

# **LAPORAN TUGAS KECIL III**

## **IF2211 STRATEGI ALGORITMA**

Penyelesaian Permainan Word Ladder Menggunakan

Algoritma UCS, Greedy Best First Search, dan A\*



Disusun oleh

Derwin Rustanly      13522115

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2024**

# Bab I

## Analisis dan Implementasi Algoritma

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. Pada tugas ini, akan diimplementasikan 3 jenis algoritma untuk menyelesaikan permainan Word Ladder, yakni Algoritma *Uniform Cost Search* (UCS), *Greedy Best First Search* (GBFS) dan A\*.

Algoritma UCS merupakan algoritma penentuan rute (*route planning*) yang berbasis pencarian *Uninformed Search* (*Blind Search*) di mana tidak terdapat informasi tambahan berupa data eksternal selain dari struktur graf tersebut sendiri. Pembangkitan simpul hidup berdasarkan algoritma ini memprioritaskan nilai *cost* ( $g(n)$ ) yang merupakan biaya operasi terendah dari simpul akar menuju simpul ke- $n$ . Dalam penyelesaian dari permainan Word Ladder ini, algoritma UCS diimplementasikan struktur data *priority queue* dengan langkah-langkah seperti berikut.

1. Masukkan kata awal ke dalam *priority queue* dengan cost,  $g(n) = 0$ .
2. Ambil kata paling depan dari *queue*.
3. Validasi apakah kata yang telah diambil merupakan kata tujuan. Jika ya, maka pencarian selesai. Jika tidak, cari seluruh kata yang dapat dihasilkan dari kata awal dengan cara mengganti salah satu huruf dengan mempertimbangkan adanya validasi tambahan untuk mengambil kata - kata terdapat dalam kamus bahasa Inggris.
4. Kata - kata yang dihasilkan tersebut memiliki jarak dari kata awal ( $g(n)$ ) yakni jarak kata yang dicek ke kata awal ditambah 1, karena terdapat penggantian satu huruf dari kata awal.
5. Jika kata yang dihasilkan belum pernah dicek atau kata tersebut memiliki  $g(n)$  lebih kecil dari  $g(n)$  kata yang sama pada pengecekan sebelumnya, maka masukkan kata

tersebut ke dalam antrian sesuai prioritas, dimana  $g(n)$  terkecil akan ditaruh paling depan.

6. Ulangi langkah dari langkah kedua hingga seluruh kemungkinan telah dicek atau kata target ditemukan.

Sementara itu, algoritma GBFS merupakan algoritma penentuan rute yang berbasis *Informed Search* dengan adanya estimasi heuristik ( $h(n)$ ) yang merupakan estimasi biaya yang diperlukan untuk mencapai simpul tujuan dari simpul ke- $n$ . Sama halnya dengan algoritma UCS, digunakan pula struktur data *priority queue* untuk membangkitkan simpul hidup, namun dengan prioritas simpul dengan nilai estimasi heuristik terkecil. Secara teoritis, sesuai dengan namanya yang bersifat *Greedy*, algoritma GBFS tidak menjamin ditemukannya solusi optimum global dari suatu simpul menuju simpul tujuan, karena algoritma ini seringkali terjebak dalam solusi optimum lokal, sehingga optimalitasnya tidak dapat dijamin. Dalam permainan Word Ladder, algoritma GBFS diimplementasikan dengan langkah-langkah seperti berikut.

1. Masukkan kata awal ke dalam *priority queue*.
2. Ambil kata paling depan dari *queue*.
3. Validasi apakah kata yang telah diambil merupakan kata tujuan. Jika ya, maka pencarian selesai. Jika tidak, cari seluruh kata yang dapat dihasilkan dari kata awal dengan cara mengganti salah satu huruf dengan mempertimbangkan adanya validasi tambahan untuk mengambil kata - kata terdapat dalam kamus bahasa Inggris.
4. Kata-kata yang dihasilkan memiliki nilai heuristik ( $h(n)$ ) berupa jumlah perbedaan huruf dengan kata tujuan, sebagai contoh kata BEST dan REST memiliki 1 perbedaan huruf, sehingga  $h(n)$  bernilai 1.
5. Jika kata yang dihasilkan belum pernah dicek atau kata tersebut memiliki  $h(n)$  lebih kecil dari  $h(n)$  kata yang sama pada pengecekan sebelumnya, maka masukkan kata tersebut ke dalam antrian sesuai prioritas, dimana  $h(n)$  terkecil akan ditaruh paling depan.
6. Ulangi langkah dari langkah kedua hingga seluruh kemungkinan telah dicek atau kata target ditemukan.

Kemudian, algoritma A\* merupakan kombinasi dari algoritma penentuan rute yang juga berbasis *Informed Search* yang ditentukan berdasarkan estimasi heuristik ( $h(n)$ ) dan pertimbangan nilai cost ( $g(n)$ ) yakni biaya operasi terendah dari simpul akar menuju simpul ke- $n$ . Dalam algoritma ini, perlu dipertimbangkan faktor *admissible* dari estimasi heuristik yang digunakan. Dengan kata lain, algoritma A\* yang diimplementasikan harus menggunakan estimasi heuristik ( $h(n)$ ) yang lebih rendah daripada biaya untuk mencapai simpul tujuan sebenarnya ( $h^*(n)$ ), sehingga secara matematis dapat dituliskan sebagai berikut.

$$f(n) \text{ is admissible} \leftrightarrow h(n) \leq h^*(n)$$

Algoritma A\* menjamin optimalitas dari pencarian apabila nilai heuristik yang digunakan bersifat *admissible*. Secara teoritis, algoritma A\* dengan estimasi heuristik yang *admissible* akan

cenderung lebih optimal dibandingkan dengan algoritma UCS, karena algoritma A\* menghindari ekspansi dari simpul yang terlalu “mahal” berdasarkan penjumlahan dari nilai cost dari simpul akar beserta nilai heuristik menuju simpul tujuan. Dalam permainan Word Ladder, algoritma A\* diimplementasikan dengan langkah-langkah seperti berikut.

