LAPORAN TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

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

Disusun oleh:

Marvin Scifo Y. Hutahaean 13522110

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

BAB I

Analisis Algoritma

1.1 Uniform Cost Search (UCS)

UCS adalah algoritma pencarian yang melibatkan pengunjungan sebuah simpul berdasarkan atribut simpul yang paling kecil (ditandai dengan f(n)). Pada UCS, fungsi evaluasi yang ditandai oleh f(n) = g(n) yang merupakan nilai dari simpul/kata itu sendiri. Untuk menyelesaikan masalah Word Ladder, g(n) yang digunakan adalah kata ke-n yang akan dikunjungi pada solusi. Contohnya jika best, lest, last adalah solusi dari best => last, maka g(n) masing-masing kata adalah 1, 2, dan 3. Dengan menggunakan algoritma ini, secara otomatis kata-kata yang akan ditelusuri terlebih dahulu adalah kata-kata yang dikunjungi kedua kalinya (kata pertama selalu sama) dan akan ditelusuri kata-kata yang dikunjungi ketiga kalinya dan seterusnya. Beginilah tahapan dari penggunaan UCS pada penyelesaian word ladder.

1. Kata pertama dijadikan input untuk dicari daftar kata-kata berbahasa Inggris yang bisa didapatkan jika hanya mengganti salah satu huruf dari kata pertama tersebut.
2. Kata selanjutnya yang akan dicari akan memiliki g(n) = 2. Pemilihannya bebas dan ketika menelusuri kata tersebut, kata tersebut memberikan informasi terkait daftar kata-kata berbahasa Inggris tersebut.
3. Jika kata tidak sama dengan input kata terakhir dari pengguna, kata berikutnyalah yang akan dicari dan akan memiliki g(n) yang lebih 1 dibandingkan dengan kata induk.
4. Proses ini dilakukan sampai kata yang sedang ditelusuri tersebut sama dengan input kata terakhir dari pengguna.

Penggunaan algoritma UCS seperti ini dengan g(n) yang merepresentasikan kata ke-n yang dikunjungi bisa disamakan dengan algoritma BFS dan g(n) yang bisa disamakan dengan kedalaman sebuah simpul. Hal ini disebabkan UCS yang selalu akan menelusuri simpul dengan g(n) yang terkecil jadi jika divisualisasikan, UCS sebenarnya sedang melakukan penelusuran dengan simpul dengan kedalaman yang kecil terlebih dahulu.

1.2 Greedy Best First Search (GBFS)

Greedy Best First Search (GBFS) adalah sebuah algoritma yang memprioritaskan *heuristic value* yang paling ekstrim sehingga pada setiap tahapan dan dilakukan sampai solusi ditemukan. Pada algoritma ini, backtrack tidak dilakukan sehingga ketika simpul terdapat di kata ke-n pada solusi, penelusuran tidak akan bisa dilakukan pada kata ke-(n-1) pada solusi. Pada GBFS, fungsi evaluasi f(n) = h(n) yang merupakan jarak dari kata yang ditelusuri ke kata tujuan. Fungsi h(n) didapatkan dengan mencari berapa instansi huruf dari setiap kata berbeda dengan setiap kata lainnya juga. Misalnya, book dan rest memiliki nilai 4 kata setiap huruf dari book dan rest adalah berbeda sehingga mereka memiliki nilai 4. Beginilah tahapan penggunaan GBFS dalam permainan Word Ladder.

1. Pada simpul pertama dicari semua kemungkinan kata yang jika kata pertama diganti satu huruf saja akan menjadi sama dengan kemungkinan kata selanjutnya tersebut.
2. Cari kata dengan h(n) yang paling kecil. Jika terdapat yang sama, pilih saja secara bebas.

3. Proses dilakukan sampai solusi didapat atau saat pencarian tidak bisa ditemukan solusinya.

Fakta bahwa GBFS memiliki kemungkinan bahwa solusinya bisa tidak didapatkan menjadi bukti konkrit bahwa algoritma bukanlah algoritma yang terbaik untuk mencari solusi optimal dalam permainan Word Ladder. Hal ini bisa terjadi karena saat pencarian kata dengan n huruf, kata-katanya sudah habis atau tidak bisa dicapai sehingga program mengirim pesan bahwa solusi tidak bisa ditemukan.

1.3 A-Star (A*)

A-Star (A*) adalah algoritma yang menggunakan kombinasi dari algoritma GBFS dan algoritma UCS. Tidak seperti GBFS, A* menggunakan backtrack sehingga kemungkinan solusi tidak didapat menjadi lebih kecil. A* melakukan pengunjungan ke sebuah simpul yang jumlah n dikunjungi dan beda dengan kata tujuan yang paling kecil. Pada A*, fungsi evaluasi yang ditandai dengan f(n) = g(n)+h(n). Fungsi g(n) menandakan jumlah n dikunjungi jika kata tersebut merupakan komponen solusi. Fungsi h(n) menandakan jumlah huruf yang berbeda antar 2 kata. Beginilah tahapan penyelesaian program menggunakan A*.

1. Pada simpul pertama dicari semua kemungkinan kata yang jika kata pertama diganti satu huruf saja akan menjadi sama dengan kemungkinan kata selanjutnya tersebut.
2. Cari kata dengan g(n)+h(n) yang paling kecil. Jika terdapat yang sama, pilih saja secara bebas.
3. Proses dilakukan sampai solusi didapat atau saat pencarian tidak bisa ditemukan solusinya.

Menggunakan algoritma A* akan menghasilkan solusi yang jumlah katanya selalu sama dengan algoritma UCS yang notabene memeriksa semua kemungkinan solusi dari beberapa g(n) secara berurutan (Solusi optimal pastinya didapat). Dari hal yang didapat ini, A* bisa dikatakan memiliki heuristik yang admissible karena solusinya selalu memberikan solusi yang optimal. Berdasarkan beberapa tes, algoritma A* lebih efisien karena mengunjungi node yang lebih sedikit jika dibandingkan dengan UCS meskipun jumlah kata pada solusi yang sama.

BAB II

Source Code

2.1 Source Code

1. Engine

   Class ini bertugas untuk menjalankan garis besar dari program yaitu mengambil informasi dari interface dan membawanya ke search untuk dilakukan pencarian dan

memberikan informasi jawaban kembali ke interface. Beginilah implementasi dari class Engine.

```
import java.util.*;
import java.util.HashMap;

public class Engine {
    private HashMap<String,Boolean> dictionary;
    private Node start;
    private String end;
    private Node resultado;
    private String[] message;
    private long exec;
    private long runtime;
    private int visited;

    public Engine() {
        this.start = new Node();
        this.end = "";
    }
    public String[] getMessage() {
        return message;
    }
    public String getMessageGet(int i) {
        return message[i];
    }
    public long getExec() {
        return exec;
    }
    public long getMem() {
        return runtime;
    }
    public int getVisited() {
        return visited;
    }
    public void Start(String start, String end, int choice) {
        this.dictionary = FileController.readToList();
        this.resultado = new Node();

        this.start.setCost(0);
        this.start.setInfo(start);
        this.end = end;
        PriorityQueue<Node> emptyList = new PriorityQueue<Node>(new
NodeComparator());

        if (!(this.dictionary.get(start) != null &&
this.dictionary.get(end) != null)) {
            message = new String[1];
            message[0] = "Please insert proper english word!";
            return;
        }

        long startTime = System.currentTimeMillis();
        Search results = new Search(this.dictionary);
        switch(choice) {
            case 1:
                this.resultado = results.SearchSolution(this.start,
this.end, this.dictionary, emptyList, "UCS");
                break;
            case 2:
```

```
                this.resultado = results.SearchSolution(this.start,
this.end, this.dictionary, emptyList, "GBFS");
                break;
            case 3:
                this.resultado = results.SearchSolution(this.start,
this.end, this.dictionary, emptyList, "AS");
                break;
            default:
                break;
        }
        Runtime runtime = Runtime.getRuntime();
        long endTime = System.currentTimeMillis();
        this.message = this.resultado.moveToList();
        this.exec = endTime-startTime;
        this.runtime =
(runtime.totalMemory()-runtime.freeMemory())/(1024L*1024L);
        this.visited = results.getVisited();
    }
}
```

2. FileController

   Class ini bertugas untuk mengambil informasi terkait kata-kata bahasa inggris di file root pengerjaan. Ini dilakukan agar program memiliki referensi yang tepat untuk melakukan pemeriksaan terkait kata-kata yang dicari dan ditelusuri. Beginilah implementasi dari kelas FileController.

```
import java.util.*;
import java.io.*;
import java.util.HashMap;

public class FileController {
    public static HashMap<String,Boolean> readToList() {
        HashMap<String,Boolean> temp = new HashMap<String,Boolean>();
        String placer;
        try {
            File diction = new File("../dictionary(1).txt");
            Scanner reader = new Scanner(diction);
            while (reader.hasNextLine()) {
                placer = reader.nextLine();
                temp.put(placer,false);
            }
            reader.close();
            return temp;
        } catch(FileNotFoundException e) {
            System.out.println("File not found");
            return temp;
        }
    }
}
```

