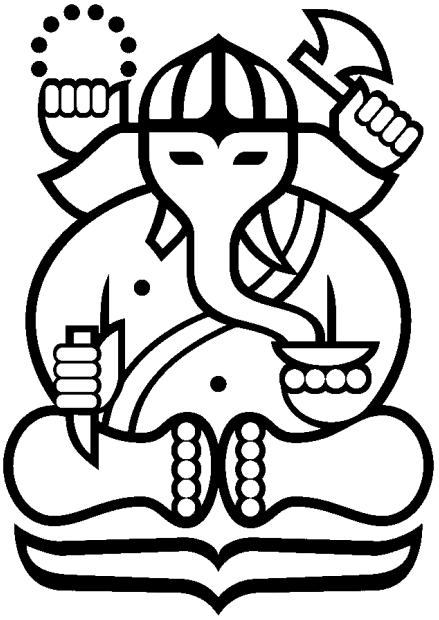
TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2023/2024

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



OLEH:

Diero Arga Purnama 13522056

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I
DESKRIPSI TUGAS

Word Ladder adalah permainan kata yang ditemukan oleh Lewis Carroll pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yaitu *start word* dan *end word* yang memiliki banyak huruf yang sama. Tugas pemain adalah untuk menemukan rantai kata yang menghubungkan *start word* dengan *end word*. Tiap kata yang berdekatan dalam rantai kata hanya boleh berbeda satu huruf saja. Pada permainan ini, solusi yang dicari adalah solusi optimal, yaitu solusi yang memiliki jumlah kata dalam rantai kata yang minimal.

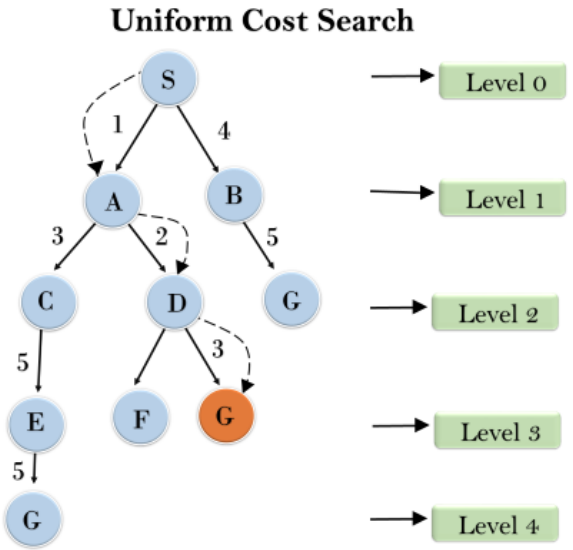
Gambar 1.1 Contoh Permainan *Word Ladder*

Pada Tugas Kecil 3 ini, penulis merancang program yang dapat menyelesaikan permainan *Word Ladder* menggunakan algoritma *Uniform Cost Search* (UCS), *Greedy Best First Search* dan A* dalam bahasa Java.

BAB II
LANDASAN TEORI

2.1. Algoritma *Uniform Cost Search* (UCS)

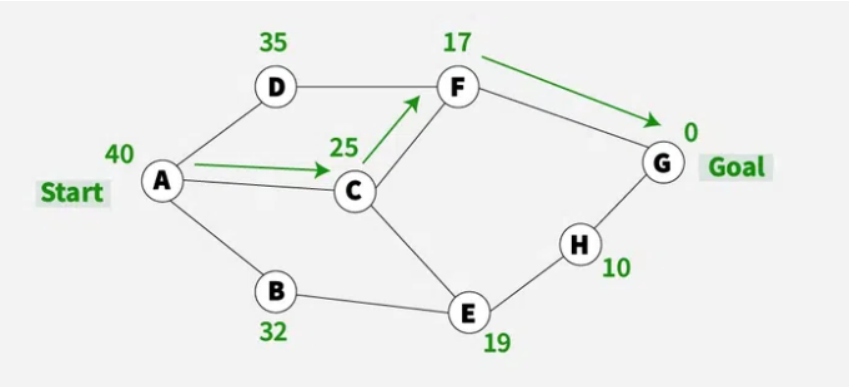
Algoritma *Uniform Cost Search* (UCS) adalah algoritma pencarian dalam graf berbobot. Pencarian menggunakan algoritma ini akan mengutamakan simpul-simpul yang memiliki biaya yang paling rendah. Algoritma UCS berguna untuk menemukan rute yang memiliki total bobot yang minimal antara dua simpul dalam graf berbobot.



Gambar 2.1 Contoh Alur Algoritma *Uniform Cost Search* (UCS)

2.2. Algoritma *Greedy Best First Search*

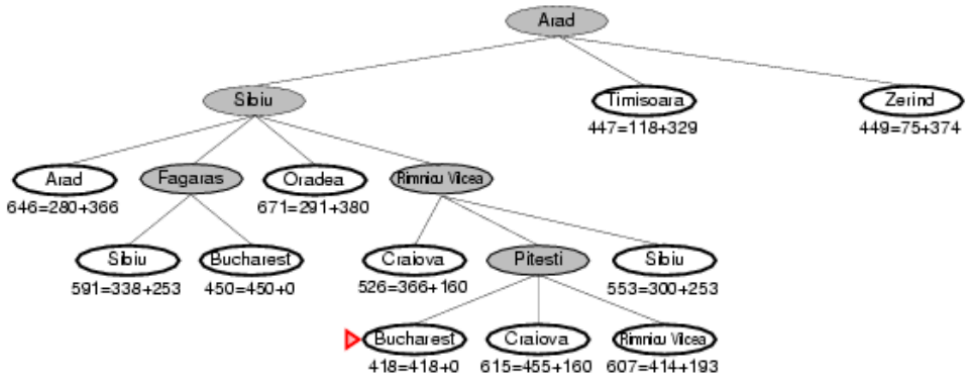
Algoritma *greedy best first search* adalah algoritma pencarian dalam graf yang menggunakan pendekatan heuristik untuk menentukan simpul yang akan dikunjungi berikutnya. Algoritma ini akan mengutamakan simpul yang memiliki nilai heuristik yang terkecil tanpa mempertimbangkan langkah berikutnya dan bobot yang diambil. Karena pada dasarnya algoritma ini merupakan algoritma *greedy*, solusi yang ditemukan belum tentu merupakan solusi yang optimal.