1. Masukkan kata awal ke dalam *priority queue*.
2. Ambil kata paling depan dari *queue*.
3. Validasi apakah kata yang telah diambil merupakan kata tujuan. Jika ya, maka pencarian selesai. Jika tidak, cari seluruh kata yang dapat dihasilkan dari kata awal dengan cara mengganti salah satu huruf dengan mempertimbangkan adanya validasi tambahan untuk mengambil kata - kata terdapat dalam kamus bahasa Inggris.
4. Kata-kata yang dihasilkan memiliki nilai  $f(n)$  berupa jumlah dari  $g(n)$  yaitu jarak kata yang dicek ke kata awal ditambah 1 dan  $h(n)$  berupa perbedaan huruf dengan kata tujuan.
5. Jika kata yang dihasilkan belum pernah dicek atau kata tersebut memiliki  $f(n)$  lebih kecil dari  $f(n)$  kata yang sama pada pengecekan sebelumnya, maka masukkan kata tersebut ke dalam antrian sesuai prioritas, dimana  $f(n)$  terkecil akan ditaruh paling depan.
6. Ulangi langkah dari langkah kedua hingga seluruh kemungkinan telah dicek atau kata target ditemukan.

## Bab II

### Source Code dan Implementasi Program

Dalam implementasi penyelesaian permainan Word Ladder berbasis algoritma UCS, GBFS, dan A\* digunakan bahasa pemrograman Java. Berikut ini merupakan source code Java yang digunakan dalam implementasi program beserta penjelasan kelas dan *methodnya*.

#### 1. WorldLadderSolver.java

##### WorldLadderSolver.java

```
public class WordLadderSolver {
    private Set<String> dictionary;

    public WordLadderSolver(String dictionaryFile) throws IOException {
        loadDictionary(dictionaryFile);
    }

    private void loadDictionary(String filename) throws IOException {
        dictionary = new HashSet<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                dictionary.add(line.trim().toUpperCase());
            }
        }
        Utils.setDictionary(dictionary);
    }

    public Solution solve(String start, String end, String algorithm) {
        start = start.toUpperCase(); end = end.toUpperCase();
        if (!dictionary.contains(start) || !dictionary.contains(end)) {
            throw new IllegalArgumentException("Start or end word not in dictionary.");
        } else if (start.length() != end.length()) {
            throw new IllegalArgumentException("Start and end word must have the same length.");
        }
        Algo Solver;
        switch (algorithm.toLowerCase()) {
            case "ucs":
                Solver = new UCS();
                return Solver.solve(start, end);
            case "gbfs":
                Solver = new GBFS();
                return Solver.solve(start, end);
            case "a*":
                Solver = new Astar();
                return Solver.solve(start, end);
            default:
                throw new IllegalArgumentException("Unknown algorithm type.");
        }
    }

    public static void main(String[] args) {
        try {
            WordLadderSolver solver = new WordLadderSolver("dictionary.txt");
            long start = System.currentTimeMillis();
            Solution solution = solver.solve("idea", "plan", "UCS");
            if (solution.path.isEmpty()) {
                System.out.println("No solution found.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        } else {
            solution.path.forEach(System.out::println);
            System.out.println("Visited words: " + solution.visited_words);
        }
        long end = System.currentTimeMillis();
        System.out.println("Duration: " + (end-start) + " ms");
    } catch (IOException e) {
        System.err.println("Failed to load dictionary: " + e.getMessage());
    } catch (IllegalArgumentException e){
        System.err.println("Error: " + e.getMessage());
    }
}
}
}

```

Pada file ini, diimplementasikan kelas WordLadderSolver yang berfungsi sebagai program utama yang memiliki metode loadDictionary untuk membaca kamus berbahasa Inggris berbasis file .txt serta metode Solve yang mengembalikan objek Solution yang memiliki atribut berupa lintasan terpilih beserta jumlah kata yang dikunjungi.

## 2. WorldLadderGUI.java

### WorldLadderGUI.java

```

import javax.swing.*;
import javax.swing.text.html.HTMLEditorKit;

import java.awt.*;
import java.io.IOException;

public class WordLadderGUI extends JFrame {
    private JTextField startWordField;
    private JTextField endWordField;
    private JButton ucsButton;
    private JButton gbfsButton;
    private JButton aStarButton;
    private JTextPane resultArea;
    private WordLadderSolver solver;

    public WordLadderGUI() {
        initUI();
        solver = createSolver("dictionary.txt");
        if (solver == null) {
            System.exit(1); // Terminates the application
        }
        setTitle("Word Ladder Solver");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(700, 400); // Adjusted for better layout display
        setLocationRelativeTo(null);
        setResizable(true); // Allow resizing
    }

    private void initUI() {
        Font commonFont = new Font("Monospaced", Font.PLAIN, 16); // Create a common font object

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(1, 2, 10, 0)); // Changed to 1 row, 2 columns
        getContentPane().add(mainPanel);

        // Panel that will center the leftPanel contents vertically and horizontally
        JPanel centeringPanel = new JPanel(new GridBagLayout());
        centeringPanel.setBackground(new Color(45, 45, 45));
        mainPanel.add(centeringPanel);
    }
}

