LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA

Disusun Oleh :
Maria Flora Renata Siringoringo (13522010)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

Bab 1: Algoritma

1.1. Struktur Data yang Digunakan

Untuk program ini, digunakan dua struktur data utama, yaitu TreeNode, dan *priority queue*. TreeNode adalah sebuah simpul graf, yang memiliki atribut *root* yaitu kata yang direpresentasikan simpul tersebut, *children* yaitu larik yang berisi kata-kata yang berbeda satu huruf dari *root*, *path* yaitu larik yang berisi kata-kata yang dilewati untuk mencapai simpul ini dari kata awal, g(n), dan h(n).

G(n) adalah *cost* atau harga dari simpul awal ke simpul ekspan. Dalam konteks tugas ini, g(n) didefinisikan sebagai banyaknya perubahan yang perlu dilakukan untuk mencapai kata di simpul ekspan dari kata awal.

H(n) adalah perkiraan *cost* atau harga dari simpul ekspan ke simpul akhir. Dalam konteks tugas ini, h(n) didefinisikan sebagai banyaknya huruf yang berbeda antara sebuah simpul dengan simpul akhir.

Selanjutnya, TreeNode tersebut disimpan di dalam sebuah *priority queue*. *Priority queue* inilah yang akan menentukan urutan simpul hidup dipilih menjadi simpul ekspan. Prioritas dalam *priority queue* ini berdasarkan pada f(n), yaitu sebuah fungsi evaluasi yang berbeda untuk setiap algoritma. Oleh karena itu, terdapat tiga jenis *push* yang bisa dilakukan untuk *priority queue* ini, yaitu *push* berdasarkan g(n), *push* berdasarkan h(n), atau *push* berdasarkan total dari kedua nilai tersebut.

1.2.Uniform Cost Search

Uniform Cost Search (UCS) adalah sebuah algoritma penentuan rute yang bertujuan untuk mencari rute dengan *cost*/harga terendah. Algoritma UCS yang diimplementasikan pada tugas ini menggunakan sebuah *priority queue* untuk menyimpan simpul-simpul (kata) hidup berdasarkan g(n) dengan urutan membesar, sedangkan simpul ekspan adalah elemen pertama dari *priority queue*.

Dalam algoritma UCS untuk tugas ini, program memiliki sebuah *loop* utama yang akan mengecek apakah indeks pertama dari *queue* merupakan kata akhir. Jika iya, maka *loop* selesai, dan jika tidak, program akan mencari semua kata dalam kamusnya yang hanya berbeda satu huruf dari indeks pertama *queue*. Jika ada, maka kata-kata tersebut yang belum pernah menjadi simpul ekspan dimasukkan kedalam *queue*. *Loop* akan terus berjalan hingga tidak ada simpul hidup (*queue* kosong) atau kata akhir ditemukan.

Dalam tugas ini, karena g(n) dari setiap simpul hanya merupakan banyaknya perubahan dari simpul awal ke simpul tersebut, semua jalan antara simpul memiliki *cost* yang sama, yaitu 1. Hal ini menyebabkan UCS sama persis dengan BFS untuk kasus *word ladder*.

1.3.Greedy Best First Search

Greedy Best First Search (GBFS) adalah sebuah algoritma penentuan rute yang menggunakan konsep algoritma *greedy*. Algoritma ini bertujuan untuk mencari rute dengan *cost*/harga terendah dengan memilih simpul dengan h(n) terendah pada tiap langkah.

Implementasi algoritma GBFS dalam tugas ini cukup mirip dengan algoritma UCS, perbedaannya terletak pada *priority queue* yang diurutkan berdasarkan h(n) membesar.

Karena hanya melihat jarak simpul ke simpul akhir, hasil dari algoritma ini sering terjebak dalam minimum lokal, sehingga hasil dari algoritma ini tidak selalu optimal.

1.4. A*

A* adalah sebuah algoritma penentuan rute yang bertujuan untuk mencari rute dengan *cost*/harga terkecil, dengan melihat jarak dari sebuah simpul ke simpul awal dan akhir, tidak hanya salah satu saja. Simpul yang dipilih berdasarkan f(n) = g(n) + h(n).

Implementasi algoritma A* dalam tugas ini juga mirip dengan algoritma UCS, perbedaannya hanya terdapat pada *priority queue* yang mengurutkan berdasarkan g(n) + h(n) secara membesar.

H(n) yang digunakan pada tugas ini, baik untuk A* dan GBFS, *admissible*. Artinya, h(n) yang digunakan akan selalu bernilai lebih rendah atau sama dengan jarak asli dari simpul tersebut ke simpul akhir. H(n) yang digunakan adalah jumlah huruf yang berbeda dari sebuah simpul ke simpul akhir, sedangkan banyaknya perubahan yang dibutuhkan sebenarnya adalah paling sedikit sama dengan h(n), dan bisa lebih besar dari h(n) jika tidak ada kata yang valid untuk perubahan secara langsung. Sebagai contoh, simpul “east” dengan tujuan “earn” memiliki h(n) = 2, sedangkan “east” tidak dapat diubah menjadi “easn” atau “eart”, sehingga jarak aslinya lebih besar dari 2.

Secara teoritis, A* lebih baik dari UCS, karena A* adalah *informed search*, sedangkan UCS adalah *uninformed search*. A* mengetahui perkiraan jarak dari setiap simpul ke simpul sebelumnya dan tujuan, sedangkan UCS tidak. Oleh karena itu, dalam tugas ini, A* akan lebih cepat dari UCS walaupun keduanya memiliki hasil yang sama-sama optimal.

Bab 2: Source code

2.1. Main.java

```java
import javax.swing.SwingUtilities;

You, 15 hours ago | 1 author (You)
class Main {
    Run | Debug
    public static void main(String[] args) {
        You, 15 hours ago | 1 author (You)
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new WordLadderSolverGUI();        You, 15 hours ago
            }
        });
    }
}
```

File ini hanya memiliki kelas Main, yang hanya bertugas untuk menyalakan GUI.

