

**LAPORAN TUGAS KECIL 3**  
**IF2211 STRATEGI ALGORITMA**  
**ALGORITMA BRUTEFORCE**  
**SEMESTER I TAHUN 2023/2024**



Disusun oleh:

Tazkia Nizami

13522032

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

**2024**

# **BAB I**

## **PENJELASAN ALGORITMA**

### **A. Landasan Teori**

Pada laporan ini, terdapat tiga algoritma utama yang digunakan, yaitu algoritma Unified Cost Search (UCS), Greedy Best First Search, dan A\*.

Algoritma UCS termasuk dalam kategori informed search yang menggunakan biaya aktual sebagai metrik untuk mengevaluasi sebuah node. UCS bekerja dengan cara menjelajahi semua jalur yang mungkin dari titik awal hingga menemukan jalur dengan biaya terendah. Algoritma ini menggunakan antrian prioritas untuk menyimpan simpul-simpul yang akan dikunjungi. Simpul dengan biaya terendah akan diprioritaskan untuk dikunjungi terlebih dahulu.

Algoritma Greedy Best-First Search (GBFS) merupakan salah satu algoritma pencarian terinformasi yang termasuk dalam kategori heuristik. Algoritma ini bekerja dengan memilih langkah selanjutnya yang tampak paling menjanjikan untuk mencapai tujuan akhir. GBFS menggunakan fungsi heuristik untuk memperkirakan jarak dari simpul saat ini ke simpul tujuan. Fungsi heuristik ini membantu algoritma untuk fokus pada jalur yang paling mungkin mengarah ke solusi optimal.

Algoritma A\* adalah salah satu algoritma pencarian terinformasi yang efisien untuk menemukan jalur terpendek antara dua titik dalam sebuah graf berbobot. Algoritma ini menggabungkan pencarian dengan algoritma UCS yang menggunakan biaya untuk mengunjungi node dan GBFS yang menggunakan heuristik untuk memperkirakan jarak dari simpul saat ini ke simpul tujuan.

Ketiga algoritma ini akan digunakan untuk menemukan solusi rute kata dalam permainan Word Ladder. 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.

### **B. Implementasi**

Pada pengimplementasiannya, terdapat beberapa fungsi yang digunakan untuk membantu penerapan ketiga algoritma tersebut, yaitu fungsi  $g(n)$ ,  $h(n)$ , dan  $f(n)$ .

Fungsi  $g(n)$  mengacu kepada fungsi untuk menghitung biaya dari start word hingga mencapai suatu word  $n$ . Setiap langkah akan dihitung dengan biaya 2 poin. Hal ini karena pada pencarian rute dari start word menuju end word, algoritma yang diterapkan harus menghasilkan rute terpendek. Pengimplementasian lebih detail akan terlihat pada implementasi algoritma A\*.

Fungsi  $h(n)$  mengacu kepada fungsi untuk menghitung heuristik dari word  $n$  ke end word. Heuristik yang dilakukan pada implementasi ini adalah membandingkan setiap huruf pada word  $n$  dengan end word. Kemudian, setiap huruf yang berbeda akan menambahkan 1 poin ke dalam hasil fungsi  $h(n)$ . Sehingga, jika terdapat word  $n$  yang sama persis dengan end word (contohnya word  $n$  = base dan end word = base), maka nilai fungsi  $h(n)$ nya adalah 0.

Fungsi  $f(n)$  adalah fungsi yang menentukan berapa besar poin / bobot yang dimiliki oleh sebuah word  $n$  atau Node. Semakin kecil nilainya, maka akan semakin menunjukkan bahwa  $n$  word yang paling optimal sebagai rute.

Pada algoritma UCS,  $f(n)$  dari sebuah Node adalah  $g(n)$ , atau dengan kata lain algoritma UCS hanya melihat berapa bobot sebuah Node berdasarkan berapa jarak dari start word hingga word  $n$  tersebut. Sedangkan pada algoritma Greedy Best First Search,  $f(n)$  dari sebuah Node adalah  $h(n)$ , atau algoritma GBFS ini hanya melihat bobot sebuah Node berdasarkan berapa banyak huruf yang berbeda antara word  $n$  dengan end word.

Pada algoritma A\*,  $f(n)$  dari sebuah Node adalah  $g(n) + h(n)$ , atau bisa dibilang bahwa algoritma A\* menerapkan UCS dan GBFS secara bersamaan. Algoritma A\* akan memandang bobot sebuah Node berdasarkan jarak dari start word hingga ke word  $n$  dan berapa banyak huruf yang berbeda antara word  $n$  dengan end word.

Pada algoritma UCS, berikut adalah tahapan implementasinya pada program pencarian rute terpendek untuk permainan Word Ladder.

1. Buatlah sebuah priority queue untuk menyimpan kata-kata yang akan dijelajahi
2. Tambahkan start word ke priority queue dengan nilai  $f(n) = g(n) = 0$  (biaya awal = 0).
3. Buatlah sebuah visited list untuk menyimpan kata-kata yang sudah dijelajahi.
4. Ambil kata terdepan (dengan nilai  $f(n)$ ) terkecil. Kita sebut sebagai “kata saat ini”.
5. Periksa apakah kata saat ini adalah end word. Jika iya, maka tampilkan hasilnya.
6. Masukkan ke dalam visited list.
7. Temukan semua kata baru yang memiliki perbedaan 1 huruf dari kata saat ini.
8. Untuk setiap kata baru:
  - a. Jika kata baru sudah ada pada visited list, abaikan.
  - b. Jika kata baru tersebut belum terdapat dalam visited list, masukkan ke dalam visited list dan masukan ke dalam priority queue.
  - c. Atur nilai  $f(n) = g(n)$  dari Node tersebut dengan  $f(n)$  kata saat ini ditambah 2.
9. Ulangi lagi langkah ke-4 dan seterusnya hingga mencapai akhir / kesimpulan tidak ditemukan.
10. Jika priority queue berisi kosong, maka artinya tidak ada kata yang bisa dibangkitkan lagi, maka tampilkan kesimpulan bahwa rute tidak bisa ditemukan.

Pada algoritma GBFS, berikut adalah tahapan implementasinya pada program pencarian rute terpendek untuk permainan Word Ladder.

1. Buatlah sebuah priority queue untuk menyimpan kata-kata yang akan dijelajahi
2. Tambahkan start word ke priority queue dengan nilai  $f(n) = h(n)$  sesuai dengan banyak perbedaan huruf antara start word dan end word. Untuk setiap huruf yang berbeda, tambahkan 1.
3. Buatlah sebuah visited list untuk menyimpan kata-kata yang sudah dijelajahi.
4. Ambil kata terdepan (dengan nilai  $f(n)$ ) terkecil. Kita sebut sebagai “kata saat ini”.
5. Hapus isi dari priority queue, hal ini untuk mencegah algoritma untuk memproses Node yang dibangkitkan dari Node sebelumnya.
6. Periksa apakah kata saat ini adalah end word. Jika iya, maka tampilkan hasilnya.
7. Masukkan ke dalam visited list.
8. Temukan semua kata baru yang memiliki perbedaan 1 huruf dari kata saat ini.
9. Untuk setiap kata baru:

- a. Jika kata baru sudah ada pada visited list, abaikan.
  - b. Jika kata baru tersebut belum terdapat dalam visited list, masukkan ke dalam visited list dan masukan ke dalam priority queue.
  - c. Atur nilai  $f(n) = h(n)$ , yaitu banyak perbedaan huruf antara kata saat ini dengan end word. Untuk setiap huruf yang berbeda, tambahkan 1.
10. Ulangi lagi langkah ke-4 dan seterusnya hingga mencapai akhir / kesimpulan tidak ditemukan.
  11. Jika priority queue berisi kosong, maka artinya tidak ada kata yang bisa dibangkitkan lagi, maka tampilkan kesimpulan bahwa rute tidak bisa ditemukan.

Pada algoritma UCS, berikut adalah tahapan implementasinya pada program pencarian rute terpendek untuk permainan Word Ladder.

1. Buatlah sebuah priority queue untuk menyimpan kata-kata yang akan dijelajahi
2. Tambahkan start word ke priority queue dengan nilai  $f(n) = g(n) + h(n)$  (dengan  $g(n) = 0$  karena biaya awal = 0).
3. Buatlah sebuah visited list untuk menyimpan kata-kata yang sudah dijelajahi.
4. Ambil kata terdepan (dengan nilai  $f(n)$ ) terkecil. Kita sebut sebagai “kata saat ini”.
5. Periksa apakah kata saat ini adalah end word. Jika iya, maka tampilkan hasilnya.
6. Masukkan ke dalam visited list.
7. Temukan semua kata baru yang memiliki perbedaan 1 huruf dari kata saat ini.
8. Untuk setiap kata baru:
  - a. Jika kata baru sudah ada pada visited list, abaikan.
  - b. Jika kata baru tersebut belum terdapat dalam visited list, masukkan ke dalam visited list dan masukan ke dalam priority queue.
  - c. Atur nilai  $f(n) = g(n) + h(n)$  untuk kata baru.
9. Ulangi lagi langkah ke-4 dan seterusnya hingga mencapai akhir / kesimpulan tidak ditemukan.
10. Jika priority queue berisi kosong, maka artinya tidak ada kata yang bisa dibangkitkan lagi, maka tampilkan kesimpulan bahwa rute tidak bisa ditemukan.

