

**Laporan Tugas Kecil 3
IF2211 Strategi Algoritma
Semester II Tahun 2023/2024**

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



Disusun oleh:
Bastian H. Suryapratama
13522034
K02

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

Daftar Isi

Daftar Isi.....	2
I. Analisis dan Implementasi.....	3
A. UCS (Uniform Cost Search).....	3
B. Greedy Best-First Search.....	3
C. A*.....	3
II. Source Code Program Implementasi Algoritma.....	4
A. Brute Force 3 Titik Kontrol.....	4
B. Divide and Conquer 3 Titik Kontrol.....	5
C. Divide and Conquer n Titik Kontrol.....	6
III. Tangkapan Layar.....	9
A. Brute Force 3 Titik Kontrol.....	9
B. Divide and Conquer 3 Titik Kontrol.....	11
C. Divide and Conquer n Titik Kontrol.....	13
D. Divide and Conquer Visualisasi Setiap Iterasi.....	15
IV. Hasil Analisis Perbandingan Algoritma Brute Force dengan Divide and Conquer.....	18
V. Implementasi Bonus.....	20
A. Divide and Conquer n Titik Kontrol.....	20
B. Divide and Conquer Visualisasi Setiap Iterasi.....	21
Lampiran.....	23

I. Analisis dan Implementasi

A. UCS (Uniform Cost Search)

UCS (Uniform Cost Search) adalah algoritma pencarian berdasarkan biaya untuk menempuh perjalanan dari simpul akar ke simpul n, yang disimbolkan dengan $g(n)$. Algoritma ini akan memprioritaskan simpul dengan $g(n)$ yang terkecil.

Berikut ini adalah langkah-langkah pencarian menggunakan UCS:

1. Masukkan suatu simpul ke dalam simpul *expand*. Pada awalnya, simpul yang menjadi simpul *expand* adalah simpul akar. Dalam kasus ini, simpul akar adalah *start word*.
2. Cek apakah simpul *expand* adalah simpul tujuan (*goal*). Jika ya, berarti pencarian selesai dan solusi berhasil ditemukan. Dalam kasus ini, simpul tujuan adalah *end word*.
3. Cari semua simpul yang bertetangga dengan simpul *expand* terakhir, kemudian masukkan simpul-simpul tersebut ke dalam daftar simpul hidup. Dalam kasus ini, simpul-simpul yang bertetangga adalah kata-kata *valid* yang bisa dicapai hanya dengan satu kali melakukan perubahan pada salah satu huruf.
4. Ambil satu simpul dari daftar simpul hidup dengan $g(n)$ terkecil. Simpul inilah yang akan menjadi simpul *expand* berikutnya.
5. Ulangi langkah 1-4. Jika tidak ada simpul *expand* yang baru, berarti solusi gagal ditemukan.

Dalam algoritma ini, biaya yang dijadikan dasar dalam pencarian, disimbolkan dengan $f(n)$, hanyalah $g(n)$. Dapat disimpulkan bahwa $f(n) = g(n)$.

Dalam permainan word ladder, banyaknya sisi yang dilewati untuk menempuh perjalanan dari suatu node ke node lainnya sama dengan banyaknya perubahan huruf yang dilakukan untuk mengubah suatu kata menjadi kata yang lain. Simpul yang hanya berbeda satu level dengan simpul lainnya dalam pohon pencarian UCS tentu akan diprioritaskan untuk di-*expand* terlebih dahulu. Oleh karena itu, pohon pencarian pada algoritma UCS akan sama dengan pohon pencarian pada algoritma BFS (Breadth-First Search) sehingga urutan simpul yang dibangkitkan dan path yang dihasilkan juga sama.

B. GBFS (Greedy Best-First Search)

GBFS (Greedy Best-First Search) adalah algoritma pencarian berdasarkan estimasi biaya untuk menempuh perjalanan dari suatu simpul n ke simpul tujuan (*goal*), yang

disimbolkan dengan $h(n)$. Algoritma ini akan memprioritaskan simpul dengan $h(n)$ yang terkecil.

Berikut ini adalah langkah-langkah pencarian menggunakan GBFS:

1. Masukkan suatu simpul ke dalam simpul *expand*. Pada awalnya, simpul yang menjadi simpul *expand* adalah simpul akar. Dalam kasus ini, simpul akar adalah *start word*.
2. Cek apakah simpul *expand* adalah simpul tujuan (*goal*). Jika ya, berarti pencarian selesai dan solusi berhasil ditemukan. Dalam kasus ini, simpul tujuan adalah *end word*.
3. Cari semua simpul yang bertetangga dengan simpul *expand* terakhir, kemudian masukkan simpul-simpul tersebut ke dalam daftar simpul hidup. Dalam kasus ini, simpul-simpul yang bertetangga adalah kata-kata *valid* yang bisa dicapai hanya dengan satu kali melakukan perubahan pada salah satu huruf.
4. Ambil satu simpul dari daftar simpul hidup dengan $h(n)$ terkecil. Simpul inilah yang akan menjadi simpul *expand* berikutnya.
5. Ulangi langkah 1-4. Jika tidak ada simpul *expand* yang baru, berarti solusi gagal ditemukan.

Dalam algoritma ini, biaya yang dijadikan dasar dalam pencarian, disimbolkan dengan $f(n)$, hanyalah $h(n)$. Dapat disimpulkan bahwa $f(n) = h(n)$.

Dalam permainan word ladder, algoritma GBFS secara teoretis tidak menjamin bahwa solusi yang didapatkan adalah solusi yang optimal. Algoritma ini hanya berfokus pada kondisi optimum yang lokal. Memprioritaskan pencarian pada optimum lokal tidak akan bisa menjamin hasil akhirnya masih nilai optimum.

C. A* (A-star)

A* (A-star) adalah algoritma pencarian berdasarkan biaya untuk menempuh perjalanan dari simpul akar ke simpul n, yang disimbolkan dengan $g(n)$, serta estimasi biaya untuk menempuh perjalanan dari suatu simpul n ke simpul tujuan (*goal*), yang disimbolkan dengan $h(n)$. Algoritma ini akan memprioritaskan simpul dengan $g(n) + h(n)$ yang terkecil.

Berikut ini adalah langkah-langkah pencarian menggunakan GBFS:

1. Masukkan suatu simpul ke dalam simpul *expand*. Pada awalnya, simpul yang menjadi simpul *expand* adalah simpul akar. Dalam kasus ini, simpul akar adalah *start word*.

2. Cek apakah simpul *expand* adalah simpul tujuan (*goal*). Jika ya, berarti pencarian selesai dan solusi berhasil ditemukan. Dalam kasus ini, simpul tujuan adalah *end word*.
3. Cari semua simpul yang bertetangga dengan simpul *expand* terakhir, kemudian masukkan simpul-simpul tersebut ke dalam daftar simpul hidup. Dalam kasus ini, simpul-simpul yang bertetangga adalah kata-kata *valid* yang bisa dicapai hanya dengan satu kali melakukan perubahan pada salah satu huruf.
4. Ambil satu simpul dari daftar simpul hidup dengan $g(n) + h(n)$ terkecil. Simpul inilah yang akan menjadi simpul *expand* berikutnya.
5. Ulangi langkah 1-4. Jika tidak ada simpul *expand* yang baru, berarti solusi gagal ditemukan.

Dalam algoritma ini, biaya yang dijadikan dasar dalam pencarian, disimbolkan dengan $f(n)$, adalah total dari $g(n)$ dan $h(n)$. Dapat disimpulkan bahwa $f(n) = g(n) + h(n)$.

Dalam permainan word ladder, heuristik yang digunakan, yang disimbolkan dengan $h(n)$, adalah banyaknya huruf yang berbeda antara suatu kata dengan kata lainnya. Banyaknya perubahan huruf yang sebenarnya perlu dilakukan (dengan mempertimbangkan validitas kata), yang disimbolkan dengan $h^*(n)$, akan selalu lebih besar atau sama dengan $h(n)$. Karena $h(n) \leq h^*(n)$, heuristik $h(n)$ bersifat *admissible*.

Algoritma A* secara teoretis lebih efisien dibandingkan algoritma UCS karena A* mengambil sisi baik dari algoritma UCS dan GBFS. Oleh karena itu, algoritma ini mempunyai efisiensi yang lebih baik dibandingkan dengan algoritma UCS.

II. *Source Code* Program Implementasi Algoritma

Program ini ditulis menggunakan bahasa pemrograman Java. Program tersebut terdiri dari 6 file utama yang berperan dalam pencarian, yaitu WordList.java, Node.java, Searching.java, UniformCostSearch.java, GreedyBestFirstSearch.java, dan AStar.java.

A. WordList.java

Kelas WordList bertugas untuk membaca daftar kata dari file .txt serta menghasilkan daftar simpul yang bertetangga dengan simpul lainnya.

- Method isWordExist digunakan untuk mengecek apakah suatu kata terdapat dalam daftar kata yang telah dibaca.
- Method isWordsDifferBy1Char digunakan untuk mengecek apakah suatu kata hanya berbeda 1 huruf dengan kata lainnya.
- Method generateWordsWithSameLength digunakan untuk menghasilkan daftar kata-kata yang memiliki panjang kata yang sama.
- Method generateWordToWordsDifferBy1Char digunakan untuk menghasilkan daftar kata yang hanya berbeda 1 huruf dengan kata lainnya.
- Method getWordsDifferBy1Char digunakan untuk mengambil kata-kata yang hanya berbeda 1 huruf dengan suatu kata.

```

1 import java.io.BufferedReader;
2 import java.io.File;
3 import java.io.FileReader;
4 import java.io.IOException;
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.Map;
8 import java.util.HashMap;
9
10 class WordList {
11     private final List<String> words;
12     private Map<String, List<String>> wordToWordsDifferBy1Char;
13
14     WordList(File wordsFile) throws IOException {
15         words = new ArrayList<>();
16         BufferedReader reader = new BufferedReader(new FileReader(wordsFile));
17         String line;
18         while ((line = reader.readLine()) != null) {
19             words.add(line.toLowerCase());
20         }
21         reader.close();
22     }
23
24     boolean isWordNotExist(String word) {
25         return !words.contains(word);
26     }
27
28     private static boolean isWordsDifferBy1Char(String word1, String word2) {
29         if (word1.length() != word2.length()) {
30             throw new IllegalArgumentException("Both words must have the same length");
31         }
32         int count = 0;
33         int i = 0;
34         while (i < word1.length() && count <= 1) {
35             if (word1.charAt(i) != word2.charAt(i)) {
36                 ++count;
37             }
38             ++i;
39         }
40         return count == 1;
41     }
42
43     private List<String> generateWordsWithSameLength(int wordLength) {
44         return words.stream().filter(eachWord -> eachWord.length() == wordLength).toList();
45     }
46
47     void generateWordToWordsDifferBy1Char(int wordLength) {
48         List<String> wordsWithSameLength = generateWordsWithSameLength(wordLength);
49         wordToWordsDifferBy1Char = new HashMap<>();
50         for (String key: wordsWithSameLength) {
51             List<String> value = new ArrayList<>();
52             for (String anotherWord: wordsWithSameLength) {
53                 if (isWordsDifferBy1Char(key, anotherWord)) {
54                     value.add(anotherWord);
55                 }
56             }
57             wordToWordsDifferBy1Char.put(key, value);
58         }
59     }
60
61     List<String> getWordsDifferBy1Char(String word) {
62         if (wordToWordsDifferBy1Char == null) {
63             throw new IllegalStateException("Words differ by 1 character must be generated first");
64         }
65         return wordToWordsDifferBy1Char.get(word);
66     }
67 }
68 }
69

