

LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

1. Haikal Assyauqi (1352205)

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan
Informatika Institut Teknologi
Bandung**

2024

DAFTAR ISI

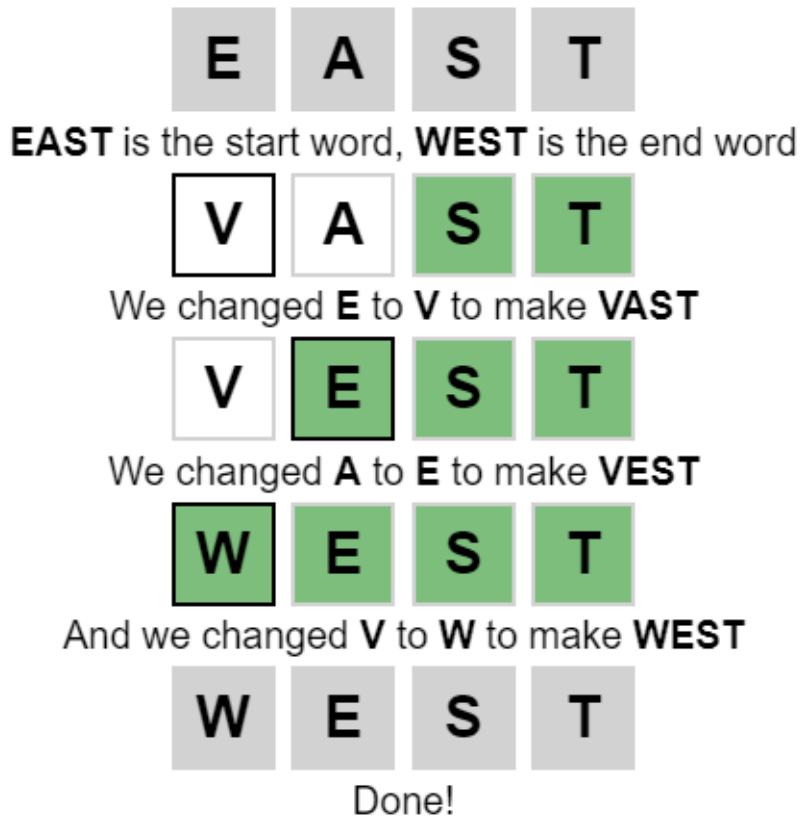
DAFTAR ISI.....	1
BAB I.....	2
BAB II.....	3
BAB III.....	6
BAB IV.....	22
BAB V.....	41
DAFTAR PUSTAKA.....	43
LAMPIRAN.....	44

BAB I

DESKRIPSI MASALAH

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.

