

# **LAPORAN TUGAS KECIL 03**

## **IF2211 STRATEGI ALGORITMA**

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



Disusun oleh:

Novelya Putri Ramadhani K-02 13522096

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2023-2024**

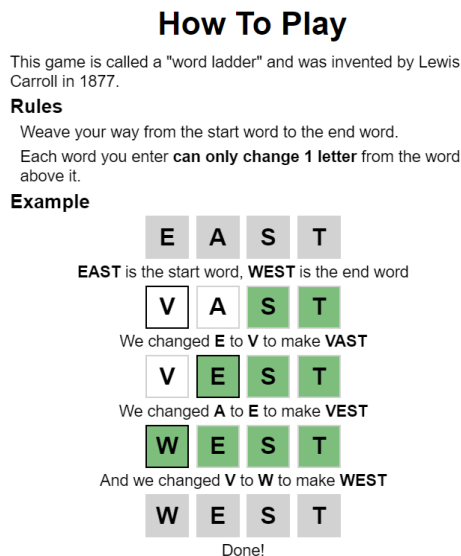
## **DAFTAR ISI**

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB 1</b>	<b>3</b>
<b>BAB 2</b>	<b>4</b>
<b>BAB 3</b>	<b>7</b>
<b>BAB 4</b>	<b>18</b>
<b>BAB 5</b>	<b>25</b>
<b>BAB 6</b>	<b>27</b>
<b>BAB 7</b>	<b>29</b>

# BAB 1

## DESKRIPSI MASALAH

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



**Gambar 1.** Ilustrasi dan Peraturan Permainan *Word Ladder*

(Sumber: <https://wordwormdormdork.com/>)

(Dikutip dari: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tucil3-2024.pdf>)

## BAB 2

### ANALISIS ALGORITMA

Terdapat tiga algoritma yang digunakan untuk menemukan solusi paling optimal dari permainan Word Ladder yang akan dibandingkan, yaitu Algoritma Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A\*.

#### 2.1 Algoritma Uniform Cost Search (UCS)

UCS adalah algoritma pencarian yang memilih node berikutnya berdasarkan cost kumulatif terendah dari node awal ke node saat ini. UCS tidak mempertimbangkan cost perkiraan ke tujuan. Langkah-langkah algoritma yang digunakan dalam menyelesaikan permainan dengan algoritma UCS adalah sebagai berikut:

1. Inisialisasi: Mulai dengan node awal yaitu node dengan kata “Start Word” yang telah dimasukkan. Node ini dimasukkan ke dalam priority queue dengan cost awal 0 untuk diproses.
2. Iterasi: Selama queue tidak kosong,
  - Ambil node dengan cost terendah dari queue.
  - Periksa apakah node ini adalah tujuan (katanya adalah “End Word”). Jika ya, hentikan pencarian.
  - Jika tidak, ekspansi semua tetangga dari node yang didapatkan dari Word Graph yang telah terdefinisi di awal dengan hubungan sisi berupa kata dengan perbedaan 1 huruf dengan kata saat ini.
  - Untuk setiap tetangga, hitung cost terakumulasi dan masukkan ke dalam queue jika kata tersebut belum pernah dikunjungi.

Pada UCS, didefinisikan  $f(n)$  dan  $g(n)$  sebagai berikut.

- $g(n)$  adalah total cost dari node awal ke node  $n$ .
- $f(n) = g(n)$ . Fungsi evaluasi  $f(n)$  sama dengan  $g(n)$  karena hanya cost saja yang diperhitungkan, tanpa heuristik.

Pada kasus Word Ladder, UCS beroperasi mirip dengan BFS karena setiap langkah, yaitu perubahan satu huruf ke kata berikutnya, memiliki cost yang sama. Akibatnya, UCS akan menjelajahi node-node dalam urutan yang sama dengan BFS.

## 2.2 Algoritma Greedy Best First Search (GBFS)

GBFS adalah algoritma pencarian yang memilih node berikutnya berdasarkan perkiraan cost terendah ke tujuan, tanpa mempertimbangkan cost yang telah dikeluarkan untuk mencapai node saat ini. Langkah-langkah algoritma yang digunakan dalam menyelesaikan permainan dengan algoritma GBFS adalah sebagai berikut:

1. Inisialisasi: Mulai dengan node awal yaitu node dengan kata “Start Word” yang telah dimasukkan. Node ini dimasukkan ke dalam priority queue dengan heuristik berupa jumlah karakter yang berbeda antara kata awal dan kata tujuan untuk diproses.
2. Iterasi: Selama queue tidak kosong,
  - Ambil node dengan heuristik terendah dari queue.
  - Periksa apakah node ini adalah tujuan (katanya adalah “End Word”). Jika ya, hentikan pencarian.
  - Jika tidak, ekspansi semua tetangga dari node yang didapatkan dari Word Graph yang telah terdefinisi di awal dengan hubungan sisi berupa kata dengan perbedaan 1 huruf dengan kata saat ini.
  - Untuk setiap tetangga, hitung kembali nilai heuristiknya dan masukkan ke dalam queue jika kata tersebut belum pernah dikunjungi.

Pada GBFS, didefinisikan  $f(n)$  dan  $g(n)$  sebagai berikut.

- $g(n)$  tidak relevan.
- $f(n) = h(n)$ . Fungsi evaluasi  $f(n)$  hanya bergantung pada heuristik dari node  $n$  ke tujuan, yaitu jumlah karakter yang berbeda dari kata pada node  $n$  dan node tujuan.

Pada kasus Word Ladder, GBFS tidak menjamin solusi optimal karena fokusnya hanya pada mencapai tujuan secepat mungkin tanpa memperhitungkan total cost perjalanan. GBFS mungkin mengabaikan rute yang lebih pendek karena lebih tertarik pada node yang tampaknya lebih dekat dengan target berdasarkan heuristik (terjebak pada kondisi lokal minimum).

