LAPORAN TUGAS KECIL 3 IF2211
STRATEGI ALGORITMA

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

Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Disusun oleh:

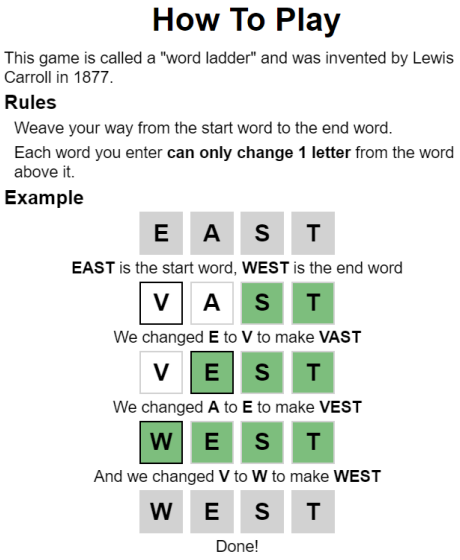
Amalia Putri (13522042)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

1. Pendahuluan

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.



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

2. Analisis dan implementasi dalam algoritma *Uniform Cost Search*, *Greedy Best First Search*, dan *A-Star*

g(n) adalah nilai biaya kumulatif dari node awal (start node) hingga node saat ini (node n). Nilai ini berfungsi untuk menentukan jarak antara node awal hingga node saat ini tanpa memperhitungkan heuristik. h(n) adalah nilai heuristik atau estimasi jarak dari node saat ini (node n) hingga node tujuan (goal node). Nilai ini menunjukkan dugaan seberapa jauh jarak yang tersisa menuju tujuan. f(n) adalah

penjumlahan dari g(n) dan h(n). Artinya, f(n) = g(n) + h(n). Nilai ini digunakan untuk mengevaluasi node berdasarkan jarak yang sudah ditempuh (g(n)) dan jarak perkiraan menuju tujuan (h(n)).

WordLadder sebagai kelas dasar tampaknya menyediakan fungsi umum yang dibutuhkan oleh ketiga algoritma ini, seperti memuat kamus kata (dictionary) dan menentukan kata-kata tetangga (getNeighbors). Keberadaan kelas ini menunjukkan bahwa ketiga algoritma menggunakan struktur data dan mekanisme serupa dalam mengeksplorasi ruang kata yang mungkin.

UCSWordLadder menggunakan pendekatan *Uniform Cost Search* (UCS), yang serupa dengan *Breadth-First Search* (BFS) namun dengan tambahan fitur pengurutan berdasarkan biaya jalur. Dalam konteks Word Ladder, UCS memperlakukan setiap transformasi kata dengan biaya yang sama sehingga dalam hal ini, UCS bekerja mirip dengan BFS. Namun, fitur utamanya adalah menggunakan antrean prioritas yang mengurutkan simpul berdasarkan costFromStart atau g(n) (biaya dari awal ke simpul saat ini). Metode findLadder memulai pencarian dengan menambahkan kata awal ke dalam antrean prioritas, lalu terus mengeksplorasi tetangga kata sampai mencapai kata tujuan. Jika kata saat ini sama dengan kata tujuan, algoritma berhenti dan merekonstruksi jalur dari kata awal ke kata tujuan menggunakan metode reconstructPath. Salah satu keuntungan dari UCS adalah jaminan menemukan jalur terpendek karena biaya aktual dari kata awal digunakan sebagai dasar dalam pengurutan. Meski begitu, UCS bisa menjadi tidak efisien jika ruang pencarian besar dan tidak ada heuristik untuk memberikan panduan.

GBFSWordLadder menggunakan pendekatan Greedy Best-First Search, yang sangat bergantung pada heuristik untuk mengarahkan pencarian. Metode findLadder dalam implementasi ini hanya mengurutkan simpul berdasarkan h(n) (heuristik), sehingga tidak mempertimbangkan biaya dari simpul awal. Hal ini membuat algoritma GBFS cenderung lebih cepat dalam menemukan jalur, tetapi karena tidak mempertimbangkan biaya aktual dari kata awal, jalur yang ditemukan tidak selalu optimal. Dalam konteks Word Ladder, pendekatan ini akan mencoba meminimalkan jarak dari kata saat ini ke tujuan, namun dapat dengan mudah salah

arah jika ada jalur yang tampak menjanjikan berdasarkan heuristik tetapi sebenarnya lebih mahal dari kata awal.

AStarWordLadder adalah implementasi algoritma A*, yang memanfaatkan heuristik untuk memperkirakan biaya dari simpul saat ini ke tujuan. Berbeda dengan UCS yang hanya mempertimbangkan g(n) (biaya dari awal ke simpul saat ini), A* mengkombinasikan g(n) dengan h(n) (heuristik) untuk menghitung f(n). Heuristik yang digunakan adalah jumlah perbedaan karakter antara kata saat ini dan kata tujuan, yang memberikan indikasi seberapa jauh kata ini dari tujuan. Dengan metode findLadder, algoritma ini menggunakan antrean prioritas yang mengurutkan simpul berdasarkan f(n), sehingga simpul yang memiliki estimasi jarak terendah dari kata awal ke tujuan akan dieksplorasi lebih dahulu. Hal ini memberikan efisiensi lebih besar dibandingkan UCS karena simpul yang tidak berpotensi biasanya tidak dijelajahi. Namun, efisiensi A* sangat bergantung pada akurasi heuristik yang digunakan. Jika heuristiknya terlalu lemah atau tidak akurat, keuntungan efisiensi yang diperoleh akan berkurang.

Secara keseluruhan, UCS, A*, dan GBFS memberikan tiga pendekatan berbeda dalam menyelesaikan masalah yang sama. UCS memberikan jaminan jalur terpendek namun cenderung lebih lambat. A* lebih efisien berkat penggunaan heuristik tetapi membutuhkan heuristik yang baik untuk mencapai efisiensi optimal. GBFS sangat cepat dalam menemukan jalur tetapi tidak menjamin optimalitas solusi. Pemilihan algoritma mana yang digunakan sangat tergantung pada kebutuhan spesifik dari aplikasi atau kasus penggunaannya.

3. Source Code Program Implementasi

Terdapat kelas abstrak sekaligus sebagai *superclass*, yakni WordLadder untuk mengurangi *reusability method* yang mirip dari ketiga algoritma sehingga menjadi faktor program ini lebih mangkus dan sangkil. Terdapat tiga *subclass* yang merupakan masing-masing algoritma untuk mencari solusi dari WordLadder, yakni kelas UCSWordLadder, GBFSWordLadder, dan AStarWordLadder. Program ini dapat diimplementasikan dengan dua cara, yakni dalam *Command Line Interface* (CLI) dan *Graphical User Interfaces* (GUI).

```java
package Algorithms;

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

public class WordLadder {
     // ANSI escape codes for colors
     public static final String RESET = "\u001B[0m";
     public static final String GREEN = "\u001B[32m";
     public static final String BLUE = "\u001B[34m";
     public static final String RED = "\u001B[31m";
     public static final String YELLOW = "\u001B[33m";
     public static final String CYAN = "\u001B[36m";

    protected HashSet<String> dictionary;

    /**
     * Constructor kelas WordLadder
     * @param filename nama file yang berisi kamus kata
     */
    public WordLadder(String filename) throws FileNotFoundException {
        dictionary = new HashSet<>();
        loadDictionary(filename);
    }

    /**
     * Method untuk memuat dictionary dari file
     * @param filename nama file dictionary
     */
    private void loadDictionary(String filename) throws FileNotFoundException {
        Scanner scanner = new Scanner(new File(filename));
        while (scanner.hasNext()) {
            String word = scanner.next().toLowerCase();
            dictionary.add(word);
        }
        scanner.close();
    }

    /**
     * Method untuk mengecek apakah dua kata berbeda satu huruf
     * @param word1 kata pertama
     * @param word2 kata kedua
     */
    public boolean isOneLetterDifferent(String word1, String word2) {
        if (word1.length() != word2.length())
            return false;

        int count = 0;
        for (int i = 0; i < word1.length(); i++) {
            if (word1.charAt(i) != word2.charAt(i)) {
                if (++count > 1)
                    return false;
            }
        }
        return count == 1;
    }
```

Gambar 3.1. Kelas WordLadder (1)

```java
    /**
     * Method untuk mendapatkan tetangga dari suatu kata
     * @param word kata yang akan dicari tetangganya
     */
    protected List<String> getNeighbors(String word) {
        List<String> neighbors = new ArrayList<>();
        char[] chars = word.toCharArray();

        for (int i = 0; i < chars.length; i++) {
            char original = chars[i];
            for (char c = 'a'; c <= 'z'; c++) {
                if (c == original)
                    continue;
                chars[i] = c;
                String newWord = new String(chars);

                // Jika kata baru ada di dictionary, maka kata tersebut adalah tetangga
                if (dictionary.contains(newWord)) {
                    neighbors.add(newWord);
                }
            }
            chars[i] = original;
        }

        return neighbors;
    }

    /**
     * Method untuk menghitung jarak antara dua kata
     * @param current kata pertama
     * @param goal kata kedua
     */
    protected int heuristic(String current, String goal) {
        int distance = 0;
        for (int i = 0; i < current.length(); i++) {
            if (current.charAt(i) != goal.charAt(i)) {
                distance++;
            }
        }
        return distance;
    }

    /**
     * Method untuk merekonstruksi path dari node akhir
     * @param endNode node akhir
     */
    protected List<String> reconstructPath(Node endNode) {
        List<String> path = new LinkedList<>();
        Node current = endNode;
        while (current != null) {
            path.add(0, current.word);
            current = current.parent;
        }
        return path;
    }
```

