

**Laporan Tugas Kecil 3 IF2211 Strategi Algoritma  
Semester II Tahun Akademik 2023/2024**

**Penyelesaian Permainan Word Ladder Menggunakan  
Algoritma UCS, Greedy Best First Search, dan A\***



Disusun Oleh:  
Edbert Eddyson Gunawan - 13522039  
K-01

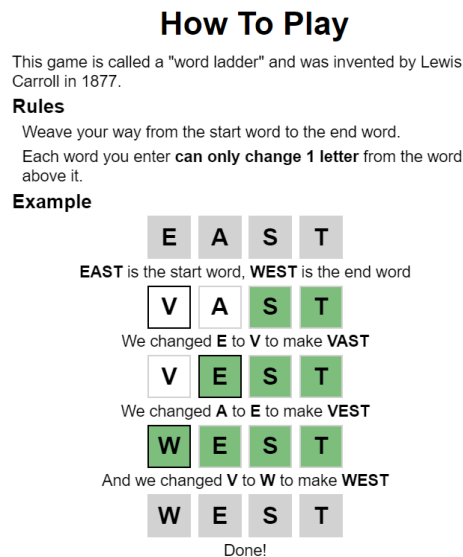
**Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2024**

# Daftar Isi

Daftar Isi.....	1
1. Deskripsi Tugas.....	2
2. Struktur Data.....	3
2.1 Node.....	3
2.2 Graph.....	3
2.3 Priority Queue.....	3
3. Algoritma Uniform Cost Search (UCS).....	4
3.1 Analisi Uniform Cost Search (UCS).....	4
3.2 Langkah-Langkah Uniform Cost Search (UCS).....	4
4. Algoritma Greedy Best First Search (GBFS).....	5
4.1 Analisis Greedy Best First Search (GBFS).....	5
4.2 Langkah-Langkah Greedy Best First Search (GBFS).....	5
5. Algoritma A*.....	6
5.1 Analisis A*.....	6
5.2 Langkah-Langkah A*.....	6
6. Source Code.....	8
6.1 [BONUS] - GUI.....	10
6.2 Main App.....	20
6.3 Data Handling.....	21
6.4 Data Structure.....	25
6.5 Algorithm.....	27
6.5.1 Algorithm.....	27
6.5.2 AlgorithmHandler.....	28
6.5.3 UCS.....	30
6.5.4 GBFS.....	32
6.5.5 A*.....	34
7. How To Use.....	36
8. Test Case.....	37
8.1 Test Case 1   toon → plea.....	37
8.2 Test Case 2   charge → comedo.....	38
8.3 Test Case 3   sex → gay.....	39
8.4 Test Case 4   flying → create.....	40
8.5 Test Case 5   atlases → cabaret.....	41
8.6 Test Case 6   boyish → painch.....	42
8.7 Test Case 7   Error Handling.....	43
8.8 Analisis Hasil Test Case.....	44
GitHub.....	45
Poin-Poin Penting.....	45

# 1. Deskripsi Tugas

*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.



**Gambar 1.** Ilustrasi dan Peraturan Permainan *World Ladder*  
(Sumber: <https://wordwormdormdork.com/>)

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

## 2. Struktur Data

Pada tugas ini, terdapat 3 struktur data yang cukup penting, *Priority Queue*, *Node* dan *Graph*.

### 2.1 Node

Untuk merepresentasikan *graph* maka kita memiliki sebuah class *Node*. Berikut rincian setiap atribut dan method.

- info: berisi kata yang berada pada kamus
- parent: merupakan reference kepada parent dari node tersebut
- depth: kedalaman dari simpul akar, sekaligus menjadi nilai dari fungsi  $g(n)$ .
- getPath: merupakan fungsi untuk mendapatkan jalur dari simpul akar menuju simpul tersebut.
- getHeuristic: fungsi untuk menghitung nilai dari  $h(n)$  dengan membandingkan info dari simpul tersebut dengan string tujuan.

```
Class Node
    String info
    Node parent
    Int depth
    getPath()
    getHeuristic()
```

### 2.2 Graph

Pada tugas ini, penulis berusaha untuk membangun graph tak berarah dari kamus bahasa inggris terlebih dahulu. Untuk merepresentasikan graph tak berarah tersebut, penulis menggunakan *Adjacency List*. Hasil dari graph ini disimpan pada file dengan nama graphMap pada folder test. Tujuannya agar tidak perlu menunggu waktu lama untuk membangkitkan kembali seluruh node, sehingga cukup di load dari saved file.

### 2.3 Priority Queue

Pada tugas ini, penulis menggunakan *Priority Queue* untuk menyimpan daftar simpul yang akan ditelusuri / *expand*. Implementasi dari *Priority Queue* ini menggunakan struktur data bawaan dari Java yakni kelas *PriorityQueue*.

### 3. Algoritma *Uniform Cost Search* (UCS)

#### 3.1 Analisi *Uniform Cost Search* (UCS)

Algoritma UCS menggunakan nilai  $g(n)$  untuk menentukan biaya dari setiap Simpul. Maka dari itu, pada laporan ini,  $g(n)$  didefinisikan sebagai jarak dari simpul akar menuju simpul  $n$ . Jarak dari simpul akar menuju simpul  $n$  ini merupakan jumlah huruf yang berbeda antara simpul akar dan simpul  $n$ .

Kasus khusus dapat kita temukan pada algoritma UCS dari permainan *Word Ladder* ini yakni UCS akan memiliki *behaviour* yang sama dengan BFS. Hal ini karena untuk perhitungan nilai  $g(n)$  berdasarkan kedalaman.

#### 3.2 Langkah-Langkah *Uniform Cost Search* (UCS)

Berikut langkah-langkah penyelesaian

1. Pertama pilih simpul akar. Lalu cek apakah dia simpul tujuan atau bukan.
2. Jika merupakan simpul tujuan, langsung kembalikan larik kosong. Dikarenakan pada laporan ini penulis menggunakan definisi, tidak terdapat jalur yang mungkin jika simpul akar dan tujuan sama, karena tidak ada simpul antara.
3. Jika bukan merupakan simpul tujuan, kita masukkan simpul akar dalam sebuah *Priority Queue*.
4. Kita ambil elemen paling depan dari *Priority Queue* misal "A" dan kita berikan tanda bahwa simpul sudah pernah dikunjungi.. maka kita cari simpul-simpul yang bertetangga dengan simpul A tersebut. Untuk setiap simpul pada larik simpul yang bertetangga ini, kita ubah nilai parent-nya menjadi "A" lalu dimasukkan ke dalam *Priority Queue* jika belum pernah dikunjungi.
5. Ulangi proses 3, dan 4 hingga kita menemukan simpul tujuan, atau *Priority Queue* sudah kosong.
6. Jika simpul tujuan ditemukan, maka kita akan berhenti dari perulangan dan menyimpan jalur yang didapatkan dengan memanggil fungsi path dari simpul.
7. Jika tidak, maka tidak ditemukan jalur dari simpul akar ke simpul tujuan.

## 4. Algoritma *Greedy Best First Search* (GBFS)

### 4.1 Analisis *Greedy Best First Search* (GBFS)

Algoritma GBFS menggunakan nilai dari fungsi heuristik  $h(n)$  untuk menentukan biaya dari setiap simpul. Maka dari itu, pada laporan ini,  $h(n)$  didefinisikan sebagai banyaknya huruf yang berbeda antara simpul  $n$  dengan simpul tujuan.