Pada algoritma A\*, nilai dari  $h(n)$  dari sebuah word n yang dibangkitkan ke dalam simpul hidup dapat dipastikan akan lebih rendah dari  $h^*(n)$ , atau nilai  $h(n)$  pertama antara start word dan end word. Hal ini karena pada pembangkitan simpul hidup, hanya simpul dengan huruf yang sudah sama tidak akan dibangkitkan lagi. Dari penjabaran tersebut, dapat disimpulkan bahwa heuristik yang digunakan pada algoritma A\* tersebut sudah admissible. Karena banyaknya huruf yang berbeda adalah batas minimum banyaknya transformasi yang perlu dilakukan dari suatu kata ke kata tujuan. Sehingga, algoritma A\* yang menggunakan heuristik akan menjamin solusi optimal.

Secara teoritis, algoritma A lebih efisien dibandingkan dengan algoritma UCS untuk menyelesaikan word ladder. Alasannya:

- A\* menggunakan heuristik, A\* mempertimbangkan perkiraan jarak ke kata akhir ( $h(n)$ ) dalam menentukan prioritas node. Ini membantu A\* fokus pada jalur yang lebih menjanjikan, sehingga mengurangi eksplorasi ruang pencarian yang tidak perlu.
- UCS hanya mempertimbangkan biaya aktual, UCS hanya mempertimbangkan biaya yang dikeluarkan sejauh ini ( $g(n)$ ). Hal ini berarti UCS dapat menjelajahi banyak jalur yang belum tentu mengarah ke kata akhir, membuatnya kurang efisien.

Secara teoritis, algoritma Greedy Best First Search (GBFS) tidak menjamin solusi optimal untuk persoalan word ladder. Alasannya:

- GBFS hanya mempertimbangkan heuristik, yang artinya GBFS hanya fokus pada Node dengan heuristik terendah ( $h(n)$  terkecil). Hal ini dapat menyebabkan GBFS terjebak dalam jalur yang tidak optimal, meskipun jalur lain yang lebih pendek mungkin ada, namun jalur tersebut tidak akan dijelajahi.
- Heuristik yang digunakan GBFS mungkin tidak selalu akurat, sehingga GBFS dapat memilih Node yang sebenarnya tidak mengarah ke solusi optimal.

## BAB II

### SOURCE CODE PROGRAM

Pada link repository, dapat terlihat bahwa inti dari program terdapat pada src/main/java/com/example dengan isinya sebagai berikut:

```
example/
├── interfaces/
│   ├── hn.java
│   └── gn.java
├── logic/
│   ├── BaseClass.java
│   ├── AStar.Java
│   ├── GreedyBestFirstSearch.java
│   └── Ucs.Java
├── tools/
│   ├── Kamus.java
│   ├── Node.java
│   └── NodePriorityQueue.java
├── App.java
└── MainController.java
```

#### A. Tools

Untuk membantu kinerja dari algoritma, maka diperlukan sebuah Class untuk mengatur segala kebutuhan tambahan untuk membantu algoritma. Setiap class yang dibangun sebagai pendukung dari algoritma disimpan di dalam folder tools. Terdapat 3 class yang dibuat, Kamus.java dibuat untuk menjadi kamus sebagai validator sebuah kata, Node.java dibuat untuk membuat sebuah Node dari sebuah kata, dan NodePriorityQueue untuk menjadi sebuah class yang menyimpan PriorityQueue untuk semua Node. Berikut adalah implementasi dari ketiganya.

Pada Kamus, akan terlihat bahwa ketika sebuah instance dibuat berdasarkan class Kamus, maka akan membaca sebuah file dict.txt. File tersebut berisi kata-kata dalam bahasa inggris yang akan digunakan untuk memvalidasi sebuah kata yang sudah digenerate pada program. Halaman untuk mengakses data tersebut dapat terlihat di lampiran.

Kamus.java

```
package com.example.tools;
```

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashSet;

public class Kamus {
    private HashSet<String> wordsSet;

    public Kamus(int panjangKata) throws IOException {
        System.out.println("Membangun kamus kata...");
        InputStream in = getClass().getResourceAsStream("/com/example/dict.txt");
        BufferedReader reader = new BufferedReader(new InputStreamReader(in));
        wordsSet = new HashSet<>();

        String kata;
        while ((kata = reader.readLine()) != null) {
            if (kata.length() == panjangKata) {
                wordsSet.add(kata);
            }
        }

        reader.close();
    }

    public boolean contains(String word) {
        word.toUpperCase();
        return wordsSet.contains(word);
    }

    public void printAllWords() {
        for (String word : wordsSet) {
            System.out.println(word);
        }
    }
}

```

Node.java

```

package com.example.tools;

import java.util.ArrayList;
import java.util.List;

```

```

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

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

    public String getWord() {
        return word;
    }

    public int getFn() {
        return fn;
    }

    public Node getParent() {
        return parent;
    }

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

    public int getCost() {
        int count = 0;
        Node current = this;
        while (current != null) {
            count += 2;
            current = current.getParent();
        }
        return count;
    }

    public ArrayList<String> structPath() {
        Node current = this;
        List<Node> path = new ArrayList<>();

        while (current != null) {
            path.add(current);
            current = current.getParent();
        }
    }
}

```



```

    }

    ArrayList<String> result = new ArrayList<>();
    for (int i = path.size() - 1; i >= 0; i--) {
        result.add(path.get(i).getWord());
    }
    return result;
}

public void printPath() {
    Node current = this;
    int num = 0;
    List<Node> path = new ArrayList<>();

    while (current != null) {
        path.add(current);
        current = current.getParent();
    }

    for (int i = path.size() - 1; i >= 0; i--) {
        System.out.println("Step " + num + ": " + path.get(i).getWord());
        num++;
    }
}
}

```

NodePriorityQueue.java

```

package com.example.tools;

import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

public class NodePriorityQueue {
    private PriorityQueue<Node> queue;
    private HashSet<String> kataDitemukan;

    public NodePriorityQueue() {
        Comparator<Node> nodeComparator = Comparator.comparingInt(Node::getFn);
        this.queue = new PriorityQueue<>(nodeComparator);
        this.kataDitemukan = new HashSet<>();
    }
}

```

```

    public void add(Node node) {
        if (!kataDitemukan.contains(node.getWord())) {
            queue.add(node);
            kataDitemukan.add(node.getWord());
        }
    }

    public Node poll() {
        return queue.poll();
    }

    public Node peek() {
        return queue.peek();
    }

    public boolean isEmpty() {
        return queue.isEmpty();
    }

    public void clear() {
        queue.clear();
    }

    public void printQueue() {
        System.out.println("\nIsi queue:");
        for (Node node : queue) {
            System.out.println(node.getWord() + " " + node.getFn());
        }
    }
}

```

## B. Interfaces

Sebelum membahas mengenai algoritma yang terdapat pada folder logic, terdapat folder interfaces yang berisi interface yang akan digunakan untuk class algoritma. Interface-interface tersebut dibuat untuk menerapkan konsep SOLID yaitu Interface Segregation. Terdapat 2 interface yang dibuat, yaitu interface untuk menghitung biaya sebuah node atau  $g(n)$  dan interface untuk menghitung nilai heuristiknya yaitu  $h(n)$ . Berikut adalah implementasi kedua file gn.java dan hn.java.

gn.java

```
package com.example.interfaces;
```

```
import com.example.tools.Node;

public interface gn {
    public int countGn(Node node);
}
```

hn.java

```
package com.example.interfaces;

public interface hn {
    public int countHn(String kataAwal, String kataAkhir, int panjangKata);
}
```

### C. Logic

Setiap perhitungan dan algoritma disimpan pada folder logic. Setiap perhitungan dan pencarian rute dengan menggunakan algoritma disimpan dan diatur pada folder tersebut. Di dalamnya, terdapat 4 buah file. Pertama adalah BaseClass yang berperan sebagai kelas dasar untuk ketiga algoritma. Kelas ini dibuat karena pengimplementasian class dari setiap algoritma memiliki banyak hal yang mirip, karena itu dibuatlah sebuah superclass yaitu BaseClass ini sehingga hanya perlu menuliskan 1 kali untuk implementasi ketiganya.

Kemudian terdapat 3 Class lainnya sesuai dengan algoritma yang ada, yaitu AStar.java, GreedyBestFirstSearch.java, dan UCS.java. Ketiga class ini masing-masing bertanggung jawab untuk pengimplementasian masing-masing algoritmanya. Berikut adalah implementasinya dalam bahasa Java.

BaseClass.java

```
package com.example.logic;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;

import com.example.tools.Kamus;
import com.example.tools.Node;
import com.example.tools.NodePriorityQueue;

public abstract class BaseClass {

    protected String kataAwal;
    protected String kataAkhir;
    protected int panjangKata;
```