```

B. Node.java

Kelas abstrak Node adalah representasi sebuah simpul dalam algoritma pencarian. Kelas Node diturunkan menjadi 3 kelas, yaitu NodeUCS, NodeGBFS, dan NodeAStar, yang masing-masing merepresentasikan sebuah simpul dalam algoritma pencarian UCS, GBFS, dan A*.

- Method getCurrentWord pada kelas Node digunakan untuk mengambil kata terakhir dari suatu simpul.
- Method getPreviousWords pada kelas Node digunakan untuk mengambil kata-kata sebelumnya dari suatu simpul.
- Method numOfCharDifference pada kelas Node digunakan untuk menghitung banyaknya karakter yang berbeda antara suatu kata dengan kata lainnya.
- Method compareTo pada Kelas NodeUCS, NodeGBFS dan NodeAStar digunakan untuk membandingkan nilai $f(n)$ antara suatu simpul (*node*) dengan simpul lainnya.

```

1 import java.util.List;
2
3 abstract class Node implements Comparable<Node> {
4     final String currentWord;
5     final List<String> previousWords;
6
7     Node(String currentWord, List<String> previousWords) {
8         this.currentWord = currentWord;
9         this.previousWords = previousWords;
10    }
11
12    String getCurrentWord() {
13        return currentWord;
14    }
15
16    List<String> getPreviousWords() {
17        return previousWords;
18    }
19
20    static int numOfCharDifference(String word1, String word2) {
21        if (word1.length() != word2.length()) {
22            throw new IllegalArgumentException("Both words must have the same length");
23        }
24        int count = 0;
25        for (int i = 0; i < word1.length(); ++i) {
26            if (word1.charAt(i) != word2.charAt(i)) {
27                ++count;
28            }
29        }
30        return count;
31    }
32 }
33
34 class NodeUCS extends Node {
35     NodeUCS(String currentWord, List<String> previousWords) {
36         super(currentWord, previousWords);
37     }
38
39     @Override
40     public int compareTo(Node otherNode) {
41         int comparisonResult = Integer.compare(previousWords.size(), otherNode.previousWords.size());
42         if (comparisonResult != 0) {
43             return comparisonResult;
44         }
45         return currentWord.compareTo(otherNode.currentWord);
46     }
47 }
48
49 class NodeGBFS extends Node {
50     private final int diffFromCurrentToEndWord;
51
52     NodeGBFS(String currentWord, List<String> previousWords, String endWord) {
53         super(currentWord, previousWords);
54         diffFromCurrentToEndWord = numOfCharDifference(currentWord, endWord);
55     }
56
57     @Override
58     public int compareTo(Node otherNode) {
59         int comparisonResult = Integer.compare(diffFromCurrentToEndWord, ((NodeGBFS) otherNode).diffFromCurrentToEndWord);
60         if (comparisonResult != 0) {
61             return comparisonResult;
62         }
63         return currentWord.compareTo(otherNode.currentWord);
64     }
65 }
66
67 class NodeAStar extends Node {
68     private final int diffFromCurrentToEndWord;
69
70     NodeAStar(String currentWord, List<String> previousWords, String endWord) {
71         super(currentWord, previousWords);
72         diffFromCurrentToEndWord = numOfCharDifference(currentWord, endWord);
73     }
74
75     @Override
76     public int compareTo(Node otherNode) {
77         int thisNodeCost = previousWords.size() + diffFromCurrentToEndWord;
78         int otherNodeCost = otherNode.previousWords.size() + ((NodeAStar) otherNode).diffFromCurrentToEndWord;
79         int comparisonResult = Integer.compare(thisNodeCost, otherNodeCost);
80         if (comparisonResult != 0) {
81             return comparisonResult;
82         }
83         return currentWord.compareTo(otherNode.currentWord);
84     }
85 }
86

```

C. Searching.java

Kelas abstrak Searching adalah kelas yang merepresentasikan algoritma pencarian.

- Method isWordNotVisited digunakan untuk mengecek apakah suatu kata belum pernah masuk ke dalam daftar simpul *expand* sebelumnya.
- Method abstrak getPath digunakan untuk mendapatkan solusi pencarian. Method abstrak ini akan diimplementasikan oleh kelas anak dari kelas Searching.
- Method getNumOfNodesVisited digunakan untuk mendapatkan banyaknya simpul yang dikunjungi atau di-*expand* selama melakukan pencarian.

```
1 import java.util.List;
2 import java.util.PriorityQueue;
3
4 abstract class Searching {
5     Node currentExpandNode;
6     List<Node> previousExpandNodes;
7     PriorityQueue<Node> lifeNodes;
8     int numOfNodesVisited;
9
10    boolean isWordNotVisited(String word) {
11        for (Node eachNode: previousExpandNodes) {
12            if (eachNode.getCurrentWord().equals(word)) {
13                return false;
14            }
15        }
16        return true;
17    }
18
19    abstract List<String> getPath(String startWord, String endWord, WordList wordList);
20
21    int getNumOfNodesVisited() {
22        return numOfNodesVisited;
23    }
24 }
25
```

D. UniformCostSearch.java

Kelas UniformCostSearch adalah kelas turunan dari kelas Searching yang merepresentasikan algoritma pencarian UCS.

- Method getPath digunakan untuk mendapatkan solusi pencarian menggunakan algoritma UCS.

```

● ● ●

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.PriorityQueue;
4
5 public class UniformCostSearch extends Searching {
6     @Override
7     List<String> getPath(String startWord, String endWord, WordList wordList) {
8         currentExpandNode = new NodeUCS(startWord, new ArrayList<>());
9         previousExpandNodes = new ArrayList<>();
10        lifeNodes = new PriorityQueue<>();
11        numOfNodesVisited = 0;
12
13        while (currentExpandNode != null) {
14            ++numOfNodesVisited;
15            if (currentExpandNode.getCurrentWord().equals(endWord)) {
16                break;
17            }
18
19            List<String> wordsDifferBy1Char = wordList.getWordsDifferBy1Char(currentExpandNode.getCurrentWord());
20            wordsDifferBy1Char = wordsDifferBy1Char.stream().filter(this::isWordNotVisited).toList();
21
22            for (String eachWord: wordsDifferBy1Char) {
23                List<String> newPreviousWords = new ArrayList<>(currentExpandNode.getPreviousWords());
24                newPreviousWords.add(currentExpandNode.getCurrentWord());
25                NodeUCS newNode = new NodeUCS(eachWord, newPreviousWords);
26                lifeNodes.add(newNode);
27            }
28
29            previousExpandNodes.add(currentExpandNode);
30            currentExpandNode = lifeNodes.poll();
31        }
32
33        if (currentExpandNode == null) {
34            return new ArrayList<>();
35        }
36        List<String> result = new ArrayList<>(currentExpandNode.getPreviousWords());
37        result.add(currentExpandNode.getCurrentWord());
38        return result;
39    }
40 }
41

```

E. GreedyBestFirstSearch.java

Kelas GreedyBestFirstSearch adalah kelas turunan dari kelas Searching yang merepresentasikan algoritma pencarian GBFS.

- Method getPath digunakan untuk mendapatkan solusi pencarian menggunakan algoritma GBFS.

```
● ● ●
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.PriorityQueue;
4
5 public class GreedyBestFirstSearch extends Searching {
6     @Override
7     List<String> getPath(String startWord, String endWord, WordList wordList) {
8         currentExpandNode = new NodeGBFS(startWord, new ArrayList<>(), endWord);
9         previousExpandNodes = new ArrayList<>();
10        lifeNodes = new PriorityQueue<>();
11        numOfNodesVisited = 0;
12
13        while (currentExpandNode != null) {
14            ++numOfNodesVisited;
15            if (currentExpandNode.getCurrentWord().equals(endWord)) {
16                break;
17            }
18
19            List<String> wordsDifferBy1Char = wordList.getWordsDifferBy1Char(currentExpandNode.getCurrentWord());
20            wordsDifferBy1Char = wordsDifferBy1Char.stream().filter(this::isWordNotVisited).toList();
21
22            for (String eachWord: wordsDifferBy1Char) {
23                List<String> newPreviousWords = new ArrayList<>(currentExpandNode.getPreviousWords());
24                newPreviousWords.add(currentExpandNode.getCurrentWord());
25                NodeGBFS newNode = new NodeGBFS(eachWord, newPreviousWords, endWord);
26                lifeNodes.add(newNode);
27            }
28
29            previousExpandNodes.add(currentExpandNode);
30            currentExpandNode = lifeNodes.poll();
31        }
32
33        if (currentExpandNode == null) {
34            return new ArrayList<>();
35        }
36        List<String> result = new ArrayList<>(currentExpandNode.getPreviousWords());
37        result.add(currentExpandNode.getCurrentWord());
38        return result;
39    }
40 }
41
```