## 2.3 Algoritma A\*

A\* adalah algoritma pencarian yang memilih node berikutnya berdasarkan jumlah cost kumulatif untuk mencapai node saat ini dan perkiraan cost untuk mencapai tujuan. Langkah-langkah algoritma yang digunakan dalam menyelesaikan permainan dengan algoritma A\* adalah sebagai berikut:

1. Inisialisasi: Mulai dengan node awal yaitu node dengan kata “Start Word” yang telah dimasukkan. Node ini dimasukkan ke dalam priority queue dengan cost awal 0

ditambah dengan heuristik berupa jumlah karakter yang berbeda antara kata awal dan kata tujuan untuk diproses ( $f(n) = g(n) + h(n)$ ).

2. Iterasi: Selama queue tidak kosong,

- Ambil node dengan nilai  $f(n)$  terendah dari queue.
- Periksa apakah node ini adalah tujuan (katanya adalah “End Word”). Jika ya, hentikan pencarian.
- Jika tidak, ekspansi semua tetangga dari node yang didapatkan dari Word Graph yang telah terdefinisi di awal dengan hubungan sisi berupa kata dengan perbedaan 1 huruf dengan kata saat ini.
- Untuk setiap tetangga, hitung kembali nilai heuristiknya ditambah dengan cost saat ini ( $f(n) = g(n) + h(n)$ ) dan masukkan ke dalam queue jika kata tersebut belum pernah dikunjungi.

Pada  $A^*$ , didefinisikan  $f(n)$  dan  $g(n)$  sebagai berikut.

- $g(n)$  adalah total cost dari node awal ke node  $n$ .
- $f(n) = g(n) + h(n)$ . Fungsi evaluasi  $f(n)$  menambahkan cost dari awal ke  $n$  ditambah dengan heuristik dari node  $n$  ke tujuan, yaitu jumlah karakter yang berbeda dari kata pada node  $n$  dan node tujuan.

Heuristik yang digunakan pada  $A^*$  adalah jarak Hamming, yaitu jumlah karakter yang berbeda dari kata saat ini dengan kata tujuan. Heuristik ini admissible, yaitu tidak estimasi cost ke tujuan tidak pernah lebih dari cost sebenarnya. Hal ini dikarenakan jarak Hamming adalah jumlah minimum langkah yang harus diambil untuk mengubah satu kata menjadi yang lain.

Pada kasus Word Ladder,  $A^*$  secara teoritis lebih efisien daripada UCS dikarenakan selain memperhitungkan cost sampai node saat ini,  $A^*$  juga memperhitungkan informasi heuristik untuk mengarahkan pencarian, yang dapat secara signifikan mengurangi jumlah node yang dijelajahi sebelum menemukan solusi.

## BAB 3

### IMPLEMENTASI

#### 3.1 Struktur Data

##### 3.1.1 WordGraph.java

Kelas WordGraph merepresentasikan graf dengan setiap node adalah kata dari kamus dan sisi menghubungkan kata yang memiliki perbedaan satu huruf.

```
1  import java.util.*;
2
3  public class WordGraph {
4      private Map<String, Set<String>> graph = new HashMap<>();
5
6      /* Add a word to the graph */
7      public void addWord(String word){
8          graph.putIfAbsent(word, new HashSet<String>());
9      }
10
11     /* Add an edge between two words */
12     public void addEdge(String word1, String word2){
13         graph.get(word1).add(word2);
14         graph.get(word2).add(word1);
15     }
16
17     /* Get the neighbors of a word */
18     public Set<String> getNeighbors(String word){
19         return graph.getOrDefault(word, Collections.emptySet());
20     }
21
22     /* Get all words in the graph */
23     public Set<String> getWords(){
24         return graph.keySet();
25     }
26 }
```

Method	Deskripsi
addWord	Menambahkan kata ke graf jika belum ada sebelumnya, dengan empty set sebagai value
addEdge	Menambahkan sisi antara dua kata di graf dengan menambahkan satu sama lain ke dalam set value
getNeighbors	Mengembalikan set kata yang merupakan tetangga dari kata yang diberikan
getWords	Mengembalikan semua kata pada graf yaitu semua kunci pada HashMap

### 3.1.2 Node.java

Kelas Node merepresentasikan node dalam graf yang sedang diproses dalam algoritma pencarian.

```

1  public class Node {
2      String word;
3      int cost;
4      int heuristic;
5      int priority;
6      int insertionOrder;
7      Node origin;
8
9      /* Constructor */
10     public Node(String word, int cost, int heuristic, int priority, int insertionOrder, Node origin){
11         this.word = word;
12         this.cost = cost;
13         this.heuristic = heuristic;
14         this.priority = priority;
15         this.insertionOrder = insertionOrder;
16         this.origin = origin;
17     }
18 }

```



### 3.1.3 PrioQueueFIFO.java

Kelas PrioQueueFIFO merepresentasikan antrian pemrosesan node berdasarkan priority dan insertionOrder sehingga mempertahankan FIFO dalam priority queue. Kelas ini menggunakan kelas PriorityQueue built-in dari Java dengan tambahan penyesuaian urutan.

```
1 import java.util.PriorityQueue;
2 import java.util.Comparator;
3
4 public class PrioQueueFIFO {
5     private PriorityQueue<Node> queue;
6     private int insertionOrder = 0;
7
8     /* Priority queue with FIFO ordering */
9     public PrioQueueFIFO() {
10         queue = new PriorityQueue<Node>(new Comparator<Node>() {
11             public int compare(Node n1, Node n2){
12                 int priorityComparison = Integer.compare(n1.priority, n2.priority);
13                 if (priorityComparison != 0){
14                     return priorityComparison;
15                 }
16                 return Integer.compare(n1.insertionOrder, n2.insertionOrder);
17             }
18         });
19     }
20
21     /* Get the current insertion order */
22     public int getCurrentInsertionOrder(){
23         return insertionOrder;
24     }
25
26     /* Add a node to the queue */
27     public void add(Node node){
28         queue.add(new Node(node.word, node.cost, node.heuristic, node.priority, insertionOrder++, node.origin));
29     }
30
31     /* Poll the node with the highest priority */
32     public Node poll(){
33         return queue.poll();
34     }
35
36     /* Check if the queue is empty */
37     public boolean isEmpty(){
38         return queue.isEmpty();
39     }
40 }
41
```

