

LAPORAN TUGAS KECIL 3

IF2211 STRATEGI

ALGORITMA

Memecahkan Persoalan Word Ladder Menggunakan Algoritma Uniform Cost Search (UCS), Greedy Best First Search, dan A*



Disusun oleh:

Muhammad Syarafi Akmal 13522076

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan

Informatika Institut Teknologi Bandung

2024

I. Analisis dan Implementasi Algoritma *Uniform Cost Search*, *Greedy BFS*, dan *A**

Word Ladder merupakan persoalan dimana kita harus mencari kata-kata penghubung dari kata mula ke kata tujuan dengan batasan hanya boleh mengganti 1 huruf setiap iterasi.

Persoalan ini diselesaikan menggunakan 3 dari algoritma-algoritma *route planning* yaitu *UCS*, *Greedy BFS*, dan *A**. Ketiga algoritma ini memiliki keterhubungan yang akan dijelaskan pada analisis berikut:

a. Uniform Cost Search

Algoritma ini kurang lebih merupakan perpaduan dari *Uninformed Search* IDS dan BFS yang telah ditambah dengan beberapa *constraints* untuk meminimalisir langkah yang akan diambil menuju goal. *Constraints* tambahan yang diberikan berupa pembatasan ekspansi/pembangkitan pada algoritma berdasarkan sebuah fungsi *cost* yang biasanya diambil dari sebuah nilai harga acuan terhadap *root* kepada masing-masing *child* hasil ekspansi (Simpul Hidup). Ekspansi akan dilakukan berdasarkan kata yang diperoleh melalui nilai fungsi *cost* yang memiliki nilai paling minimal dari simpul hidup (Simpul E). Algoritma ini juga membatasi pengunjungan ulang dari hasil ekspansi sehingga tidak mungkin terjadi kondisi *circular looping*. Hal ini membuat UCS jauh lebih efisien dibandingkan BFS dan IDS dengan hasil jalur yang sama (optimum). Maka, pada algoritma ini akan ada tiga struktur data utama yaitu, simpul E, simpul hidup (Priority Queue), *visited nodes* (Hashset).

Pada kasus persoalan ini, fungsi cost atau $g(n)$ adalah jarak kata hasil ekspansi dari kata mula atau *root*. Jarak disini dapat diperoleh dari jumlah perbedaan huruf dari kata hasil ekspansi terhadap kata *root*. Berikut rumus $g(n)$ dalam perumpamaan himpunan huruf penyusun kata:

$$\begin{aligned} \text{start_word}(\text{root}) &= \text{abc} \\ \text{current_expanded_word} &= \text{add} \\ \text{set_start_word} &= \{\text{a}, \text{b}, \text{c}\} \\ \text{set_current_expanded_word} &= \{\text{a}, \text{d}, \text{d}\} \\ g(n) &= \text{length of } (\text{set_current_expanded_word} - \text{set_start_word}) \end{aligned}$$

Fungsi $g(n)$ inilah yang akan menentukan simpul E yang akan diambil dari Simpul Hidup. Berikut adalah langkah keseluruhan dari algoritma:

1. Inisialisasi simpul hidup yang diisi dengan *start word*, simpul E kosong, dan *visited nodes* berisi *start word*.
2. Pop atau tarik elemen pertama dari simpul hidup dan ditetapkan sebagai nilai simpul E.
3. Lakukan ekspansi terhadap simpul E dan ditambahkan ke simpul hidup berdasarkan *constraints* yang ada.
4. Lakukan berulang hingga ditemukan simpul E yang menjadi solusi atau hingga simpul hidup kosong.

b. Greedy Best First Search

Algoritma ini mirip dengan BFS tetapi ekspansi hanya dilakukan pada salah satu dari elemen yang memiliki nilai *heuristic* terkecil. Nilai *heuristic* adalah suatu nilai acuan yang diambil hanya memandang pada nilai evaluasi berdasarkan elemen sekarang dan elemen tujuan. Pada konteks persoalan ini, nilai *heuristic* adalah nilai jarak dari kata yang ditinjau terhadap kata yang menjadi tujuan atau disebut $h(n)$. Maka, pada algoritma ini akan hanya ada satu struktur data yang digunakan yaitu, path (Array/List).

Pada kasus ini, fungsi heuristik $h(n)$ dapat dirumuskan sebagai berikut:

$$\begin{aligned} \text{end_word (root)} &= \text{abc} \\ \text{current_expanded_word} &= \text{add} \\ \text{set_start_word} &= \{\text{a, b, c}\} \\ \text{set_current_expanded_word} &= \{\text{a, d, d}\} \\ f(n) &= \text{length of (set_current_expanded_word - set_start_word)} \end{aligned}$$

Fungsi heuristik inilah yang nantinya akan digunakan untuk menentukan elemen yang akan menjadi kandidat dari path yang akan diambil. Berikut adalah langkah keseluruhan dari algoritma ini:

1. Inisialisasi *path* dengan berisi start word.
2. Ambil elemen terakhir dari *path* dan lakukan ekspansi terhadapnya.
3. Hasil dari ekspansi akan dicari nilai elemen yang memiliki nilai heuristik terkecil.
4. *Append path* dengan elemen tersebut.
5. Panggil ulang kembali algoritma secara rekursif, basis dari rekursinya adalah saat elemen terakhir dari *path* bernilai setara dengan end word.

Secara teoritis, algoritma ini memiliki kekurangan dimana hasil dari jalur yang ditemukan tidak selalu optimal. Hal ini karena *Greedy BFS* hanya memandang optimal local dan tidak memandang secara holistik dari persoalan tersebut sehingga memungkinkan adanya pemilihan jalur yang tidak optimum secara keseluruhan.

Algoritma ini juga memiliki kekurangan lain yaitu pada konteks persoalan ini akan banyak terjadi local optima. Local optima adalah sebuah kasus dimana pembangkitan simpul akan memicu *circular looping* dan algoritma ini tidak akan pernah menemukan jalurnya menuju tujuan. Berikut beberapa faktor mengapa hal ini bisa terjadi:

1. Pembangkitan kata pada kamus memungkinkan banyak elemen yang bernilai minimum, tetapi algoritma ini tidak akan melakukan ekspansi terhadap semuanya sehingga menutup kemungkinan akan menemukan jalur tujuan.
2. Algoritma ini tidak memiliki pembatasan terhadap pembangkitan seperti *visited nodes* sehingga memungkinkan membangkitkan elemen sebelumnya dan terjadi *circular looping*.

c. *Algoritma A**

Algoritma A* adalah perpaduan antara UCS dan *Greedy BFS* dengan menggunakan algoritma UCS dan menggunakan kedua nilai fungsi heuristik $h(n)$ dan fungsi *cost* berdasarkan root $g(n)$. Akan tetapi hal ini tidak bisa diaplikasikan untuk semua kasus karena memerlukan suatu syarat yang perlu dipenuhi yaitu, fungsi $h(n)$ harus merupakan estimasi dan bernilai lebih kecil daripada nilai *cost* aslinya menuju goal, apabila memenuhi maka heuristik yang digunakan *admissible* dan A* akan selalu menemukan jalur optimal. Berikut adalah perumusan $f(n)$ dari A*:

$$f(n) = h(n) + g(n)$$

Pada kasus persoalan ini, fungsi $h(n)$ bersifat *admissible* karena hanya merupakan estimasi jarak berdasarkan kemiripan karakter dengan end word, bukan jarak atau harga asli menuju *goal word* dan bernilai pasti akan bernilai lebih kecil sebab merupakan batas minimum untuk langkah menuju ke end word. Berikut adalah langkah-langkah algoritma ini secara keseluruhan:

1. Inisialisasi simpul hidup yang diisi dengan *start word*, simpul E kosong, dan *visited nodes* berisi *start word*.
2. Pop atau tarik elemen pertama dari simpul hidup dan ditetapkan sebagai nilai simpul E.
3. Lakukan ekspansi terhadap simpul E dan ditambahkan ke simpul hidup berdasarkan dan disortir *constraint* $f(n) = h(n) + g(n)$.
4. Lakukan berulang hingga ditemukan simpul E yang menjadi solusi atau hingga simpul hidup kosong.

Perpaduan dari elemen *Greedy BFS* dan elemen UCS membuat algoritma ini jauh lebih efisien dari UCS dan selalu memberikan hasil optimum.

II. Source Code Implementasi (Bahasa Pemrograman Java)

Pada implementasi ini, saya menggunakan kelas *BaseClass* sebagai kerangka utama dari kelas secara keseluruhan dan sebagai parent dari method-method dan atribut-atribut yang general bagi semua keperluan masing-masing algoritma. Berikut adalah gambar dari *BaseClass*:

```
package Utils;

import java.util.ArrayList;

public class BaseClass {
    public String end_word;
    public String start_word;
    public char[] alphabet = {
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
        'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
        'u', 'v', 'w', 'x', 'y', 'z'
    };

    public BaseClass(String start_word, String end_word) {
        this.start_word = start_word;
        this.end_word = end_word;
    }

    public int costCounter(String word) {
        int counter = 0;
        for (int i = 0; i < end_word.length(); i++) {
            if (end_word.toLowerCase().charAt(i) != word.toLowerCase().charAt(i)) {
                counter++;
            }
        }
        return counter;
    }

    public int depthCounter(String word) {
        int counter = 0;
        for (int i = 0; i < end_word.length(); i++) {
            if (start_word.toLowerCase().charAt(i) == word.toLowerCase().charAt(i)) {
                counter++;
            }
        }
        return counter;
    }

    public void printPath(ArrayList<String> path) {
        for (int i = 0; i < path.size() - 1; i++) {
            if (!path.get(i).equals(" ")) {
                System.out.print(path.get(i) + "->");
            }
        }
        System.out.println(path.get(path.size() - 1));
    }

    public String toPath(ArrayList<String> path) {
        String res = "";
        for (int i = 0; i < path.size() - 1; i++) {
            if (!path.get(i).equals(" ")) {
                res += path.get(i).toString() + "->";
            }
        }
        res += path.get(path.size() - 1).toString();
        return res;
    }
}
```

Penjelasan atribut dan method:

1. *CostCounter* : fungsi yang mengembalikan nilai heuristik dari sebuah elemen string
2. *DepthCounter* : fungsi yang mengembalikan harga jarak elemen string terhadap rootnya.
3. *PrintPath* : fungsi formatting untuk mencetak jalur hasil
4. *ToPath* : fungsi formatting untuk mengembalikan string jalur hasil
5. *Start_word* : kata mula dari *word ladder*.
6. *End_word* : kata akhir dari *word ladder*.
7. *Alphabet* : atribut untuk menyimpan alfabet sebagai kebutuhan ekspansi kata.