Gambar 2.2 Contoh Greedy Best First Search

2.3. Algoritma A*

Algoritma A* adalah algoritma pencarian dalam graf berbobot yang menggabungkan pendekatan Greedy Best First search dengan pendekatan Uniform Cost Search (UCS). Algoritma ini akan mengutamakan simpul yang memiliki jumlah nilai heuristik dengan *cost* yang terendah. Tujuan dari algoritma ini adalah untuk menemukan rute terpendek dengan *cost* yang minimum.



Gambar 2.3 Contoh A*

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Penyelesaian Masalah

Masalah yang ingin dipecahkan oleh program yang dibuat adalah mencari rantai rantai kata yang menghubungkan *start word* dengan *end word*. Tahapan yang diambil program untuk menyelesaikan masalah adalah dengan menelusuri graf berbobot. Graf berbobot yang digunakan memiliki sebuah kata bahasa inggris sebagai simpul dan semua simpul yang bertetangga hanya memiliki perbedaan satu huruf. Program dapat menyelesaikan masalah menggunakan 3 algoritma, yaitu algoritma *Uniform Cost Search* (UCS), *Greedy Best First Search*, dan A*.

Algoritma-algoritma tersebut menyelesaikan masalah dengan cara yang berbeda-beda, tetapi terdapat beberapa langkah yang diambil oleh semua algoritma sebelum memulai pencarian, yaitu:

1) Buat sebuah objek Priority Queue yang akan diisi oleh pasangan kata dan nilai f(n)-nya, dengan nilai f(n) merupakan hasil penjumlahan dari g(n), yaitu jumlah langkah selama ini yang telah diambil untuk mencapai *n*, dengan h(n), yaitu estimasi langkah yang diperlukan untuk mencapai *end word* dari *n*. Untuk algoritma UCS, nilai f(n) sama dengan g(n), sedangkan untuk algoritma *Greedy Best First Search*, nilai f(n) sama dengan h(n). Priority queue akan mengurutkan semua elemen berdasarkan nilai f(n)-nya, dengan urutan terdepan diambil oleh elemen dengan nilai f(n) terkecil.
2) Buat sebuah objek HashSet untuk menyimpan semua kata-kata yang telah diekspansi, hal ini dilakukan agar program tidak melakukan ekspansi terhadap kata-kata yang sama secara terus menerus.

3.1.1 Penyelesaian Masalah dengan Algoritma UCS

Langkah penyelesaian masalah dengan menggunakan algoritma UCS adalah sebagai berikut:

1) Masukkan *start word* ke dalam queue dan mulai ekspansi, program akan mencari semua kata-kata bahasa inggris valid yang memiliki jumlah huruf yang sama dan memiliki perbedaan satu huruf dengan *start word*. Untuk

tiap-tiap kata yang ditemukan, akan dihitung nilai g(n) untuk kata tersebut dan dimasukkan ke dalam queue.

2) Ambil elemen terdepan dalam queue dan lakukan ekspansi yang sama terhadap elemen tersebut.
3) Program akan melakukan loop sampai ketemu kata yang sama dengan end word atau queue kosong.
4) Jika ditemukan end word dari start word, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.
5) Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari start word dan berakhir di end word. Program akan berakhir dan mengeluarkan pesan "Algoritma Uniform Cost Search (UCS) tidak menemukan jalan dari <*start word*> ke <*end word*>".

Dalam algoritma UCS, urutan simpul yang diekspansi akan ditentukan oleh nilai g(n) simpul tersebut, yaitu jumlah langkah yang diambil dari simpul awal menuju simpul tersebut. Karena nilai g(n) sama dengan nilai g(n) simpul sebelumnya ditambah satu, urutan ekspansi simpul pasti akan dilakukan secara melebar. Hal ini menyebabkan untuk kasus permain *Word Ladder*, pencarian solusi dengan menggunakan algoritma UCS dapat dibilang sama dengan pencarian dengan menggunakan algoritma *Breadth-First Search* (BFS) karena urutan simpul yang diekspansi dan rute yang dihasilkan akan sama.

3.1.2 Penyelesaian Masalah dengan Algoritma *Greedy Best First Search*

Langkah penyelesaian masalah dengan menggunakan algoritma *Greedy Best First Search* adalah sebagai berikut:

1) Masukkan *start word* ke dalam queue dan mulai ekspansi, program akan mencari semua kata-kata bahasa inggris valid yang memiliki jumlah huruf yang sama dan memiliki perbedaan satu huruf dengan *start word*. Untuk tiap-tiap kata yang ditemukan, akan dihitung nilai h(n) untuk kata tersebut. Nilai h(n) didapatkan dengan menghitung jumlah huruf yang berbeda antara *n* dengan *end word*. Hanya kata dengan nilai h(n) yang paling rendah akan dimasukkan ke dalam queue.

2) Ambil elemen terdepan dalam queue dan lakukan ekspansi yang sama terhadap elemen tersebut.
3) Program akan melakukan *loop* sampai ketemu kata yang sama dengan *end word* atau queue kosong.
4) Jika ditemukan *end word* dari *start word*, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.
5) Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari *start word* dan berakhir di *end word*. Program akan berakhir dan mengeluarkan pesan "Algoritma *Greedy Best First Search* tidak menemukan jalan dari <*start word*> ke <*end word*>".