Gambar 3.2. Kelas WordLadder (2)

```java
    /**
     * Kelas Node untuk merepresentasikan node dalam algoritma
     */
    protected class Node {
        String word;
        Node parent;
        int heuristic; // Used for GBFS -> mencari jarak dari node saat ini ke goal: h(n)
        int costFromStart; // Used for A* -> mencari jarak dari start ke node saat ini: g(n)
        int priority; // Used for A* -> costFromStart + heuristic: f(n) = g(n) + h(n)

        // User-defined Constructor GBFS
        public Node(String word, Node parent, int heuristic) {
            this.word = word;
            this.parent = parent;
            this.heuristic = heuristic;
        }

        // User-defined Constructor A*
        public Node(String word, Node parent, int costFromStart, int priority) {
            this.word = word;
            this.parent = parent;
            this.costFromStart = costFromStart;
            this.priority = priority;
        }
    }
}

```

Gambar 3.3. Kelas WordLadder (3)

Tabel 3.1. Kelas WordLadder

No	Constructor/Method	Penjelasan
1	**WordLadder**	Konstruktor kelas **WordLadder**. Menerima parameter **filename** yang menunjukkan lokasi file kamus. Memuat semua kata dalam file kamus ke dalam struktur **HashSet**.
2	**loadDictionary**	Memuat kata-kata dari file kamus ke dalam struktur **HashSet<String>** bernama **dictionary**.
3	**isOneLetterDifferent → boolean**	Memeriksa apakah dua kata berbeda hanya satu huruf.
4	**getNeighbors → List<String>**	Menghasilkan daftar kata tetangga untuk kata yang diberikan. Kata tetangga adalah kata yang berbeda satu huruf dengan kata asal dan juga ada di kamus.
5	**heuristic → int**	Menghitung jarak (heuristik) antara dua kata.
6	**reconstructPath → List<String>**	Mereknstruksi jalur dari node akhir ke node awal. Jalur disimpan dalam bentuk daftar kata-kata dari awal ke akhir.
7	**Node → Inner Class**	Kelas Node digunakan untuk menyimpan data dalam algoritma. Node berisi kata, parent node, heuristic, costFromStart, dan priority.

		1. Node(String word, Node parent, int heuristic): **Konstruktor untuk GBFS** yang mengatur kata, parent, dan heuristik node. 2. Node(String word, Node parent, int costFromStart, int priority): **Konstruktor untuk A*** yang mengatur kata, parent, cost, dan priority node.

3.1. Algoritma *Uniform Cost Search* (UCS)

```java
package Algorithms;

import java.io.FileNotFoundException;
import java.util.*;

public class UCSWordLadder extends WordLadder {
    /**
     * Constructor kelas UCSWordLadder
     * @param filename nama file yang berisi kamus kata
     */
    public UCSWordLadder(String filename) throws FileNotFoundException {
        super(filename);
    }
```

Gambar 3.1.1. Algoritma *Uniform Cost Search* (UCS) (1)

```java
    /**
     * Method untuk mencari ladder dari kata awal ke kata akhir
     * @param start kata awal
     * @param end kata akhir
     */
    public Map<String, Object> findLadder(String start, String end) {
        long startTime = System.currentTimeMillis();

        // Jika kata awal atau akhir tidak ada di kamus atau panjang kata awal tidak sama dengan panjang kata akhir
        if (!dictionary.contains(start) || !dictionary.contains(end) || start.length() != end.length()) {
            return Collections.emptyMap();
        }

        // Priority Queue untuk menyimpan node yang akan dikunjungi
        Queue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparingInt(node -> node.costFromStart));
        priorityQueue.add(new Node(start, null, 0, 0)); // Menambahkan node awal ke priority queue

        // Set untuk menyimpan kata yang sudah dikunjungi
        Set<String> visited = new HashSet<>();

        // Jumlah node yang dikunjungi
        int nodesVisited = 0;

        while (!priorityQueue.isEmpty()) {
            // Mengambil node dengan cost terkecil
            Node current = priorityQueue.poll();
            nodesVisited++;

            // Jika kata saat ini sama dengan kata akhir -> selesai
            if (current.word.equals(end)) {
                long endTime = System.currentTimeMillis();
                List<String> path = reconstructPath(current);
                Map<String, Object> result = new HashMap<>();
                result.put("Execution Time", (endTime - startTime) + " ms");
                result.put("Nodes Visited", nodesVisited);
                result.put("Path", path);
                result.put("Path Length", path.size());
                return result;
            }

            visited.add(current.word); // Menandai kata saat ini sebagai sudah dikunjungi

            // Menambahkan tetangga kata saat ini ke priority queue
            for (String neighbor : getNeighbors(current.word)) {
                // Jika tetangga belum dikunjungi, tambahkan ke priority queue
                if (!visited.contains(neighbor)) {
                    priorityQueue.add(new Node(neighbor, current, current.costFromStart + 1, 0));
                    visited.add(neighbor);
                }
            }
        }

        long endTime = System.currentTimeMillis();
        Map<String, Object> result = new HashMap<>();
        result.put("Execution Time", (endTime - startTime) + " ms");
        result.put("Nodes Visited", nodesVisited);
        result.put("Path", Collections.emptyList());
        result.put("Path Length", 0);
        return result; // No path found
    }
}
```

Gambar 3.1.2. Algoritma *Uniform Cost Search* (UCS) (2)

Tabel 3.1.1. Kelas UCSWordLadder

No	Constructor/Method	Penjelasan
1	**UCSWordLadder**	Konstruktor untuk kelas **UCSWordLadder**, yang menerima parameter **filename** yang menentukan file kamus. Konstruktor menginisialisasi kelas induk dengan nama file ini.
2	**findLadder → Map<String, Object>**	Mencari ladder (rangkaian kata) dari kata awal ke kata akhir dan mengembalikan peta yang berisi detail tentang waktu eksekusi, simpul yang dikunjungi, jalur yang ditemukan, dan panjangnya. 1. Menginisialisasi antrian prioritas (Queue<Node>) untuk menyimpan simpul berdasarkan biaya dari awal (Uniform Cost Search). 2. Menambahkan kata awal ke antrian prioritas dengan biaya awal sebesar 0. 3. Menginisialisasi Set<String> untuk melacak kata yang dikunjungi. 4. Penghitung (nodesVisited) melacak jumlah simpul yang dikunjungi selama pencarian. 5. Selama masih ada simpul dalam antrian prioritas: a. Mengeluarkan simpul dengan biaya terkecil. b. Jika kata saat ini sama dengan kata akhir, kembalikan peta dengan detail eksekusi: Execution Time, Nodes Visited, Path, Path Length. c. Tandai kata saat ini sebagai telah dikunjungi dan temukan semua tetangga yang berbeda satu huruf. d. Tambahkan kata tetangga yang belum dikunjungi ke antrian prioritas dengan biaya yang meningkat. 6. Jika tidak ada jalur yang ditemukan, kembalikan peta yang menunjukkan Execution Time, Nodes Visited, jalur kosong, dan panjang jalur 0.

3.2. Algoritma *Greedy Best First Search* (GBFS)

```java
package Algorithms;

import java.io.FileNotFoundException;
import java.util.*;

public class GBFSWordLadder extends WordLadder {
    private Map<String, Integer> heuristicCache = new HashMap<>();

    /**
     * Constructor kelas GBFSWordLadder
     * @param filename nama file yang berisi kamus kata
     */
    public GBFSWordLadder(String filename) throws FileNotFoundException {
        super(filename);
    }

    /**
     * Method untuk mencari ladder dari kata awal ke kata akhir
     * @param start kata awal
     * @param end kata akhir
     */
    public Map<String, Object> findLadder(String start, String end) {
        long startTime = System.currentTimeMillis();

        // Jika kata awal atau akhir tidak ada di kamus atau panjang kata awal tidak sama dengan panjang kata akhir
        if (!dictionary.contains(start) || !dictionary.contains(end) || start.length() != end.length()) {
            return Collections.emptyMap();
        }

        // Priority queue untuk menyimpan node yang akan dikunjungi
        PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparingInt(node -> node.heuristic));
        priorityQueue.add(new Node(start, null, calculateHeuristic(start, end)));

        // Set untuk menyimpan kata yang sudah dikunjungi
        Set<String> visited = new HashSet<>();
        visited.add(start); // Menandai kata awal sebagai sudah dikunjungi

        // Jumlah node yang dikunjungi
        int nodesVisited = 0;

        while (!priorityQueue.isEmpty()) {
            // Mengambil node dengan heuristic terkecil
            Node current = priorityQueue.poll();
            nodesVisited++;

            // Jika kata saat ini sama dengan kata akhir -> selesai
            if (current.word.equals(end)) {
                long endTime = System.currentTimeMillis();
                List<String> path = reconstructPath(current);
                Map<String, Object> result = new HashMap<>();
                result.put("Execution Time", (endTime - startTime) + " ms");
                result.put("Nodes Visited", nodesVisited);
                result.put("Path", path);
                result.put("Path Length", path.size());
                return result;
            }

            // Menambahkan tetangga kata saat ini ke priority queue
            for (String neighbor : getNeighbors(current.word)) {
                // Jika tetangga belum dikunjungi, tambahkan ke priority queue
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    priorityQueue.add(new Node(neighbor, current, calculateHeuristic(neighbor, end)));
                }
            }
        }

        long endTime = System.currentTimeMillis();
        Map<String, Object> result = new HashMap<>();
        result.put("Execution Time", (endTime - startTime) + " ms");
        result.put("Nodes Visited", nodesVisited);
        result.put("Path", Collections.emptyList());
        result.put("Path Length", 0);
        return result; // No path found
    }
```