a. Source Code Algoritma *Uniform Cost Search*

```
import Utils.*;
import java.util.*;

public class UCS extends BaseClass {
    public PriorityQueue<Node> simpul_hidup = new PriorityQueue<Node>(new UCSComparator());
    public Set<String> visited = new HashSet<String>();
    public WordValidator w = new WordValidator(this.start_word);

    public UCS(String start_word, String end_word) {
        super(start_word.toLowerCase(), end_word.toLowerCase());
        ArrayList<String> start_path = new ArrayList<String>();
        start_path.add(start_word);
        Node temp_start = new Node(start_word, start_path);
        this.simpul_hidup.add(temp_start);
    }

    public ArrayList<String> expandWord(String word) {
        ArrayList<String> exp_word = new ArrayList<String>();
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < 26; j++) {
                StringBuilder new_word = new StringBuilder(word);
                new_word.setCharAt(i, this.alphabet[j]);
                if (!new_word.toString().toLowerCase().equals(word)
                    && w.isWordValid(new_word.toString().toLowerCase())
                    && !this.visited.contains(new_word.toString())) {
                    exp_word.add(new_word.toString().toLowerCase());
                    this.visited.add(new_word.toString());
                }
            }
        }
        return exp_word;
    }

    public String toPath(ArrayList<String> path) {
        String res = "";
        for (int i = 0; i < path.size() - 1; i++) {
            if (!path.get(i).equals(anObject: " ")) {
                res += path.get(i).toString() + "->";
            }
        }
        res += path.get(path.size() - 1).toString();
        return res;
    }
}
```

```

public Node UCSPrioQueue() {
    while (!this.simpul_hidup.isEmpty()) {
        Node current_node = this.simpul_hidup.poll();
        String exp_word = current_node.word;
        ArrayList<String> available_words = this.expandWord(exp_word);
        for (int i = 0; i < available_words.size(); i++) {
            ArrayList<String> temp_path = new ArrayList<String>(current_node.current_path);
            temp_path.add(available_words.get(i));
            Node temp = new Node(available_words.get(i), temp_path);
            this.simpul_hidup.add(temp);
            if (available_words.get(i).equals(this.end_word.toLowerCase())) {
                return temp;
            }
        }
    }
    Node no_res = null;
    return no_res; // dummy return
}

public String[] mainUCS() {
    UCS ucs = new UCS(this.start_word, this.end_word);
    Node res = new Node(word:null, path:null);
    String[] result = { "No Path", "Took 0ms to execute", "Visited 0 node(s)" };
    final long startTime = System.currentTimeMillis();
    res = (ucs.UCSPrioQueue());
    final long endTime = System.currentTimeMillis();
    if (res != null) {
        ucs.printPath(res.current_path);
        result[0] = ucs.toPath(res.current_path);
    }
    result[1] = "Took " + (endTime - startTime) + "ms to execute";
    result[2] = "Visited " + ucs.visited.size() + " node(s)";
    System.out.println("Took " + (endTime - startTime) + "ms to execute");
    System.out.println("Visited " + ucs.visited.size() + " node(s)");
    return result;
}
}

```

Penjelasan atribut dan method:

- Constructor UCS() : untuk menginisialisasi start_word, end_word, dan simpul_hidup
- *ExpandWord* : method yang mengembalikan list of array berisi hasil ekspansi
- UCSPrioQueue : method yang merupakan algoritma utama dari UCS
- mainUCS : method yang akan dipanggil di main program sebagai formatting dari hasil.
- simpul_hidup : struktur data Priority Queue yang digunakan sebagai simpul hidup pada algoritma UCS
- *visited_nodes* : struktur data HashSet digunakan untuk menyimpan kata-kata yang sudah di ekspansi.
- WordValidator : instansiasi objek kelas WordValidator sebagai referensi kamus.

b. Source Code Algoritma Greedy BFS

```
1 import java.util.*;
2
3 import Utils.*;
4
5 public class GreedyBFS extends BaseClass {
6     public ArrayList<String> gbfs_buffer = new ArrayList<String>();
7     public WordValidator w = new WordValidator(this.start_word);
8     public Boolean local_optima = false;
9     public Boolean no_path = false;
10
11    public GreedyBFS(String start_word, String end_word) {
12        super(start_word, end_word);
13        this.gbfs_buffer.add(start_word);
14    }
15
16    public Boolean containsDupe() {
17        Set<String> dupe_checker = new HashSet<>();
18        for (int i = 0; i < this.gbfs_buffer.size(); i++) {
19            Boolean dupe = dupe_checker.add(this.gbfs_buffer.get(i));
20            if (!dupe) {
21                return true;
22            }
23        }
24        return false;
25    }
26
27    public String expandWord(String word) {
28        ArrayList<String> exp_word = new ArrayList<String>();
29        for (int i = 0; i < word.length(); i++) {
30            for (int j = 0; j < 26; j++) {
31                StringBuilder new_word = new StringBuilder(word);
32                if (word.toLowerCase().charAt(i) != this.alphabet[j]) {
33                    new_word.setCharAt(i, this.alphabet[j]);
34                }
35                if (!new_word.toString().toLowerCase().equals(word)
36                    && w.isWordValid(new_word.toString().toLowerCase())) {
37                    if (new_word.toString().equalsIgnoreCase(this.end_word)) {
38                        return new_word.toString();
39                    }
40                    exp_word.add(new_word.toString().toLowerCase());
41                }
42            }
43        }
44        Collections.sort(exp_word, new GreedyComparator(this.start_word, this.end_word));
45        return exp_word.get(index:0);
46    }
47}
```

```

public void GreedyBFSRecursion() {
    String current_node = this.gbfs_buffer.get(this.gbfs_buffer.size() - 1);
    String next_node = expandWord(current_node);
    System.out.println(this.gbfs_buffer);
    if (this.containsDupe()) {
        this.local_optima = true;
        return;
    }
    if (next_node == null) {
        this.no_path = true;
        return;
    }
    System.out.println(current_node);
    this.gbfs_buffer.add(next_node);
    if (next_node.equalsIgnoreCase(this.end_word)) {
        return;
    }
    GreedyBFSRecursion();
}

public String toPath(ArrayList<String> path) {
    String res = "";
    for (int i = 0; i < path.size() - 1; i++) {
        if (!path.get(i).equals(" ")) {
            res += path.get(i).toString() + "->";
        }
    }
    res += path.get(path.size() - 1).toString();
    return res;
}

public String[] mainGBFS() {
    GreedyBFS gr = new GreedyBFS(this.start_word, this.end_word);
    String[] result = { "No Path", "Took 0ms to execute", "Visited 0 node(s)" };
    final long startTime = System.currentTimeMillis();
    gr.GreedyBFSRecursion();
    final long endTime = System.currentTimeMillis();
    System.out.println(gr.gbfs_buffer.size());
    if (!gr.local_optima && !gr.no_path) {
        System.out.println(gr.gbfs_buffer.size());
        result[0] = gr.toPath(gr.gbfs_buffer);
        // gr.printPath(gr.gbfs_buffer);
    }
    result[1] = "Took " + (endTime - startTime) + "ms to execute";
    result[2] = "Visited " + gr.gbfs_buffer.size() + " node(s)";
    if (gr.local_optima) {
        result[0] = "Local Optima";
    }
    System.out.println(result[0]);
    System.out.println(result[1]);
    System.out.println(result[2]);
    return result;
}
}

```

Penjelasan atribut dan method:

- Constructor GreedyBFS() : untuk menginisialisasi start_word, end_word, dan gbfs_buffer yang merupakan path hasil.
- *ExpandWord* : method yang mengembalikan string minimum secara heuristik dari ekspansi
- GreedyBFSRecursion : method yang merupakan algoritma utama dari Greedy BFS
- mainGBFS : method yang akan dipanggil di main program sebagai formatting dari hasil.
- gbfs_buffer : struktur data ArrayList yang digunakan untuk menyimpan hasil akhir path algoritma Greedy BFS
- WordValidator : instansiasi objek kelas WordValidator sebagai referensi kamus.

c. **Source Code Algoritma A***

```

import java.util.*;
import Utils.*;

public class Astar extends BaseClass {
    public PriorityQueue<Node> simpul_hidup = new PriorityQueue<Node>(
        new AStarComparator(this.start_word, this.end_word));
    public Set<String> visited = new HashSet<String>();
    public WordValidator w = new WordValidator(this.start_word);

    public Astar(String start_word, String end_word) {
        super(start_word.toLowerCase(), end_word.toLowerCase());
        ArrayList<String> start_path = new ArrayList<String>();
        start_path.add(start_word);
        Node temp_start = new Node(start_word, start_path);
        this.simpul_hidup.add(temp_start);
    }

    public ArrayList<String> expandWord(String word) {
        ArrayList<String> exp_word = new ArrayList<String>();
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < 26; j++) {
                StringBuilder new_word = new StringBuilder(word);
                new_word.setCharAt(i, this.alphabet[j]);
                if (!new_word.toString().toLowerCase().equals(word)
                    && w.isWordValid(new_word.toString().toLowerCase())
                    && !this.visited.contains(new_word.toString())) {
                    exp_word.add(new_word.toString().toLowerCase());
                    this.visited.add(new_word.toString());
                }
            }
        }
        return exp_word;
    }

    public String toPath(ArrayList<String> path) {
        String res = "";
        for (int i = 0; i < path.size() - 1; i++) {
            if (!path.get(i).equals(" ")) {
                res += path.get(i).toString() + "->";
            }
        }
        res += path.get(path.size() - 1).toString();
        return res;
    }

    public void prinSimpulHidup() {
        PriorityQueue<Node> temp = new PriorityQueue<>(this.simpul_hidup);
        for (int i = 0; i < this.simpul_hidup.size(); i++) {
            Node temp_node = temp.poll();
            System.out.print(
                "{ " + temp_node.current_path + " -> " + (temp_node.current_path.size()
                    + this.costCounter(temp_node.word)) + " b(n) :" + temp_node.current_path.size()
                    + " f(n) :" + this.costCounter(temp_node.word) + " }");
        }
    }
}