```

private Kamus kamus;
protected HashSet<String> visited;
protected NodePriorityQueue pQueue;

protected int nodeDikunjungi;
protected List<String> pathHasil;

public BaseClass(String kataAwal, String kataAkhir, Kamus kamus) {
    this.kataAwal = kataAwal;
    this.kataAkhir = kataAkhir;
    this.panjangKata = kataAwal.length();
    this.kamus = kamus;
    this.nodeDikunjungi = 0;
    this.pQueue = new NodePriorityQueue();
    this.visited = new HashSet<>();
    this.pathHasil = new ArrayList<>();
}

public String getKataAwal() {
    return kataAwal;
}

public String getKataAkhir() {
    return kataAkhir;
}

public int getPanjangKata() {
    return panjangKata;
}

protected boolean isKata(String kata) {
    return kamus.contains(kata);
}

public int getNodeDikunjungi() {
    return nodeDikunjungi;
}

public List<String> getPathHasil() {
    return pathHasil;
}

protected List<Node> generateKata(Node parentNode) {
    List<Node> hasil = new ArrayList<>();

```

```

        for (int i = 0; i < getPanjangKata(); i++) {
            char[] chars = parentNode.getWord().toCharArray();
            if (chars[i] == kataAkhir.charAt(i)) {
                continue;
            }
            for (char c = 'A'; c <= 'Z'; c++) {
                if (c != chars[i]) {
                    chars[i] = c;
                    String kata = new String(chars);
                    if (isKata(kata) && !visited.contains(kata)) {
                        Node nodeBaru = new Node(kata, 0, parentNode);
                        nodeBaru.setFn(countFn(nodeBaru));
                        hasil.add(nodeBaru);
                    }
                }
            }
        }
        return hasil;
    }

    public void displayHasil(Node node) {
        System.out.println("\u001B[33m\n ----- HASIL -----
\u001B[0m");
        System.out.println("\u001B[35mJumlah node yang dikunjungi: \u001B[0m" +
nodeDikunjungi);
        System.out.println("\u001B[35mJumlah step: \u001B[0m" + ((int)
(node.getCost() / 2) - 1));
        pathHasil = node.structPath();
        node.printPath();
    }

    public abstract int countFn(Node node);

    public void driver() {
        Node nodeAwal = new Node(kataAwal, 0, null);
        nodeAwal.setFn(countFn(nodeAwal));
        pQueue.add(nodeAwal);
        boolean isStuckOrFound = false;

        while (!isStuckOrFound) {
            Node currentNode = pQueue.poll();

            if (currentNode == null) {

```

```

        System.out.println("\u001B[31m\nKata Tidak bisa diubah ke kata
lain lagi.\u001B[0m");
        System.out.println("\u001B[34mBanyak node yang dikunjungi:
\u001B[0m" + nodeDikunjungi);
        isStuckOrFound = true;
        break;
    }

    nodeDikunjungi++;

    if (currentNode.getWord().equals(kataAkhir)) {
        displayHasil(currentNode);
        isStuckOrFound = true;
        break;
    }

    for (Node node : generateKata(currentNode)) {
        pQueue.add(node);
        visited.add(node.getWord());
    }
}
}
}
}

```

AStar.java

```

package com.example.logic;

import com.example.interfaces.*;
import com.example.tools.Kamus;
import com.example.tools.Node;

public class AStar extends BaseClass implements hn, gn {

    public AStar(String kataAwal, String kataAkhir, Kamus kamus) {
        super(kataAwal, kataAkhir, kamus);
    }

    @Override
    public int countGn(Node node) {
        return node.getCost();
    }

    @Override

```

```

    public int countHn(String kataAwal, String kataAkhir, int panjangKata) {
        int h = 0;
        for (int i = 0; i < panjangKata; i++) {
            if (kataAwal.charAt(i) != kataAkhir.charAt(i)) {
                h++;
            }
        }
        return h;
    }

    @Override
    public int countFn(Node node) {
        return countGn(node) + countHn(node.getWord(), kataAkhir, panjangKata);
    }
}

```

GreedyBestFirstSearch.java

```

package com.example.logic;

import java.util.List;

import com.example.interfaces.hn;
import com.example.tools.Kamus;
import com.example.tools.Node;

public class GreedyBestFirstSearch extends BaseClass implements hn {

    public GreedyBestFirstSearch(String kataAwal, String kataAkhir, Kamus kamus)
    {
        super(kataAwal, kataAkhir, kamus);
    }

    @Override
    public int countHn(String kataAwal, String kataAkhir, int panjangKata) {
        int h = 0;
        for (int i = 0; i < panjangKata; i++) {
            if (kataAwal.charAt(i) != kataAkhir.charAt(i)) {
                h++;
            }
        }
        return h;
    }
}

```

```

@Override
public int countFn(Node node) {
    return countHn(node.getWord(), kataAkhir, panjangKata);
}

@Override
public void driver() {
    Node start = new Node(kataAwal, 0, null);
    pQueue.add(start);
    boolean isStuckOrFound = false;
    while (!isStuckOrFound) {
        Node current = pQueue.poll();
        if (current == null) {
            System.out.println("\u001B[31m\nTidak ada kata yang bisa di-
generate lagi.\u001B[0m");
            System.out.println("\u001B[34mBanyak node yang dikunjungi:
\u001B[0m" + nodeDikunjungi);
            isStuckOrFound = true;
            break;
        }
        nodeDikunjungi++;
        pQueue.clear();
        if (current.getWord().equals(kataAkhir)) {
            displayHasil(current);
            isStuckOrFound = true;
        } else {
            List<Node> generatedNodes = generateKata(current);
            for (Node node : generatedNodes) {
                if (visited.contains(node.getWord())) {
                    continue;
                }
                pQueue.add(node);
                visited.add(node.getWord());
            }
            // pQueue.printQueue();
        }
    }
}
}
}

```

UCS.java



```
package com.example.logic;

import com.example.interfaces.gn;
import com.example.tools.Kamus;
import com.example.tools.Node;

public class UCS extends BaseClass implements gn {

    public UCS(String kataAwal, String kataAkhir, Kamus kamus) {
        super(kataAwal, kataAkhir, kamus);
    }

    @Override
    public int countGn(Node node) {
        return node.getCost();
    }