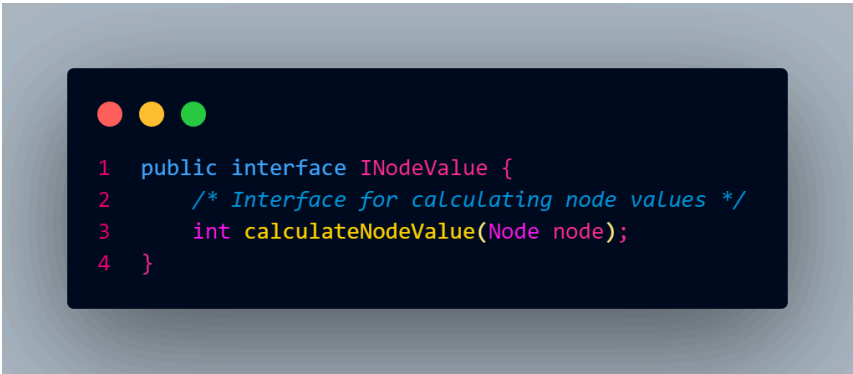
Method	Deskripsi
PrioQueueFIFO	Membuat PriorityQueue yang jika prioritynya sama, maka akan diperhatikan urutan dimasukkannya node (insertionOrder)
getCurrentInsertionOrder	Mengembalikan urutan pemasukan node saat ini
add	Menambahkan node ke queue

poll	Mengambil node di urutan pertama queue
isEmpty	Mengecek apakah queue kosong atau tidak

## 3.2 Utils

### 3.2.1 INodeValue.java

INodeValue adalah sebuah interface dengan method untuk menghitung nilai dari node.



```
1 public interface INodeValue {  
2     /* Interface for calculating node values */  
3     int calculateNodeValue(Node node);  
4 }
```

### 3.2.2 NodeValues.java

Kelas NodeValues berisi kelas untuk setiap algoritma yang mengimplementasikan method dari interface INodeValue ditambah method untuk menghitung nilai heuristik.

```
1  public class NodeValues {
2      /* Class for calculating node values based on the algorithm */
3      public static class UCSNodeValue implements INodeValue {
4          @Override
5          public int calculateNodeValue(Node node){
6              return node.cost;
7          }
8      }
9
10     public static class GBFSNodeValue implements INodeValue {
11         @Override
12         public int calculateNodeValue(Node node){
13             return node.heuristic;
14         }
15     }
16
17     public static class AStarNodeValue implements INodeValue {
18         @Override
19         public int calculateNodeValue(Node node){
20             return node.cost + node.heuristic;
21         }
22     }
23
24     public static int calculateHeuristic(String word1, String word2){
25         int diffs = 0;
26         for (int i = 0; i < word1.length(); i++){
27             if (word1.charAt(i) != word2.charAt(i)){
28                 diffs++;
29             }
30         }
31         return diffs;
32     }
33 }
34
```

## 3.3 Algoritma

### 3.3.1 Algorithm.java

Kelas Algorithm mengimplementasikan algoritma pencarian solusi Word Ladder yang akan diturunkan menjadi kelas UCS, BGFS, dan AStar.

```

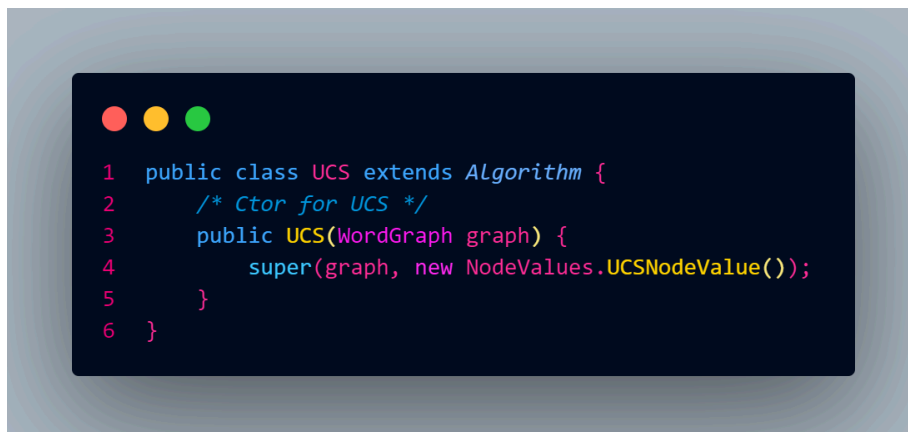
1 import java.util.Set;
2 import java.util.HashSet;
3 import java.util.List;
4 import java.util.ArrayList;
5 import java.util.Collections;
6
7 public abstract class Algorithm {
8     protected WordGraph graph;
9     protected PriorityQueue frontier;
10    protected Set<String> visited = new HashSet<String>();
11    protected INodeValue nodeValue;
12
13    /* Constructor */
14    public Algorithm(WordGraph graph, INodeValue nodeValue){
15        this.graph = graph;
16        this.nodeValue = nodeValue;
17        frontier = new PriorityQueue();
18    }
19
20    /* Setup the algorithm */
21    protected void startSetup(String startWord, String endWord){
22        int cost = 0;
23        int heuristic = NodeValues.calculateHeuristic(startWord, endWord);
24        int priority = nodeValue.calculateNodeValue(new Node(startWord, cost, heuristic, 0, 0, null));
25        Node startNode = new Node(startWord, cost, heuristic, priority, 0, null);
26        frontier.add(startNode);
27    }
28
29    /* Solve the algorithm */
30    public List<String> solve(String startWord, String endWord){
31        startSetup(startWord, endWord);
32        while (!frontier.isEmpty()){
33            Node current = frontier.poll();
34            markVisited(current.word);
35            if (current.word.equals(endWord)){
36                return buildPath(current);
37            }
38            exploreNeighbors(current, endWord);
39        }
40        return Collections.emptyList();
41    }
42
43    /* Explore the neighbors of the current node */
44    protected void exploreNeighbors(Node current, String endWord){
45        for (String neighbor : graph.getNeighbors(current.word)){
46            if (!hasVisited(neighbor)){
47                int insertionOrder = frontier.getCurrentInsertionOrder();
48                int newCost = current.cost + 1;
49                int newHeuristic = NodeValues.calculateHeuristic(neighbor, endWord);
50                int newPriority = nodeValue.calculateNodeValue(new Node(neighbor, newCost, newHeuristic, 0, insertionOrder, current));
51                Node newNode = new Node(neighbor, newCost, newHeuristic, newPriority, insertionOrder, current);
52                frontier.add(newNode);
53            }
54        }
55    }
56
57    /* Build the path from the end node to the start node */
58    protected List<String> buildPath(Node endNode){
59        List<String> path = new ArrayList<String>();
60        Node current = endNode;
61        while (current != null){
62            path.add(current.word);
63            current = current.origin;
64        }
65        Collections.reverse(path);
66        return path;
67    }
68
69    /* Check if a word has been visited */
70    private boolean hasVisited(String word){
71        return visited.contains(word);
72    }
73
74    /* Mark a word as visited */
75    private void markVisited(String word){
76        visited.add(word);
77    }
78
79    /* Get the number of visited nodes */
80    public int getVisitedNodeCount(){
81        return visited.size();
82    }
83 }
84