Example

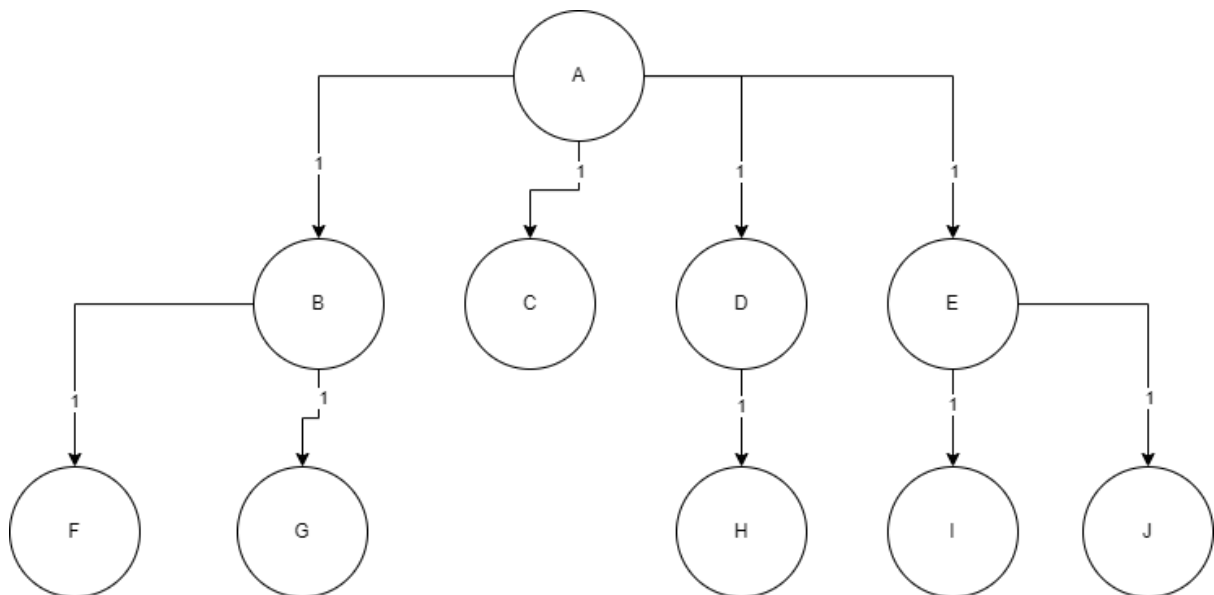


Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder

BAB II

LANGKAH PEMECAHAN MASALAH

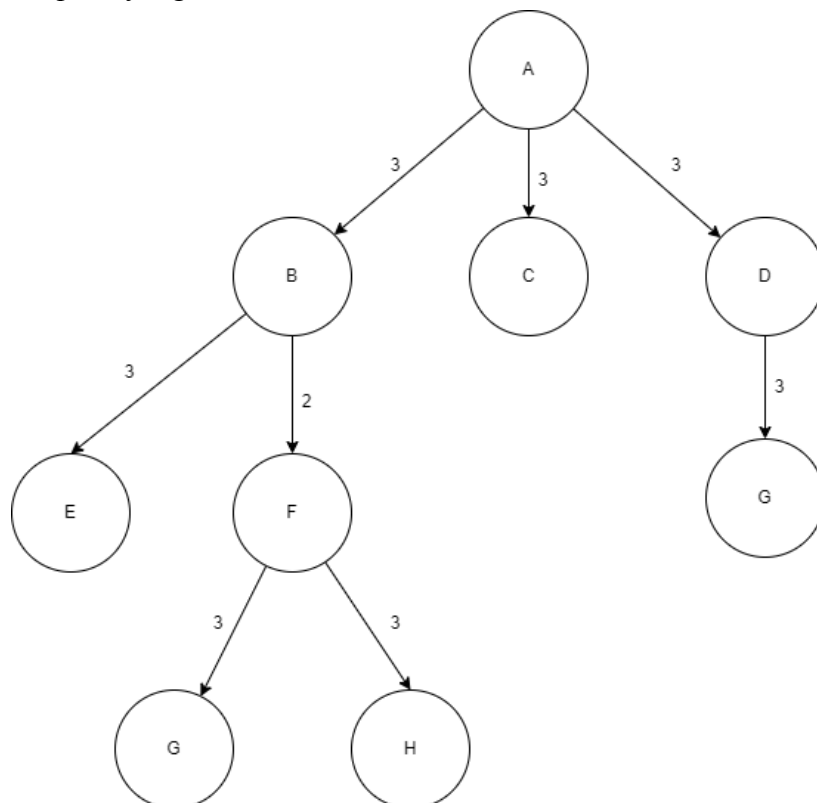
1. Penyelesaian Menggunakan UCS (Uniform Cost Search)
 - a. Program menerima masukan pada *text field* yang disediakan
 - b. Program melakukan validasi pada kata yang terdapat pada kamus yang berformat .txt
 - c. Nilai $g(n)$ yang digunakan pada algoritma UCS ini yaitu jumlah *node* yang telah dikunjungi dalam satu *path*.
 - d. Program akan mengambil path dengan nilai $g(n)$ terkecil, lalu akan diambil kata terakhir dari path tersebut.
 - e. Dilakukan pengecekan pada kata yang telah diambil, apabila kata yang diambil merupakan kata yang sesuai dengan *end word*, program berhenti dan *path* dicetak pada kolom yang disediakan pada GUI.
 - f. Jika kata yang diambil bukan *end word*, maka kata tersebut akan di*expand* dan setiap kata baru yang tidak menyebabkan sirkuit dan belum pernah di*expand* akan digabungkan dengan *path* sebelumnya dan akan dimasukkan dalam sebuah *array*.
 - g. Lakukan langkah dari c hingga f hingga solusi ditemukan atau ketika *path* tidak ditemukan



Jika menggunakan algoritma UCS maka *path* yang dilalui yaitu A, B, C, D, E, F, G, H, I, J.

2. Penyelesaian menggunakan Greedy Best First Search

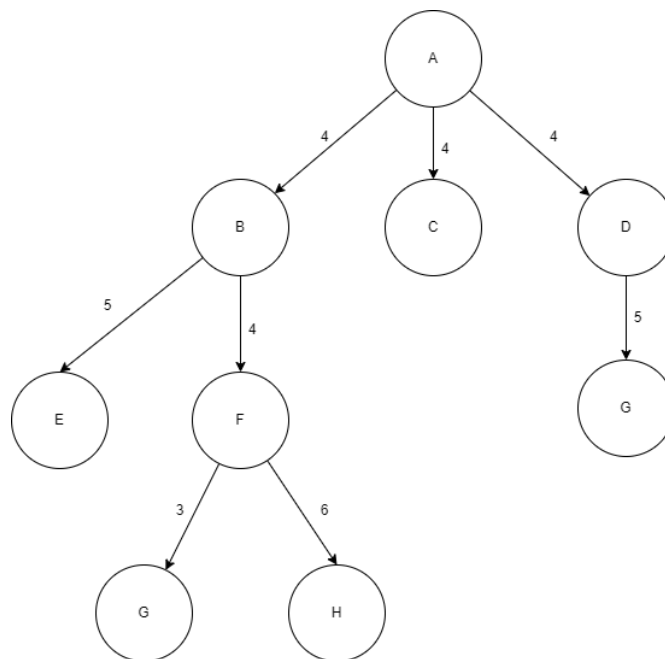
- Program menerima masukan pada *text field* yang disediakan
- Program melakukan validasi pada kata yang terdapat pada kamus yang berformat .txt
- Penyelesaian pada algoritma ini menggunakan nilai $h(n)$ yang mana nilai heuristiknya menggunakan jumlah huruf yang berbeda dengan hasil akhir
- Program akan mengambil path dengan nilai $h(n)$ terkecil, lalu akan diambil kata terakhir dari path tersebut.
- Usai mengambil *path* yang diinginkan, hapus semua *path* yang tersedia dalam *queue*
- Dilakukan pengecekan pada kata yang telah diambil, apabila kata yang diambil merupakan kata yang sesuai dengan *end word*, program berhenti dan *path* dicetak pada kolom yang disediakan pada GUI.
- Jika kata yang diambil bukan *end word*, maka kata tersebut akan di*expand* dan setiap kata baru yang tidak menyebabkan sirkuit dan belum pernah di*expand* akan digabungkan dengan *path* sebelumnya dan akan dimasukkan dalam sebuah *array*.
- Lakukan langkah dari c hingga g hingga solusi ditemukan atau ketika tidak ada *path* yang sesuai



Jika menggunakan algoritma Greedy Best First Search maka *path* yang dilalui yakni A, B, F, G.

3. Penyelesaian Menggunakan A*

- Program menerima masukan pada *text field* yang disediakan
- Program melakukan validasi pada kata yang terdapat pada kamus yang berformat .txt
- Nilai yang digunakan pada algoritma A* ini yaitu jumlah *node* yang telah dikunjungi dalam satu *path* dan nilai heuristik, sehingga $f(n) = g(n) + h(n)$
- Program akan mengambil path dengan nilai terkecil, lalu akan diambil kata terakhir dari path tersebut.
- Dilakukan pengecekan pada kata yang telah diambil, apabila kata yang diambil merupakan kata yang sesuai dengan *end word*, program berhenti dan *path* dicetak pada kolom yang disediakan pada GUI.
- Jika kata yang diambil bukan *end word*, maka kata tersebut akan diexpand dan setiap kata baru yang tidak menyebabkan sirkuit dan belum pernah diexpand akan digabungkan dengan *path* sebelumnya dan akan dimasukkan dalam sebuah *array*.
- Lakukan langkah dari c hingga f hingga solusi ditemukan atau ketika *path* tidak ditemukan



