search

A\*

SOLVE

PATH:

STEP :

VISITED NODES:

EXEC TIME:

## Invalid Input Kosong

# WORD LEDENG

word ladder solver

START WORD:

END WORD:


ALGORITHM:

UCS

SOLVE

PATH:

Error

 Please enter both start and end words.

OK

STEP :

VISITED NODES:

EXEC TIME:

## Input Bukan Kata Valid

# WORD LEDENG

word ladder solver

START WORD:

UOSAS

END WORD:

MIXUE

ALGORITHM:

UCS

SOLVE

PATH:

Error

Invalid start or end word.

OK

STEP :

VISITED NODES:

EXEC TIME:

### 3.2 Hasil Pengujian

#### Soal 1:

#### Kata dengan 2 Huruf

Start word: GO

End word: IF

#### Algoritma UCS:

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input fields: "START WORD:" with the value "GO", "END WORD:" with the value "IF", and "ALGORITHM:" with a dropdown menu showing "UCS". Below these is a pink "SOLVE" button. On the right, a "PATH:" label is followed by a list of words: "GO", "SO", "SH", "OH", "OF", and "IF". At the bottom, there are three status boxes: "STEP:" with the value "5", "VISITED NODES:" with the value "1036", and "EXEC TIME:" with the value "23 ms".

| START WORD: | END WORD: | ALGORITHM: | PATH:                            | STEP: | VISITED NODES: | EXEC TIME: |
|-------------|-----------|------------|----------------------------------|-------|----------------|------------|
| GO          | IF        | UCS        | GO<br>SO<br>SH<br>OH<br>OF<br>IF | 5     | 1036           | 23 ms      |

## Algoritma GBFS:

# WORD LADDER

word ladder solver

START WORD:

GO

END WORD:

IF

ALGORITHM:

Greedy Best First Search

SOLVE

STEP :

43

VISITED NODES:

44

EXEC TIME:

1 ms

PATH:

PI  
SI  
XI  
XU  
MU  
NU  
NE  
AE  
BE  
DE  
HE  
ME  
OE  
OF  
IF



### Algoritma A\*:

## WORD LEDENG

word ladder solver

START WORD:

GO

END WORD:

IF

ALGORITHM:

A\*

SOLVE

STEP :  
5

VISITED NODES:  
243

EXEC TIME:  
3 ms

PATH:

GO  
YO  
YE  
OE  
OF  
IF

**Keterangan:** dari hasil ketiga algoritma tersebut, didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 5 langkah, sementara GBFS memerlukan 43 langkah. Dari segi efisiensi, waktu yang diperlukan oleh UCS paling lama di antara ketiganya. Disusul dengan A\* dan yang paling cepat adalah GBFS.

**Soal 2:**

**Kata dengan 4 Huruf**

Start word: BASE

End word: ROOT

**Algoritma UCS:**

The image shows a web application titled "WORD LEDENG" with the subtitle "word ladder solver". The interface is dark-themed with light blue and pink accents. On the left, there are three input fields: "START WORD:" with the value "BASE", "END WORD:" with the value "ROOT", and "ALGORITHM:" with a dropdown menu showing "UCS". Below these is a pink "SOLVE" button. On the right, a "PATH:" label is above a large light blue box containing a list of words: BASE, CASE, CAST, COST, COOT, and ROOT. At the bottom, there are three status boxes: "STEP : 5", "VISITED NODES: 18570", and "EXEC TIME: 205 ms".