2.2. Prioqueue.java

```java
You, 1 hour ago | 1 author (You)
import java.util.ArrayList;

You, 1 hour ago | 1 author (You)
public class Prioqueue {
    public ArrayList<TreeNode> queue;

    public Prioqueue() {
        this.queue = new ArrayList<TreeNode>();
    }
```

```java
    public void pushGn(TreeNode n) {
        if (this.queue.isEmpty()) this.queue.add(index:0, n);
        else {
            for (int i=0; i<queue.size(); i++) {
                if (queue.get(i).getGn() > n.getGn()) {
                    queue.add(i, n);
                    break;
                }
            }
            if (queue.get(queue.size()-1).getGn() <= n.getGn()) queue.addLast(n);
        }
    }

    public void pushHn(TreeNode n) {
        if (this.queue.isEmpty()) this.queue.add(index:0, n);
        else {
            for (int i=0; i<queue.size(); i++) {
                if (queue.get(i).getHn() > n.getHn()) {
                    queue.add(i, n);
                    break;
                }
            }
            if (queue.get(queue.size()-1).getHn() <= n.getHn()) queue.addLast(n);
        }
    }
```

```java
public void pushGnHn(TreeNode n) {
    if (this.queue.isEmpty()) this.queue.add(index:0, n);
    else {
        for (int i=0; i<queue.size(); i++) {
            if ((queue.get(i).getGn() + queue.get(i).getHn()) > (n.getGn() + n.getHn())) {
                queue.add(i, n);
                break;
            }
        }
        if ((queue.get(queue.size()-1).getGn() + queue.get(queue.size()-1).getHn()) <= (n.getGn() + n.getHn())) queue.addLast(n);
    }
}

public TreeNode pop() {
    TreeNode temp = this.queue.get(index:0);
    this.queue.remove(index:0);
    return temp;
}

public boolean isEmpty() {
    return this.queue.isEmpty();
}
```

File ini memiliki kelas Prioqueue. Kelas ini memiliki atribut ArrayList<TreeNode> dengan metode push (dengan tiga cara seperti pada penjelasan pada 1.1.), pop, dan isEmpty untuk mengecek kekosongan sebuah Prioqueue.

2.3. DictReader.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Map;

You, yesterday | 1 author (You)
public class DictReader {
    private File dict;
    private HashSet<String> words;
    private Map<String, ArrayList<String>> differentByOne;

    public DictReader(String path) {
        this.dict = new File(path);
        this.differentByOne = new HashMap<>();
    }

    public Boolean dictValid() {
        return (this.dict.exists());
    }
```

```java
public void makeList() {
    this.words = new HashSet<>();
    try {
        Scanner read = new Scanner(this.dict);
        while (read.hasNextLine()) {
            this.words.add(read.nextLine().toLowerCase());
        }
        read.close();
    }
    catch (FileNotFoundException e) {
        System.err.println(e.getMessage());
    }
}
    You, yesterday • feat: finish basics
private Boolean hasWord(String word) {
    return (this.words.contains(word));
}
```

```java
public TreeNode findAdjacent(TreeNode n) {
    if (!differentByOne.containsKey(n.getRoot())) { // Jika kata yang dicari belum ada di map, cari manual dan masukkan
        Character temp;
        StringBuilder curWord = new StringBuilder(n.getRoot());
        ArrayList<String> children = new ArrayList<>();
        for (int j = 0; j < n.getRoot().length(); j++) {
            for (char i = 'a'; i <= 'z'; i++) {
                if (curWord.charAt(j) != i){
                    temp = curWord.charAt(j);
                    curWord.setCharAt(j, i);
                    if (this.hasWord(curWord.toString())) {
                        n.AddChild(curWord.toString());
                        children.add(curWord.toString());
                    }
                    curWord.setCharAt(j, temp);
                }
            }
        }
        differentByOne.put(n.getRoot(), children);
    }
    else { // Kata sudah ada di map, tinggal get
        for (String children : differentByOne.get(n.getRoot())) {
            n.AddChild(children);
        }
    }
    return n;
}
}
```

File ini memiliki kelas DictReader yang bertugas untuk membaca kamus. Kelas ini memiliki atribut berupa file kamus, HashSet yang menyimpan semua kata dalam kamus, serta Map<String, ArrayList<String>> yang menyimpan sebuah kata sebagai kunci dan kata-kata yang hanya berbeda satu huruf dengannya sebagai nilai dari kunci tersebut. Kelas ini memiliki metode dictValid untuk mengecek apakah file kamus dapat dibuka, makeList untuk memindahkan semua kata dari kamus ke HashSet, hasWord untuk mengecek apakah sebuah kata terdapat pada kamus, dan findAdjacent yang digunakan untuk mencari kata-kata yang hanya beda satu huruf dengan sebuah TreeNode. findAdjacent akan membaca dari Map jika kata tersebut ada di Map, jika tidak, findAdjacent akan mengubah semua huruf dari TreeNode dan mengecek apakah kata

hasil perubahan berada di kamus. Jika kata tersebut ditemukan, akan ditambahkan ke children dari TreeNode dan ditambahkan ke Map.

2.4. TreeNode.java

```java
public class TreeNode {
    private ArrayList<String> path;
    private String root;
    private ArrayList<String> children;
    private int gn;
    private int hn;

    public TreeNode(String root) {
        this.root = root;       You, yesterday • feat: finish basics
        this.path = new ArrayList<>();
        this.path.add(root);
        this.children = new ArrayList<>();
        this.gn = 0;
        this.hn = 0;
    }

    public TreeNode(String root, ArrayList<String> path, int gn, int hn) {
        this.root = root;
        this.path = new ArrayList<>(path);
        this.path.add(root);
        this.children = new ArrayList<>();
        this.gn = gn;
        this.hn = hn;
    }