Jika menggunakan algoritma A* maka path yang akan dikunjungi yakni A, B, C, D, F, E, G, H

BAB III

IMPLEMENTASI PROGRAM

Program menggunakan bahasa Java, terdapat beberapa file yang memiliki fungsi berbeda-beda:

1. ListArray.java

Berfungsi untuk mengurus path yang telah dilalui dan *cost*-nya

```
import java.util.*;

public class ListArray {
    ArrayList<ArrayList<String>> path;
    ArrayList<Integer> cost;

    public ListArray() {
        path = new ArrayList<ArrayList<String>>();
        cost = new ArrayList<Integer>();
    }
}
```

2. Function.java

Berfungsi untuk menyediakan fungsi-fungsi manipulasi

```
import java.util.*;

public class Function {
    public boolean isWordExist(String word, ArrayList<String> words) {
        if(words.contains(word)) {
            return true;
        }
        return false;
    }

    public ArrayList<String> cuma_beda_satu(String word,
    ArrayList<String> words) {
        ArrayList<String> result = new ArrayList<String>();
        for (String w : words) {
            int count = 0;
            for (int i = 0; i < word.length(); i++) {
                if (word.charAt(i) != w.charAt(i)) {
                    count++;
                }
            }
        }
    }
}
```

```

        }
    }
    if (count == 1) {
        result.add(w);
    }
}
return result;
}

public ArrayList<String> get_same_Length(String word,
ArrayList<String> words) {
    ArrayList<String> result = new ArrayList<String>();
    for (String w : words) {
        if (w.length() == word.length()) {
            result.add(w);
        }
    }
    return result;
}

public int diff_Letter (String word1, String word2) {
    int count = 0;
    for (int i = 0; i < word1.length(); i++) {
        if (word1.charAt(i) != word2.charAt(i)) {
            count++;
        }
    }
    return count;
}

public int get_index (Integer path_cost, ArrayList<Integer> costs)
{
    for (int i = 0; i < costs.size(); i++) {
        if (costs.get(i) > path_cost) {
            return i;
        }
    }
    return costs.size();
}

```



```

    public boolean the_only_minimum (ArrayList<Integer> costs, int
path_cost) {
        for (int i = 0; i < costs.size(); i++) {
            if (costs.get(i) >= path_cost) {
                return false;
            }
        }
        return true;
    }
}

```

3. MainFrame.java

Menampilkan GUI Java

```

import javax.swing.*.*;
// import javax.swing.border.EmptyBorder;

import java.awt.*.*;
import java.awt.event.*.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*.*;

public class MainFrame extends JFrame implements ActionListener{
    JPanel paneltitle, panelstartword, panelendword, panelbutton,
contentPane, panelradio;
    JLabel title, label1, label2, label3;
    JTextField textField, textField2;
    JButton button;
    JRadioButton radio1, radio2, radio3;
    ImageIcon image;
    ButtonGroup group;
    ArrayList<String> wordList;
    Function function = new Function();
    Solution solution = new Solution();

    MainFrame(ArrayList<String> words) {
        // Set ArrayList
        wordList = new ArrayList<String>();
        wordList = words;
    }
}

```

```

// Set icon
image = new ImageIcon("image.png");

// Set label
title = new JLabel();
title.setText("Kalin");
title.setFont(new Font("Georgia", Font.BOLD, 30));
title.setForeground(Color.decode("0x141414"));
// title.setPreferredSize(new Dimension(100, 25));

label1 = new JLabel();
label1.setText("Start Word");
// label1.setPreferredSize(new Dimension(100, 25));

label2 = new JLabel();
label2.setText("End Word");

// Status pencarian
label3 = new JLabel();

// Set text field
textField = new JTextField(20);
// textField.setPreferredSize(new Dimension(100, 25));
textField2 = new JTextField(20);

// Set button
button = new JButton();
button.setText("Search");
button.setBackground(Color.decode("0x253354"));
button.setForeground(Color.decode("0xF2F3F4"));
button.addActionListener(this);

// Set RadioButton
radio1 = new JRadioButton("UCS");
radio1.setBackground(Color.decode("0x7587B4"));
radio2 = new JRadioButton("Greedy Best First Search");
radio2.setBackground(Color.decode("0x7587B4"));
radio3 = new JRadioButton("A*");
radio3.setBackground(Color.decode("0x7587B4"));

```

```

        // Set ButtonGroup
        group = new ButtonGroup();
        group.add(radius1);
        group.add(radius2);
        group.add(radius3);

        //Set panel
        paneltitle = new JPanel(new FlowLayout(FlowLayout.CENTER, 0,
10));
        paneltitle.setPreferredSize(new Dimension(600, 50));
        paneltitle.add(title);
        paneltitle.setBackground(Color.decode("0x7587B4"));
        // paneltitle.setBackground(Color.BLUE);

        panelstartword = new JPanel(new FlowLayout(FlowLayout.CENTER,
0, 10));
        panelstartword.setPreferredSize(new Dimension(250, 70));
        panelstartword.add(label1);
        panelstartword.add(textField);
        panelstartword.setBackground(Color.decode("0x7587B4"));

        panelendword = new JPanel(new FlowLayout(FlowLayout.CENTER, 0,
10));
        panelendword.setPreferredSize(new Dimension(250, 70));
        panelendword.add(label2);
        panelendword.add(textField2);
        panelendword.setBackground(Color.decode("0x7587B4"));
        // panelstartword.setBackground(Color.);

        panelradio = new JPanel(new FlowLayout(FlowLayout.CENTER, 0,
10));
        panelradio.setPreferredSize(new Dimension(800, 30));
        panelradio.add(radius1);
        panelradio.add(radius2);
        panelradio.add(radius3);
        panelradio.setBackground(Color.decode("0x7587B4"));

        panelbutton = new JPanel(new FlowLayout(FlowLayout.CENTER, 0,
10));
        panelbutton.setPreferredSize(new Dimension(800, 50));
        panelbutton.add(button);