3. Interface

   Class ini bertugas untuk membuat GUI dari program yang dibuat dan menerima input dari pengguna. Detail lebih lanjut akan dijelaskan pada bagian Bonus. Beginilah implementasi dari class Interface.

```
import javax.swing.*;
```

```
import java.awt.event.*;
import java.awt.*;

public class Interface extends JFrame implements ItemListener {
    static JFrame f;

    // label
    static JLabel l, l1, l2, l3;

    // combobox
    static JComboBox c1;

    // input
    static JTextField t1,t2;

    // button
    static JButton b;

    static JPanel resultPanel;

    public Interface() {
        f = new JFrame("Word Solver");
        f.setLayout(new FlowLayout());

        String[] option = {"Uniform Cost Search", "Greedy Best First
Search", "A*"};

        t1 = new JTextField(50);
        t2 = new JTextField(50);
        c1 = new JComboBox(option);
        c1.addItemListener(this);
        l = new JLabel("Insert the first word");
        l1 = new JLabel("Insert the final word");
        l2 = new JLabel("Select your method");
        l3 = new JLabel("Chosen algorithm: Uniform Cost Search");
        l.setForeground(Color.red);
        l1.setForeground(Color.blue);
        JPanel p = new JPanel(new GridLayout(0, 1));
        b = new JButton("Search");
        p.add(l);
        p.add(t1);
        p.add(l1);
        p.add(t2);
        p.add(l2);
        p.add(c1);
        p.add(l3);
        p.add(b);
        f.add(p);

        // Initialize the result panel
        resultPanel = new JPanel(new BorderLayout());
        resultPanel.setPreferredSize(new Dimension(350, 400));
        f.add(resultPanel);

        f.setSize(800, 700);
        f.setVisible(true);
        f.setLocationRelativeTo(null);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close
terminal when pressing x
```

```
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String starte = t1.getText();
                String ende = t2.getText();
                if (starte.length() == ende.length()) {
                    Engine eng = new Engine();
                    if (c1.getSelectedItem().equals("Uniform Cost
Search")) {
                        eng.Start(starte, ende, 1);
                    } else if (c1.getSelectedItem().equals("Greedy
Best First Search")) {
                        eng.Start(starte, ende, 2);
                    } else {
                        eng.Start(starte, ende, 3);
                    }
                    DefaultListModel<String> listModel = new
DefaultListModel<>();
                    if (!(eng.getMessageGet(0).equals("Solution not
found") || eng.getMessageGet(0).equals("Please insert proper english
word!"))) {
                        listModel.addElement("Execution time: " +
Long.toString(eng.getExec()) + " ms");
                        listModel.addElement("Memory taken:
"+eng.getMem()+" MB");
                        listModel.addElement("Score: " +
(eng.getMessage().length-1));
                    }
                    listModel.addElement("Words visited:
"+(eng.getVisited()));
                    for (int i = 0; i < eng.getMessage().length; i++)
{
                        listModel.addElement(eng.getMessageGet(i));
                    }
                    JList<String> li = new JList<>(listModel); // Use
listModel instead of eng.getMessage()

li.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                    JScrollPane scrollPane = new JScrollPane(li);

                    // Remove previous search results
                    resultPanel.removeAll();
                    resultPanel.add(scrollPane, BorderLayout.CENTER);

                    // Refresh layout
                    f.revalidate();
                    f.repaint();
                } else {
                    return;
                }
            }
        });
    }

    public void itemStateChanged(ItemEvent e) {
        if (e.getSource() == c1) {
            l3.setText("Chosen algorithm: "+c1.getSelectedItem());
        }
    }
}
```

4. Main

   Class ini digunakan untuk menjalankan program Java. Beginilah implementasi dari class Main.

```
public class Main {
    public static void main(String[] args) {
        new Interface();
    }
}
```

5. Node

   Class ini digunakan sebagai tempat penyimpanan solusi. Di dalam Node terdapat informasi berupa kata, nilai dari Node tersebut dan nilai sebelum dari Node tersebut. Beginilah implementasi dari Class Node.

```
import java.util.ArrayList;

public class Node {
    private Node prev;
    private String info;
    private int cost;

    public Node() {
        this.info = "";
        this.cost = 0;
    }

    public Node(String info, Node prev, int cost) {
        this.info = info;
        this.prev = prev;
        this.cost = cost;
    }

    public Node(final Node other) {
        this.info = other.info;
        this.prev = other.prev;
        this.cost = other.cost;
    }

    public Node getPrev() {
        return this.prev;
    }

    public void setPrev(Node prev) {
        this.prev = prev;
    }

    public String getInfo() {
        return this.info;
    }

    public void setInfo(String info) {
        this.info = info;
    }

    public int getCost() {
        return this.cost;
    }
```

```
    public void setCost(int cost) {
        this.cost = cost;
    }

    public void printNodes() {
        ArrayList<String> container = new ArrayList<>();
        Node containerNode = new Node(this);
        container.add(info);
        while (containerNode.prev != null) {
            containerNode = containerNode.prev;
            container.add(containerNode.info);
        }
        for (int i = container.size()-1; i >= 0; i--) {
            System.out.print(container.get(i)+" ");
        }
        System.out.println();
    }

    public int getSize() {
        int count = 0;
        Node tracker = new Node(this);
        while (tracker != null) {
            count++;
            tracker = tracker.getPrev();
        }
        return count;
    }

    public boolean compareList(String teste) {
        boolean checker = false;
        Node tracker = new Node(this);
        while(tracker != null) {
            if (teste.equals(tracker.getInfo())) {
                checker = true;
                return checker;
            }
            tracker = tracker.getPrev();
        }
        return checker;
    }

    public String[] moveToList() {
        String[] temp = new String[this.getSize()];
        int sizeRef = this.getSize()-1;
        int i = 0;
        Node tracker = new Node(this);
        while (tracker != null) {
            temp[sizeRef-i] = tracker.getInfo();
            tracker = tracker.getPrev();
            i++;
        }
        return temp;
    }
}
```

6. NodeComparator

   Class ini bertugas sebagai standar perbandingan antara satu Node dengan lainnya

dengan menggunakan Priority Queue. Beginilah implementasi NodeComparator pada program ini.

```
import java.util.Comparator;

public class NodeComparator implements Comparator<Node> {
    public int compare(Node a, Node b) {
        if (a.getCost() < b.getCost()) {
            return -1;
        } else if (a.getCost() > b.getCost()) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

7. Search

   Class ini bertugas sebagai pelaku pencarian solusi dari permainan Word Ladder baik dengan menggunakan UCS, GBFS, atau A*. Pada program ini, pencarian langsung menggunakan fungsi yang sama karena perbedaan algoritma yang tidak terlalu signifikan sehingga pembeda hanya dilakukan pada beberapa aspek saja dan pembedaan juga dilakukan pada kelas lain yaitu kelas Util. Beginilah implementasi Class Search.

```
import java.util.HashMap;
import java.util.PriorityQueue;

public class Search {
    private HashMap<String,Boolean> dictionary;

    private int visited;

    public Search(final HashMap<String,Boolean> dictionary) {
        this.dictionary = dictionary;
    }

    public int getVisited() {
        return visited;
    }

    public Node SearchSolution(Node start, String end,
HashMap<String,Boolean> dict, PriorityQueue<Node> cont, String
choice) {
        Node gottenCostElement;
        Node tracker;
        PriorityQueue<Node> temp;
        int sizeRef;
        while (true) {
            if (cont.peek() == null) {
                gottenCostElement = new Node(start);
            } else {
                gottenCostElement = cont.poll();
                visited++;
            }
            temp = Util.getPossibleValues(gottenCostElement,
this.dictionary, end, choice);
```

```
            sizeRef = temp.size();
            if (choice.equals("GBFS")) {
                cont.removeAll(cont);
            }
            for (int i = 0; i < sizeRef; i++) {
                tracker = temp.poll();
                if (tracker != null) {
                    cont.add(tracker);
                }
                if (tracker.getInfo().equals(end)) {
                    return tracker;
                }
            }
            if (cont.size() <= 0) {
                return new Node("Solution not found", null, 0);
            }
        }
    }
}
```

8. Util

Class ini bertugas untuk membantu program mempermudah tugas mereka sehingga ketika terdapat masalah yang perlu diselesaikan dengan fungsi/prosedur yang kompleks, pengguna hanya perlu memanggil kembali fungsi/prosedur tersebut. Beginilah implementasi dari Class Util.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.PriorityQueue;

public class Util {
    public static boolean checkAvail(ArrayList<String> liste, String
teste) {
        boolean checker = false;
        for (int i = 0; i < liste.size(); i++) {
            if (teste.equals(liste.get(i))) {
                checker = true;
                return checker;
            }
        }
        return checker;
    }

    public static boolean checkAvailCostly(Node liste, String teste)
{
        boolean checker = false;
        Node tracker = new Node(liste);
        while(tracker != null) {
            if (tracker.getInfo().equals(teste)) {
                checker = true;
                return checker;
            }
            tracker = tracker.getPrev();
        }
        return checker;
    }

    public static int mapChar(char a) {
```

```
        return (a - 97);
    }

    public static char mapInt(int a) {
        return (char) ('a' + a);
    }

    public static int find_cost(String a, String b) {
        int count = 0;
        if (a.length() <= b.length()) {
            for (int i = 0; i < a.length(); i++) {
                if (a.charAt(i) != b.charAt(i)) {
                    count++;
                }
            }
        }
        return count;
    }

    public static Node smallestDistance(ArrayList<Node> liste) {
        System.out.println(liste.size());
        Node temp = liste.get(0);
        for (int i = 0; i < liste.size(); i++) {
            if (liste.get(i).getCost() < temp.getCost()) {
                temp = liste.get(i);
            }
        }
        return temp;
    }

    public static boolean isInTheEnd(ArrayList<Node> liste, String
teste) {
        boolean checker = false;
        for (int i = 0; i < liste.size(); i++) {
            if (liste.get(i).compareList(teste)) {
                    checker = true;
                    return checker;
            }
        }
        return checker;
    }

    public static PriorityQueue<Node> getPossibleValues(Node word,
HashMap<String,Boolean> dict, String end, String choice) {
        String temp = word.getInfo();
        PriorityQueue<Node> temp_list = new PriorityQueue<>(new
NodeComparator());
        Node tracker;
        int cost = 0;
        for (int i = 0; i < temp.length(); i++) {
            for (int j = 0; j < 26; j++) {
                temp = temp.substring(0, i) + mapInt(j) +
temp.substring(i + 1);
                if (dict.get(temp) != null && !dict.get(temp) &&
!word.getInfo().equals(temp) && word.getInfo() != null) {
                    if (choice.equals("UCS")) {
                        cost = word.getSize()+1;
                    } else if (choice.equals("GBFS")) {
                        cost = find_cost(end, temp);
                    } else if (choice.equals("AS")) {
```