Algoritma *Greedy Best First Search*, tidak bisa menjamin solusi optimal untuk persoalan *Word Ladder*. Secara teoritis, terdapat beberapa alasan mengapa hal ini terjadi, yang pertama adalah bahwa algoritma *greedy* tidak dapat melakukan *backtracking*, sehingga algoritma tidak dapat mempertimbangkan bagaimana langkah yang diambil mempengaruhi solusi secara keseluruhan. Alasan yang kedua adalah bahwa nilai heuristik tidak akurat dalam menentukan jumlah langkah yang dibutuhkan untuk mencapai *end word* dari suatu kata. Alasan terakhir adalah bahwa algoritma *greedy* hanya mencoba satu jalur tanpa mempertimbangkan kemungkinan adanya jalur lain yang memberikan solusi yang lebih optimal.

3.1.3 Penyelesaian Masalah dengan Algoritma A*

Algoritma A* mengambil langkah-langkah yang sama dengan algoritma UCS dengan perbedaan pengurutan elemen dalam queue berdasarkan hasil penjumlahan dari nilai g(n) dengan nilai h(n) dibandingkan dengan UCS yang hanya menggunakan nilai g(n). Langkah-langkah penyelesaian masalah dengan algoritma A* adalah sebagai berikut:

1) Masukkan *start word* ke dalam queue dan mulai ekspansi, program akan mencari semua kata-kata bahasa inggris valid yang memiliki jumlah huruf yang sama dan memiliki perbedaan satu huruf dengan *start word*. Untuk tiap-tiap kata yang ditemukan, akan dihitung nilai f(n) untuk kata tersebut dan dimasukkan ke dalam queue.
2) Ambil elemen terdepan dalam queue dan lakukan ekspansi yang sama terhadap elemen tersebut.

3) Program akan melakukan loop sampai ketemu kata yang sama dengan end word atau queue kosong.
4) Jika ditemukan end word dari start word, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.
5) Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari start word dan berakhir di end word. Program akan berakhir dan mengeluarkan pesan "Algoritma A* tidak menemukan jalan dari <*start word*> ke <*end word*>".

Algoritma A* menggunakan nilai hasil penjumlahan nilai heuristik h(n) dan f(n) untuk menentukan simpul yang akan diekspansi berikutnya. Oleh karena itu, heuristik yang digunakan adalah heuristik *admissible*, dalam kasus *word ladder* hal ini berarti h(n) tidak pernah meng-*overestimate* jumlah langkah yang dibutuhkan untuk mencapai *end word*. Dalam program ini, nilai heuristik sebuah kata didapatkan dari perhitungan jumlah huruf yang berbeda antara kata tersebut dengan *end word*. Pendekatan ini adalah *admissible* karena nilai h(n) yang diberikan pasti kurang dari atau sama dengan jumlah langkah sebenarnya untuk mencapai *end word*.

Secara teoritis, algoritma A* akan lebih efisien dalam pencarian solusi optimal dalam permainan *word ladder* jika dibandingkan dengan algoritma UCS. Dibandingkan dengan algoritma UCS yang hanya menggunakan jumlah langkah dari simpul awal untuk mempertimbangkan urutan simpul yang akan diekspansi berikutnya, algoritma A* juga mempertimbangkan nilai heuristik simpul tersebut, hal ini berarti bahwa algoritma A* dapat memilih simpul berikutnya dengan lebih baik, mengurangi jumlah simpul yang harus diekspansi untuk mencapai tujuan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

Program terdiri dari 7 kelas, yaitu Main, Graf, UCS, Greedy, A, Word, dan WordIntPair.

4.1.1 Main

Kelas Main berfungsi untuk menjalankan program.

No.	Metode	Fungsi
1	public static void main(String[] args)	Menerima input dan memanggil algoritma UCS, *Greedy Best First Search*, atau A* sesuai dengan masukan pengguna.

Main.java

```
import java.io.IOException;
import java.util.Scanner;

import src.*;
import src.structs.Word;

/*
 * CARA RUN:
 * - javac -d bin *.java
 * - java -cp bin Main
 */

public class Main {
    public static void main(String[] args) {
        System.out.println("~=----------------------------=~");
        System.out.println(
                        "  _______          _ _    ____   \r\n" + //
                        " |__   __|        (_) | |___ \\ \r\n" + //
                        "    | |_   _  ___ _| |   __) |\r\n" + //
                        "    | | | | |/ __| | |  |__ < \r\n" + //
                        "    | | |_| | (__| | |  ___) |\r\n" + //
```

```
                        "    |_|\\__,_|\\___|_|_| |____/ ");
        System.out.println("\n Diero Arga Purnama (13522056)");
        System.out.println("~=----------------------------=~\n");

        Scanner input = new Scanner(System.in);
        while (true) {
            // Baca words.txt
            if (Word.validWords.size() == 0) {
                try {
                    Word.readWordFile();
                } catch (IOException e) {
                    System.out.println("Gagal buka words.txt");
                    break;
                }
            } else {
                System.out.print("Ketik [Y] untuk keluar dari
program: ");
                String exit = input.nextLine();
                if (exit.equals("Y")) {
                    System.out.println("Berhasil keluar dari
program!");
                    break;
                }
            }

            // Input
            System.out.print("Kata awal: ");
            String startWord = input.nextLine();
            System.out.print("Kata tujuan: ");
            String endWord = input.nextLine();

            if (startWord.length() != endWord.length()) {
                System.out.println("Kata awal & kata tujuan harus
memiliki panjang yang sama!");
                break;
            } else if (!Word.isValidWord(startWord.toLowerCase())
|| !Word.isValidWord(endWord.toLowerCase())) {
                System.out.println("Kata awal atau kata tujuan
tidak valid.");
                break;
            } else if (startWord.equals(endWord)) {
```

```
                System.out.println("Kata awal & kata tujuan harus
berbeda.");
                break;
            }

            System.out.println("Algoritma yang ingin digunakan:
");
            System.out.println("[1]. UCS");
            System.out.println("[2]. Greedy Best First Search");
            System.out.println("[3]. A*");
            System.out.print(">> ");

            int algoritma;
            try {
                algoritma = Integer.parseInt(input.nextLine());
                if (algoritma > 3 || algoritma < 1) {
                    System.out.println("Input harus merupakan 1,
2, atau 3.");
                    break;
                }
            } catch (NumberFormatException e) {
                System.out.println("Input harus merupakan 1, 2,
atau 3.");
                break;
            }

            // Algoritma
            long startTime = System.currentTimeMillis();

            Graf t;
            switch (algoritma) {
                case 1:
                    t = new UCS(startWord, endWord);
                    break;
                case 2:
                    t = new Greedy(startWord, endWord);
                    break;
                default:
                    t = new A(startWord, endWord);
            }
            t.start();
```

```
            long endTime = System.currentTimeMillis();
            long runtime = endTime - startTime;

            // Output
            System.out.println("Runtime: " + runtime + " ms\n");
        }
        input.close();
    }
}
```

