

**Laporan Tugas Kecil 3**  
**IF2211 Strategi Algoritma**  
**Julian Caleb Simandjuntak / 13522099**



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

## Daftar Isi

|   |           |
|---|-----------|
| <b>Daftar Isi</b>                         | <b>2</b>  |
| <b>Bab 1 Deskripsi Tugas</b>              | <b>4</b>  |
| <b>Bab 2 Dasar Teori dan Implementasi</b> | <b>5</b>  |
| Dasar Teori                               | 5         |
| Implementasi Teori                        | 6         |
| Penjelasan Tambahan                       | 7         |
| <b>Bab 3 Implementasi Source Code</b>     | <b>9</b>  |
| Class Main                                | 9         |
| Attribute                                 | 9         |
| Method                                    | 9         |
| Screenshot                                | 10        |
| Class GUI                                 | 10        |
| Attribute                                 | 10        |
| Method                                    | 10        |
| Screenshot                                | 11        |
| Static Class EnglishDictionary            | 11        |
| Attribute                                 | 11        |
| Method                                    | 12        |
| Screenshot                                | 13        |
| Class Node                                | 14        |
| Attribute                                 | 14        |
| Method                                    | 14        |
| Screenshot                                | 16        |
| Class WordLadder                          | 16        |
| Attribute                                 | 16        |
| Method                                    | 17        |
| Screenshot                                | 19        |
| How To Run                                | 22        |
| <b>Bab 4 Testing</b>                      | <b>24</b> |
| <b>Bab 5 Hasil Analisis</b>               | <b>30</b> |

|                         |           |
|-------------------------|-----------|
| <b>Bab 6 Kesimpulan</b> | <b>31</b> |
| <b>Bab 7 Pranala</b>    | <b>32</b> |
| <b>Lampiran</b>         | <b>33</b> |
| <b>Daftar Pustaka</b>   | <b>34</b> |

## **Bab 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.

Tugas Kecil 3 IF2211 Strategi Algoritma meminta mahasiswa membuat program penyelesaian masalah Word Ladder antara 2 kata dengan panjang huruf yang sama. Tugas ini diselesaikan dengan menggunakan bahasa pemrograman Java melalui CLI atau GUI jika mengerjakan bonus. Penyelesaian masalah menggunakan harus algoritma Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A\*. Kata yang digunakan adalah bahasa Inggris. Program menerima start word, end word, dan algoritma yang ingin digunakan, mengeluarkan path, node yang dikunjungi, dan waktu eksekusi.

## Bab 2

### Dasar Teori dan Implementasi

#### Dasar Teori

Struktur data graf dan tree, yang terdiri atas vertices ( $V$ ) dan edges ( $E$ ). Dapat digunakan sebagai representasi masalah yang kemudian dapat dijalankan sebuah algoritma penyelesaian masalah, salah satunya dalam hal penentuan rute. Algoritma penyelesaian masalah untuk penentuan rute dapat dibagi menjadi 2,

1. Uninformed Search, yang bekerja secara "blind" dan mungkin hanya menyimpan cost dari masing-masing edges, terdiri dari
  - a. BFS (Breadth First Search)
  - b. DFS (Depth First Search)
  - c. DLS (Depth Limited Search)
  - d. IDS (Iterative Deepening Search)
  - e. UCS (Uniform Cost Search)
2. Informed search, atau Heuristic search, yang memiliki evaluation function untuk setiap vertices yang dikunjungi ke goal vertices, terdiri dari
  - a. GBFS (Greedy Best First Search)
  - b.  $A^*$

Pada Tugil kali ini, akan dibahas dan digunakan algoritma UCS, GBFS, dan  $A^*$ .

UCS adalah algoritma pencarian rute yang digunakan untuk menemukan jalur cost terendah dalam graf berbobot dari node awal tertentu ke node tujuan. Algoritma ini melakukan eksplorasi graf secara bertahap keluar dari node awal, selalu memilih node dengan cost terendah (yang dihitung dengan fungsi  $g(n)$ ) untuk dikunjungi berikutnya. Karena yang dikunjungi selalu memiliki cost terendah antar node, secara teoritis, UCS pasti menghasilkan solusi optimal.

GBFS adalah algoritma pencarian rute secara heuristik yang memilih node berdasarkan fungsi evaluasi heuristik  $h(n)$  yang menghitung cost antara node yang dikunjungi dengan node tujuan tanpa mempertimbangkan cost sebenarnya untuk mencapai node tersebut. Hal ini membuat GBFS cepat namun belum tentu optimal, karena dapat terhenti di local optima jika fungsi  $h(n)$  tidak admissible, yaitu jika

untuk setiap node  $n$ ,  $h(n) \leq h^*(n)$ , dimana  $h^*(n)$  adalah cost sebenarnya untuk mencapai node tujuan.