    public void AddChild(String child) {
        this.children.add(child);
    }
```

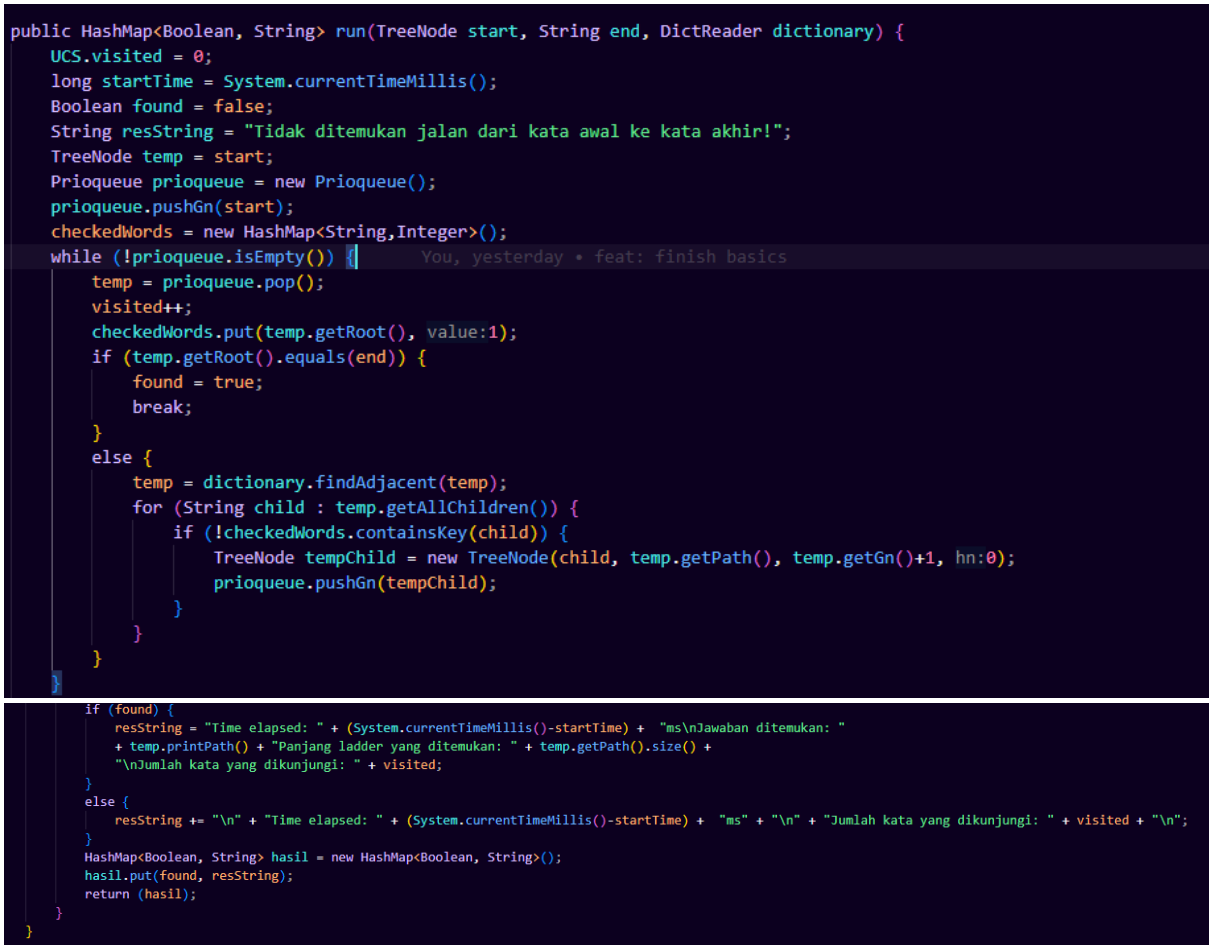
```java
    public String getChild(int n) {
        return this.children.get(n);
    }

    public ArrayList<String> getAllChildren() {
        return this.children;
    }

    public String getRoot() {
        return this.root;
    }

    public ArrayList<String> getPath() {
        return this.path;
    }

    public String printPath() {
        String res = "";
        for (int i=0; i<this.path.size()-1; i++) {
            res += this.path.get(i);
            res += " -> ";
        }
        res += this.path.getLast();
        res += "\n";
        return res;
    }
```

```java
    public int getGn() {
        return this.gn;
    }

    public int getHn() {
        return this.hn;
    }
}
```

File ini memiliki kelas TreeNode yang sesuai dengan penjelasan pada 1.1.. Selain *getter*, kelas ini hanya memiliki metode printPath yang akan mengubah *path* sebuah TreeNode menjadi sebuah String.

2.5. UCS.java

```java
import java.util.Map;
import java.util.HashMap;

You, 54 minutes ago | 1 author (You)
public class UCS {
    private static int visited;
    private Map<String, Integer> checkedWords;
```



```
public HashMap<Boolean, String> run(TreeNode start, String end, DictReader dictionary) {
    UCS.visited = 0;
    long startTime = System.currentTimeMillis();
    Boolean found = false;
    String resString = "Tidak ditemukan jalan dari kata awal ke kata akhir!";
    TreeNode temp = start;
    Prioqueue prioqueue = new Prioqueue();
    prioqueue.pushGn(start);
    checkedWords = new HashMap<String,Integer>();
    while (!prioqueue.isEmpty()) {      You, yesterday • feat: finish basics
        temp = prioqueue.pop();
        visited++;
        checkedWords.put(temp.getRoot(), value:1);
        if (temp.getRoot().equals(end)) {
            found = true;
            break;
        }
        else {
            temp = dictionary.findAdjacent(temp);
            for (String child : temp.getAllChildren()) {
                if (!checkedWords.containsKey(child)) {
                    TreeNode tempChild = new TreeNode(child, temp.getPath(), temp.getGn()+1, hn:0);
                    prioqueue.pushGn(tempChild);
                }
            }
        }
    }
```

```
        if (found) {
            resString = "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms\nJawaban ditemukan: "
            + temp.printPath() + "Panjang ladder yang ditemukan: " + temp.getPath().size() +
            "\nJumlah kata yang dikunjungi: " + visited;
        }
        else {
            resString += "\n" + "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms" + "\n" + "Jumlah kata yang dikunjungi: " + visited + "\n";
        }
        HashMap<Boolean, String> hasil = new HashMap<Boolean, String>();
        hasil.put(found, resString);
        return (hasil);
    }
}
```

File ini memiliki kelas UCS. Kelas ini memiliki atribut *visited* untuk menyimpan jumlah kata yang sudah dikunjungi dan sebuah Map<String, Integer> untuk mencatat semua kata yang sudah dikunjungi. Kelas ini hanya memiliki satu metode *run*, yang menerima simpul TreeNode awal, kata akhir dalam bentuk String, dan sebuah obyek DictReader dan mengembalikan sebuah HashMap<Boolean, String> yang berisi satu pasang Boolean dan String yang menunjukkan hasil pencarian (baik berhasil atau tidak).

Pertama, *run* akan me-*reset* semua atribut dan variabel, lalu run akan memulai sebuah *while loop* yang akan berhenti jika PrioQueue sudah kosong. *Loop* berjalan sesuai dengan penjelasan pada 1.2., *loop* akan dihentikan jika kata akhir sudah ditemukan dan akan terus mencari simpul anak dari simpul ekspan sekarang jika belum. Jika pencarian berhasil, String yang dikembalikan akan berisi jalur dan panjang jalur dari kata awal ke kata akhir yang ditemukan, jika tidak, string akan berisi sebuah pesan kesalahan. Dalam kedua kasus tersebut, string tetap akan memiliki waktu lamanya fungsi berjalan dan jumlah kata yang sudah dikunjungi saat *loop* berjalan.

2.6. GBFS.java