F. AStar.java

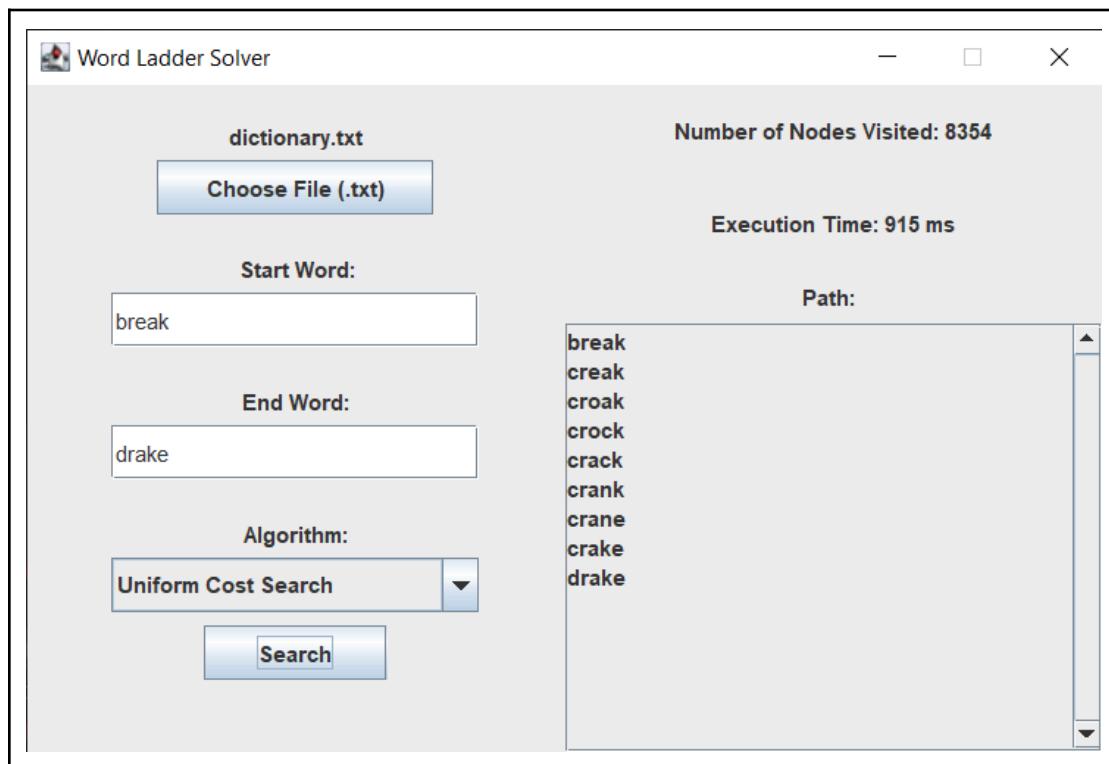
Kelas AStar adalah kelas turunan dari kelas Searching yang merepresentasikan algoritma pencarian A*.

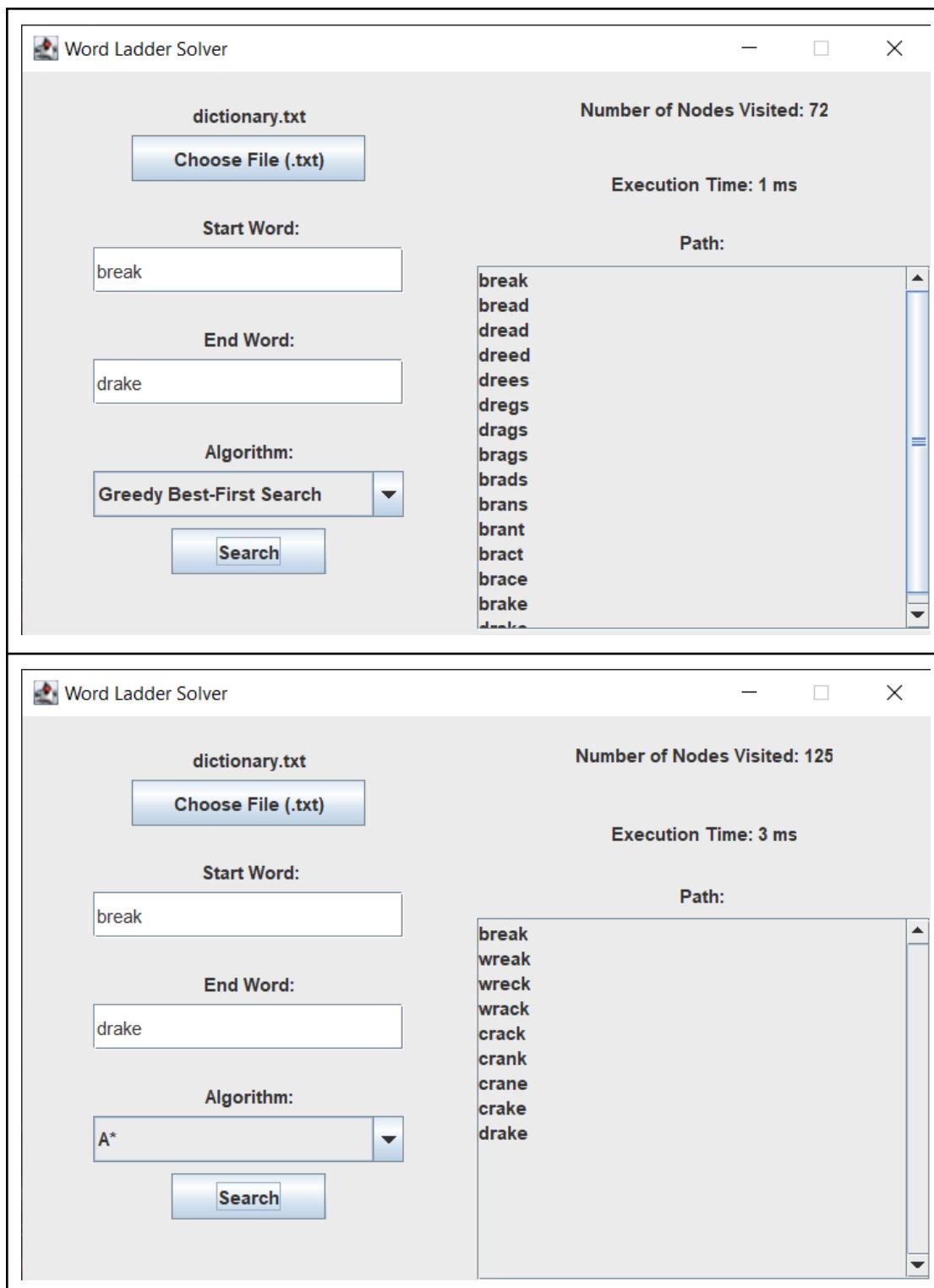
- Method getPath digunakan untuk mendapatkan solusi pencarian menggunakan algoritma A*.

```
● ● ●
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.PriorityQueue;
4
5 public class AStar extends Searching {
6     @Override
7     List<String> getPath(String startWord, String endWord, WordList wordList) {
8         currentExpandNode = new NodeAStar(startWord, new ArrayList<>(), endWord);
9         previousExpandNodes = new ArrayList<>();
10        lifeNodes = new PriorityQueue<>();
11        numOfNodesVisited = 0;
12
13        while (currentExpandNode != null) {
14            ++numOfNodesVisited;
15            if (currentExpandNode.getCurrentWord().equals(endWord)) {
16                break;
17            }
18
19            List<String> wordsDifferBy1Char = wordList.getWordsDifferBy1Char(currentExpandNode.getCurrentWord());
20            wordsDifferBy1Char = wordsDifferBy1Char.stream().filter(this::isWordNotVisited).toList();
21
22            for (String eachWord: wordsDifferBy1Char) {
23                List<String> newPreviousWords = new ArrayList<>(currentExpandNode.getPreviousWords());
24                newPreviousWords.add(currentExpandNode.getCurrentWord());
25                NodeAStar newNode = new NodeAStar(eachWord, newPreviousWords, endWord);
26                lifeNodes.add(newNode);
27            }
28
29            previousExpandNodes.add(currentExpandNode);
30            currentExpandNode = lifeNodes.poll();
31        }
32
33        if (currentExpandNode == null) {
34            return new ArrayList<>();
35        }
36        List<String> result = new ArrayList<>(currentExpandNode.getPreviousWords());
37        result.add(currentExpandNode.getCurrentWord());
38        return result;
39    }
40 }
41
```