    @Override
    public int countFn(Node node) {
        return countGn(node);
    }
}
```

## BAB III

### HASIL INPUT / OUTPUT

#### A. Hasil Pengujian Test 1

- Hasil Test 1 dengan UCS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'base' and the 'Kata Akhir' (End Word) is 'root'. The 'Algoritma' (Algorithm) is set to 'Uniform Cost Search'. The 'Cari Rute' (Find Route) button is highlighted. The results section shows: 'Waktu Eksekusi' (Execution Time) of 29 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 2590, and 'Panjang Solusi' (Solution Length) of 5. The solution path is listed as: Step 0: BASE, Step 1: BASK, Step 2: BOSK, Step 3: BOOK, Step 4: BOOT, Step 5: ROOT.

Field	Value
Kata Awal	base
Kata Akhir	root
Algoritma	Uniform Cost Search
Waktu Eksekusi	29 ms
Banyak Node yang dikunjungi	2590
Panjang Solusi	5

Step 0: BASE  
Step 1: BASK  
Step 2: BOSK  
Step 3: BOOK  
Step 4: BOOT  
Step 5: ROOT

- Hasil Test 1 dengan GBFS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'base' and the 'Kata Akhir' (End Word) is 'root'. The 'Algoritma' (Algorithm) is set to 'Greedy Best First Search'. The 'Cari Rute' (Find Route) button is highlighted. The results section shows: 'Waktu Eksekusi' (Execution Time) of 4 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 9, and 'Panjang Solusi' (Solution Length) of 8. The solution path is listed as: Step 0: BASE, Step 1: RASE, Step 2: ROSE, Step 3: ROBE, Step 4: ROBS, Step 5: ROCS, Step 6: ROCK, Step 7: ROOK, Step 8: ROOT.

Field	Value
Kata Awal	base
Kata Akhir	root
Algoritma	Greedy Best First Search
Waktu Eksekusi	4 ms
Banyak Node yang dikunjungi	9
Panjang Solusi	8

Step 0: BASE  
Step 1: RASE  
Step 2: ROSE  
Step 3: ROBE  
Step 4: ROBS  
Step 5: ROCS  
Step 6: ROCK  
Step 7: ROOK  
Step 8: ROOT

- Hasil Test 1 dengan A\*

Word Ladder Solver

WORD LADDER SOLVER

Kata Awal

base

Kata Akhir

root

Algoritma

A\*

Cari Rute

Waktu Eksekusi

17 ms

Banyak Node yang dikunjungi

814

Panjang Solusi

5

Step 0: BASE

Step 1: BASK

Step 2: BOSK

Step 3: BOOK

Step 4: BOOT

Step 5: ROOT

## B. Hasil Pengujian Test 2

- Hasil Test 2 dengan UCS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'earn' and the 'Kata Akhir' (End Word) is 'make'. The 'Algoritma' (Algorithm) is set to 'Uniform Cost Search'. A green 'Cari Rute' (Find Route) button is visible. The results section shows: 'Waktu Eksekusi' (Execution Time) of 9 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 395, and 'Panjang Solusi' (Solution Length) of 4. The solution path is listed as: Step 0: EARN, Step 1: YARN, Step 2: YARE, Step 3: MARE, Step 4: MAKE.

Parameter	Value
Kata Awal	earn
Kata Akhir	make
Algoritma	Uniform Cost Search
Waktu Eksekusi	9 ms
Banyak Node yang dikunjungi	395
Panjang Solusi	4

Step 0: EARN  
Step 1: YARN  
Step 2: YARE  
Step 3: MARE  
Step 4: MAKE

- Hasil Test 2 dengan GBFS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'earn' and the 'Kata Akhir' (End Word) is 'make'. The 'Algoritma' (Algorithm) is set to 'Greedy Best First Search'. A green 'Cari Rute' (Find Route) button is visible. The results section shows: 'Waktu Eksekusi' (Execution Time) of 1 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 5, and 'Panjang Solusi' (Solution Length) of 4. The solution path is listed as: Step 0: EARN, Step 1: BARN, Step 2: BARE, Step 3: MARE, Step 4: MAKE.

Parameter	Value
Kata Awal	earn
Kata Akhir	make
Algoritma	Greedy Best First Search
Waktu Eksekusi	1 ms
Banyak Node yang dikunjungi	5
Panjang Solusi	4

Step 0: EARN  
Step 1: BARN  
Step 2: BARE  
Step 3: MARE  
Step 4: MAKE

- Hasil Test 2 dengan A\*

Word Ladder Solver

WORD LADDER SOLVER

Kata Awal

earn

Kata Akhir

make

Algoritma

A\*

Cari Rute

Waktu Eksekusi

3 ms

Banyak Node yang dikunjungi

80

Panjang Solusi

4

Step 0: EARN

Step 1: EARS

Step 2: MARS

Step 3: MARE

Step 4: MAKE

### C. Hasil Pengujian Test 3

- Hasil Test 3 dengan UCS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'solar' and the 'Kata Akhir' (End Word) is 'panel'. The 'Algoritma' (Algorithm) is set to 'Uniform Cost Search'. A green 'Cari Rute' (Find Route) button is visible. The results section shows: 'Waktu Eksekusi' (Execution Time) of 23 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 1584, and 'Panjang Solusi' (Solution Length) of 6. The solution path is listed as: Step 0: SOLAR, Step 1: POLAR, Step 2: POLER, Step 3: POLES, Step 4: PALES, Step 5: PANES, Step 6: PANEL.

Parameter	Value
Kata Awal	solar
Kata Akhir	panel
Algoritma	Uniform Cost Search
Waktu Eksekusi	23 ms