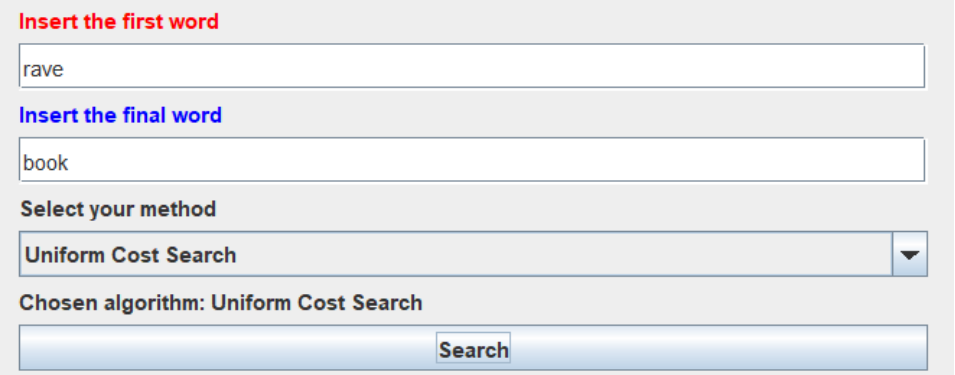
```
                        cost = word.getSize()+1+find_cost(end, temp);
                    }
                    tracker = new Node(temp, word, cost);
                    dict.put(temp, true);
                    temp_list.add(tracker);
                }
            }
            temp = word.getInfo();
        }
        return temp_list;
    }

    public static ArrayList<Node> toList(PriorityQueue<Node> pq) {
        PriorityQueue<Node> pq2 = new PriorityQueue<Node>(pq);
        ArrayList<Node> liste = new ArrayList<>();
        while (pq2.size() > 0) {
            Node n = new Node();
            n = pq2.poll();
            liste.add(n);
        }
        return liste;
    }

    public static void printList(ArrayList<Node> liste) {
        for (int i = 0; i < liste.size(); i++) {
            System.out.print(liste.get(i).getInfo()+" ");
        }
        System.out.println();
    }
}
```