```

```

        panelbutton.setBackground(Color.decode("0x7587B4"));

        contentPane = new JPanel(new FlowLayout(FlowLayout.CENTER, 0,
10));
        contentPane.setPreferredSize(new Dimension(600, 5000));
        contentPane.setBackground(Color.decode("0xB3BDD4"));

        JScrollPane scrollPane = new JScrollPane(contentPane);
scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AL
WAYS);

        scrollPane.setPreferredSize(new Dimension(600, 700));

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(800, 800);
        this.setTitle("Word Ladder");
        this.setLayout(new FlowLayout(FlowLayout.CENTER, 800, 5));
        this.getContentPane().setBackground(Color.decode("0x7587B4"));
        // this.setResizable(false);
        this.add(paneltitle);
        this.add(panelstartword);
        this.add(panelendword);
        this.add(panelradio);
        this.add(panelbutton);
        this.add(label3);
        this.add(scrollPane);
        this.setIconImage(image.getImage());
        this.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button) {
            contentPane.removeAll();
            label3.setText(" ");
            String startWord = textField.getText();
            startWord = startWord.toUpperCase();
            String endWord = textField2.getText();
            endWord = endWord.toUpperCase();
            ArrayList<String> wordList_same_length = new
ArrayList<String>();

```

```

        if (startWord.length() != endWord.length() ||
!wordList.contains(startWord) || !wordList.contains(endWord)) {
            label3.setText("Kata yang anda memasukkan memiliki
panjang yang berbeda atau tidak valid dalam kamus, silahkan masukkan
kata yang benar");
        }
        else {
            wordList_same_length =
function.get_same_Length(startWord, wordList);
            // System.out.println(wordList);
            if (radio1.isSelected()) {
                label3.setText(" ");
                UCS ucs = new UCS();
                solution = ucs.get_path_UCS(startWord, endWord,
wordList_same_length);
            } else if (radio2.isSelected()) {
                label3.setText(" ");
                Greedy_Best_First greedy = new Greedy_Best_First();
                solution = greedy.get_path_UCS(startWord, endWord,
wordList_same_length);
            } else if (radio3.isSelected()) {
                label3.setText(" ");
                Astar astar = new Astar();
                solution = astar.get_path_Astar(startWord, endWord,
wordList_same_length);
            } else {
                label3.setText("Silahkan pilih algoritma yang ingin
digunakan");
            }

            if (solution.wordList.size() == 0) {
                JLabel path = new JLabel("Tidak ada jalur yang
ditemukan");
                path.setPreferredSize(new Dimension(600, 15));
                contentPane.add(path);
                JLabel time = new JLabel("Waktu yang diperlukan: "
+ solution.time + " detik");
                time.setPreferredSize(new Dimension(600, 15));
                contentPane.add(time);
                JLabel node = new JLabel("Node yang dilalui: " +
solution.visited_nodes);

```

```

        node.setPreferredSize(new Dimension(600, 15));
        contentPane.add(node);
        for(int i = solution.wordList.size() + 2; i < 30;
i++) {
            JLabel empty = new JLabel("
");
            empty.setPreferredSize(new Dimension(600, 15));
            contentPane.add(empty);
        }
    }
    else {
        for (int i = 0; i < solution.wordList.size(); i++)
{
            JLabel path = new JLabel(String.valueOf(i+1)+"
"+solution.wordList.get(i));
            path.setPreferredSize(new Dimension(600, 15));
            contentPane.add(path);
        }
        JLabel time = new JLabel("Waktu yang diperlukan: "
+ solution.time + " detik");
        time.setPreferredSize(new Dimension(600, 15));
        contentPane.add(time);
        JLabel node = new JLabel("Node yang dilalui: " +
solution.visited_nodes);
        node.setPreferredSize(new Dimension(600, 15));
        contentPane.add(node);
        for(int i = solution.wordList.size() + 2; i < 30;
i++) {
            JLabel empty = new JLabel("
");
            empty.setPreferredSize(new Dimension(600, 15));
            contentPane.add(empty);
        }
    }
}
System.out.println(startWord);
System.out.println(endWord);
// label3.setText(wordList.get(0));
}
}

```

```

public static void main(String[] args) {
    ArrayList<String> words = new ArrayList<String>();
    try {
        File myObj = new File("dictionary.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            words.add(data.toUpperCase());
        }
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
    new MainFrame(words);
}
}

```

4. Solution.java

Menyimpan hasil akhir

```

import java.util.ArrayList;

public class Solution {
    public ArrayList<String> wordList;
    public int visited_nodes;
    public double time;

    public Solution() {
        wordList = new ArrayList<String>();
        visited_nodes = 0;
        time = 0;
    }

    public Solution(ArrayList<String> words, double time) {
        wordList = new ArrayList<String>();
        wordList = words;
        visited_nodes = 0;
        this.time = time;
    }
}

```

5. UCS.java

Mencari dengan algoritma UCS

```
import java.util.*;

public class UCS {
    Function function = new Function();

    public Solution get_path_UCS(String startWord, String endWord,
    ArrayList<String> sameLengthWords) {
        Solution solution = new Solution();
        ListArray listArray = new ListArray(); // Create a new list
        array
        listArray.path.add(new
        ArrayList<String>(Arrays.asList(startWord))); // Add the start word to
        the path
        listArray.cost.add(0); // Add the cost to the path
        boolean found = false; // Set found to false
        int count = 0; // Set count to 0
        ArrayList<String> alreadyExpand = new ArrayList<String>();
        long startTime = System.nanoTime(); // Get the start time

        // Check new words
        while (found == false && listArray.path.size() > 0) {
            ArrayList<String> currentPath = listArray.path.get(0);
            String currentWord = currentPath.get(currentPath.size() -
1);

            if (currentWord.equals(endWord)) {
                System.out.println("Path found!");
                System.out.print("Path: ");
                System.out.println(currentPath);
                System.out.print("Cost: ");
                // System.out.println(currentCost + 1
                System.out.println("Node yang dilalui: " + count);
                long endTime = System.nanoTime();
                long timeElapsed = endTime - startTime;
                // System.out.println("Execution time in milliseconds:
" + (timeElapsed / 1000000000.00) + " detik");
                // System.out.println(listArray.path);
                solution.wordList = currentPath;
```



```

        solution.time = (timeElapsed / 1000000000.00);
        solution.visited_nodes = count;
        return solution;
    }

    // System.out.println(currentWord);
    int currentCost = listArray.cost.get(0);
    listArray.cost.remove(0);
    listArray.path.remove(0);
    count += 1;
    ArrayList<String> nextPath =
function.cuma_beda_satu(currentWord, sameLengthWords);
    if (alreadyExpand.contains(currentWord) == false) {
        alreadyExpand.add(currentWord);
    }

    for (String s : nextPath) {
        if(currentPath.contains(s) == false &&
alreadyExpand.contains(s) == false){
            ArrayList<String> newPath = new
ArrayList<String>(currentPath);
            newPath.add(s);
            listArray.path.add(newPath);
            listArray.cost.add(currentCost + 1);
        }
    }
    // System.out.println(currentPath);
    // System.out.println(listArray.path.s/ize());
    // if (currentPath.contains("LASE")){
    // System.out.println(currentPath);
    // }
}
long endTime = System.nanoTime();
long timeElapsed = endTime - startTime;
solution.time = timeElapsed / 1000000000.00;
solution.visited_nodes = count;
return solution;
}
}