**WORD LEDENG**  
word ladder solver

START WORD:  
BASE

END WORD:  
ROOT

ALGORITHM:  
UCS

SOLVE

PATH:

BASE  
CASE  
CAST  
COST  
COOT  
ROOT

STEP :  
5

VISITED NODES:  
18570

EXEC TIME:  
205 ms

## Algoritma GBFS:

# WORD LEDENG

word ladder solver

START WORD:

BASE

END WORD:

ROOT

ALGORITHM:

Greedy Best First Search

SOLVE

PATH:

BASE  
RASE  
ROSE  
ROBE  
RODE  
ROLE  
ROPE  
ROTE  
ROUE  
ROUT  
ROOT

STEP :

10

VISITED NODES:

11

EXEC TIME:

0 ms

### Algoritma A\*:

## WORD LEDENG

word ladder solver

START WORD:  
BASE

END WORD:  
ROOT

ALGORITHM:  
A\*

SOLVE

STEP :  
5

VISITED NODES:  
56

EXEC TIME:  
0 ms

PATH:

BASE  
LASE  
LOSE  
LOST  
LOOT  
ROOT

**Keterangan:** didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 5 langkah, sementara GBFS memerlukan 10 langkah. Dari segi efisiensi, waktu yang diperlukan oleh UCS paling lama di antara ketiganya. Sementara GBFS dan A\* tidak terlihat perbandingannya karena kurang dari 0 ms.

**Soal 3:**

**Kata dengan 4 Huruf Rute Terpanjang**

Start word: ATOM

End word: UNAU

**Algoritma UCS:**

The screenshot shows a web application titled "WORD LEDENG" with the subtitle "word ladder solver". On the left, there are three input fields: "START WORD:" with the value "ATOM", "END WORD:" with the value "UNAU", and "ALGORITHM:" with a dropdown menu showing "UCS". Below these is a pink "SOLVE" button. On the right, a large light blue box displays the "PATH:" of the solution, listing the words: STOP, SLOP, SLOE, ALOE, ALEE, ALES, ANES, ANTS, ANTI, INTI, INTO, UNTO, UNCO, UNCI, UNAI, and UNAU. At the bottom, there are three status boxes: "STEP:" with the value "17", "VISITED NODES:" with the value "84409", and "EXEC TIME:" with the value "827 ms".

**WORD LEDENG**  
word ladder solver

START WORD:  
ATOM

END WORD:  
UNAU

ALGORITHM:  
UCS

SOLVE

PATH:

STOP  
SLOP  
SLOE  
ALOE  
ALEE  
ALES  
ANES  
ANTS  
ANTI  
INTI  
INTO  
UNTO  
UNCO  
UNCI  
UNAI  
UNAU

STEP :  
17

VISITED NODES:  
84409

EXEC TIME:  
827 ms

## Algoritma GBFS:

# WORD LEDENG

word ladder solver

START WORD:

ATOM

END WORD:

UNAU

ALGORITHM:

Greedy Best First Search

SOLVE

PATH:

No path found.

STEP :

VISITED NODES:

3

EXEC TIME:

0 ms

### Algoritma A\*:

## WORD LEDENG

word ladder solver

START WORD:

ATOM

END WORD:

UNAU

ALGORITHM:

A\*

SOLVE

PATH:

ATOM  
SNOW  
KNOW  
KNEW  
ANEW  
ANES  
ANTS  
ANTI  
INTI  
INTO  
UNTO  
UNCO  
UNCI  
UNAI  
UNAU

STEP :

17

VISITED NODES:

78634

EXEC TIME:

668 ms

**Keterangan:** ini adalah *ladder* dengan 4 huruf terpanjang menurut <http://datagenetics.com/>. Didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 17 langkah, sementara GBFS ternyata tidak mampu menghasilkan solusi karena terjebak di lokal minima dan dari kata yang diekspan tersebut tidak dapat mencapai kata tujuan. Dari segi efisiensi, waktu yang diperlukan oleh UCS adalah yang terlama. Disusul dengan A\* yang sedikit lebih cepat.

**Soal 4:**

**Kata dengan 5 Huruf**

Start word: MELON

End word: LEMON

**Algoritma UCS:**

# WORD LEDENG

word ladder solver

START WORD:

MELON

END WORD:

LEMON

ALGORITHM:

UCS

PATH:

MELON  
MESON  
MASON  
MACON  
RACON  
RECON  
REDON  
REDAN  
REMAN  
LEMAN  
LEMON

SOLVE

STEP :

10

VISITED NODES:

157

EXEC TIME:

1 ms



## Algoritma GBFS:

# WORD LADDER

word ladder solver

START WORD:  
MELON

END WORD:  
LEMON

ALGORITHM:  
Greedy Best First Search

SOLVE

STEP :  
5

VISITED NODES:  
5

EXEC TIME:  
0 ms

PATH:

No path found.