```

```

// Left Panel for inputs and buttons
JPanel leftPanel = new JPanel();
leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
leftPanel.setBackground(new Color(45, 45, 45)); // Dark background for the left panel
centeringPanel.add(leftPanel); // Add leftPanel to the centering panel

// Input panel with labels
JPanel inputPanel = new JPanel(new GridLayout(2, 2, 5, 5));
inputPanel.setBackground(new Color(45, 45, 45)); // Dark background for the input panel
JLabel startWordLabel = new JLabel("<html><font color='white'>Start Word:</font></html>");
startWordLabel.setFont(commonFont);
inputPanel.add(startWordLabel);
startWordField = new JTextField();
startWordField.setFont(commonFont);
inputPanel.add(startWordField);
JLabel endWordLabel = new JLabel("<html><font color='white'>End Word:</font></html>");
endWordLabel.setFont(commonFont);
inputPanel.add(endWordLabel);
endWordField = new JTextField();
endWordField.setFont(commonFont);
inputPanel.add(endWordField);

// Buttons panel
JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
buttonPanel.setBackground(new Color(45, 45, 45)); // Dark background for the button panel
ucsButton = new JButton("UCS");
gbfsButton = new JButton("GBFS");
aStarButton = new JButton("A*");
// Set font for buttons
ucsButton.setFont(commonFont);
gbfsButton.setFont(commonFont);
aStarButton.setFont(commonFont);
// Set foreground and background for buttons
ucsButton.setForeground(Color.WHITE);
gbfsButton.setForeground(Color.WHITE);
aStarButton.setForeground(Color.WHITE);
ucsButton.setBackground(new Color(70, 70, 70));
gbfsButton.setBackground(new Color(70, 70, 70));
aStarButton.setBackground(new Color(70, 70, 70));
buttonPanel.add(ucsButton);
buttonPanel.add(gbfsButton);
buttonPanel.add(aStarButton);

// Adding a rigid area for spacing before the input panel
leftPanel.add(Box.createVerticalGlue());
leftPanel.add(inputPanel);
leftPanel.add(buttonPanel);
leftPanel.add(Box.createVerticalGlue()); // Adding space after the button panel

// Result area
resultArea = new JTextPane();
resultArea.setEditorKit(new HTMLEditorKit());
resultArea.setContentType("text/html");
resultArea.setEditable(false);
String initialHtml = "<html><body style='color:white; background-color:rgb(70,70,70); font-family:Monospace; font-size:16pt;'>";
resultArea.setText(initialHtml);
JScrollPane scrollPane = new JScrollPane(resultArea);
mainPanel.add(scrollPane); // Add result area to the main panel as the second column

// Setup buttons with actions
ucsButton.addActionListener(e -> solve("ucs"));
gbfsButton.addActionListener(e -> solve("gbfs"));
aStarButton.addActionListener(e -> solve("a*"));
}

```

```

private void solve(String algorithm) {
    String start = startWordField.getText().trim().toUpperCase();
    String end = endWordField.getText().trim().toUpperCase();
    try {
        long startTime = System.nanoTime();
        Solution solution = solver.solve(start, end, algorithm);
        long endTime = System.nanoTime();
        displayResults(solution.visited_words, solution.path, startTime, endTime);
    } catch (Exception e) {
        resultArea.setText("<html><body style='font-family:Monospace; font-size:16pt; color:white; background-color:rgb(70,70,70);'><p>Error: " + e.getMessage() + "</p></body></html>");
    }
}

private void displayResults(int visitedNodesCount, java.util.List<String> ladder, long startTime, long endTime) {
    long duration = (endTime - startTime) / 1000000; // Convert to milliseconds

    StringBuilder sb = new StringBuilder();
    sb.append(
        "<html><body style='font-family:Monospace; font-size:16pt; color:white; background-color:rgb(70,70,70); text-align:center;'>");

    if (ladder != null && !ladder.isEmpty()) {
        sb.append(
            "<h2 style='color: #AAA; font-weight: bold; margin-bottom: 10px; border-bottom: 2px solid #666; display: inline-block; padding-bottom: 5px;'>Shortest Ladder</h2>");
        sb.append("<div style='margin:auto; width:fit-content;'>");
        sb.append("<table style='border-collapse:collapse; margin:auto;'>");
        sb.append("<tr><th style='padding: 5px;'>Step</th><th>Word</th></tr>");
        sb.append("<tr><td>1</td><td>").append(formatWord(ladder.get(0), ladder.get(0))).append("</td></tr>");
        for (int i = 0; i < ladder.size() - 1; i++) {
            sb.append("<tr><td>")
                .append(i + 2)
                .append("</td><td>")
                .append(formatWord(ladder.get(i), ladder.get(i + 1)))
                .append("</td></tr>");
        }
        sb.append("</table></div>");
    } else {
        sb.append("<p>No ladder found.</p>");
    }

    sb.append("<p style='margin-top:20px;'>Visited words:");
    sb.append(visitedNodesCount).append("</p>");
    sb.append("<p>Execution time: ").append(duration).append(" ms</p>");
    sb.append("</body></html>");

    resultArea.setText(sb.toString());
}

private String formatWord(String word1, String word2) {
    StringBuilder formatted = new StringBuilder("<table style='margin: auto;'><tr>");
    for (int i = 0; i < word1.length(); i++) {
        if (i < word2.length()) {
            String bgColor = word1.charAt(i) == word2.charAt(i) ? "lightgray" : "red";
            String color = "white";
            formatted.append("<td style='width:20px; height:20px; background-color:'")
                .append(bgColor)
                .append("; color:'")
                .append(color)
                .append("; text-align:center; border:1px solid black;'>")
                .append(word2.charAt(i))
                .append("</td>");
        }
    }
}

```



```

    }
    formatted.append("</tr></table>");
    return formatted.toString();
}

private WordLadderSolver createSolver(String dictionaryPath) {
    try {
        return new WordLadderSolver(dictionaryPath);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Failed to load dictionary: " + e.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
        return null;
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new WordLadderGUI().setVisible(true));
}
}

```

Pada file ini, diimplementasikan algoritma yang digunakan untuk membangun GUI untuk memvisualisasikan hasil penyelesaian permainan Word Ladder menggunakan kaskas bawaan Java Swing dan AWT. Secara garis besar, program akan menunggu masukan pengguna berupa kata simpul dan tujuan yang akan divalidasi menggunakan kamus Bahasa Inggris, serta *event* klik pengguna terhadap tombol dari algoritma yang dipilih. Kemudian, hasil penyelesaian akan divisualisasikan dengan tabel HTML.

### 3. Algo.java

Algo.java

```

public interface Algo {
    public Solution solve(String start, String end);
}

```

File ini berisi definisi *ABC Abstract Base Class*) dengan metode `solve` yang kemudian akan diwarisi oleh kelas Astar, UCS, dan GBFS.