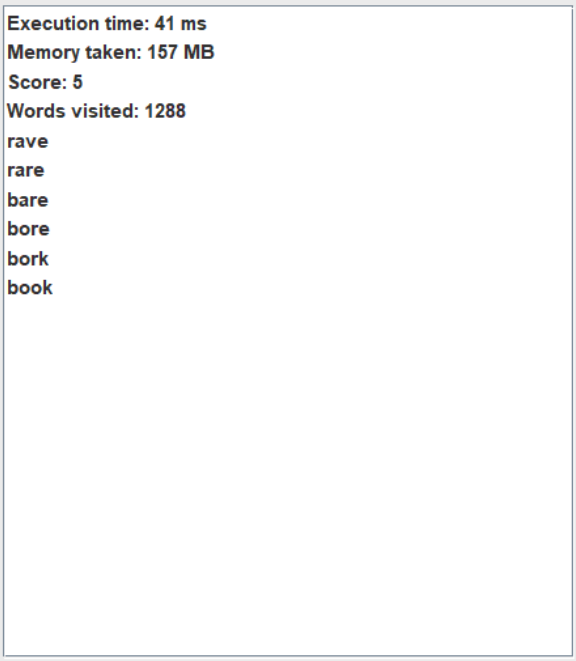
BAB III

Input dan Output

3.1 Uniform Cost Search (UCS)

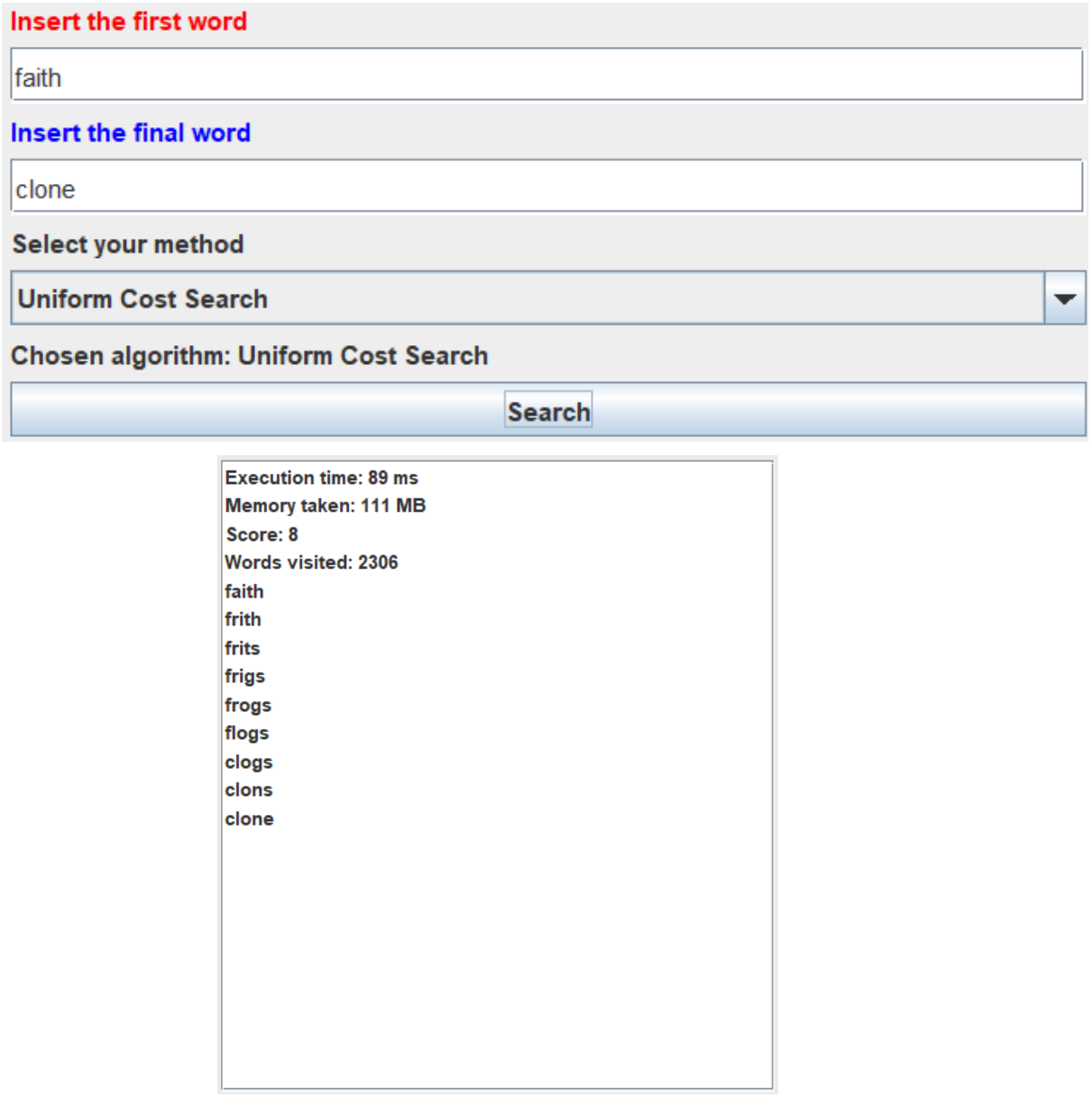
1. Rave => Book





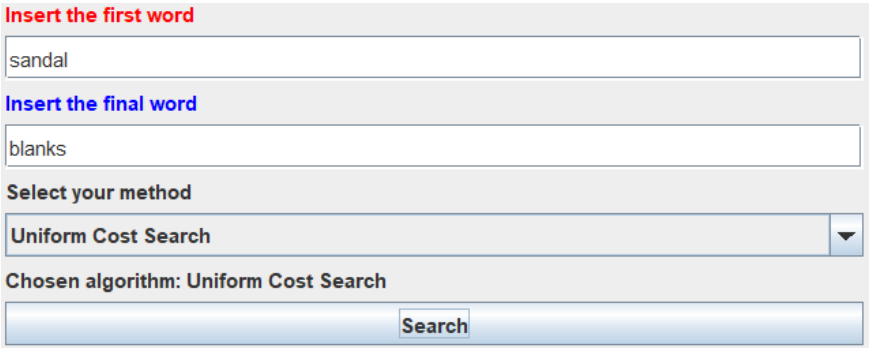
Gambar 3.1.1 - Rave => Book

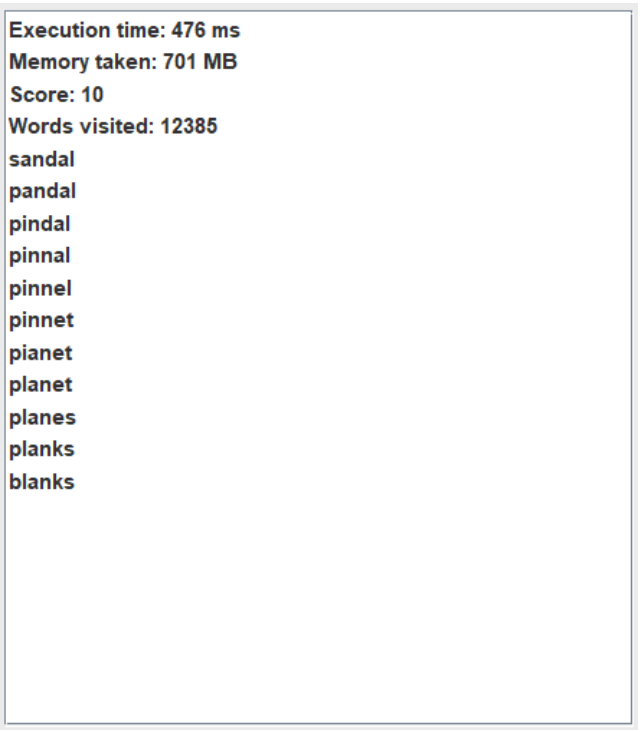
2. Faith => Clone



Gambar 3.1.2 - Faith => Clone

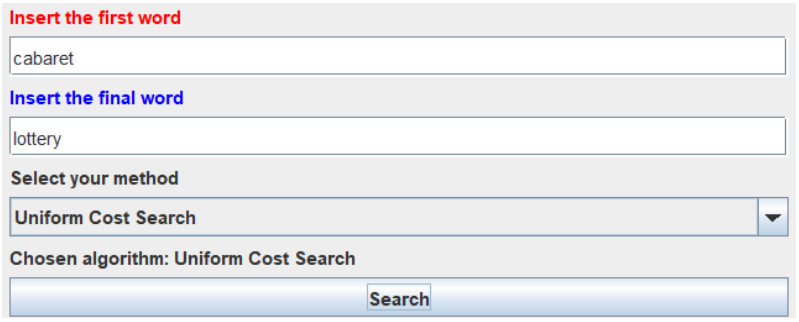
3. Sandal => Blanks

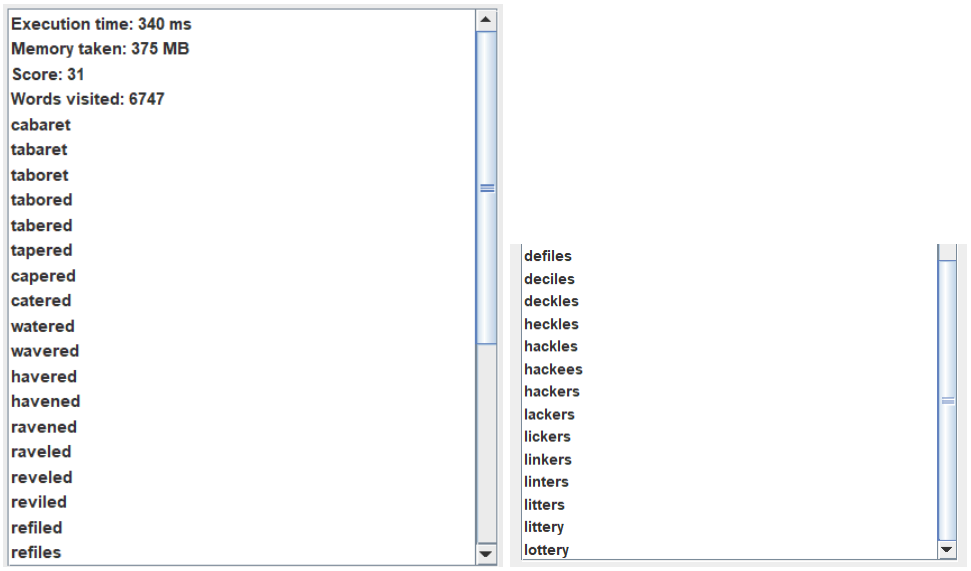