```

```

public String toPath(ArrayList<String> path) {
    String res = "";
    for (int i = 0; i < path.size() - 1; i++) {
        if (!path.get(i).equals(" ")) {
            res += path.get(i).toString() + "->";
        }
    }
    res += path.get(path.size() - 1).toString();
    return res;
}

public void printSimpulHidup() {
    PriorityQueue<Node> temp = new PriorityQueue<>(this.simpul_hidup);
    for (int i = 0; i < this.simpul_hidup.size(); i++) {
        Node temp_node = temp.poll();
        System.out.print(
            "{" + temp_node.current_path + " -> " + (temp_node.current_path.size()
                + this.costCounter(temp_node.word)) + " b(n) :" + temp_node.current_path.size()
                + " f(n) :" + this.costCounter(temp_node.word) + "}"
        );
    }
}

public Node AstarPrioQueue() {
    while (!this.simpul_hidup.isEmpty()) {
        Node current_node = this.simpul_hidup.poll();
        String exp_word = current_node.word;
        ArrayList<String> available_words = this.expandWord(exp_word);
        for (int i = 0; i < available_words.size(); i++) {
            ArrayList<String> temp_path = new ArrayList<String>(current_node.current_path);
            temp_path.add(available_words.get(i));
            Node temp = new Node(available_words.get(i), temp_path);
            this.simpul_hidup.add(temp);
            if (available_words.get(i).equals(this.end_word.toLowerCase())) {
                return temp;
            }
        }
    }
    Node no_res = new Node(word:null, path:null);
    return no_res; // dummy return
}

public String[] mainAstar() {
    Astar as = new Astar(this.start_word, this.end_word);
    Node res = new Node(word:null, path:null);
    String[] result = { "No Path", "Took 0ms to execute", "Visited 0 node(s)" };
    final long startTime = System.currentTimeMillis();
    res = (as.AstarPrioQueue());
    final long endTime = System.currentTimeMillis();
    if (res != null) {
        as.printPath(res.current_path);
        result[0] = as.toPath(res.current_path);
    }
    result[1] = "Took " + (endTime - startTime) + "ms to execute";
    result[2] = "Visited " + as.visited.size() + " node(s)";
    System.out.println("Took " + (endTime - startTime) + "ms to execute");
    System.out.println("Visited " + as.visited.size() + " node(s)");
    return result;
}
}

```

Penjelasan atribut dan method:

- Constructor Astar() : untuk menginisialisasi start_word, end_word, dan simpul_hidup
- *ExpandWord* : method yang mengembalikan list of array berisi hasil ekspansi
- UCSPriorityQueue : method yang merupakan algoritma utama dari A*
- mainUCS : method yang akan dipanggil di main program sebagai formatting dari hasil.
- simpul_hidup : struktur data Priority Queue yang digunakan sebagai simpul hidup pada algoritma A*
- visited_nodes : struktur data HashSet digunakan untuk menyimpang kata-kata yang sudah di ekspansi.
- WordValidator : instansiasi objek kelas WordValidator sebagai referensi kamus.

d. Comparator PrioQueue

```
package Utils;

import java.util.Comparator;

public class AStarComparator extends BaseClass implements Comparator<Node> {
    public AStarComparator(String start_word, String end_word) {
        super(start_word, end_word);
    }

    public int compare(Node e11, Node e12) {
        int value1 = e11.current_path.size() + this.costCounter(e11.word);
        int value2 = e12.current_path.size() + this.costCounter(e12.word);
        if (value1 > value2) {
            return 1;
        } else if (value1 < value2) {
            return -1;
        }
        return 0;
    }
}
```

```
package Utils;
import java.util.*;

public class UCSComparator implements Comparator<Node> {
    public int compare(Node e11, Node e12) {
        if (e11.current_path.size() > e12.current_path.size()) {
            return 1;
        } else if (e11.current_path.size() < e12.current_path.size()) {
            return -1;
        }
        return 0;
    }
}
```

```

package Utils;

import java.util.Comparator;

public class AStarComparator extends BaseClass implements Comparator<Node> {
    public AStarComparator(String start_word, String end_word) {
        super(start_word, end_word);
    }

    public int compare(Node el1, Node el2) {
        int value1 = el1.current_path.size() + this.costCounter(el1.word);
        int value2 = el2.current_path.size() + this.costCounter(el2.word);
        if (value1 > value2) {
            return 1;
        } else if (value1 < value2) {
            return -1;
        }
        return 0;
    }
}

```

e. WordValidator

```

package Utils;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class WordValidator {
    public String dictionary_filename = "oracle_dictionary.txt";
    public Set<String> dictionary_map = new HashSet<String>();

    public WordValidator(String word) {
        File file = new File(
            "C:\\\\Users\\\\ASUS\\\\Documents\\\\Sem 4\\\\Stima\\\\Tucil3_13522076\\\\src\\\\Utils\\\\" + this.dictionary_filename);
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(file));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        String st;
        // System.out.println("Initializing..");
        try {
            while ((st = br.readLine()) != null) {
                if (st.length() == word.length()) {
                    this.dictionary_map.add(st);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        // System.out.println("Done");
    }

    public Boolean isWordValid(String word) {
        return this.dictionary_map.contains(word.toLowerCase());
    }
}

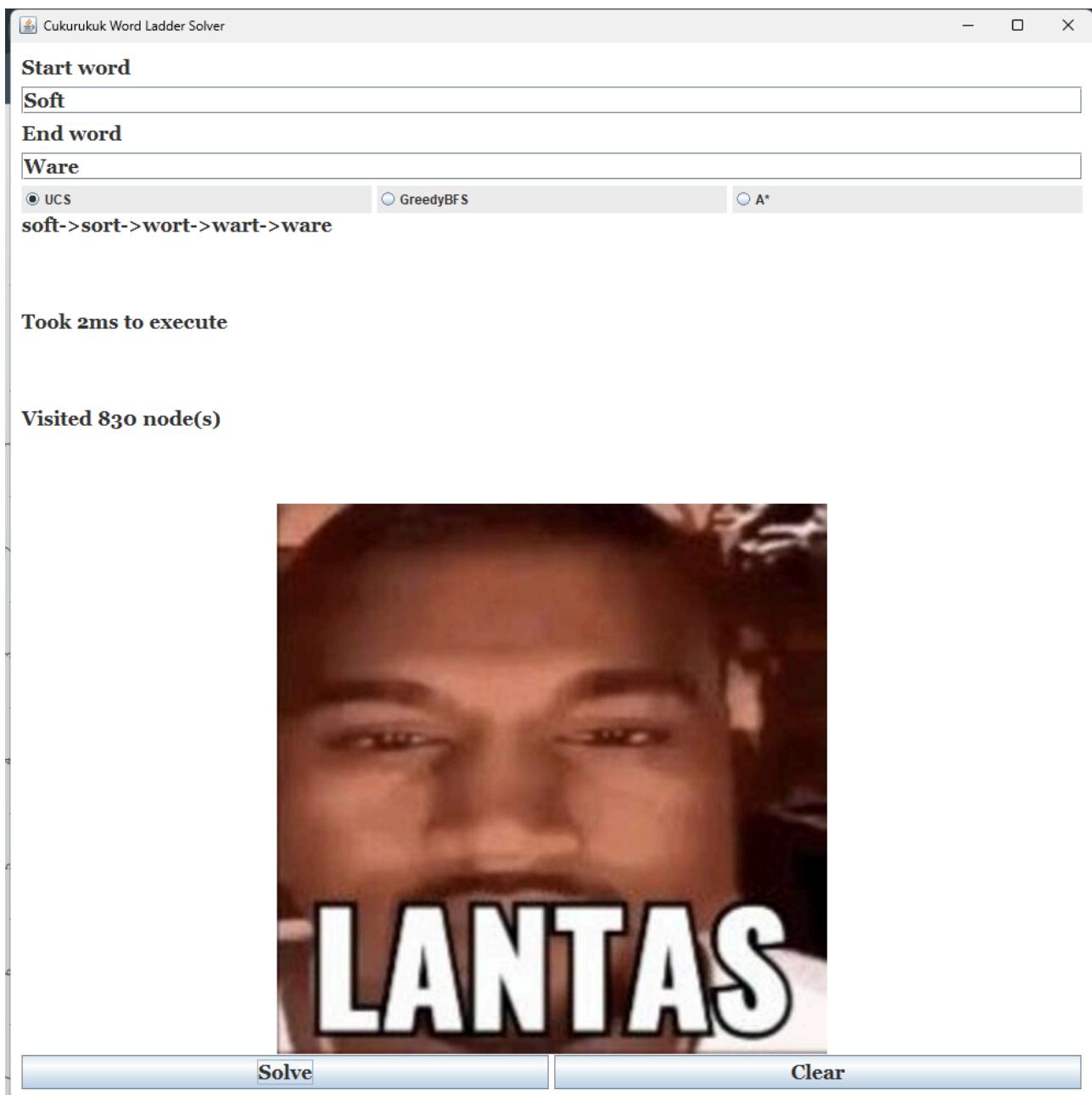
```