## Algoritma A\*:

# WORD LEDENG

word ladder solver

START WORD:  
MELON

END WORD:  
LEMON

ALGORITHM:  
A\*

SOLVE

PATH:

MELON  
MESON  
MASON  
MACON  
RACON  
RECON  
REWON  
REWAN  
REMAN  
LEMAN  
LEMON

STEP :  
10

VISITED NODES:  
46

EXEC TIME:  
0 ms

**Keterangan:** didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 10 langkah, sementara GBFS ternyata tidak mampu menghasilkan solusi karena terjebak di lokal minima dan dari kata yang diekspan tersebut tidak dapat mencapai kata tujuan.. Dari segi efisiensi, waktu yang diperlukan oleh UCS adalah yang terlama, terlihat dari jumlah simpul yang dikunjunginya yang juga terbanyak.

**Soal 5:**

**Kata dengan 5 Huruf Rute Terpanjang**

Start word: NYLON

End word: ILLER

**Algoritma UCS:**

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input sections: "START WORD:" with the value "NYLON", "END WORD:" with the value "ILLER", and "ALGORITHM:" with a dropdown menu showing "UCS". Below these is a pink "SOLVE" button. On the right, a light blue box labeled "PATH:" contains a list of words: SPREE, SPRUE, SPRUG, SPRIG, SPRIT, SPLIT, UPLIT, UNLIT, UNLET, INLET, ISLET, ISLES, IDLES, IDLER, and ILLER. At the bottom, there are three status boxes: "STEP:" with the value "30", "VISITED NODES:" with the value "126137", and "EXEC TIME:" with the value "1479 ms".

| START WORD: | END WORD: | ALGORITHM: | PATH:   | STEP: | VISITED NODES: | EXEC TIME: |
|-------------|-----------|------------|---|-------|----------------|------------|
| NYLON       | ILLER     | UCS        | SPREE<br>SPRUE<br>SPRUG<br>SPRIG<br>SPRIT<br>SPLIT<br>UPLIT<br>UNLIT<br>UNLET<br>INLET<br>ISLET<br>ISLES<br>IDLES<br>IDLER<br>ILLER | 30    | 126137         | 1479 ms    |

## Algoritma GBFS:

# WORD LADDER

word ladder solver

START WORD:

NYLON

END WORD:

ILLER

ALGORITHM:

Greedy Best First Search

SOLVE

STEP :

VISITED NODES:

14

EXEC TIME:

0 ms

PATH:

No path found.

### Algoritma A\*:

The screenshot shows the 'WORD LEDENG word ladder solver' interface. On the left, there are input fields for 'START WORD: NYLON', 'END WORD: ILLER', and 'ALGORITHM: A\*'. A pink 'SOLVE' button is below these. On the right, a 'PATH:' list shows the sequence of words: SPARE, SPALE, SPALL, SPAIL, SPAIT, SPLIT, UPLIT, UNLIT, UNLET, INLET, ISLET, ISLED, IDLED, IDLER, and ILLER. At the bottom, three status boxes show 'STEP: 30', 'VISITED NODES: 111534', and 'EXEC TIME: 1227 ms'.

**Keterangan:** ini adalah ladder dengan 5 huruf terpanjang menurut <http://datagenetics.com/>. Didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 30 langkah, sementara GBFS ternyata tidak mampu menghasilkan solusi karena terjebak di lokal minima dan dari kata yang diekspan tersebut tidak dapat mencapai kata tujuan.. Dari segi efisiensi, waktu yang diperlukan oleh UCS paling lama di antara ketiganya. Disusul dengan A\* yang sedikit lebih cepat. Hal ini sebanding dengan banyaknya simpul yang dikunjungi oleh masing-masing algoritma.