$A^*$  adalah algoritma pencarian rute yang menggabungkan UCS dan GBFS, yaitu menghitung cost tiap node yang dikunjungi dengan penjumlahan dari cost ke node tersebut atau  $g(n)$  dan cost evaluasi heuristik dari node tersebut ke node tujuan atau  $h(n)$ , dengan kata lain,  $f(n) = g(n) + h(n)$ .  $A^*$  memilih node dengan jumlah nilai  $f(n)$  terendah untuk perluasan, menjadikannya optimal (dijamin menemukan jalur terpendek) dan efisien dalam banyak kasus, terutama ketika fungsi heuristik admissible.

### Implementasi Teori

Pada program, struktur utama pencarian rute kurang lebih sama, yang membedakan antara 3 algoritma hanyalah fungsi untuk menghitung cost ke node, yaitu  $g(n)$  untuk UCS,  $h(n)$  untuk GBFS, dan  $f(n) = g(n) + h(n)$  untuk  $A^*$ . Sebelum membahas struktur utama, akan didefinisikan  $g(n)$  dan  $h(n)$  terlebih dahulu, di mana cost akan dihitung berdasarkan perubahan huruf yang dilakukan pada node yang berupa kata. Mengambil basis start word memiliki current cost = 0, maka

- $g(n)$  adalah perubahan huruf dari kata yang sekarang ke kata berikutnya. Karena word ladder hanya membolehkan perubahan 1 huruf setiap step,  $g(n)$  selalu sama, adalah current cost dari kata sekarang + 1.
- $h(n)$  adalah perbedaan huruf dari kata berikutnya ke kata tujuan (end word). Misal dari "east" menjadi "west",  $h(n) = 2$ , yaitu perbedaan antara "e" dan "w", dan "a" dan "e".
- $f(n)$  adalah jumlah dari  $g(n)$  dengan  $h(n)$

Algoritma utamanya,

1. Dimulai dengan mengambil start word, end word, algorithm dari input, dan kamus kata bahasa Inggris (pada implementasi menggunakan local txt file). Jika start word sama dengan end word, program otomatis selesai.
2. Jika tidak, dilanjutkan dengan menginstansiasi priority queue berdasarkan cost tiap node (yang menyimpan kata, cost, dan path yang dilaluinya), dan sistem penyimpanan kata yang sudah dilalui sehingga tidak perlu dicek lagi (bisa

menggunakan set, namun pada implementasi algoritma mengambil approach dengan menghapus kata yang dikunjungi pada kamus yang sudah diduplikat sebelumnya). Dimulai juga perhitungan waktu untuk algoritma.

3. Start word di queue ke dalam priority queue.
4. Melakukan dequeue (mengambil kata paling depan, jika pertama kali dilakukan berarti kembali mengambil start word), dan membangkitkan setiap kata yang memiliki perbedaan 1 huruf dari kata yang di dequeue berdasarkan kamus.
5. Jika di antara kata-kata yang dibangkitkan terdapat end word, program keluar dari loop.
6. Untuk setiap kata, instansiasi node dengan katanya adalah kata yang dibangkitkan, costnya berdasarkan algoritma, jika UCS,  $\text{cost} = g(n)$ , jika GBFS,  $\text{cost} = h(n)$ , dan jika  $A^*$ ,  $\text{cost} = f(n)$ , serta path adalah array gabungan path sementara (jika baru pertama kali dilakukan, berarti hanya ada kata yang di dequeue) ditambah dengan kata tersebut. Dilakukan juga perhitungan banyak kata yang dikunjungi (dibangkitkan dan dibuat nodenya).
7. Setelah dibuat node, node dimasukkan ke dalam priority queue yang secara otomatis melakukan sort antar node berdasarkan costnya, semakin kecil semakin di depan.
8. Ulangi loop 4 - 7 hingga ditemukan end word.
9. Setelah end word ditemukan, hentikan perhitungan waktu, ambil path pada node sekarang, ditambahkan dengan end word. Program menampilkan path, waktu eksekusi, jumlah kata yang dikunjungi, dan langkah, yaitu panjang path - 1.

### Penjelasan Tambahan

Algoritma UCS yang digunakan pada penyelesaian masalah word ladder ini sebenarnya tidak berbeda dengan BFS biasa, karena hasil perhitungan cost  $g(n)$  yang digunakan pada UCS tidak memberikan pengaruh kepada order kata-kata pada queue, sehingga sama seperti BFS.

Misal, dari kata "crust", dibangkitkan "crest" dan "trust", dengan cost masing-masing kata sama, yaitu 1, masukkan ke queue menjadi

`[{"crest", 1, ["crust", "crest"]}, {"trust", 1, ["crust", "trust"]}].`

Dilakukan dequeue, dan dari "crest", dibangkitkan "chest" dan "crept" ("crust" tidak dibangkitkan kembali), dan diberi cost yang sama yaitu 2, dimasukkan ke queue menjadi

[{"trust", 1, ["crust", "trust"]}, {"chest", 2, ["crust", "crest", "chest"]}, {"chest", 2, ["crust", "crest", "chest"]}].

Sedangkan, jika menggunakan BFS, melakukan pembangkitan kata yang sama, akan menghasilkan queue

[{"trust", ["crust", "trust"]}, {"chest", ["crust", "crest", "chest"]}, {"chest", ["crust", "crest", "chest"]}].