Gambar 3.2.1. Algoritma *Greedy Best First Search* (GBFS) (1)

```
76
77      // Method untuk menghitung heuristic dari kata saat ini ke kata akhir
78      private int calculateHeuristic(String word, String end) {
79          // Jika kata sudah pernah dihitung heuristicnya, maka gunakan hasil sebelumnya
80          if (heuristicCache.containsKey(word)) {
81              return heuristicCache.get(word);
82          }
83
84          // Jika panjang kata tidak sama, maka heuristic = MAX_VALUE
85          if (word.length() != end.length()) {
86              return Integer.MAX_VALUE;
87          }
88
89          // Menghitung jarak antara kata saat ini dengan kata akhir
90          int heuristic = 0;
91          for (int i = 0; i < word.length(); i++) {
92              if (word.charAt(i) != end.charAt(i)) {
93                  heuristic++;
94              }
95          }
96
97          heuristicCache.put(word, heuristic);
98          return heuristic;
99      }
100 }
101
```

Gambar 3.2.2. Algoritma *Greedy Best First Search* (GBFS) (2)

Tabel 3.2.1. Kelas GBFSWordLadder

No	Constructor/Method	Penjelasan
1	**GBFSWordLadder**	Konstruktor untuk kelas **GBFSWordLadder**. Menerima parameter **filename** untuk menginisialisasi file kamus yang akan digunakan oleh metode pencarian.
2	**findLadder → Map<String, Object>**	Mencari ladder (rangkaian kata) dari kata awal ke kata akhir dan mengembalikan peta yang berisi detail tentang waktu eksekusi, simpul yang dikunjungi, jalur yang ditemukan, dan panjangnya. 1. Menginisialisasi antrian prioritas (PriorityQueue<Node>) yang menyimpan simpul berdasarkan heuristic (Greedy Best First Search). 2. Menambahkan kata awal ke antrian prioritas dengan heuristic yang dihitung menggunakan metode calculateHeuristic. 3. Menginisialisasi Set<String> untuk melacak kata yang dikunjungi dan menambahkan kata awal. 4. Penghitung (nodesVisited) melacak jumlah simpul yang dikunjungi selama pencarian. 5. Selama masih ada simpul dalam antrian prioritas: a. Mengeluarkan simpul dengan heuristic terkecil. b. Jika kata saat ini sama dengan kata

		akhir, kembalikan peta dengan detail eksekusi: Execution Time, Nodes Visited, Path, Path Length. c. Temukan semua tetangga kata saat ini yang berbeda satu huruf dan tambahkan ke antrian prioritas jika belum dikunjungi. 6. Jika tidak ada jalur yang ditemukan, kembalikan peta yang menunjukkan Execution Time, Nodes Visited, jalur kosong, dan panjang jalur 0.
3	**calculateHeuristic → int**	Menghitung nilai heuristic untuk kata tertentu dibandingkan dengan kata akhir. 1. Jika kata sudah ada dalam cache, gunakan nilai heuristic dari cache. 2. Jika panjang kata berbeda dengan kata akhir, kembalikan Integer.MAX_VALUE. 3. Hitung perbedaan antara kata saat ini dan kata akhir, dan jumlahkan perbedaannya untuk menentukan heuristic. 4. Simpan nilai heuristic dalam cache dan kembalikan nilai tersebut.

3.3. Algoritma *A-Star* (A*)

```java
package Algorithms;

import java.io.FileNotFoundException;
import java.util.*;

public class AStarWordLadder extends WordLadder {
    /**
     * Constructor untuk kelas AStarWordLadder
     *
     * @param filename nama file yang berisi kamus kata
     */
    public AStarWordLadder(String filename) throws FileNotFoundException {
        super(filename);
    }

    /**
     * Method untuk mencari ladder dari kata awal ke kata akhir
     *
     * @param start kata awal
     * @param end   kata akhir
     */
    public Map<String, Object> findLadder(String start, String end) {
        long startTime = System.currentTimeMillis();

        // Jika kata awal atau akhir tidak ada di kamus atau panjang kata awal tidak
        // sama dengan panjang kata akhir
        if (!dictionary.contains(start) || !dictionary.contains(end) || start.length() != end.length()) {
            return Collections.emptyMap();
        }

        // Priority queue untuk menyimpan node yang akan dikunjungi
        PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparingInt(node -> node.priority));
        priorityQueue.add(new Node(start, null, 0, heuristic(start, end)));

        // Map untuk menyimpan cost terbaik dari node awal ke node tertentu
        Map<String, Integer> bestCosts = new HashMap<>();
        bestCosts.put(start, 0);

        // Set untuk menyimpan kata yang sudah dikunjungi
        Set<String> visited = new HashSet<>();

        // Jumlah node yang dikunjungi
        int nodesVisited = 0;

        while (!priorityQueue.isEmpty()) {
            // Mengambil node dengan priority terkecil
            Node current = priorityQueue.poll();
            nodesVisited++;

            // Jika kata saat ini sama dengan kata akhir -> selesai
            if (current.word.equals(end)) {
                long endTime = System.currentTimeMillis();
                List<String> path = reconstructPath(current);
                Map<String, Object> result = new HashMap<>();
                result.put("Execution Time", (endTime - startTime) + " ms");
                result.put("Nodes Visited", nodesVisited);
                result.put("Path", path);
                result.put("Path Length", path.size());
                return result;
            }

            visited.add(current.word); // Menandai kata saat ini sebagai sudah dikunjungi

            // Menambahkan tetangga kata saat ini ke priority queue
            for (String neighbor : getNeighbors(current.word)) {
                if (visited.contains(neighbor)) {
                    continue;
                }

                int newCost = current.costFromStart + 1; // Cost dari node awal ke tetangga
                // Jika cost baru lebih kecil dari cost terbaik sebelumnya, update cost terbaik
                if (!bestCosts.containsKey(neighbor) || newCost < bestCosts.get(neighbor)) {
                    bestCosts.put(neighbor, newCost); // Update cost terbaik
                    int priority = newCost + heuristic(neighbor, end);  // Priority = cost + heuristic : f(n) = g(n) + h(n)
                    priorityQueue.add(new Node(neighbor, current, newCost, priority));
                }
            }
        }

        long endTime = System.currentTimeMillis();
        Map<String, Object> result = new HashMap<>();
        result.put("Execution Time", (endTime - startTime) + " ms");
        result.put("Nodes Visited", nodesVisited);
        result.put("Path", Collections.emptyList());
        result.put("Path Length", 0);
        return result; // No path found
    }
}

```

Gambar 3.3.1. Algoritma *A-Star* (A*)