Banyak Node yang dikunjungi	1584
Panjang Solusi	6

Step 0: SOLAR  
Step 1: POLAR  
Step 2: POLER  
Step 3: POLES  
Step 4: PALES  
Step 5: PANES  
Step 6: PANEL

- Hasil Test 3 dengan GBFS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'solar' and the 'Kata Akhir' (End Word) is 'panel'. The 'Algoritma' (Algorithm) is set to 'Greedy Best First Search'. A green 'Cari Rute' (Find Route) button is visible. The results section shows: 'Waktu Eksekusi' (Execution Time) of 4 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) of 8, and 'Panjang Solusi' (Solution Length) of 7. The solution path is listed as: Step 0: SOLAR, Step 1: POLAR, Step 2: POLER, Step 3: PALER, Step 4: PACER, Step 5: PACED, Step 6: PANED, Step 7: PANEL.

Parameter	Value
Kata Awal	solar
Kata Akhir	panel
Algoritma	Greedy Best First Search
Waktu Eksekusi	4 ms
Banyak Node yang dikunjungi	8
Panjang Solusi	7

Step 0: SOLAR  
Step 1: POLAR  
Step 2: POLER  
Step 3: PALER  
Step 4: PACER  
Step 5: PACED  
Step 6: PANED  
Step 7: PANEL

- Hasil Test 3 dengan A\*

Word Ladder Solver

WORD LADDER SOLVER

Kata Awal

solar

Kata Akhir

panel

Algoritma

A\*

Cari Rute

Waktu Eksekusi

3 ms

Banyak Node yang dikunjungi

143

Panjang Solusi

6

Step 0: SOLAR

Step 1: POLAR

Step 2: POLER

Step 3: PALER

Step 4: PALED

Step 5: PANED

Step 6: PANEL

#### D. Hasil Pengujian Test 4

- Hasil Test 4 dengan UCS

The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'moth' and the 'Kata Akhir' (End Word) is 'lamp'. The 'Algoritma' (Algorithm) is set to 'Uniform Cost Search'. The 'Cari Rute' (Find Route) button is highlighted. The results section shows: 'Waktu Eksekusi' (Execution Time) is 12 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) is 2059, and 'Panjang Solusi' (Solution Length) is 5. The solution path is listed as: Step 0: MOTH, Step 1: MATH, Step 2: LATH, Step 3: LATE, Step 4: LAME, Step 5: LAMP.

Parameter	Value
Kata Awal	moth
Kata Akhir	lamp
Algoritma	Uniform Cost Search
Waktu Eksekusi	12 ms
Banyak Node yang dikunjungi	2059
Panjang Solusi	5

Step 0: MOTH  
Step 1: MATH  
Step 2: LATH  
Step 3: LATE  
Step 4: LAME  
Step 5: LAMP

- Hasil Test 4 dengan GBFS

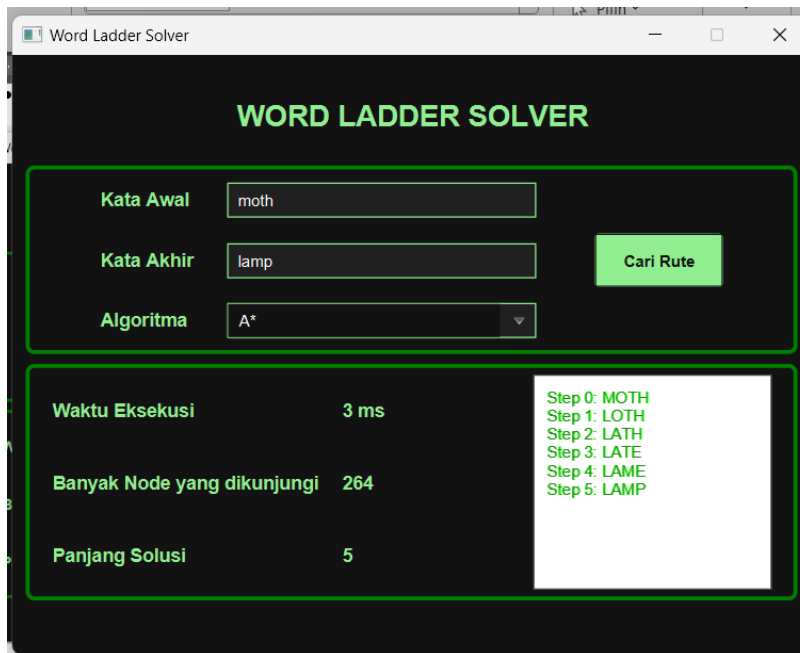
The screenshot shows the 'WORD LADDER SOLVER' interface. The 'Kata Awal' (Start Word) is 'moth' and the 'Kata Akhir' (End Word) is 'lamp'. The 'Algoritma' (Algorithm) is set to 'Greedy Best First Search'. The 'Cari Rute' (Find Route) button is highlighted. The results section shows: 'Waktu Eksekusi' (Execution Time) is 0 ms, 'Banyak Node yang dikunjungi' (Number of nodes visited) is 7, and 'Panjang Solusi' (Solution Length) is 6. The solution path is listed as: Step 0: MOTH, Step 1: LOTH, Step 2: LATH, Step 3: LAKH, Step 4: LAKE, Step 5: LAME, Step 6: LAMP.

Parameter	Value
Kata Awal	moth
Kata Akhir	lamp
Algoritma	Greedy Best First Search
Waktu Eksekusi	0 ms
Banyak Node yang dikunjungi	7
Panjang Solusi	6

Step 0: MOTH  
Step 1: LOTH  
Step 2: LATH  
Step 3: LAKH  
Step 4: LAKE  
Step 5: LAME  
Step 6: LAMP

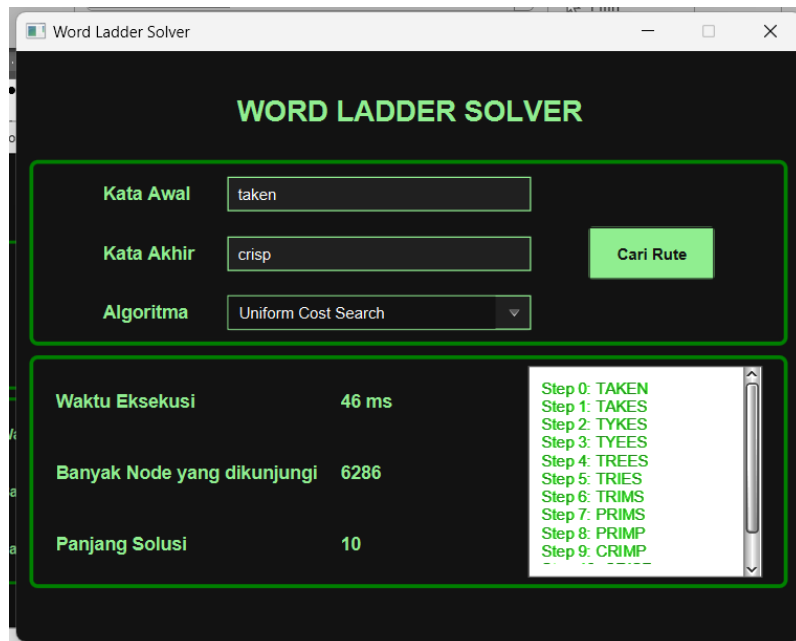
- Hasil Test 4 dengan A\*



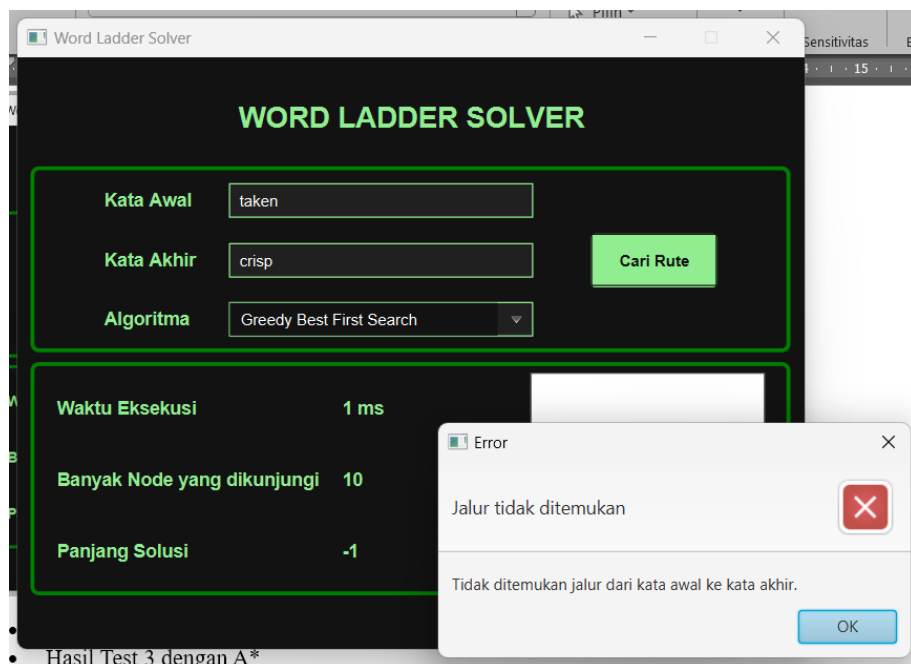


## E. Hasil Pengujian Test 5

- Hasil Test 5 dengan UCS

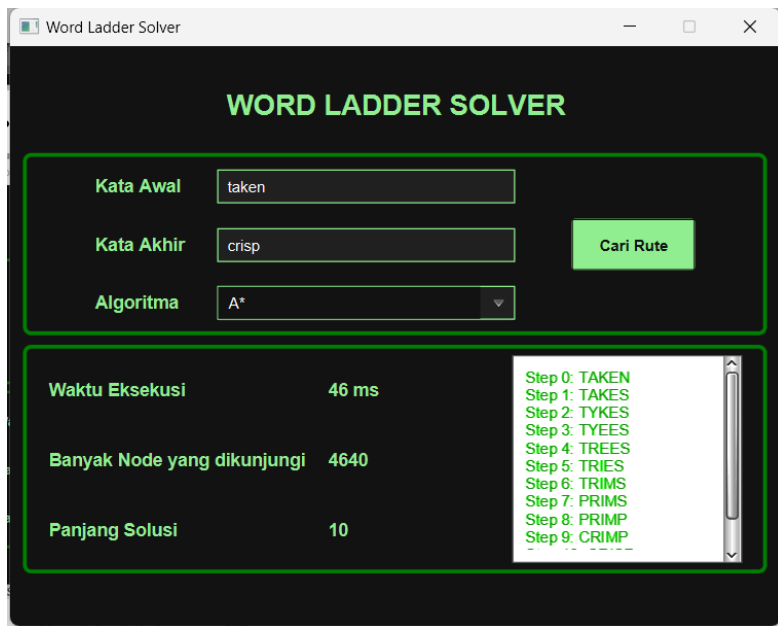


- Hasil Test 5 dengan GBFS



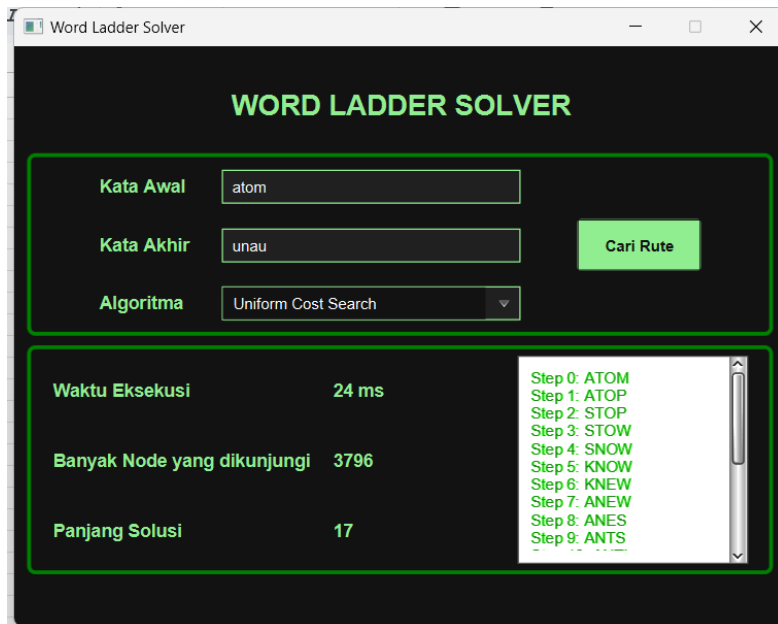
- Hasil Test 3 dengan A\*

- Hasil Test 5 dengan A\*

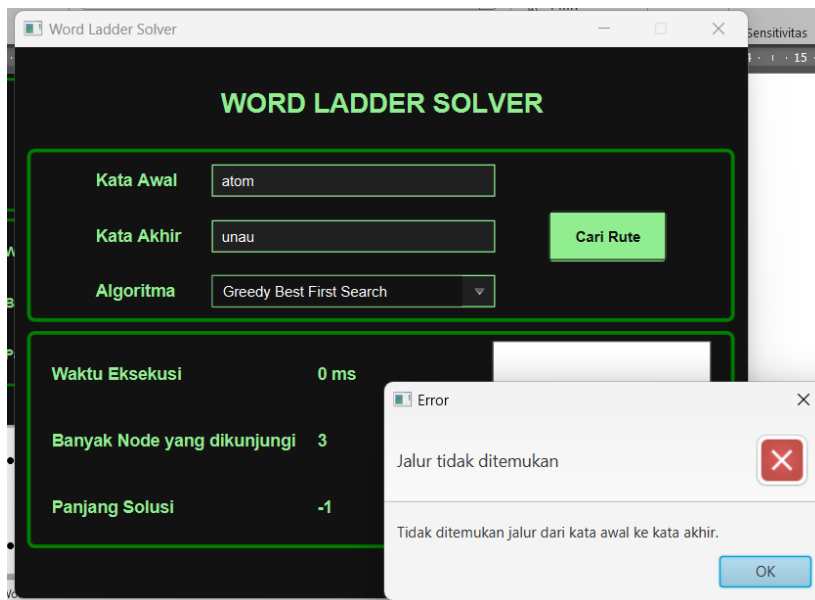


## F. Hasil Pengujian Test 6

- Hasil Test 6 dengan UCS



- Hasil Test 6 dengan GBFS



- Hasil Test 6 dengan A\*

Word Ladder Solver

WORD LADDER SOLVER

Kata Awal

atom

Kata Akhir

unau

Algoritma

A\*

Cari Rute

Waktu Eksekusi

17 ms

Banyak Node yang dikunjungi

3796

Panjang Solusi

17

Step 0: ATOM

Step 1: ATOP

Step 2: STOP

Step 3: STOW

Step 4: SNOW

Step 5: KNOW

Step 6: KNEW

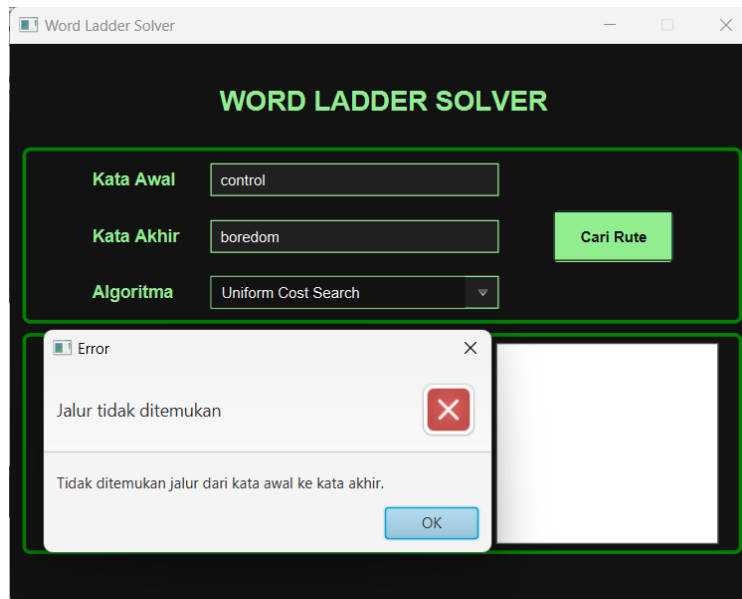
Step 7: ANEW

Step 8: ANES

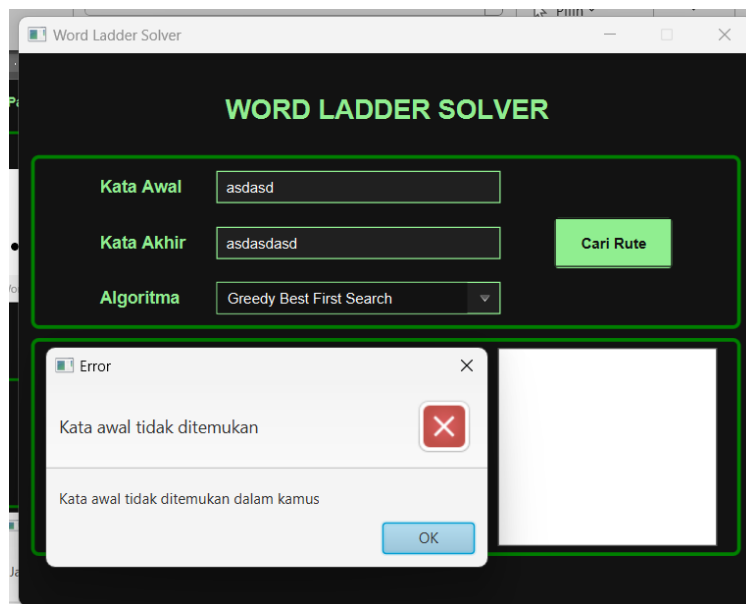
Step 9: ANTS

## G. Hasil Pengujian Tambahan

- Kata yang tidak ada jalurnya



- Start Word atau End Word tidak ada pada kamus



## H. Analisis Pengujian

- Optimalitas

Program yang dibuat sudah berhasil mendapatkan hasil yang optimal (setelah dibandingkan dengan Word Ladder Solver di internet) untuk algoritma A\* dan UCS. GBFS sendiri tidak selalu berhasil mendapatkan hasil yang optimal, bahkan memiliki kemungkinan lebih besar untuk tidak mendapatkan solusi karena tersendat di Node optimum lokal.

Salah satu hasil perbandingan hasil dapat dilihat pada gambar berikut.