Ciri khas dari GBFS adalah dengan mengambil jalur dengan simpul pertama dengan biaya terendah pada simpul yang bertetangga. Hal ini menyebabkan GBFS *not complete*, yang berarti GBFS ada kemungkinan tidak dapat mencapai tujuan. Selain itu GBFS juga dapat terjebak dalam *local optimum*. Sehingga jalur yang dihasilkan belum tentu paling sangkil. Selain itu, karena GBFS hanya mengambil simpul pertama, maka GBFS tidak dapat mundur atau kembali, *irrevocable*.

### 4.2 Langkah-Langkah *Greedy Best First Search* (GBFS)

Berikut langkah-langkah penyelesaian

1. Pertama pilih simpul akar. Lalu cek apakah dia simpul tujuan atau bukan.
2. Jika merupakan simpul tujuan, langsung kembalikan larik kosong. Dikarenakan pada laporan ini penulis menggunakan definisi, tidak terdapat jalur yang mungkin jika simpul akar dan tujuan sama, karena tidak ada simpul antara.
3. Jika bukan merupakan simpul tujuan, kita masukkan simpul akar dalam sebuah *Priority Queue*.
4. Kita ambil elemen paling depan dari *Priority Queue* misal "A" lalu kita tandai bahwa simpul sudah pernah dikunjungi. Lalu kita hapus isi dari *Priority Queue*, dengan tujuan agar hanya 1 jalan yang dipilih setiap waktu. Kemudian kita cari simpul-simpul yang bertetangga dengan simpul akar. Larik simpul yang bertetangga ini kita ubah nilai parent-nya menjadi elemen yang sudah diambil dan dimasukkan ke dalam *Priority Queue* jika belum pernah dikunjungi.
5. *Priority Queue* akan secara otomatis mengurutkan simpul berdasarkan fungsi  $h(n)$ .
6. Ulangi langkah 3 4 5 hingga kita menemukan simpul tujuan, atau *Priority Queue* kosong.
7. Jika simpul tujuan ditemukan, maka kita akan berhenti dari perulangan dan menyimpan jalur yang didapatkan dengan memanggil fungsi path dari simpul.
8. Jika tidak, maka tidak ditemukan jalur dari simpul akar ke simpul tujuan.

## 5. Algoritma A\*

### 5.1 Analisis A\*

Algoritma A\* menggunakan fungsi  $f(n)$  untuk menentukan biaya dari setiap simpul. Fungsi  $f(n)$  merupakan penjumlahan dari fungsi  $g(n)$  dan  $h(n)$ . Dimana fungsi  $g(n)$  menghasilkan jarak dari simpul akar menuju simpul  $n$ , dan fungsi heuristik  $h(n)$  menghasilkan banyaknya huruf yang berbeda antara simpul  $n$  dengan simpul tujuan.

Pada kasus *World Ladder* fungsi heuristik  $h(n)$  tidak pernah *overestimate*. Hal ini karena banyak langkah dari simpul  $n$  ke simpul tujuan yang ditebak / *heuristic* tidak pernah melebihi banyak langkah sebenarnya. Sehingga dapat disimpulkan fungsi heuristik  $h(n)$  *Admissible*.

Selain itu, karena algoritma A\* menggunakan fungsi  $g(n)$  dan  $h(n)$  maka algoritma tersebut pada kasus umum hanya akan membangkitkan simpul-simpul yang menuju simpul tujuan, tidak keseluruhan simpul. Sedangkan algoritma UCS akan membangkitkan seluruh simpul-simpul yang ada, hingga ditemukan simpul tujuan.

### 5.2 Langkah-Langkah A\*

Berikut langkah-langkah penyelesaian

1. Pertama pilih simpul akar. Lalu cek apakah dia simpul tujuan atau bukan.
2. Jika merupakan simpul tujuan, langsung kembalikan larik kosong. Dikarenakan pada laporan ini penulis menggunakan definisi, tidak terdapat jalur yang mungkin jika simpul akar dan tujuan sama, karena tidak ada simpul antara.
3. Jika bukan merupakan simpul tujuan, kita masukkan simpul akar dalam sebuah *Priority Queue*.
4. Kita ambil elemen paling depan dari *Priority Queue* misal "A" dan kita berikan tanda bahwa simpul sudah pernah dikunjungi.. maka kita cari simpul-simpul yang bertetangga dengan simpul A tersebut. Untuk setiap simpul pada larik simpul yang bertetangga ini, kita ubah nilai parent-nya menjadi "A" lalu dimasukkan ke dalam *Priority Queue*.
5. *Priority Queue* akan secara otomatis mengurutkan simpul berdasarkan fungsi  $f(n)$ .
6. Ulangi langkah 3 4 5 hingga kita menemukan simpul tujuan, atau *Priority Queue* kosong.

7. Jika simpul tujuan ditemukan, maka kita akan berhenti dari perulangan dan menyimpan jalur yang didapatkan dengan memanggil fungsi path dari simpul.
8. Jika tidak, maka tidak ditemukan jalur dari simpul akar ke simpul tujuan.



## 6. Source Code

Penulis membagi file menjadi 4 bagian utama yaitu App utama, Algoritma, Data Handling, GUI. Struktur folder sebagai berikut

```
/Tucil3_13522039
.
├── LICENSE
├── Makefile
├── README.md
├── bin
│   ├── App.class
│   ├── algorithm
│   │   ├── Algohandler.class
│   │   ├── Algorithm.class
│   │   ├── Astar$PQComparator.class
│   │   ├── Astar.class
│   │   ├── GBFS$PQComparator.class
│   │   ├── GBFS.class
│   │   └── UCS.class
│   ├── datahandling
│   │   └── Datahandling.class
│   ├── datastructure
│   │   └── Node.class
│   ├── filereader
│   │   └── Filereader.class
│   └── gui
│       ├── Gui.class
│       ├── MyButtonListener.class
│       ├── MyRadioListener.class
│       └── MyWindowListener.class
└── src
    ├── App.java
    ├── algorithm
    │   ├── Algohandler.java
    │   ├── Algorithm.java
    │   ├── Astar.java
    │   ├── GBFS.java
    │   └── UCS.java
    ├── datahandling
    │   └── Datahandling.java
    ├── datastructure
    │   └── Node.java
    └── gui
```

```
|      └─ Gui.java
└─ test
    ├── dictionary
    ├── dictionary.txt
    └─ graphMap
```

## 6.1 [BONUS] - GUI

Pada tugas ini penulis mengimplementasikan *Graphical User Interface* (GUI) dengan menggunakan pustaka SWING. Kelas GUI ini akan dipanggil dari kelas App.

```
package gui;

import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;

import javax.swing.*;
import javax.swing.GroupLayout.Alignment;
import javax.xml.crypto.Data;

import datahandling.Datahandling;
import algorithm.*;

class MyWindowListener extends WindowAdapter {
    @Override
    public void windowClosing(WindowEvent e) {
        // TODO Auto-generated method stub
        System.err.println("closing called");
        System.exit(0);
    }
}

class MyRadioListener implements ItemListener {
    JRadioButton radioUCS;
    JRadioButton radioGBFS;
    JRadioButton radioAStar;
    JTextField hiddenField;
```

```

        public MyRadioListener(JRadioButton radioUCS, JRadioButton
radioGBFS, JRadioButton radioAStar, JTextField hiddenField){
            this.radioUCS = radioUCS;
            this.radioGBFS = radioGBFS;
            this.radioAStar = radioAStar;
            this.hiddenField = hiddenField;
        }

@Override
public void itemStateChanged(ItemEvent e) {
    // TODO Auto-generated method stub
    if (e.getSource() == this.radioUCS) {
        if (e.getStateChange() == 1) {
            this.hiddenField.setText("1");
        }
    }
    else if (e.getSource() == this.radioGBFS) {
        if (e.getStateChange() == 1) {
            this.hiddenField.setText("2");
        }
    }
    else if (e.getSource() == this.radioAStar) {
        if (e.getStateChange() == 1) {
            this.hiddenField.setText("3");
        }
    }
    else {
        this.hiddenField.setText("0");
    }
}
}

class MyButtonListener implements ActionListener {
    JTextField textField1;
    JTextField textField2;
    JTextField hField;
    JTextArea textArea;
    JLabel timeLabel;
    JLabel nodeVisitedLabel;
    JLabel memoryLabel;
    Datahandling datahandling;
}