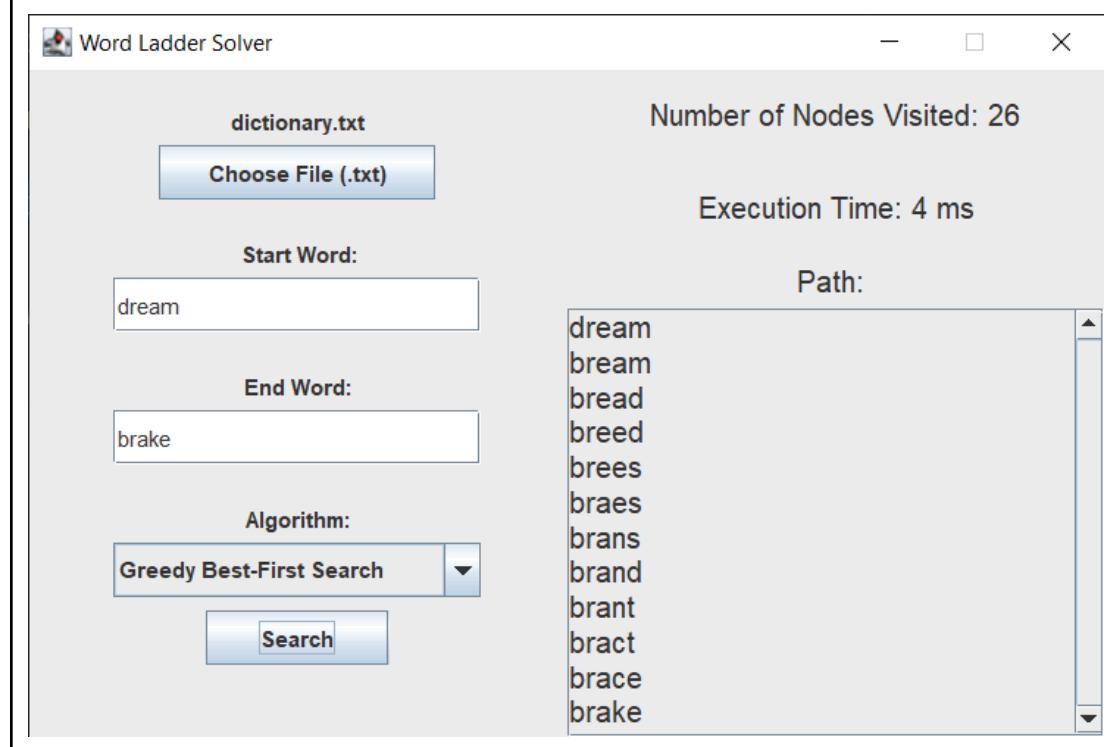
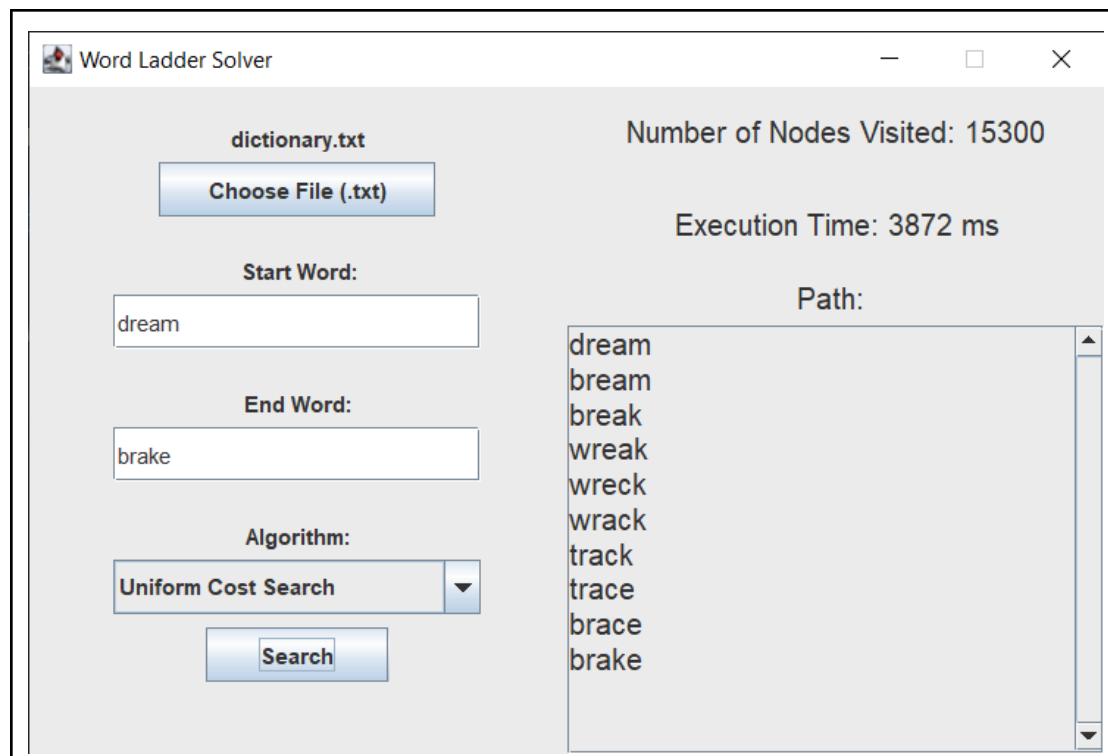
III. Tangkapan Layar

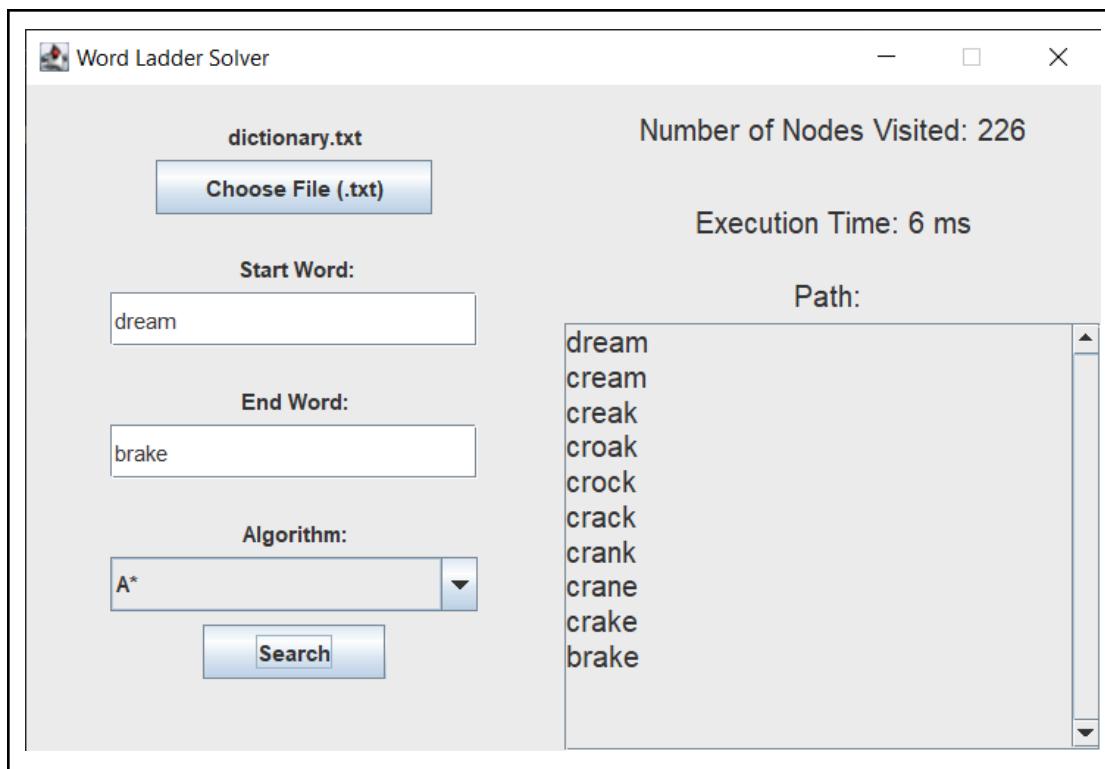
A. Test Case 1



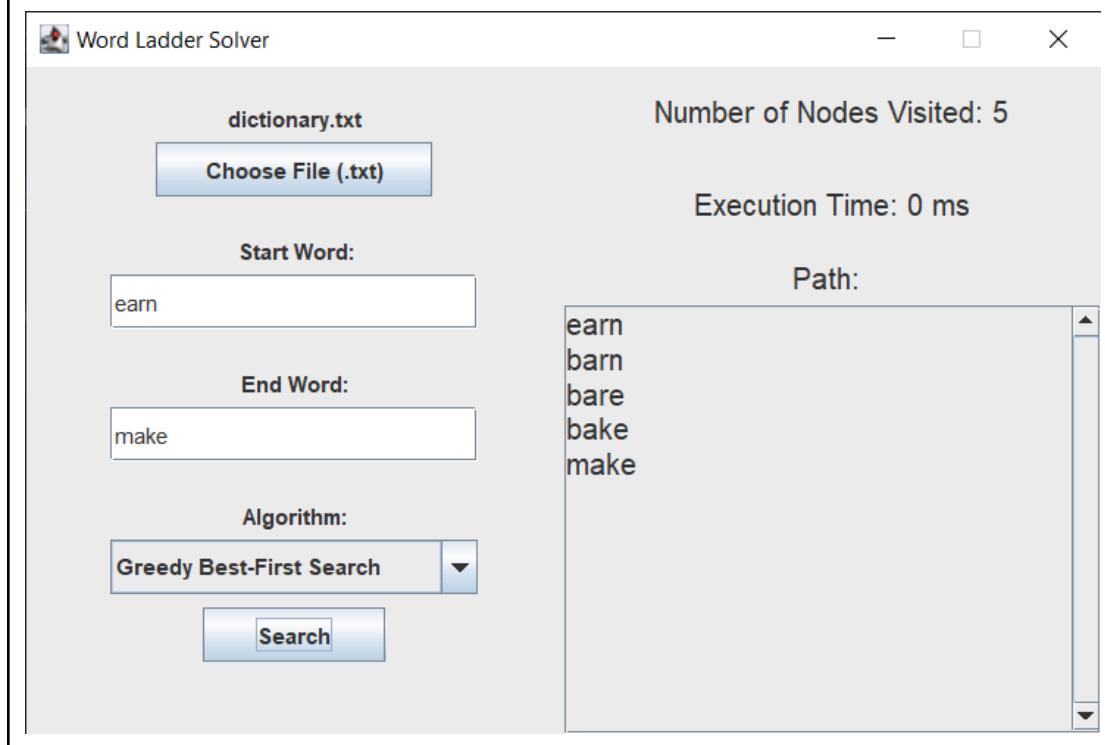
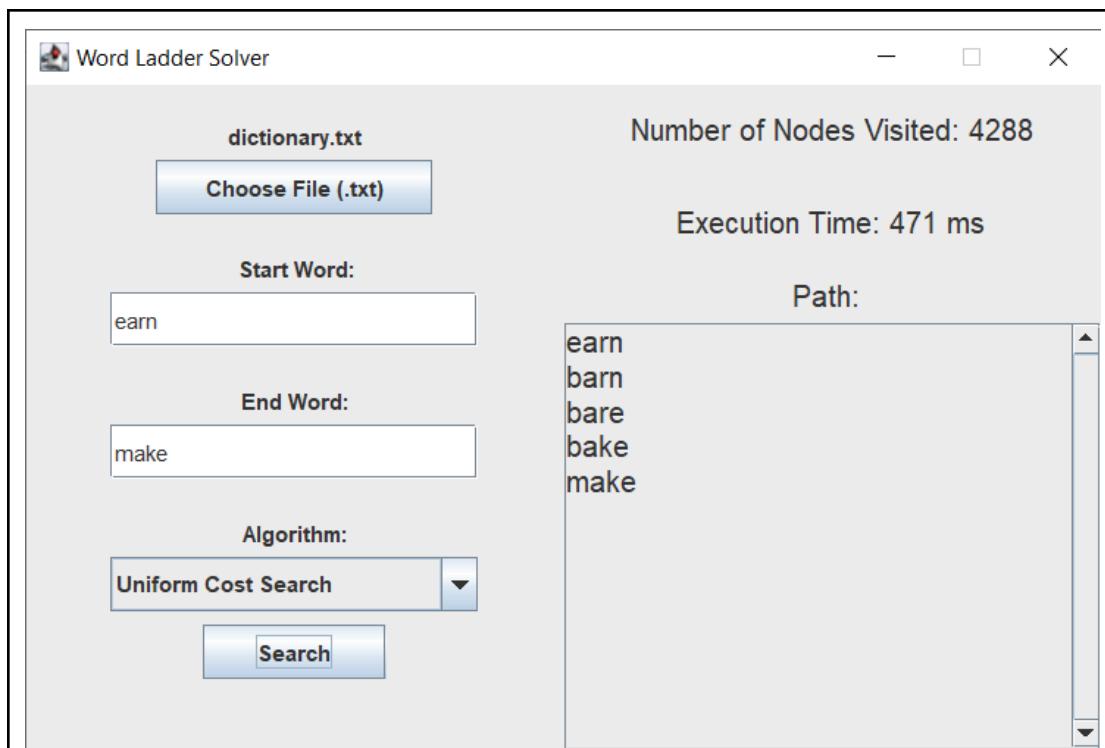