```

6. Greedy_Best_First.java

Mencari dengan algoritma Greedy Best First

```
import java.util.*;

public class Greedy_Best_First {
    Function function = new Function();

    public Solution get_path_UCS(String startWord, String endWord,
    ArrayList<String> sameLengthWords) {
        Solution solution = new Solution();
        ListArray listArray = new ListArray();
        listArray.path.add(new
    ArrayList<String>(Arrays.asList(startWord)));
        listArray.cost.add(function.diff_Letter(startWord, endWord));
        boolean found = false;
        int count = 0;
        long startTime = System.nanoTime();
        ArrayList<String> alreadyExpand = new ArrayList<String>();

        while (found == false && listArray.path.size() > 0) {
            ArrayList<String> currentPath = listArray.path.get(0); //
    Get the first path
            String currentWord = currentPath.get(currentPath.size() -
    1); // Get the last word in the path
            // Kata sesuai dengan kata akhir
            if (currentWord.equals(endWord)) {
                System.out.println("Path found!");
                System.out.print("Path: ");
                System.out.println(currentPath);
                System.out.println("Node yang dilalui: " + count);
                long endTime = System.nanoTime();
                long timeElapsed = endTime - startTime;
                solution.wordList = currentPath;
                solution.time = (timeElapsed / 1000000000.00);
                solution.visited_nodes = count;
                return solution;
            }

            // int currentCost = listArray.cost.get(0); // Get the cost
    of the first path
        }
```

```

        ArrayList<String> nextPath =
function.cuma_beda_satu(currentWord, sameLengthWords); // Get the next
path

        if (alreadyExpand.contains(currentWord) == false) {
            alreadyExpand.add(currentWord);
        }
        listArray.path.clear();
        listArray.cost.clear();
        // listArray.cost.remove(0); // Remove the cost
        // listArray.path.remove(0); // Remove the path
        count++;

        for (String s : nextPath) { // Loop through the next path
            if ((currentPath.contains(s) == false) &&
(alreadyExpand.contains(s) == false)) { // If the path doesn't contain
the word

                ArrayList<String> newPath = new
ArrayList<String>(currentPath); // Create a new path
                newPath.add(s); // Add the word to the new path
                int index =
function.get_index(function.diff_Letter(s, endWord), listArray.cost);
// Get the index

                listArray.path.add(index, newPath); // Add the new
path to the list

                listArray.cost.add(index, function.diff_Letter(s,
endWord)); // Add the cost to the list
            }
        }
        // System.out.println(listArray.path);
    }
    long endTime = System.nanoTime();
    long timeElapsed = endTime - startTime;
    solution.time = timeElapsed / 1000000000.00;
    solution.visited_nodes = count;
    return solution;
}
}

```

7. Astar.java

Mencari dengan algoritma A*

```
import java.util.*;

public class Astar {
    ListArray listArray = new ListArray();
    Function function = new Function();

    public Solution get_path_Astar(String startWord, String endWord,
    ArrayList<String> sameLengthWords) {

        // Declare the variables
        Solution solution = new Solution();
        ListArray listArray = new ListArray();
        listArray.path.add(new
    ArrayList<String>(Arrays.asList(startWord)));
        listArray.cost.add(function.diff_Letter(startWord, endWord));
        boolean found = false;
        int count = 0;
        long startTime = System.nanoTime();
        ArrayList<String> alreadyExpand = new ArrayList<String>();

        while (found == false && listArray.path.size() > 0) {
            ArrayList<String> currentPath = listArray.path.get(0); //
    Get the first path
            String currentWord = currentPath.get(currentPath.size() -
    1); // Get the last word in the path

            if (currentWord.equals(endWord)) {
                System.out.println("Path found!");
                System.out.print("Path: ");
                System.out.println(currentPath);
                System.out.print("Cost: ");
                System.out.println(currentPath.size());
                System.out.println("Node yang dilalui: " + count);
                long endTime = System.nanoTime();
                long timeElapsed = endTime - startTime;
                solution.wordList = currentPath;
                solution.time = (timeElapsed / 1000000000.00);
                solution.visited_nodes = count;
            }
        }
    }
}
```

```

        System.out.println("Waktu yang diperlukan: " +
solution.time + " detik");
        return solution;
    }

    ArrayList<String> nextPath =
function.cuma_beda_satu(currentWord, sameLengthWords); // Get the next
path

    // System.out.println("Kemungkinan path: "+ nextPath);
    if (alreadyExpand.contains(currentWord) == false) {
        alreadyExpand.add(currentWord);
    }
    listArray.cost.remove(0); // Remove the cost
    listArray.path.remove(0); // Remove the path
    count++;
    for (String s : nextPath) { // Loop through the next path
        if ((currentPath.contains(s) == false) &&
(alreadyExpand.contains(s) == false)) { // If the path doesn't contain
the word

            ArrayList<String> newPath = new
ArrayList<String>(currentPath); // Create a new path
            newPath.add(s); // Add the word to the new path
            int index =
function.get_index(function.diff_Letter(s, endWord) + newPath.size()-1,
listArray.cost); // Get the index
            listArray.path.add(index, newPath); // Add the new
path to the list

            listArray.cost.add(index, function.diff_Letter(s,
endWord) + newPath.size()-1); // Add the cost to the list
        }
    }
    // System.out.println(currentPath);
    // System.out.print (listArray.path.get(0) + " ");
    // System.out.println(listArray.path.get(0).size());
    // System.out.println(listArray.path.size());
    // System.out.println(alreadyExpand);1
}
long endTime = System.nanoTime();
long timeElapsed = endTime - startTime;
solution.time = timeElapsed / 1000000000.00;
solution.visited_nodes = count;