```

```

    public MyButtonListener(JTextField t1, JTextField t2, JTextArea t3,
Datahandling dh, JTextField hiddenField, JLabel time, JLabel mem,
JLabel node){
        this.textField1 = t1;
        this.textField2 = t2;
        this.textArea = t3;
        this.datahandling = dh;
        this.hField = hiddenField;

        this.timeLabel = time;
        this.memoryLabel = mem;
        this.nodeVisitedLabel = node;
    }

    public boolean verifyWord(String s){
        return datahandling.checkWord(s);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // clear textArea
        this.textArea.setText(null);

        String val1 = this.textField1.getText().toLowerCase();
        String val2 = this.textField2.getText().toLowerCase();
        String val3 = this.hField.getText().toLowerCase();
        System.err.println(val1);
        System.err.println(val2);
        System.err.println(val3);

        // JOptionPane.showMessageDialog(null, "Value: " + val1 +
"\nValue2: " + val2); // Display the value
        if(!verifyWord(val1)){
            this.textArea.setText("INVALID INPUT 1");
            this.textArea.setForeground(Color.red);
            return;
        }

        if(!verifyWord(val2)){
            this.textArea.setText("INVALID INPUT 2");
            this.textArea.setForeground(Color.red);
            return;
        }
    }

```

```

        if(val3.equals("0")){
            this.textArea.setText("INVALID ALGORITHM");
            this.textArea.setForeground(Color.red);
            return;
        }

        if(val1.length() != val2.length()){
            this.textArea.setText("DIFFERENT INPUT LENGTH");
            this.textArea.setForeground(Color.red);
            return;
        }

        // call algorithm handler
        AlgoHandler algo = new AlgoHandler();
        ArrayList<String> pathResult = new ArrayList<String>();
        pathResult = algo.solve(val1, val2, val3, this.datahandling);

        if(pathResult.size() == 0){
            this.textArea.setText("NO PATH FOUND");
            this.textArea.setForeground(Color.red);
        } else {
            String ans = "";
            for(int i=0; i<pathResult.size(); i++){
                ans = pathResult.get(i) + "\n" + ans;
            }

            this.textArea.setText(ans);
            this.textArea.setForeground(Color.black);
        }

        this.memoryLabel.setText("memory used: " +
(algo.memoryUsed/1000) + "kb");
        this.nodeVisitedLabel.setText("node visited: " +
algo.nodeVisited + " | ");
        this.timeLabel.setText("time executed: " + algo.timeExecuted +
"ms" + " | ");
    }
}

/**
 * Class to handle all GUI for the application
 * @author WazeAzure
 */

```

```

public class Gui {
    /** frame width in px */
    private int width;
    /** frame height in px*/
    private int height;
    /** default frame width in px {@code 500px} */
    public static int DEFAULT_WIDTH = 500;
    /** default frame width in px {@code 1000px} */
    public static int DEFAULT_HEIGHT = 750;
    /** default column width for text field input {@code 20} */
    public static int DEFAULT_COLUMN_TEXT_FIELD = 20;
    /** program title */
    private String title;
    /** main frame */
    JFrame frame;

    private Datahandling dh;

    MyButtonListener buttonListener;

    /**
     * initialize a GUI with default value width {@code 500px} and
height {@code 1000px}
     */
    public Gui(Datahandling f){
        this.width = DEFAULT_WIDTH;
        this.height = DEFAULT_HEIGHT;
        this.title = "World Ladder Solver - WazeAzure";
        this.dh = f;

        // main frame
        this.frame = new JFrame();

    }
    /**
     * initialize a GUI.
     * @param width - frame width in px
     * @param height - frame height in px
     */
    public Gui(int width, int height){
        this.width = width;
        this.height = height;
    }
}

```

```

        this.title = "World Ladder Solver - WazeAzure";

        // main frame
        this.frame = new JFrame();
    }
    /**
     * Function to build the main frame for the gui
     * @param args
     */
    public void main(){
        addListener();

        // set title
        frame.setTitle(this.title);
        // set frame size
        frame.setSize(this.width, this.height);

        // create top part
        createTopRegion();
        // create middle content
        createMiddleRegion();
        // create bottom part
        createBottomRegion();

        // show frame
        frame.setVisible(true);

        System.out.println("FROM gui");
    }

    public void createTopRegion(){
        // create panel
        JPanel top = createPanel();
        top.setBackground(new Color(18, 84, 136));

        // create text & insert to panel
        JLabel t = createLabel("Let's Solve Your Game!", Color.white,
top, JLabel.CENTER);
        top.add(t, BorderLayout.CENTER);

        // add panel to frame
        this.frame.getContentPane().add(top, BorderLayout.PAGE_START);
    }

```



```

public void addListener(){
    MyWindowListener s = new MyWindowListener();
    this.frame.addWindowListener(s);
}

public void createMiddleRegion(){
    // create panel
    JPanel middle = createPanel();
    middle.setBackground(new Color(173, 217, 216));
    // middle.setLayout(new BoxLayout(middle, BoxLayout.Y_AXIS));

    // create first text field
    JTextField t1 = createTextField(20, new Font("Arial",
Font.PLAIN, 20));
    t1.setHorizontalAlignment(JTextField.CENTER);
    t1.setSize(50, 30);
    middle.add(t1);

    // create scrolling region
    JTextArea textArea = new JTextArea(20, 20);
    textArea.setFont(new Font("Arial", Font.PLAIN, 20));
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);
    middle.add(scrollPane);

    // create second text field
    JTextField t2 = createTextField(20, new Font("Arial",
Font.PLAIN, 20));
    t2.setHorizontalAlignment(JTextField.CENTER);
    t2.setSize(50, 30);
    middle.add(t2);

    // create algorithm option
    JPanel radios = createPanel();
    JTextField hiddenField = new JTextField("0");
    ButtonGroup radioButtonGroup = new ButtonGroup();
    JRadioButton radioUCS = new JRadioButton("UCS");
    JRadioButton radioGBFS = new JRadioButton("GBFS");
    JRadioButton radioAStar = new JRadioButton("A*");