Tabel 3.3.1. Kelas AStarWordLadder

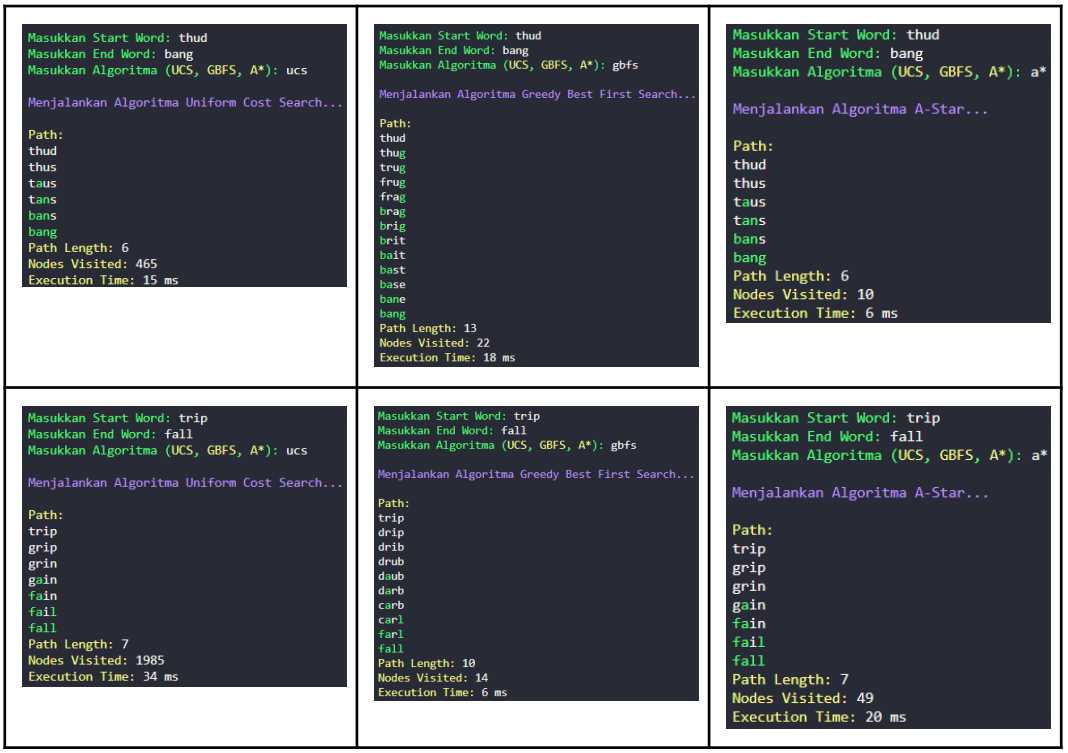
No	Constructor/Method	Penjelasan
1	**AStarWordLadder**	Konstruktor untuk kelas **AStarWordLadder**. Menerima parameter **filename** untuk menginisialisasi file kamus yang akan digunakan oleh metode pencarian.
2	**findLadder → Map<String, Object>**	Mencari ladder (rangkaian kata) dari kata awal ke kata akhir menggunakan algoritma A* dan mengembalikan peta yang berisi detail tentang waktu eksekusi, simpul yang dikunjungi, jalur yang ditemukan, dan panjang jalur. 1. Menginisialisasi antrian prioritas (PriorityQueue<Node>) untuk menyimpan simpul berdasarkan priority (heuristik A*). 2. Menambahkan kata awal ke antrian prioritas dengan cost nol dan heuristic yang dihitung menggunakan metode heuristic. 3. Menginisialisasi Map<String, Integer> untuk melacak cost terbaik dari kata awal ke kata tertentu dan menetapkan cost nol ke kata awal. 4. Penghitung (nodesVisited) melacak jumlah simpul yang dikunjungi selama pencarian. 5. Set (Set<String>) digunakan untuk melacak kata yang sudah dikunjungi. 6. Selama masih ada simpul dalam antrian prioritas: a. Mengeluarkan simpul dengan priority terkecil. b. Jika kata saat ini sama dengan kata akhir, kembalikan peta dengan detail eksekusi: Execution Time, Nodes Visited, Path, dan Path Length. c. Temukan semua tetangga kata saat ini yang berbeda satu huruf, hitung cost dan priority mereka, dan tambahkan ke antrian prioritas jika cost baru lebih rendah dari sebelumnya. 7. Jika tidak ada jalur yang ditemukan, kembalikan peta yang menunjukkan Execution Time, Nodes Visited, jalur kosong, dan panjang jalur 0.

4. Tangkapan Layar Hasil Keluaran dari Masukan

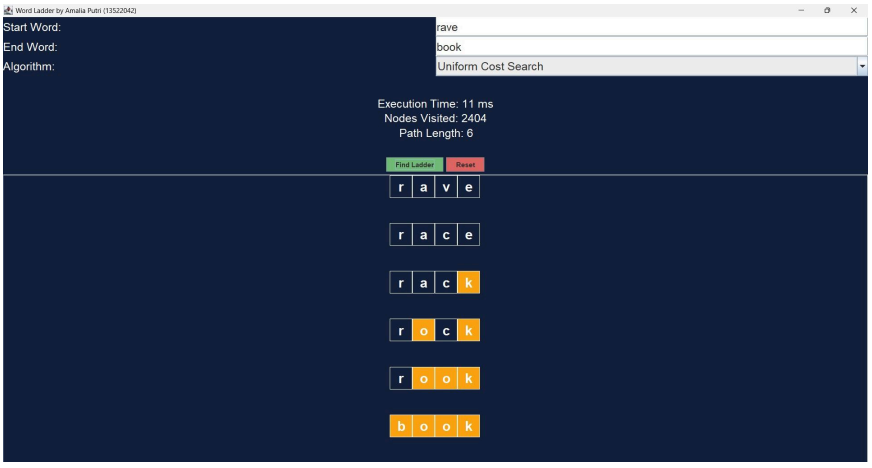
Tabel 4.1. Tangkapan Layar Hasil I/O *Command Line Interface* (CLI)

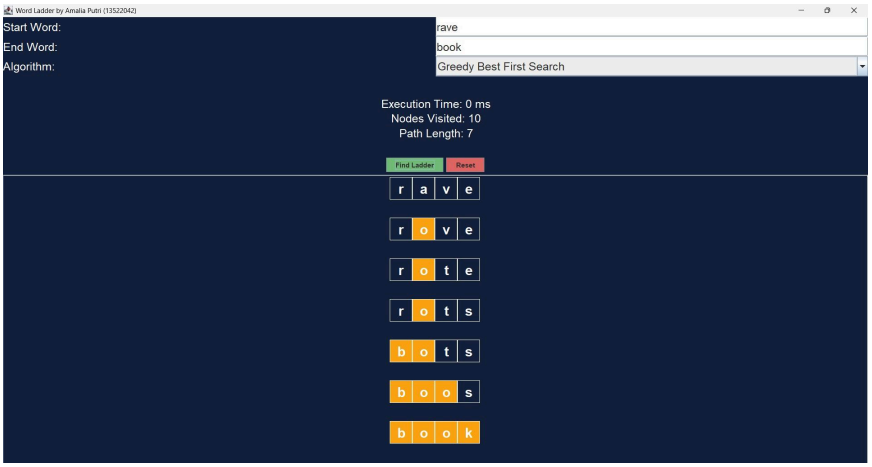
***Uniform Cost Search Algorithm* (UCS)**	***Greedy Best First Search* (GBFS)**	***A-Star* (A*)**
Masukkan Start Word: rave Masukkan End Word: book Masukkan Algoritma (UCS, GBFS, A*): ucs Menjalankan Algoritma Uniform Cost Search... Path: rave race rack rock rook book Path Length: 6 Nodes Visited: 2404 Execution Time: 38 ms	Masukkan Start Word: rave Masukkan End Word: book Masukkan Algoritma (UCS, GBFS, A*): gbfs Menjalankan Algoritma Greedy Best First Search... Path: rave rove rote rots bots boos book Path Length: 7 Nodes Visited: 10 Execution Time: 16 ms	Masukkan Start Word: rave Masukkan End Word: book Masukkan Algoritma (UCS, GBFS, A*): a* Menjalankan Algoritma A-Star... Path: rave race rack back bock book Path Length: 6 Nodes Visited: 45 Execution Time: 8 ms
Masukkan Start Word: atlases Masukkan End Word: cabaret Masukkan Algoritma (UCS, GBFS, A*): ucs Menjalankan Algoritma Uniform Cost Search... Path: atlases anlases anlaces unlaces unlaced unladed unfaded unfaked uncaked uncakes uncases uneases ureases creases cresses crosses crosser crosier crozier crazier brazier brakier beakier peakier peckier pickier dickier dickies hickies hackies hackles heckles deckles deciles defiles refiles reviles reviled reveled raveled ravened havened havered wavered watered catered capered tapered tabered tabored taboret tabaret cabaret Path Length: 53 Nodes Visited: 7648 Execution Time: 111 ms	Masukkan Start Word: atlases Masukkan End Word: cabaret Masukkan Algoritma (UCS, GBFS, A*): gbfs Menjalankan Algoritma Greedy Best First Search... Path: atlases anlases anlaces unlaces unlaced unladed unfaded unfaked unbaked unbased uncased uncases uneases ureases creases crates created cheated cheater cheaper cheeper cheerer cheered cheeked checked chucked shucked shucker shicker snicker snicked snacked swacked swanked swanker spanker spanner scanner scanter scanted scatted scathed scathes swathes swather slather slatier platier peatier pettier pottier cottier cattier catties patties panties pandies candies candles candler handler handier hardier	Masukkan Start Word: atlases Masukkan End Word: cabaret Masukkan Algoritma (UCS, GBFS, A*): a* Menjalankan Algoritma A-Star... Path: atlases anlases anlaces unlaces unlaced unladed unfaded unfaked uncaked uncakes uncases uneases ureases creases cresses crosses crosser crosier crozier crazier brazier brakier beakier peakier peckier pickier dickier dickies hickies hackies hackles heckles deckles deciles defiles defiled deviled develed reveled raveled ravened havened havered wavered watered catered capered tapered tabered tabored taboret tabaret cabaret Path Length: 53 Nodes Visited: 7853 Execution Time: 139 ms

