

**Tugas Kecil 3 IF2211 Strategi Algoritma**  
**Penyelesaian Permainan Word Ladder Menggunakan Algoritma**  
**UCS, Greedy Best First Search, dan A\***  
**Semester II Tahun 2023/2024**



Disusun oleh

Muhammad Dava Fathurrahman

13522114

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2024**

## Daftar Isi

<b>Daftar Isi.....</b>	<b>2</b>
<b>I. Deskripsi Tugas.....</b>	<b>3</b>
<b>II. Implementasi Algoritma.....</b>	<b>4</b>
A. Uniform Cost Search.....	4
B. Greedy Best First Search.....	4
C. A* Search (A Star Search).....	5
<b>III. Kode Sumber.....</b>	<b>5</b>
A. Kelas Node.....	5
B. Kelas ExtendedNode.....	6
C. Kelas App.....	7
D. Kelas UCS.....	11
E. Kelas GBFS.....	12
F. Kelas AStar.....	14
<b>IV. Hasil Pengujian.....</b>	<b>15</b>
<b>V. Analisis Hasil Uji.....</b>	<b>28</b>
<b>VI. Implementasi Bonus.....</b>	<b>29</b>
<b>VII. Lampiran.....</b>	<b>30</b>

## I. Deskripsi Tugas

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

### How To Play

This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

**Rules**

Weave your way from the start word to the end word.

Each word you enter **can only change 1 letter** from the word above it.

**Example**

E A S T

EAST is the start word, WEST is the end word

V A S T

We changed E to V to make VAST

V E S T

We changed A to E to make VEST

W E S T

And we changed V to W to make WEST

W E S T

Done!

**Privacy**

[Privacy Policy](#)

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

Tentu akan sangat banyak rantai kata dari start word menuju end word sehingga memerlukan waktu yang lebih banyak. Oleh karena itu, Anda diminta untuk membuat sebuah solver permainan tersebut dengan harapan menemukan solusi paling optimal untuk menyelesaikan permainan Word Ladder ini.

## II. Implementasi Algoritma

### A. Uniform Cost Search

Uniform Cost Search (UCS) adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) kumulatif terendah. Ongkos kumulatif simpul  $n$  didefinisikan sebagai  $g(n)$ , yaitu jarak dari akar ke simpul  $n$ .

Implementasi algoritma UCS pada penyelesaian permainan Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue dengan harga nol.
3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi, dengan harga kumulatif ditambah satu.
6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

### B. Greedy Best First Search

Greedy Best First Search (GBFS) adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) terendah. Ongkos simpul  $n$  didefinisikan sebagai  $h(n)$ , yaitu banyaknya huruf yang berbeda dengan huruf pada kata tujuan pada posisi yang sama.

Implementasi algoritma UCS pada penyelesaian permainan Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue.

3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi.
6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

### C. A\* Search (A Star Search)

A\* Search adalah metode penelusuran graf yang mempertimbangkan biaya atau cost yang terkait dengan simpul awal. Algoritma ini memulai pencarian dari root node, kemudian dilanjutkan ke node-node selanjutnya. Simpul yang diekspansi merupakan simpul dengan harga (cost) kumulatif terendah ditambah estimasi harga dari simpul n ke simpul tujuan. Ongkos kumulatif simpul n didefinisikan sebagai  $f(n) = g(n) + h(n)$ , yaitu jarak dari akar ke simpul n.

Implementasi algoritma A\* Search pada penyelesaian permainan Word Ladder adalah sebagai berikut,

1. Masukan start word akan dicek apakah tersedia pada kamus lokal. Jika tidak, pesan pemberitahuan akan ditampilkan pada layar.
2. Start word dimasukkan ke dalam priority queue dengan harga nol.
3. Jika priority queue tidak kosong, elemen terdepan priority queue (dijamin terurut mulai dari harga terendah) dikeluarkan
4. Jika simpul yang dikeluarkan adalah simpul tujuan, proses dihentikan. Jika tidak, lanjut ke langkah ke-5 untuk ekspansi simpul.
5. Simpul dimasukkan ke dalam senarai visited dan diekspansi ke simpul tetangganya yang belum pernah dikunjungi, dengan harga kumulatif ditambah satu.
6. Seluruh simpul ekspansi dimasukkan ke priority queue dan kembali ke langkah ke-3.

## III. Kode Sumber

### A. Kelas Node

Kelas Node merepresentasikan simpul dalam struktur pohon dengan atribut kata (word) dan simpul induk (parent), serta memiliki metode untuk mengakses kata, mendapatkan simpul induk, dan mendapatkan jalur dari akar ke simpul saat ini dalam bentuk string. Kelas Node digunakan pada metode pencarian Greedy Best

First Search (GBFS) karena simpul tidak perlu menyimpan informasi harga kumulatif.

```
public class Node {
    protected String word;
    protected Node parent;

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

    public String getWord(){
        return this.word;
    }

    public Node getParent(){
        return this.parent;
    }

    public String get_path_from_root(){
        if (this.parent == null){
            return this.word;
        }

        return this.parent.get_path_from_root() + " " + this.word;
    }
}
```

#### B. Kelas ExtendedNode

Kelas ExtendedNode merupakan turunan dari kelas Node yang menambahkan atribut jarak (distance) dan memiliki metode untuk mengakses jarak serta mendapatkan jalur dari akar ke simpul saat ini dalam bentuk string. Kelas ini digunakan pada metode pencarian UCS dan A\* karena simpul perlu menyimpan informasi harga kumulatif.

```
public class ExtendedNode extends Node{
    private int distance;

    public ExtendedNode(String word, Node parent, int distance){
        super(word, parent);
        this.distance = distance;
    }

    public int get_distance(){
```

```

        return this.distance;
    }

    @Override
    public String get_path_from_root(){
        if (this.parent == null){
            return this.word;
        }

        return this.parent.get_path_from_root() + " " + this.word;
    }
}