### 4. UCS.java

UCS.java

```

public class UCS implements Algo{
    public Solution solve(String start, String end) {
        Queue<Node> pqueue = new PriorityQueue<>(Comparator.comparingInt(n -> n.cost));
        Map<String, Integer> visited = new HashMap<>();
        pqueue.add(new Node(start, null, 0));

        while (!pqueue.isEmpty()) {
            Node current = pqueue.poll();

            if (current.word.equals(end)) {

```

```

        List<String> path = Utils.buildPath(current);
        int visited_words = visited.size();
        return new Solution(path, visited_words);
    }

    visited.put(current.word, current.cost);

    for (String neighbor : Utils.getNeighbors(current.word)) {
        int newCost = current.cost + 1;
        if (!visited.containsKey(neighbor) || visited.get(neighbor) > newCost) {
            visited.put(neighbor, newCost);
            pqueue.add(new Node(neighbor, current, newCost));
        }
    }
}

return new Solution();
}
}

```

File ini berisi implementasi dari algoritma UCS yang memiliki method solve sebagai subkelas dari kelas Algo dengan memanfaatkan struktur data *priority queue* berbasis *cost* ( $g(n)$ ) minimal dengan penjelasan algoritma yang dapat diamati pada bagian sebelumnya.

## 5. GBFS.java

### GBFS.java

```

public class GBFS implements Algo {
    public Solution solve(String start, String end) {
        Queue<Node> pqueue = new PriorityQueue<>(Comparator.comparingInt(n -> n.heuristic));
        Set<String> visited = new HashSet<>();
        pqueue.add(new Node(start, null, 0, Utils.heuristic(start, end)));

        while (!pqueue.isEmpty()) {
            Node current = pqueue.poll();

            if (current.word.equals(end)) {
                List<String> path = Utils.buildPath(current);
                int visited_words = visited.size();
                return new Solution(path, visited_words);
            }

            visited.add(current.word);

            for (String neighbor : Utils.getNeighbors(current.word)) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    pqueue.add(new Node(neighbor, current, 0, Utils.heuristic(neighbor, end)));
                }
            }
        }
    }
}

```

```

        return new Solution();
    }
}

```

File ini berisi implementasi dari algoritma GBFS yang memiliki method solve sebagai subkelas dari kelas Algo dengan memanfaatkan struktur data *priority queue* berbasis estimasi heuristik ( $h(n)$ ) minimal dengan penjelasan algoritma yang dapat diamati pada bagian sebelumnya.

## 6. Astar.java

Astar.java

```

public class Astar implements Algo {
    public Solution solve(String start, String end) {
        Queue<Node> pqueue = new PriorityQueue<>(Comparator.comparingInt(n -> n.cost + n.heuristic));
        Map<String, Integer> visited = new HashMap<>();
        pqueue.add(new Node(start, null, 0, Utils.heuristic(start, end)));

        while (!pqueue.isEmpty()) {
            Node current = pqueue.poll();

            if (current.word.equals(end)) {
                List<String> path = Utils.buildPath(current);
                int visited_words = visited.size();
                return new Solution(path, visited_words);
            }

            visited.put(current.word, current.cost);

            for (String neighbor : Utils.getNeighbors(current.word)) {
                int newCost = current.cost + 1;
                if (!visited.containsKey(neighbor) || visited.get(neighbor) > newCost) {
                    visited.put(neighbor, newCost);
                    pqueue.add(new Node(neighbor, current, newCost, Utils.heuristic(neighbor, end)));
                }
            }
        }

        return new Solution();
    }
}

```

File ini berisi implementasi dari algoritma A\* yang memiliki method solve sebagai subkelas dari kelas Algo dengan memanfaatkan struktur data *priority queue* berbasis ( $f(n)$ ) yakni penjumlahan nilai *cost* ( $g(n)$ ) dan estimasi heuristik ( $h(n)$ ) minimal dengan penjelasan algoritma yang dapat diamati pada bagian sebelumnya.

## 7. Utils.java

### Utils.java

```
public class Utils {
    public static Set<String> dictionary;
    public static void setDictionary(Set<String> dict){
        dictionary = dict;
    }

    public static List<String> getNeighbors(String word) {
        List<String> neighbors = new ArrayList<>();
        char[] chars = word.toCharArray();
        for (int i = 0; i < chars.length; i++) {
            char oldChar = chars[i];
            for (char c = 'A'; c <= 'Z'; c++) {
                if (c != oldChar) {
                    chars[i] = c;
                    String newWord = new String(chars);
                    if (dictionary.contains(newWord)) {
                        neighbors.add(newWord);
                    }
                }
            }
            chars[i] = oldChar;
        }
        return neighbors;
    }

    public static int heuristic(String word, String target) {
        int diff = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) != target.charAt(i)) {
                diff++;
            }
        }
        return diff;
    }

    public static List<String> buildPath(Node endNode) {
        List<String> path = new LinkedList<>();
        Node current = endNode;
        while (current != null) {
            path.add(0, current.word); // Add to the front
            current = current.parent;
        }
        return path;
    }
}
```