```
import java.util.HashMap;
import java.util.Map;
      You, yesterday • feat: finish basics
You, 1 hour ago | 1 author (You)
public class GBFS {
    private static int visited = 0;
    private Map<String, Integer> checkedWords;

    public HashMap<Boolean,String> run(TreeNode start, String end, DictReader dictionary) {
        GBFS.visited = 0;
        long startTime = System.currentTimeMillis();
        Boolean found = false;
        String resString = "Tidak ditemukan jalan dari kata awal ke kata akhir!";
        TreeNode temp = start;
        Prioqueue prioqueue = new Prioqueue();
        prioqueue.pushHn(start);
        checkedWords = new HashMap<String,Integer>();
        while (!prioqueue.isEmpty()) {
            temp = prioqueue.pop();
            prioqueue.clear();
            visited++;
            checkedWords.put(temp.getRoot(), value:1);
            if (temp.getRoot().equals(end)) {
                found = true;
                break;
            }
            else {
                temp = dictionary.findAdjacent(temp);
                for (String child : temp.getAllChildren()) {
                    if (!checkedWords.containsKey(child)) {
                        TreeNode tempChild = new TreeNode(child, temp.getPath(), gn:0, this.countHn(child, end));
                        prioqueue.pushHn(tempChild);
                    }
                }
            }
        }
        if (found) {
            resString = "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms" + "\n" + "Jawaban ditemukan: "
            + temp.printPath() + "Panjang ladder yang ditemukan: " + temp.getPath().size() +
            "\n" + "Jumlah kata yang dikunjungi: " + visited + "\n";
        }
        else {
            resString += "\n" + "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms" + "\n" + "Jumlah kata yang dikunjungi: " + visited + "\n";
        }
        HashMap<Boolean, String> hasil = new HashMap<Boolean, String>();
        hasil.put(found, resString);
        return (hasil);
    }

    private int countHn(String word, String goal) {
        int count = 0;
        for (int i=0; i<word.length(); i++) {
            if(word.charAt(i)!=goal.charAt(i)) count++;
        }
        return count;
    }
}
```

File ini memiliki kelas GBFS yang mirip dengan kelas UCS. Kelas ini memiliki atribut *visited* untuk menyimpan jumlah kata yang sudah dikunjungi dan sebuah Map<String, Integer> untuk mencatat semua kata yang sudah dikunjungi. Kelas ini hanya memiliki satu metode *run*, yang menerima simpul TreeNode awal, kata akhir dalam bentuk String, dan sebuah obyek DictReader dan mengembalikan sebuah HashMap<Boolean, String> yang berisi satu pasang Boolean dan String yang menunjukkan hasil pencarian (baik berhasil atau tidak).

Pertama, *run* akan me-*reset* semua atribut dan variabel, lalu run akan memulai sebuah *while loop* yang akan berhenti jika PrioQueue sudah kosong. *Loop* berjalan sesuai dengan penjelasan pada 1.3., *loop* akan dihentikan jika kata akhir sudah ditemukan dan

akan terus mencari simpul anak dari simpul ekspan sekarang jika belum. Jika pencarian berhasil, String yang dikembalikan akan berisi jalur dan panjang jalur dari kata awal ke kata akhir yang ditemukan, jika tidak, string akan berisi sebuah pesan kesalahan. Dalam kedua kasus tersebut, string tetap akan memiliki waktu lamanya fungsi berjalan dan jumlah kata yang sudah dikunjungi saat *loop* berjalan.

2.7. Astar.java

```java
import java.util.HashMap;
import java.util.Map;

You, 1 hour ago | 1 author (You)
public class Astar {
    private static int visited = 0;
    private Map<String, Integer> checkedWords;
```

```java
  public HashMap<Boolean,String> run(TreeNode start, String end, DictReader dictionary) {
      Astar.visited = 0;
      long startTime = System.currentTimeMillis();
      Boolean found = false;
      String resString = "Tidak ditemukan jalan dari kata awal ke kata akhir!";
      TreeNode temp = start;
      Prioqueue prioqueue = new Prioqueue();
      prioqueue.pushGnHn(start);
      checkedWords = new HashMap<String,Integer>();

      while (!prioqueue.isEmpty()) {
          temp = prioqueue.pop();
          visited++;
          checkedWords.put(temp.getRoot(), value:1);

          if (temp.getRoot().equals(end)) {
              found = true;
              break;
                You, 1 second ago • Uncommitted changes
          else {
              temp = dictionary.findAdjacent(temp);
              for (String child : temp.getAllChildren()) {
                  if (!checkedWords.containsKey(child)) {
                      TreeNode tempChild = new TreeNode(child, temp.getPath(), temp.getGn()+1,  this.countHn(child, end));
                      prioqueue.pushGnHn(tempChild);
                  }
              }
          }
      }
```

```java
      if (found) {
          resString = "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms\nJawaban ditemukan: "
          + temp.printPath() + "Panjang ladder yang ditemukan: " + temp.getPath().size() +
          "\nJumlah kata yang dikunjungi: " + visited;
      }
      else {
          resString += "\n" + "Time elapsed: " + (System.currentTimeMillis()-startTime) +  "ms" + "\n" + "Jumlah kata yang dikunjungi: " + visited + "\n";
      }
      HashMap<Boolean, String> hasil = new HashMap<Boolean, String>();
      hasil.put(found, resString);
      return (hasil);
  }