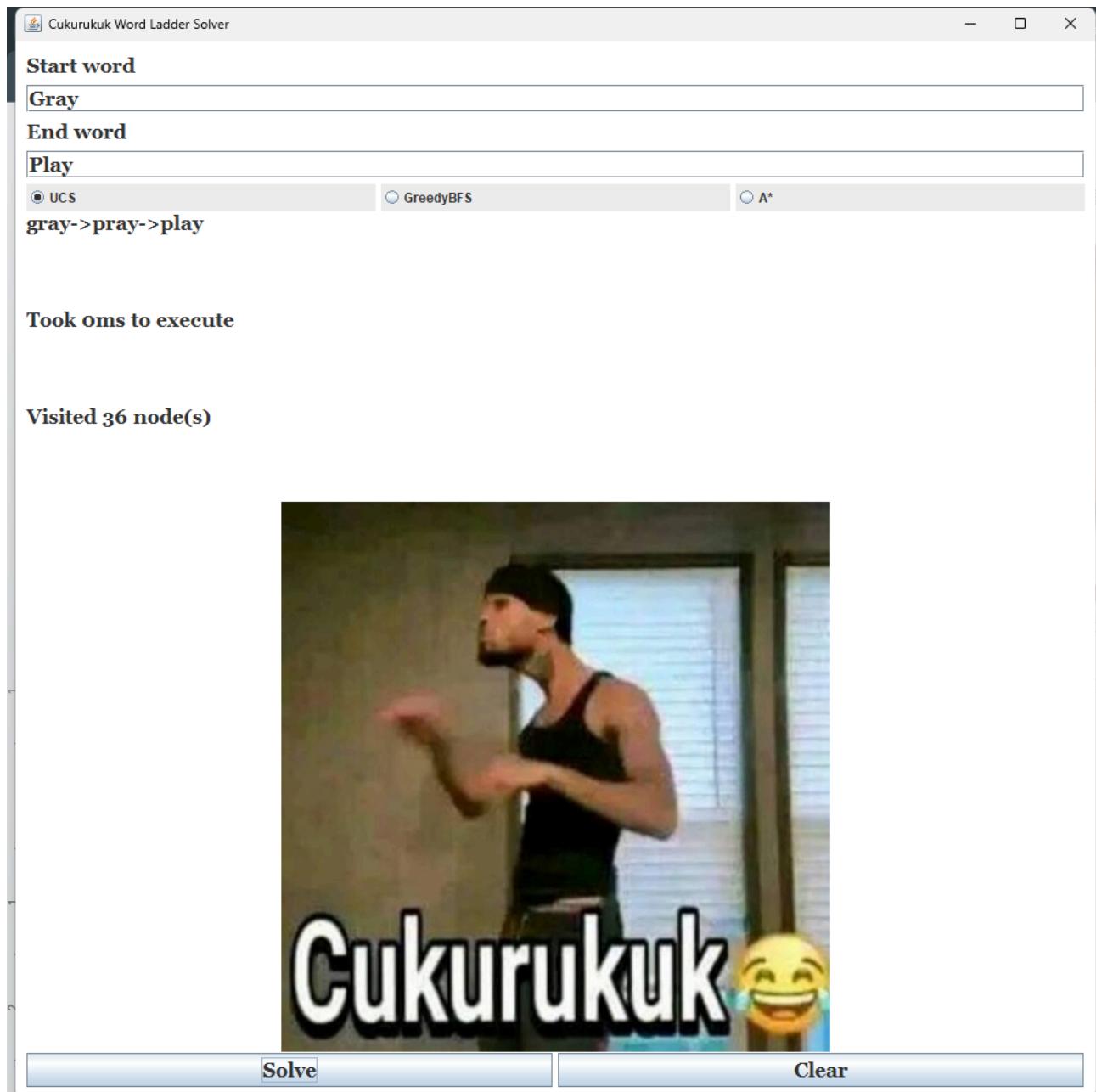
III. Test Case

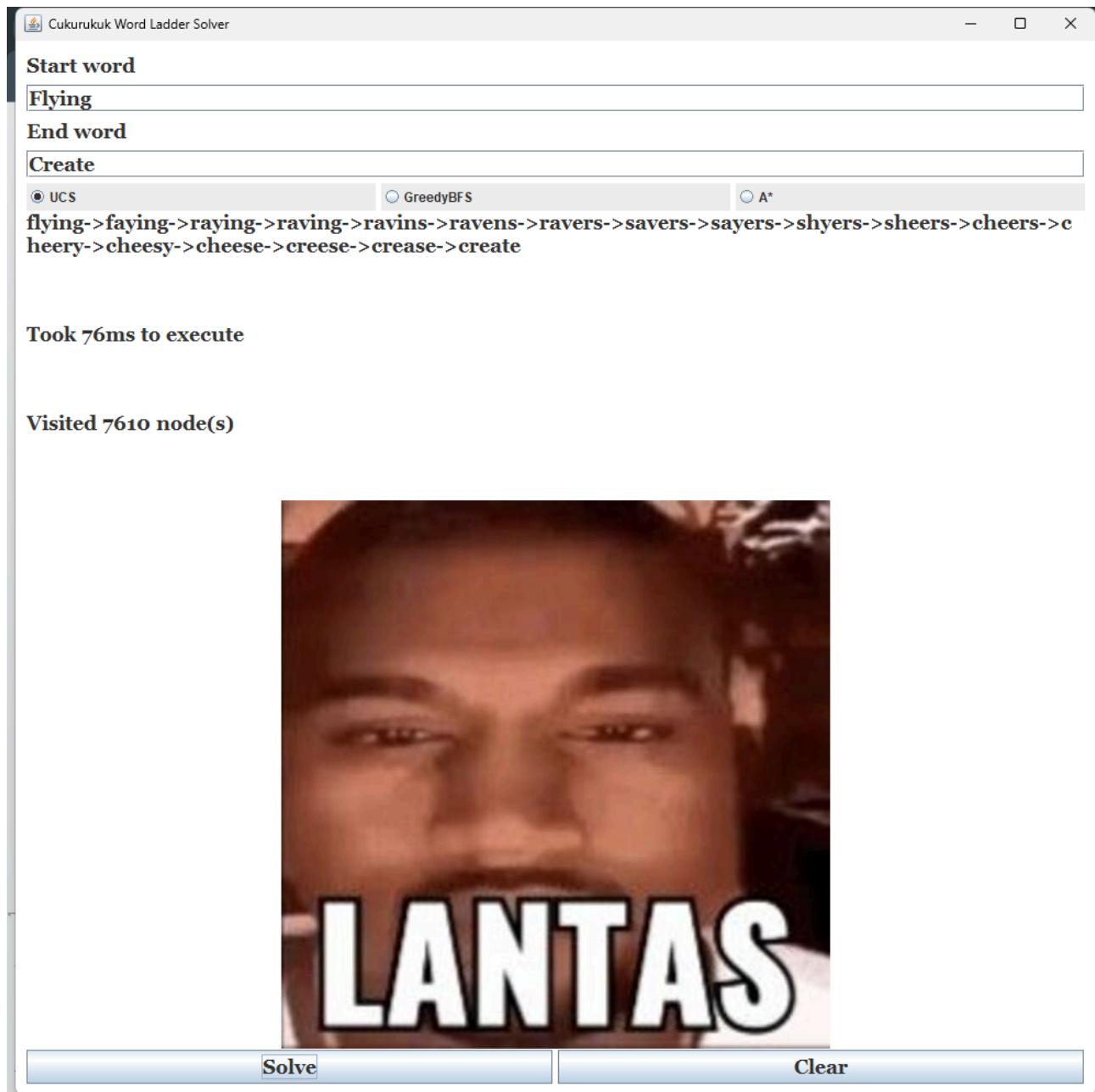
Test case umum :

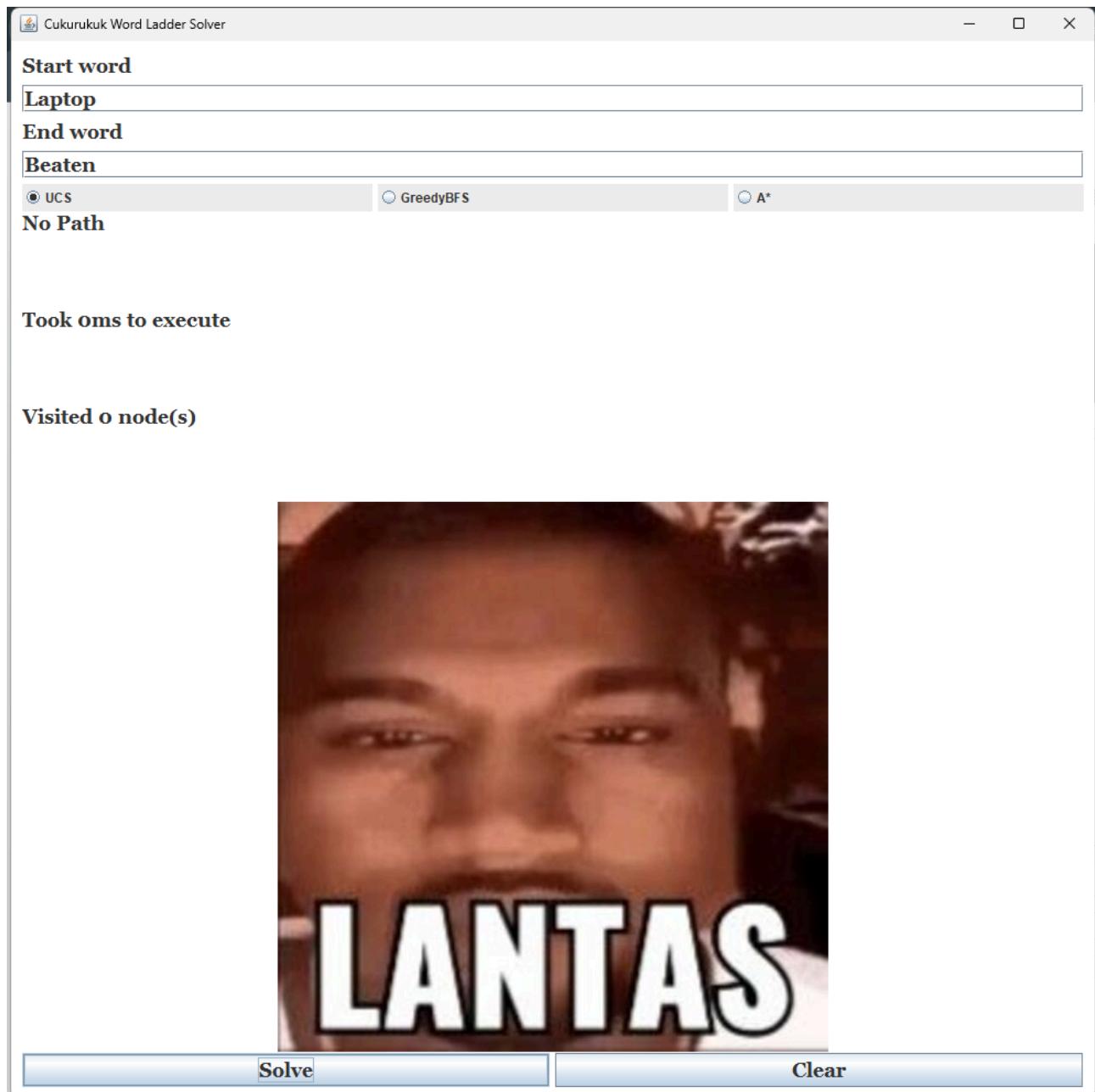
1. Soft -Ware
2. Gray - Play
3. Flying - Create
4. Laptop - Beaten
5. Tame - Mild
6. Dirty - Clean