```

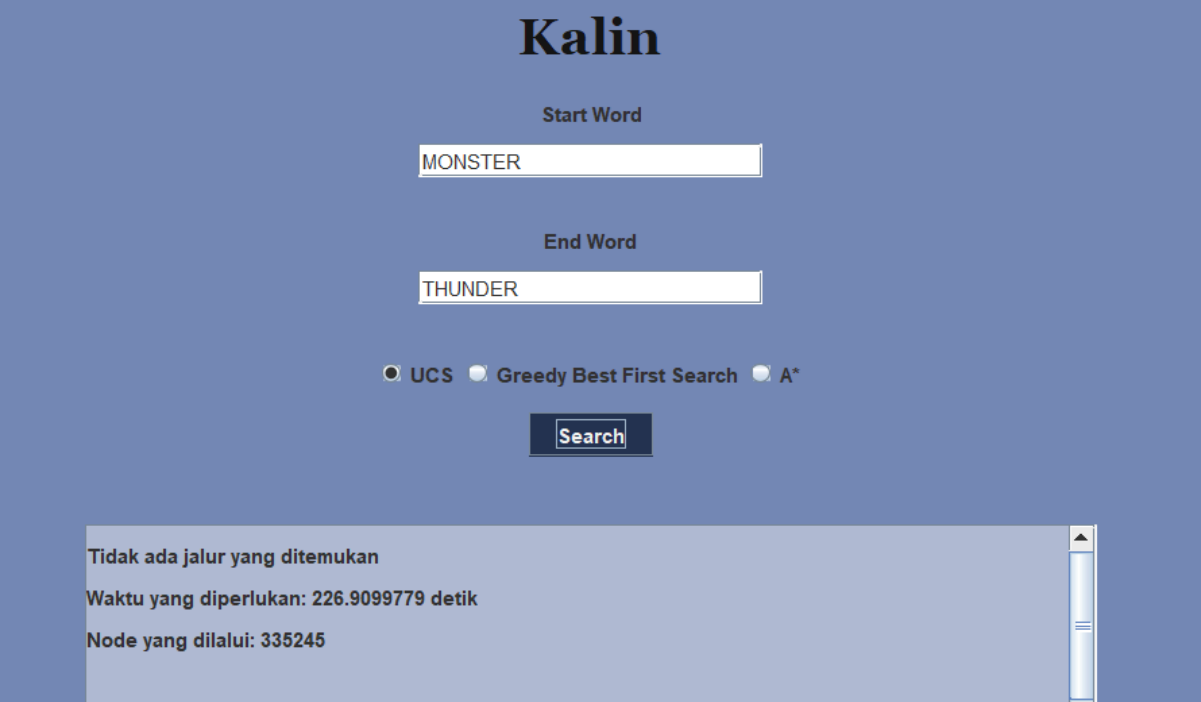
```
        return solution;
    }
}
```

BAB IV

UJI COBA

1. MONSTER - THUNDER

a. UCS



The screenshot shows the 'Kalin' web application interface. At the top, the title 'Kalin' is displayed in a large, bold, black font. Below the title, there are two input fields: 'Start Word' with the value 'MONSTER' and 'End Word' with the value 'THUNDER'. Underneath these fields, there are three radio buttons for selecting an algorithm: 'UCS' (which is selected), 'Greedy Best First Search', and 'A*'. A 'Search' button is located below the radio buttons. At the bottom of the interface, a light blue box contains the following text: 'Tidak ada jalur yang ditemukan', 'Waktu yang diperlukan: 226.9099779 detik', and 'Node yang dilalui: 335245'.

Kalin

Start Word
MONSTER

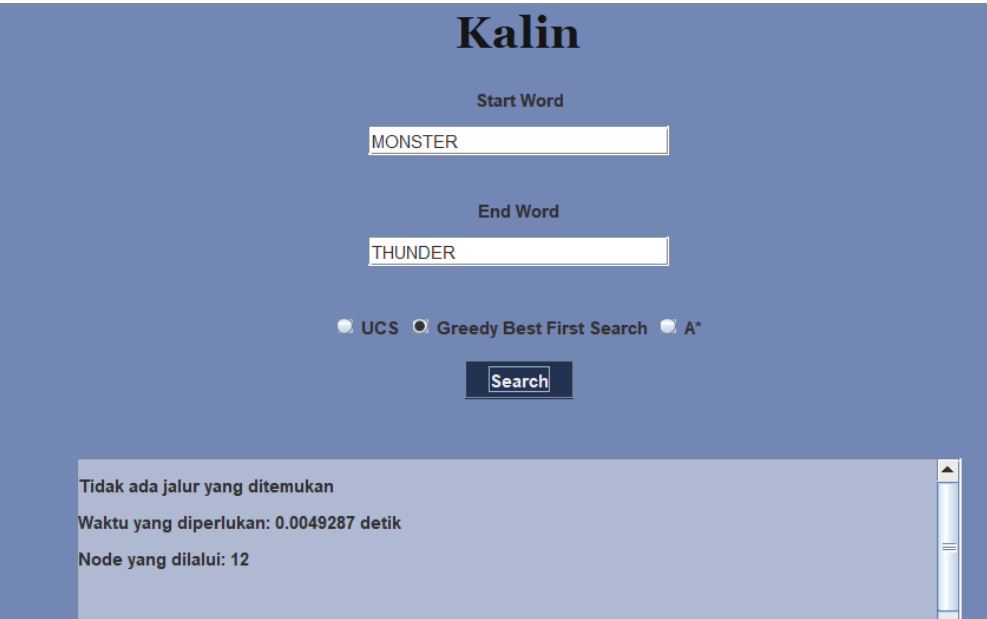
End Word
THUNDER

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan
Waktu yang diperlukan: 226.9099779 detik
Node yang dilalui: 335245