Secara teoritis, algoritma Greedy Best First Search atau GBFS belum tentu menjamin solusi optimal, atau dengan kata lain menghasilkan shortest path, karena fungsi  $h(n)$  hanya menghitung jarak antara kata berikutnya ke kata tujuan tanpa mengetahui rute sebenarnya antara kata berikutnya ke kata tujuan, belum tentu admissible, sehingga GBFS bisa saja memberikan rute yang panjang atau bahkan tidak ditemukan rutanya karena terjebak dalam local optima. Walaupun begitu, GBFS jauh lebih cepat daripada algoritma lainnya.

Berbeda dengan GBFS, A\* menggunakan  $f(n) = g(n) + h(n)$  untuk membangkitkan cost, sehingga lebih admissible, memungkinkan optimal path, dan karena gabungan dari UCS dan GBFS, A\* lebih efisien dari UCS.



## Bab 3

### Implementasi Source Code

Karena menggunakan Java, beberapa class dengan attribute dan methodnya dibuat untuk menyelesaikan masalah Word Ladder ini.

Sebelum masuk ke algoritma utama, akan dijelaskan terlebih dahulu class-class pendukung program berjalan.

#### **Class Main**

Class Main menjadi eksekutor utama program Word Ladder ini.

#### **Attribute**

Tidak memiliki atribut.

#### **Method**

```
public static void main(String[] args) {}
```

Hanya memiliki public static void main, yang di dalamnya menjalankan SwingUtilities Java dan kemudian melakukan construct class GUI.

## Screenshot

```
import javax.swing.SwingUtilities;

// Main untuk memulai program
public class Main {
    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new GUI();
            }
        });
    }
}
```

## Class GUI

### Attribute

```
private JTextField startWordField, endWordField;
private JButton concatenateButton;
private JComboBox<String> algorithmComboBox;
private JTextArea pathTextArea;
```

Attribute-attribute di atas merupakan attribute dari Class JFrame, sebuah API Java yang berasal dari Abstract Window Toolkit (AWT) Java, yang digunakan untuk membuat GUI. JTextField digunakan memasukkan input, button untuk execute algoritma pencarian rute, JComboBox untuk memilih algoritma, dan JTextArea untuk menampilkan hasil

### Method

```
public GUI() {}
```

Method yang dimiliki hanyalah konstruktor class GUI, di mana akan memposisikan attribute-attribute yang dimiliki dalam bentuk grid, sekaligus memprogram button untuk mengambil input (dan validasi), melakukan algoritma pencarian rute, dan menampilkan hasilnya.

## Screenshot

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.net.*;
import java.net.event.*;

// Class GUI untuk menampilkan program
public class GUI extends JFrame {
    private JTextField startWordField, endWordField;
    private JButton concatenateButton;
    private JComboBox<String> algorithmComboBox;
    private JTextArea pathTextArea;

    public GUI() {
        // Menampilkan GUI
        setTitle("Word Ladder");
        setSize(width:1200, height:800);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        startWordField = new JTextField(columns:10);
        endWordField = new JTextField(columns:10);
        concatenateButton = new JButton(text:"Find the path!");

        pathTextArea = new JTextArea(rows:10, columns:30);
        pathTextArea.setEditable(false);
        pathTextArea.setLineWrap(wrap:true);
        pathTextArea.setWrapStyleWord(wrap:true);

        algorithmComboBox = new JComboBox<>();
        algorithmComboBox.addItem("UCS");
        algorithmComboBox.addItem("BFS");
        algorithmComboBox.addItem("A*");

        // Yang terpenting adalah, jika tombol ditekan, maka akan melakukan validasi
        // lalu jika valid, akan dilakukan pencarian path
        concatenateButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String startWord = startWordField.getText();
                String endWord = endWordField.getText();
                if (!EnglishDictionary.checkWord(startWord)) {
                    pathTextArea.setText("Start word is not an English word!");
                } else if (!EnglishDictionary.checkWord(endWord)) {
                    pathTextArea.setText("End word is not an English word!");
                } else if (startWord.length() != endWord.length()) {
                    pathTextArea.setText("The lengths are not the same!");
                } else {
                    String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();

                    // debug
                    System.out.println("Your start word is: " + startWord);
                    System.out.println("Your end word is: " + endWord);
                    System.out.println("Your algorithm is: " + selectedAlgorithm);
                    WordLadder path = new WordLadder(startWord, endWord, selectedAlgorithm);

                    // Result
                    String result = "Path: " + path.getPath().toString() + "\n" + "Steps: " + path.getSteps() + "\n" + "Words Visited: " + path.getWordVisitedAmount() + "\n" + "Time Elapsed: " + path.getTimeElapsed() + "ms";
                    pathTextArea.setText(result);
                }
            }
        });