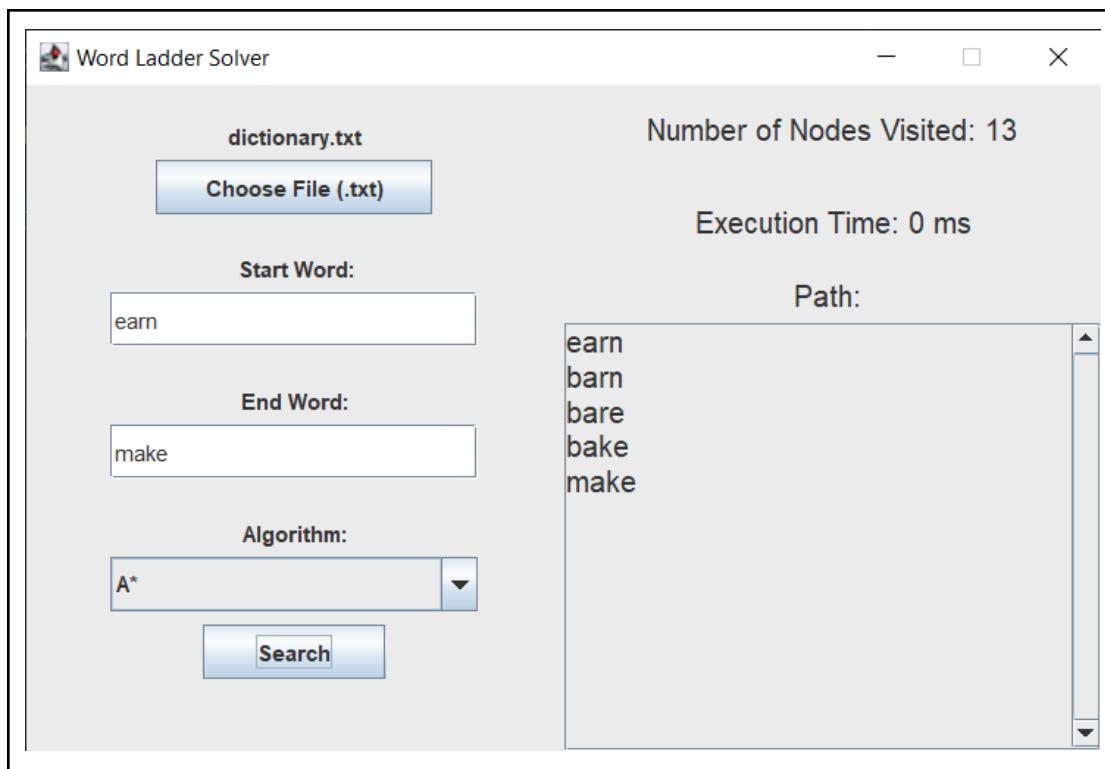
B. Test Case 2



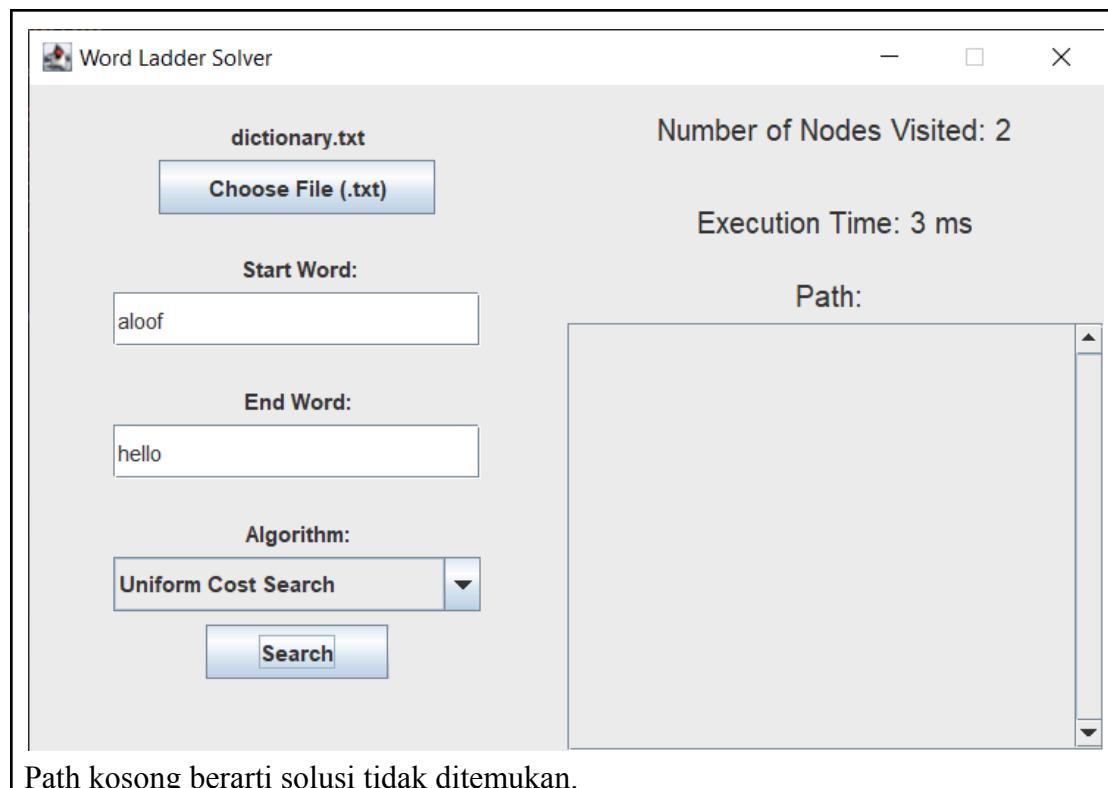


C. Test Case 3

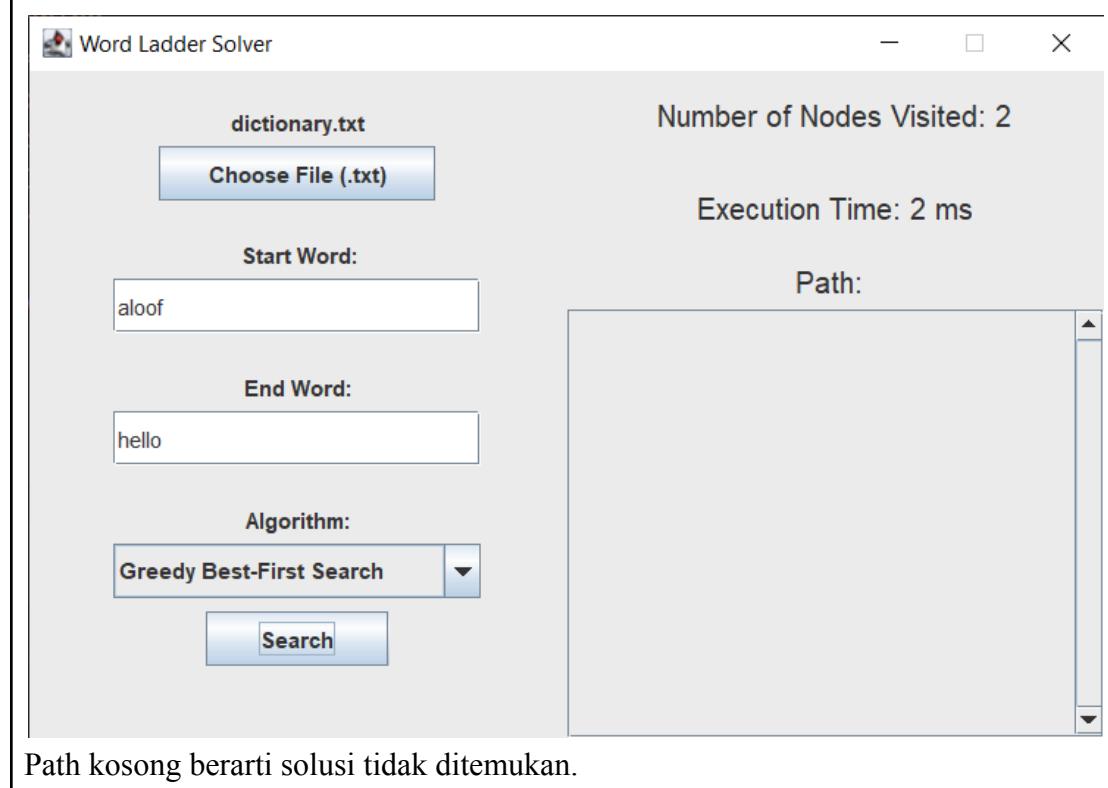




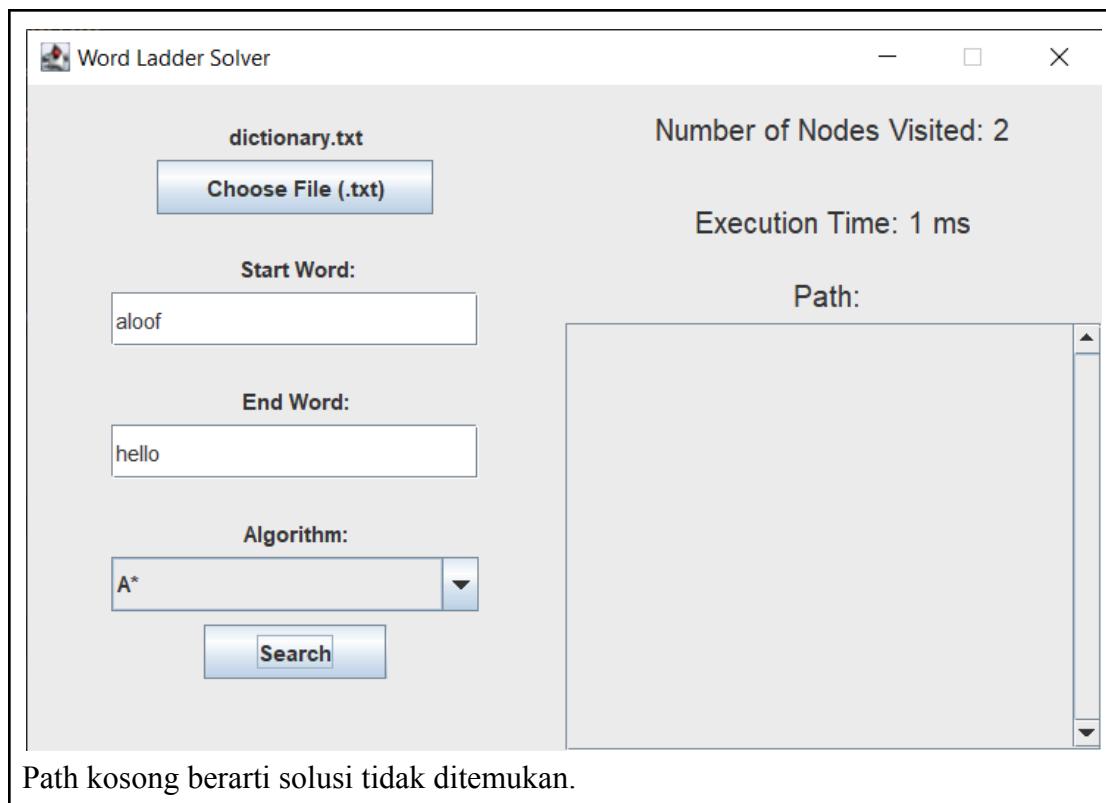
D. Test Case 4