```

```

        radioUCS.addItemListener(new MyRadioListener(radioUCS,
radioGBFS, radioAStar, hiddenField));
        radioGBFS.addItemListener(new MyRadioListener(radioUCS,
radioGBFS, radioAStar, hiddenField));
        radioAStar.addItemListener(new MyRadioListener(radioUCS,
radioGBFS, radioAStar, hiddenField));

        radioButtonGroup.add(radioUCS);
        radioButtonGroup.add(radioGBFS);
        radioButtonGroup.add(radioAStar);

        radios.add(radioUCS);
        radios.add(radioGBFS);
        radios.add(radioAStar);

        middle.add(radios);

        // add time exection, visited node, and memory used
        JPanel information = createPanel();
        JLabel timeLabel = new JLabel();
        JLabel memoryLabel = new JLabel();
        JLabel nodeVisitedLabel = new JLabel();

        information.add(timeLabel);
        information.add(nodeVisitedLabel);
        information.add(memoryLabel);

        // set button listener
        buttonListener = new MyButtonListener(t1, t2, textArea,
this.dh, hiddenField, timeLabel, memoryLabel, nodeVisitedLabel);

        // create submit button
        JButton submit = new JButton("SOLVE");
        submit.setFont(new Font("Arial", Font.PLAIN, 20));
        submit.setHorizontalAlignment(JButton.CENTER);
        submit.setPreferredSize(new Dimension(200, 30));

        submit.addActionListener(buttonListener);
        middle.add(submit);

```

```

        // add button listener
        this.frame.getRootPane().setDefaultButton(submit);

        // add info to panel
        middle.add(information);

        // add panel to frame
        this.frame.add(middle, BorderLayout.CENTER);
    }

    public void createSubmitButton(){
        JPanel submitRegion = createPanel();

        this.frame.add(submitRegion);
    }

    public void createBottomRegion(){
        // create panel
        JPanel bottom = createPanel();
        bottom.setBackground(new Color(18, 84, 136));

        // create text & insert to panel
        JLabel t = createLabel("COPYRIGHT @ 2024 | WazeAzure",
Color.white, bottom, JLabel.CENTER);
        bottom.add(t, BorderLayout.CENTER);

        // add panel to frame
        this.frame.add(bottom, BorderLayout.SOUTH);
    }

    public JPanel createPanel(){
        JPanel p = new JPanel();
        return p;
    }

    public JLabel createLabel(String content, Color color, JPanel
panel, int textHorizontalAlignment){
        JLabel text = new JLabel(content);
        text.setHorizontalAlignment(textHorizontalAlignment);
        text.setForeground(color);
        return text;
    }

```

```
public JTextField createTextField(int column, Font font){
    JTextField textField = new JTextField();
    textField.setColumns(column);
    textField.setText("example");
    textField.setBorder(BorderFactory.createCompoundBorder(
        textField.getBorder(),
        BorderFactory.createEmptyBorder(5, 5, 5, 5)));

    // if font not null
    if(font != null){
        textField.setFont(font);
    }

    return textField;
}
```

Pada bagian ini, Main App akan memanggil method main() dari class tersebut.

## 6.2 Main App

```
import datahandling.Datahandling;
import gui.*;

/**
 * Main Program Class.
 * @author WazeAzure
 */
public class App {

    /**
     * main method
     *
     * @param args Get arguments from runtime
     */
    public static void main(String[] args) {
        Datahandling f = new Datahandling();
        // f.createDir("test/dictionary/");
        f.main("test/dictionary.txt");

        Gui mainGui = new Gui(f);
        mainGui.main();
    }
}
```

## 6.3 Data Handling

```
package datahandling;

import java.io.*;
import java.util.*;
import datastructure.*;

public class Datahandling {
    public int max_len;
    public int count;
    private HashSet<String> keyword_table;

    private Map<String, ArrayList<Node>> graphMap;

    private Map<String, Node> graphStringNode;

    public Datahandling(){
        this.max_len = 0;
        this.count = 0;
        keyword_table = new HashSet<>();

        graphMap = new HashMap<String, ArrayList<Node>>();
        graphStringNode = new HashMap<String, Node>();
    }

    public void createDir(String dirname){
        boolean status = new File(dirname).mkdirs();
        if(!status){
            System.out.println("ERROR CREATING DIR");
        }
    }

    public void main(String filename){
        try {
            File file = new File("test/graphMap");
            if (file.exists()) {
                deserializeFileToMap();

                File myObj = new File(filename);
                Scanner myReader = new Scanner(myObj);
                while (myReader.hasNextLine()) {
                    String data = myReader.nextLine();
                }
            }
        }
    }
}
```

```

        // System.out.println(data);
        if(data.length() > this.max_len){
            max_len = data.length();
        }
        this.count++;
        addWordList(data);

        Node n2 = new Node(data, null);
        graphStringNode.put(data, n2);
    }
    myReader.close();
} else {
    File myObj = new File(filename);
    Scanner myReader = new Scanner(myObj);
    while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        // System.out.println(data);
        if(data.length() > this.max_len){
            max_len = data.length();
        }
        this.count++;
        addWordList(data);
        addNode(data);

        Node n2 = new Node(data, null);
        graphStringNode.put(data, n2);
    }
    myReader.close();

    System.out.println(this.max_len);
    System.out.println(this.count);

    // save graph to file
    serializeMapToFile("test/graphMap");
}
} catch (FileNotFoundException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
}

public void serializeMapToFile(String filename){

```

```

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(this.graphMap);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void deserializeFileToMap(){
        Map<String, ArrayList<Node>> map = new HashMap<String,
ArrayList<Node>>();
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("test/graphMap"))) {
            map = (Map<String, ArrayList<Node>>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        this.graphMap = map;
    }

    public void addWordList(String word){
        if(!checkWord(word)){
            keyword_table.add(word);
        }
    }

    public boolean checkWord(String word){
        if(this.keyword_table.contains(word)){
            return true;
        }
        return false;
    }

    public void addNode(String data){
        ArrayList<Node> tempArrayList = new ArrayList<Node>();
        graphMap.put(data, tempArrayList);

        for(String key: graphMap.keySet()){
            if(key.length() == data.length()){
                if(nodeIsParent(key, data)){
                    Node keyNode = getNodeFromString(key);
                    Node n = new Node(data, keyNode);
                    graphMap.get(key).add(n);
                }
            }
        }
    }

```



```

graphMap.get(data).add(keyNode);

Node n2 = new Node(data, null);
graphStringNode.put(data, n2);
    }
    }
}

public boolean nodeIsParent(String s1, String s2){
    int len = s1.length();
    int difference = 0;

    for(int i=0; i<len; i++){
        if(s1.charAt(i) != s2.charAt(i)){
            difference++;
            if(difference == 2){
                return false;
            }
        }
    }

    return true;
}

public Node getNodeFromString(String s){
    return this.graphStringNode.get(s);
}

public ArrayList<Node> getNodeList(String n){
    return this.graphMap.get(n);
}
}

```

## 6.4 Data Structure

```
package datastructure;

import java.util.*;
import java.io.Serializable;

public class Node implements Serializable {
    public String info;
    public Node parent;
    public int depth;

    public Node(String info, Node parentNode){
        this.info = info;
        this.parent = parentNode;
    }

    public ArrayList<String> getPath(Node end){
        ArrayList<String> path = new ArrayList<String>();

        Node current = this;

        while (!current.info.equals(end.info)) {
            path.add(current.info);
            current = current.parent;
        }
        // current == end
        if (current.info.equals(end.info)){
            path.add(end.info);
        }

        return path;
    }

    public int getHeuristic(String target){
        int count = 0;
        for(int i=0; i<this.info.length(); i++){
            if(target.charAt(i) != this.info.charAt(i)){
                count++;
            }
        }
    }
}
```

```
        return count;
    }
}
```

## 6.5 Algorithm

### 6.5.1 Algorithm

```
package algorithm;

import java.util.*;

import datahandling.Datahandling;
import datastructure.*;

public class Algorithm {
    protected HashSet<String> visitedNode;
    protected Datahandling datahandling;

    public Algorithm(){
        this.visitedNode = new HashSet<String>();
    }

    public boolean checkisVisited(Node s){
        if(visitedNode.contains(s.info)){
            return true;
        }
        return false;
    }

    public void setVisitedNode(Node s){
        visitedNode.add(s.info);
    }
}
```