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(rows:6, cols:1, hgap:5, vgap:5));
        panel.setBorder(new EmptyBorder(top:10, left:10, bottom:10, right:10));
        panel.add(new JLabel(text:"Enter start word:"));
        panel.add(startWordField);
        panel.add(new JLabel(text:"Enter end word:"));
        panel.add(endWordField);
        panel.add(new JLabel(text:"Select search algorithm:"));
        panel.add(algorithmComboBox);
        panel.add(concatenateButton);
        panel.add(new JScrollPane(pathTextArea));

        add(panel);
        setVisible(true);
    }
}
```

## Static Class EnglishDictionary

### Attribute

```
public static String dictionary = "dictionary/dictionary2.txt";
public static ArrayList<String> currLocalDictionary = new ArrayList<>();
```

Attribute dictionary menjadi file path txt di mana terdapat kumpulan kata bahasa Inggris yang digunakan dalam algoritma program, dengan setiap kata pada text dipisah dengan newline. Attribute currLocalDictionary adalah array yang berfungsi untuk menampung kumpulan kata, yang sudah di filter dari kamus di awal program (panjangnya sama dengan input), dan akan digunakan untuk iterasi pengecekan kata, dengan setiap kata yang sudah dikunjungi akan di remove dari array.

## Method

```
public static boolean checkWord(String word) {  
    // ...  
}  
  
public static void createDictionaryLength(int length) {  
    // ...  
}  
  
public static ArrayList<String> findWordsWithOneLetterDifference(String  
inputWord) {  
    // ...  
}  
  
public static int findAmountLettersDifference(String word1, String word2)  
{  
    // ...  
}
```

Method `checkWord` digunakan untuk mengecek apakah suatu kata ada pada kamus atau tidak, digunakan di awal program untuk validasi input. Method `createDictionaryLength` digunakan untuk iterasi kamus, mengambil kata-kata yang panjangnya sama dengan input, dan meletakkannya ke dalam array `currLocalDictionary`. Method `findWordsWithOneLetterDifference` mencari kata pada array `currLocalDictionary` yang berbeda 1 huruf dengan kata pada parameter, di mana array berisi kata yang berbeda 1 huruf dikembalikan dan dihapus dari `currLocalDictionary`. Method `findAmountLettersDifference` merupakan method untuk mencari berapa huruf yang berbeda di antara 2 kata.

## Screenshot

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

// Static Class EnglishDictionary, menyimpan text file berisi kamus bahasa Inggris secara lokal.
// Dibuat static, singleton, karena cukup hanya perlu ada satu.

// Versi terakhir, yaitu dengan menggunakan array (sebelumnya file temp) sebagai penampungan sementara.
public class EnglishDictionary {

    // Kamus disimpan secara lokal, ditampung pathnya relatif kepada .class.
    public static String dictionary = "dictionary/dictionary2.txt";
    // Array untuk menampung dan memproses kata-kata secara sementara
    public static ArrayList<String> currLocalDictionary = new ArrayList<>();

    // Fungsi untuk mengecek apakah ada kata di kamus, digunakan untuk validasi di awal.
    public static boolean checkWord(String word) {
        try (BufferedReader br = new BufferedReader(new FileReader(EnglishDictionary.dictionary))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.equalsIgnoreCase(word)) {
                    return true;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;
    }

    // Untuk mempercepat proses, dari kamus awal, diambil kata-kata yang berukuran sama dengan kata input (start dan end).
    // Dimasukkan ke array untuk diproses.
    public static void createDictionaryLength(int length) {
        try (BufferedReader br = new BufferedReader(new FileReader(EnglishDictionary.dictionary))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.trim().length() == length) {
                    EnglishDictionary.currLocalDictionary.add(line.trim());
                }
            }
            System.out.println("Dictionary array with words length " + length + " created successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

// Fungsi pembangkit, yaitu mencari kata-kata yang berbeda satu huruf dari kata input.
public static ArrayList<String> findWordsWithOneLetterDifference(String inputWord) {
    ArrayList<String> wordsWithOneDifference = new ArrayList<>();
    ArrayList<String> restOfWords = new ArrayList<>();
    for (String word : EnglishDictionary.currLocalDictionary) {
        if (EnglishDictionary.findAmountLettersDifference(word.toLowerCase(), inputWord) == 1) {
            // Jika berbeda satu huruf, dimasukkan ke array dan akan di return.
            wordsWithOneDifference.add(word.toLowerCase());
        } else {
            // Jika berbeda lebih dari satu huruf, dimasukkan ke array, yang diakhir akan mengganti kamus sementara sekarang.
            restOfWords.add(word);
        }
    }

    // Mengubah kamus sementara.
    // Hal ini dilakukan sebagai alternatif array kata-kata yang sudah di visit, sehingga bisa mempercepat proses.
    EnglishDictionary.currLocalDictionary.clear();
    EnglishDictionary.currLocalDictionary.addAll(restOfWords);

    return wordsWithOneDifference;
}

// Fungsi untuk mencari berapa banyak huruf yang beda antara kedua kata.
// Jika length tidak sama, mengembalikan -1, namun selama fungsi ini digunakan dalam eksekusi program,
// pasti length kedua kata akan sama (karena kamus sementara mempunyai panjang kata yang sama semua).
public static int findAmountLettersDifference(String word1, String word2) {
    if (word1.length() != word2.length()) {
        return -1;
    }
    int diff = 0;
    for (int i = 0; i < word1.length(); i++) {
        if (word1.charAt(i) != word2.charAt(i)) {
            diff++;
        }
    }
    return diff;
}
}