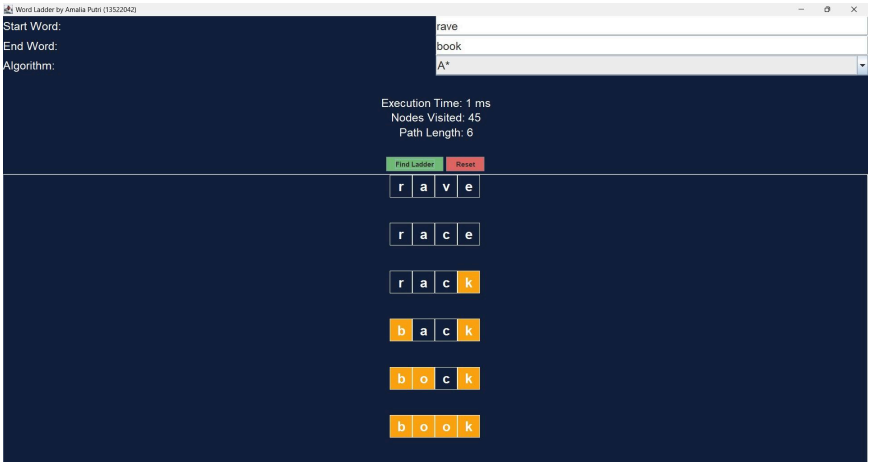
	lardier larkier barkier balkier talkier tackier tackler cackler cackles hackles heckles deckles deciles defiles refiles refines repines rapines ravines ravined ravened havened havered wavered watered catered capered tapered tabered tabored taboret tabaret cabaret Path Length: 96 Nodes Visited: 4712 Execution Time: 98 ms	
Masukkan Start Word: frown Masukkan End Word: smile Masukkan Algoritma (UCS, GBFS, A*): ucs Menjalankan Algoritma Uniform Cost Search... Path: frown frows flows slows slots spots spits spite smite smile Path Length: 10 Nodes Visited: 4347 Execution Time: 70 ms	Masukkan Start Word: frown Masukkan End Word: smile Masukkan Algoritma (UCS, GBFS, A*): gbfs Menjalankan Algoritma Greedy Best First Search... Path: frown brown brows broos brios trios trips tripe trine thine shine spine spile smile Path Length: 14 Nodes Visited: 24 Execution Time: 6 ms	Masukkan Start Word: frown Masukkan End Word: smile Masukkan Algoritma (UCS, GBFS, A*): a* Menjalankan Algoritma A-Star... Path: frown frows flows slows slots slits skits skite smite smile Path Length: 10 Nodes Visited: 285 Execution Time: 14 ms
Masukkan Start Word: hope Masukkan End Word: wish Masukkan Algoritma (UCS, GBFS, A*): ucs Menjalankan Algoritma Uniform Cost Search... Path: hope hose host wost wist wish Path Length: 6 Nodes Visited: 2023 Execution Time: 48 ms	Masukkan Start Word: hope Masukkan End Word: wish Masukkan Algoritma (UCS, GBFS, A*): gbfs Menjalankan Algoritma Greedy Best First Search... Path: hope hose nose nosh posh pish wish Path Length: 7 Nodes Visited: 13 Execution Time: 4 ms	Masukkan Start Word: hope Masukkan End Word: wish Masukkan Algoritma (UCS, GBFS, A*): a* Menjalankan Algoritma A-Star... Path: hope hose pose posh pish wish Path Length: 6 Nodes Visited: 27 Execution Time: 8 ms



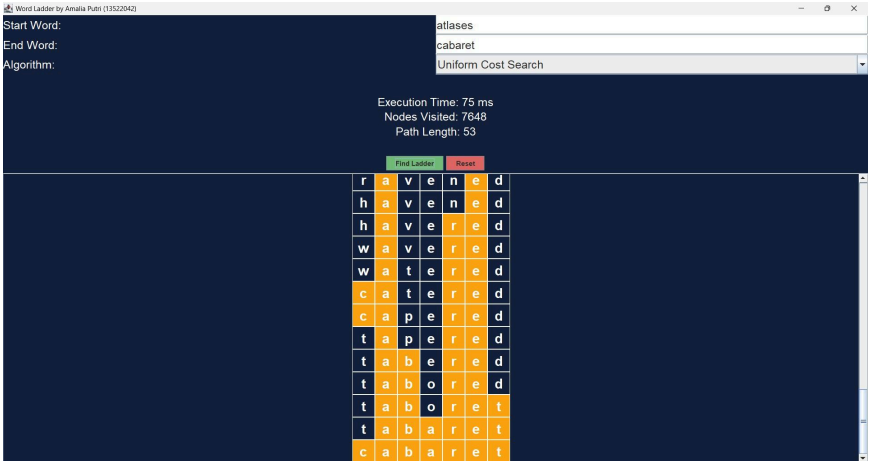
Tabel 4.2. Tangkapan Layar Hasil I/O *Graphical User Interfaces* (GUI)

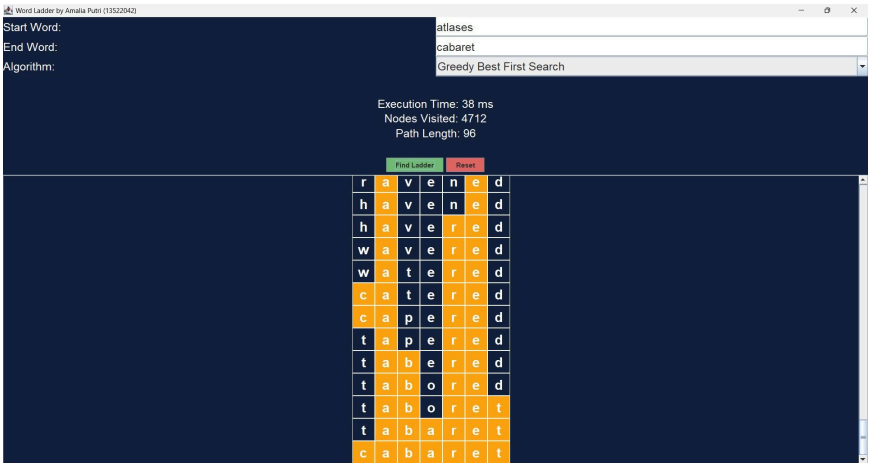
Test-Case 1


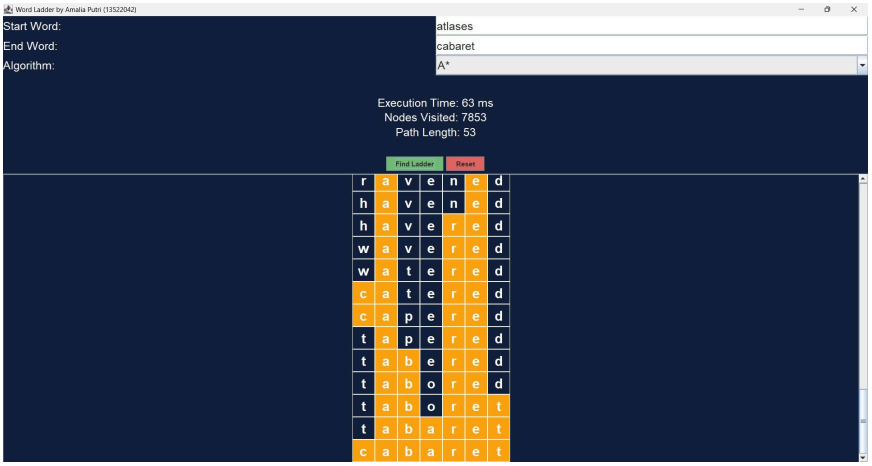

Word Ladder by Amalia Putri (13522042)
Start Word:
rave
End Word:
book
Algorithm:
Greedy Best First Search
Execution Time: 0 ms
Nodes Visited: 10
Path Length: 7
Find Ladder
Reset
r a v e
r o v e
r o t e
r o t s
b o t s
b o o s
b o o k


Word Ladder by Amalia Putri (13522042)
Start Word:
rave
End Word:
book
Algorithm:
A*
Execution Time: 1 ms
Nodes Visited: 45
Path Length: 6
Find Ladder
Reset
r a v e
r a c e
r a c k
b a c k
b o c k
b o o k

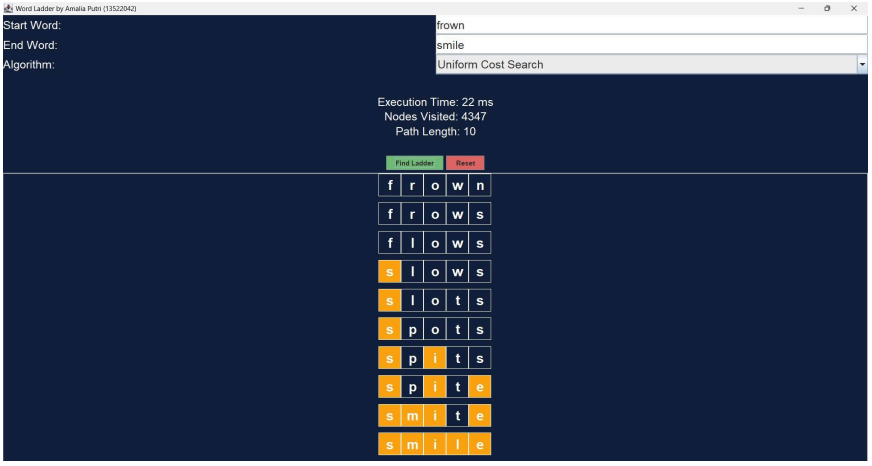
Test-Case 2

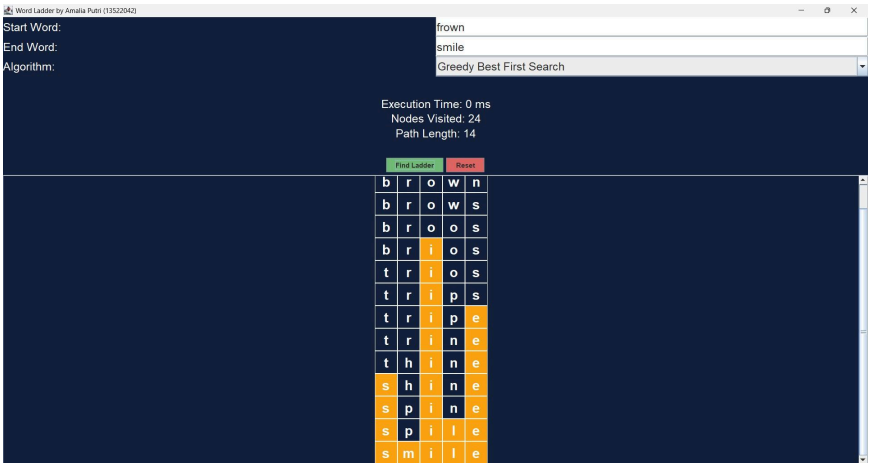

Word Ladder by Amalia Putri (13522042)
Start Word:
atlases
End Word:
cabaret
Algorithm:
Uniform Cost Search
Execution Time: 75 ms
Nodes Visited: 7648
Path Length: 53
Find Ladder
Reset
r a v e n e d
h a v e n e d
h a v e r e d
w a v e r e d
w a t e r e d
c a t e r e d
c a p e r e d
t a p e r e d
t a b e r e d
t a b o r e d
t a b o r e t
t a b a r e t
c a b a r e t