The image displays two sets of comparisons between a locally developed Word Ladder Solver and an online version (ceptimus.co.uk). Each set includes a screenshot of the web interface and a screenshot of the local application window.

**Comparison 1: BAND to FLIP**

**Web Interface:**

- Start Word: BAND
- Target Word: FLIP
- Dictionary: World Scrabble
- Solve button
- 5-Step Solution 1: BAND, SAND, SAID, SLID, SLIP, FLIP
- 5-Step Solution 2: BAND, SAND, SAID, SLID, SLIP, FLIP
- Tips: 1. Hover mouse pointer over (or tap) solution words for extra information. 2. Different information for the start and target words in the two solutions. 3. Solve again to (perhaps) find alternative solutions.
- About...

**Local Application:**

- Kata Awal: band
- Kata Akhir: flip
- Algoritma: Uniform Cost Search
- Waktu Eksekusi: 16 ms
- Banyak Node yang dikunjungi: 2672
- Panjang Solusi: 5
- Step 0: BAND, Step 1: SAND, Step 2: SAID, Step 3: SLID, Step 4: SLIP, Step 5: FLIP

**Comparison 2: BURN to CLAW**

**Web Interface:**

- Start Word: BURN
- Target Word: CLAW
- Dictionary: World Scrabble
- Solve button
- 6-Step Solution 1: BURN, CURN, CURT, CUIT, CLIT, CLAT, CLAW
- 9-Step Solution 2: BURN, BURY, BUSY, BUST, BEST, BEAT, FEAT, FLAT, FLAW, CLAW
- Tips: 1. Hover mouse pointer over (or tap) solution words for extra information. 2. Different information for the start and target words in the two solutions. 3. Solve again to (perhaps) find alternative solutions.
- About...

**Local Application:**

- Kata Awal: burn
- Kata Akhir: claw
- Algoritma: A\*
- Waktu Eksekusi: 6 ms
- Banyak Node yang dikunjungi: 1011
- Panjang Solusi: 6
- Step 0: BURN, Step 1: BORN, Step 2: BORT, Step 3: BOAT, Step 4: BLAT, Step 5: BLAW, Step 6: CLAW

Namun, tentunya terdapat beberapa hasil perbandingan yang berbeda antara program yang telah dibuat dengan Solver di internet. Hal ini terjadi karena terdapat perbedaan referensi kamus antara program dan solver di internet, dan bukan karena tidak optimalan program yang sudah dibuat.

Kesimpulan bahwa algoritma A\* dan algoritma UCS menghasilkan solusi yang optimal juga diperoleh dari hasil pengujian dari 6 test case pada bagian sebelumnya. Pada pengujiannya, sebagian besar hasil

pencarian dengan kedua algoritma tersebut menghasilkan solusi rute yang sama (Lihat Test 1, 3, 4, 5, 6). Namun terdapat beberapa kejadian di mana solusi rute yang dihasilkan berbeda, namun tetap optimal karena solusi yang dihasilkan memiliki panjang rute yang sama (Lihat Test 2).

- Waktu Eksekusi

Waktu eksekusi untuk ketiga algoritma tersebut cenderung cepat, kurang dari 200ms. Hal ini membuat ketiganya sulit untuk dibandingkan karena terjadi sangat cepat. Namun, dengan melihat waktu eksekusi yang ditampilkan pada GUI, dapat terlihat bahwa diantara algoritma yang dapat mendapatkan solusi optimal, A\* dan UCS, A\* memiliki waktu eksekusi yang lebih cepat (lihat semua Test).

Faktor penyebab dari A\* yang memiliki waktu eksekusi yang relatif lebih cepat adalah banyak Node yang dikunjungi. Heuristik pada algoritma A\* membuat algoritma A\* tidak memeriksa Node sebanyak algoritma UCS. Hal ini tentunya mempercepat eksekusi program dan menjadi nilai tambah bagi algoritma A\* dibandingkan dengan UCS.

Di lain sisi, algoritma GBFS memiliki waktu eksekusi yang lebih cepat lagi dari A\*. Salah satu faktor utamanya adalah, kembali lagi, banyaknya Node yang diperiksa. GBFS hanya memeriksa Node sesuai dengan heuristik saja, kemudian melupakan Node yang sudah dibangkitkan sebelumnya. Hal ini membuat GBFS hanya memeriksa sedikit Node saja, yaitu Node yang paling mendekati end word dari Node yang saat ini sedang diekspan. Dalam pengujiannya, algoritma GBFS sering kali hanya memiliki waktu eksekusi 1 digit dalam ms, bahkan mendapatkan waktu kurang dari 0ms (akan tertulis 0ms pada GUI program, lihat Test 4 dan 6).

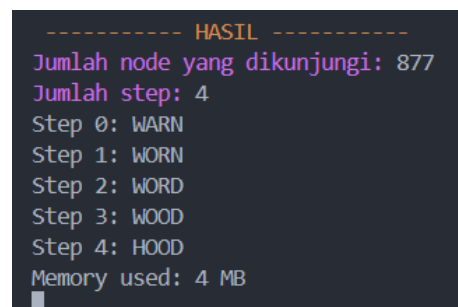
Namun, harga yang harus dibayar oleh algoritma GBFS adalah kehilangan kemampuan untuk “pasti menemukan solusi”, akan ada kemungkinan algoritma GBFS akan tersendat di tengah pencarian karena terjebak pada solusi optimum lokal, yaitu Node yang mendekati solusi, namun belum mencapai akhir, dan Node yang saat ini sedang diekspan tidak menghasilkan simpul hidup apapun (lihat Test 5 dan 6).

- Memori

Untuk memeriksa penggunaan memori pada program, perlu ditambahkan beberapa kode untuk menunjukkan berapa besar penggunaan memori, kemudian ditampilkan. Hasil perhitungan akan ditampilkan pada terminal tempat program dijalankan. Kode yang ditambahkan adalah sebagai berikut