Gambar 3.1.3 - Sandal => Blanks

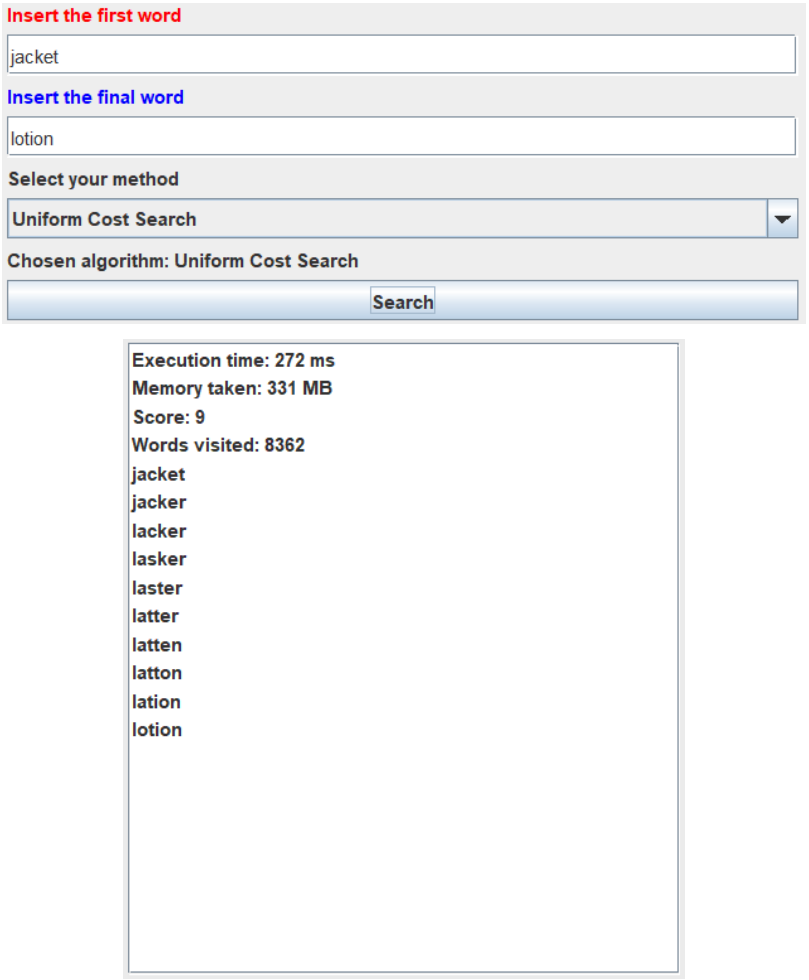
4. Cabaret => Lottery





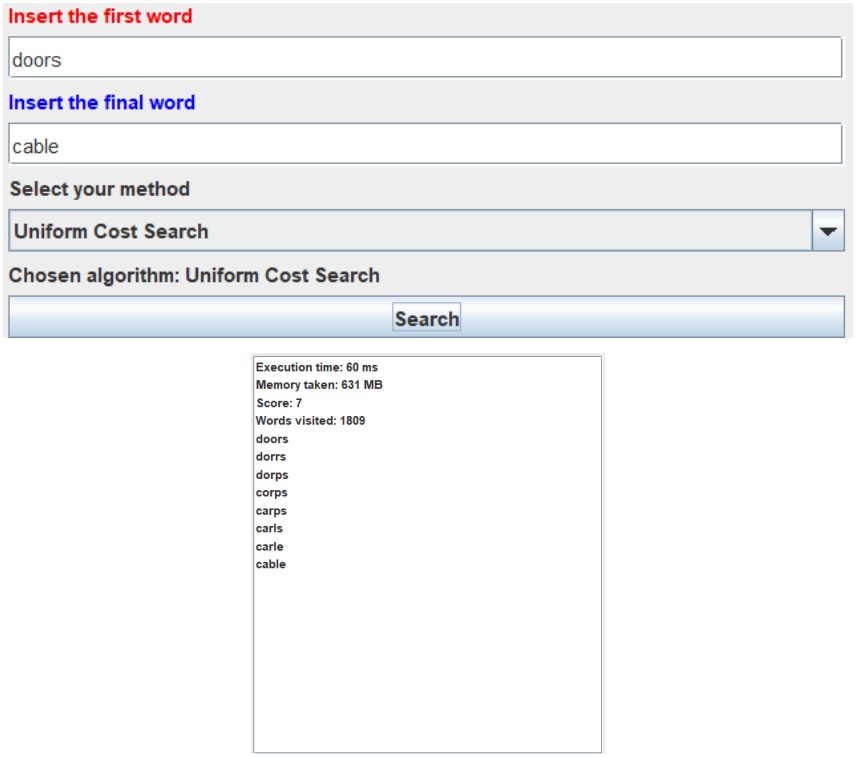
Gambar 3.1.4 - Cabaret => Lottery

5. Jacket => Lotion



Gambar 3.1.5 - Jacket => Lotion

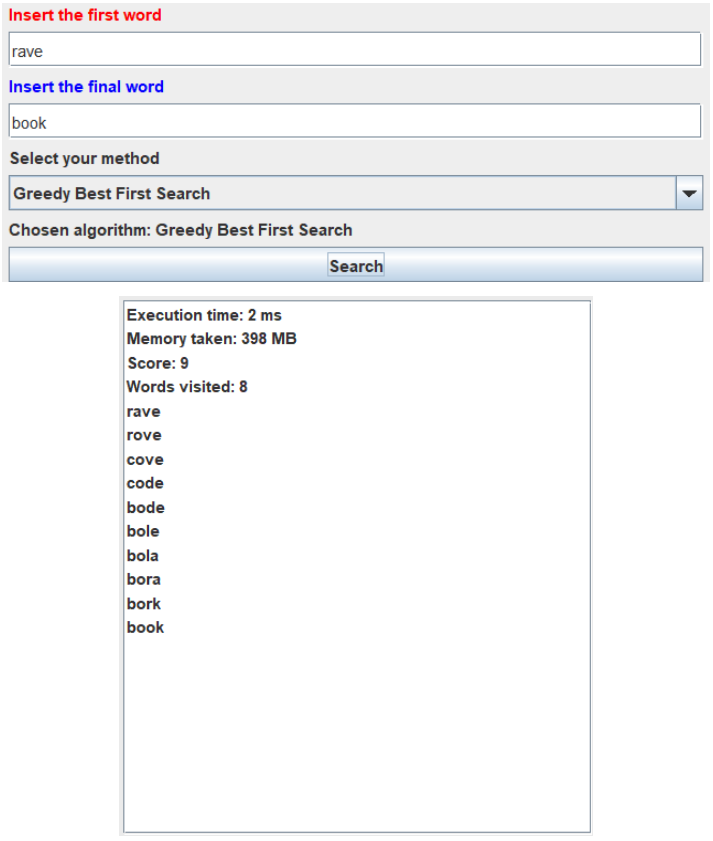
6. Doors => Cable



Gambar 3.1.6 - Doors -> Cable

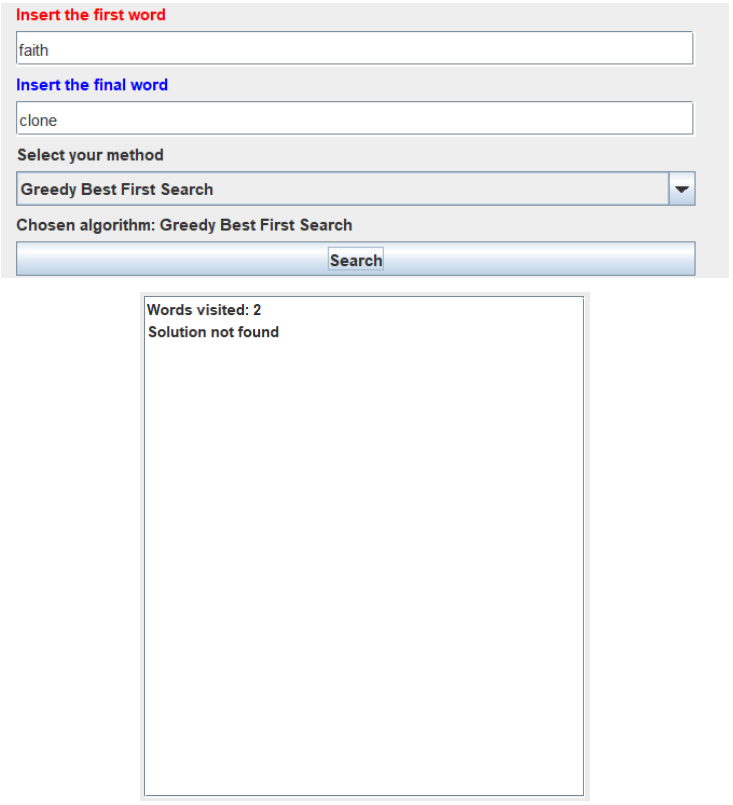
3.2 Greedy Best First Search (GBFS)