```

### C. Kelas App

Kelas App merupakan kelas titik masuk program yang terdiri atas konstruktor pembangun antarmuka pengguna grafis, metode memuat fail kamus, metode mencari simpul tetangga. Berikut adalah potongan kode sumber yang sudah direduksi untuk meningkatkan keterbacaan.

```

public class App extends JFrame {
    public static Set<String> dictionary;

    private JTextField startWordField;
    private JTextField endWordField;
    private JComboBox<String> algorithmComboBox;
    private JTextPane resultPane; // Text area to display ladder
    information

    public App() {
        setTitle("Word Ladder Solver");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(450, 400);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.insets = new Insets(5, 5, 5, 5);

        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String startWord =
startWordField.getText().toUpperCase();
                String endWord = endWordField.getText().toUpperCase();
            }
        });
    }
}

```

```

        String searchAlgorithm = (String)
algorithmComboBox.getSelectedItem();

        // Get dictionary words with same length from the file
and store them in a set
        loadDictionary("src/dictionary.txt", startWord);

        // Perform the search algorithm and update the result
area
        performSearch(startWord, endWord, searchAlgorithm);
    }
});
}

private void performSearch(String startWord, String endWord, String
searchAlgorithm) {
    StringBuilder result = new StringBuilder();

    // Check for valid input and dictionary initialization
    if (!isValidInput(startWord, endWord)) {
        resultPane.setText("Invalid input or dictionary not
initialized.");
        return;
    }

    // Check if the start word and the end word are in the dictionary
    if (!dictionary.contains(startWord) ||
!dictionary.contains(endWord)) {
        resultPane.setText("Start word or end word not in the
dictionary.");
        return;
    }

    long startTime = System.currentTimeMillis();
    long endTime;
    long executionTime;

    App.Pair<Integer, List<String>> wordLadder = null;

    switch (searchAlgorithm) {
        case "Uniform Cost Search (UCS)":
            UCS ucs = new UCS(endWord);
            ExtendedNode startUCSNode = new ExtendedNode(startWord,
null, 0);
            wordLadder = ucs.search(startUCSNode);

            break;
        case "Greedy Best First Search (GBFS)":
            GBFS gbfs = new GBFS(endWord);

```



```

        Node startGFBSNode = new Node(startWord, null);
        wordLadder = gbfs.search(startGFBSNode);

        break;
    case "A* Search":
        AStar astar = new AStar(endWord);
        ExtendedNode startAStarNode = new ExtendedNode(startWord,
null, 0);

        wordLadder = astar.search(startAStarNode);

        break;
    }

    endTime = System.currentTimeMillis();

    if (!wordLadder.getSecond().isEmpty()){
        int size = wordLadder.getSecond().size();
        for (int i = 0; i < size; i++) {
            String word = wordLadder.getSecond().get(i);
            if (i == size - 1) {
                result.append("<font color='green'>[" + i + "]" + " " +
word + "</font>");
            }
            else if (i == 0) {
                result.append("<font color='blue'>[" + i + "]" + " " +
word + "</font>");
            }
            else {
                result.append(word);
            }
            result.append("<br>");
        }
    }
    else{
        result.append("No ladder found.");
    }

    executionTime = endTime - startTime;
    result.append("<br>" + wordLadder.getFirst() + " node has been
visited in " + executionTime + " ms.");

    resultPane.setText(result.toString());
}

private boolean isValidInput(String startWord, String endWord) {
    if (!startWord.matches("[A-Z]+") || !endWord.matches("[A-Z]+")) {
        return false; // Alphabetic check failed
    }
}

```

```

        if (dictionary == null) {
            return false; // Dictionary not initialized
        }

        if (startWord.length() != endWord.length()) {
            return false; // Word length mismatch
        }

        return true;
    }

    private static void loadDictionary(String filePath, String startWord)
    {
        dictionary = new HashSet<>();
        try {
            File file = new File(filePath);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String word = scanner.nextLine();
                if (word.length() == startWord.length()) {
                    dictionary.add(word.toUpperCase());
                }
            }
            scanner.close();
        }
        catch (FileNotFoundException e) {}
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            App app = new App(); // Create an instance of the application
            app.setVisible(true);
        });
    }

    public static List<String> getNeighbor(String word) {
        List<String> nextWords = new ArrayList<>();
        for (int i = 0; i < word.length(); i++) {
            char[] chars = word.toCharArray();
            for (char c = 'A'; c <= 'Z'; c++) {
                chars[i] = c;
                String newWord = new String(chars);
                if (!newWord.equals(word) &&
dictionary.contains(newWord)) {
                    nextWords.add(newWord);
                }
            }
        }
    }
}

```

```

        return nextWords;
    }
}

```

#### D. Kelas UCS

Kelas UCS adalah implementasi algoritma Uniform Cost Search (UCS) yang menggunakan struktur data priority queue untuk mengelola simpul yang akan dieksplorasi berdasarkan prioritas jarak terpendek. Metode search menerima simpul awal dan mencari jalur terpendek dari simpul awal ke simpul tujuan (endWord). Selama pencarian, simpul yang akan dieksplorasi disimpan dalam frontier, sementara simpul yang telah dieksplorasi disimpan dalam visited. Algoritma berhenti ketika simpul tujuan ditemukan atau semua simpul yang dapat dieksplorasi telah dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```

public class UCS {
    private String endWord;
    private PriorityQueue<ExtendedNode> frontier;
    private Set<String> visited;

    public UCS(String endWord) {
        this.endWord = endWord;
        this.frontier = new
PriorityQueue<>(Comparator.comparingInt((currentNode) ->
currentNode.get_distance()));
        this.visited = new HashSet<>();
    }