## 6.5.2 AlgorithmHandler

```
package algorithm;

import java.util.*;

import datahandling.Datahandling;
import datastructure.*;

public class Algohandler {
    /** time execution in milliseconds */
    public long timeExecuted;
    public int nodeVisited;
    public long memoryUsed;

    public ArrayList<String> solve(String start, String end, String
algo, Datahandling dh){
        ArrayList<String> pathAns = new ArrayList<String>();

        Node startNode = dh.getNodeFromString(start);
        Node endNode = dh.getNodeFromString(end);

        long startTime = System.currentTimeMillis();
        long startMemory = Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
        long endMemory = 0;
        if(algo.equals("1")){
            // panggil UCS
            UCS ucs = new UCS(dh);

            nodeVisited = ucs.solve(startNode, endNode);
            pathAns = ucs.path;

            endMemory = Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
        }

        else if(algo.equals("2")){
            // panggil GBFS
            GBFS gbfs = new GBFS(dh);

            nodeVisited = gbfs.solve(startNode, endNode);
            pathAns = gbfs.path;
        }
    }
}
```

```

        endMemory = Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
    }

    else if(algo.equals("3")){
        // panggil A*

        Astar astar = new Astar(dh);

        nodeVisited = astar.solve(startNode, endNode);
        pathAns = astar.path;

        endMemory = Runtime.getRuntime().totalMemory() -
Runtime.getRuntime().freeMemory();
    }

    long endTime = System.currentTimeMillis();
    this.timeExecuted = endTime - startTime;
    this.memoryUsed = endMemory - startMemory;

    System.out.println("Time execution: " + timeExecuted + "ms");
    System.out.println("Memory used: " + memoryUsed + " bytes");
    return pathAns;
}
}

```

### 6.5.3 UCS

```
package algorithm;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Queue;

import datahandling.Datahandling;
import datastructure.Node;

/**
 * UCS algorithm
 *
 * @param awikwok A class to represent a point on the x-y coordinate
plane.
 */
public class UCS extends Algorithm {

    public ArrayList<String> path;

    public UCS(Datahandling dh) {
        super();
        this.datahandling = dh;
        this.path = new ArrayList<String>();
    }

    public int solve(Node start, Node end) {
        if(start.info.equals(end.info)) {
            return 0;
        }

        int nodeVisited = 0;

        ArrayDeque<Node> queue = new ArrayDeque<>();
        start.depth = 0;
        queue.add(start);

        while(!queue.isEmpty()) {
            Node n = queue.getFirst();
            queue.removeFirst();
            nodeVisited++;
        }
    }
}
```

```

        if(!checkisVisited(n)){
            setVisitedNode(n);
            if(n.info.equals(end.info)){
                this.path = n.getPath(start);
                return nodeVisited;
            }

            ArrayList<Node> temp =
datahandling.getNodeList(n.info);
            for(int i=0; i<temp.size(); i++){
                if(temp.get(i) != null){
                    Node x = new Node(temp.get(i).info, n);

                    if(x != null && !checkisVisited(x)){
                        x.parent = n;
                        x.depth = x.parent.depth + 1;
                        queue.add(x);
                    }
                }
            }
        }

        return nodeVisited;
    }
}

```



### 6.5.4 GBFS

```
package algorithm;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

import datahandling.Datahandling;
import datastructure.Node;

public class GBFS extends Algorithm {
    public ArrayList<String> path;
    private String endString;

    public GBFS(Datahandling dh){
        super();
        this.datahandling = dh;
        this.path = new ArrayList<String>();
    }

    class PQComparator implements Comparator<Node>{
        @Override
        public int compare(Node o1, Node o2) {
            // System.out.println("cOMPARATOR CALLED");
            int h1 = o1.getHeuristic(endString);
            // System.out.println(o1.info + " | " + h1);
            int h2 = o2.getHeuristic(endString);
            // System.out.println(o2.info + " | " + h2);
            if (h1 > h2){
                return 1;
            } else if(h1 < h2){
                return -1;
            }
            return 0;
        }
    }

    public int solve(Node start, Node end){
        this.endString = end.info;

        if(start.info.equals(end.info)){
            return 0;
        }
    }
}
```

```

    }

    int nodeVisited = 0;

    PriorityQueue<Node> queue = new PriorityQueue<Node>(new
PQComparator());
    queue.offer(start);

    while(!queue.isEmpty()){
        Node n = queue.poll();
        nodeVisited++;
        queue.clear();

        if(!checkisVisited(n)){
            setVisitedNode(n);
            if(n.info.equals(end.info)){

                this.path = n.getPath(start);
                return nodeVisited;
            }

            ArrayList<Node> temp =
datahandling.getNodeList(n.info);
            for(int i=0; i<temp.size(); i++){
                Node x = temp.get(i);
                if(x != null && !checkisVisited(x)){
                    x.parent = n;
                    queue.add(x);
                }
            }
        }
    }

    return nodeVisited;
}
}

```

### 6.5.5 A\*

```
package algorithm;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

import datahandling.Datahandling;
import datastructure.Node;

public class Astar extends Algorithm {
    public ArrayList<String> path;
    private String endString;

    public Astar(Datahandling dh){
        super();
        this.datahandling = dh;
        this.path = new ArrayList<String>();
    }

    class PQComparator implements Comparator<Node>{
        @Override
        public int compare(Node o1, Node o2) {
            // System.out.println("COMPARATOR CALLED");
            int h1 = o1.getHeuristic(endString) + o1.depth;
            // System.out.println(o1.info + " | " + h1);
            int h2 = o2.getHeuristic(endString) + o2.depth;
            // System.out.println(o2.info + " | " + h2);
            if (h1 > h2){
                return 1;
            } else if(h1 < h2){
                return -1;
            }
            return 0;
        }
    }

    public int solve(Node start, Node end){
        this.endString = end.info;

        if(start.info.equals(end.info)){
            return 0;
        }
    }
}
```

```

        int nodeVisited = 0;

        PriorityQueue<Node> queue = new PriorityQueue<Node>(new
PQComparator());
        start.depth = 0;
        queue.offer(start);

        while(!queue.isEmpty()){
            Node n = queue.poll();
            nodeVisited++;

            if(!checkisVisited(n)){
                setVisitedNode(n);
                if(n.info.equals(end.info)){

                    this.path = n.getPath(start);
                    return nodeVisited;
                }

                ArrayList<Node> temp =
datahandling.getNodeList(n.info);
                for(int i=0; i<temp.size(); i++){
                    if(temp.get(i) != null){
                        Node x = new Node(temp.get(i).info, n);

                        if(x != null && !checkisVisited(x)){
                            x.parent = n;
                            x.depth = x.parent.depth + 1;
                            queue.add(x);
                        }
                    }
                }
            }
        }

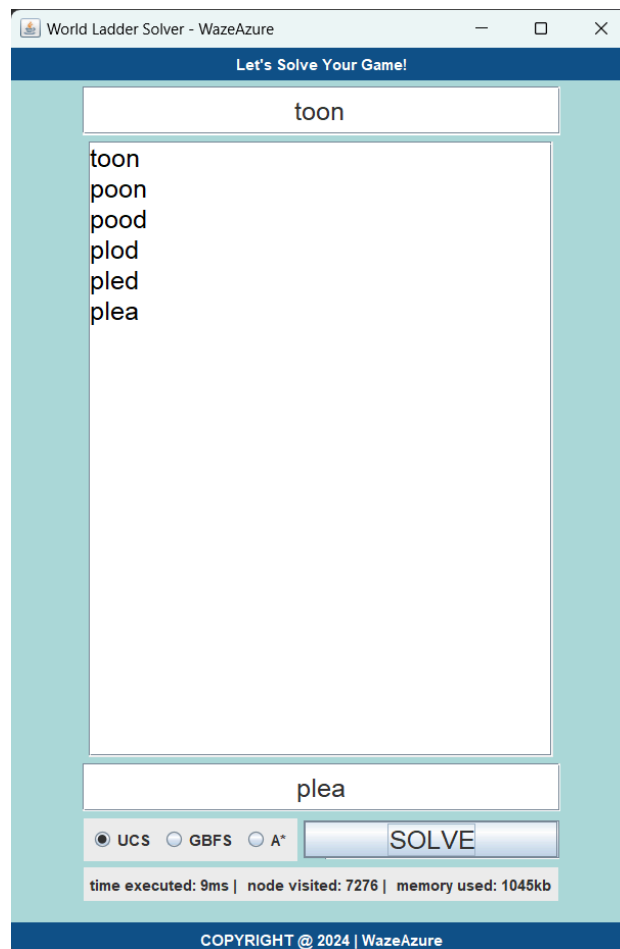
        return nodeVisited;
    }
}