1. Rave => Book



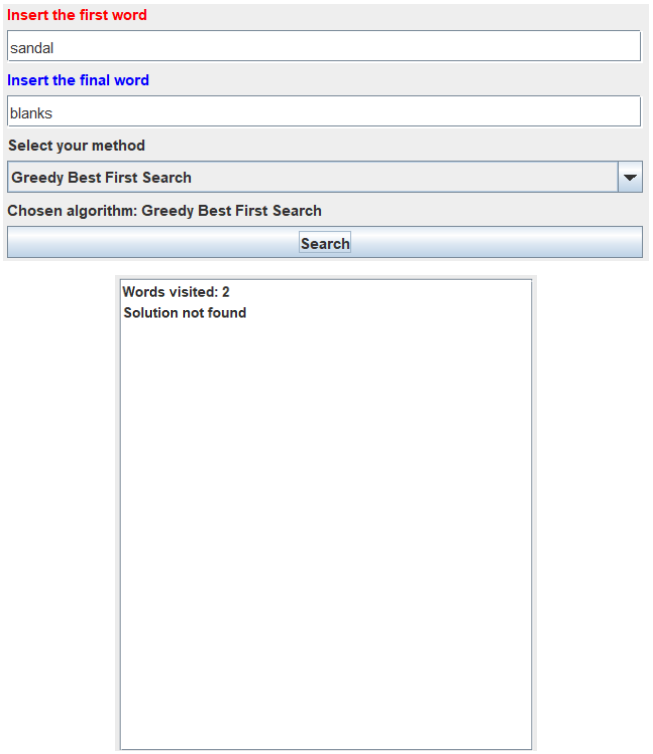
Gambar 3.2.1 - Rave => Book

2. Faith => Clone



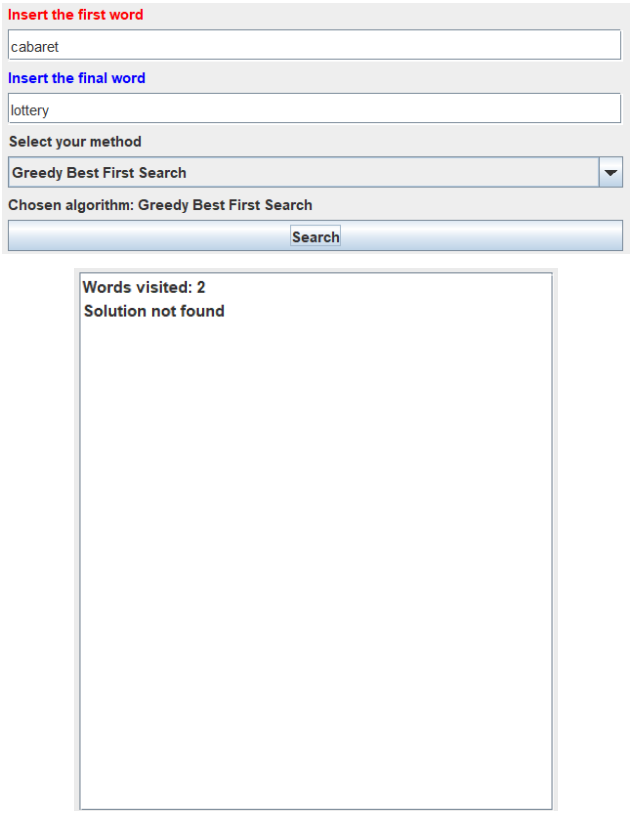
Gambar 3.2.2 - Faith => Clone

3. Sandal => Blanks



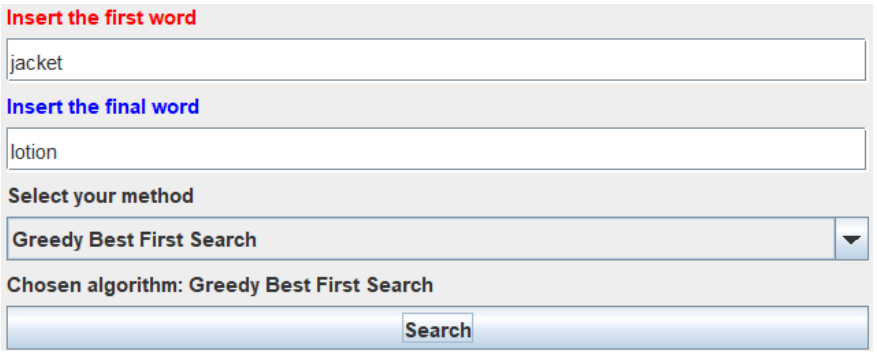
Gambar 3.2.3 - Sandal => Blanks

4. Cabaret => Lottery



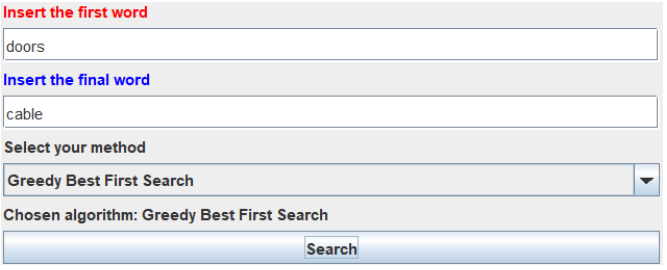
Gambar 3.2.4 - Cabaret => Lottery

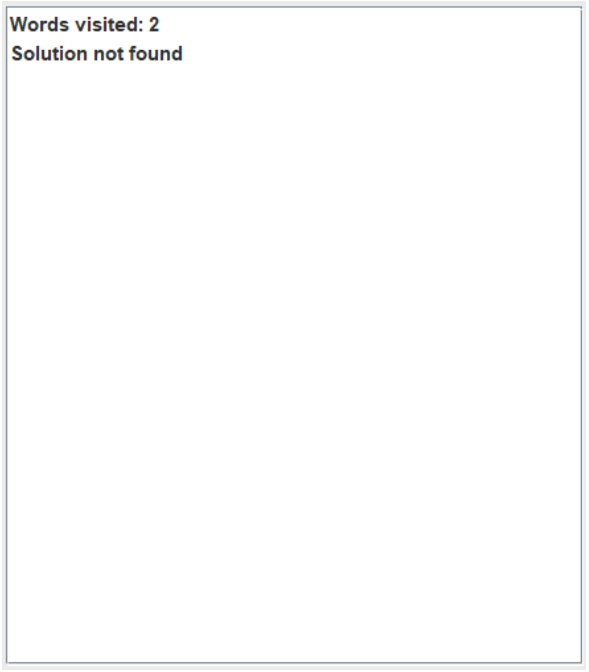
5. Jacket => Lotion



Gambar 3.2.5 - Jacket & Shallow

6. Doors => Cable

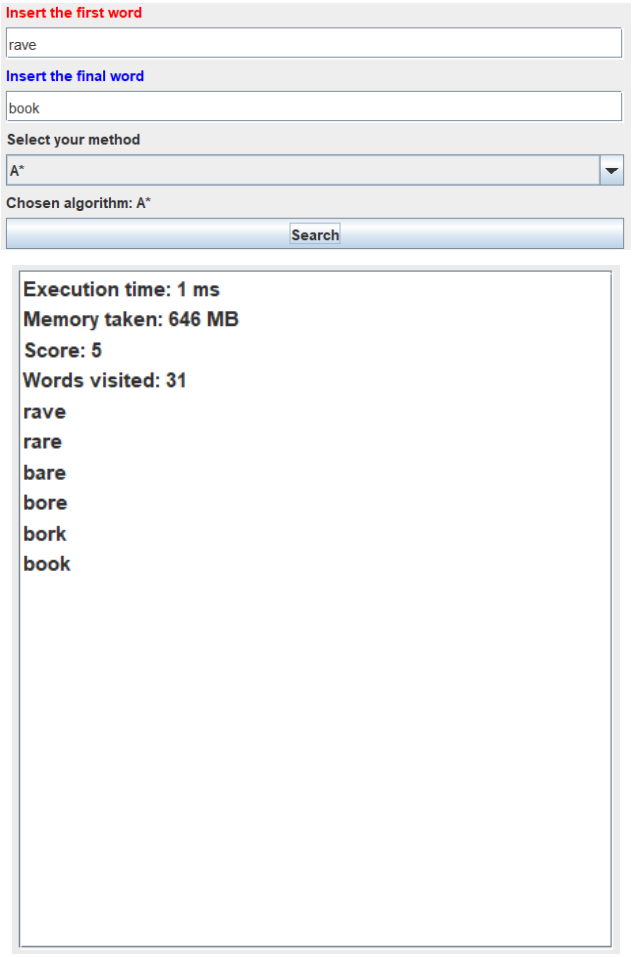




Gambar 3.2.6 - Jacket & Shallow

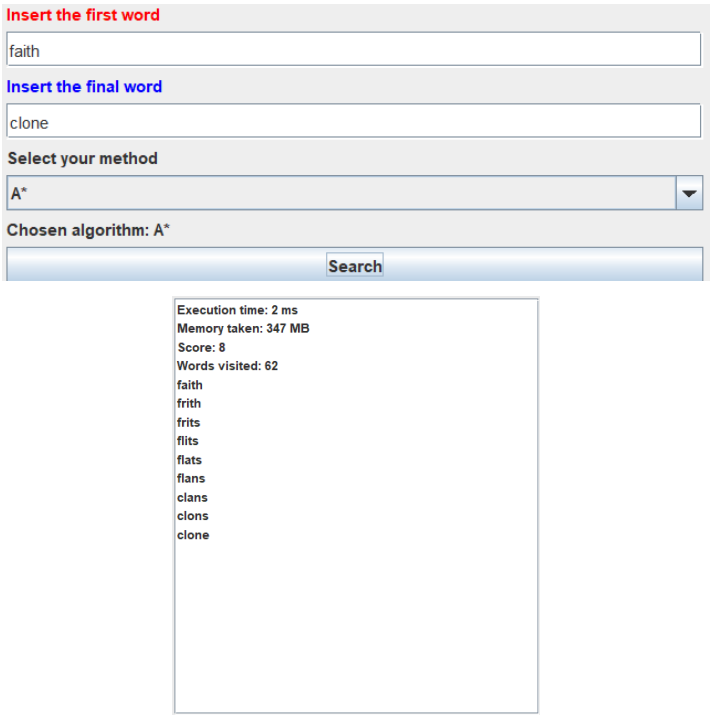
3.3 A-Star (A*)