    public App.Pair<Integer, List<String>> search(ExtendedNode startNode)
{
        if (startNode == null) {
            return new App.Pair<>(visited.size(),
Collections.emptyList());
        }

        frontier.offer(startNode);

        while (!frontier.isEmpty()) {
            ExtendedNode currentNode = frontier.poll();

            if (currentNode.getWord().equals(endWord)) {

```

```

        return new App.Pair<>(visited.size()+1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
    }

    visited.add(currentNode.getWord());

    for (String nextWord :
App.getNeighbor(currentNode.getWord())) {
        if (!visited.contains(nextWord)) {
            int newCost = currentNode.get_distance() + 1;
            ExtendedNode nextNode = new ExtendedNode(nextWord,
currentNode, newCost);
            frontier.offer(nextNode);
        }
    }

    return new App.Pair<>(visited.size(), Collections.emptyList());
}

```

#### E. Kelas GBFS

Kelas GBFS (Greedy Best-First Search) merupakan implementasi dari algoritma pencarian heuristik yang menggunakan pendekatan greedy untuk mencari jalur terpendek dari simpul awal ke simpul tujuan (endWord). Dalam kelas ini, sebuah priority queue digunakan untuk mengelola simpul yang akan dieksplorasi berdasarkan heuristik, yaitu jumlah karakter yang tidak sesuai berdasarkan posisinya antara kata saat ini dan kata tujuan. Metode search menerima simpul awal dan mencari jalur terpendek ke simpul tujuan. Algoritma berhenti saat simpul tujuan ditemukan atau tidak ada lagi simpul yang dapat dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```

import java.util.*;

public class GBFS {
    private String endWord;
    private PriorityQueue<Node> prioQueue;
    private Set<String> visited;

    public GBFS(String endWord) {
        this.endWord = endWord;
        this.prioQueue = new
PriorityQueue<>(Comparator.comparingInt((currentNode) -> {

```

```

        String a = currentNode.getWord();
        int missChar = 0;
        for (int i = 0; i < a.length(); i++) {
            if (a.charAt(i) != endWord.charAt(i)) {
                missChar++;
            }
        }

        return missChar;
    }));
    this.visited = new HashSet<>();
}

public App.Pair<Integer, List<String>> search(Node startNode) {
    if (startNode == null) {
        return new App.Pair<>(visited.size(),
Collections.emptyList());
    }

    // Add the start word to the prioQueue
    prioQueue.offer(startNode);
    visited.add(startNode.getWord());

    while (!prioQueue.isEmpty()) {
        Node currentNode = prioQueue.poll();
        String currentWord = currentNode.getWord();

        if (currentWord.equals(endWord)) {
            return new App.Pair<>(visited.size() + 1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
        }

        visited.add(currentNode.getWord());

        for (String nextWord : App.getNeighbor(currentWord)) {
            if (!visited.contains(nextWord)) {
                Node nextNode = new Node(nextWord, currentNode);
                prioQueue.offer(nextNode);
            }
        }
    }

    return new App.Pair<>(visited.size(), Collections.emptyList());
}
}