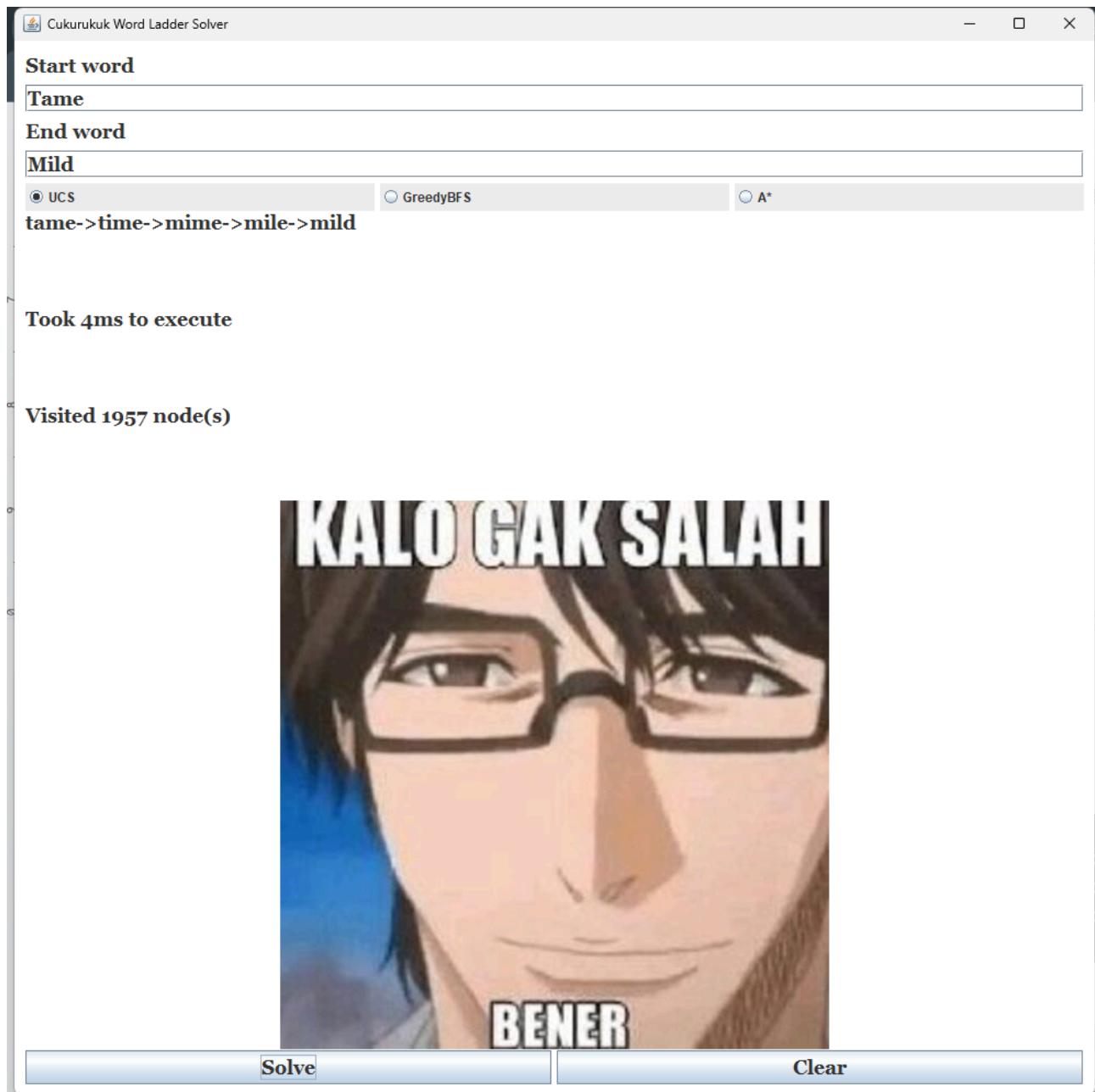
a. UCS













b. *Greedy BFS*

Cukurukuk Word Ladder Solver

Start word
soft

End word
ware

UCS GreedyBFS A*

soft->sort->wort->wart->ware

Took 4ms to execute

Visited 5 node(s)



Solve Clear

Cukurukuk Word Ladder Solver

Start word
Gray

End word
Play

UCS GreedyBFS A*

Gray->pray->play

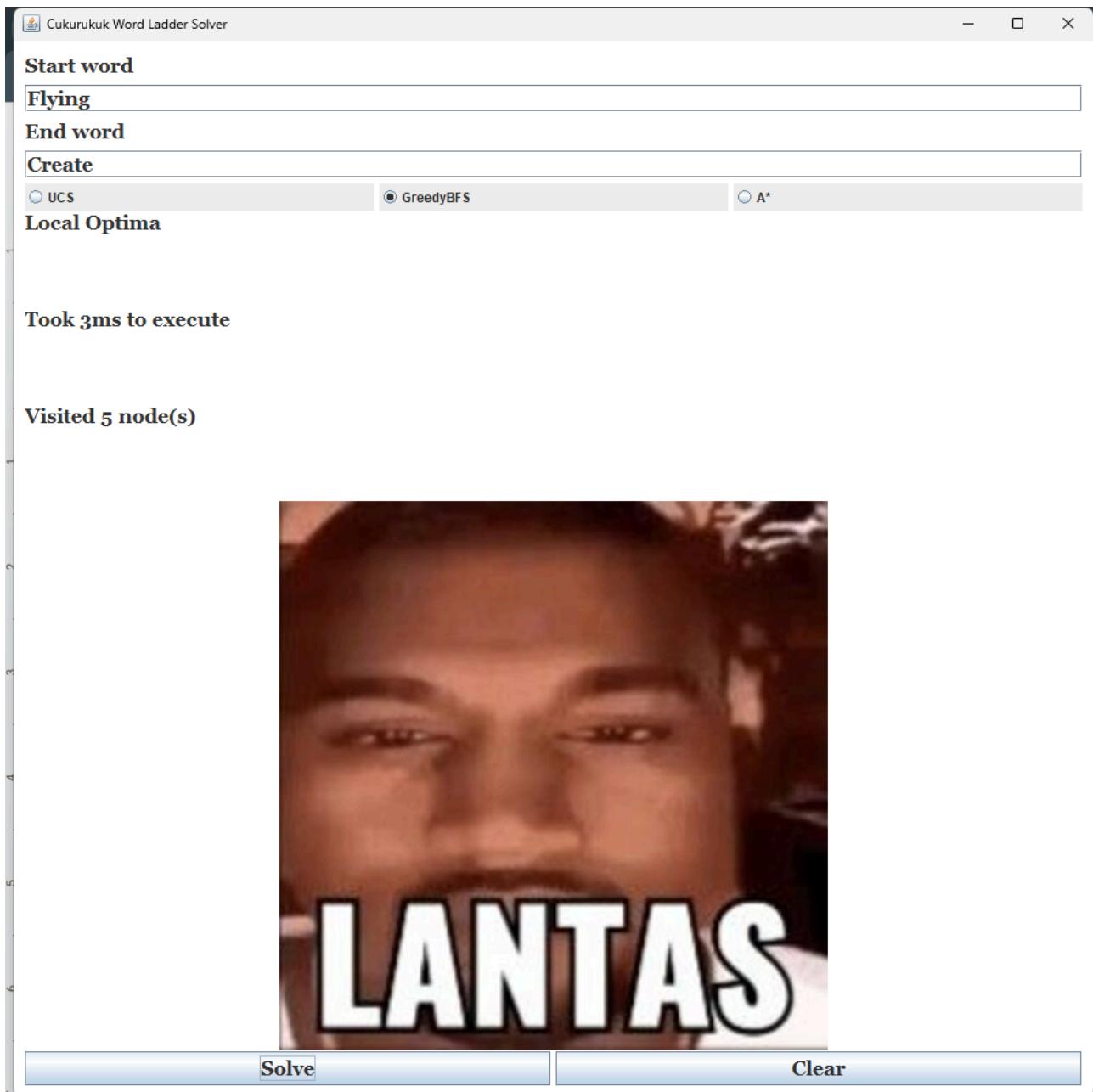
Took 2ms to execute

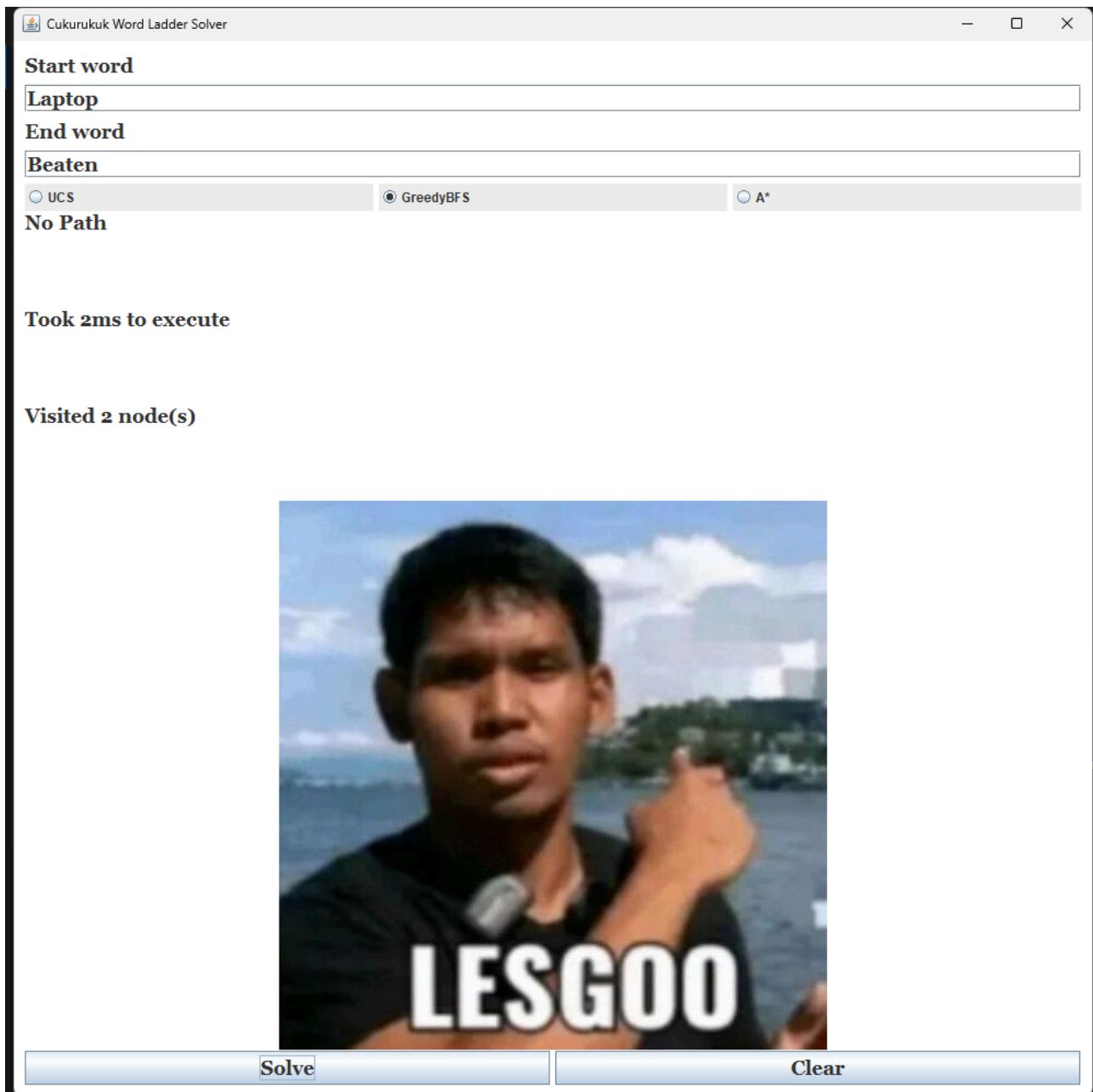
Visited 3 node(s)