4.1.2 Graf

Kelas graf merupakan kelas abstrak yang berfungsi sebagai *parent* dari kelas UCS, Greedy, dan A. Kelas ini memiliki 5 atribut, yaitu pq, digunakan untuk menentukan urutan ekspansi simpul, visited, digunakan untuk mencatat kata-kata yang telah diekspansi, startWord, yaitu kata awal, endWord, yaitu kata akhir, dan currentWord, yaitu kata yang sedang diekspansi. Algoritma UCS, *Greedy Best First Search*, dan A* dijadikan *child class* dari kelas ini karena garis besar ketiga algoritma sama hanya dibedakan pada nilai f(n) yang digunakan serta serta algoritma yang hanya menambahkan satu simpul ke dalam queue pada setiap iterasi dibandingkan dengan algoritma lainnya yang memasukkan semua simpul hidup yang ada ke dalam queue.

No.	Metode	Fungsi
1	public Graf(String startWord, String endWord)	Konstruktor, menginstansiasi sebuah objek kelas Graf.
2	public WordIntPair getHead()	Mengembalikan elemen pertama dalam queue.
3	public void add(WordIntPair val)	Menambahkan sebuah simpul ke akhir queue.
4	public void next() throws NullPointerException	Melakukan ekspansi terhadap elemen pertama queue. Menambahkan semua kata valid yang didapatkan dari mengubah satu huruf yang ada dalam elemen ke dalam queue. Melemparkan exception NullPointerException jika

		queue kosong.
5	public void start()	Memulai pencarian, memanggil fungsi next sampai ditemukan solusi atau queue kosong.
6	public abstract int calculateG(WordIntPair prev)	Menghitung g(n) simpul, didefinisikan dalam kelas anak.
7	public abstract int calculateH(Word word)	Menghitung h(n) simpul, didefinisikan dalam kelas anak.
8	public abstract int calcF(int g, String str)	Menghitung f(n) simpul, digunakan untuk menampilkan nilai f(n) ketika di print, didefinisikan dalam kelas anak.
9	public abstract String getName()	Mengembalikan nama algoritma, didefinisikan dalam kelas anak.

Graf.java

```
package src;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

import src.structs.WordIntPair;
import src.structs.Word;

public abstract class Graf {

    /*
     * Attributes
    */

    protected PriorityQueue<WordIntPair> pq;
    protected HashSet<String> visited;
    protected String startWord;
    protected String endWord;
    protected Word currentWord;
```

```
    /*
     * Methods
    */

    // Ctor
    public Graf(String startWord, String endWord) {
        visited = new HashSet<>();
        pq = new
PriorityQueue<>(Comparator.comparing(WordIntPair::getSecond));

        this.startWord = startWord.toLowerCase();
        this.endWord = endWord.toLowerCase();
        this.currentWord = new Word(this.startWord);
    }

    // Getter
    public WordIntPair getHead() {
        // Mengembalikan elemen pertama dalam priority queue
        // dan menghapus elemen tsb dari priority queue.
        return pq.poll();
    }

    // Lainnya
    public void add(WordIntPair val) {
        // Menambahkan tuple (first, second) ke tail priority
queue
        pq.offer(val);
    }

    public void next() throws NullPointerException {
        // Cari semua kata berikutnya yang valid kemudian
        // tambahkan ke tail priority queue.

        if (pq.size() == 0) {
            throw (new NullPointerException());
        }

        WordIntPair head = getHead();
        currentWord = head.getFirst();

        // Kembali kalau sudah pernah di-ekspan
```

```
        if (visited.contains(currentWord.getStr()))
            return;
        visited.add(currentWord.getStr());

        ArrayList<Word> nextWords =
head.getFirst().getNextWords();
        for (Word el : nextWords) {
            // Tambahkan kata-kata yang belum pernah di-ekspan ke
dalam queue
            if (visited.contains(el.getStr()))
                continue;

            WordIntPair currentTuple = new WordIntPair(el,
calculateG(head) + calculateH(el));
            add(currentTuple);
        }
    }

    public void start() {
        // Mulai pencarian

        while (true) {
            // Berhenti ketika sudah sampai endWord
            if (currentWord.getStr().equals(endWord)) {
                // Print rute
                System.out.println();
                int cnt = 0;
                for (String el: currentWord.getPrev()) {
                    System.out.print(el + " (" + calcF(cnt, el) +
")" + " -> ");
                    cnt++;
                }
                System.out.println(currentWord.getStr() + " (" +
calcF(cnt, currentWord.getStr()) + ")");
                // Print jumlah node yang dikunjungi
                System.out.println("Jumlah node yang dikunjungi: "
+ currentWord.getPrev().size());

                break;
            }
```

```
            try {
                next();
            } catch (NullPointerException e) {
                System.out.println("\nAlgoritma " + getName() + "
tidak menemukan jalan dari " + startWord + " ke " + endWord +
".");
                break;
            }
        }
    }

    public abstract int calculateG(WordIntPair prev);

    public abstract int calculateH(Word word);

    // Output
    public abstract int calcF(int g, String str);

    public abstract String getName();
}
```

4.1.3 UCS

Kelas UCS merupakan kelas turunan kelas Graf. Kelas ini bertugas untuk mencari solusi dari permainan *Word Ladder* dengan menggunakan algoritma *Uniform Cost Search*.