  private int countHn(String word, String goal) {
      int count = 0;
      for (int i=0; i<word.length(); i++) {
          if(word.charAt(i)!=goal.charAt(i)) count++;
      }
      return count;
  }
```

File ini memiliki kelas Astar yang mirip dengan kelas UCS. Kelas ini memiliki atribut *visited* untuk menyimpan jumlah kata yang sudah dikunjungi dan sebuah Map<String, Integer> untuk mencatat semua kata yang sudah dikunjungi. Kelas ini hanya memiliki satu metode *run*, yang menerima simpul TreeNode awal, kata akhir dalam bentuk String, dan sebuah obyek DictReader dan mengembalikan sebuah HashMap<Boolean, String> yang berisi satu pasang Boolean dan String yang menunjukkan hasil pencarian (baik berhasil atau tidak).

Pertama, *run* akan me-*reset* semua atribut dan variabel, lalu run akan memulai sebuah *while loop* yang akan berhenti jika PrioQueue sudah kosong. *Loop* berjalan sesuai dengan penjelasan pada 1.4., *loop* akan dihentikan jika kata akhir sudah ditemukan dan akan terus mencari simpul anak dari simpul ekspan sekarang jika belum. Jika pencarian berhasil, String yang dikembalikan akan berisi jalur dan panjang jalur dari kata awal ke kata akhir yang ditemukan, jika tidak, string akan berisi sebuah pesan kesalahan. Dalam kedua kasus tersebut, string tetap akan memiliki waktu lamanya fungsi berjalan dan jumlah kata yang sudah dikunjungi saat *loop* berjalan.

Bab 3: Pengujian

Untuk bab ini, karena adanya sistem *mapping*, semua pencarian yang digunakan adalah hasil *run* pertama kalinya, agar semua hasil didapatkan adil dan tanpa *mapping*.

3.1. “East” -> “West”

UCS	Time elapsed: 16ms Jawaban ditemukan: east -> wast -> west Panjang ladder yang ditemukan: 3 Jumlah kata yang dikunjungi: 140
GBFS	Time elapsed: 0ms Jawaban ditemukan: east -> wast -> west Panjang ladder yang ditemukan: 3 Jumlah kata yang dikunjungi: 3
A*	Time elapsed: 0ms Jawaban ditemukan: east -> wast -> west Panjang ladder yang ditemukan: 3 Jumlah kata yang dikunjungi: 3

3.2. “East” -> “Earn”

UCS	Time elapsed: 398ms Jawaban ditemukan: east -> cast -> cart -> carn -> earn Panjang ladder yang ditemukan: 5 Jumlah kata yang dikunjungi: 3559
GBFS	Time elapsed: 0ms Jawaban ditemukan: east -> ease -> eave -> cave -> care -> carn -> earn Panjang ladder yang ditemukan: 7 Jumlah kata yang dikunjungi: 7
A*	Time elapsed: 5ms Jawaban ditemukan: east -> cast -> cart -> carn -> earn Panjang ladder yang ditemukan: 5 Jumlah kata yang dikunjungi: 26

3.3. “Green” -> “Olive”

UCS	Time elapsed: 81327ms Jawaban ditemukan: green -> grees -> glees -> glens -> glans -> alans -> alane -> aline -> alive -> olive Panjang ladder yang ditemukan: 10 Jumlah kata yang dikunjungi: 151599
GBFS	Time elapsed: 4ms Jawaban ditemukan: green -> preen -> treen -> tween -> tweed -> treed -> tried -> cried -> dried -> fried -> flied -> plied -> plier -> flier -> slier -> shier -> skier -> spier -> spied -> shied -> skied -> stied -> sties -> shies -> shivs -> shive -> chive -> chide -> chile -> while -> whine -> chine -> cline -> aline -> alive -> olive Panjang ladder yang ditemukan: 36 Jumlah kata yang dikunjungi: 36
A*	Time elapsed: 38ms Jawaban ditemukan: green -> grees -> glees -> glens -> glans -> alans -> alane -> aline -> alive -> olive Panjang ladder yang ditemukan: 10 Jumlah kata yang dikunjungi: 1409

3.4. “Throne” -> “Castle”

UCS	Time elapsed: 44072ms Jawaban ditemukan: throne -> throve -> shrove -> shrive -> shrine -> serine -> serins -> series -> serges -> sarges -> parges -> parged -> parted -> patted -> pattee -> pattie -> cattie -> cattle -> castle Panjang ladder yang ditemukan: 19 Jumlah kata yang dikunjungi: 205437
GBFS	Time elapsed: 2ms Jawaban ditemukan: throne -> throve -> shrove -> shrive -> shrine -> serine -> ferine -> feline -> reline -> refine -> refile -> defile -> decile -> deckle -> heckle -> hackle -> tackle -> tickle -> tinkle -> tingle -> tangle -> mangle -> mantle -> cantle -> castle Panjang ladder yang ditemukan: 25 Jumlah kata yang dikunjungi: 264
A*	Time elapsed: 568ms Jawaban ditemukan: throne -> throve -> shrove -> shrive -> shrine -> serine -> serins -> series -> serges -> sarges -> parges -> parged -> parted -> patted -> pattee -> pattie -> cattie -> cattle -> castle Panjang ladder yang ditemukan: 19 Jumlah kata yang dikunjungi: 17015

3.5. “Comedy” -> “Jester”

UCS	Time elapsed: 4974ms Jawaban ditemukan: comedy -> comely -> homely -> homily -> hominy -> homing -> coming -> coning -> conins -> conies -> cosies -> cosier -> rosier -> roster -> rester -> jester Panjang ladder yang ditemukan: 16 Jumlah kata yang dikunjungi: 36805
GBFS	Time elapsed: 1ms Jawaban ditemukan: comedy -> comely -> homely -> homily -> hominy -> homing -> hosing -> rosing -> rosins -> resins -> resids -> resods -> resows -> resews -> resees -> reseed -> rested -> jested -> jester Panjang ladder yang ditemukan: 19 Jumlah kata yang dikunjungi: 53
A*	Time elapsed: 48ms Jawaban ditemukan: comedy -> comely -> homely -> homily -> hominy -> homing -> coming -> coning -> conins -> conies -> cosies -> cosier -> coster -> foster -> fester -> jester Panjang ladder yang ditemukan: 16 Jumlah kata yang dikunjungi: 2125

3.6. “Robbery” -> “Mailing”

UCS	Time elapsed: 2370ms Jawaban ditemukan: robbery -> robbers -> cobbers -> combers -> cambers -> campers -> carpers -> carters -> barters -> batters -> battens -> lattens -> lattins -> mattins -> martins -> marlins -> marling -> mailing Panjang ladder yang ditemukan: 18 Jumlah kata yang dikunjungi: 92869
GBFS	Time elapsed: 16ms Jawaban ditemukan: robbery -> robbers -> cobbers -> combers -> cambers -> campers -> dampers -> dammers -> mammers -> maimers -> mailers -> bailers -> bailees -> bailies -> dailies -> doilies -> dollies -> collies -> collins -> codlins -> codling -> coiling -> moiling -> mailing Panjang ladder yang ditemukan: 24 Jumlah kata yang dikunjungi: 543
A*	Time elapsed: 113ms Jawaban ditemukan: robbery -> robbers -> cobbers -> combers -> cambers -> campers -> carpers -> carters -> barters -> batters -> battens -> lattens -> lattins -> mattins -> matting -> malting -> malling -> mailing Panjang ladder yang ditemukan: 18 Jumlah kata yang dikunjungi: 6532

Bab 4: Analisa Perbandingan

Dari data pengujian, dapat dilakukan analisa dari tiga sisi, yaitu optimalitas, waktu eksekusi, dan memori yang dibutuhkan.

4.1. Optimalitas

Dari hasil pengujian, didapatkan bahwa UCS dan A* optimal, sedangkan GBFS tidak selalu optimal. Dari keenam pengujian yang dilakukan, UCS dan A* selalu mendapatkan hasil dengan panjang *ladder* yang sama, walaupun kata yang digunakan di dalam *ladder* tersebut berbeda. Sedangkan GBFS hanya mendapatkan hasil dengan panjang yang sama sebanyak satu kali pada pengujian pertama, untuk pengujian lainnya hasil dari GBFS selalu lebih panjang.

Berikut adalah contoh pengujian dari “Greedy” ke “Normal” dengan menggunakan GBFS (atas) dan UCS (bawah). Dapat dilihat bahwa hasil yang didapatkan oleh GBFS memiliki panjang 22, sedangkan UCS memiliki panjang 20.