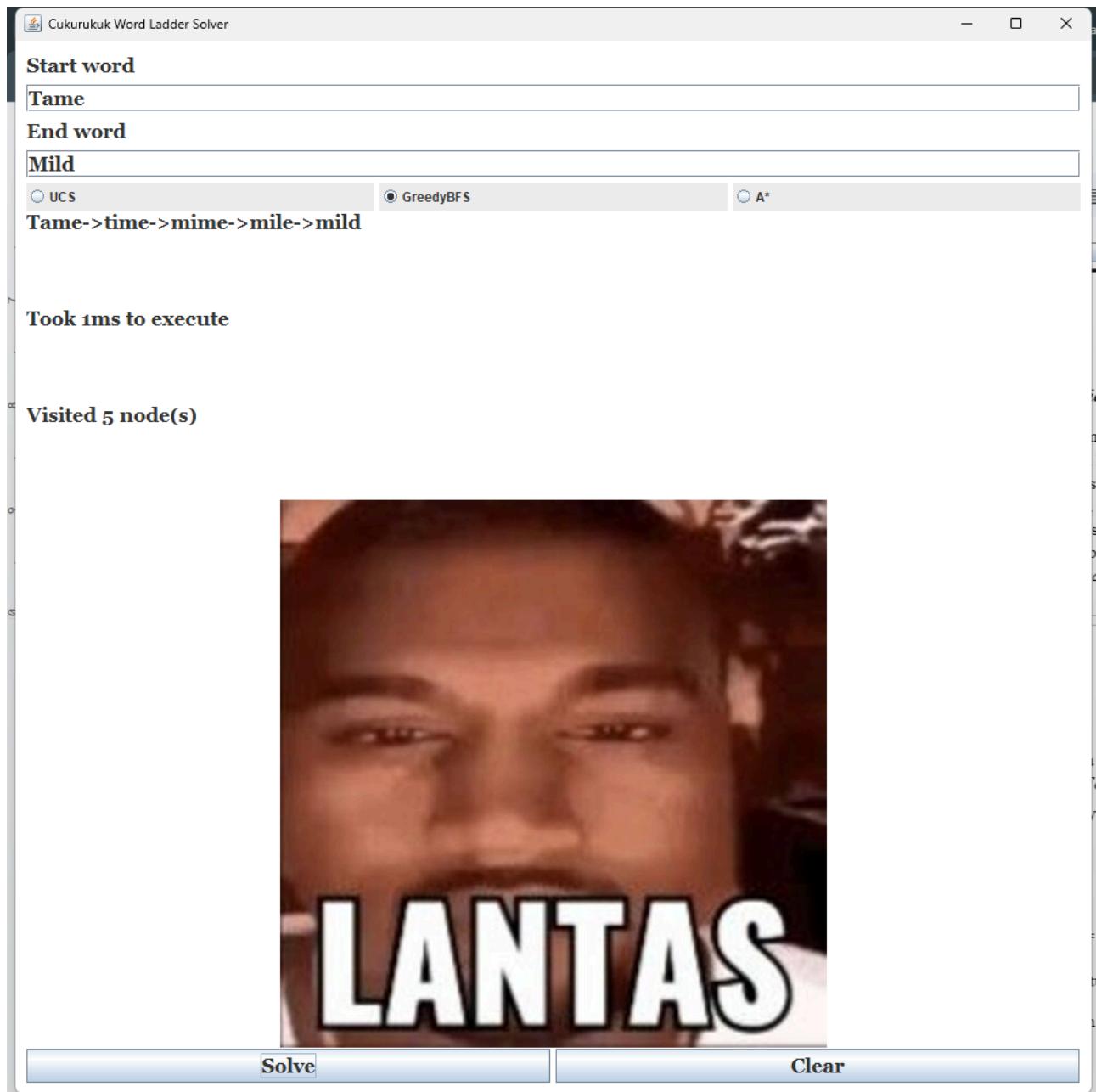
KALO GAK SALAH

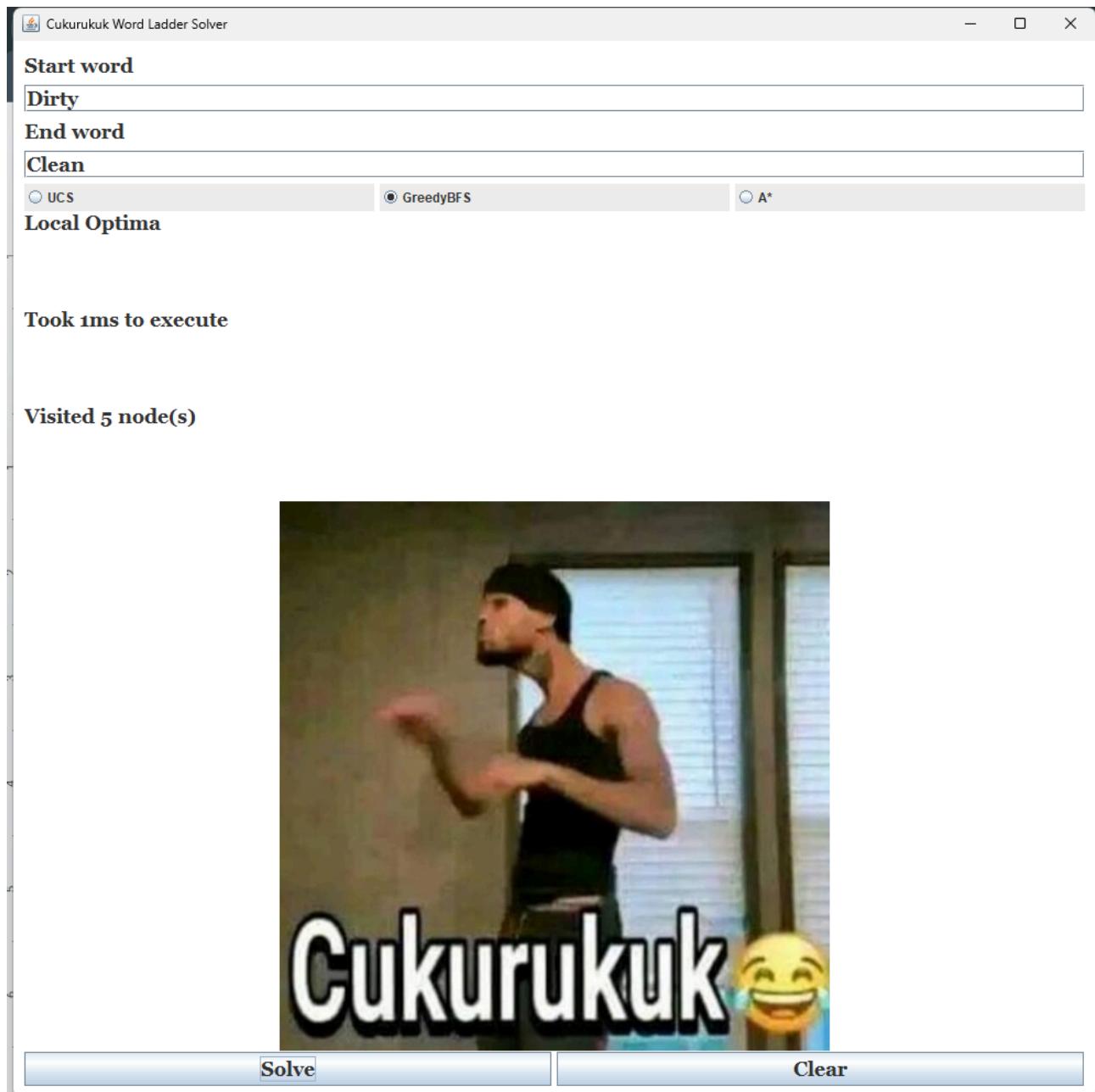
BENER

Solve Clear









c. A^*

Cukurukuk Word Ladder Solver

Start word
Soft

End word
Ware

UCS GreedyBFS A*

soft->sort->wort->wart->ware

Took 0ms to execute

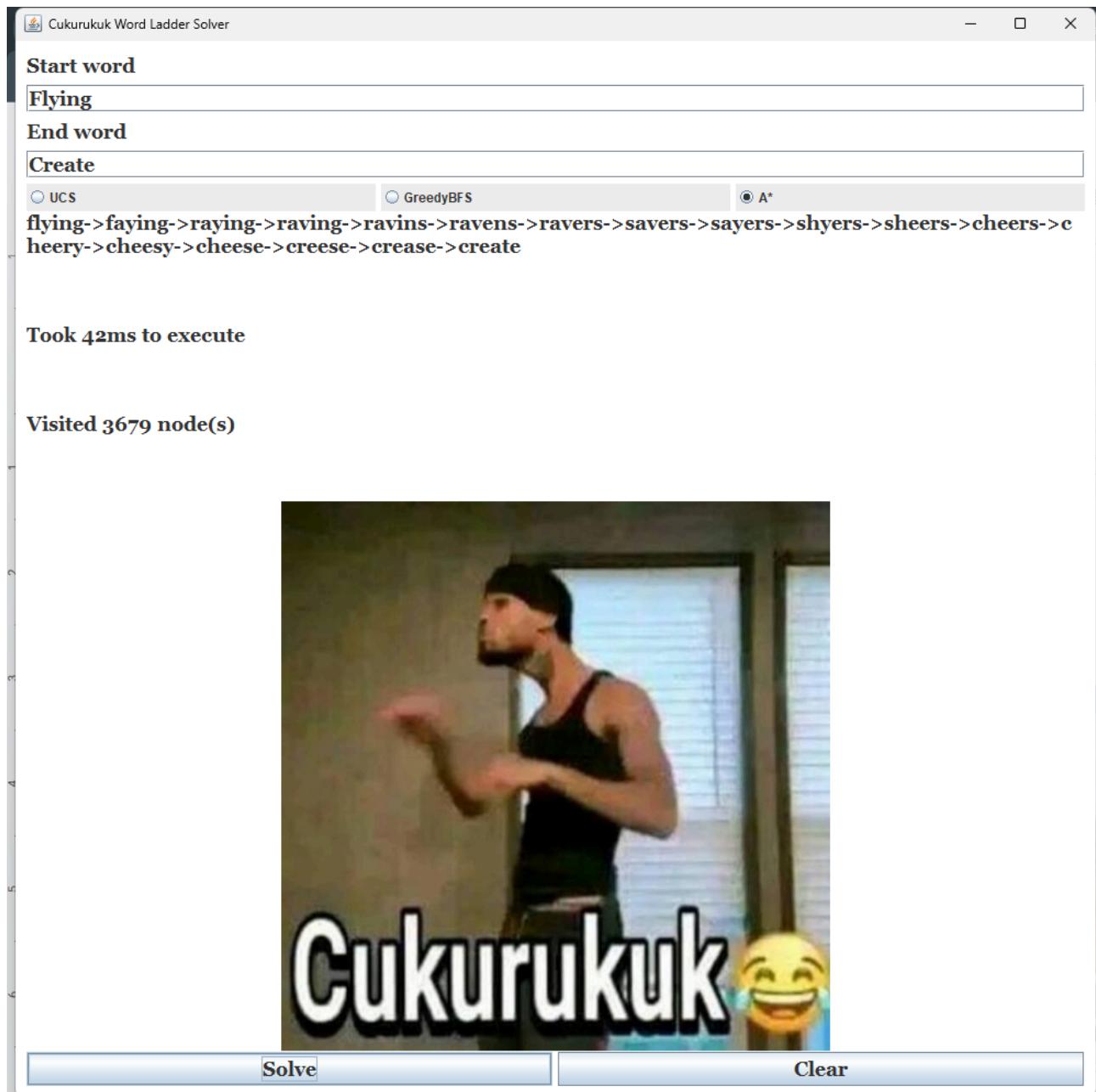
Visited 68 node(s)