No.	Metode	Fungsi
1	public UCS(String startWord, String endWord)	Konstruktor, menginstansiasi sebuah objek kelas UCS
2	public int calculateG(WordIntPair prev)	Mengembalikan nilai g(n), yaitu nilai g(n - 1) + 1
3	public int calculateH(Word word)	Mengembalikan nilai h(n), karena dalam algoritma UCS, f(n) = g(n), nilai h(n) yang dikembalikan adalah 0.
4	public int calcF(int g, String str)	Mengembalikan nilai f(n).

5	public String getName()	Mengembalikan “Uniform Cost Search (UCS)”

UCS.java

```java
package src;

import src.structs.*;

public class UCS extends Graf {

    /*
     * Methods
    */

    // Ctor
    public UCS(String startWord, String endWord) {
        super(startWord, endWord);

        pq.add(new WordIntPair(this.currentWord, 0));
    }

    public int calculateG(WordIntPair prev) {
        // Mengembalikan nilai g(n), yaitu
        // jumlah langkah yang sudah diambil
        return (prev.getSecond() + 1);
    }

    public int calculateH(Word word) {
        // Mengembalikan nilai h(n)
        // Karena dalam UCS f(n) = g(n), h(n) = 0
        return 0;
    }

    // Output
    public int calcF(int g, String str) {
        // Hitung "f(n)" untuk print
        return (g);
    }
```

```
    public String getName() {
        // Mengembalikan nama algoritma untuk print
        return "Uniform Cost Search (UCS)";
    }
}
```

4.1.4 Greedy

Kelas Greedy merupakan kelas turunan kelas Graf. Kelas ini bertugas untuk mencari solusi dari permainan *Word Ladder* dengan menggunakan algoritma *Greedy Best First Search*.

No.	Metode	Fungsi
1	public Greedy(String startWord, String endWord)	Konstruktor, menginstansiasi sebuah objek kelas Greedy
2	public void next() throws NullPointerException	Meng-override method next dalam Graf. Mencari simpul yang memiliki nilai h(n) terendah kemudian ditambahkan ke queue.
3	public int calculateG(WordIntPair prev)	Mengembalikan nilai g(n), karena dalam algoritma *Greedy Best First Search* nilai f(n) = h(n), g(n) yang dikembalikan adalah 0.
4	public int calculateH(Word word)	Mengembalikan nilai h(n), yaitu jumlah huruf berbeda dalam word dibandingkan dengan *end word*.
5	public int calcF(int g, String str)	Mengembalikan nilai f(n).
6	public String getName()	Mengembalikan “Greedy Best First Search”

Greedy.java

```
package src;

import java.util.ArrayList;

import src.structs.WordIntPair;
import src.structs.Word;
```

```
public class Greedy extends Graf {

    /*
     * Methods
    */

    // Ctor
    public Greedy(String startWord, String endWord) {
        super(startWord, endWord);

        pq.add(new WordIntPair(this.currentWord,
calculateH(this.currentWord)));
    }


    public void next() throws NullPointerException {
        // Cari kata berikutnya yang memiliki h(n) terendah
        // tambahkan ke tail priority queue.

        if (pq.size() == 0) {
            throw (new NullPointerException());
        }

        WordIntPair head = getHead();
        currentWord = head.getFirst();

        // Kembali kalau sudah pernah di-ekspan
        if (visited.contains(currentWord.getStr()))
           return;
        visited.add(currentWord.getStr());

        ArrayList<Word> nextWords =
head.getFirst().getNextWords();
        WordIntPair min = null;
        for (Word el : nextWords) {
            // Tambahkan kata-kata yang memiliki nilai heuristik
terendah
            if (visited.contains(el.getStr()))
                continue;

            WordIntPair currentTuple = new WordIntPair(el,
```

```
calculateG(head) + calculateH(el));
            if (min == null)
                min = currentTuple;
            else
                if (min.getSecond() > currentTuple.getSecond())
                    min = currentTuple;
        }

        if (min != null)
            add(min);
    }

    public int calculateH(Word word) {
        // Mengembalikan nilai h(n), yaitu
        // jumlah huruf berbeda antara word dengan endWord
        char[] charsWord = word.getStr().toCharArray();
        char[] charsEndWord = endWord.toCharArray();

        int cnt = 0;
        for (int i = 0; i < charsWord.length; i++) {
            if (charsWord[i] != charsEndWord[i])
            cnt++;
        }

        return cnt;
    }

    public int calculateG(WordIntPair prev) {
        // Mengembalikan nilai g(n), karena dalam
        // greedy best first search f(n) = h(n),
        // g(n) = 0
        return 0;
    }

    // Output
    public int calcF(int g, String str) {
        // Hitung "f(n)" untuk print
        return (calculateH(new Word(str)));
    }

    public String getName() {
```

```
        // Mengembalikan nama algoritma untuk print
        return "Greedy Best First Search";
    }
}
```

4.1.5 A

Kelas A merupakan kelas turunan kelas Graf. Kelas ini bertugas untuk mencari solusi dari permainan *Word Ladder* dengan menggunakan algoritma A*.