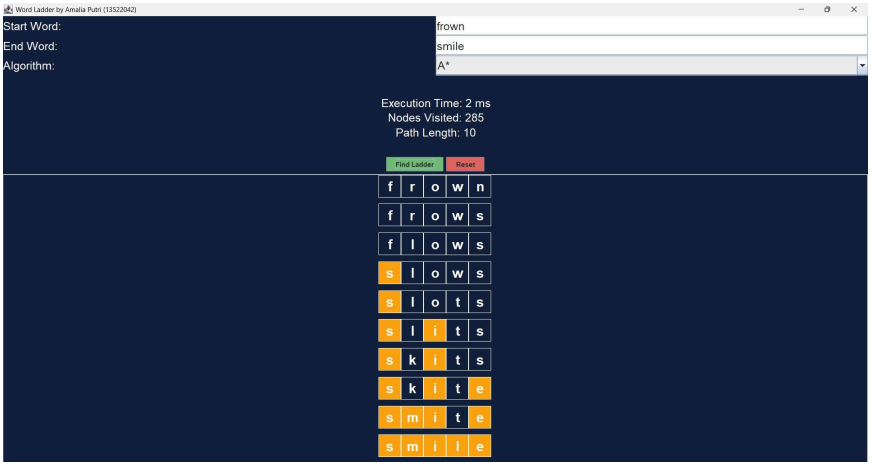

Word Ladder by Amalia Putri (13522042)
Start Word:
atlases
End Word:
cabaret
Algorithm:
Greedy Best First Search
Execution Time: 38 ms
Nodes Visited: 4712
Path Length: 96
Find Ladder
Reset
r a v e n e d
h a v e n e d
h a v e r e d
w a v e r e d
w a t e r e d
c a t e r e d
c a p e r e d
t a p e r e d
t a b e r e d
t a b o r e d
t a b o r e t
t a b a r e t
c a b a r e t


Word Ladder by Amalia Putri (13522042)
Start Word:
atlases
End Word:
cabaret
Algorithm:
A*
Execution Time: 63 ms
Nodes Visited: 7853
Path Length: 53
Find Ladder
Reset
r a v e n e d
h a v e n e d
h a v e r e d
w a v e r e d
w a t e r e d
c a t e r e d
c a p e r e d
t a p e r e d
t a b e r e d
t a b o r e d
t a b o r e t
t a b a r e t
c a b a r e t

Test-Case 3

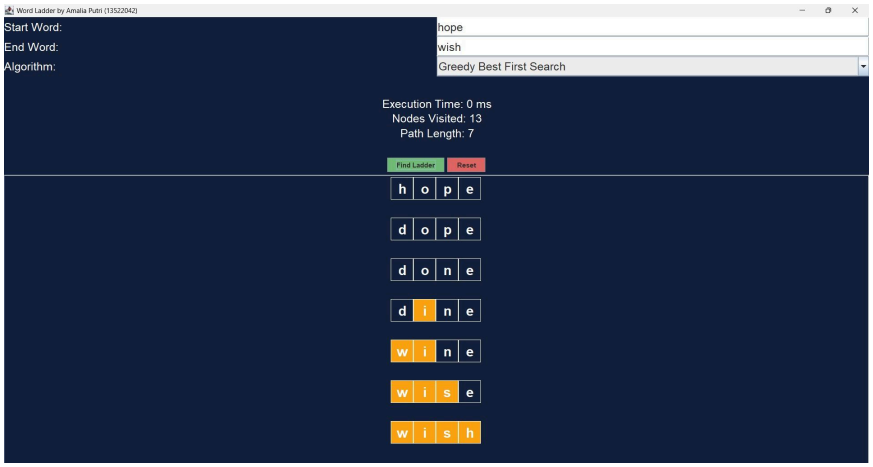

Word Ladder by Amalia Putri (13522042)
Start Word:
frown
End Word:
smile
Algorithm:
Uniform Cost Search
Execution Time: 22 ms
Nodes Visited: 4347
Path Length: 10
Find Ladder
Reset
f r o w n
f r o w s
f l o w s
s l o w s
s l o t s
s p o t s
s p i t s
s p i t e
s m i t e
s m i l e

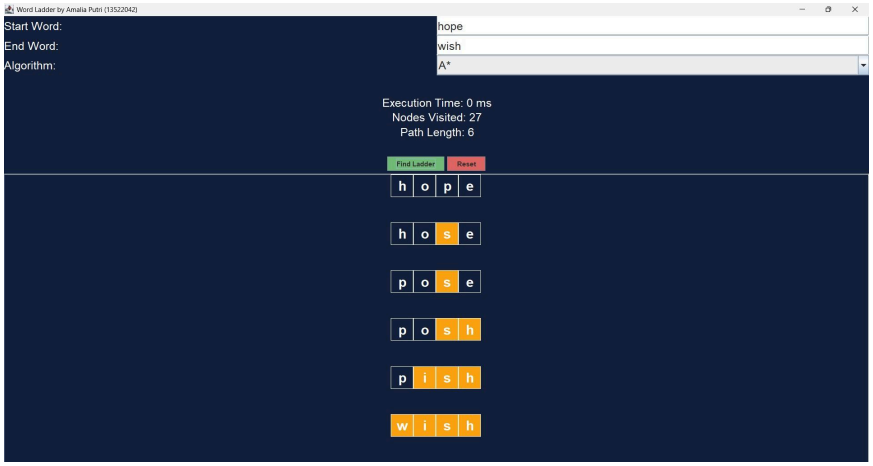

Word Ladder by Amalia Putri (13522042)
Start Word:
frown
End Word:
smile
Algorithm:
Greedy Best First Search
Execution Time: 0 ms
Nodes Visited: 24
Path Length: 14
Find Ladder
Reset
b r o w n
b r o w s
b r o o s
b r i o s
t r i o s
t r i p s
t r i p e
t r i n e
t h i n e
s h i n e
s p i n e
s p i l e
s m i l e


Word Ladder by Amalia Putri (13522042)
Start Word:
frown
End Word:
smile
Algorithm:
A*
Execution Time: 2 ms
Nodes Visited: 285
Path Length: 10
Find Ladder
Reset
f r o w n
f r o w s
f l o w s
s l o w s
s l o t s
s l i t s
s k i t s
s k i t e
s m i t e
s m i l e

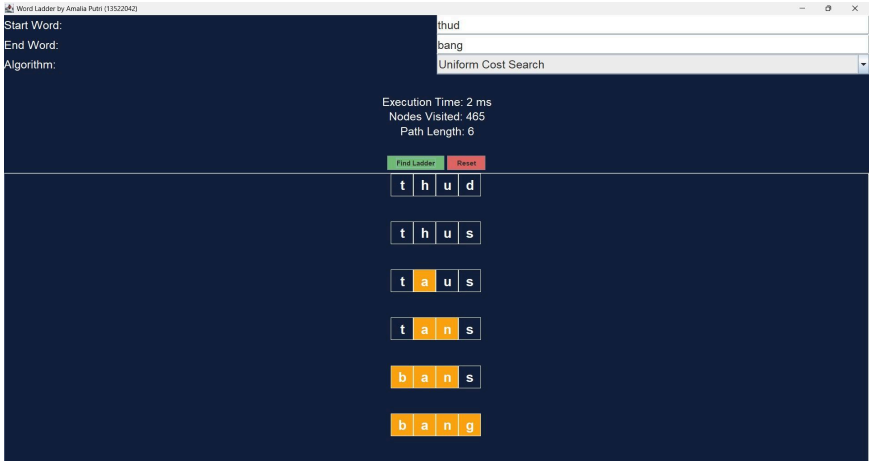
Test-Case 4


Word Ladder by Amalia Putri (13522042)
Start Word:
hope
End Word:
wish
Algorithm:
Uniform Cost Search
Execution Time: 9 ms
Nodes Visited: 2023
Path Length: 6
Find Ladder
Reset
h o p e
h o s e
h o s t
w o s t
w i s t
w i s h