```
long beforeUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
... {sisa kode program utama} ...
long afterUsedMem=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
System.out.println("Memory used = "afterUsedMem-beforeUsedMem);
```

Berikut adalah beberapa hasil keluaran saat pengujian



```
----- HASIL -----
Jumlah node yang dikunjungi: 877
Jumlah step: 4
Step 0: WARN
Step 1: WORN
Step 2: WORD
Step 3: WOOD
Step 4: HOOD
Memory used: 4 MB
```

← dengan menggunakan UCS



```

----- HASIL -----
Jumlah node yang dikunjungi: 91
Jumlah step: 4
Step 0: WARN
Step 1: WORN
Step 2: WORD
Step 3: WOOD
Step 4: HOOD
Memory used: 0 MB

```

← Dengan menggunakan A\*

```

Tidak ada kata yang bisa di-generate lagi.
Banyak node yang dikunjungi: 11
Memory used: 0 MB

```

← Dengan menggunakan GBFS

Dapat dilihat bahwa untuk Test tersebut, penggunaan memorinya sangat kecil, dan sangat berkolerasi dengan jumlah node yang dikunjungi. Hal ini terjadi karena untuk setiap Node yang dikunjungi, maka akan membangkitkan banyak simpul hidup yang mungkin. Penyimpanan simpul hidup itulah yang membuat penggunaan memori menjadi lebih besar. Berikut adalah hasil untuk kasus yang memiliki Banyak node yang dikunjungi lebih banyak.

```

----- HASIL -----
Jumlah node yang dikunjungi: 3591
Jumlah step: 10
Step 0: CRISP
Step 1: CRIMP
Step 2: PRIMP
Step 3: PRIMS
Step 4: TRIMS
Step 5: TRIES
Step 6: TREES
Step 7: TYEES
Step 8: TYKES
Step 9: TAKES
Step 10: TAKEN
Memory used: 21 MB

```

← dengan menggunakan UCS

Dapat terlihat bahwa ketika jumlah Node yang dikunjungi lebih banyak, maka memori yang digunakan juga meningkat. Untuk algoritma UCS dan A\* yang relatif memiliki Node yang dikunjungi banyak, memori yang digunakan akan cenderung lebih banyak daripada GBFS.

Untuk algoritma GBFS, penggunaan memori tentunya akan jauh lebih kecil. Hal ini karena Node yang dikunjungi jauh lebih sedikit dan untuk algoritma ini juga tidak menyimpan simpul hidup dari Node yang sudah dibangkitkan sebelumnya. Berikut adalah contohnya.

```
----- HASIL -----  
Jumlah node yang dikunjungi: 6  
Jumlah step: 5  
Step 0: WIDE  
Step 1: BIDE  
Step 2: BODE  
Step 3: BOLE  
Step 4: BOLT  
Step 5: BOOT  
Memory used: 0 MB
```

← dengan menggunakan GBFS

Namun, pada saat pengujian perlu berhati-hati karena terkadang muncul hasil memori negatif. Kejadian tersebut bisa dianggap bug dan diabaikan saja. Berikut contohnya.

```
Step 1: BASK  
Step 2: BOSK  
Step 3: BOOK  
Step 4: BOOT  
Step 5: ROOT  
Memory used: 4 MB  
Membangun kamus kata...  
  
----- HASIL -----  
Jumlah node yang dikunjungi: 2590  
Jumlah step: 5  
Step 0: BASE  
Step 1: BASK  
Step 2: BOSK  
Step 3: BOOK  
Step 4: BOOT  
Step 5: ROOT  
Memory used: -52 MB
```

## BAB IV

### PENJELASAN IMPLEMENTASI BONUS

Bonus GUI diimplementasikan dengan memanfaatkan teknologi Jawa FX dan dibantu dengan manajemen proyek Maven. JavaFX adalah framework UI (User Interface) yang memungkinkan pengembang untuk membuat aplikasi desktop dan mobile yang kaya fitur dengan Java. JavaFX memiliki beberapa keunggulan, Aplikasi JavaFX dapat dijalankan di berbagai platform seperti Windows, macOS, Linux, Android, dan iOS. Kemudian, JavaFX menggunakan bahasa deklaratif untuk mendefinisikan UI, yang memudahkan pembuatan dan pemeliharaan UI yang kompleks. Lalu, JavaFX menyediakan berbagai kontrol UI seperti tombol, label, tabel, dan grafik, serta efek visual dan animasi yang canggih. Terakhir, JavaFX terintegrasi dengan baik dengan Java, memungkinkan pengembang untuk memanfaatkan kekuatan bahasa pemrograman Java dalam aplikasi UI mereka.

Maven adalah alat manajemen proyek build automation untuk Java. Maven membantu pengembang dalam beberapa poin, yaitu:

- **Mengelola Ketergantungan:** Maven memungkinkan pengembang untuk mendeklarasikan dan mengelola dependensi proyek dengan mudah, termasuk library JavaFX.
- **Membangun Proyek:** Maven menyediakan mekanisme build yang konsisten dan terdokumentasi untuk membangun dan mengemas aplikasi JavaFX.
- **Menyederhanakan Pengembangan:** Maven dapat mengotomatiskan tugas-tugas umum seperti pengujian unit, integrasi berkelanjutan, dan penerapan, sehingga pengembang dapat fokus pada pengembangan aplikasi.

Secara garis besar, file App.java berperan sebagai titik mulai program, dan MainController bertanggung jawab sebagai class yang mengatur berinteraksi langsung dengan UI yang dibuat dengan JawaFX dan handle setiap input dan output dari program. Berikut adalah implementasinya

App.java

```
package com.example;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;
```

```

@Override
public void start(Stage stage) throws IOException {
    scene = new Scene(loadFXML("main"), 600, 450);
    stage.setTitle("Word Ladder Solver");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();
}

static void setRoot(String fxml) throws IOException {
    scene.setRoot(loadFXML(fxml));
}

private static Parent loadFXML(String fxml) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml +
".fxml"));
    return fxmlLoader.load();
}

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

```

MainController.java

```

package com.example;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.example.logic.AStar;
import com.example.logic.GreedyBestFirstSearch;
import com.example.logic.UCS;
import com.example.tools.Kamus;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;

```

```

import javafx.scene.control.TextField;
import javafx.scene.text.Text;

public class MainController {

    ObservableList<String> algorithmList =
FXCollections.observableArrayList("A*", "Greedy Best First Search",
    "Uniform Cost Search");
    private String kataAwal;
    private String kataAkhir;
    private String algoritma;

    @FXML
    private ComboBox<String> algorithmComboBox;

    @FXML
    private TextField textKataAwal;

    @FXML
    private TextField textKataAkhir;

    @FXML
    private Label timeLabel;

    @FXML
    private Label banyakNodeLabel;

    @FXML
    private Label panjangJalurLabel;

    @FXML
    private Text pathText;

    @FXML
    private void initialize() {
        algorithmComboBox.setItems(algorithmList);
        algorithmComboBox.getSelectionModel().select(0);
    }

    @FXML
    private void solve() throws IOException {
        kataAwal = textKataAwal.getText().toUpperCase();
        kataAkhir = textKataAkhir.getText().toUpperCase();
        algoritma = algorithmComboBox.getValue();
    }
}

```

```

        reset();

        List<String> pathHasil = new ArrayList<>();
        StringBuilder path = new StringBuilder();

        Kamus kamus = new Kamus(kataAwal.length());

        if (!kamus.contains(kataAwal)) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Kata awal tidak ditemukan");
            alert.setContentText("Kata awal tidak ditemukan dalam kamus");
            alert.showAndWait();
            return;
        }

        if (!kamus.contains(kataAkhir)) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Kata akhir tidak ditemukan");
            alert.setContentText("Kata akhir tidak ditemukan dalam kamus");
            alert.showAndWait();
            return;
        }

        long startTime = System.currentTimeMillis();
        switch (algoritma) {
            case "A*":
                AStar aStar = new AStar(kataAwal, kataAkhir, kamus);
                aStar.driver();
                banyakNodeLabel.setText(String.valueOf(aStar.getNodeDikunjungi()));
            );
                panjangJalurLabel.setText(String.valueOf(aStar.getPathHasil().size() - 1));
                pathHasil = aStar.getPathHasil();
                break;
            case "Greedy Best First Search":
                GreedyBestFirstSearch greedyBestFirstSearch = new
GreedyBestFirstSearch(kataAwal, kataAkhir, kamus);
                greedyBestFirstSearch.driver();
                banyakNodeLabel.setText(String.valueOf(greedyBestFirstSearch.getNodeDikunjungi()));
                panjangJalurLabel.setText(String.valueOf(greedyBestFirstSearch.getPathHasil().size() - 1));

```