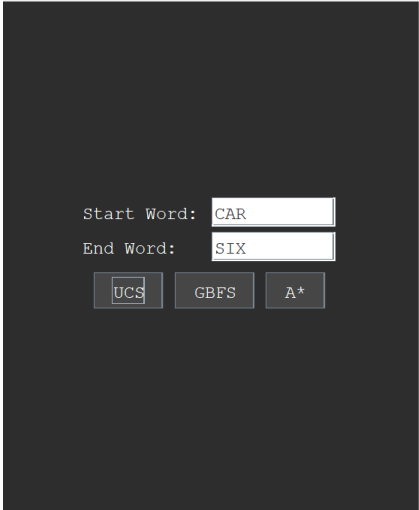
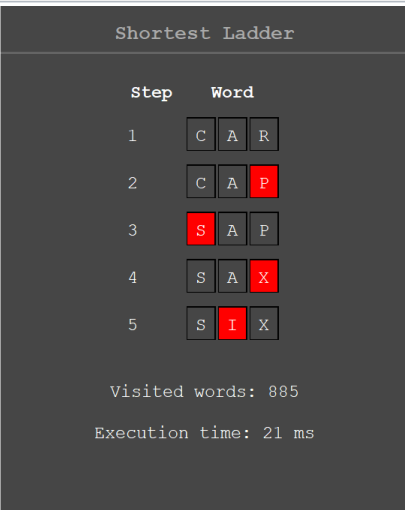
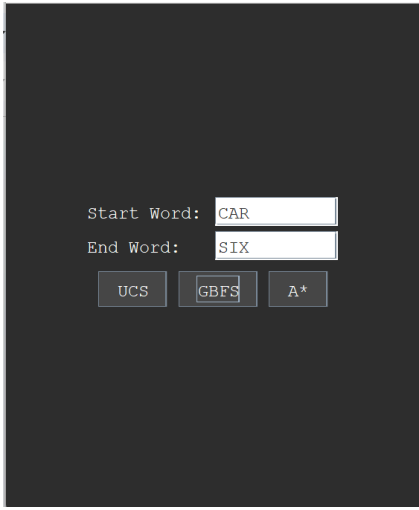
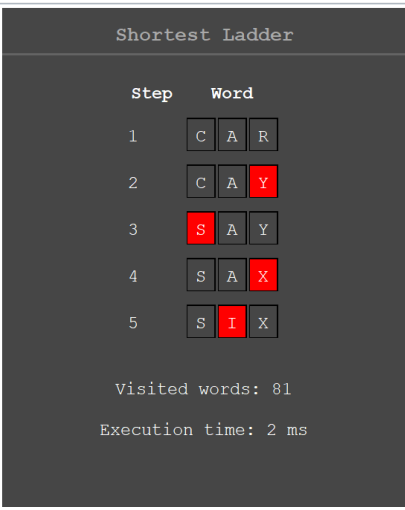
File ini berisi kumpulan metode yang digunakan untuk mengimplementasikan penyelesaian dari Word Ladder, yang terdiri dari *method* `getNeighbors` untuk membangkitkan simpul anak yang berbeda 1 huruf dengan simpul induk, serta memvalidasi simpul tersebut dalam kamus Bahasa

Inggris, *method* heuristik untuk menghitung perbedaan huruf yang dimiliki oleh suatu kata dengan kata tujuan, serta *method* buildPath untuk membangun lintasan simpul solusi dari kata awal menuju kata tujuan dengan memanfaatkan struktur data list berkait (*linked list*).

## Bab III Pengujian dan Analisis

### 1. Pengujian

- a. Pengujian dengan kata sepanjang 3 huruf

Algoritma	Hasil	
UCS		
GBFS		

A*	<div> <div> Start Word: <input type="text" value="CAR"/> End Word: <input type="text" value="SIX"/> </div> <div> <div>UCS</div> <div>GBFS</div> <div>A*</div> </div> </div> <div> Shortest Ladder <table> <tr> <th>Step</th><th>Word</th></tr> <tr> <td>1</td><td>C A R</td></tr> <tr> <td>2</td><td>C A P</td></tr> <tr> <td>3</td><td>S A P</td></tr> <tr> <td>4</td><td>S I P</td></tr> <tr> <td>5</td><td>S I X</td></tr> </table> <p> Visited words: 223  Execution time: 4 ms </p> </div>	Step	Word	1	C A R	2	C A P	3	S A P	4	S I P	5	S I X
Step	Word												
1	C A R												
2	C A P												
3	S A P												
4	S I P												
5	S I X												

b. Pengujian dengan kata sepanjang 4 huruf (a)

Algoritma	Hasil														
UCS	<div> <div> Start Word: <input type="text" value="TRIM"/> End Word: <input type="text" value="NEXT"/> </div> <div> <div>UCS</div> <div>GBFS</div> <div>A*</div> </div> </div> <div> Shortest Ladder <table> <tr> <th>Step</th><th>Word</th></tr> <tr> <td>1</td><td>T R I M</td></tr> <tr> <td>2</td><td>T R A M</td></tr> <tr> <td>3</td><td>T E A M</td></tr> <tr> <td>4</td><td>T E A T</td></tr> <tr> <td>5</td><td>T E X T</td></tr> <tr> <td>6</td><td>N E X T</td></tr> </table> <p> Visited words: 2272  Execution time: 11 ms </p> </div>	Step	Word	1	T R I M	2	T R A M	3	T E A M	4	T E A T	5	T E X T	6	N E X T
Step	Word														
1	T R I M														
2	T R A M														
3	T E A M														
4	T E A T														
5	T E X T														
6	N E X T														

<p><b>GBFS</b></p>	<div data-bbox="581 210 1006 724"> <p>Start Word: <input type="text" value="TRIM"/></p> <p>End Word: <input type="text" value="NEXT"/></p> <p> <input type="button" value="UCS"/> <input type="button" value="GBFS"/> <input type="button" value="A*"/> </p> </div> <div data-bbox="1019 210 1421 724"> <p>Shortest Ladder</p> <table> <thead> <tr> <th>Step</th> <th>Word</th> </tr> </thead> <tbody> <tr><td>1</td><td>T R I M</td></tr> <tr><td>2</td><td>B R I M</td></tr> <tr><td>3</td><td>B R I T</td></tr> <tr><td>4</td><td>B R A T</td></tr> <tr><td>5</td><td>B E A T</td></tr> <tr><td>6</td><td>N E A T</td></tr> <tr><td>7</td><td>N E X T</td></tr> </tbody> </table> <p>Visited words: 61</p> <p>Execution time: 0 ms</p> </div>	Step	Word	1	T R I M	2	B R I M	3	B R I T	4	B R A T	5	B E A T	6	N E A T	7	N E X T
Step	Word																
1	T R I M																
2	B R I M																
3	B R I T																
4	B R A T																
5	B E A T																
6	N E A T																
7	N E X T																
<p><b>A*</b></p>	<div data-bbox="581 770 1006 1285"> <p>Start Word: <input type="text" value="TRIM"/></p> <p>End Word: <input type="text" value="NEXT"/></p> <p> <input type="button" value="UCS"/> <input type="button" value="GBFS"/> <input type="button" value="A*"/> </p> </div> <div data-bbox="1019 770 1421 1285"> <p>Shortest Ladder</p> <table> <thead> <tr> <th>Step</th> <th>Word</th> </tr> </thead> <tbody> <tr><td>1</td><td>T R I M</td></tr> <tr><td>2</td><td>T R A M</td></tr> <tr><td>3</td><td>T E A M</td></tr> <tr><td>4</td><td>T E A T</td></tr> <tr><td>5</td><td>N E A T</td></tr> <tr><td>6</td><td>N E X T</td></tr> </tbody> </table> <p>Visited words: 75</p> <p>Execution time: 0 ms</p> </div>	Step	Word	1	T R I M	2	T R A M	3	T E A M	4	T E A T	5	N E A T	6	N E X T		
Step	Word																
1	T R I M																
2	T R A M																
3	T E A M																
4	T E A T																
5	N E A T																
6	N E X T																