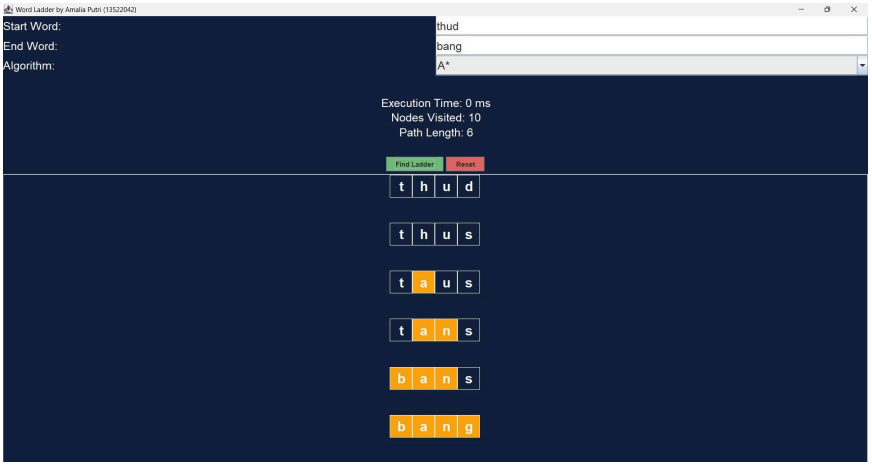

Word Ladder by Amalia Putri (13522042)
Start Word:
hope
End Word:
wish
Algorithm:
Greedy Best First Search
Execution Time: 0 ms
Nodes Visited: 13
Path Length: 7
Find Ladder
Reset
h o p e
d o p e
d o n e
d i n e
w i n e
w i s e
w i s h


Word Ladder by Amalia Putri (13522042)
Start Word:
hope
End Word:
wish
Algorithm:
A*
Execution Time: 0 ms
Nodes Visited: 27
Path Length: 6
Find Ladder
Reset
h o p e
h o s e
p o s e
p o s h
p i s h
w i s h

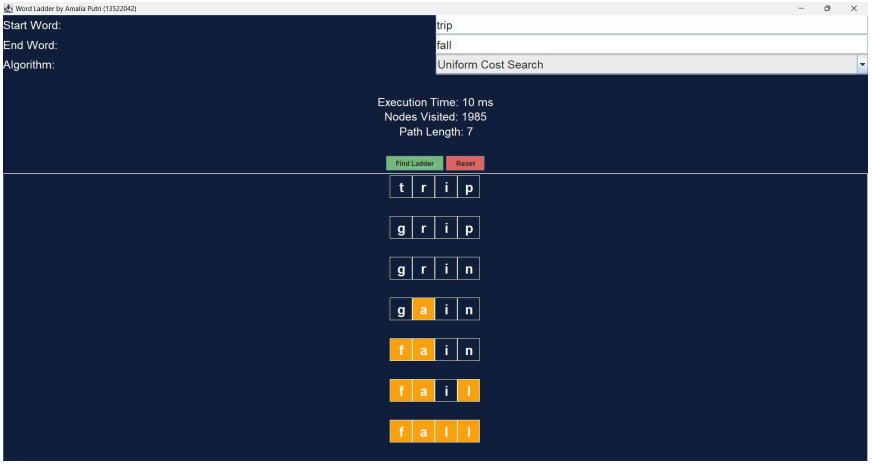
Test-Case 5

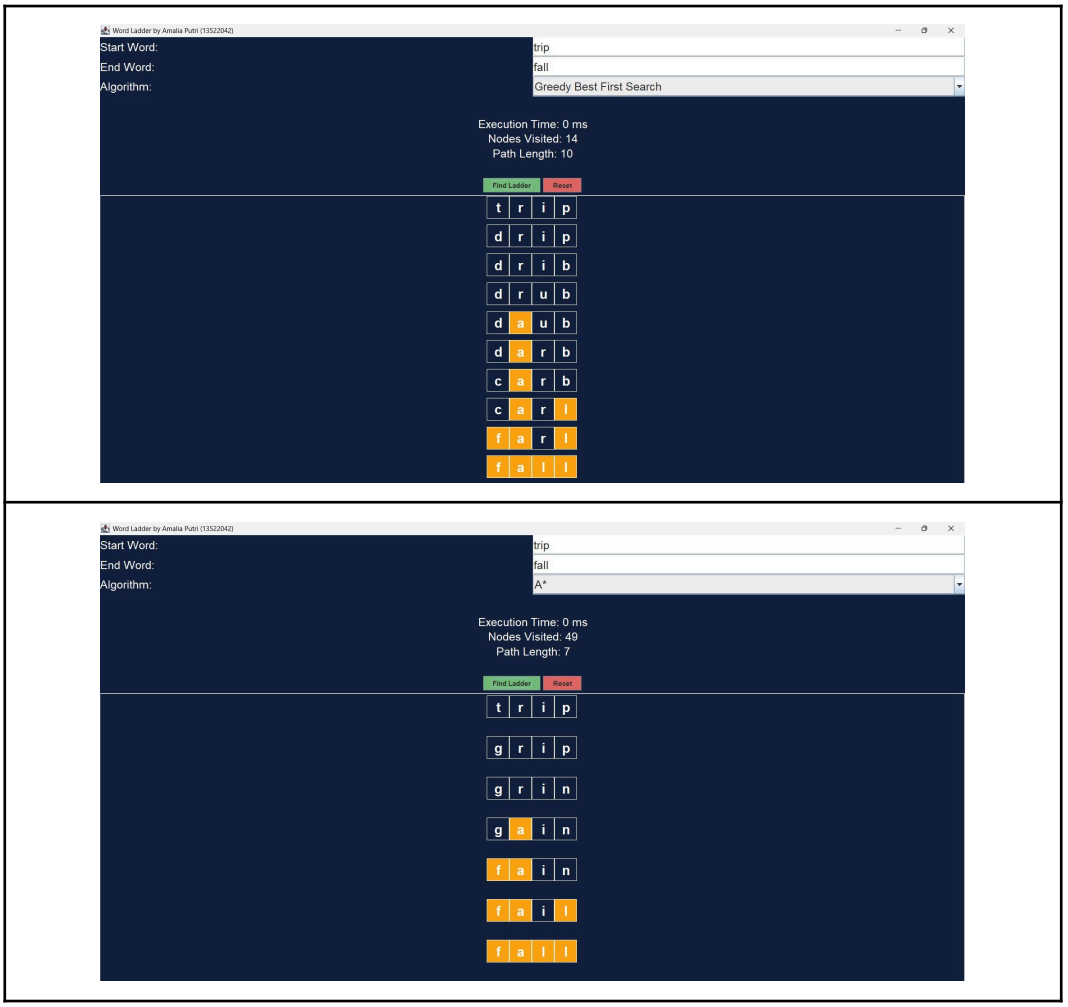

Word Ladder by Amalia Putri (13522042)
Start Word:
thud
End Word:
bang
Algorithm:
Uniform Cost Search
Execution Time: 2 ms
Nodes Visited: 465
Path Length: 6
Find Ladder
Reset
t h u d
t h u s
t a u s
t a n s
b a n s
b a n g


Word Ladder by Amalia Putri (13522042)
Start Word:
thud
End Word:
bang
Algorithm:
Greedy Best First Search
Execution Time: 0 ms
Nodes Visited: 30
Path Length: 14
Find Ladder
Reset
t h u g
t r u g
f r u g
f r a g
b r a g
b r i g
b r i s
b r o s
b o o s
b o g s
b a g s
b a n s
b a n g


Word Ladder by Amalia Putri (13522042)
Start Word:
thud
End Word:
bang
Algorithm:
A*
Execution Time: 0 ms
Nodes Visited: 10
Path Length: 6
Find Ladder
Reset
t h u d
t h u s
t a u s
t a n s
b a n s
b a n g

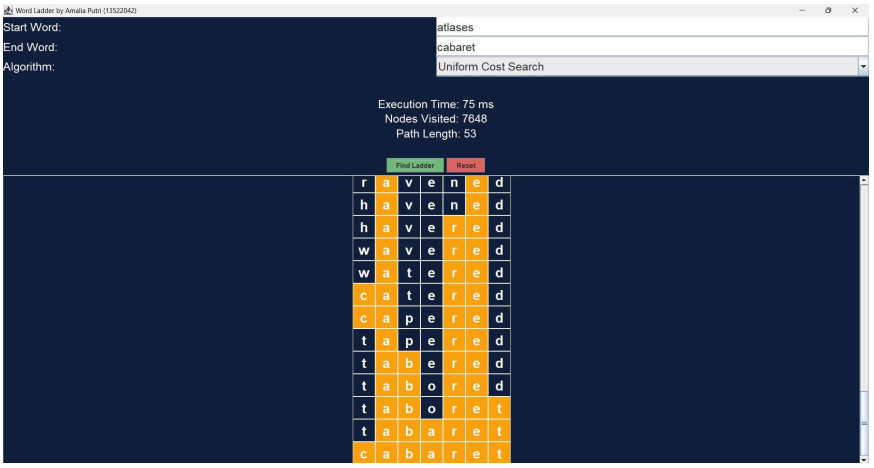
Test-Case 6


Word Ladder by Amalia Putri (13522042)
Start Word:
trip
End Word:
fall
Algorithm:
Uniform Cost Search
Execution Time: 10 ms
Nodes Visited: 1985
Path Length: 7
Find Ladder
Reset
t r i p
g r i p
g r i n
g a i n
f a i n
f a i l
f a l l