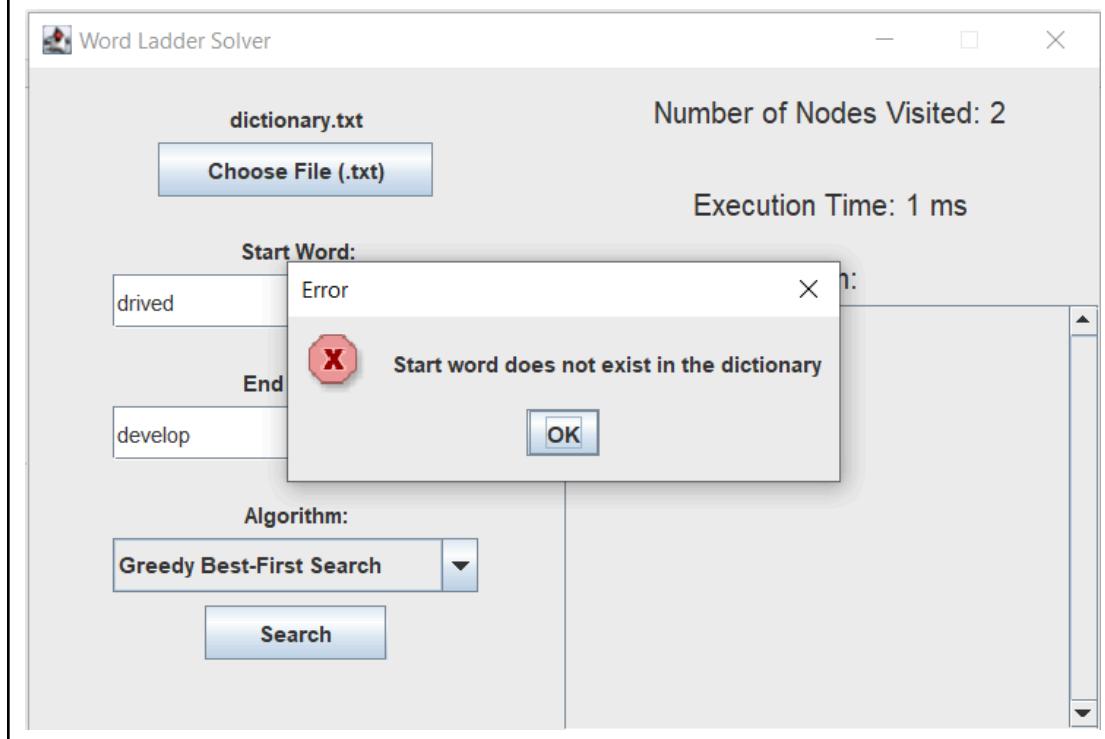
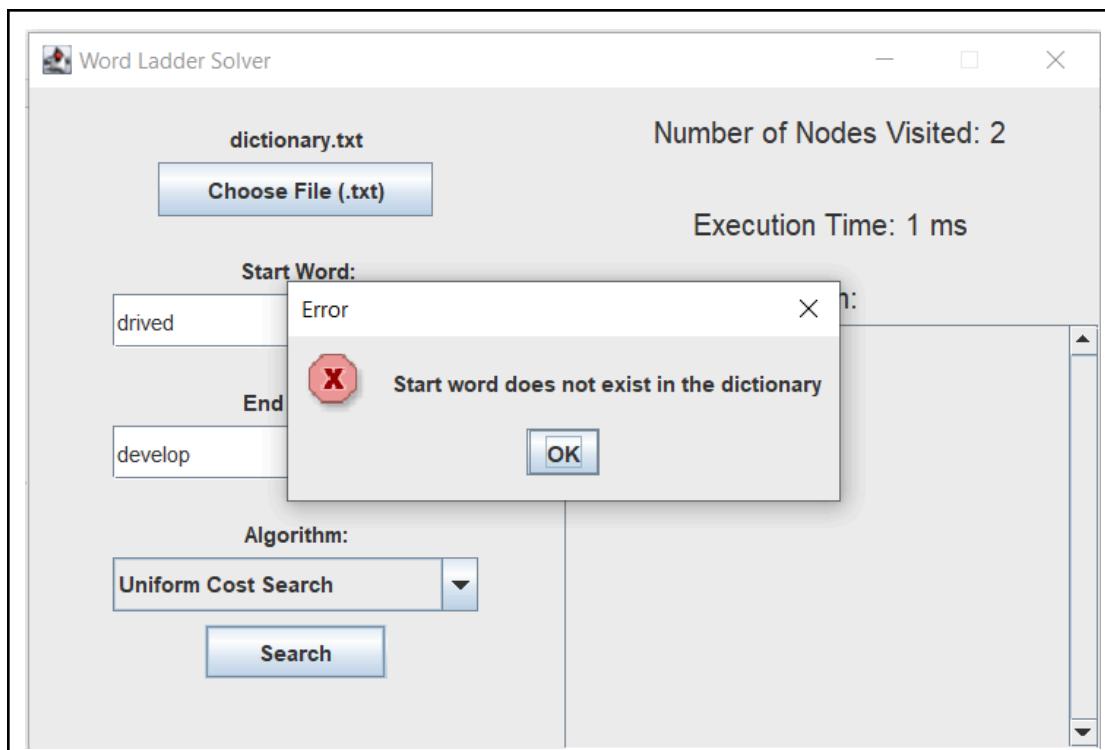
Path kosong berarti solusi tidak ditemukan.

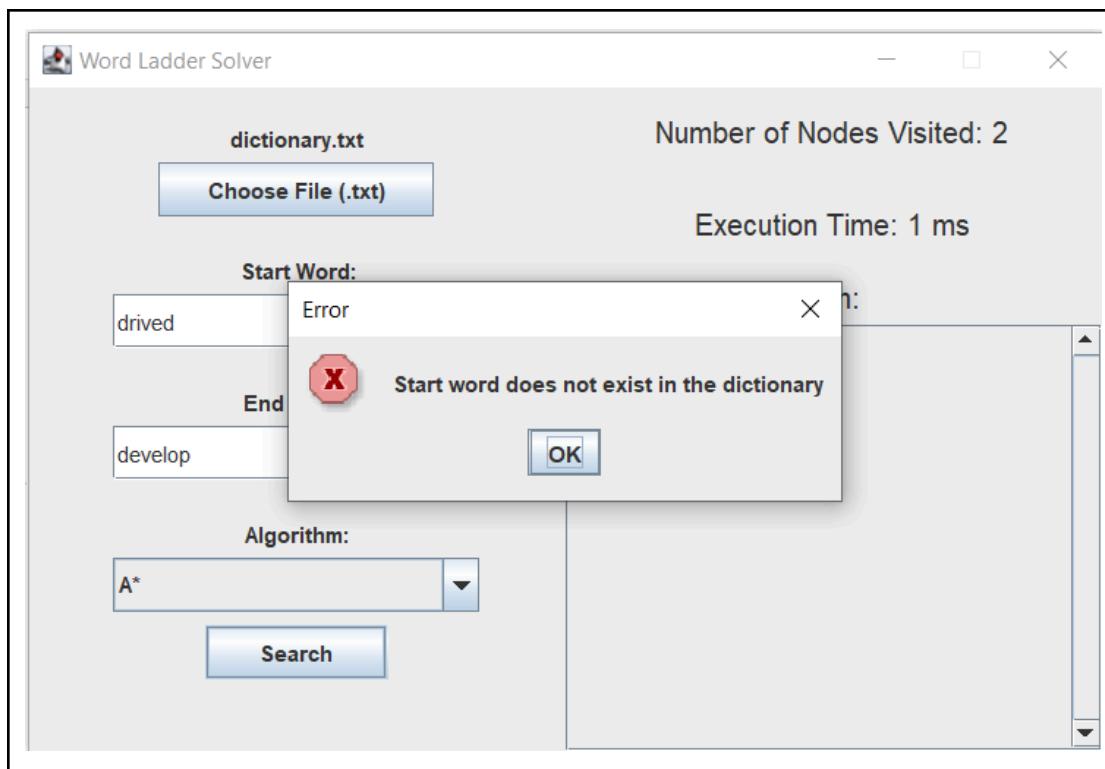


Path kosong berarti solusi tidak ditemukan.