1. Rave => Books



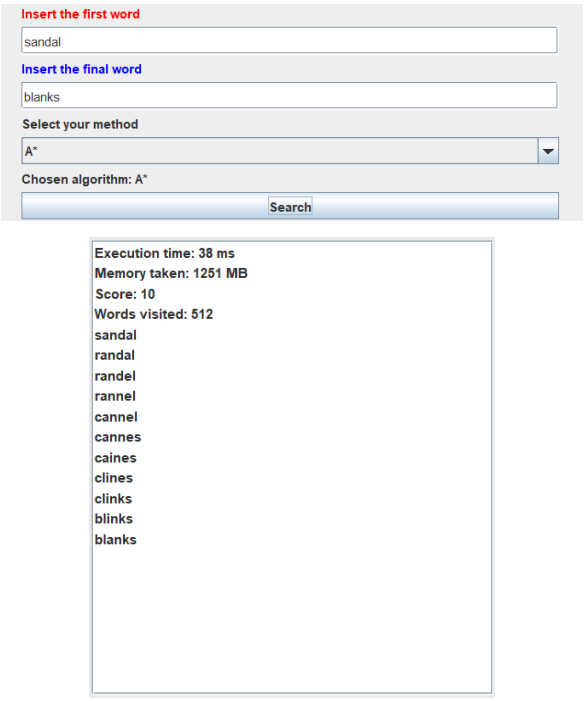
Gambar 3.3.1 - Jacket & Shallow

2. Faith => Clone



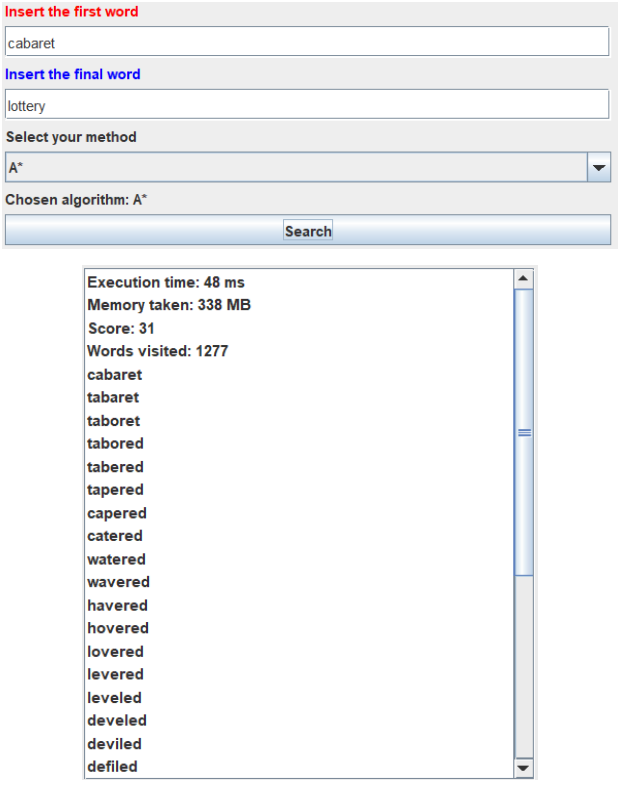
Gambar 3.3.3 - Faith => Clone

3. Sandal => Blanks



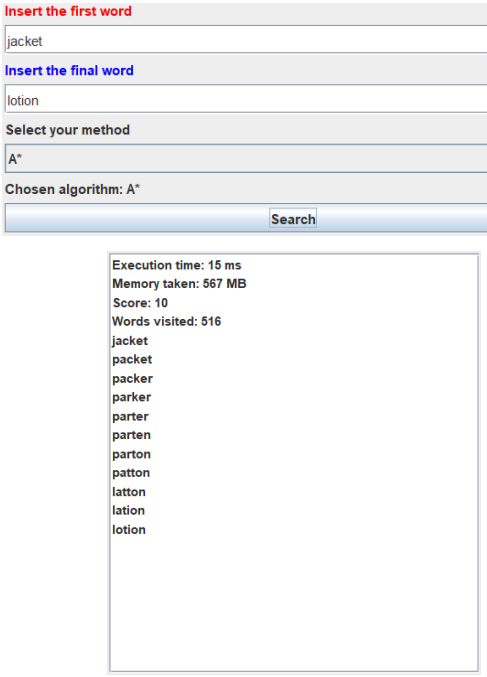
Gambar 3.3.3 - Sandal => Blanks

4. Cabaret => Lottery



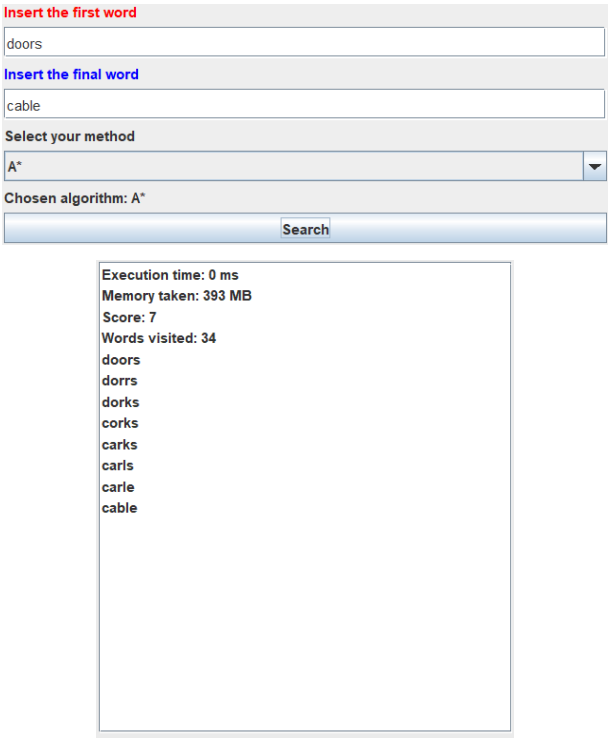
Gambar 3.3.4 - Jarak terdiki

5. Jacket => Lotion



Gambar 3.3.5 - Jacket => Lotion

6. Doors => Cable



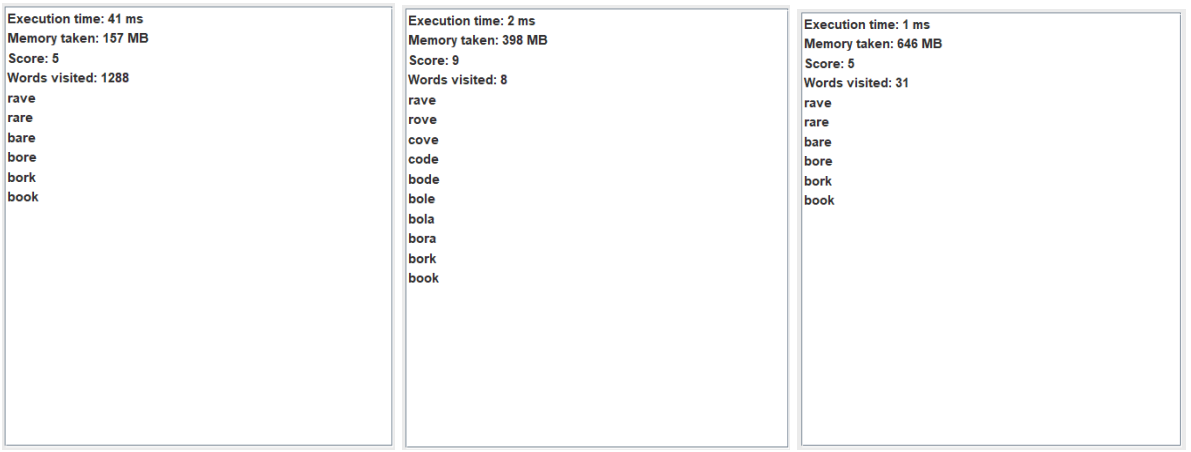
Gambar 3.3.6 = doors ->cable

BAB IV

Analisis Perbandingan Algoritma

4.1 Optimalitas

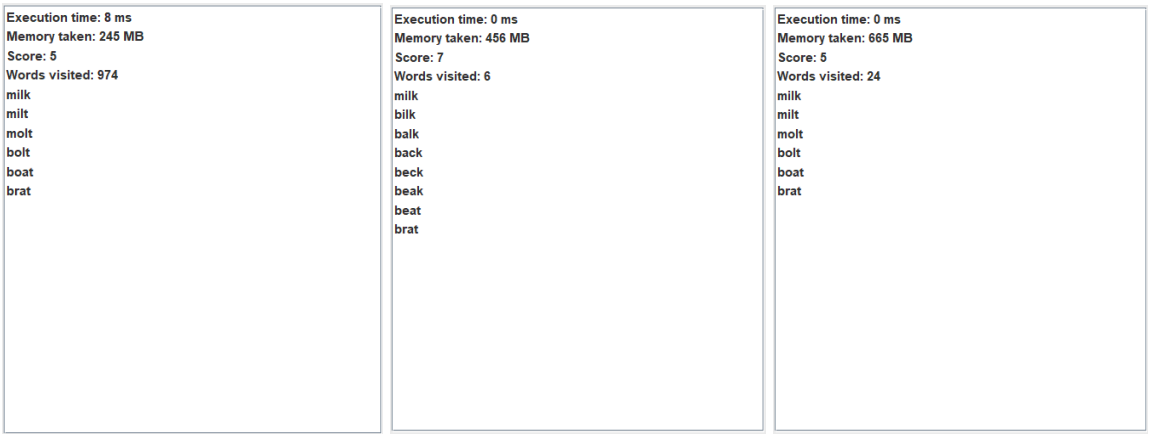
Berdasarkan jumlah kata-kata yang didapat dalam satu solusi, algoritma yang paling optimal adalah algoritma UCS dan A*. Kedua algoritma ini selalu memiliki jumlah kata-kata pada solusi yang sama. Sedangkan pada GBFS, jumlah kata-kata tidak pernah lebih sedikit daripada UCS dan A*. Hal ini disebabkan karena algoritma UCS dan A* selalu akan mencari solusi pertama yang dia temukan yang juga merupakan solusi yang mempunyai f(n) yang paling kecil. Berdasarkan jumlah kata-kata yang ditelusuri, GBFS adalah algoritma yang paling optimal karena GBFS tidak melakukan backtracking sehingga jika pencarian berhasil, jumlah kata yang dicari sama dengan jumlah kata pada solusi sedangkan algoritma UCS dan A* yang memerlukan backtracking harus mencari kata-kata yang lebih banyak. Jika dibandingkan dengan UCS, A* mencari jumlah kata-kata lebih sedikit (Tidak sesedikit GBFS). Dari kedua faktor ini, dari segi optimalitas, algoritma A* adalah yang paling baik. Beginilah tangkapan hasil layar yang menunjukkan optimalitas ketiga algoritma.



Gambar 4.1.1 - (Dari kiri ke kanan => UCS,GBFS,A*) - Perbandingan optimalitas algoritma (Dilihat dari score dan words visited)

4.2 Waktu Eksekusi

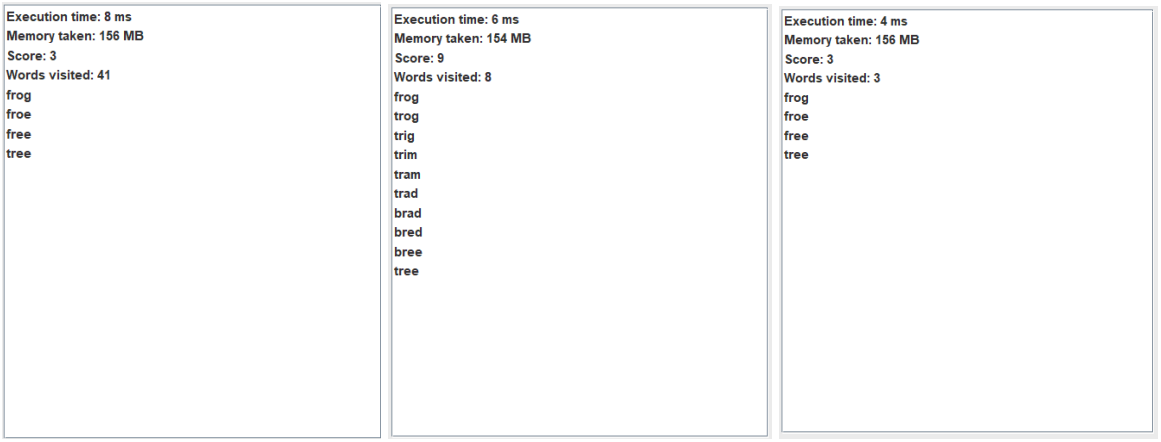
Berdasarkan hasil-hasil yang didapat, algoritma dengan waktu eksekusi yang paling kecil adalah A*. GBFS mempunyai waktu eksekusi yang kecil juga tetapi pada beberapa kasus A* memiliki waktu eksekusi yang sedikit lebih kecil daripada GBFS. Algoritma UCS memiliki solusi yang paling lama dalam pencarian solusi Word Ladder. Hal ini disebabkan karena f(n) yang dimiliki oleh UCS banyak yang sama dibandingkan dengan f(n) yang dimiliki oleh GBFS dan A*. UCS yang memprioritaskan kata ke-n pada pencarian artinya UCS perlu menelusuri semua solusi yang ada n terlebih dahulu dan dilanjutkan dengan n+1. Ini pastinya membuat pencarian lama karena UCS tidak mempunyai informasi tambahan yang berupa heuristic cost. Algoritma seperti GBFS dan A* memiliki heuristic cost yang berasal dari informasi dari kata tujuan yang bisa mempercepat pencarian pada GBFS dan A*. Bukti dari waktu eksekusi ini bisa dilihat pada gambar di bawah ini.