```

Method	Deskripsi
startSetup	Menginisialisasi algoritma dengan membuat node untuk start word dan memasukkannya ke prio queue
solve	Method utama untuk algoritma pencarian solusi yang menginisialisasi, memproses node di queue, mengekspansi ke tetangga, dan mengembalikan jalur solusi
exploreNeighbors	Iterasi pada semua tetangga dari node saat ini dan memasukkannya ke queue jika belum pernah dikunjungi
buildPath	Membangun path solusi dari end node ke start node menggunakan atribut origin dari Node
hasVisited	Mengecek apakah sebuah node telah dikunjungi
markVisited	Menandai sebuah node sudah dikunjungi
getVisitedNodeCount	Mengembalikan jumlah node yang dikunjungi selama pencarian

Ketiga algoritma kemudian diturunkan dari kelas ini dan menggunakan kalkulasi node values masing-masing.

### 3.3.2 UCS.java



```

1  public class UCS extends Algorithm {
2      /* Ctor for UCS */
3      public UCS(WordGraph graph) {
4          super(graph, new NodeValues.UCSNodeValue());
5      }
6  }

```

### 3.3.3 GBFS.java

```
1 public class GBFS extends Algorithm {
2     /* Ctor for GBFS */
3     public GBFS(WordGraph graph) {
4         super(graph, new NodeValues.GBFSNodeValue());
5     }
6 }
```

### 3.3.4 AStar.java

```
1 public class AStar extends Algorithm {
2     /* Ctor for AStar */
3     public AStar(WordGraph graph) {
4         super(graph, new NodeValues.AStarNodeValue());
5     }
6 }
```

## 3.4 Main

### 3.4.1 WordLadder.java

Kelas WordLadder berfungsi sebagai kelas utama dan bertindak sebagai game manager.

```

1  import java.io.*;
2  import java.util.List;
3
4  public class WordLadder {
5      private WordGraph wordGraph = new WordGraph();
6      private Algorithm algorithm;
7
8      /* Load words from a file */
9      public void loadWords(String filename) throws IOException {
10         BufferedReader reader = null;
11         try {
12             reader = new BufferedReader(new FileReader(filename));
13             String line;
14             while ((line = reader.readLine()) != null){
15                 wordGraph.addWord(line);
16             }
17         } finally {
18             if (reader != null){
19                 reader.close();
20             }
21         }
22         // Add edges between words that differ by one character
23         for (String word1 : wordGraph.getWords()){
24             for (String word2 : wordGraph.getWords()){
25                 if (differByOne(word1, word2)){
26                     wordGraph.addEdge(word1, word2);
27                 }
28             }
29         }
30     }
31
32     /* Check if two words differ by one character */
33     private boolean differByOne(String word1, String word2){
34         if (word1.length() != word2.length()){
35             return false;
36         }
37         int diffCount = 0;
38         for (int i = 0; i < word1.length(); i++){
39             if (word1.charAt(i) != word2.charAt(i)){
40                 diffCount++;
41             }
42         }
43         return diffCount == 1;
44     }
45
46     /* Get graph */
47     public WordGraph getGraph(){
48         return wordGraph;
49     }
50
51     /* Solve the Word Ladder */
52     public List<String> solve(String startWord, String endWord, int algorithmChoice){
53         switch (algorithmChoice){
54             case 1:
55                 algorithm = new UCS(wordGraph);
56                 break;
57             case 2:
58                 algorithm = new GBFS(wordGraph);
59                 break;
60             case 3:
61                 algorithm = new AStar(wordGraph);
62                 break;
63             default:
64                 throw new IllegalArgumentException("Invalid algorithm choice");
65         }
66
67         return algorithm.solve(startWord, endWord);
68     }
69
70     /* Get the number of visited nodes */
71     public int getVisitedNodes(){
72         return algorithm.getVisitedNodeCount();
73     }
74 }

```



Method	Deskripsi
loadWords	Memuat semua kata dari file dan membuat graf
differByOne	Mengecek apakah 2 kata memiliki perbedaan sebanyak 1 huruf saja
getGraph	Mengembalikan graf
solve	Memanggil instance algoritma berdasarkan parameter algoritma yang dimasukkan
getVisitedNodes	Mengembalikan banyak node yang dikunjungi selama pencarian solusi