```

        pathHasil = greedyBestFirstSearch.getPathHasil();
        break;
    case "Uniform Cost Search":
        UCS ucs = new UCS(kataAwal, kataAkhir, kamus);
        ucs.driver();
        banyakNodeLabel.setText(String.valueOf(ucs.getNodeDikunjungi()));
        panjangJalurLabel.setText(String.valueOf(ucs.getPathHasil().size(
) - 1));

        pathHasil = ucs.getPathHasil();
        break;
    default:
        break;
}
long endTime = System.currentTimeMillis();

if (pathHasil.isEmpty()) {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Error");
    alert.setHeaderText("Jalur tidak ditemukan");
    alert.setContentText("Tidak ditemukan jalur dari kata awal ke kata
akhir.");
    alert.showAndWait();
    return;
}

int i = 0;
for (String kata : pathHasil) {
    path.append("Step ").append(i).append(":
").append(kata).append("\n");
    i++;
}

pathText.setText(path.toString());

long executionTime = endTime - startTime;
timeLabel.setText(String.valueOf(executionTime) + " ms");
}

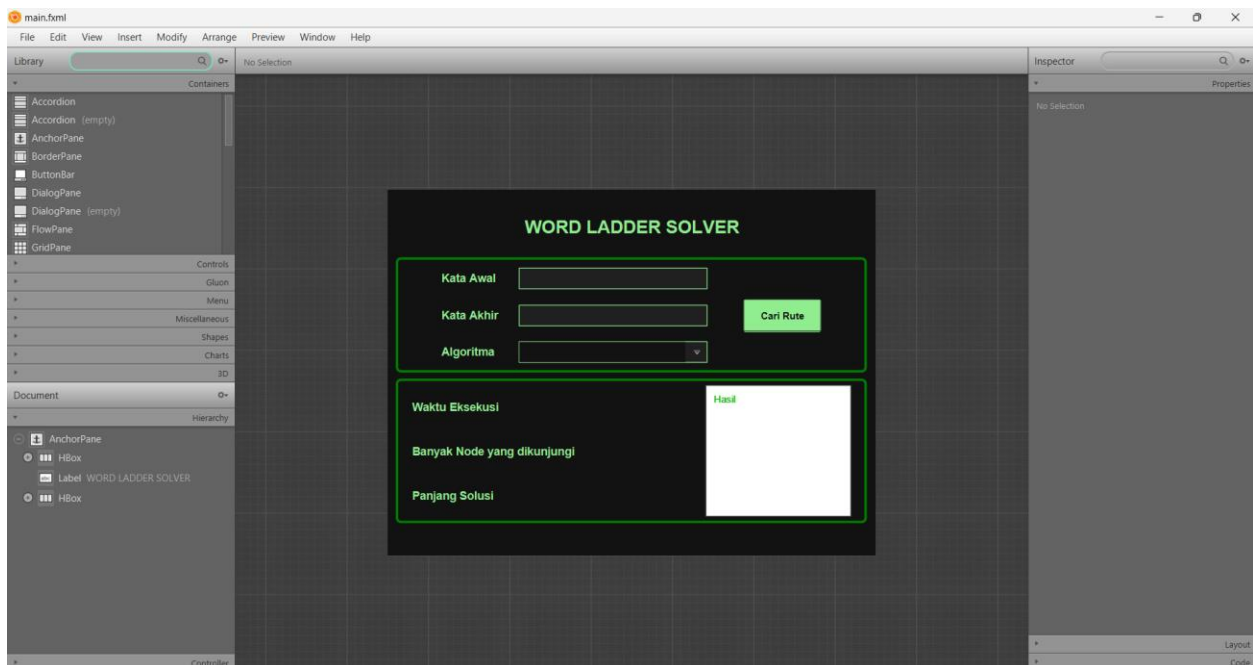
private void reset() {
    pathText.setText("");
    timeLabel.setText("");
    banyakNodeLabel.setText("");
    panjangJalurLabel.setText("");
}

```

```
}
```

Selain itu, untuk tampilan sendiri diatur pada file yang berbeda, file tersebut terdapat pada folder resource di tautan repositori, yaitu pada file main.fxml. File dengan ekstensi .fxml (Form eXtensible Markup Language) adalah file teks yang digunakan untuk mendefinisikan antarmuka pengguna (UI) dalam aplikasi JavaFX. File ini berisi kode XML yang mendeskripsikan elemen UI seperti tombol, label, tabel, dan layout aplikasi. Jika dilihat sekilas, bentuknya akan terlihat mirip seperti HTML, namun dengan komponen dan tag yang berbeda.

File tersebut dibuat dengan bantuan sebuah program bernama SceneBuilder. Scene Builder menyediakan antarmuka drag-and-drop yang intuitif dan memungkinkan pengembang untuk melihat UI mereka secara real-time saat mereka mendesainnya. Berikut adalah tampilan program SceneBuilder ketika digunakan untuk mendesain tampilan program.



Hasilnya akan disimpan dalam format .fxml seperti main.fxml berikut

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ComboBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
```



```

<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="450.0"
    prefWidth="600.0" stylesheets="@main_style.css"
xmlns="http://javafx.com/javafx/21"
    xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.MainController">
    <HBox alignment="CENTER" layoutX="10.0" layoutY="83.0" prefHeight="141.0"
prefWidth="579.0"
        style="-fx-border-color: green; -fx-border-width: 3px;"
styleClass="input-hbox">
        <GridPane prefHeight="102.0" prefWidth="326.0">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES"
maxWidth="157.20001220703125" minWidth="10.0"
                    prefWidth="94.0"/>
                <ColumnConstraints hgrow="SOMETIMES"
maxWidth="233.20001220703125" minWidth="10.0"
                    prefWidth="230.0"/>
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
            </rowConstraints>
            <Label prefHeight="18.0" prefWidth="101.0" text="Kata Awal"/>
            <Label text="Kata Akhir" GridPane.rowIndex="1"/>
            <TextField fx:id="textKataAwal" prefHeight="26.0" prefWidth="154.0"
GridPane.columnIndex="1"/>
            <TextField fx:id="textKataAkhir" prefHeight="26.0" prefWidth="154.0"
GridPane.columnIndex="1"
                GridPane.rowIndex="1"/>
            <ComboBox fx:id="algorithmComboBox" prefHeight="26.0"
prefWidth="239.0" GridPane.columnIndex="1"
                GridPane.rowIndex="2"/>
            <Label text="Algoritma" GridPane.rowIndex="2"/>

```

```

        </GridPane>
        <Button mnemonicParsing="false" onAction="#solve" prefHeight="40.0"
prefWidth="96.0" text="Cari Rute"
            textFill="#6e0b0b">
            <HBox.margin>
                <Insets left="45.0"/>
            </HBox.margin>
        </Button>
    </HBox>
    <Label alignment="CENTER" contentDisplay="CENTER" layoutX="74.0"
layoutY="25.0" prefHeight="40.0" prefWidth="452.0"
        styleClass="title-label" text="WORD LADDER SOLVER"
textAlignment="CENTER">
        <font>
            <Font name="Ravie" size="28.0"/>
        </font>
    </Label>
    <HBox alignment="CENTER" layoutX="10.0" layoutY="232.0" prefHeight="177.0"
prefWidth="579.0"
        style="-fx-border-color: green; -fx-border-width: 3px;"
styleClass="bottom-hbox">
        <padding>
            <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"/>
        </padding>
        <GridPane prefHeight="151.0" prefWidth="326.0" styleClass="info-label">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES"
maxWidth="232.00001220703126" minWidth="10.0"
                    prefWidth="214.3999755859375"/>
                <ColumnConstraints hgrow="SOMETIMES"
maxWidth="109.60001220703128" minWidth="10.0"
                    prefWidth="104.80002441406253"/>
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES"/>
            </rowConstraints>
            <Label text="Waktu Eksekusi"/>
            <Label text="Banyak Node yang dikunjungi" GridPane.rowIndex="1"/>
            <Label text="Panjang Solusi" GridPane.rowIndex="2"/>

```

```

        <Label fx:id="timeLabel" GridPane.columnIndex="1"/>
        <Label fx:id="banyakNodeLabel" GridPane.columnIndex="1"
GridPane.rowIndex="1"/>
        <Label fx:id="panjangJalurLabel" GridPane.columnIndex="1"
GridPane.rowIndex="2"/>
    </GridPane>
    <ScrollPane blendMode="COLOR_DODGE" prefHeight="142.0" prefWidth="178.0"
styleClass="solution-area">
        <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0"/>
        </padding>
        <HBox.margin>
            <Insets left="35.0"/>
        </HBox.margin>
        <Text fx:id="pathText" fill="#7ddc68" strokeType="OUTSIDE"
strokeWidth="0.0" text="Hasil"
            textOrigin="TOP"/>
    </ScrollPane>
</HBox>
</AnchorPane>

```

## LAMPIRAN

### A. Tautan Penting

Link repository Github: [https://github.com/TazakiN/Tucil3\\_13522032](https://github.com/TazakiN/Tucil3_13522032)

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

### B. Checklist Program

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