```

## Class Node

### Attribute

```

private String word;
private Integer cost;
private ArrayList<String> path;

```

Node merupakan sebuah class yang menyimpan sebuah string word, integer cost, dan array of string path. Node akan dibuat setelah word dibangkitkan dan dihitung costnya, dengan word adalah katanya, cost adalah hasil  $g(n)$ ,  $h(n)$ , atau  $f(n)$ , dan path adalah path kata sebelumnya ditambah kata tersebut.

### Method

```

public Node(String word, Integer cost, ArrayList<String> path) {
    // ...
}

public String getWord() {

```

```
    // ...  
}  
public int getCost() {  
    // ...  
}  
public ArrayList<String> getPath() {  
    // ...  
}
```

Method dari Node cukuplah sederhana, yaitu constructornya dan getter untuk masing-masing attributenya.

## Screenshot

```
import java.util.ArrayList;

// Class Node, menampung kata, cost, dan path yang dilaluinya.
// Dibuat class demi mengenkapsulasi dan supaya lebih mudah mengakses.
public class Node {

    private String word;
    private Integer cost;
    private ArrayList<String> path;

    public Node(String word, Integer cost, ArrayList<String> path) {

        this.word = word;
        this.cost = cost;
        this.path = path;
    }

    // Getter
    public String getWord() {
        return word;
    }

    public int getCost() {
        return cost;
    }

    public ArrayList<String> getPath() {
        return path;
    }
}
```

## Class WordLadder

### Attribute

```
private String startWord;
private String endWord;
private String algorithm;
private Integer wordsVisitedAmount;
private Integer steps;
private ArrayList<String> path;
private long timeElapsed;
```



Sekarang class yang menampung algoritma utamanya, yaitu class WordLadder. Attribute yang dimiliki dibagi menjadi 2, yaitu attribute dari input dan untuk outputnya. Attribute dari input, seperti yang pada class GUI, adalah string startWord dan string endWord sebagai kata awal dan kata akhir dan string algorithm yang digunakan untuk menentukan algoritma yang digunakan (UCS, GBFS, atau A\*). Attribute untuk output terdiri atas array path yaitu hasil path berdasarkan algoritma, integer wordsVisitedAmount yang menampung jumlah kata yang dibangkitkan dan dicek, integer steps yang menunjukkan jumlah langkah yang dilakukan, dan long timeElapsed yang menunjukkan lama waktu algoritma dijalankan dalam ms.

## Method

```
public WordLadder(String startWord, String endWord, String algorithm) {
    // ...
}
public ArrayList<String> getPath() {
    // ...
}
public Integer getWordVisitedAmount() {
    // ...
}
public Integer getSteps() {
    // ...
}
public long getTimeElapsed() {
    // ...
}
public int gn(int cost) {
    // ...
}
public int hn(String word) {
    // ...
}
public int fn(int cost, String word) {
    // ...
}
private void findPath() {
    // ...
}
```

```
}
```

Method pada WordLadder dapat dibagi menjadi 4 jenis. Yang pertama adalah constructornya sendiri, yang memasukkan input dari GUI ke dalam attribute WordLadder. Kedua adalah getternya, yang mengambil masing-masing attribute untuk output. Kemudian ada method gn, hn, dan fn. Sesuai bab sebelumnya, gn atau  $g(n)$  mengambil cost kata yang sedang diproses dan ditambah 1 (karena hanya 1 perbedaan huruf), hn atau  $h(n)$  menghitung perbedaan huruf antara kata yang dibangkitkan dengan kata tujuan, dan fn atau  $f(n) = gn + hn$ .

Akhirnya, method findPath, yang dibuat private demi abstraksi. findPath (detail dapat dilihat di screenshot),

1. Dimulai dengan menginstansiasi variabel-variabel yang diperlukan, yaitu waktu sekarang untuk perhitungan lama waktu eksekusi, array untuk menampung sementara kata-kata yang dibangkitkan, queue untuk urutan pemrosesan kata, sebuah node dan array untuk menampung node dan path sementara, serta boolean sebagai penanda apakah kata akhir sudah ditemukan atau belum.
2. Melakukan pengecekan apakah startWord = endWord, jika sama, program langsung mencatat waktu, mengembalikan wordsVisitedAmount dan steps sebagai 0, path sebagai array yang berisi hanya kata tersebut, dan timeElapsed berdasarkan selisih waktu yang dicatat.
3. Jika tidak sama, startWord dibuat sebagai node dengan cost 0 dan path berisi hanya kata itu, lalu dimasukkan ke queue.
4. Looping dilakukan seperti penjelasan pada bab sebelumnya, selama queue tidak kosong dan endWord tidak ditemukan pada array penampung kata-kata yang dibangkitkan, looping tetap berjalan.
5. Looping berisi pengambilan elemen terdepan dari queue, pembangkitan kata-kata berikutnya dan pengecekan apakah endWord terdapat di dalam array atau tidak. Jika ada, boolean dibuat true. Jika tidak, untuk setiap kata yang dibangkitkan, iterasi wordsVisitedAmount, dibuat node dan dimasukkan ke dalam queue, dengan cost menggunakan gn, hn, atau fn sesuai dengan algoritma yang digunakan.
6. Jika looping berhenti karena endWord ditemukan, catat waktu akhir dan hitung selisihnya sebagai timeElapsed, ambil path dari node kata sekarang ditambah