### 3.4.2 Main.java

```

1 import java.util.Scanner;
2 import java.util.List;
3
4 public class Main {
5     public static void main(String[] args){
6         WordLadder wordLadder = new WordLadder();
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.print("Enter the length of the words you want to play (2-15 words): ");
10        int length = scanner.nextInt();
11        // word length validation
12        while (length < 2 || length > 15){
13            System.out.print("Invalid length. Please enter a length between 2 and 15: ");
14            length = scanner.nextInt();
15        }
16        // Load words from dictionary
17        try {
18            System.out.println("Loading dictionary...");
19            wordLadder.loadWords("src/data/dictionary_" + length + ".txt");
20        } catch (Exception e){
21            System.out.println("Error loading words: " + e.getMessage());
22            scanner.close();
23            return;
24        }
25        System.out.println("Loaded dictionary");
26
27        // input start and end word
28        System.out.print("Enter the start word: ");
29        String startWord = scanner.next();
30        // word length and existence validation
31        while (startWord.length() != length || !wordLadder.getGraph().getWords().contains(startWord)){
32            System.out.print("Invalid word. Please enter a word with " + length + " characters that is in the dictionary: ");
33            startWord = scanner.next();
34        }
35        System.out.print("Enter the end word: ");
36        String endWord = scanner.next();
37        // word length and existence validation
38        while (endWord.length() != length || !wordLadder.getGraph().getWords().contains(endWord)){
39            System.out.print("Invalid word. Please enter a word with " + length + " characters that is in the dictionary: ");
40            endWord = scanner.next();
41        }
42
43        // choose algorithm
44        System.out.println("Choose an algorithm:");
45        System.out.println("1. Uniform Cost Search");
46        System.out.println("2. Greedy Best First Search");
47        System.out.println("3. A* Search");
48        System.out.print("Enter the number of the algorithm you want to use: ");
49        int algorithmChoice = scanner.nextInt();
50
51        scanner.close();
52
53        // solve word ladder
54        long startTime = System.currentTimeMillis();
55        System.out.println("Solving...");
56        List<String> path = wordLadder.solve(startWord, endWord, algorithmChoice);
57        long endTime = System.currentTimeMillis();
58
59        // print results
60        if (path.isEmpty()){
61            System.out.println("No path found");
62        } else {
63            System.out.println("Path found: " + path);
64            System.out.println("Path length: " + (path.size() - 1) + " steps");
65            System.out.println("Number of nodes visited: " + wordLadder.getVisitedNodes());
66            System.out.println("Execution time: " + (endTime - startTime) + "ms");
67        }
68    }
69 }
70

```

## BAB 4

### EKSPERIMEN

Format masukan: string dengan panjang 2-15 karakter dan case insensitive

#### 4.1 Algoritma UCS

##### 4.1.1 Test Case 1 [TREE - ROOT]

```
PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 4
Loading dictionary...
Loaded dictionary
Enter the start word: TREE
Enter the end word: ROOT
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [TREE, TRET, TROT, TOOT, ROOT]
Path length: 4 steps
Number of nodes visited: 576
Execution time: 12ms
```

##### 4.1.2 Test Case 2 [MELON - PEARS]

```
PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 5
Loading dictionary...
Loaded dictionary
Enter the start word: MELON
Enter the end word: PEARS
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [MELON, MESON, MASON, MACON, RACON, RADON, REDON, REDOS, REDDS, READS, REARS, PEARS]
Path length: 11 steps
Number of nodes visited: 327
Execution time: 7ms
```

##### 4.1.3 Test Case 3 [BOYISH - LAUNCH]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 6
Loading dictionary...
Loaded dictionary
Enter the start word: BOYISH
Enter the end word: LAUNCH
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [BOYISH, TOYISH, TONISH, MONISH, MOPISH, POPISH, POLISH, PALISH, PARISH, PARIAN, PARTAN, TARTAN, TARTAR, TARTER, CARTER, CARTES, CARSES,
CORSES, HORSES, HOISES, HOISTS, JOISTS, JOINTS, POINTS, POINDS, POUNDS, FOUNDS, FOUNTS, COUNTS, COUNTY, BOUNTY, BOUNCY, JOUNCY, JOUNCE, JAUNCE, LAUNCE, LAU
NCH]
Path length: 37 steps
Number of nodes visited: 8831
Execution time: 246ms

```

#### 4.1.4 Test Case 4 [ATLASES - CABARET]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 7
Loading dictionary...
Loaded dictionary
Enter the start word: ATLASES
Enter the end word: CABARET
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [ATLASES, ANLASES, ANLACES, UNLACES, UNLADES, UNLADED, UNFADED, UNFAKED, UNCAKED, UNCAGED, UNCASES, UNEASES, UREASES, CREASES, CRESSES, CROSSES,
CROSSER, CRASSER, CRASHER, BRASHER, BRASIER, BRAKIER, BEAKIER, PEAKIER, PECKIER, PICKIER, DICKIER, DICKIES, HICKIES, HACKIES, HACKLES, HECKLES, DECKLES, DE
CILES, DEFILES, DEFILED, DEVELED, REVELED, RAVELED, RAVENED, HAVENED, HAVERED, WAVERED, WATERED, CATERED, CAPERED, TAPERED, TABERED, TABORED, TABOR
ET, TABARET, CABARET]
Path length: 52 steps
Number of nodes visited: 7972
Execution time: 5812ms

```

#### 4.1.5 Test Case 5 [SILLINESSES - WACKINESSES]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 11
Loading dictionary...
Loaded dictionary
Enter the start word: SILLINESSES
Enter the end word: WACKINESSES
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [SILLINESSES, SILKINESSES, SULKINESSES, BULKINESSES, BALKINESSES, TALKINESSES, TACKINESSES, WACKINESSES]
Path length: 7 steps
Number of nodes visited: 21
Execution time: 5ms

```

#### 4.1.6 Test Case 6 [DEVOLUTIONISTS - REVOLUTIONIZED]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 14
Loading dictionary...
Loaded dictionary
Enter the start word: DEVOLUTIONISTS
Enter the end word: REVOLUTIONIZED
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 1
Solving...
Path found: [DEVOLUTIONISTS, REVOLUTIONISTS, REVOLUTIONISES, REVOLUTIONIZES, REVOLUTIONIZED]
Path length: 4 steps
Number of nodes visited: 6
Execution time: 4ms

```

## 4.2 Algoritma BGFS

### 4.2.1 Test Case 1 [TREE - ROOT]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 4
Loading dictionary...
Loaded dictionary
Enter the start word: TREE
Enter the end word: ROOT
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [TREE, TRET, TROT, TOOT, ROOT]
Path length: 4 steps
Number of nodes visited: 5
Execution time: 5ms

```

### 4.2.2 Test Case 2 [MELON - PEARS]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 5
Loading dictionary...
Loaded dictionary
Enter the start word: MELON
Enter the end word: PEARS
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [MELON, MESON, MASON, MACON, RACON, RECON, REDON, REDOS, REDDS, READS, REARS, PEARS]
Path length: 11 steps
Number of nodes visited: 24
Execution time: 4ms