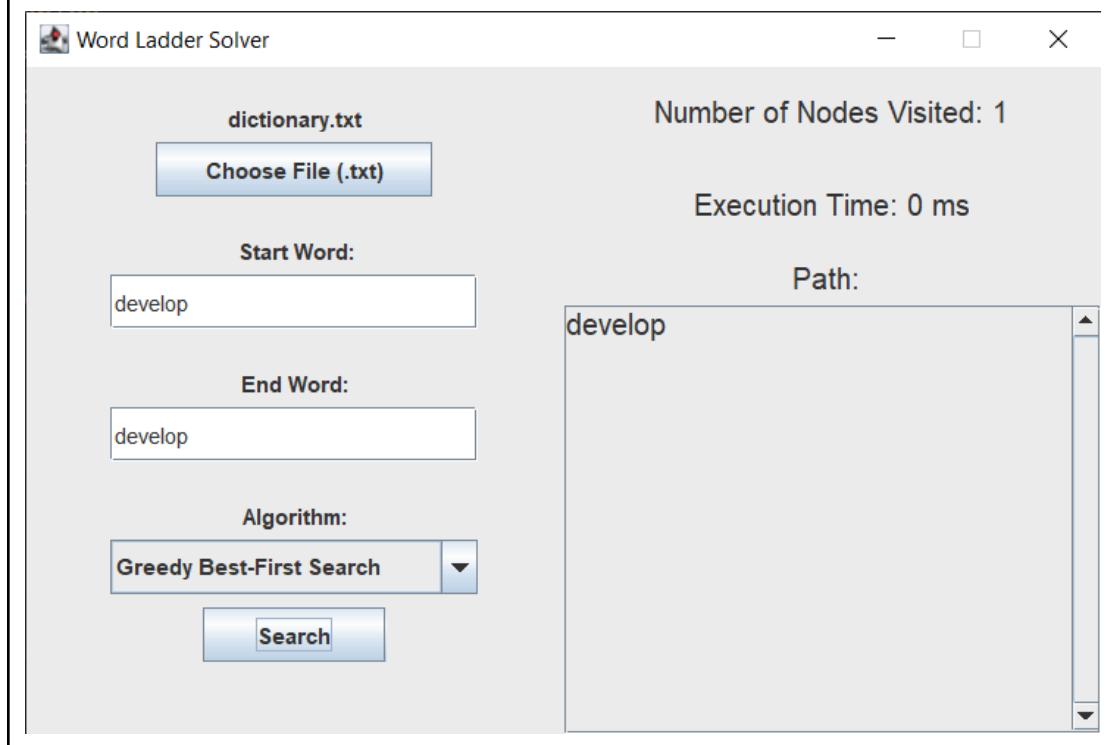
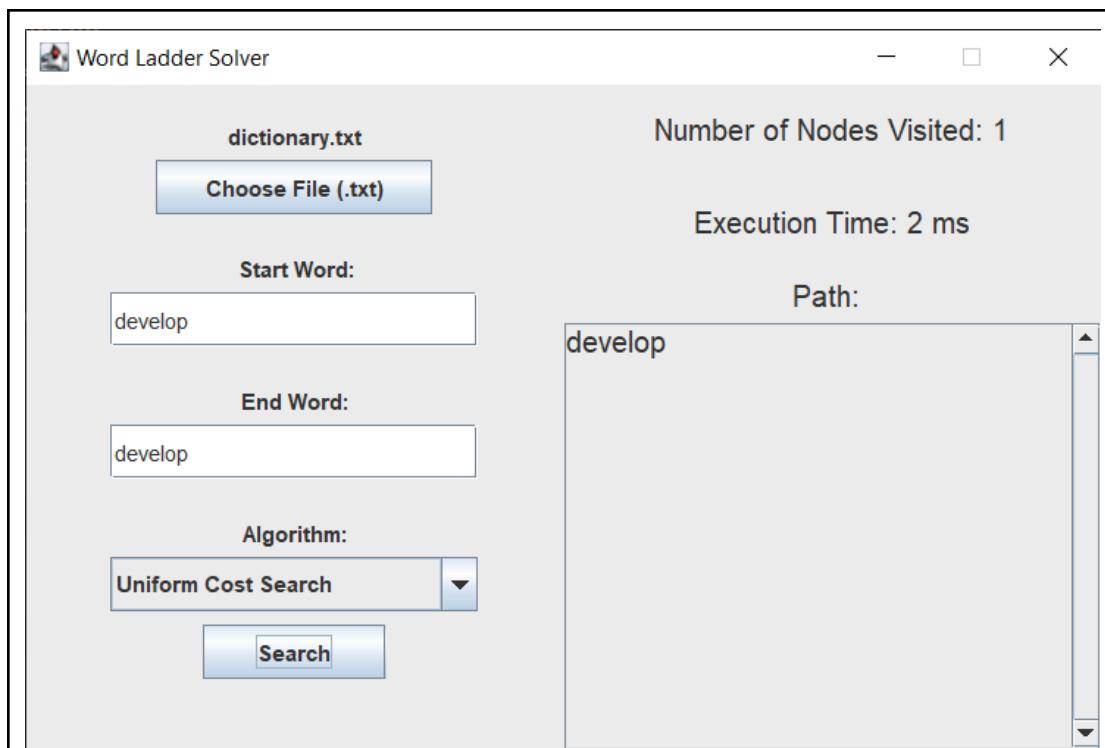


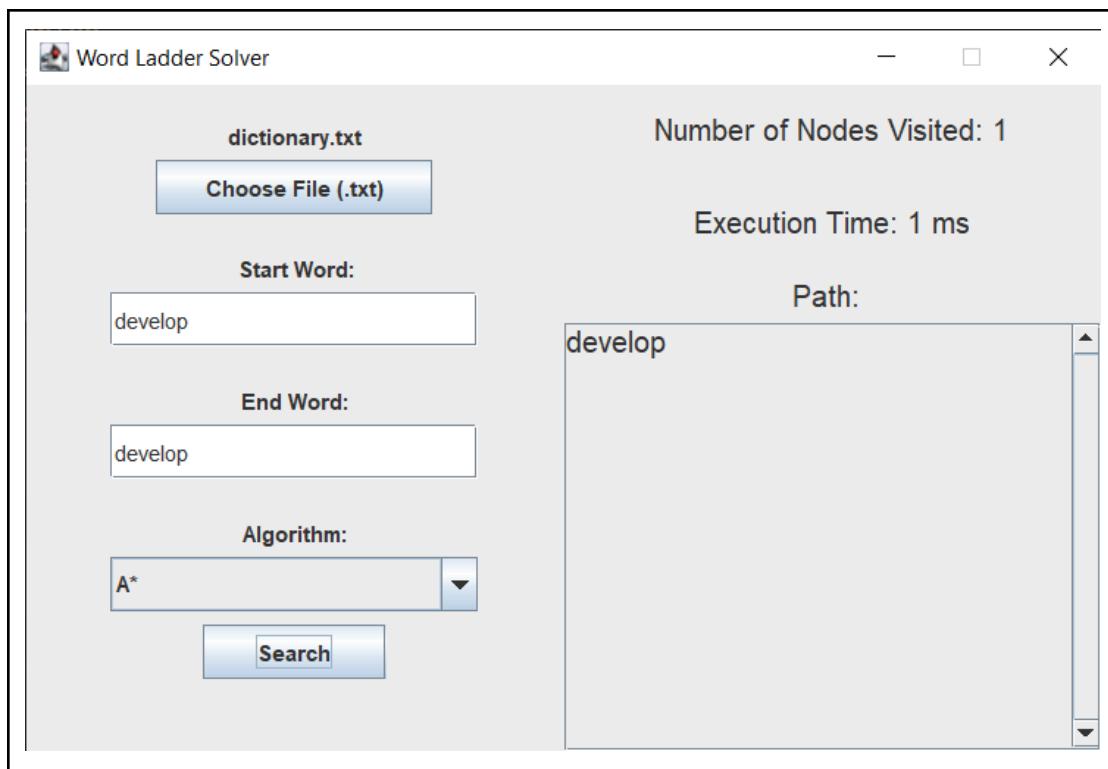
E. *Test Case 5*





F. Test Case 6





IV. Hasil Analisis Perbandingan Solusi Uniform Cost Search, Greedy Best-First Search, dan A*

Algoritma UCS (Uniform Cost Search) dan A* (A-star) selalu menghasilkan solusi yang optimal, sedangkan algoritma GBFS (Greedy Best-First Search) tidak selalu menghasilkan solusi yang optimal. Hal ini ditunjukkan oleh tangkapan layar pada test case 1 dan test case 2. Algoritma GBFS menghasilkan solusi yang lebih panjang dibandingkan algoritma UCS maupun A*.

Berdasarkan test case 1 sampai 3, *execution time* algoritma GBFS paling cepat dibandingkan algoritma lainnya. Algoritma A* sedikit lebih lama dibandingkan algoritma GBFS, sedangkan algoritma UCS jauh lebih lama dibandingkan 2 algoritma lainnya. Hal ini menunjukkan bahwa algoritma GBFS memiliki efisiensi yang paling baik, sedangkan algoritma UCS memiliki efisiensi yang paling buruk.

Berdasarkan test case 1 sampai 3, *number of nodes visited* algoritma GBFS paling sedikit dibandingkan algoritma lainnya. Algoritma A* sedikit lebih banyak dibandingkan algoritma GBFS, sedangkan algoritma UCS jauh lebih banyak dibandingkan 2 algoritma lainnya. Jumlah *node* yang dikunjungi berhubungan dengan banyaknya memori yang diperlukan untuk menyimpan daftar *node* yang pernah dikunjungi. Hal ini menunjukkan bahwa algoritma GBFS membutuhkan memori yang paling sedikit, sedangkan algoritma UCS membutuhkan memori yang paling banyak.

V. Implementasi Bonus

GUI (*Graphical User Interface*) diimplementasikan pada file GUI.java di dalam kelas GUI. Kelas GUI adalah kelas turunan dari kelas JFrame yang disediakan oleh library bawaan bahasa Java. Konstruktor GUI akan membuat semua komponen-komponen pada GUI.