No.	Metode	Fungsi
1	public A(String startWord, String endWord)	Konstruktor, menginstansiasi sebuah objek kelas A
2	public int calculateG(WordIntPair prev)	Mengembalikan nilai g(n), yaitu nilai g(n - 1) + 1
3	public int calculateH(Word word)	Mengembalikan nilai h(n), yaitu jumlah huruf berbeda dalam word dibandingkan dengan *end word*.
4	public int calcF(int g, String str)	Mengembalikan nilai f(n).
5	public String getName()	Mengembalikan “A*”

A.java

```
package src;

import src.structs.WordIntPair;
import src.structs.Word;

public class A extends Graf {

    /*
     * Methods
    */

    // Ctor
    public A(String startWord, String endWord) {
        super(startWord, endWord);
```

```
        pq.add(new WordIntPair(this.currentWord,
calculateH(currentWord)));
    }

    public int calculateG(WordIntPair prev) {
        // Mengembalikan nilai g(n), yaitu
        // jumlah langkah yang sudah diambil
        return (prev.getSecond() + 1);
    }

    public int calculateH(Word word) {
        // Mengembalikan nilai h(n), yaitu
        // jumlah huruf berbeda antara word dengan endWord
        char[] charsWord = word.getStr().toCharArray();
        char[] charsEndWord = endWord.toCharArray();

        int cnt = 0;
        for (int i = 0; i < charsWord.length; i++) {
            if (charsWord[i] != charsEndWord[i])
                cnt++;
        }

        return cnt;
    }

    // Output
    public int calcF(int g, String str) {
        // Hitung "f(n)" untuk print
        return (calculateH(new Word(str)) + g);
    }

    public String getName() {
        // Mengembalikan nama algoritma untuk print
        return "A*";
    }
}
```

4.1.6 WordIntPair

Kelas WordIntPair adalah sebuah kelas yang terdiri atas pasangan sebuah Word, dan sebuah Integer. Kelas ini digunakan simpul dalam kelas Graf.

No.	Metode	Fungsi
1	Public WordIntPair(Word first, int second)	Konstruktor, menginstansiasi sebuah objek kelas WordIntPair
2	public Word getFirst()	Mengembalikan atribut first.
3	public int getSecond()	Mengembalikan atribut second.
4	public void setFirst(String val)	Mengubah nilai atribut first.
5	public void setSecond()	Mengubah nilai atribut second.

WordIntPair.java

```
package src.structs;

public class WordIntPair {

    /*
     * Attributes
    */

    private Word first;
    private int second;

    /*
     * Methods
    */

    // Ctor
    public WordIntPair(Word first, int second) {
        this.first = first;
        this.second = second;
    }

    // Getter
    public Word getFirst() {
        return this.first;
```

```
    }

    public int getSecond() {
        return this.second;
    }

    // Setter
    public void setFirst(String val) {
        this.first = new Word(val);
    }

    public void setSecond(int val) {
        this.second = val;
    }
}
```

4.1.7 Word

Kelas Word adalah kelas yang menyimpan set semua kata bahasa inggris yang valid, string, dan rute yang diambil untuk mencapai kata tersebut dari *start word*.

No.	Metode	Fungsi
1	public Word(String str)	Konstruktor, menginstansiasi sebuah objek kelas Word
2	public String getStr()	Mengembalikan atribut str
3	public ArrayList<String> prev()	Mengembalikan atribut prev
4	public void setStr(String val)	Mengubah nilai atribut str
5	public void setPrev(Word prevWord)	Mengubah nilai atribut prev
6	public static void readWordFile() throws IOException	Membaca words.txt, melemparkan IOException jika file tidak ditemukan
7	public static boolean isValidWord(String str)	Mengembalikan true jika str terdapat dalam set validWords
8	public ArrayList<Word> getNextWords()	Mengembalikan list yang berisi semua kata valid yang didapatkan dari mengubah satu huruf dalam word.

Word.java

```java
package src.structs;

import java.util.Set;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;

public class Word {

    /*
     * Attributes
    */

    public static Set<String> validWords = new HashSet<>();
    private String str;
    private ArrayList<String> prev;

    /*
     * Methods
    */

    // Ctor
    public Word(String str) {
        this.str = str;
        this.prev = new ArrayList<>();
    }

    // Getter
    public String getStr() {
        return this.str;
    }

    public ArrayList<String> getPrev() {
        return this.prev;
    }
```

```
    // Setter
    public void setStr(String val) {
        this.str = val;
    }

    public void setPrev(Word prevWord) {
        this.prev.addAll(prevWord.getPrev());
        this.prev.add(prevWord.getStr());
    }

    // Lain-lain
    public static void readWordFile() throws IOException {
        // Baca words.txt dan simpan dalam validWords

        BufferedReader reader = new BufferedReader(new
FileReader("src/words.txt"));

        String s;
        while ((s = reader.readLine()) != null)
            validWords.add(s.toLowerCase());

        reader.close();
    }

    public static boolean isValidWord(String str) {
        // Mengembalikan true jika str merupakan kata
        // yang valid dalam bahasa inggris

        return validWords.contains(str);
    }

    public ArrayList<Word> getNextWords() {
        // Mengembalikan array yang berisi semua kata-kata dalam
        // bahasa inggris valid dan tidak ada dalam prev yang
didapatkan
        // dari mengganti satu huruf dalam str menjadi huruf lain.

        ArrayList<Word> ret = new ArrayList<>();
        char[] chars = str.toCharArray();
        for (int i = 0; i < chars.length; i++) {
```

```
            for (char c = 'a'; c != '{'; c++) {
                if (c == chars[i])
                    continue;

                char[] tempChars = Arrays.copyOf(chars,
chars.length);
                tempChars[i] = c;

                // Masukkan dalam ret jika tempWord tidak ada
dalam prev & merupakan kata valid
                Word tempWord = new Word(new String(tempChars));
                if (!getPrev().contains(tempWord.getStr()) &&
isValidWord(tempWord.getStr())) {
                    tempWord.setPrev(this);
                    ret.add(tempWord);
                }
            }
        }
        return ret;
    }
}
```