**Soal 6:**

**Kata dengan 7 Huruf Rute Terpanjang**

Start word: ATLASES

End word: CABARET

**Algoritma UCS:**

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input sections: "START WORD:" with "ATLASES", "END WORD:" with "CABARET", and "ALGORITHM:" with a dropdown menu set to "UCS". Below these is a pink "SOLVE" button. On the right, a large light blue box labeled "PATH:" contains a list of 15 words: REVELED, RAVELED, RAVENED, HAVENED, HAVERED, WAVERED, WATERED, CATERED, CAPERED, TAPERED, TABERED, TABORED, TABORET, TABARET, and CABARET. At the bottom, three status boxes show "STEP : 52", "VISITED NODES: 1304170", and "EXEC TIME: 19370 ms".

**WORD LEDENG**  
word ladder solver

START WORD:  
ATLASES

END WORD:  
CABARET

ALGORITHM:  
UCS

SOLVE

PATH:

- REVELED
- RAVELED
- RAVENED
- HAVENED
- HAVERED
- WAVERED
- WATERED
- CATERED
- CAPERED
- TAPERED
- TABERED
- TABORED
- TABORET
- TABARET
- CABARET

STEP : 52

VISITED NODES: 1304170

EXEC TIME: 19370 ms

## Algoritma GBFS:

# WORD LEDENG

word ladder solver

START WORD:

ATLASES

END WORD:

CABARET

ALGORITHM:

Greedy Best First Search

SOLVE

STEP :

10

VISITED NODES:

10

EXEC TIME:

0 ms

PATH:

No path found

### Algoritma A\*:

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input fields: "START WORD:" with the value "ATLASES", "END WORD:" with the value "CABARET", and "ALGORITHM:" with a dropdown menu showing "A\*". Below these is a pink "SOLVE" button. On the right, a large light blue box displays the "PATH:" of words: REVELED, RAVELED, RAVENED, HAVENED, HAVERED, WAVERED, WATERED, CATERED, CAPERED, TAPERED, TABERED, TABORED, TABORET, TABARET, and CABARET. At the bottom, three status boxes show "STEP : 52", "VISITED NODES: 168683", and "EXEC TIME: 2497 ms".

**Keterangan:** ini adalah *ladder* dengan 7 huruf terpanjang menurut <http://datagenetics.com/>. Didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 52 langkah, sementara GBFS ternyata tidak mampu menghasilkan solusi karena terjebak di lokal minima dan dari kata yang diekspan tersebut tidak dapat mencapai kata tujuan. Dari segi efisiensi, waktu yang diperlukan oleh UCS paling lama di antara ketiganya, disusul dengan A\*. Hal ini sebanding dengan banyaknya simpul yang dikunjungi oleh masing-masing algoritma.



**Soal 7:**

**Kata dengan 5 Huruf**

Start word: MELON

End word: PIZZA

**Algoritma UCS:**

# WORD LEDENG

word ladder solver

START WORD:  
MELON

END WORD:  
PIZZA

ALGORITHM:  
UCS

SOLVE

PATH:

No path found.

STEP :

VISITED NODES:  
122391

EXEC TIME:  
1505 ms

## Algoritma GBFS:

# WORD LEDENG

word ladder solver

START WORD:

MELON

END WORD:

PIZZA

ALGORITHM:

Greedy Best First Search

PATH:

No path found|

SOLVE

STEP :

VISITED NODES:

4

EXEC TIME:

0 ms

### Algoritma A\*:

## WORD LEDENG

word ladder solver

START WORD:  
MELON

END WORD:  
PIZZA

ALGORITHM:  
A\*

SOLVE

PATH:

No path found.

STEP :

VISITED NODES:  
118093

EXEC TIME:  
1337 ms

**Keterangan:** didapat bahwa setiap algoritma tidak dapat menemukan solusi rute dari start word ke end word. UCS mengunjungi paling banyak node hingga memberikan hasil "No path found" dengan waktu eksekusi terlama, disusul dengan A\* dan GBFS yang paling sedikit dan tercepat.