```

## F. Kelas AStar

Kelas AStar adalah implementasi dari algoritma A\* yang merupakan kombinasi dari algoritma Uniform Cost Search (UCS) dan Greedy Best-First Search. Priority queue digunakan untuk mengelola simpul yang akan dieksplorasi berdasarkan nilai heuristik. Metode search menerima simpul awal dan mencari jalur terpendek ke simpul tujuan. Algoritma berhenti saat simpul tujuan ditemukan atau tidak ada lagi simpul yang dapat dieksplorasi. Metode search mengembalikan pasangan nilai yang berisi jumlah simpul yang telah dieksplorasi dan jalur terpendek dari simpul awal ke simpul tujuan jika ditemukan, atau jumlah simpul yang telah dieksplorasi dan daftar kosong jika jalur tidak ditemukan.

```
public class AStar {
    private String endWord;
    private PriorityQueue<ExtendedNode> prioQueue;
    private Set<String> visited;

    public AStar(String endWord) {
        this.endWord = endWord;
        this.prioQueue = new
PriorityQueue<>(Comparator.comparingInt((currentNode) -> {
        int missChar = 0;
        String word = currentNode.getWord();
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) != endWord.charAt(i)) {
                missChar++;
            }
        }

        return missChar + currentNode.get_distance();
    }));
        this.visited = new HashSet<>();
    }

    public App.Pair<Integer, List<String>> search(ExtendedNode startNode)
    {
        if (startNode == null) {
            return new App.Pair<>(visited.size(),
Collections.emptyList());
        }

        prioQueue.offer(startNode);

        while (!prioQueue.isEmpty()) {
            ExtendedNode currentNode = prioQueue.poll();

            if (currentNode.getWord().equals(endWord)) {
```

```

        return new App.Pair<>(visited.size()+1,
Arrays.asList(currentNode.get_path_from_root().split(" ")));
    }

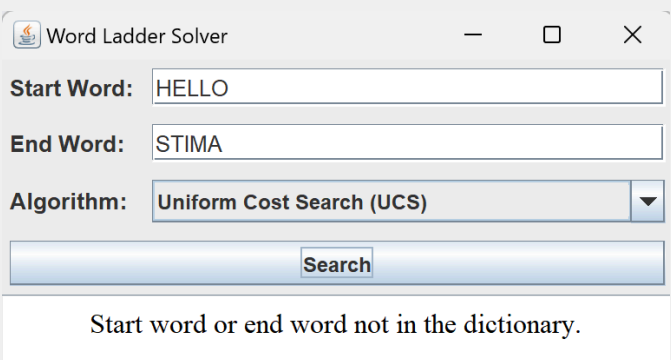
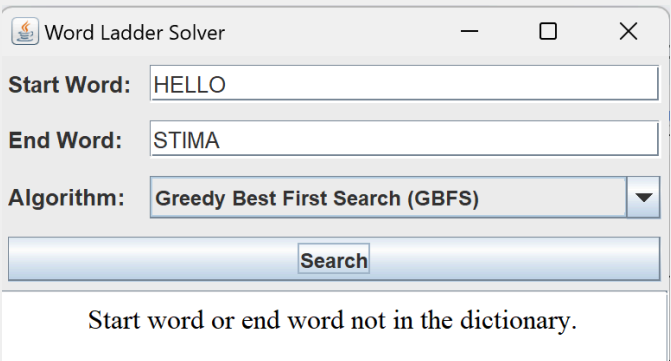
    visited.add(currentNode.getWord());

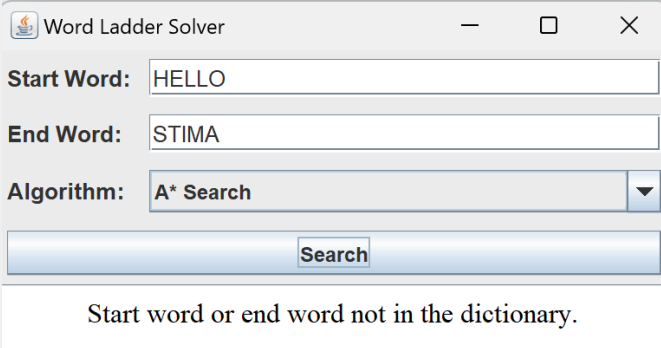
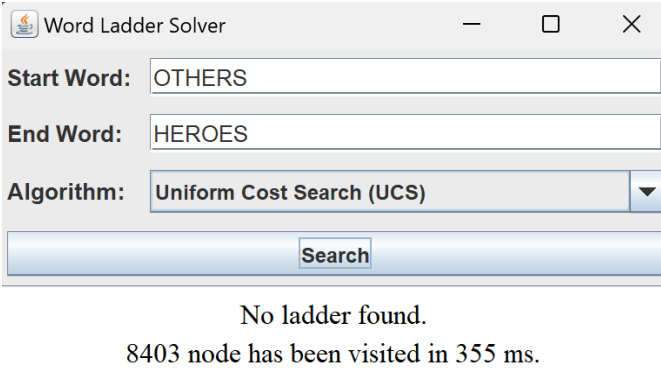
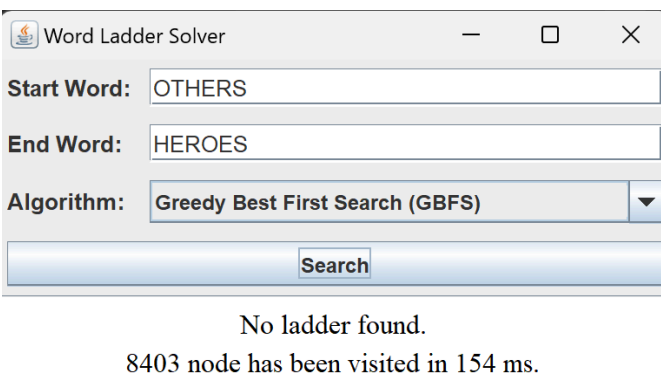
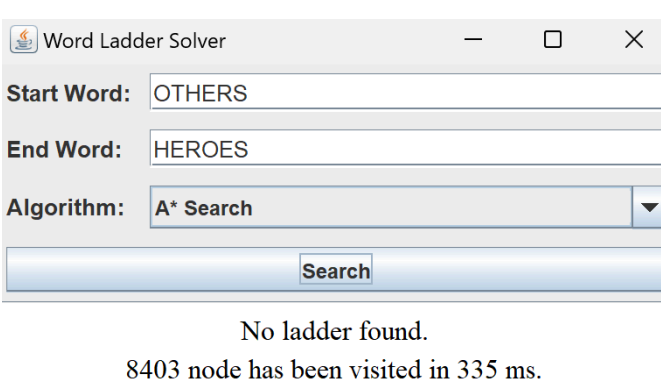
    for (String nextWord :
App.getNeighbor(currentNode.getWord())) {
        if (!visited.contains(nextWord)) {
            int newCost = currentNode.get_distance() + 1;
            ExtendedNode nextNode = new ExtendedNode(nextWord,
currentNode, newCost);
            prioQueue.offer(nextNode);
        }
    }
}

return new App.Pair<>(visited.size(), Collections.emptyList());
}
}

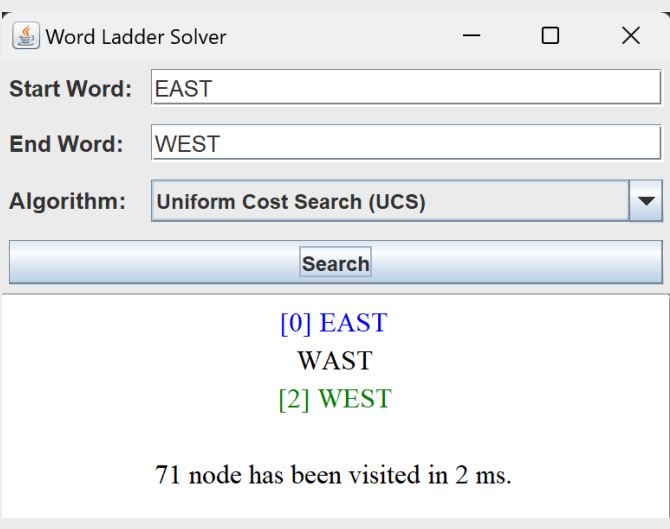
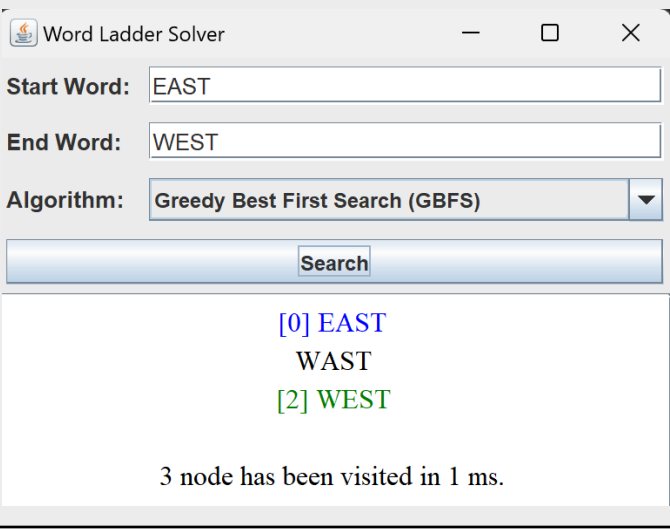
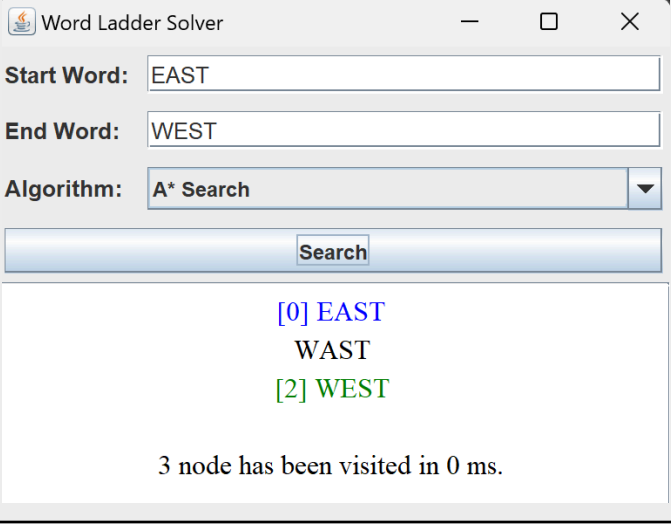
```