4.2 Hasil Pengujian

Untuk menguji program yang telah dibuat, dilakukan pengujian dengan berbagai data uji sebagai berikut:

No	*Start Word*	*End Word*
1	bad	dad
2	dad	tea
3	sand	grit
4	smile	tarot
5	before	county
6	boredom	control

a. Test Case 1

```
Kata awal: bad
Kata tujuan: dad
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 1

bad (0) -> dad (1)
Jumlah node yang dikunjungi: 1
Runtime: 18 ms
```

Gambar 4.2.1 Test Case 1 (UCS)

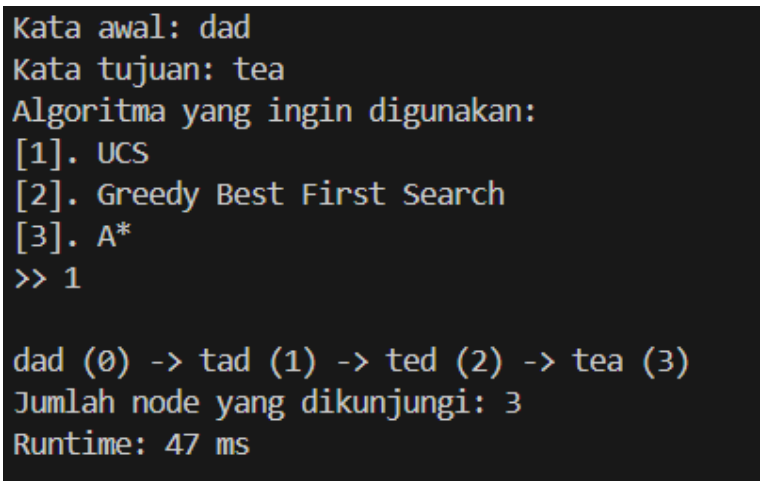
```
Kata awal: bad
Kata tujuan: dad
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 2

bad (1) -> dad (0)
Jumlah node yang dikunjungi: 1
Runtime: 2 ms
```

Gambar 4.2.2 Test Case 1 (*Greedy Best First Search*)

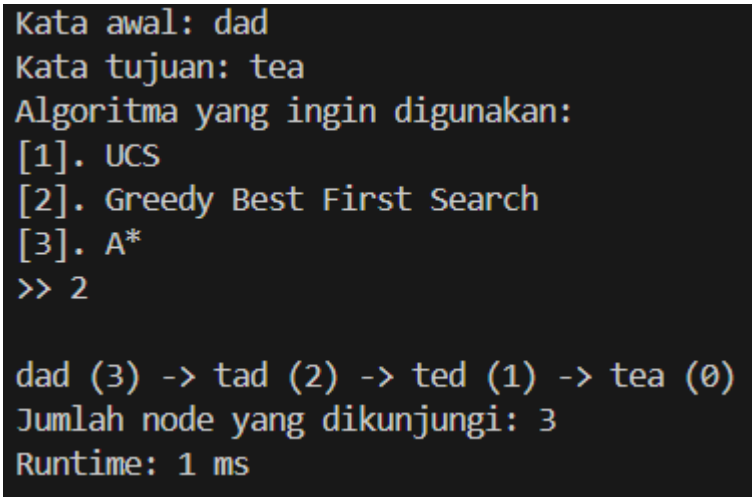
```
Kata awal: bad
Kata tujuan: dad
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 3

bad (1) -> dad (1)
Jumlah node yang dikunjungi: 1
Runtime: 1 ms
```

Gambar 4.2.3 Test Case 1 (A*)

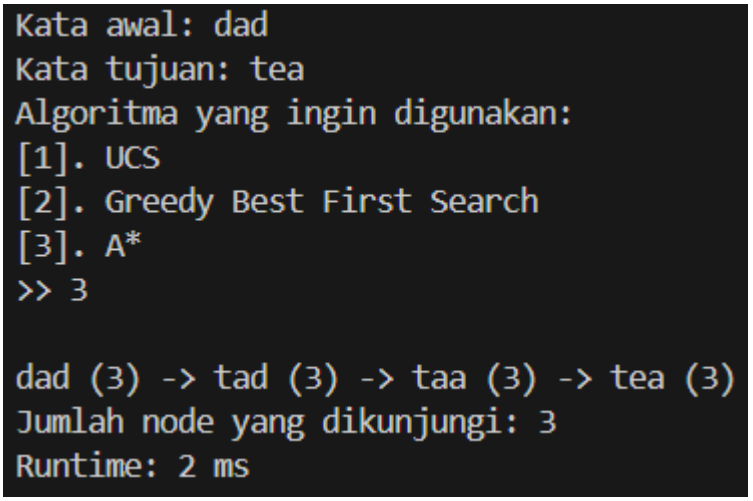
b. Test Case 2



Gambar 4.2.4 Test Case 2 (UCS)



Gambar 4.2.5 Test Case 2 (*Greedy Best First Search*)



Gambar 4.2.6 Test Case 2 (A*)

c. Test Case 3

```
Kata awal: sand
Kata tujuan: grit
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 1

sand (0) -> sant (1) -> gant (2) -> gait (3) -> grit (4)
Jumlah node yang dikunjungi: 4
Runtime: 77 ms
```

Gambar 4.2.7 Test Case 3 (UCS)

```
Kata awal: sand
Kata tujuan: grit
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 2

sand (4) -> said (3) -> caid (3) -> kaid (3) -> laid (3) -> lait (2) -> gait (1) -> grit (0)
Jumlah node yang dikunjungi: 7
Runtime: 1 ms
```