b. GBFS



The screenshot shows the 'Kalin' web application interface. At the top, the title 'Kalin' is displayed in a large, bold, black font. Below the title, there are two input fields: 'Start Word' with the value 'MONSTER' and 'End Word' with the value 'THUNDER'. Underneath these fields, there are three radio buttons for selecting an algorithm: 'UCS', 'Greedy Best First Search' (which is selected), and 'A*'. A 'Search' button is located below the radio buttons. At the bottom of the interface, a light blue box contains the following text: 'Tidak ada jalur yang ditemukan', 'Waktu yang diperlukan: 0.0049287 detik', and 'Node yang dilalui: 12'.

Kalin

Start Word
MONSTER

End Word
THUNDER

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan
Waktu yang diperlukan: 0.0049287 detik
Node yang dilalui: 12

c. A*

The screenshot shows a web application titled "Kalin". It has two input fields: "Start Word" with the value "MONSTER" and "End Word" with the value "THUNDER". Below these fields are three radio buttons for selecting a search algorithm: "UCS", "Greedy Best First Search", and "A*". The "A*" radio button is selected. A "Search" button is located below the radio buttons. At the bottom of the interface, there is a light blue box containing the following text: "Tidak ada jalur yang ditemukan", "Waktu yang diperlukan: 236.1708761 detik", and "Node yang dilalui: 293774".

Kalin

Start Word
MONSTER

End Word
THUNDER

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

Tidak ada jalur yang ditemukan
Waktu yang diperlukan: 236.1708761 detik
Node yang dilalui: 293774

2. PURPLE - PLANET

a. UCS

Kalin

Start Word

PURPLE

End Word

PLANET

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. PURPLE
2. PURPLY
3. PURELY
4. SURELY
5. SORELY
6. SORELS
7. MORELS
8. MOTELS
9. BOTELS
10. BOWELS
11. BOWERS
12. COWERS
13. COOERS
14. COOEES
15. COOEED
16. COOLED
17. COALED

18. COATED
19. CRATED
20. PRATED
21. PLATED
22. PLANED
23. PLANET

Waktu yang diperlukan: 84.0621052 detik

Node yang dilalui: 216614

b. GBFS

Start Word

PURPLE

End Word

PLANET

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan

Waktu yang diperlukan: 0.0010408 detik

Node yang dilalui: 2

c. A*

Start Word

PURPLE

End Word

PLANET

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. PURPLE
2. PURPLY
3. PURELY
4. SURELY
5. SORELY
6. SORELS
7. MORELS
8. MOTELS
9. BOTELS
10. BOWELS
11. BOWERS
12. COWERS
13. COOERS
14. COOEES
15. COOEED
16. COOLED
17. COALED

18. COATED
19. CRATED
20. PRATED
21. PLATED
22. PLANED
23. PLANET

Waktu yang diperlukan: 2.5056298 detik

Node yang dilalui: 11718

3. CIVIL - RIGHT

a. UCS

Start Word

CIVIL

End Word

RIGHT

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. CIVIL
2. CAVIL
3. CAVIE
4. MAVIE
5. MAVIS
6. MAXIS
7. MAXES
8. MANES
9. MINES
10. SINES
11. SINHS
12. SIGHS
13. SIGHT
14. RIGHT

Waktu yang diperlukan: 41.0257618 detik

Node yang dilalui: 192138

b. GBFS

Start Word

CIVIL

End Word

RIGHT

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan

Waktu yang diperlukan: 0.0013995 detik

Node yang dilalui: 5

c. A*

Start Word

CIVIL

End Word

RIGHT

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. CIVIL
2. CIVIE
3. CAVIE
4. MAVIE
5. MAVIS
6. MAXIS
7. MAXES
8. MIXES
9. MINES
10. SINES
11. SINHS
12. SIGHS
13. SIGHT
14. RIGHT

Waktu yang diperlukan: 0.7588024 detik

Node yang dilalui: 5304

4. APPLE - HAPPY

a. UCS

Start Word

APPLE

End Word

HAPPY

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. APPLE
2. AMPLE
3. AMOLE
4. ANOLE
5. ANILE
6. ANILS
7. ARILS
8. ARIAS
9. ARRAS
10. AURAS
11. AURES
12. CURES
13. CARES
14. CARPS
15. HARPS
16. HARPY
17. HAPPY

Waktu yang diperlukan: 49.3809959 detik

Node yang dilalui: 211510

b. GBFS

Start Word

APPLE

End Word

HAPPY

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan

Waktu yang diperlukan: 0.0010951 detik

Node yang dilalui: 4

c. A*

Start Word

APPLE

End Word

HAPPY

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. APPLE
2. AMPLE
3. AMOLE
4. ANOLE
5. ANILE
6. ANISE
7. ARISE
8. PRISE
9. PAISE
10. PASSE
11. PASTE
12. PASTY
13. PARTY
14. PARDY
15. HARDY
16. HARPY
17. HAPPY

Waktu yang diperlukan: 0.4266787 detik

Node yang dilalui: 3827

5. BALL - PLAY

a. UCS

Start Word

BALL

End Word

PLAY

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. BALL
2. BAAL
3. BAAS
4. BRAS
5. BRAY
6. PRAY
7. PLAY

Waktu yang diperlukan: 6.9643399 detik

Node yang dilalui: 59104

b. GBFS

Start Word

BALL

End Word

PLAY

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

1. BALL
2. BAAL
3. BAAS
4. BIAS
5. PIAS
6. PEAS
7. PEAG
8. PEAK
9. PEAL
10. PEAN
11. PLAN
12. PLAY

Waktu yang diperlukan: 0.0030107 detik

Node yang dilalui: 11

c. A*

Start Word

BALL

End Word

PLAY

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. BALL
2. PALL
3. PILL
4. PIAL
5. PIAN
6. PLAN
7. PLAY