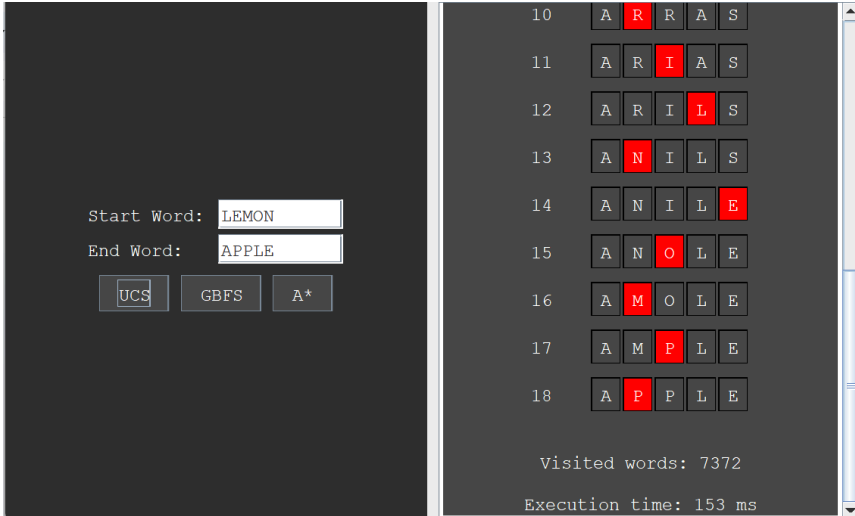
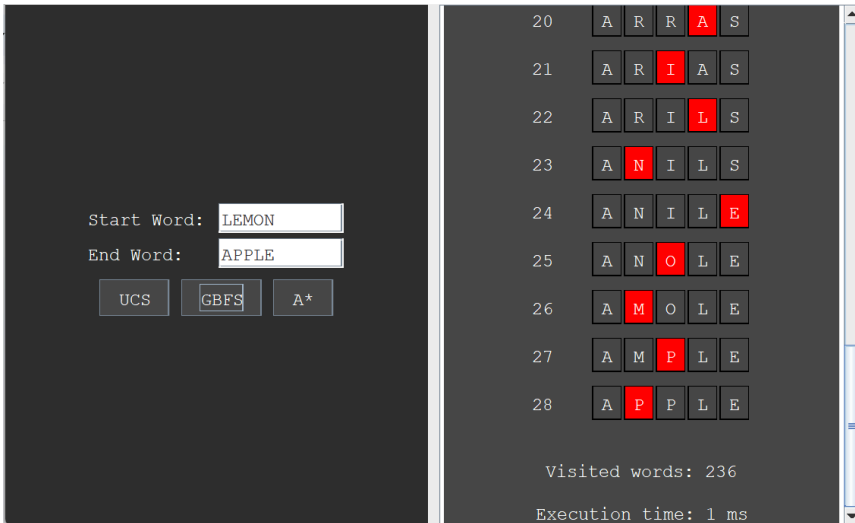
c. Pengujian dengan kata sepanjang 4 huruf (b)

<b>Algoritma</b>	<b>Hasil</b>
------------------	--------------



UCS	<div><div>Start Word: <input type="text" value="BEST"/></div><div>End Word: <input type="text" value="CASE"/></div><div><div>UCS</div><div>GBFS</div><div>A*</div></div></div>	<div><div>Shortest Ladder</div><div><div><div>Step</div><div>Word</div></div><div><div>1</div><div><div>B</div><div>E</div><div>S</div><div>T</div></div></div><div><div>2</div><div><div>B</div><div>A</div><div>S</div><div>T</div></div></div><div><div>3</div><div><div>C</div><div>A</div><div>S</div><div>T</div></div></div><div><div>4</div><div><div>C</div><div>A</div><div>S</div><div>E</div></div></div></div><div><div>Visited words: 1784</div><div>Execution time: 9 ms</div></div></div>
GBFS	<div><div>Start Word: <input type="text" value="BEST"/></div><div>End Word: <input type="text" value="CASE"/></div><div><div>UCS</div><div>GBFS</div><div>A*</div></div></div>	<div><div>Shortest Ladder</div><div><div><div>Step</div><div>Word</div></div><div><div>1</div><div><div>B</div><div>E</div><div>S</div><div>T</div></div></div><div><div>2</div><div><div>B</div><div>A</div><div>S</div><div>T</div></div></div><div><div>3</div><div><div>C</div><div>A</div><div>S</div><div>T</div></div></div><div><div>4</div><div><div>C</div><div>A</div><div>S</div><div>E</div></div></div></div><div><div>Visited words: 45</div><div>Execution time: 0 ms</div></div></div>
A*	<div><div>Start Word: <input type="text" value="BEST"/></div><div>End Word: <input type="text" value="CASE"/></div><div><div>UCS</div><div>GBFS</div><div>A*</div></div></div>	<div><div>Shortest Ladder</div><div><div><div>Step</div><div>Word</div></div><div><div>1</div><div><div>B</div><div>E</div><div>S</div><div>T</div></div></div><div><div>2</div><div><div>B</div><div>A</div><div>S</div><div>T</div></div></div><div><div>3</div><div><div>C</div><div>A</div><div>S</div><div>T</div></div></div><div><div>4</div><div><div>C</div><div>A</div><div>S</div><div>E</div></div></div></div><div><div>Visited words: 58</div><div>Execution time: 0 ms</div></div></div>

d. Pengujian dengan kata sepanjang 5 huruf

Algoritma	Hasil
UCS	 <p>Start Word: LEMON End Word: APPLE</p> <p>UCS GBFS A*</p> <p>Visited words: 7372 Execution time: 153 ms</p>
GBFS	 <p>Start Word: LEMON End Word: APPLE</p> <p>UCS GBFS A*</p> <p>Visited words: 236 Execution time: 1 ms</p>