```

## 7. How To Use

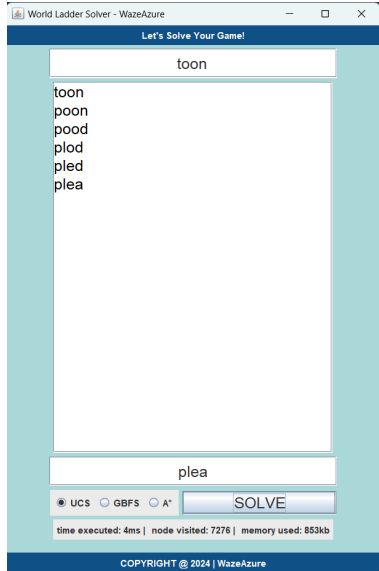
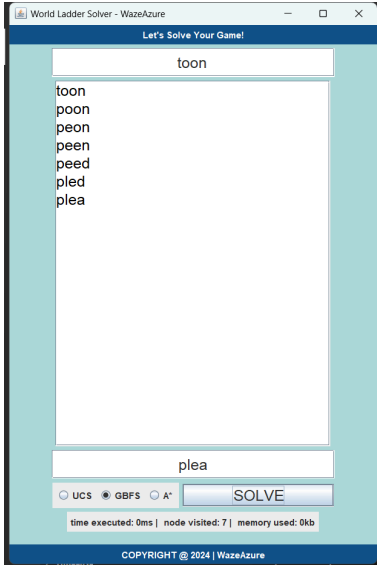
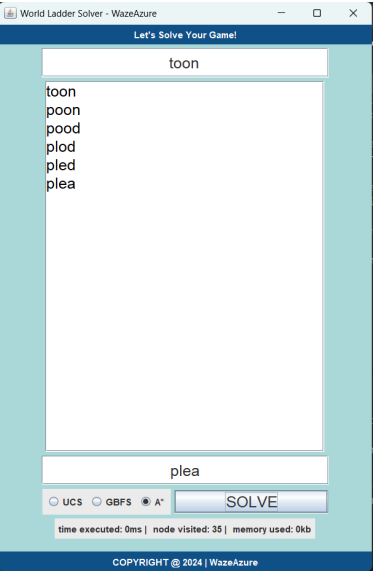
Berikut merupakan langkah-langkah untuk menjalankan program.

1. Clone repository dari github. Kemudian masuk ke dalam folder.
2. Jalankan perintah **make run** pada command line, atau tekan **F5** pada visual studio code.
3. Jika belum pernah melakukan caching maka startup process akan berlangsung selama 5 - 10 menit. Jika sudah maka  $< 5$  detik.
4. Masukkan input dan pilih algoritma yang diinginkan. Lalu tekan solve.

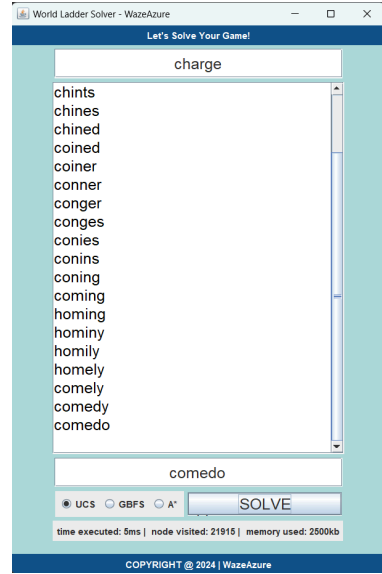

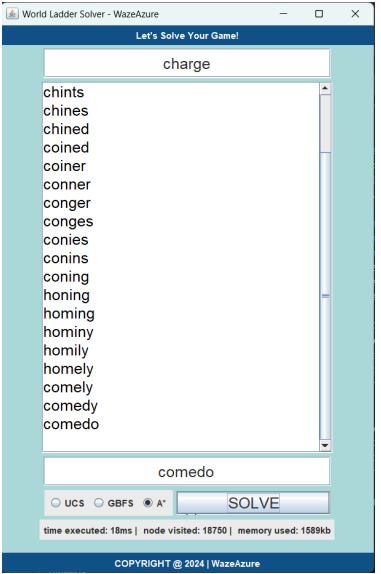


## 8. Test Case

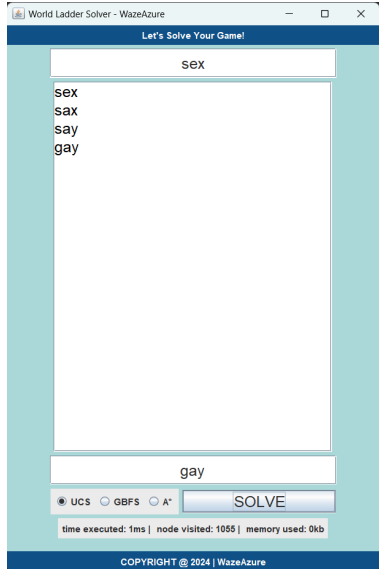
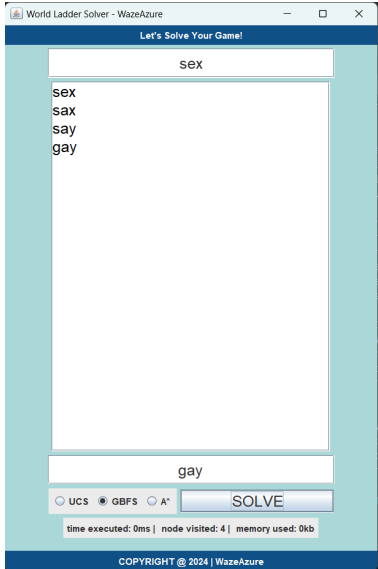
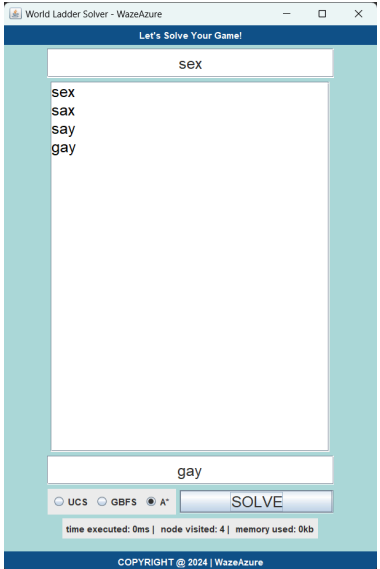
### 8.1 Test Case 1 | toon → plea

Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 4ms	Time: 0ms	Time: 0ms
Node: 7276	Node Visited: 7	Node: 35
Memory: 853 kb	Memory: 0 kb	Memory: 0 kb

## 8.2 Test Case 2 | charge → comedo

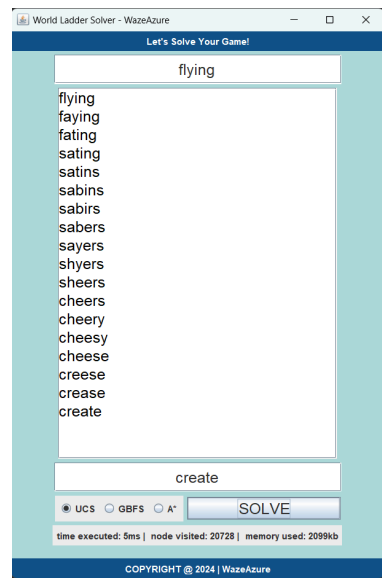

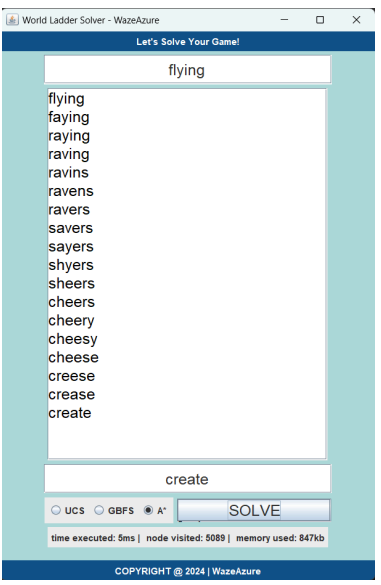
Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 5ms	Time: 1ms	Time: 18ms
Node: 21915	Node: 11	Node: 18750
Memory: 2500 kb	Memory: 0 kb	Memory: 1589 kb

### 8.3 Test Case 3 | sex → gay

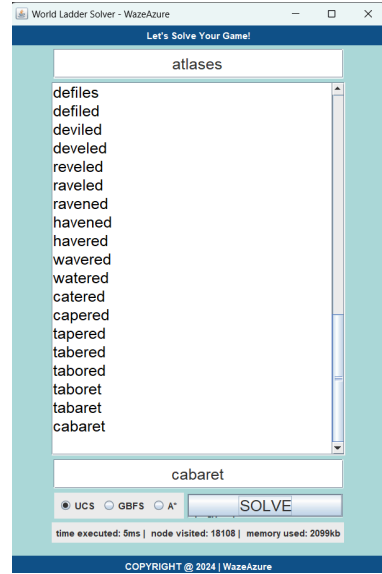
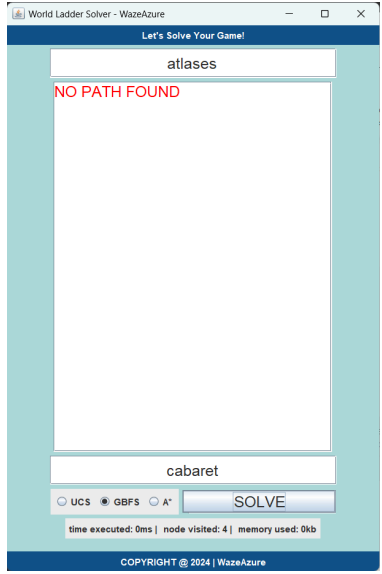
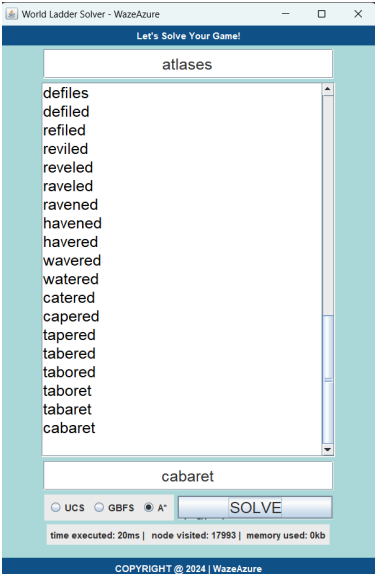
Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 1ms	Time: 0ms	Time: 0ms
Node: 1055	Node: 4	Node: 4
Memory: 0 kb	Memory: 0 kb	Memory: 0 kb



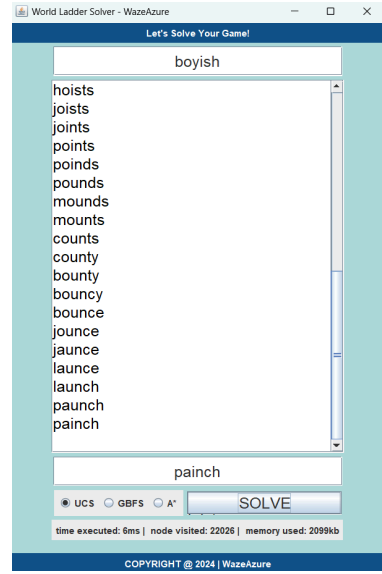
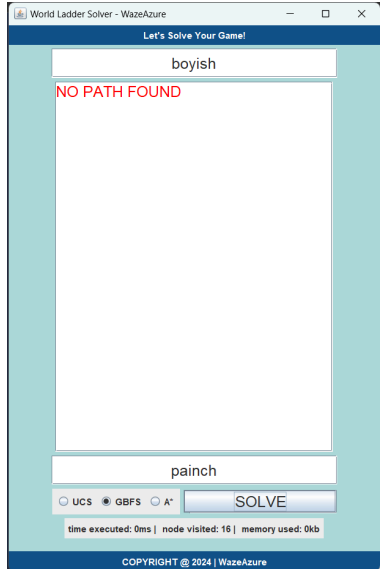
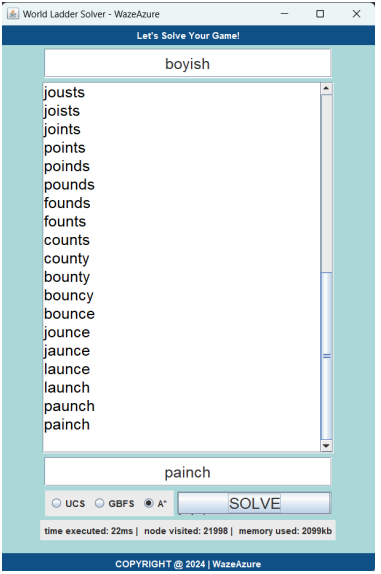
## 8.4 Test Case 4 | flying → create

Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 5ms	Time: 0ms	Time: 5ms
Node: 2078	Node: 42	Node: 5089
Memory: 2099 kb	Memory: 0 kb	Memory: 847 kb