**Soal 8:**

**Kata dengan 8 Huruf Rute Terpanjang**

Start word: QUIRKING

End word: WRATHING

**Algoritma UCS:**

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input sections: "START WORD:" with "QUIRKING", "END WORD:" with "WRATHING", and "ALGORITHM:" with "UCS" selected. A pink "SOLVE" button is at the bottom left. On the right, a "PATH:" label is above a scrollable list of words: BLASTING, BOASTING, COASTING, COACTING, COACHING, COUCHING, COUGHING, SOUGHING, SOUTHING, SOOTHING, TOOTHING, TROTHING, TRITHING, WRITHING, and WRATHING. At the bottom right, three status boxes show "STEP : 31", "VISITED NODES: 6653", and "EXEC TIME: 147 ms".

**WORD LEDENG**  
word ladder solver

START WORD:  
QUIRKING

END WORD:  
WRATHING

ALGORITHM:  
UCS

SOLVE

PATH:

- BLASTING
- BOASTING
- COASTING
- COACTING
- COACHING
- COUCHING
- COUGHING
- SOUGHING
- SOUTHING
- SOOTHING
- TOOTHING
- TROTHING
- TRITHING
- WRITHING
- WRATHING

STEP : 31

VISITED NODES: 6653

EXEC TIME: 147 ms

## Algoritma GBFS:

# WORD LEDENG

word ladder solver

START WORD:

QUIRKing

END WORD:

WRATHING

ALGORITHM:

Greedy Best First Search

SOLVE

STEP :

9

VISITED NODES:

0 ms

PATH:

No path found.

### Algoritma A\*:

The screenshot shows a web application titled "WORD LEDENG word ladder solver". On the left, there are three input fields: "START WORD:" with the value "QUIRKING", "END WORD:" with the value "WRATHING", and "ALGORITHM:" with a dropdown menu showing "A\*". Below these is a pink "SOLVE" button. On the right, under the heading "PATH:", a list of words is displayed in a scrollable box: "QUIRKING", "BOASTING", "COASTING", "COACTING", "COACHING", "COUCHING", "MOUCHING", "MOUTHING", "SOUTHING", "SOOTHING", "TOOTHING", "TROTHING", "TRITHING", "WRITHING", and "WRATHING". At the bottom, there are three status boxes: "STEP : 31", "VISITED NODES: 2979", and "EXEC TIME: 68 ms".

**Keterangan:** ini adalah *ladder* dengan 8 huruf terpanjang menurut <http://datagenetics.com/>. Didapat bahwa setiap algoritma mampu memberikan solusi rute yang dihasilkan dari start word ke end word. Terlihat pada UCS dan A\* hasil rute yang diberikan adalah minimal yakni hanya 31 langkah, sementara GBFS ternyata tidak mampu menghasilkan solusi karena terjebak di lokal minima dan dari kata yang diekspan tersebut tidak dapat mencapai kata tujuan.. Dari segi efisiensi, waktu yang diperlukan oleh UCS paling lama di antara ketiganya, disusul dengan A\*. Hal ini sebanding dengan banyaknya simpul yang dikunjungi oleh masing-masing algoritma.

### 3.3 Analisis Perbandingan Solusi dari Hasil Pengujian

Berdasarkan bukti hasil tangkapan layar pengujian pada bagian sebelumnya, didapat beberapa kesamaan karakteristik dari hasil *test case* antara algoritma UCS, Greedy Best First Search, dan A\*. Dapat dibandingkan secara general dari implementasi algoritma tersebut adalah sebagai berikut.