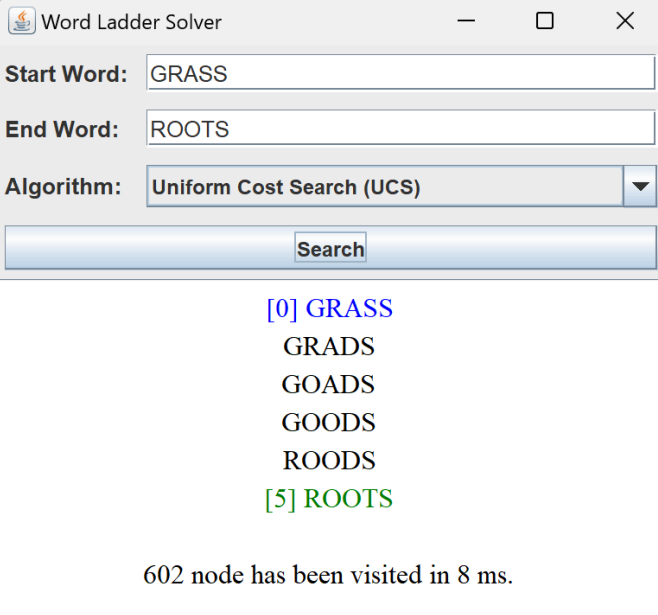
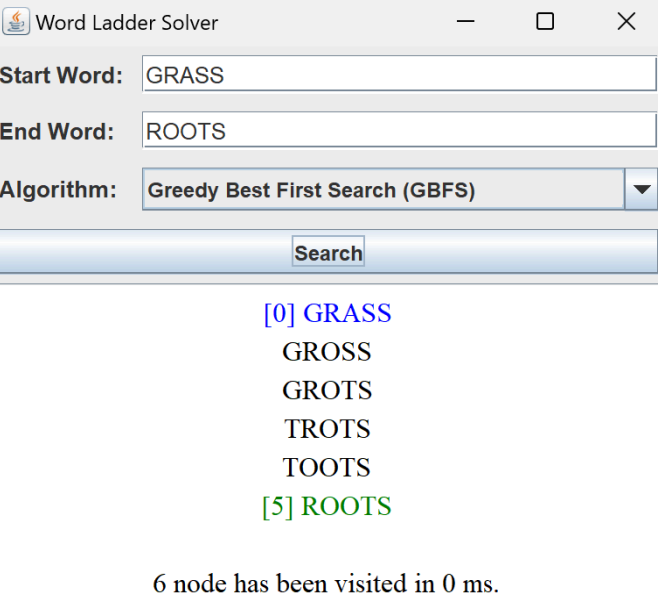
#### IV. Hasil Pengujian