endWord, hitung steps berdasarkan panjang path - 1, dan kembalikan wordsVisitedAmount, steps, path, dan timeElapsed.

7. Jika looping berhenti karena queue kosong, hentikan perhitungan waktu dan hitung selisihnya sebagai timeElapsed, berarti path tidak ditemukan, kembalikan path sebagai empty array, steps sebagai 0, wordsVisitedAmount dan timeElapsed.

## Screenshot

```
import java.util.*;

// Ah yes, Class WordLadder, class utama untuk path finding.
// Bukan singleton karena, ya gapapa sih. Sepertinya bisa dibuat lebih dari satu dalam satu program
// sebagai history penggunaan.
public class WordLadder {

    // Atribut (self-explanatory)
    private String startWord;
    private String endWord;
    private String algorithm;

    private Integer wordsVisitedAmount;
    private Integer steps;
    private ArrayList<String> path;
    private long timeElapsed;

    // Constructor, yang hanya memasukkan parameter ke atribut, inisialisasi untuk eksekusi program
    // dan menjalankan path finding.
    public WordLadder(String startWord, String endWord, String algorithm) {
        EnglishDictionary.createDictionaryLength(startWord.length());
        this.startWord = startWord.toLowerCase();
        this.endWord = endWord.toLowerCase();
        this.algorithm = algorithm;
        this.wordsVisitedAmount = 0;
        this.steps = 0;
        this.path = new ArrayList<>();

        this.findPath();
    }

    // Getter
    public ArrayList<String> getPath() {
        return this.path;
    }

    public Integer getWordVisitedAmount() {
        return this.wordsVisitedAmount;
    }
}
```

```

public Integer getSteps() {
    return this.steps;
}

public long getTimeElapsed() {
    return this.timeElapsed;
}

// Fungsi g(n), penghitung cost untuk UCS.
// g(n) = cost sementara + banyak perbedaan huruf kata sekarang dengan kata berikutnya, yaitu 1.
public int gn(int cost) {
    return cost + 1;
}

// Fungsi h(n), fungsi evaluasi heuristik untuk GBFS.
// h(n) = banyak beda huruf antara word dengan endWord.
public int hn(String word) {
    return EnglishDictionary.findAmountLettersDifference(word, this.endWord);
}

// Fungsi f(n) = g(n) + h(n), fungsi penghitung cost untuk A*.
public int fn(int cost, String word) {
    return gn(cost) + hn(word);
}

```

```

// Ini dia fungsi utamanya, fungsi untuk mencari path antar kedua kata.
// Menggunakan versi array wordVisited yang selalu berubah.
private void findPath() {
    // Mulai waktu
    long startTime = System.currentTimeMillis();

    // Buat struktur data
    ArrayList<String> tempWords = new ArrayList<>(); // Untuk menampung kata-kata yang sudah dibangkitkan sementara.
    PriorityQueue<Node> queue = new PriorityQueue<>(Comparator.comparingInt(Node::getCost)); // Untuk queue pemrosesan kata.

    Node currNode; // Untuk node sementara
    ArrayList<String> currPath; // Untuk path sementara
    Boolean found = false; // Untuk memberi tahu apakah sudah ditemukan atau belum

    // Kalau startWord == endWord
    if (this.startWord.equals(this.endWord)) {
        this.path = new ArrayList<>();
        this.path.add(this.endWord);
        this.steps = this.path.size() - 1;
        System.out.println(x:"Found!");
    } else {
        // Kalau tidak

        // Buat node startNode
        currPath = new ArrayList<>();
        currPath.add(this.startWord);
        currNode = new Node(startWord, cost:0, currPath);

        // Masukkan ke queue untuk diproses
        queue.add(currNode);
    }
}

```

```

// Proses queue
while (!queue.isEmpty() && !found) {
    // Ambil isinya (paling pertama)
    currNode = queue.poll();

    // debug
    System.out.println("Current word: " + currNode.getWord());
    System.out.println();

    // Bangkitkan next word
    tempWords.clear();
    tempWords = EnglishDictionary.findWordsWithOneLetterDifference(currNode.getWord());

    // debug
    System.out.println("Found words: " + tempWords);
    System.out.println();

    if (tempWords.contains(this.endWord)) {
        // Kalau menemukan end word
        found = true;
    } else {
        // Kalau tidak
        for (String word : tempWords) {
            this.wordsVisitedAmount++;

            // Ambil path sekarang, ditambah kata yang baru
            currPath = new ArrayList<>();
            currPath.addAll(currNode.getPath());
            currPath.add(word);

            // Mengambil cost yang baru
            int newCost = 0;
            if (this.algorithm == "UCS") {
                newCost = gn(currNode.getCost());
            } else if (this.algorithm == "GBFS") {
                newCost = hn(word);
            } else {
                newCost = fn(currNode.getCost(), word);
            }
        }
    }
}