```

#### 4.2.3 Test Case 3 [BOYISH - LAUNCH]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 6
Loading dictionary...
Loaded dictionary
Enter the start word: BOYISH
Enter the end word: LAUNCH
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [BOYISH, TOYISH, TONISH, MONISH, MOYISH, POYISH, POLISH, PALISH, PARISH, PARIAS, PARIAN, PARTAN, TARTAN, TARTAR, TARTER, TAUTER, TAUTED, SAUTED, SAUCED, SAUCER, SAUGER, MAUGER, MAULER, MAULED, WAULED, WAUKED, JAUKEJ, JAUPED, YAUPED, YAUPER, PAUPER, PAUSER, PAUSES, CAUSES, CAULES, CAULDS, FAULDS, FAULTS, VAULTS, VAUNTS, TAUNTS, TAINTS, PAINTS, POINTS, POINDS, POUNDS, FOUNDS, FOUNTS, COUNTS, COUNTY, BOUNTY, BOUNCY, JOUNCY, JOUNCE, JAUNCE, LAUNCE, LAUNCH]
Path length: 56 steps
Number of nodes visited: 931
Execution time: 16ms

```

#### 4.2.4 Test Case 4 [ATLASES - CABARET]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 7
Loading dictionary...
Loaded dictionary
Enter the start word: ATLASES
Enter the end word: CABARET
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [ATLASES, ANLASES, ANLACES, UNLACES, UNLADES, UNLADED, UNFADED, UNFAKED, UNBAKED, UNBASED, UNCASSED, UNCASSED, UNEASES, UREASES, CREASES, CREASED, CREAKED, CROAKED, CLOAKED, CLONKED, CLINKED, CHINKED, CHIRKED, CHIRRED, CHARRED, CHARTED, CHATTED, CHATTER, CLATTER, PLATTER, PLATIER, PEATIER, PESTIER, PARTIER, PARTIER, PARRIER, BARRIER, BARKIER, BALKIER, TALKIER, TACKIER, TACKLER, CACKLER, CACKLES, HACKLES, HECKLES, DECKLES, DECILES, DEFILES, REFILES, REFIR ES, RETIRES, RETINES, RATINES, RAVINES, RAVINED, RAVENED, HAVENED, HAVERED, WAVERED, WATERED, CATERED, CAPERED, TAPERED, TABERED, TABORED, TABORET, TABARET, CABARET]
Path length: 68 steps
Number of nodes visited: 4739
Execution time: 102ms

```

#### 4.2.5 Test Case 5 [SILLINESSES - WACKINESSES]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 11
Loading dictionary...
Loaded dictionary
Enter the start word: SILLINESSES
Enter the end word: WACKINESSES
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [SILLINESSES, SILKINESSES, SULKINESSES, BULKINESSES, BALKINESSES, TALKINESSES, TACKINESSES, WACKINESSES]
Path length: 7 steps
Number of nodes visited: 9
Execution time: 5ms

```

#### 4.2.6 Test Case 6 [DEVOLUTIONISTS - REVOLUTIONIZED]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 14
Loading dictionary...
Loaded dictionary
Enter the start word: DEVOLUTIONISTS
Enter the end word: REVOLUTIONIZED
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 2
Solving...
Path found: [DEVOLUTIONISTS, REVOLUTIONISTS, REVOLUTIONISES, REVOLUTIONIZES, REVOLUTIONIZED]
Path length: 4 steps
Number of nodes visited: 5
Execution time: 4ms

```

### 4.3 Algoritma A\*

#### 4.3.1 Test Case 1 [TREE - ROOT]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 4
Loading dictionary...
Loaded dictionary
Enter the start word: TREE
Enter the end word: ROOT
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [TREE, TRET, TROT, TOOT, ROOT]
Path length: 4 steps
Number of nodes visited: 5
Execution time: 7ms

```

#### 4.3.2 Test Case 2 [MELON - PEARS]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 5
Loading dictionary...
Loaded dictionary
Enter the start word: MELON
Enter the end word: PEARS
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [MELON, MESON, MASON, MACON, RACON, RECON, REDON, REDOS, REDDS, READS, REARS, PEARS]
Path length: 11 steps
Number of nodes visited: 38
Execution time: 4ms

```

#### 4.3.3 Test Case 3 [BOYISH - LAUNCH]

```

PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 6
Loading dictionary...
Loaded dictionary
Enter the start word: BOYISH
Enter the end word: LAUNCH
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [BOYISH, TOYISH, TONISH, MONISH, MOYISH, POYISH, POLISH, PALISH, PARISH, PARIAS, PARIAN, PARTAN, TARTAN, TARTAR, TARTER, TAUTER, TOUTER, ROUTER,
ROUTES, ROUSES, ROUSTS, JOUSTS, JOISTS, JOINTS, POINTS, POINDS, POUNDS, FOUNDS, FOUNTS, COUNTS, COUNTY, BOUNTY, BOUNCY, JOUNCY, JOUNCE, JAUNCE, LAUNCE, LAU
NCH]
Path length: 37 steps
Number of nodes visited: 8767
Execution time: 252ms

```



#### 4.3.4 Test Case 4 [ATLASES - CABARET]

```
PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 7
Loading dictionary...
Loaded dictionary
Enter the start word: ATLASES
Enter the end word: CABARET
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [ATLASES, ANLASES, ANLACES, UNLACES, UNLADES, UNLADED, UNFADED, UNFAKED, UNCAKED, UNCASSED, UNCASES, UNEASES, UREASES, CREASES, CRESSES, CROSSES,
CROSSER, CROSIER, CROZIER, CRAZIER, BRAZIER, BRAKIER, BEAKIER, PEAKIER, PECKIER, PICKIER, DICKIER, DICKIES, HICKIES, HACKIES, HACKLES, HECKLES, DECKLES, DE
CILES, DEFILES, DEFILED, DEVELED, REVELED, RAVELED, RAVENED, HAVENED, HAVERED, WAVERED, WATERED, CATERED, CAPERED, TAPERED, TABERED, TABORED, TABOR
ET, TABARET, CABARET]
Path length: 52 steps
Number of nodes visited: 7922
Execution time: 2112ms
```