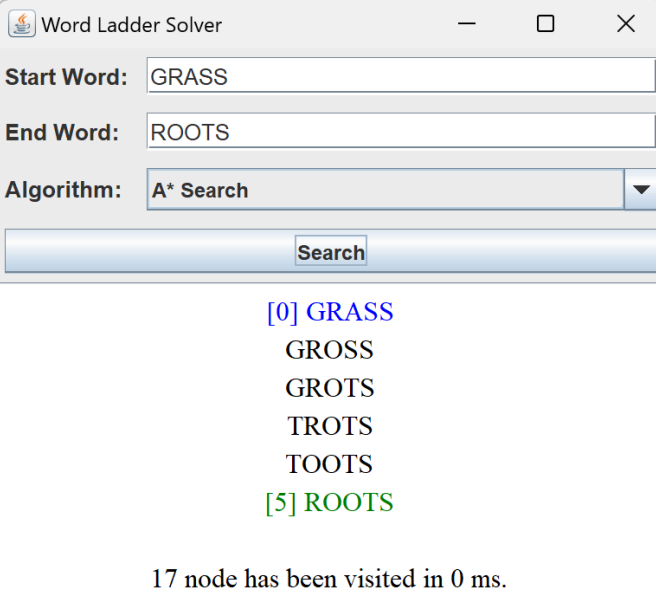
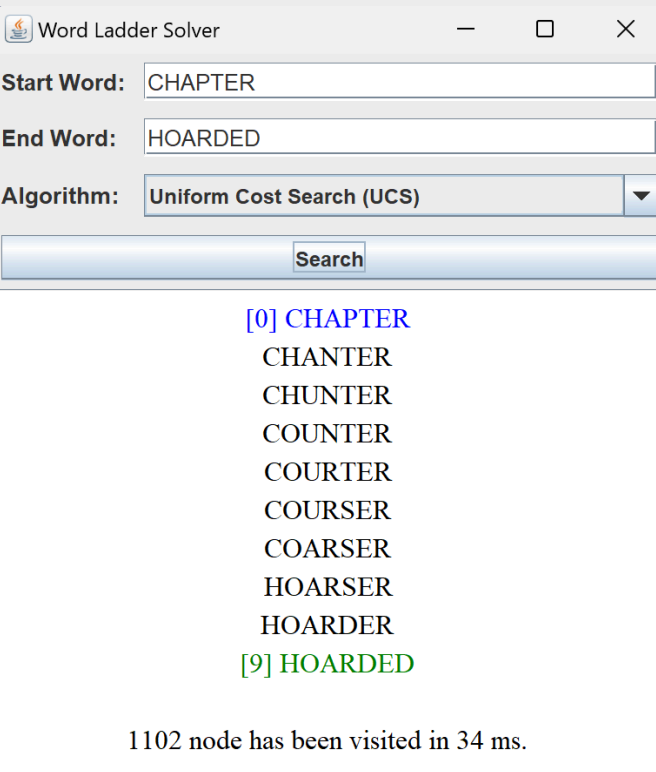
No.	Data	Hasil	
1	Start: HELLO End: STIMA	UCS	
		GDBFS	

		A*	
2	Start: OTHERS End: HEROES	UCS	
		GBFS	
		A*	



3	Start: EAST End: WEST	UCS	 <p>Word Ladder Solver</p> <p>Start Word: EAST</p> <p>End Word: WEST</p> <p>Algorithm: Uniform Cost Search (UCS)</p> <p>Search</p> <p>[0] EAST WAST [2] WEST</p> <p>71 node has been visited in 2 ms.</p>
		GBFS	 <p>Word Ladder Solver</p> <p>Start Word: EAST</p> <p>End Word: WEST</p> <p>Algorithm: Greedy Best First Search (GBFS)</p> <p>Search</p> <p>[0] EAST WAST [2] WEST</p> <p>3 node has been visited in 1 ms.</p>
		A*	 <p>Word Ladder Solver</p> <p>Start Word: EAST</p> <p>End Word: WEST</p> <p>Algorithm: A* Search</p> <p>Search</p> <p>[0] EAST WAST [2] WEST</p> <p>3 node has been visited in 0 ms.</p>

4	Start: GRASS End: ROOTS	UCS	 <p>Word Ladder Solver</p> <p>Start Word: GRASS</p> <p>End Word: ROOTS</p> <p>Algorithm: Uniform Cost Search (UCS)</p> <p>Search</p> <p>[0] GRASS GRADS GOADS GOODS ROODS [5] ROOTS</p> <p>602 node has been visited in 8 ms.</p>
		GBFS	 <p>Word Ladder Solver</p> <p>Start Word: GRASS</p> <p>End Word: ROOTS</p> <p>Algorithm: Greedy Best First Search (GBFS)</p> <p>Search</p> <p>[0] GRASS GROSS GROTS TROTS TOOTS [5] ROOTS</p> <p>6 node has been visited in 0 ms.</p>

		A*	 <p>Word Ladder Solver</p> <p>Start Word: GRASS</p> <p>End Word: ROOTS</p> <p>Algorithm: A* Search</p> <p>Search</p> <p>[0] GRASS GROSS GROTS TROTS TOOTS [5] ROOTS</p> <p>17 node has been visited in 0 ms.</p>
5	Start: CHAPTER End: HOARDED	UCS	 <p>Word Ladder Solver</p> <p>Start Word: CHAPTER</p> <p>End Word: HOARDED</p> <p>Algorithm: Uniform Cost Search (UCS)</p> <p>Search</p> <p>[0] CHAPTER CHANTER CHUNTER COUNTER COURTER COURSER COARSER HOARSER HOARDER [9] HOARDED</p> <p>1102 node has been visited in 34 ms.</p>

			<div><div>Word Ladder Solver</div><div><div>Start Word: CHAPTER</div><div>End Word: HOARDED</div><div>Algorithm: Greedy Best First Search (GBFS)</div><div>Search</div><div><div>[0] CHAPTER</div><div>CHARTER</div><div>CHARTED</div><div>CHARKED</div><div>SHARKED</div><div>SHARPED</div><div>SCARPED</div><div>SCARTED</div><div>SCANTED</div><div>SLANTED</div><div>SLATTED</div><div>BLATTED</div><div>BLASTED</div><div>BOASTED</div><div>BOOSTED</div><div>ROOSTED</div><div>ROUSTED</div><div>JOUSTED</div><div>JOISTED</div><div>JOINTED</div><div>POINTED</div><div>POINDED</div><div>POUNDED</div><div>MOUNDED</div><div>MOUNTED</div><div>COUNTED</div><div>COURTED</div><div>COURTER</div><div>COURSER</div><div>COARSER</div><div>HOARSER</div><div>HOARDER</div><div>[32] HOARDED</div></div><div>305 node has been visited in 3 ms.</div></div></div>
--	--	--	--