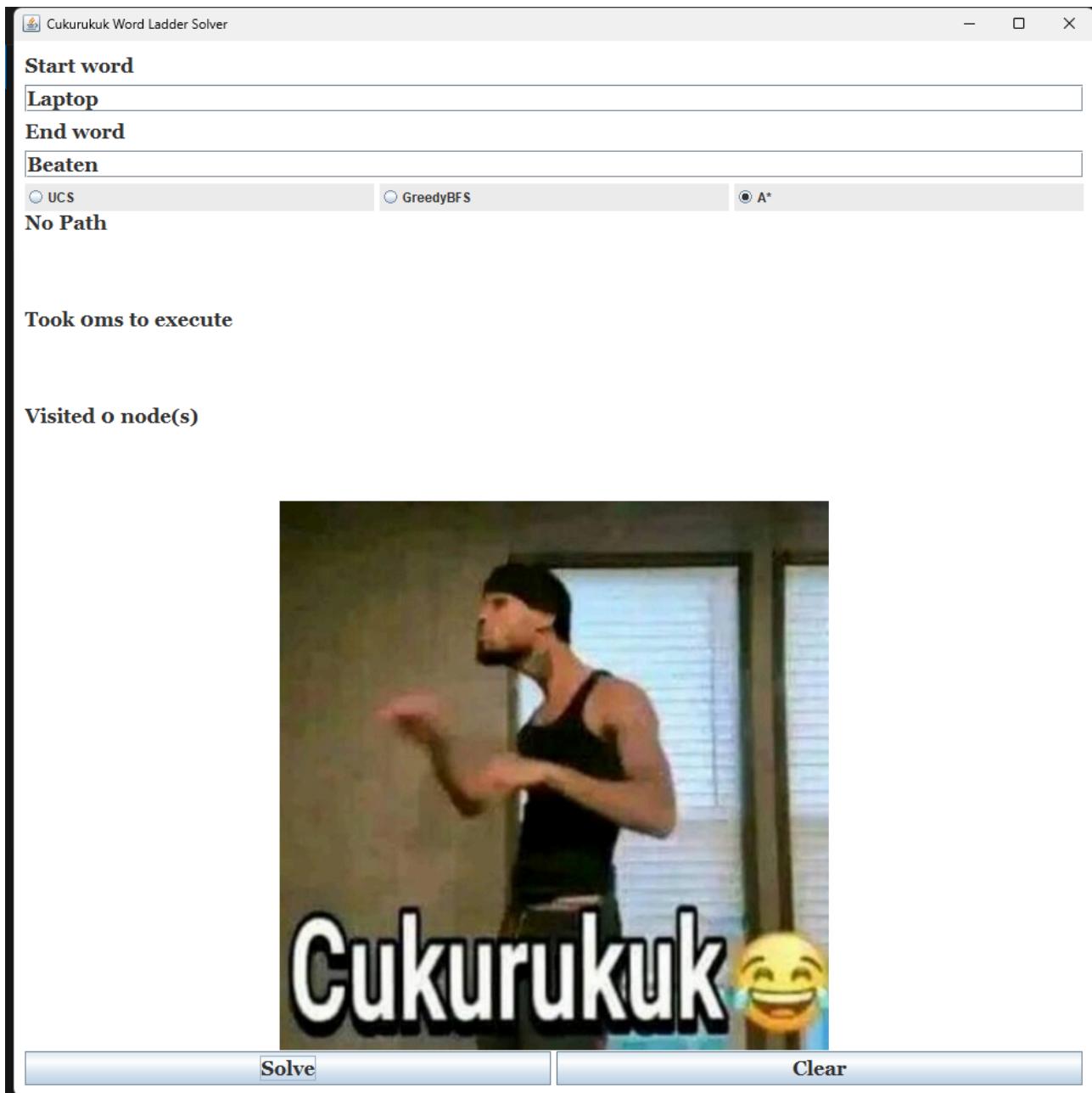


Cukurukuk 😊

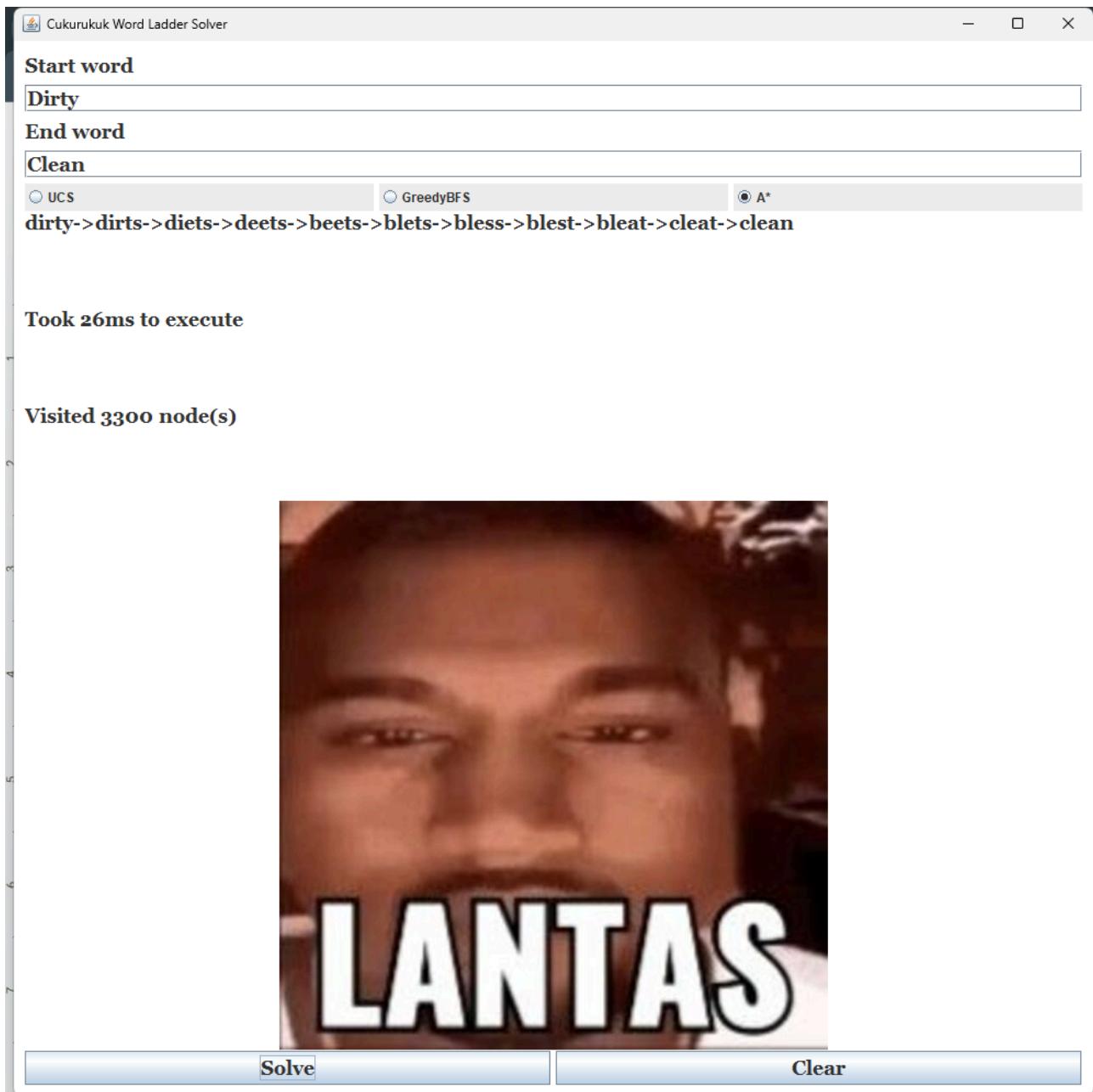
Solve **Clear**











IV. Perbandingan Ketiga Algoritma

Solusi memperlihatkan bahwa A* selalu optimal dan memiliki usage memory dan waktu yang lebih efisien dibandingkan dengan UCS. Dapat dilihat bahwa dari keseluruhan test case A* menghasilkan output yang sama seperti UCS (optimum) dengan waktu dan usage memory yang secara signifikan jauh lebih kecil. Greedy BFS memiliki solusi yang waktu eksekusinya sangat cepat dan memory usage yang lebih sedikit, namun di sisi lain hasil yang diberikan tidak selalu optimal seperti terjadinya local optima. Berikut salah satu perbandingan dari ketiga algoritma di test case tame - mild:

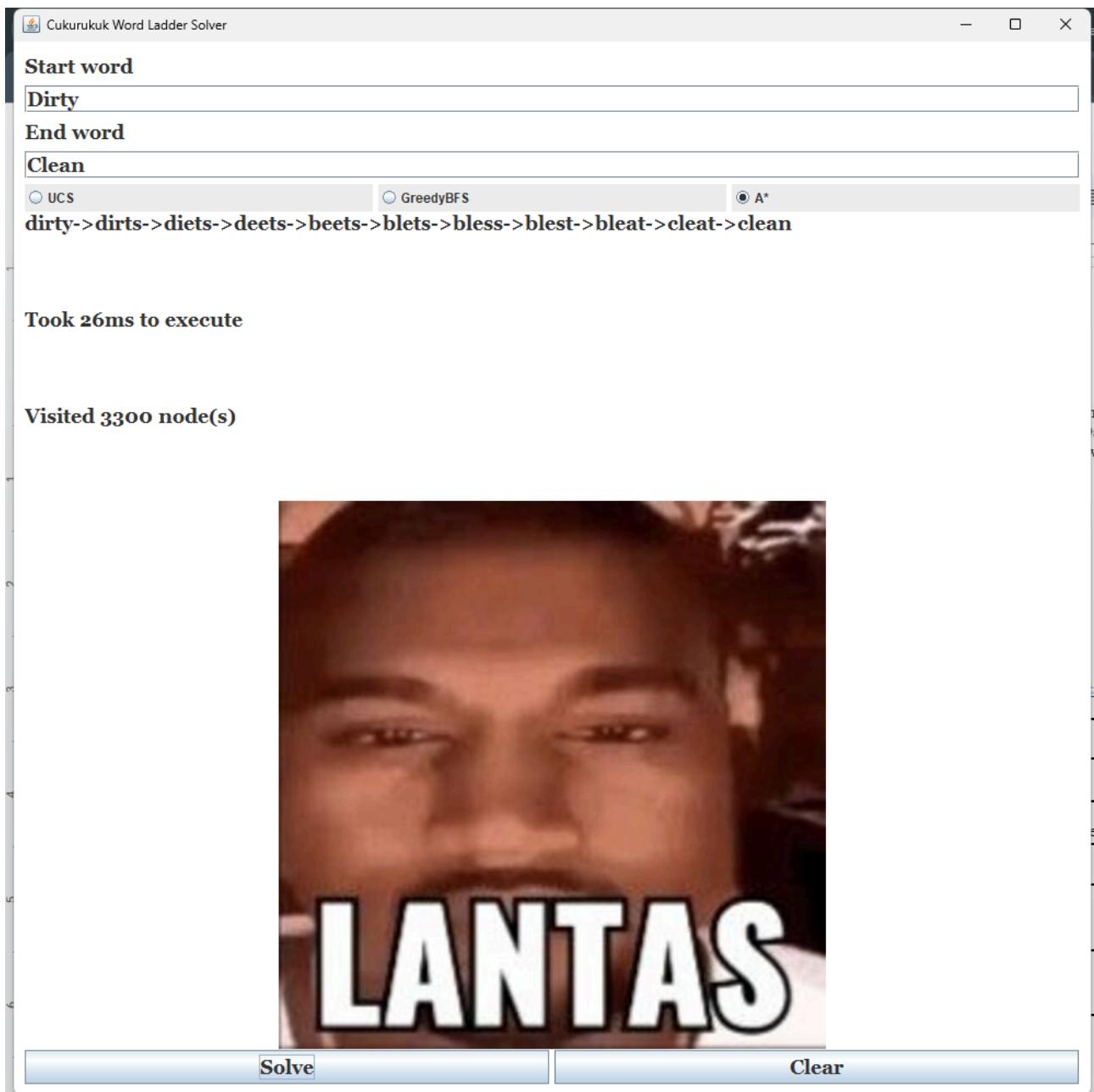
- A* = Solution (optimum), time exec : 1ms, nodes visited (memory usage) : 86 nodes.
- UCS = Solution (optimum), time exec : 4ms, nodes visited (memory usage) : 1976 nodes.
- Greedy BFS = Solution (optimum), time exec : 1ms, nodes visited (memory usage) :