```
Time elapsed: 22ms
Jawaban ditemukan: greedy -> greeds -> breeds -> bleeds -> bleeps -> sleeps -> steeps -> streps -> strips -> scrips -> scries -> series -> serges -> surges -> gurges -> gorges -> forges -> forget -> forgat -> format -> formal -> normal
Panjang ladder yang ditemukan: 22
Jumlah kata yang dikunjungi: 771
```

```
Time elapsed: 4679ms
Jawaban ditemukan: greedy -> greeds -> breeds -> bleeds -> blends -> blinds -> blinks -> blanks -> flanks -> flanes -> flames -> flamed -> foamed -> formed -> forged -> forget -> forgat -> format -> formal -> normal
Panjang ladder yang ditemukan: 20
Jumlah kata yang dikunjungi: 56779
```

4.2. Waktu Eksekusi

Dari hasil pengujian, didapatkan rata-rata lama fungsi berjalan (untuk 6 *test case* di bab 3) sebagai berikut

UCS	22.192,83ms
GBFS	3,83ms
A*	128,67ms

Maka, didapatkan bahwa GBFS adalah algoritma yang paling cepat, lalu A*, dan UCS adalah algoritma paling lambat.

Hal ini terjadi karena GBFS selalu memilih simpul yang terlihat paling dekat dengan simpul tujuan tanpa melihat simpul lain, sehingga walaupun tidak optimal, GBFS berjalan dengan sangat cepat.

UCS akan selalu memilih simpul yang paling dekat dengan simpul awal, sehingga pengecekan akan sangat lambat. A* terletak di tengah-tengah GBFS dan UCS, A* lebih cepat daripada UCS karena juga memperhitungkan kedekatan simpul dengan tujuan

akhir, sedangkan lebih lambat dari GBFS karena masih memperhitungkan kedekatan simpul dengan simpul awal.

4.3. Memori

Dari hasil pengujian, didapatkan rata-rata banyaknya kata yang dikunjungi (untuk 6 *test case* di bab 3) sebagai berikut

UCS	81734,83
GBFS	151
A*	4518,33

Maka, didapatkan bahwa GBFS adalah algoritma yang menggunakan memori paling sedikit, lalu A*, dan UCS menggunakan paling banyak memori.

Penjelasan untuk hal ini cukup mirip dengan alasan GBFS menjadi algoritma paling cepat. Karena GBFS langsung memilih simpul yang terlihat paling dekat dengan tujuan, GBFS tidak mengecek simpul lain kecuali menemukan jalan buntu, sehingga kata yang dikunjungi sangat sedikit. UCS mengecek semua kata pada satu level g(n)/kedekatan sebelum mencari di g(n) berikutnya, sehingga UCS pasti mengunjungi paling banyak kata dan menggunakan paling banyak memori.

Bab 5: Bonus

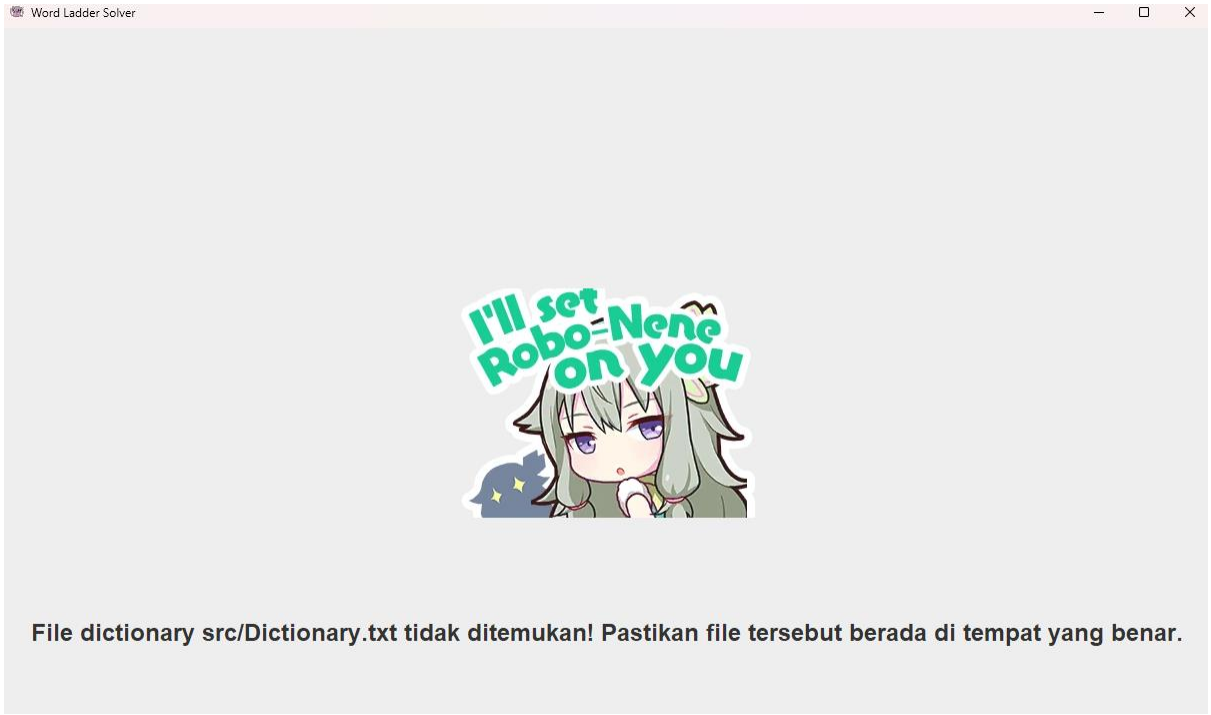
Untuk tugas ini, bonus GUI diimplementasikan dengan java swing.