		A*	<div><div>Word Ladder Solver</div><div>Start Word: CHAPTER</div><div>End Word: HOARDED</div><div>Algorithm: A* Search</div><div>Search</div><div><div>[0] CHAPTER</div><div>CHANTER</div><div>CHUNTER</div><div>COUNTER</div><div>COURTER</div><div>COURSER</div><div>COARSER</div><div>HOARSER</div><div>HOARDER</div><div>[9] HOARDED</div></div><div>152 node has been visited in 3 ms.</div></div>
--	--	----	--

6	Start: ATLASES End: CABARET	UCS	<div><div>Word Ladder Solver</div><div>Start Word: ATLASES</div><div>End Word: CABARET</div><div>Algorithm: Uniform Cost Search (UCS)</div><div>Search</div><div>[0] ATLASES ANLASES ANLACES UNLACES UNLADES UNLADED UNFADED UNFAKED UNCAKED UNCAKES UNCASES UNEASES UREASES CREASES CRESSES TRESSES TRASSES BRASSES BRASHES BRASHER BRASIER BRAKIER BEAKIER PEAKIER PECKIER PICKIER DICKIER DICKIES HICKIES HACKIES HACKLES HECKLES DECKLES</div></div>
---	--------------------------------	-----	--

			<div>DECILES DEFILES DEFILED DEVEILED DEVELED REVELED RAVELED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED TAPERED TABERED TABORED TABORET TABARET [52] CABARET</div> <div>7648 node has been visited in 5754 ms.</div>
--	--	--	--

		GBFS	<div><div>Word Ladder Solver</div><div><div>Start Word: ATLASES</div><div>End Word: CABARET</div><div>Algorithm: Greedy Best First Search (GBFS)</div><div>Search</div></div><div><div>[0] ATLASES</div><div>ANLASES</div><div>ANLACES</div><div>UNLACES</div><div>UNLACED</div><div>UNLADED</div><div>UNFADED</div><div>UNFAZED</div><div>UNFAKED</div><div>UNBAKED</div><div>UNBASED</div><div>UNBATED</div><div>UNSATED</div><div>UNSAWED</div><div>UNSAVED</div><div>UNPAVED</div><div>UNPAGED</div><div>UNCAGED</div><div>UNCAGES</div><div>UNCASES</div><div>UNEASES</div><div>UREASES</div><div>CREASES</div><div>CREASED</div><div>CREATED</div><div>CHEATED</div><div>CHEATER</div><div>CHEAPER</div><div>CHEEPER</div><div>CHEERER</div><div>CHEERED</div><div>CHEEKED</div><div>CHECKED</div></div></div>
--	--	------	--



			<div>CHOCKED CLOCKED CLOCKER CLICKER CLICKED CLICHED CLICHES CLOCHES COOCHES COUCHES COUCHED COUCHER COUGHER ROUGHER ROUGHEN ROUGHED SOUGHED SOUTHED SOUTHER SOOTHER SOOTIER ROOTIER ROOMIER ROOMIES ROOKIES COOKIES COOLIES COLLIES COLLIED CULLIED GULLIED GALLIED SALLIED</div>
--	--	--	--

			<div>TALLIED TALLIER TALKIER TACKIER TACKLER CAACKLER CAACKLES HACKLES HUCKLES HECKLES HECKLER HECKLED KECKLED KECKLES DECKLES DECILES DEFILES REFILES REFIRES REHIRES RETIRES RETINES RATINES RAVINES RAVINED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED CAPERER</div> <div>TAPERER TAPERED TABERED TABORED TABORET TABARET [105] CABARET</div> <div>4647 node has been visited in 84 ms.</div>
--	--	--	---

		A*	<div><div>Word Ladder Solver</div><div><div>Start Word: ATLASES</div><div>End Word: CABARET</div><div>Algorithm: A* Search</div><div>Search</div><div><div>[0] ATLASES</div><div>ANLASES</div><div>ANLACES</div><div>UNLACES</div><div>UNLADES</div><div>UNLADED</div><div>UNFADED</div><div>UNFAKED</div><div>UNCAKED</div><div>UNCAKES</div><div>UNCASES</div><div>UNEASES</div><div>UREASES</div><div>CREASES</div><div>CRESES</div><div>CROSSES</div><div>CROSSER</div><div>CROSIER</div><div>CROZIER</div><div>CRAZIER</div><div>BRAZIER</div><div>BRAKIER</div><div>BEAKIER</div><div>PEAKIER</div><div>PECKIER</div><div>PICKIER</div><div>DICKIER</div><div>DICKIES</div><div>HICKIES</div><div>HACKIES</div><div>HACKLES</div><div>HECKLES</div><div>DECKLES</div></div></div></div>
--	--	----	---

			DECILES DEFILES REFILES REFILED REVELED RAVELED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED TAPERED TABERED TABORED TABORET TABARET [52] CABARET  7601 node has been visited in 881 ms.
--	--	--	---

Tabel 1 Hasil Pengujian

## V. Analisis Hasil Uji

Test Case	UCS			GBFS			A*		
	Panjang Solusi	Ruang	Waktu (ms)	Panjang Solusi	Ruang	Waktu (ms)	Panjang Solusi	Ruang	Waktu (ms)
2	-	8403	355	-	8403	154	-	8403	335
3	2	71	2	2	3	1	2	3	0
4	5	602	8	5	6	0	5	17	0
5	9	1102	34	32	305	3	9	152	3
6	52	7648	5754	105	4647	84	52	7601	881