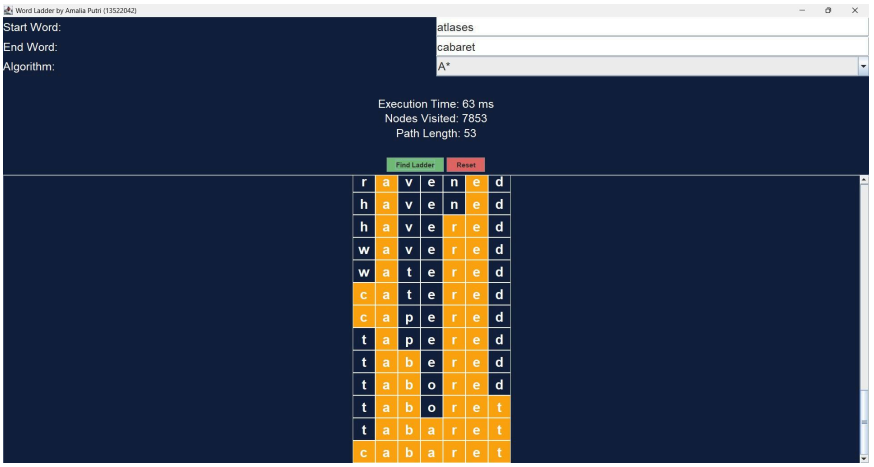
5. Hasil Analisis Perbandingan Kedua Solusi Algoritma

Pada algoritma UCS (Uniform Cost Search), waktu eksekusi mencapai 76 ms dengan jumlah simpul yang dikunjungi sebanyak 7.648 dan panjang lintasan 53 kata. Algoritma ini menggunakan metode pencarian dengan prioritas berdasarkan biaya terendah. Hal ini menjadikan UCS optimal dalam mencari solusi namun memerlukan waktu yang cukup lama dan memori yang besar saat dataset menjadi besar karena menjelajahi setiap simpul dengan lebih menyeluruh.



Gambar 5.1. Test-Case 2 UCS

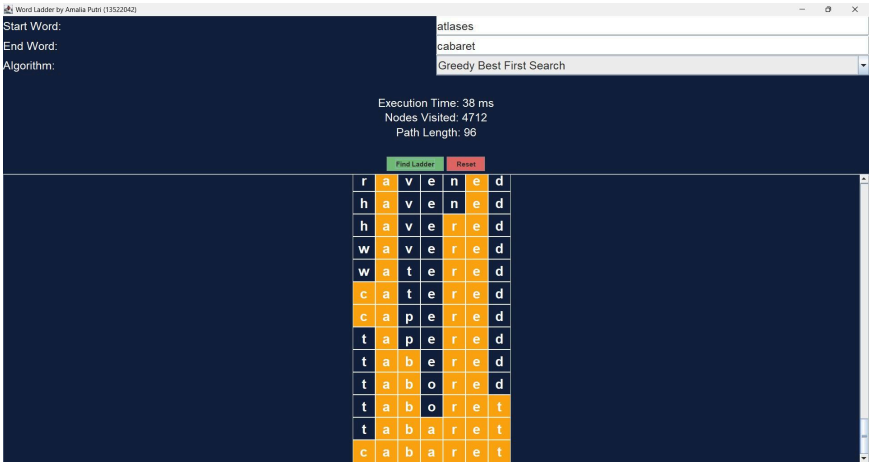
Algoritma A* memiliki waktu eksekusi 63 ms dengan simpul yang dikunjungi sebanyak 7.853 dan panjang lintasan yang sama yaitu 53 kata. A* memiliki pendekatan yang lebih efisien karena menggunakan kombinasi biaya dari simpul awal ke simpul saat ini (g(n)) dan perkiraan biaya dari simpul saat ini ke tujuan (h(n)). Kombinasi ini memungkinkan A* untuk menemukan solusi optimal dengan lebih cepat dibandingkan UCS dalam beberapa kasus karena mengurangi jumlah simpul yang dijelajahi. Namun, A* masih memerlukan memori yang cukup besar karena harus menyimpan informasi biaya setiap simpul yang telah dikunjungi.



Gambar 5.2. Test-Case 2 A*

Greedy Best First Search (GBFS) adalah algoritma yang paling cepat dari ketiganya dengan waktu eksekusi hanya 30 ms. Namun, jumlah simpul yang dikunjungi hanya 4.712 dan menghasilkan panjang lintasan terpanjang yakni 96

kata. Hal ini karena GBFS lebih fokus pada pendekatan heuristik atau perkiraan biaya menuju tujuan, yang terkadang menyebabkan eksplorasi jalan yang lebih panjang dan tidak optimal dalam hal panjang lintasan.



Gambar 5.3. Test-Case 2 GBFS

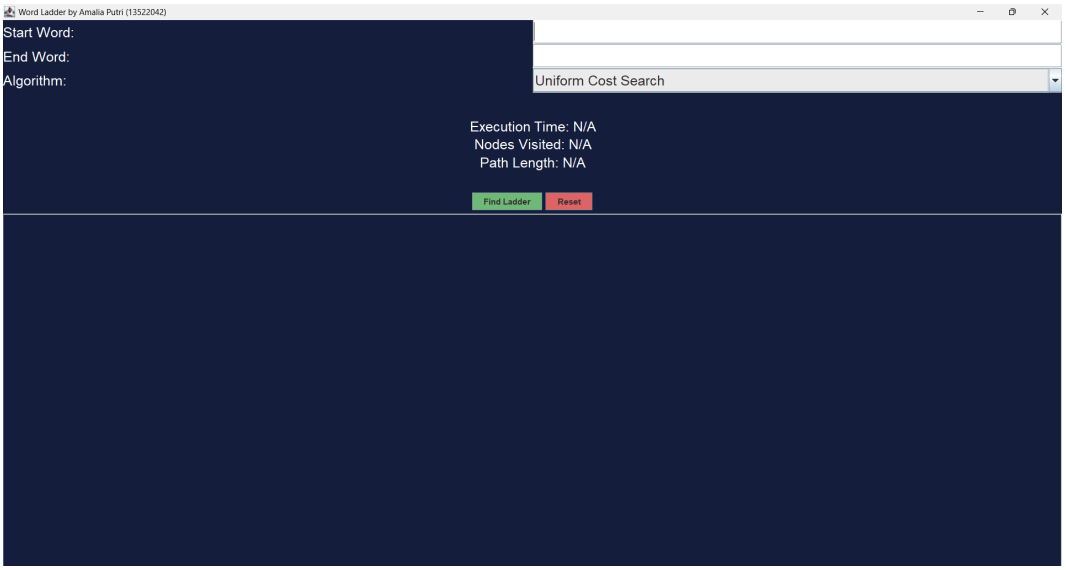
Secara keseluruhan, dari gambar yang diberikan, algoritma A* mampu mencapai solusi optimal dengan waktu eksekusi dan penggunaan memori yang lebih baik dibandingkan UCS, meskipun membutuhkan lebih banyak memori daripada GBFS. Sementara itu, GBFS lebih cepat dalam waktu eksekusi, tetapi seringkali tidak optimal dalam panjang lintasan karena fokus pada perkiraan heuristik.

6. Implementasi Bonus

Implementasi *Graphical User Interfaces* (GUI) pada Java dijelaskan pada tabel berikut ini.

Komponen	Deskripsi
Frame Utama	JFrame yang menjadi wadah utama bagi seluruh komponen GUI lainnya. Menggunakan BorderLayout.
Panel Input	1. Start Word (startField): Input teks untuk memasukkan kata awal. 2. End Word (endField): Input teks

	untuk memasukkan kata tujuan. 3. Algorithm Selection (algorithmSelection): Dropdown untuk memilih algoritma (UCS, GBFS, A*).
Panel Informasi	1. Execution Time (timeExecutionLabel): Menampilkan waktu eksekusi pencarian. 2. Nodes Visited (nodesVisitedLabel): Menampilkan jumlah simpul yang dikunjungi. 3. Path Length (pathLengthLabel): Menampilkan panjang lintasan yang ditemukan.
Panel Tombol	1. Find Ladder (findButton): Tombol untuk memulai pencarian. 2. Reset (resetButton): Tombol untuk menghapus input dan hasil.
Panel Grid	Menampilkan hasil pencarian sebagai grid kata-kata. Kata-kata yang ditemukan ditampilkan dalam kotak-kotak, dengan huruf-huruf yang sesuai dengan kata akhir disorot.



Gambar 6.1. Tampilan GUI

DAFTAR PUSTAKA

1. Munir, Rinaldi. “Penentuan Rute (Route/Path Planning) Bagian 1” (online). (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf, diakses pada 3 Mei 2024).
2. Munir, Rinaldi. “Penentuan Rute (Route/Path Planning) Bagian 2” (online). (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf, diakses pada 3 Mei 2024).

LAMPIRAN

Pengecekan 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. [Bonus]: Program memiliki tampilan GUI	✓	

Tabel 3. Tabel Pengecekan Program

Repository

Link Repository dari Tugas Kecil 03 IF2211 Strategi Algoritma adalah sebagai berikut.

https://github.com/amaliap21/Tucil3_13522042.git