8 nodes.

Dapat dilihat bahwa pada kasus ini greedy BFS memiliki harga pencarian paling murah dengan time exec 1 ms dan nodes visited hanya 8 nodes. Dapat dilihat juga perbedaan jauh dari usage memory antara UCS dan A* yang menandakan bahwa A* jauh lebih efisien dari UCS.

V. Implementasi Bonus

A. GUI (Java Swing)



Gambar di atas merupakan tampilan dari GUI yang saya implementasikan, komponen-komponen terbagi atas tiga bagian :

1. formPanel : berisi form untuk input start_word dan end_word untuk program, serta radio button untuk memilih algoritma
2. resultPanel : berisi hasil pesan string dan icon gambar yang ditrigger secara random dengan pengaksesan random index pada array of image path menggunakan metode random.nextInt(array_of_image.length).
3. buttonPanel : berisi tombol solve untuk menjalankan program dan menagkap hasil dan tombol clear untuk menghilangkan semua input dan output yang ada di layar.

Berikut adalah kode GUInya (maaf berantakan):

```

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import java.util.*;

public class GUI extends JFrame {
    final private Font mainFont = new Font("Georgia", Font.BOLD, 18);
    private String selected;
    JTextField tfStartWord, tfEndWord;
    private String[] images1 = { "Cukurukuk.jpg", "lesgo.jpeg",
    "Bener.jpeg", "Lantas.jpeg" };
    private String[] images2 = { "Lalu.jpeg", "Depresi.jpeg",
    "Hoaks.jpeg", "Awas.jpeg", "Galogis.jpeg" };
    private Random random = new Random();
    private int random_idx1 = random.nextInt(images1.length);
    private int random_idx2 = random.nextInt(images2.length);

    public void intialize() {
        JLabel lbStartWord = new JLabel("Start word");
        lbStartWord.setFont(mainFont);

        tfStartWord = new JTextField();
        tfStartWord.setFont(mainFont);

        tfEndWord = new JTextField();
    }
}

```

```
tfEndWord.setFont(mainFont);

JLabel lbEndWord = new JLabel("End word");
lbEndWord.setFont(mainFont);

JTextArea tfPath = new JTextArea();
tfPath.setFont(mainFont);
tfPath.setLineWrap(true);

JTextArea tfTime = new JTextArea();
tfTime.setFont(mainFont);
tfTime.setLineWrap(true);

JTextArea tfNode = new JTextArea();
tfNode.setFont(mainFont);
tfNode.setLineWrap(true);

ImageIcon img1 = new ImageIcon(
    "C:\\\\Users\\\\ASUS\\\\Documents\\\\Sem
4\\\\Stima\\\\Tucil3_13522076\\\\src\\\\Utils\\\\"
    + images1[random_idx1]);
JLabel image1 = new JLabel(img1);
JPanel image1Panel = new JPanel(new FlowLayout(FlowLayout.CENTER,
0, 0));
image1Panel.add(image1);
image1Panel.setOpaque(false);
image1Panel.setVisible(false);
image1Panel.setVisible(false);

ImageIcon img2 = new ImageIcon(
    "C:\\\\Users\\\\ASUS\\\\Documents\\\\Sem
4\\\\Stima\\\\Tucil3_13522076\\\\src\\\\Utils\\\\"
    + images2[random_idx2]);
JLabel image2 = new JLabel(img2);
JPanel image2Panel = new JPanel(new FlowLayout(FlowLayout.CENTER,
0, 0));
image2Panel.add(image2);
```

```
image2Panel.setOpaque(false);
image2Panel.setVisible(false);

JRadioButton radio1 = new JRadioButton("UCS");
JRadioButton radio2 = new JRadioButton("GreedyBFS");
JRadioButton radio3 = new JRadioButton("A*");

ButtonGroup radioGroup = new ButtonGroup();
radioGroup.add(radio1);
radioGroup.add(radio2);
radioGroup.add(radio3);

ActionListener radioListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JRadioButton selectedRadio = (JRadioButton) e.getSource();
        selected = selectedRadio.getText();
        System.out.println("Selected: " +
selectedRadio.getText());
    }
};

radio1.addActionListener(radioListener);
radio2.addActionListener(radioListener);
radio3.addActionListener(radioListener);

JPanel radioButtonsPanel = new JPanel();
radioButtonsPanel.setLayout(new GridLayout(1, 3, 5, 5));
radioButtonsPanel.setOpaque(false);
radioButtonsPanel.add(radio1);
radioButtonsPanel.add(radio2);
radioButtonsPanel.add(radio3);

JPanel formPanel = new JPanel();
formPanel.setLayout(new GridLayout(5, 1, 5, 5));
```

```
formPanel.setOpaque(false);

formPanel.add(lbStartWord);
formPanel.add(tfStartWord);
formPanel.add(lbEndWord);
formPanel.add(tfEndWord);
formPanel.add(buttonsPanel);

JPanel pathPanel = new JPanel();
pathPanel.setOpaque(false);
pathPanel.setLayout(new BoxLayout(pathPanel, BoxLayout.Y_AXIS));
tfPath.setAlignmentX(Component.CENTER_ALIGNMENT);
tfTime.setAlignmentX(Component.CENTER_ALIGNMENT);
tfNode.setAlignmentX(Component.CENTER_ALIGNMENT);

pathPanel.add(tfPath);
pathPanel.add(tfTime);
pathPanel.add(tfNode);

JPanel resultPanel = new JPanel();
resultPanel.setLayout(new BoxLayout(resultPanel,
BoxLayout.Y_AXIS));
resultPanel.setOpaque(false);
resultPanel.setFont(mainFont);
resultPanel.add(pathPanel);
resultPanel.add(image1Panel);
resultPanel.add(image2Panel);

JButton btnSearch = new JButton("Solve");
btnSearch.setFont(mainFont);
btnSearch.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        String startWord = tfStartWord.getText();
```

```
        String endWord = tfEndWord.getText();

        Main main = new Main();

        String[] res = main.MainGUI(selected, startWord, endWord);

        tfPath.setText(res[0]);

        tfTime.setText(res[1]);

        tfNode.setText(res[2]);

        random_idx1 = random.nextInt(images1.length);

        random_idx2 = random.nextInt(images2.length);

        // System.out.println(random_idx);

        ImageIcon img1 = new ImageIcon(
            "C:\\\\Users\\\\ASUS\\\\Documents\\\\Sem
4\\\\Stima\\\\Tucil3_13522076\\\\src\\\\Utils\\\\"
            + images1[random_idx1]);

        image1.setIcon(img1);

        ImageIcon img2 = new ImageIcon(
            "C:\\\\Users\\\\ASUS\\\\Documents\\\\Sem
4\\\\Stima\\\\Tucil3_13522076\\\\src\\\\Utils\\\\"
            + images2[random_idx2]);

        image2.setIcon(img2);

        image2Panel.setVisible(false);

        image1Panel.setVisible(false);

        if (res[0].equals("Invalid Input!")) {
            image2Panel.setVisible(true);
        } else {
            image1Panel.setVisible(true);
        }
    }

});

JButton btnClear = new JButton("Clear");

btnClear.setFont(mainFont);

btnClear.addActionListener(new ActionListener() {
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        tfStartWord.setText("");
        tfEndWord.setText("");
        tfPath.setText("");
        tfTime.setText("");
        tfNode.setText("");
        image1Panel.setVisible(false);
        image2Panel.setVisible(false);
    }
}

JPanel buttonsPanel = new JPanel();
buttonsPanel.setLayout(new GridLayout(1, 2, 5, 5));
buttonsPanel.setOpaque(false);
buttonsPanel.add(btnSearch);
buttonsPanel.add(btnClear);

JPanel mainPanel = new JPanel();
mainPanel.setLayout(new BorderLayout());
mainPanel.setBackground(new Color(255, 255, 255));
mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
// mainPanel.add(radioButtonPanel, BorderLayout.NORTH);
mainPanel.add(formPanel, BorderLayout.NORTH);
mainPanel.add(resultPanel, BorderLayout.CENTER);
mainPanel.add(buttonsPanel, BorderLayout.SOUTH);

add(mainPanel);

setTitle("Cukurukuk Word Ladder Solver");
setSize(1000, 1000);
```

```

        setMinimumSize(new Dimension(1000, 1000));

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        setVisible(true);

    }

    public static void main(String[] args) {
        GUI myFrame = new GUI();
        myFrame.intialize();
    }
}

```

VI. Lampiran

Link Github : https://github.com/Akmal2205/Tucil3_13522076

Poin	Ya	Tidak
Program berhasil dijalankan.	✓	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma UCS	✓	
Solusi yang diberikan pada algoritma UCS optimal	✓	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma <i>Greedy Best First Search</i>	✓	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma A*	✓	
[Bonus]: Program memiliki tampilan GUI	✓	