#### 4.3.5 Test Case 5 [SILLINESSES - WACKINESSES]

```
PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 11
Loading dictionary...
Loaded dictionary
Enter the start word: SILLINESS
Invalid word. Please enter a word with 11 characters that is in the dictionary: SILLINESSES
Enter the end word: WACKINESSES
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [SILLINESSES, SILKINESSES, SULKINESSES, BULKINESSES, BALKINESSES, TALKINESSES, TACKINESSES, WACKINESSES]
Path length: 7 steps
Number of nodes visited: 10
Execution time: 4ms
```

#### 4.3.6 Test Case 6 [DEVOLUTIONISTS - REVOLUTIONIZED]

```
PS C:\Users\Lenovo\Project\Tucil3_13522096> ./run.bat
=====
Welcome to the Word Ladder Game Solver!
=====
Enter the length of the words you want to play (2-15 words): 14
Loading dictionary...
Loaded dictionary
Enter the start word: DEVOLUTIONISTS
Enter the end word: REVOLUTIONIZED
Choose an algorithm:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Enter the number of the algorithm you want to use: 3
Solving...
Path found: [DEVOLUTIONISTS, REVOLUTIONISTS, REVOLUTIONISES, REVOLUTIONIZES, REVOLUTIONIZED]
Path length: 4 steps
Number of nodes visited: 6
Execution time: 3ms
```

## BAB 5

### ANALISIS PERBANDINGAN

TC	Panjang Solusi			Node yang Dikunjungi			Waktu Eksekusi (ms)		
	UCS	GBFS	A*	UCS	GBFS	A*	UCS	GBFS	A*
1	4	4	4	576	5	5	12	5	7
2	11	11	11	327	24	38	7	4	4
3	37	56	37	8831	931	8767	246	16	252
4	52	68	52	7972	4739	7922	5812	102	2112
5	7	7	7	21	9	10	5	5	4
6	4	4	4	6	5	6	4	4	3

Berdasarkan tabel di atas, dapat dilakukan analisis komparatif untuk algoritma Uniform Cost Search (UCS), Greedy Best First Search (GBFS), dan A\*. Analisis ini mencakup efisiensi, optimalitas, dan penggunaan memori berdasarkan panjang solusi, jumlah node yang dikunjungi, dan waktu eksekusi.

#### 5.1 Optimalitas

- UCS dan A\* konsisten menghasilkan solusi optimal, seperti terlihat dari kolom "Panjang Solusi" dimana mereka selalu memiliki panjang solusi terpendek atau sama dengan yang terbaik yang bisa ditemukan.
- GBFS tidak selalu menemukan solusi optimal. Pada kasus test 3 dan 4, GBFS menemukan solusi yang lebih panjang dibandingkan dengan UCS dan A\*, dikarenakan GBFS yang tidak memperhatikan cost total dan lebih mengutamakan pencapaian tujuan.

#### 5.2 Efisiensi Waktu

- A\* secara umum menunjukkan waktu eksekusi yang efisien di hampir semua kasus dibandingkan dengan UCS, terutama dalam kasus yang kompleks seperti TC3 dan TC4. Ini menunjukkan efektivitas heuristiknya dalam mengurangi ruang pencarian.

- GBFS menunjukkan waktu eksekusi yang sangat cepat di semua kasus karena kurangnya beban penghitungan cost dan fokus langsung menuju tujuan. Namun, efisiensi waktu ini tidak diimbangi dengan optimalitas solusi.
- UCS menunjukkan waktu eksekusi yang lebih lama, terutama dalam kasus yang lebih kompleks. Hal ini karena UCS melakukan eksplorasi yang sangat luas tanpa mempertimbangkan jarak heuristic ke tujuan.

### **5.3 Penggunaan Memori (Jumlah Node yang Dikunjungi)**

- UCS mengunjungi jumlah node yang lebih banyak dibandingkan dengan dua algoritma lainnya, terlihat jelas pada TC3 dan TC4, yang mengindikasikan penggunaan memori yang tinggi dan eksplorasi yang sangat luas tanpa panduan heuristic.
- A\*, meskipun lebih sedikit dari UCS dalam hal jumlah node yang dikunjungi, masih relatif tinggi, yang mencerminkan keseimbangan antara eksplorasi dan penggunaan heuristic.
- GBFS memiliki jumlah node yang dikunjungi paling rendah, yang menunjukkan penggunaan memori yang sangat efisien tetapi dengan kurangnya jaminan untuk menemukan solusi yang optimal.

### **5.4 Kesimpulan**

- A\* adalah algoritma yang paling seimbang, memberikan solusi optimal dengan efisiensi waktu dan penggunaan memori yang relatif baik sehingga A\* menjadi pilihan yang cocok untuk aplikasi di mana optimalitas dan efisiensi sama-sama penting.
- GBFS cepat dan efisien dari segi memori, tetapi kurang dapat diandalkan untuk menemukan solusi optimal dan sebaiknya hanya digunakan dalam kasus di mana waktu respons adalah prioritas utama.
- UCS menjamin solusi optimal tetapi dengan biaya waktu eksekusi yang lama dan penggunaan memori yang besar, yang menjadikannya kurang ideal untuk kasus dengan ruang pencarian yang besar atau sumber daya terbatas.