```

```

        // Membuat node, lalu dimasukkan ke prioQueue
        Node tempNode = new Node(word, newCost, currPath);
        queue.add(tempNode);
    }
}

if (found) {
    // Kalau semisal ketemu
    this.path = currNode.getPath();
    this.path.add(this.endWord);

    this.steps = this.path.size() - 1;

    System.out.println(x:"Found!");
} else {
    // Semisal tidak
    System.out.println(x:"Not found");
}
}

// Hentikan dan hitung waktu
long endTime = System.currentTimeMillis();
this.timeElapsed = endTime - startTime;

System.out.println("Path: " + this.path);
System.out.println("Steps: " + this.steps);
System.out.println("Words visited: " + this.wordsVisitedAmount);
System.out.println("Time elapsed: " + this.timeElapsed + "ms");
}
}

```

## How To Run

1. Melakukan git clone akan repository yang terdapat pada [Bab 7 Pranala](https://github.com/Julian-Caleb/Tucil3_13522099.git).

```
git clone https://github.com/Julian-Caleb/Tucil3_13522099.git
```

2. Masuk ke folder Tucil

```
cd Tucil3_13522099
```

3. Melakukan kompilasi program

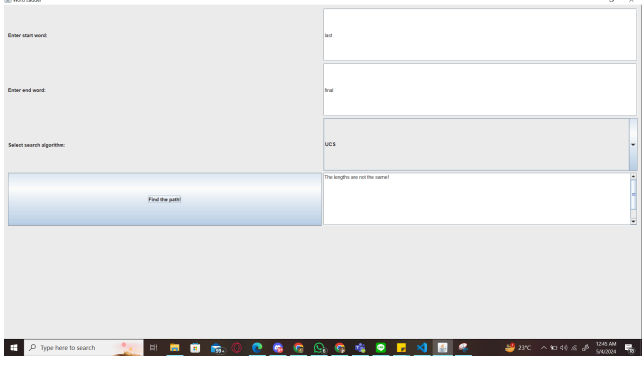
```
javac -d bin src/Main.java src/GUI.java src/EnglishDictionary.java
src/WordLadder.java src/Node.java
```

#### 4. Menjalankan Main pada folder bin

```
java -cp bin Main
```

## Bab 4 Testing

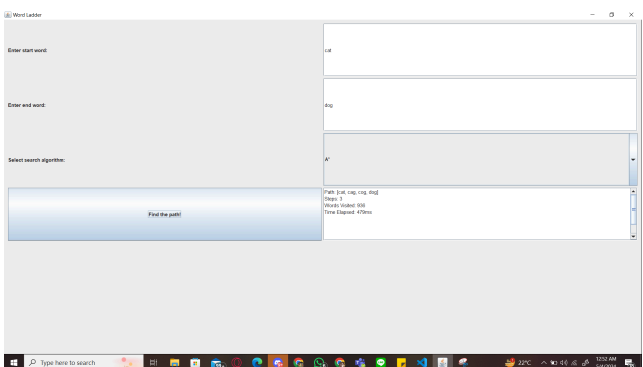
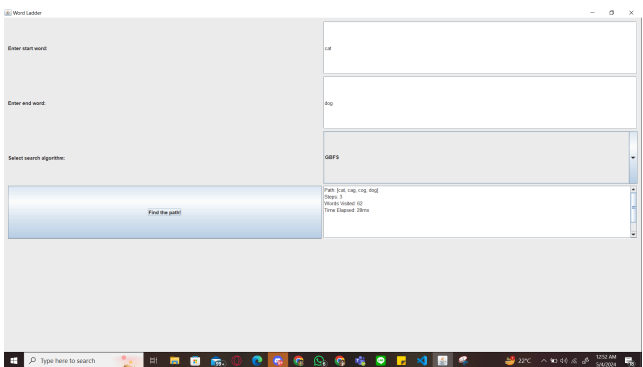
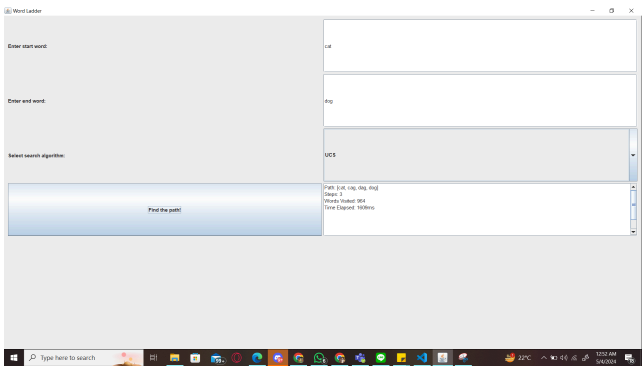
Kamus yang digunakan untuk testing disini adalah kamus dictionaryWordsAlpha.txt, sebelumnya bernama dictionary2.txt. Seluruh sumber kamus yang terdapat di source code dapat dilihat di [Bab 7 Pranala](#).