```
You, 3 hours ago | 1 author (You)
public class WordLadderSolverGUI implements ActionListener{
    private JFrame window;
    private JPanel panel;
    private JPanel panel2;
    private JPanel panel3;
    private JButton UCS;
    private JButton GBFS;
    private JButton Astar;
    private JTextField startWord;
    private JTextField endWord;
    private ImageIcon icon;
    private ImageIcon success;
    private ImageIcon fail;
    private DictReader dictionary;
    private UCS ucs;
    private GBFS gbfs;
    private Astar astar;
```

GUI program menggunakan JFrame dengan 3 JPanel dan 3 JButton. Pengguna akan melakukan input kata awal dan akhir pada dua JTextField yang berbeda, lalu program akan dijalankan setelah pengguna memencet JButton algoritma yang diinginkan. Berikut adalah *window* dari program dan *source code* yang men-*set-up* window tersebut.

```java
public WordLadderSolverGUI() {
    this.window = new JFrame();
    window.setTitle(title:"Word Ladder Solver");
    window.setSize(width:1200, height:800);
    icon = new ImageIcon(filename:"elements/icon.png");
    window.setIconImage(this.icon.getImage());
    success = new ImageIcon(filename:"elements/search_success.png");
    fail = new ImageIcon(filename:"elements/search_fail.png");

    this.dictionary = new DictReader(path:"Dictionary.txt");
    if (!this.dictionary.dictValid()) {
        JPanel failPanel = new JPanel();
        failPanel.setBorder(BorderFactory.createEmptyBorder(top:30,left:30,bottom:30,right:30));
        failPanel.setLayout(new GridLayout(rows:0, cols:1));
        JLabel failLabel = new JLabel(text:"File dictionary src/Dictionary.txt tidak ditemukan!\n Pastikan file tersebut berada di tempat yang benar.");
        System.out.println(failLabel.getFont());
        failLabel.setFont(new Font(name:"Dialog", Font.BOLD, size:24));
        failPanel.add(new Box(axis:0));
        failPanel.add(new JLabel(fail,horizontalAlignment:0));
        failPanel.add(failLabel);
        window.add(failPanel, BorderLayout.CENTER);
    }
```

Program akan membuat sebuah obyek DictReader dengan file di lokasi “src/Dictionary.txt”. Jika file tidak ditemukan, program akan memunculkan window tersebut dan pengguna tidak akan bisa melakukan apa-apa.



Jika file ditemukan, program akan melanjutkan *set-up*.

```
    else {
        this.dictionary.makeList();

        startWord = new JTextField();
        JLabel startLabel = new JLabel(text:"Kata awal");
        startLabel.setLabelFor(startWord);
        endWord = new JTextField();
        JLabel endLabel = new JLabel(text:"Kata akhir");
        endLabel.setLabelFor(endWord);
        UCS = new JButton(text:"UCS");
        UCS.addActionListener(this);
        UCS.setSize(width:40, height:20);
        GBFS = new JButton(text:"Greedy BFS");
        GBFS.addActionListener(this);
        Astar = new JButton(text:"A*");
        Astar.addActionListener(this);

        ucs = new UCS();
        gbfs = new GBFS();
        astar = new Astar();
```

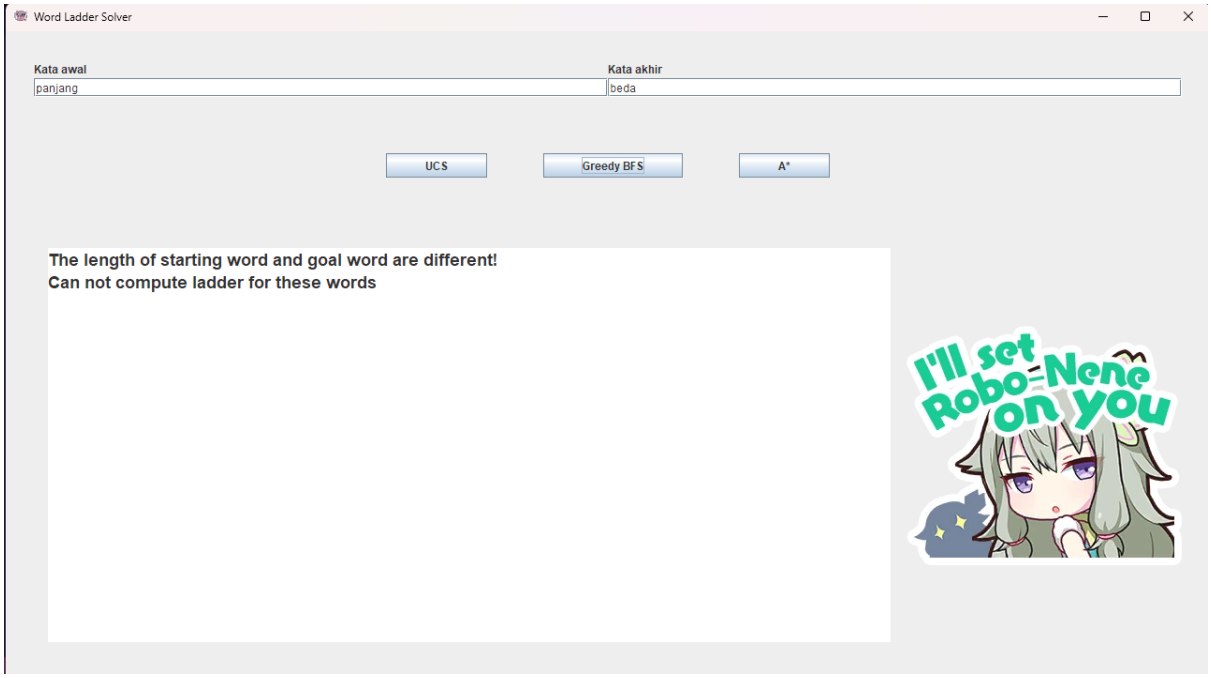
```
        panel = new JPanel();
        panel.setBorder(BorderFactory.createEmptyBorder(top:30, left:30, bottom:30, right:30));
        panel.setLayout(new GridLayout(rows:0, cols:2));
        panel.add(startLabel);
        panel.add(endLabel);
        panel.add(startWord);
        panel.add(endWord);

        panel2 = new JPanel();
        panel2.setBorder(BorderFactory.createEmptyBorder(top:30, left:30, bottom:30, right:30));
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        //gbc.gridwidth = 3;
        gbc.ipadx = 50;
        gbc.insets = new Insets(top:0, left:30, bottom:0, right:30);
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel2.setLayout(grid);
        panel2.add(UCS, gbc);
        gbc.gridx = 1;
        panel2.add(GBFS, gbc);
        gbc.gridx = 2;
        panel2.add(Astar, gbc);
```

```
        panel3 = new JPanel();
        panel3.add(new Box(axis:0));

        window.add(panel,BorderLayout.NORTH);
        window.add(panel2, BorderLayout.CENTER);
        window.add(panel3, BorderLayout.SOUTH);
    }
    window.pack();
    window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    window.setVisible(b:true);
}
```

Jika salah satu JButton ditekan, program akan men-*set-up* sebuah JTextArea dan JPanel baru untuk menampilkan jawaban.