## BAB 6

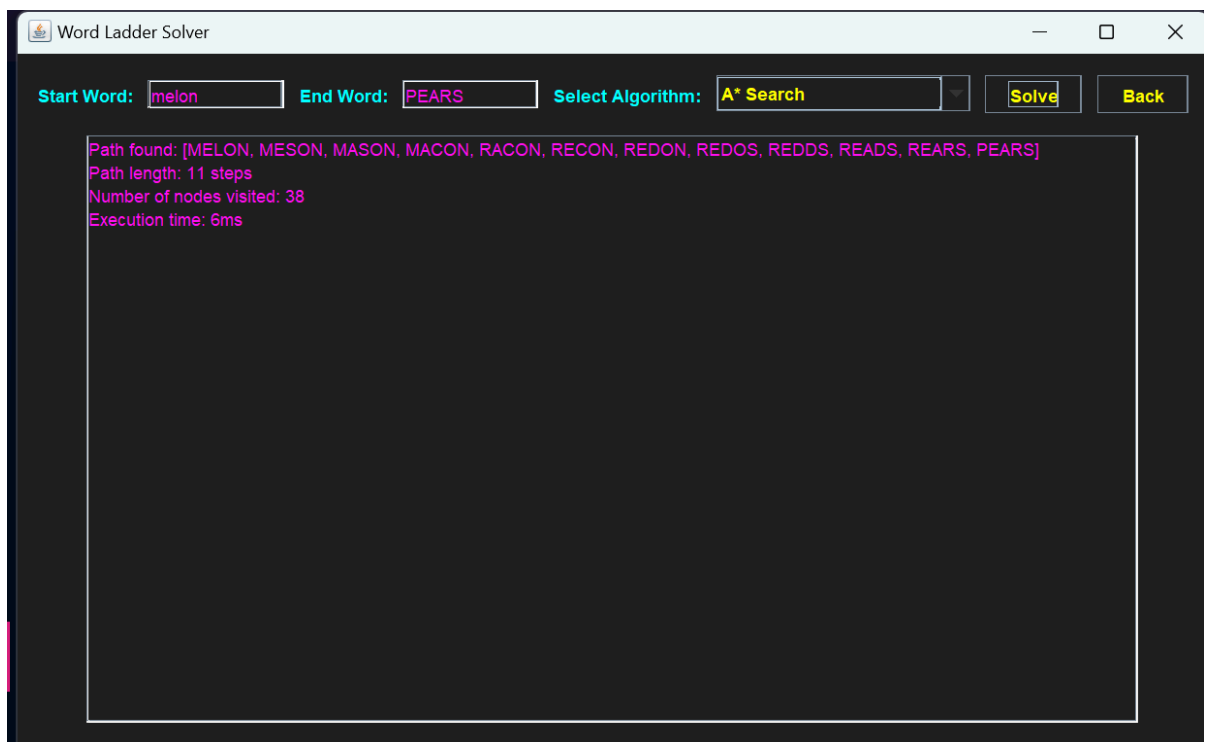
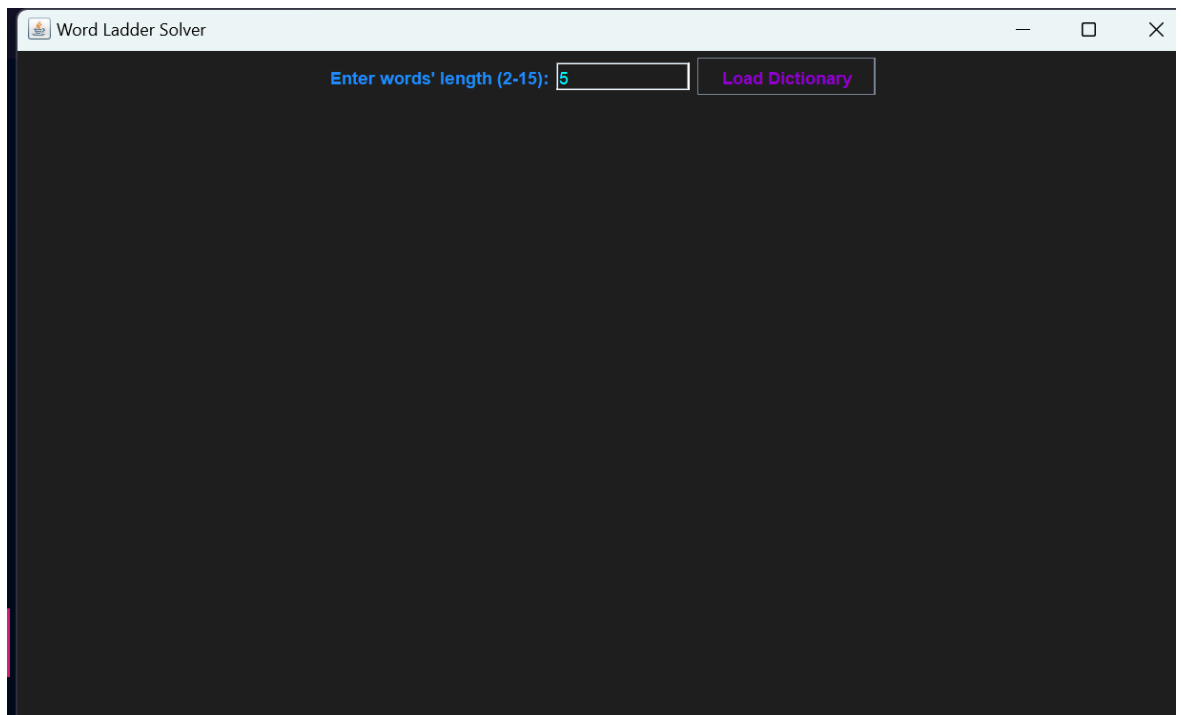
### IMPLEMENTASI BONUS

Pada Word Ladder Solver ini, program dapat berjalan dengan Graphical User Interface (GUI). Kakas yang digunakan adalah Java Swing.

#### 6.1 Source Code

Kelas & Method	Deskripsi
MainGUI	Kelas utama yang mengatur tampilan dan perilaku GUI
initUI	Method yang mengatur tampilan dan perilaku dari komponen GUI, seperti menciptakan dan mengatur panel, tombol, dan bidang teks
loadDictionary	Method yang dipanggil ketika tombol "Load Dictionary" ditekan
solveAction	Method yang dipanggil ketika tombol "Solve" ditekan
toggleInputs	Method yang mengaktifkan atau menonaktifkan bidang input dan tombol "Solve" berdasarkan apakah kamus telah dimuat atau tidak
createStyledPanel, createStyledLabel, createStyledTextField, createStyledButton, styleComboBox, styleTextArea	Methods untuk membuat dan mengatur komponen GUI

## 6.2 Tampilan



## BAB 7

### LAMPIRAN

#### 7.1 Link Repository

Link repository berisi kode program untuk tugas kecil 3 mata kuliah IF2211 Strategi Algoritma adalah sebagai berikut:

Link : [https://github.com/novelxv/Tucil3\\_13522096](https://github.com/novelxv/Tucil3_13522096)

#### 7.2 Tabel Checkpoint Program

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