Waktu yang diperlukan: 0.0299 detik

Node yang dilalui: 509

6. CELL - ATOM

a. UCS

Start Word

CELL

End Word

ATOM

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. CELL
2. SELL
3. SEEL
4. SEEP
5. STEP
6. STOP
7. ATOP
8. ATOM

Waktu yang diperlukan: 11.7192186 detik

Node yang dilalui: 101568

b. GBFS

Start Word

CELL

End Word

ATOM

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

Tidak ada jalur yang ditemukan

Waktu yang diperlukan: 0.0045249 detik

Node yang dilalui: 47

c. A*

Start Word

CELL

End Word

ATOM

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. CELL
2. SELL
3. SEEL
4. SEEP
5. STEP
6. STOP
7. ATOP
8. ATOM

Waktu yang diperlukan: 0.1028202 detik

Node yang dilalui: 1711

7. STAY - SIDE

a. UCS

Start Word

STAY

End Word

SIDE

☒ UCS ☐ Greedy Best First Search ☐ A*

Search

1. STAY
2. STAT
3. SEAT
4. SENT
5. SENE
6. SINE
7. SIDE

Waktu yang diperlukan: 2.5020449 detik

Node yang dilalui: 26319

b. GBFS

Start Word

STAY

End Word

SIDE

☐ UCS ☒ Greedy Best First Search ☐ A*

Search

1. STAY
2. SHAY
3. SHAD
4. SCAD
5. SCAB
6. SCAG
7. SCAM
8. SCAN
9. SCAR
10. SCAT
11. SCOT
12. SCOP
13. SCOW
14. SHOW
15. SHOE
16. SLOE
17. SLUE

18. SPUE
19. SPAE
20. SPAN
21. SPAR
22. SEAR
23. SEAL
24. SIAL
25. SILL
26. SILD
27. SILK
28. SICK
29. SICE
30. SIDE

Waktu yang diperlukan: 0.0041187 detik
Node yang dilalui: 29

c. A*

Start Word
STAY

End Word
SIDE

☐ UCS ☐ Greedy Best First Search ☒ A*

Search

1. STAY
2. STAT
3. SEAT
4. SENT
5. SENE
6. SINE
7. SIDE

Waktu yang diperlukan: 0.0229185 detik
Node yang dilalui: 492

8. RUN - GAS

a. UCS

Start Word

RUN

End Word

gas

☒ UCS
 ☐ Greedy Best First Search
 ☐ A*

Search

1. RUN
 2. GUN
 3. GAN
 4. GAS

Waktu yang diperlukan: 0.0214518 detik

Node yang dilalui: 913

b. GBFS

Start Word

RUN

End Word

gas

☐ UCS
 ☒ Greedy Best First Search
 ☐ A*

Search

1. RUN
 2. GUN
 3. GAN
 4. GAS

Waktu yang diperlukan: 0.0017935 detik

Node yang dilalui: 3

c. A*

☐ UCS
 ☐ Greedy Best First Search
 ☒ A*

Search

1. RUN
 2. GUN
 3. GAN
 4. GAS

Waktu yang diperlukan: 0.0026942 detik

Node yang dilalui: 6

BAB V

ANALISIS

Dalam menjalankan algoritma UCS, Greedy Best First, dan A*, kita memerlukan $g(n)$ dan $h(n)$ untuk menentukan *cost* dalam pembangkitan *node*, $g(n)$ merupakan *cost* yang diperlukan dari root menuju *node* tertentu, dan $h(n)$ merupakan perkiraan *cost* dari *node* tertentu menuju garis finish, dan untuk menentukan $g(n)$, kita menggunakan jumlah langkah yang telah digunakan dari *root* menuju *node*, dan $h(n)$ kita menggunakan jumlah huruf yang berbeda antara *node* dengan *end word*.

Dikarenakan $g(n)$ yang digunakan pada UCS memiliki nilai yang sama antar *node* tetangga, hal ini menyebabkan UCS pada program ini sangat mirip dengan BFS, yang mana algoritma BFS dan UCS pada program ini sama-sama mengecek *node* di sekitar *parent node* terlebih dahulu, dalam proses pencarian solusi, UCS selalu memberikan hasil yang optimal, namun waktu yang diperlukan sangat lama dibandingkan dengan algoritma lainnya.

Greedy Best First Search (GBFS) pada program ini berjalan sangat cepat, namun hasil yang diberikan lebih sering tidak optimal, sesuai dengan teori, karena kita hanya menjelajahi 1 *path* yang ditentukan sejak awal, sehingga tidak terjadi backtrack, keuntungan dari memakai algoritma ini, waktu yang digunakan sangat cepat, *node* yang dijelajahi sangat sedikit, namun memiliki kekurangan yakni solusi yang diberikan tidak optimal.

Algoritma yang terakhir, yakni algoritma A* merupakan algoritma yang menggabungkan $g(n)$ dan $h(n)$ dalam perhitungannya, hal ini dapat menampilkan solusi yang hampir sama keefektifannya dengan UCS dengan nilai efisiensi yang sangat tinggi, sesuai dengan teori, di mana A* akan selalu lebih cepat dibanding UCS.

Dengan menggunakan jumlah huruf berbeda sebagai nilai $h(n)$, maka nilai heuristik pada A* bersifat *admissible* di mana syarat untuk sebuah nilai heuristik disebut *admissible* ketika selalu menghasilkan nilai yang *underestimate*, sebagai contoh dengan menggunakan jumlah huruf yang berbeda sebagai nilai heuristik, maka jumlah minimal yang digunakan untuk mengubah

kata menjadi *end word* pasti sama atau lebih besar dari nilai heuristik yang digunakan

IMPLEMENTASI BONUS

Untuk bonus, GUI menggunakan API dari Swing Java, pada program terdapat 2 text field untuk pembacaan start word dan end word, terdapat tombol *submit* untuk memulai pencarian, selama pencarian *path*, tombol akan berwarna putih

DAFTAR PUSTAKA

<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>

LAMPIRAN

1. Link Github

https://github.com/Haikalin/Tucil3_13522052

2. Tabel Checkpoint

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	✓	