| Start Word | End Word | Screenshot   | Keterangan             |
|------------|----------|--|------------------------|
| abcd       | last     |    | Start word invalid     |
| last       | abcd     |  | End word invalid       |
| last       | final    |  | Length word tidak sama |



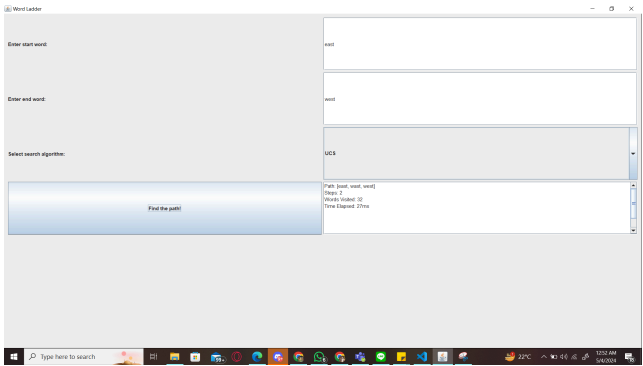
cat

dog



east

west







|        |
|--------|
| fungus |
| father |

|        |  |
|--------|--|
| counts |  |
| ladder |  |

[illegible][illegible][illegible]

|        |
|--------|
| father |
|--------|

ladder

[illegible]

|  |  |   |  |
|--|--|---|--|
|  |  | <div><div>Word Ladder</div><div><div>Enter start word:</div><div>Enter end word:</div><div>Select search algorithm:</div><div>Find the path!</div></div><div><div>Enter</div><div>Enter</div><div>GDPS</div><div>Path: [Enter, Enter, Enter, Enter, Enter]<br/>Steps: 5<br/>Words visited: 10<br/>Time Elapsed: 100ms</div></div></div> <div><div>Word Ladder</div><div><div>Enter start word:</div><div>Enter end word:</div><div>Select search algorithm:</div><div>Find the path!</div></div><div><div>Enter</div><div>Enter</div><div>A*</div><div>Path: [Enter, Enter, Enter, Enter, Enter]<br/>Steps: 5<br/>Words visited: 10<br/>Time Elapsed: 100ms</div></div></div> |  |
|--|--|---|--|

## Bab 5

### Hasil Analisis

Hasil dari testing menunjukkan bahwa:

1. Memory yang digunakan secara terurut dari yang paling kecil adalah GBFS < A\* < UCS. Hal ini dapat dilihat dari jumlah word yang dikunjungi.
2. Waktu eksekusi program dari yang tercepat adalah GBFS < A\* < UCS. Hal ini dikarenakan GBFS langsung mengambil cost terkecil dari kata berikutnya ke kata tujuan. Akan tetapi GBFS mempunyai kekurangan.
3. GBFS tidak selalu memberikan solusi optimal, dibandingkan A\* dan UCS. Hal ini dapat dilihat dari jumlah steps yang dilakukan oleh GBFS hampir selalu lebih banyak daripada A\* dan UCS. Selain itu, GBFS juga tidak menjamin solusi karena bisa saja terjebak dalam local optima.

## **Bab 6**

### **Kesimpulan**

Word Ladder dapat diselesaikan dengan algoritma pencarian rute UCS (Uniform Cost Search), GBFS (Greedy Best First Search), maupun  $A^*$ . Untuk menghitung costnya, algoritma UCS menggunakan fungsi  $g(n)$ , yaitu cost awal ditambah banyak huruf yang dibedakan dari satu kata ke kata lainnya, atau dengan kata lain ditambah 1, algoritma GBFS menggunakan fungsi  $h(n)$ , yaitu perbedaan huruf dari kata yang dibangkitkan ke kata tujuan, dan algoritma  $A^*$  menggunakan  $f(n)$ , yaitu gabungan dari  $g(n)$  dan  $h(n)$ . Algoritma UCS dan  $A^*$  selalu menghasilkan solusi yang optimal, sedangkan GBFS tidak, namun jauh lebih cepat dan lebih sedikit menggunakan memori dibandingkan UCS maupun  $A^*$ .

## Bab 7

### Pranala

Github: [https://github.com/Julian-Caleb/Tucil3\\_13522099](https://github.com/Julian-Caleb/Tucil3_13522099)

Kamus dictionaryWordsAlpha:

[https://github.com/dwyl/english-words/blob/master/words\\_alpha.txt](https://github.com/dwyl/english-words/blob/master/words_alpha.txt)

Kamus dictionaryScrabble:

<https://github.com/redbo/scrabble/blob/master/dictionary.txt>

Kamus dictionaryOracle:

<https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt>



### Lampiran

| 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. [Bonus]: Program memiliki tampilan GUI   | V  |       |

## 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>