```
public void actionPerformed(ActionEvent e) {
    TreeNode startNode = new TreeNode(startWord.getText());
    String end = endWord.getText();

    StringBuilder forLabel = new StringBuilder();
    JTextArea ta = new JTextArea();
    ta.setFont(new Font(name:"Dialog", Font.BOLD, size:18));
    ta.setEditable(b:false);
    ta.setLineWrap(wrap:true);
    ta.setWrapStyleWord(word:true);
    ta.setPreferredSize(new Dimension(width:900,height:400));

    panel3.removeAll();
    panel3.revalidate();
    panel3.setBorder(BorderFactory.createEmptyBorder(top:30, left:30, bottom:30, right:30));
    GridBagLayout gh = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();
    panel3.setLayout(gh);
    gbc.ipady = 20;
    gbc.gridwidth = 3;
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.insets = new Insets(top:15, left:15, bottom:15, right:15);
```

Jika panjang kata awal dan kata akhir berbeda, program akan memunculkan tampilan sebagai berikut.



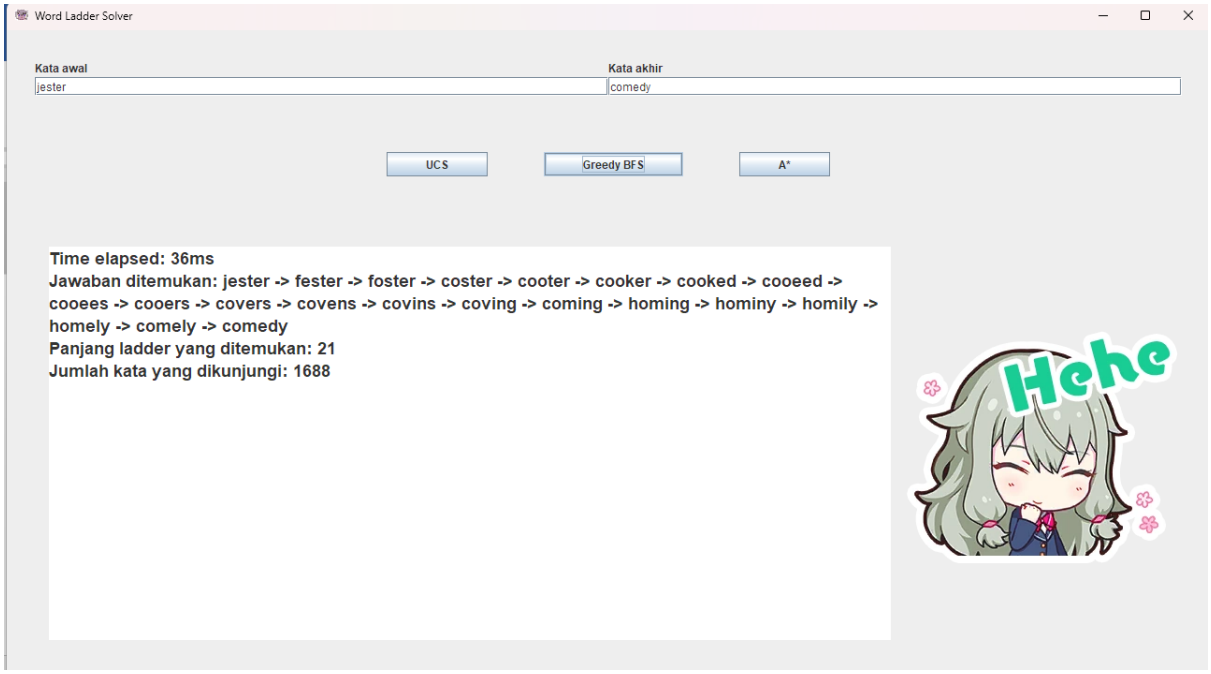
```
if (startNode.getRoot().length() != end.length()) {
    ta.setText(t:"The length of starting word and goal word are different!\nCan not compute ladder for these words");
    panel3.add(ta, gbc);
    gbc.gridwidth = 1;
    gbc.gridx = 3;
    panel3.add(new JLabel(fail));
}

else {
    HashMap<Boolean, String> res = new HashMap<Boolean,String>();
    if (e.getSource() == UCS) {
        res = ucs.run(startNode, end, this.dictionary);
    }
    if (e.getSource() == GBFS) {
        res = gbfs.run(startNode, end, this.dictionary);
    }
    if (e.getSource() == Astar) {
        res = astar.run(startNode, end, this.dictionary);
    }
```

Jika panjang kedua kata sama, program akan menjalankan algoritma yang diminta, lalu memasukkan hasilnya ke JTextArea dan JPanel yang diinginkan. Gambar di samping lokasi teks jawaban akan berubah tergantung jawabannya didapatkan atau tidak.

```
        if (res.containsKey(key:false)) {
            forLabel.append(res.get(key:false));

            ta.setText(forLabel.toString());
            panel3.add(ta, gbc);
            gbc.gridwidth = 1;
            gbc.gridy = 3;
            panel3.add(new JLabel(fail));
            res.remove(key:false);
        }
        else if (res.containsKey(key:true)) {
            forLabel.append(res.get(key:true));

            ta.setText(forLabel.toString());
            panel3.add(ta, gbc);
            gbc.gridwidth = 1;
            gbc.gridx = 3;
            panel3.add(new JLabel(success));
            res.remove(key:true);
        }
    }
    panel3.revalidate();
    window.pack();
}       You, 17 hours ago • feat: nene brainrot (finished GUI)
```

Berikut adalah tampilan jika tidak didapatkan jawawab, baik karena kata tidak ditemukan di dalam kamus atau karena memang tidak ada jalan:

Berikut adalah tampilan jika jalan ditemukan:



Lampiran

Link ke *repository*:
https://github.com/florars/Tucil3_13522010

Tabel:

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	✓	