Gambar 4.2.1 - (Dari kiri ke kanan => UCS,GBFS,A*) - Perbandingan waktu eksekusi algoritma (Dilihat dari Execution time)

4.3 Memori

Penggunaan memori pada penyelesaian masalah ini sepertinya tidak terlalu bergantung pada algoritma. Kemungkinan jumlah kata yang dikunjungi bisa berpengaruh tetapi dari hasil beberapa tes, hasil memori yang diambil fluktuatif sehingga kesimpulan dari memori terkecil tidak bisa semudah itu saja pada ketiga algoritma ini. Dari hasil tes, jumlah pengguna melakukan pencarian biasa akan menambah jumlah memori yang digunakan. Pada pertama kali penggunaan, jumlah memorinya sekitar 156 MB. Beginilah hasilnya.



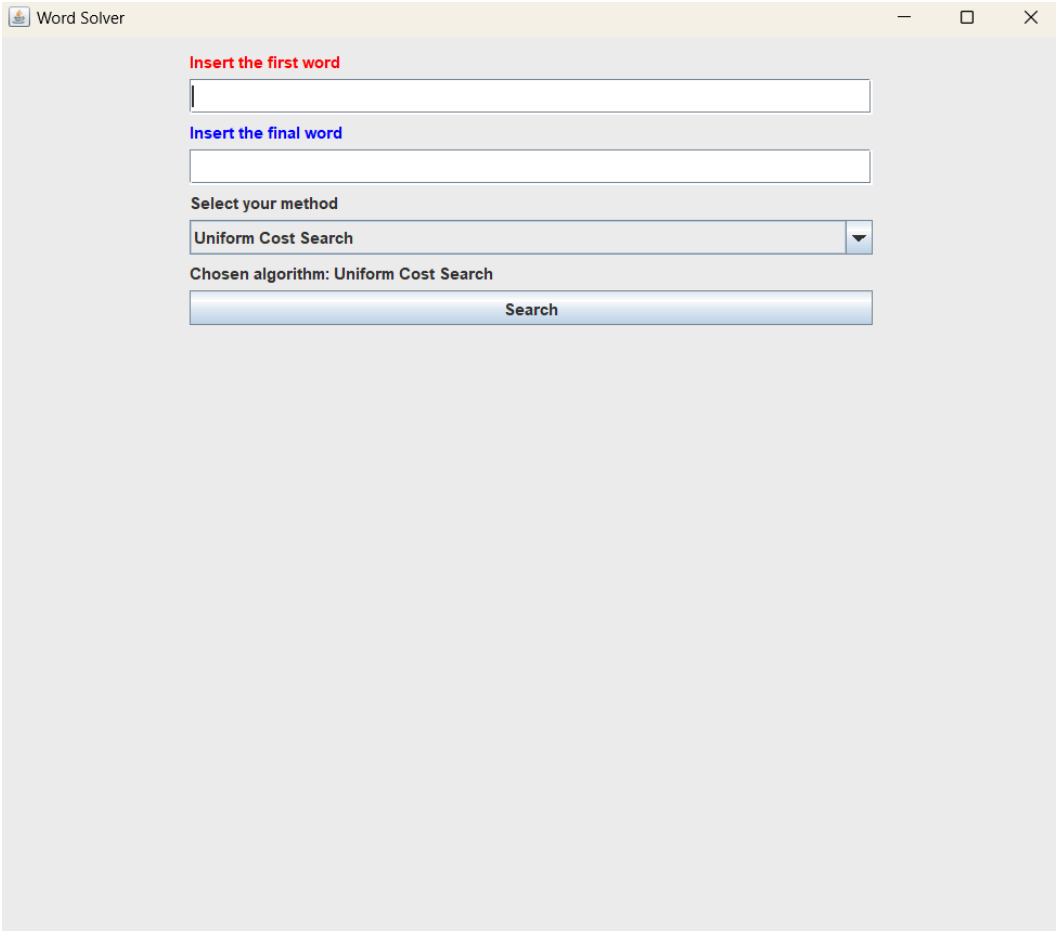
Gambar 4.3.1 - (Dari kiri ke kanan => UCS,GBFS,A*) - Perbandingan waktu eksekusi algoritma (Dilihat dari Memory taken)

BAB V

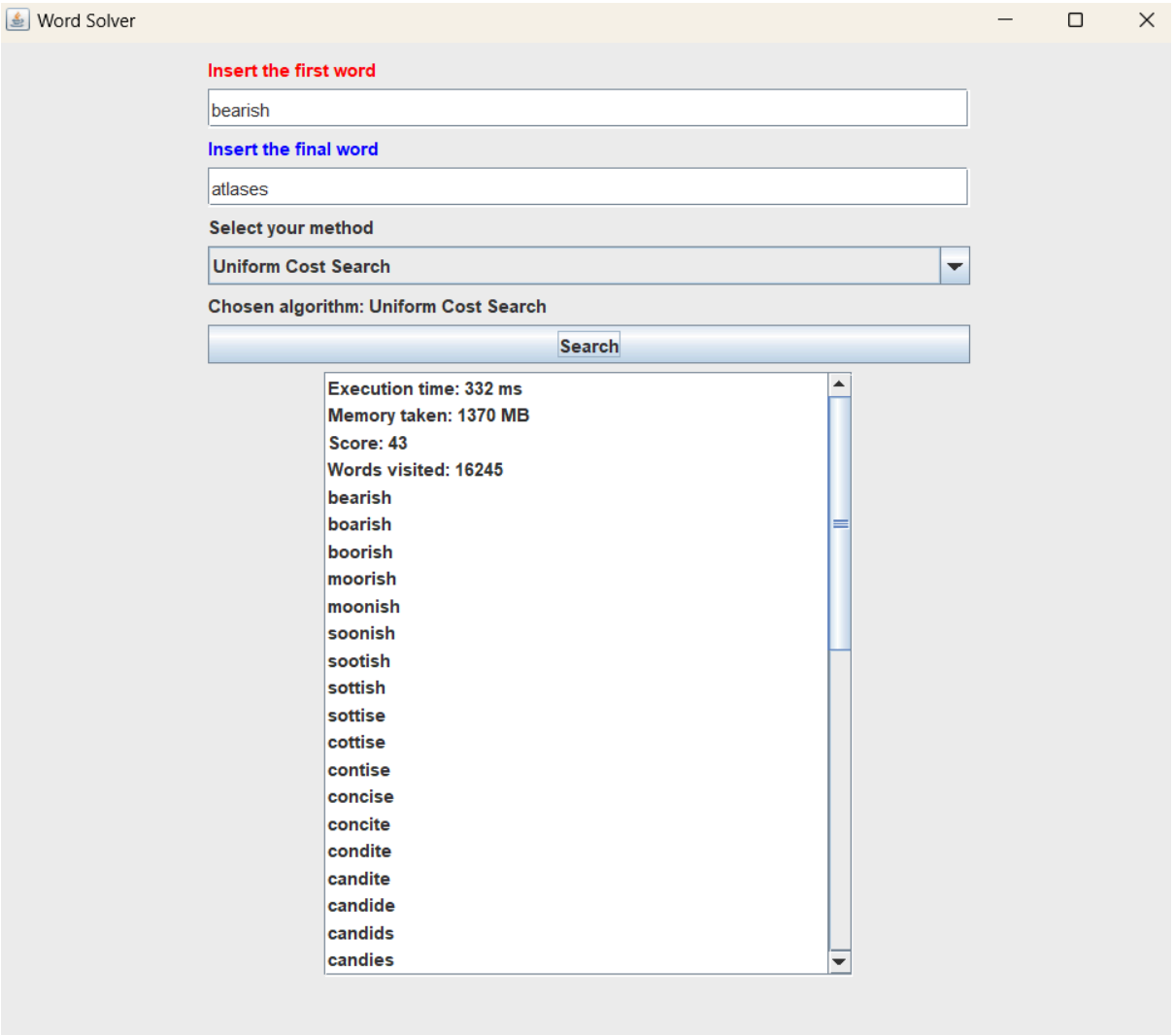
Bonus

5.1 Penggunaan GUI Javax Swing

Pada tugas kecil ini, Javax Swing digunakan sebagai tools pembuatan Graphical User Interface (GUI) pada program yang dibuat. Komponen dari GUI sangatlah sederhana. Berikut adalah gambarnya.



Gambar 5.1.1 - Tampak Depan Aplikasi



Gambar 5.1.2 - Tampak Depan dengan Jawaban

GUI ini diimplementasikan di Class Interface dalam program. Penggunaan konsep kelas Boundary digunakan pada tugas kecil ini dengan Interface sebagai Boundary dan Class Engine sebagai Controller. Hal ini dilakukan untuk mempermudah pembacaan program. Program ini menerima 3 input yaitu kata awal, kata akhir, dan opsi algoritma. Pastikan kata awal dan kata akhir berbahasa Inggris dan memiliki jumlah kata yang sama. Setelah semuanya telah dimasukkan, Class Engine akan mengambil informasi dari Class Interface untuk diproses pada Class Search. Setelah hasil didapatkan, Search akan memberikan hasil ke Class Engine yang akan meneruskannya ke Class Interface agar bisa ditunjukkan dalam GUI.

DAFTAR PUSTAKA

Route-Planning-Bagian1-2021.pdf (itb.ac.id)
Route-Planning-Bagian2-2021.pdf (itb.ac.id)

LAMPIRAN

Link Repository:

scifo04/Tucil3_13522110_IF2211 (github.com)

Check List:

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	✓	