A*	<div> <div> <div>Start Word: LEMON</div> <div>End Word: APPLE</div> <div> <div>UCS</div> <div>GBFS</div> <div>A*</div> </div> </div> <div> <div>10 A R R A S</div> <div>11 A R I A S</div> <div>12 A R I L S</div> <div>13 A N I L S</div> <div>14 A N I L E</div> <div>15 A N O L E</div> <div>16 A M O L E</div> <div>17 A M P L E</div> <div>18 A P P L E</div> <div>Visited words: 7213</div> <div>Execution time: 147 ms</div> </div> </div>
----	---

e. Pengujian dengan kata sepanjang 6 huruf

Algoritma	Hasil
UCS	<div> <div> <div>Start Word: CHARGE</div> <div>End Word: COMEDY</div> <div> <div>UCS</div> <div>GBFS</div> <div>A*</div> </div> </div> <div> <div>14 C O N I N S</div> <div>15 C O N I N G</div> <div>16 H O N I N G</div> <div>17 H O M I N G</div> <div>18 H O M I N Y</div> <div>19 H O M I L Y</div> <div>20 H O M E L Y</div> <div>21 C O M E L Y</div> <div>22 C O M E D Y</div> <div>Visited words: 8294</div> <div>Execution time: 166 ms</div> </div> </div>

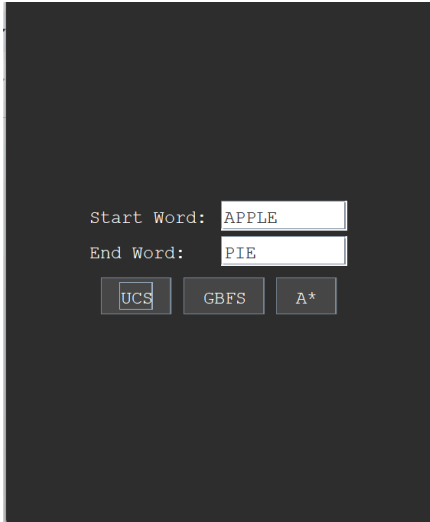
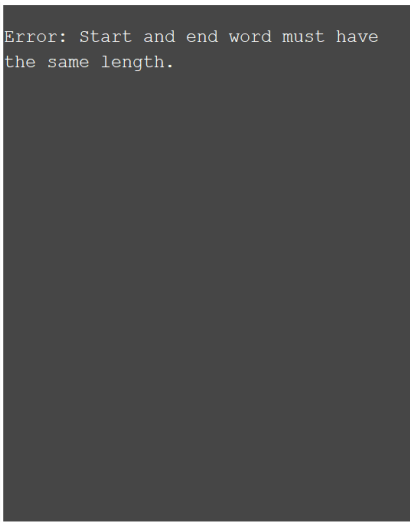
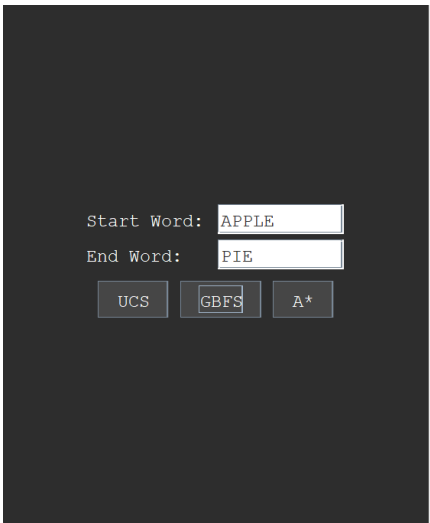
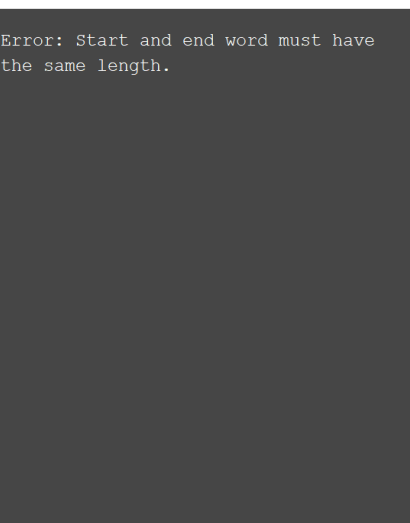
<b>GBFS</b>	<div data-bbox="581 210 1003 730"> <p>Start Word: <input type="text" value="CHARGE"/></p> <p>End Word: <input type="text" value="COMEDY"/></p> <p> <input type="button" value="UCS"/> <input type="button" value="GBFS"/> <input type="button" value="A*"/> </p> </div> <div data-bbox="1019 210 1425 730"> <p>20 C O V I N S</p> <p>21 C O V I N G</p> <p>22 C O M I N G</p> <p>23 H O M I N G</p> <p>24 H O M I N Y</p> <p>25 H O M I L Y</p> <p>26 H O M E L Y</p> <p>27 C O M E L Y</p> <p>28 C O M E D Y</p> <p>Visited words: 358</p> <p>Execution time: 5 ms</p> </div>
<b>A*</b>	<div data-bbox="581 766 1003 1291"> <p>Start Word: <input type="text" value="CHARGE"/></p> <p>End Word: <input type="text" value="COMEDY"/></p> <p> <input type="button" value="UCS"/> <input type="button" value="GBFS"/> <input type="button" value="A*"/> </p> </div> <div data-bbox="1019 766 1425 1291"> <p>14 C O N I N S</p> <p>15 C O N I N G</p> <p>16 C O M I N G</p> <p>17 H O M I N G</p> <p>18 H O M I N Y</p> <p>19 H O M I L Y</p> <p>20 H O M E L Y</p> <p>21 C O M E L Y</p> <p>22 C O M E D Y</p> <p>Visited words: 7178</p> <p>Execution time: 141 ms</p> </div>

f. Pengujian error handling (Kata Kosong)