Tabel 2 Statistik Hasil Uji

Tabel di atas merupakan hasil pengujian beberapa algoritma dalam penyelesaian permainan word ladder. Algoritma UCS menggunakan fungsi evaluasi  $f(n) = g(n)$ , dengan  $g(n)$  merupakan jarak dari akar ke simpul  $n$ . Algoritma GBFS menggunakan fungsi evaluasi  $f(n) = h(n)$ , dengan  $h(n)$

merupakan fungsi heuristik yaitu banyaknya huruf yang berbeda dengan huruf pada kata tujuan pada posisi yang sama. Algoritma A\* menggunakan fungsi evaluasi  $f(n) = g(n) + h(n)$ , dengan  $f(n)$  = estimasi jarak total dari kata awal ke kata tujuan dengan melalui n,  $g(n)$  merupakan jarak dari akar ke simpul n, dan  $h(n)$  merupakan fungsi heuristik yaitu banyaknya huruf yang berbeda dengan huruf pada kata tujuan pada posisi yang sama.

Berdasarkan tabel hasil uji, diperoleh informasi sebagai berikut,

1. Algoritma UCS dan A\* selalu memberikan solusi yang optimal (panjang solusi minimum) sedangkan algoritma GBFS belum tentu memberikan solusi optimal.
2. Urutan ruang (jumlah node) dari yang terkecil adalah GBFS, A\*, UCS.
3. Urutan waktu dari yang terkecil adalah GBFS, A\*, UCS

Algoritma GBFS tidak selalu memberikan solusi yang optimal karena hanya mempertimbangkan heuristik untuk memilih simpul yang paling menjanjikan secara lokal tanpa mempertimbangkan jarak aktual dari titik awal. Meskipun pendekatannya dapat mempercepat pencarian, masih besar kemungkinannya untuk mengarah pada solusi yang tidak optimal.

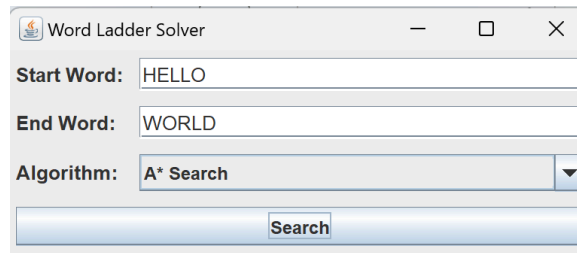
Dalam ekspansi simpul UCS, algoritma memprioritaskan jarak kumulatif paling efisien di setiap langkah, sehingga simpul yang diekspansi menempati posisi akhir dalam antrian prioritas. Pendekatan ini mirip dengan Breadth First Search, yang mana hasil perluasan diurutkan secara konsisten. Algoritma UCS menelusuri lebih banyak simpul dan durasi pencarian daripada penelusuran dengan A\*. Namun, hasil yang diberikan keduanya selalu optimal. Akan tetapi, apabila fungsi heuristik dari penelusuran A\* tidak admissible, solusi yang diperoleh belum tentu optimal. Oleh karena itu, dalam konteks Word Ladder, memilih A\* sebagai algoritma penyelesaian dapat meningkatkan efisiensi dan kecepatan dalam menemukan solusi yang optimal.

Berdasarkan salindia kuliah, sebuah fungsi heuristik  $h(n)$  dikatakan admissible jika untuk setiap simpul n dalam ruang pencarian, nilai  $h(n)$  tidak pernah melebihi biaya sebenarnya untuk mencapai tujuan dari simpul n. Dengan kata lain,  $h(n) \leq h^*(n)$  untuk setiap simpul n, di mana  $h^*(n)$  adalah biaya sebenarnya untuk mencapai tujuan dari simpul n. Heuristik yang digunakan pada kasus ini sudah admissible karena  $h(n)$  yang digunakan akan selalu memiliki nilai yang lebih kecil dibandingkan harga sesungguhnya.

## VI. Implementasi Bonus

Implementasi bonus yang dibuat menggunakan Java Swing untuk membangun antarmuka pengguna. Kelas App mewarisi dari JFrame dan digunakan untuk membuat jendela aplikasi. Pada jendela tersebut, terdapat panel input yang menggunakan GridBagLayout untuk mengatur komponen-komponen dengan lebih fleksibel. Komponen input terdiri dari label dan JTextField untuk memasukkan kata awal dan akhir, serta sebuah JComboBox untuk memilih algoritma pencarian. Terdapat juga sebuah tombol "Search" yang akan memulai pencarian saat diklik. Hasil pencarian ditampilkan dalam sebuah JTextPane yang telah diset dengan tipe konten HTML untuk memudahkan penampilan teks. Saat tombol "Search" diklik, kata-kata awal dan akhir serta algoritma pencarian yang dipilih akan diambil dari input pengguna. Kemudian, kamus kata-kata yang sesuai dengan panjang kata awal akan dimuat dari file dan disimpan dalam sebuah set.

Setelah itu, pencarian akan dilakukan menggunakan algoritma yang dipilih dan hasilnya akan ditampilkan dalam JTextPane yang dapat digulirkan.



[0] HELLO

HELLS

HEALS

WEALS

WEALD

WOALD

[6] WORLD

46 node has been visited in 1 ms.

Ilustrasi 2. Antarmuka pengguna Word Ladder Solver  
(Sumber: dokumen pribadi)

## VII. Lampiran

Repositori GitHub : [https://github.com/mdavaf17/Tucil3\\_13522114](https://github.com/mdavaf17/Tucil3_13522114)

No	Poin	Ya	Tidak
1	Program berhasil dijalankan.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	Solusi yang diberikan pada algoritma UCS optimal	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma A*	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	Solusi yang diberikan pada algoritma A* optimal	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	<b>[Bonus]</b> : Program memiliki tampilan GUI	<input checked="" type="checkbox"/>	<input type="checkbox"/>