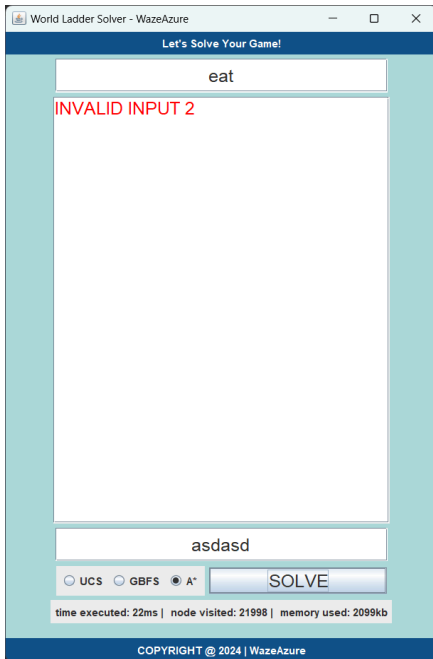


## 8.5 Test Case 5 | atlases → cabaret

Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 5ms	Time: 0ms	Time: 20ms
Node: 18108	Node: 4	Node: 17993
Memory: 2099 kb	Memory: 0 kb	Memory: 0 kb

## 8.6 Test Case 6 | boyish → painch

Algoritma UCS	Algoritma GBFS	Algoritma A*
		
Time: 6ms	Time: 0ms	Time: 22ms
Node: 22026	Node: 16	Node: 21998
Memory: 2099 kb	Memory: 0 kb	Memory: 2099kb

## 8.7 Test Case 7 | Error Handling

Invalid Input 1	Invalid Input 2
 <p>The screenshot shows the 'World Ladder Solver' application window. The title bar reads 'World Ladder Solver - WazeAzure'. The main window has a blue header with the text 'Let's Solve Your Game!'. Below the header, there is a text input field containing 'awikwok'. A red error message 'INVALID INPUT 1' is displayed below the input field. At the bottom, there is another text input field containing 'painch', three radio buttons for 'UCS', 'GBFS', and 'A*' (with 'A*' selected), a 'SOLVE' button, and a status bar showing 'time executed: 22ms   node visited: 21998   memory used: 2099kb'. The footer contains 'COPYRIGHT @ 2024   WazeAzure'.</p>	 <p>The screenshot shows the 'World Ladder Solver' application window. The title bar reads 'World Ladder Solver - WazeAzure'. The main window has a blue header with the text 'Let's Solve Your Game!'. Below the header, there is a text input field containing 'eat'. A red error message 'INVALID INPUT 2' is displayed below the input field. At the bottom, there is another text input field containing 'asdasd', three radio buttons for 'UCS', 'GBFS', and 'A*' (with 'A*' selected), a 'SOLVE' button, and a status bar showing 'time executed: 22ms   node visited: 21998   memory used: 2099kb'. The footer contains 'COPYRIGHT @ 2024   WazeAzure'.</p>
Invalid Length	Invalid Algorithm
 <p>The screenshot shows the 'World Ladder Solver' application window. The title bar reads 'World Ladder Solver - WazeAzure'. The main window has a blue header with the text 'Let's Solve Your Game!'. Below the header, there is a text input field containing 'eat'. A red error message 'DIFFERENT INPUT LENGTH' is displayed below the input field. At the bottom, there is another text input field containing 'peach', three radio buttons for 'UCS', 'GBFS', and 'A*' (with 'A*' selected), a 'SOLVE' button, and a status bar showing 'time executed: 22ms   node visited: 21998   memory used: 2099kb'. The footer contains 'COPYRIGHT @ 2024   WazeAzure'.</p>	 <p>The screenshot shows the 'World Ladder Solver' application window. The title bar reads 'World Ladder Solver - WazeAzure'. The main window has a blue header with the text 'Let's Solve Your Game!'. Below the header, there is a text input field containing 'example'. A red error message 'INVALID ALGORITHM' is displayed below the input field. At the bottom, there is another text input field containing 'example', three radio buttons for 'UCS', 'GBFS', and 'A*' (with 'A*' selected), a 'SOLVE' button, and a status bar showing 'time executed: 22ms   node visited: 21998   memory used: 2099kb'. The footer contains 'COPYRIGHT @ 2024   WazeAzure'.</p>

## 8.8 Analisis Hasil Test Case

Berdasarkan hasil test case 1 didapatkan hasil jalur Algoritma A\* dan UCS sama. Namun, baik waktu eksekusi maupun node yang dikunjungi dari A\* lebih efektif daripada UCS. GBFS dapat memberikan sebuah hasil, namun algoritma tersebut terjebak pada optimum lokal, karena path yang dihasilkan lebih panjang. Namun, waktu node yang dikunjungi terendah diantara ketiganya.

Pada pengujian kedua GBFS tidak mendapatkan solusi, hal ini karena GBFS terjebak pada jalan buntu. Saat GBFS sudah memilih kata dengan biaya terendah ternyata kata tersebut tidak memiliki simpul yang bertetangga yang belum pernah dikunjungi. GBFS ini juga tidak dapat menemukan solusinya pada test case empat, lima, dan enam.

Pada test case kita dapat melihat bahwa UCS secara konsisten sangat cepat. Hal ini karena dilakukannya *preprocessing* terhadap kamus, sehingga menghasilkan *adjacency list*. Dengan begitu UCS dengan fungsi  $g(n)$  dapat langsung mengakses simpul-simpul yang bertetangga dengannya. Lebih cepat, daripada A\* yang mengalami *bottleneck*.

Terdapat kasus unik pada program penulis untuk algoritma A\*. Meskipun algoritma A\* mengunjungi node yang lebih sedikit daripada UCS, waktu eksekusi dari A\* lebih besar daripada UCS. Hal ini karena pada algoritma A\*, program akan memanggil fungsi `getHeuristic()` yang menghitung nilai heuristic dengan membandingkan karakter pada string secara iteratif. Hal ini lah yang menjadi *bottleneck* sehingga waktu eksekusi A\* seperti pada test case 2, 5, 6 lebih besar daripada UCS. Namun perbedaan kurang dari 50ms tentunya *negligible* untuk menyelesaikan problem tersebut dengan terjaminnya waktu eksekusi dan memori yang lebih optimal untuk test case yang sangat panjang.

Terlepas dari waktu eksekusi ini, algoritma A\* dapat selalu memberikan hasil yang optimal, dengan memori yang digunakan juga lebih sedikit daripada UCS. Disisi lain, GBFS tidak selalu memberikan hasil yang optimal, sehingga penyelesaian *World Ladder* dengan GBFS tentunya tidak disarankan. Untuk algoritma UCS kita tetap dapat menggunakannya namun waktu eksekusi, memori, dan simpul yang dikunjungi tentunya lebih besar daripada A\*, tapi solusinya terjamin ada.

Berdasarkan hasil test case dapat disimpulkan bahwa algoritma A\* paling optimal pada permainan *Word Ladder* ini dengan solusi, jumlah simpul yang dikunjungi, dan penggunaan memori yang efektif dibanding UCS maupun BFS.

# GitHub

Kode dapat diakses pada repository GitHub [https://github.com/WazeAzure/Tucil3\\_13522039](https://github.com/WazeAzure/Tucil3_13522039)

Repository akan di public pada Hari Selasa 7 Mei.

Jika tidak dapat diakses, mohon kontak penulis di LINE yenyenhui atau TEAMS.

## Poin-Poin Penting

No	Poin	Ya	Tidak
1.	Program berhasil dijalankan.	V	
2.	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS	V	
3.	Solusi yang diberikan pada algoritma UCS optimal	V	
4.	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search	V	
5.	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma A*	V	
6.	Solusi yang diberikan pada algoritma A* optimal	V	
7.	<b>[Bonus]:</b> Program memiliki tampilan GUI	V	