### 1. Optimalitas:

- a. UCS: Secara teoritis, UCS menjamin solusi optimal karena mengembangkan simpul dengan biaya yang semakin meningkat. Hal ini terbukti dari setiap hasil pengujian di atas yang selalu menampilkan rute dengan langkah paling minimal.
- b. Greedy Best First Search: Algoritma Greedy Best First Search tidak menjamin solusi optimal. GBFS cenderung untuk terjebak dalam lokal optimum karena hanya mempertimbangkan heuristik dan tidak mempertimbangkan biaya sebenarnya. GBFS mengekskpan node yang "tampaknya" lebih dekat menuju tujuan namun belum tentu secara aktual. Hal ini terbukti dari hasil pengujian yang menunjukkan hampir semua kasus GBFS memberikan rute dengan langkah yang lebih panjang dibanding UCS dan A\*, atau ternyata tidak mampu memberikan rute penelusuran sama sekali.
- c. A\*: A\* menjamin solusi optimal jika heuristik yang digunakan adalah admissible. Karena heuristik  $h(n)$  pada kasus ini adalah admissible dan tetap mempertimbangkan biaya aktual  $g(n)$ , A\* akan menemukan jalur yang optimal dengan biaya minimum dari simpul awal ke simpul tujuan. Terbukti dari hasil pengujian semua kasus A\* dapat memberi rute dengan langkah minimal seperti UCS.

### 2. Waktu Eksekusi:

- a. UCS: Waktu eksekusi UCS bisa bervariasi tergantung pada kompleksitas graf dan jumlah simpul yang harus dieksplorasi. UCS harus memeriksa setiap simpul secara berurutan sesuai dengan biaya aktual, sehingga bisa membutuhkan waktu yang cukup lama, terutama jika ada banyak simpul dalam graf. Terbukti dari hasil pengujian, UCS membutuhkan waktu paling lama diantara yang lain.
- b. Greedy Best First Search: Waktu eksekusi Greedy Best First Search cenderung lebih cepat bahkan paling cepat karena hanya mempertimbangkan heuristik. Dengan heuristik ini, GBFS tidak harus memeriksa setiap simpul sesuai biayanya melainkan hanya berdasarkan heuristik yang tampak dekat dengan tujuan. Hasil pengujian membuktikan waktu yang dibutuhkan jauh lebih cepat dibanding algoritma lainnya (secara signifikan).
- c. A\*: Waktu eksekusi A\* bisa lebih efisien dibandingkan dengan UCS, namun tidak secepat GBFS. Dengan mempertimbangkan kedua faktor biaya aktual dan heuristik, A\* dapat menemukan solusi dengan jumlah simpul yang lebih sedikit tapi tetap mengusahkan waktu yang singkat, karena terbantu oleh panduan heuristiknya. Terbukti dari hasil pengujian yang menunjukkan hasil optimal walau tak memakan waktu banyak.

### 3. Memori yang Dibutuhkan:

Dengan  $b$  sebagai faktor cabang setiap simpul dan  $m$  adalah kedalaman solusi terpendek,

- a. UCS: Memori yang dibutuhkan oleh UCS tergantung pada jumlah simpul yang harus disimpan dalam priority queue untuk dieksplorasi. UCS menyimpan semua simpul yang akan diekskan sehingga memori yang dibutuhkan menjadi relatif besar, dapat menjadi  $O(b^m)$ .

- b. Greedy Best First Search: Memori yang dibutuhkan oleh Greedy Best First Search lebih kecil daripada UCS karena hanya mempertimbangkan simpul dengan heuristik terendah. Simpul yang ditelusuri algoritma ini belum "lengkap" sehingga dapat memangkas ruang pencarian dan kompleksitas ruangnya dapat menjadi polinomial.
- c. A\*: A\* memerlukan lebih banyak memori untuk menyimpan simpul dalam priority queue dibanding GBFS, tetapi karena algoritma ini mempertimbangkan kedua faktor biaya aktual dan heuristik, bisa lebih efisien dalam penggunaan memori daripada UCS. A\* tetap perlu menyimpan semua simpul saat melakukan pencarian, sehingga memori yang dibutuhkan dapat menjadi  $O(b^m)$ .

Semua perbandingan atas memori yang dibutuhkan ini dapat dilihat dari jumlah node yang dikunjungi oleh masing-masing algoritma pada tangkapan layar hasil pengujian sebelumnya.



## BAB IV LAMPIRAN

### 4.1 Link Repository

Github: [https://github.com/NopalAul/Tucil3\\_13522074](https://github.com/NopalAul/Tucil3_13522074)

### 4.2 Checklist

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

## **REFERENSI**

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>