Algoritma	Hasil
-----------	-------

<p><b>UCS</b></p>	<p>Start Word: <input type="text"/></p> <p>End Word: <input type="text"/></p> <p> <input checked="" type="button" value="UCS"/> <input type="button" value="GBFS"/> <input type="button" value="A*"/> </p>	<p>Error: Start or end word not in dictionary.</p>
<p><b>GBFS</b></p>	<p>Start Word: <input type="text"/></p> <p>End Word: <input type="text"/></p> <p> <input type="button" value="UCS"/> <input checked="" type="button" value="GBFS"/> <input type="button" value="A*"/> </p>	<p>Error: Start or end word not in dictionary.</p>
<p><b>A*</b></p>	<p>Start Word: <input type="text"/></p> <p>End Word: <input type="text"/></p> <p> <input type="button" value="UCS"/> <input type="button" value="GBFS"/> <input checked="" type="button" value="A*"/> </p>	<p>Error: Start or end word not in dictionary.</p>

- g. Pengujian error handling (Kata dengan panjang berbeda)

Algoritma	Hasil	
UCS		
GBFS		

A*	<div style="background-color: #2e3436; color: white; padding: 20px; border-radius: 10px;"> <div style="display: flex; justify-content: space-between;"> <div> <p>Start Word: <input style="background-color: white; color: black;" type="text" value="APPLE"/></p> <p>End Word: <input style="background-color: white; color: black;" type="text" value="PIE"/></p> </div> <div> <p>Error: Start and end word must have the same length.</p> </div> </div> <div style="display: flex; justify-content: center; margin-top: 10px;"> <div style="border: 1px solid white; padding: 5px 10px; margin: 0 5px;">UCS</div> <div style="border: 1px solid white; padding: 5px 10px; margin: 0 5px;">GBFS</div> <div style="border: 1px solid white; padding: 5px 10px; margin: 0 5px;">A*</div> </div> </div>
----	--

## 2. Analisis

Berdasarkan hasil pengujian pada bagian sebelumnya, dapat diamati perbedaan optimalitas, waktu eksekusi dan penggunaan memori dari ketiga algoritma yang digunakan untuk menyelesaikan permainan Word Ladder.

Pada algoritma UCS, pembangkitan simpul anak dilakukan berdasarkan prioritas nilai cost ( $g(n)$ ) yang merupakan jarak dari kata asal menuju kata yang sedang diperiksa. Dalam hal ini, perbedaan cost diantara suatu simpul induk dan simpul anak adalah 1, karena perbedaan yang dianggap valid oleh permainan ini adalah hanya 1 huruf di antara 2 kata yang berdekatan. Secara tidak langsung, algoritma UCS sama halnya dengan algoritma BFS (*Breadth-first Search*) karena seluruh simpul anak dengan nilai  $g(n)$  yang lebih rendah akan diperiksa terlebih dahulu sebelum simpul anak dengan nilai  $g(n)$  yang lebih tinggi. Dengan kata lain, UCS akan berperilaku mirip dengan BFS karena kedua algoritma tersebut akan menjelajahi semua kemungkinan perubahan satu huruf pada kedalaman yang sama sebelum beralih ke kedalaman berikutnya. Oleh karena itu, sama halnya dengan BFS, algoritma UCS dapat dijamin optimalitasnya, namun cukup boros dalam kompleksitas waktu dan ruang karena penjelajahannya yang bersifat menyeluruh sehingga membutuhkan waktu dan ruang untuk melakukan validasi yang lebih banyak.

Pada algoritma GBFS, pembangkitan simpul anak dilakukan berdasarkan prioritas estimasi heuristik ( $h(n)$ ) yang merupakan jumlah perbedaan huruf kata yang sedang diperiksa dengan kata tujuan. Dalam implementasinya, GBFS dapat memiliki waktu

eksekusi dan penggunaan memori yang lebih sedikit dibandingkan kedua algoritma lainnya, karena sifat natural dari algoritma ini yang “*greedy*” sehingga simpul yang dipilih dan diperiksa hanya berdasarkan estimasi heuristik yang “semata-mata” lebih mendekati simpul tujuan, namun pada akhirnya membutuhkan lebih banyak langkah untuk mencapai simpul tujuan, sehingga solusi optimum globalnya tidak terjamin.

Pada algoritma A\*, pembangkitan simpul anak dilakukan berdasarkan kombinasi prioritas dari algoritma UCS dan GBFS yakni nilai  $f(n) = g(n) + h(n)$ , yang menggabungkan perbedaan biaya operasi dari simpul akar menuju simpul ke-n dan estimasi nilai biaya dari simpul ke-n menuju simpul tujuan. Dalam hal ini, penggunaan nilai estimasi heuristik ( $h(n)$ ) bersifat *admissible* karena jumlah langkah yang diperlukan untuk mencapai simpul tujuan dari simpul ke-n sekurang-kurangnya adalah sejumlah perbedaan huruf yang dimiliki kedua kata tersebut. Dengan demikian, sebagaimana telah diuraikan pada bab I, solusi yang dihasilkan oleh algoritma A\* dijamin optimal dengan faktor *admissible* yang telah dipenuhi. Selain itu, algoritma A\* juga memiliki waktu eksekusi dan penggunaan memori yang lebih baik daripada algoritma UCS karena algoritma ini memanfaatkan nilai  $h(n)$  untuk menemukan solusi yang lebih efisien dengan mengurangi pembangkitan simpul-simpul yang terlalu “mahal” dari segi  $f(n)$ , sementara algoritma UCS membangkitkan seluruh simpul tanpa mempertimbangkan jarak dari simpul yang dibangkitkan menuju simpul tujuan, sehingga menggunakan waktu dan memori yang lebih banyak untuk membangkitkan simpul dan memvalidasinya.

### 3. Penjelasan Bonus

Pada program ini, diimplementasikan fitur Graphical User Interface (GUI) untuk memvisualisasikan hasil permainan Word Ladder dengan memanfaatkan kakas Java AWT dan Java Swing dengan penjelasan teknis yang dapat diamati pada bab 2 bagian WorldLadderGUI.java.



## Lampiran

Pranala menuju Repositori: [https://github.com/DerwinRustanly/Tucil3\\_13522115](https://github.com/DerwinRustanly/Tucil3_13522115)

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. <b>[Bonus]:</b> Program memiliki tampilan GUI	✓	