- Method actionPerformed digunakan untuk menangkap interaksi user pada GUI.
- Method getDictionaryFile digunakan untuk mengambil daftar kata dalam format .txt.
- Method search digunakan untuk memanggil algoritma pencarian UCS, GBFS atau A* serta menampilkan hasil pencarian ke GUI.

```
● ○ ● ●
1 import javax.swing.*;
2 import javax.swing.filechooser.FileFilter;
3 import java.awt.*;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.io.IOException;
9 import java.util.List;
10 import java.util.NoSuchElementException;
11 import java.util.Objects;
12
13 class GUI extends JFrame implements ActionListener {
14
15     private final JLabel chooseFileLabel;
16     private final JButton chooseFileButton;
17     private File selectedfile;
18     private final JTextField startWordField;
19     private final JTextField endWordField;
20     private final JComboBox<String> algorithmComboBox;
21     private final JButton searchButton;
22     private final JLabel numberOfNodesVisited;
23     private final JLabel executionTime;
24     private final JPanel resultPanel;
25     private final String baseTextNodesVisited;
26     private final String baseTextExecutionTime;
27
28     public GUI() {
29         setVisible(true);
30         setSize(600, 400);
31         setDefaultCloseOperation(EXIT_ON_CLOSE);
32         setTitle("Word Ladder Solver");
33         setLocationRelativeTo(null);
34         setResizable(false);
35         setLayout(new GridLayout(1, 2));
36
37         JPanel inputPanel = new JPanel();
38         inputPanel.setLayout(new GridLayout(5, 1));
39         add(inputPanel);
40
41         JPanel filePanel = new JPanel();
42         inputPanel.add(filePanel);
43
44         chooseFileLabel = new JLabel("No Dictionary File Selected");
45         chooseFileLabel.setPreferredSize(new Dimension(250, 30));
46         chooseFileLabel.setHorizontalAlignment(JLabel.CENTER);
47         chooseFileLabel.setVerticalAlignment(JLabel.BOTTOM);
48         chooseFileLabel.setOpaque(true);
49         filePanel.add(chooseFileLabel);
50
51         chooseFileButton = new JButton("Choose File (.txt)");
52         chooseFileButton.setPreferredSize(new Dimension(150, 30));
53         chooseFileButton.addActionListener(this);
54         filePanel.add(chooseFileButton);
55
56         JPanel startWordPanel = new JPanel();
57         inputPanel.add(startWordPanel);
58
59         JLabel startWordLabel = new JLabel("Start Word:");
60         startWordLabel.setPreferredSize(new Dimension(250, 30));
61         startWordLabel.setHorizontalAlignment(JLabel.CENTER);
62         startWordLabel.setVerticalAlignment(JLabel.BOTTOM);
63         startWordPanel.add(startWordLabel);
64
65         startWordField = new JTextField();
66         startWordField.setPreferredSize(new Dimension(200, 30));
67         startWordPanel.add(startWordField);
```

```

1 JPanel endWordPanel = new JPanel();
2 inputPanel.add(endWordPanel);
3
4 JLabel endWordLabel = new JLabel("End Word:");
5 endWordLabel.setPreferredSize(new Dimension(250, 30));
6 endWordLabel.setHorizontalAlignment(JLabel.CENTER);
7 endWordLabel.setVerticalAlignment(JLabel.BOTTOM);
8 endWordPanel.add(endWordLabel);
9
10 JTextField endWordField = new JTextField();
11 endWordField.setPreferredSize(new Dimension(200, 30));
12 endWordPanel.add(endWordField);
13
14 JPanel algorithmPanel = new JPanel();
15 inputPanel.add(algorithmPanel);
16
17 JLabel algorithmLabel = new JLabel("Algorithm:");
18 algorithmLabel.setPreferredSize(new Dimension(250, 30));
19 algorithmLabel.setHorizontalAlignment(JLabel.CENTER);
20 algorithmLabel.setVerticalAlignment(JLabel.BOTTOM);
21 algorithmPanel.add(algorithmLabel);
22
23 JComboBox<> algorithmComboBox = new JComboBox<>(new String[]{"Uniform Cost Search", "Greedy Best-First Search", "A*"});
24 algorithmComboBox.setPreferredSize(new Dimension(200, 30));
25 algorithmPanel.add(algorithmComboBox);
26
27 JPanel searchPanel = new JPanel();
28 inputPanel.add(searchPanel);
29
30 JButton searchButton = new JButton("Search");
31 searchButton.setPreferredSize(new Dimension(100, 30));
32 searchButton.addActionListener(this);
33 searchPanel.add(searchButton);
34
35 JPanel outputPanel = new JPanel();
36 outputPanel.setLayout(new BoxLayout(outputPanel, BoxLayout.Y_AXIS));
37 add(outputPanel);
38
39 baseTextNodesVisited = "Number of Nodes Visited: ";
40 JLabel nodesVisited = new JLabel(baseTextNodesVisited);
41 nodesVisited.setAlignmentX(Component.CENTER_ALIGNMENT);
42 nodesVisited.setPreferredSize(new Dimension(0, 50));
43 nodesVisited.setFont(new Font("Arial", Font.PLAIN, 16));
44 outputPanel.add(nodesVisited);
45
46 baseTextExecutionTime = "Execution Time: ";
47 JLabel executionTime = new JLabel(baseTextExecutionTime);
48 executionTime.setAlignmentX(Component.CENTER_ALIGNMENT);
49 executionTime.setPreferredSize(new Dimension(0, 50));
50 executionTime.setFont(new Font("Arial", Font.PLAIN, 16));
51 outputPanel.add(executionTime);
52
53 JLabel pathLabel = new JLabel("Path: ");
54 pathLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
55 pathLabel.setPreferredSize(new Dimension(0, 30));
56 pathLabel.setFont(new Font("Arial", Font.PLAIN, 16));
57 outputPanel.add(pathLabel);
58
59 JPanel resultPanel = new JPanel();
60 resultPanel.setLayout(new BoxLayout(resultPanel, BoxLayout.Y_AXIS));
61
62 JScrollPane resultScroll = new JScrollPane(resultPanel);
63 resultScroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
64 resultScroll.setAlignmentX(Component.CENTER_ALIGNMENT);
65 outputPanel.add(resultScroll);
66
67 }
68

```

```
1  @Override
2  public void actionPerformed(ActionEvent event) {
3      try {
4          if (event.getSource() == chooseFileButton) {
5              getDictionaryFile();
6          } else if (event.getSource() == searchButton) {
7              search();
8          }
9      } catch (Exception exception) {
10         exception.printStackTrace();
11         JOptionPane.showMessageDialog(this, exception.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
12     }
13 }
14
15 private void getDictionaryFile() {
16     JFileChooser fileChooser = new JFileChooser();
17     fileChooser.setCurrentDirectory(new File("test"));
18     fileChooser.setFileFilter(new FileFilter() {
19         @Override
20         public boolean accept(File f) {
21             return f.getName().toLowerCase().endsWith(".txt") || f.isDirectory();
22         }
23
24         @Override
25         public String getDescription() {
26             return "Text Files (*.txt)";
27         }
28     });
29     int response = fileChooser.showOpenDialog(this);
30     if (response == JFileChooser.APPROVE_OPTION) {
31         selectedFile = fileChooser.getSelectedFile();
32         chooseFileLabel.setText(selectedFile.getName());
33     }
34 }
35 }
```

```
1  private void search() throws IOException {
2      resultPanel.removeAll();
3      resultPanel.revalidate();
4      resultPanel.repaint();
5      resultPanel.paintImmediately(resultPanel.getBounds());
6
7      if (selectedFile == null) {
8          throw new FileNotFoundException("No dictionary file selected");
9      }
10     WordList wordList = new WordList(selectedFile);
11
12     String startWord = startWordField.getText().toLowerCase();
13     if (wordList.isWordNotExist(startWord)) {
14         throw new NoSuchElementException("Start word does not exist in the dictionary");
15     }
16
17     String endWord = endWordField.getText().toLowerCase();
18     if (wordList.isWordNotExist(endWord)) {
19         throw new NoSuchElementException("End word does not exist in the dictionary");
20     }
21
22     if (startWord.length() != endWord.length()) {
23         throw new IllegalArgumentException("Start word and end word length must be the same");
24     }
25
26     String loadAndGenerateWordListMessage = "Loading and generating word list";
27     System.out.println(loadAndGenerateWordListMessage);
28
29     wordList.generateWordToWordsDifferBy1Char(startWord.length());
30     String selectedAlgorithm = (String) algorithmComboBox.getSelectedItem();
31
32     List<String> result;
33     int numOfNodesVisited;
34
35     String searchingStartedMessage = "Searching has started";
36     System.out.println(searchingStartedMessage);
37
38     long startTime = System.nanoTime();
39
40     switch (Objects.requireNonNull(selectedAlgorithm)) {
41         case "Uniform Cost Search":
42             UniformCostSearch uniformCostSearch = new UniformCostSearch();
43             result = uniformCostSearch.getPath(startWord, endWord, wordList);
44             numOfNodesVisited = uniformCostSearch.getNumOfNodesVisited();
45             break;
46         case "Greedy Best-First Search":
47             GreedyBestFirstSearch greedyBestFirstSearch = new GreedyBestFirstSearch();
48             result = greedyBestFirstSearch.getPath(startWord, endWord, wordList);
49             numOfNodesVisited = greedyBestFirstSearch.getNumOfNodesVisited();
50             break;
51         case "A*":
52             AStar aStar = new AStar();
53             result = aStar.getPath(startWord, endWord, wordList);
54             numOfNodesVisited = aStar.getNumOfNodesVisited();
55             break;
56         default:
57             throw new IllegalArgumentException("Invalid algorithm selection");
58     }
59
60     long endTime = System.nanoTime();
61     long timeDifference = endTime - startTime;
62
63     String searchingEndedMessage = "Searching has ended";
64     System.out.println(searchingEndedMessage);
65
66     System.out.println(baseTextNodesVisited + numOfNodesVisited);
67     System.out.println(baseTextExecutionTime + timeDifference / 1000000 + " ms");
68     System.out.println("Path: " + result);
69
70     System.out.println("-----");
71
72     numOfNodesVisited.setText(baseTextNodesVisited + numOfNodesVisited);
73     executionTime.setText(baseTextExecutionTime + timeDifference / 1000000 + " ms");
74
75     for (String word : result) {
76         JLabel wordLabel = new JLabel(word);
77         wordLabel.setText(word);
78         wordLabel.setFont(new Font("Arial", Font.PLAIN, 16));
79         resultPanel.add(wordLabel);
80     }
81 }
82 }
83 }
```

Lampiran

Pranala repository GitHub: https://github.com/bastianhs/Tucil3_13522034

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	✓	