Gambar 4.2.8 Test Case 3 (*Greedy Best First Search*)

```
Kata awal: sand
Kata tujuan: grit
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 3

sand (4) -> sant (4) -> gant (4) -> gait (4) -> grit (4)
Jumlah node yang dikunjungi: 4
Runtime: 1 ms
```

Gambar 4.2.9 Test Case 3 (A*)

d. Test Case 4

```
Kata awal: smile
Kata tujuan: tarot
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 1

smile (0) -> smily (1) -> saily (2) -> taily (3) -> tails (4) -> tains (5) -> tarns (6) -> taros (7) -> tarot (8)
Jumlah node yang dikunjungi: 8
Runtime: 129 ms
```

Gambar 4.2.10 Test Case 4 (UCS)

```
Kata awal: smile
Kata tujuan: tarot
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 2

Algoritma Greedy Best First Search tidak menemukan jalan dari smile ke tarot.
Runtime: 12 ms
```

Gambar 4.2.11 Test Case 4 (*Greedy Best First Search*)

```
Kata awal: smile
Kata tujuan: tarot
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 3

smile (5) -> smite (6) -> saite (6) -> caite (7) -> carte (7) -> tarte (7) -> tarts (8) -> taros (8) -> tarot (8)
Jumlah node yang dikunjungi: 8
Runtime: 42 ms
```

Gambar 4.2.12 Test Case 4 (A*)

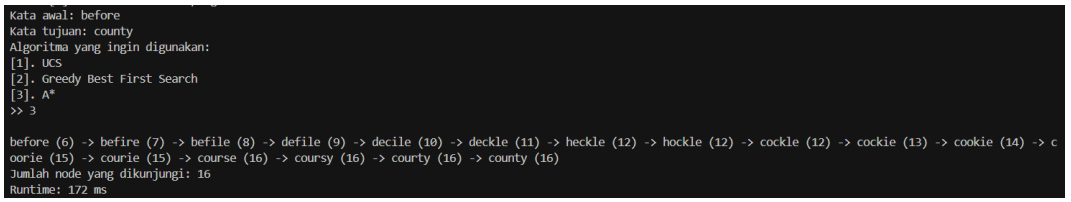
e. Test Case 5

```
Kata awal: before
Kata tujuan: county
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 1

before (0) -> befire (1) -> befile (2) -> defile (3) -> decile (4) -> deckle (5) -> heckle (6) -> hockle (7) -> cockle (8) -> cockie (9) -> cookie (10) -> coorie
(11) -> courie (12) -> course (13) -> coursy (14) -> courty (15) -> county (16)
Jumlah node yang dikunjungi: 16
Runtime: 348 ms
```

Gambar 4.2.13 Test Case 5 (UCS)

```
Kata awal: before
Kata tujuan: county
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 2

Algoritma Greedy Best First Search tidak menemukan jalan dari before ke county.
Runtime: 11 ms
```

Gambar 4.2.14 Test Case 5 (*Greedy Best First Search*)



Gambar 4.2.15 Test Case 5 (A*)

f. Test Case 6

```
Kata awal: boredom
Kata tujuan: control
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 1

Algoritma Uniform Cost Search (UCS) tidak menemukan jalan dari boredom ke control.
Runtime: 1 ms
```

Gambar 4.2.16 Test Case 6 (UCS)

```
Kata awal: boredom
Kata tujuan: control
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 2

Algoritma Greedy Best First Search tidak menemukan jalan dari boredom ke control.
Runtime: 0 ms
```

Gambar 4.2.17 Test Case 6 (*Greedy Best First Search*)

```
Kata awal: boredom
Kata tujuan: control
Algoritma yang ingin digunakan:
[1]. UCS
[2]. Greedy Best First Search
[3]. A*
>> 3

Algoritma A* tidak menemukan jalan dari boredom ke control.
Runtime: 0 ms
```

Gambar 4.2.18 Test Case 6 (A*)

g. Test Case 7 (Test Case 3 dengan tambahan output panjang queue)

```
sand (0) -> sant (1) -> gant (2) -> gait (3) -> grit (4)
Jumlah node yang dikunjungi: 4
Panjang queue: 24462
Runtime: 86 ms
```

Gambar 4.2.19 Test Case 7 (UCS)

```
sand (4) -> said (3) -> caid (3) -> kaid (3) -> laid (3) -> lait (2) -> gait (1) -> grit (0)
Jumlah node yang dikunjungi: 7
Panjang queue: 1
Runtime: 2 ms
```

Gambar 4.2.20 Test Case 7 (*Greedy Best First Search*)

```
sand (4) -> sant (4) -> gant (4) -> gait (4) -> grit (4)
Jumlah node yang dikunjungi: 4
Panjang queue: 1178
Runtime: 4 ms
```

Gambar 4.2.21 Test Case 7 (A*)

4.3 Analisis Hasil Pengujian

Dari hasil pengujian diatas, diperoleh data sebagai berikut:

Test Case	Solusi Optimal			Waktu Eksekusi (ms)		
	UCS	Greedy	A*	UCS	Greedy	A*
1	✓	✓	✓	18	2	1
2	✓	✓	✓	47	1	2
3	✓	✗	✓	77	1	1
4	✓	✗	✓	129	12	42
5	✓	✗	✓	348	11	172
6	Tidak ada solusi			1	0	0

Berdasarkan data diatas, dapat dilihat optimalitas dan waktu eksekusi masing-masing algoritma untuk tiap-tiap test case. Untuk semua test case yang memiliki solusi, algoritma *Uniform Cost Search* (UCS) dan A* memberikan hasil yang optimal, sedangkan algoritma *Greedy Best First Search* hanya dapat memberikan solusi yang optimal untuk kata-kata yang memiliki jumlah huruf yang sedikit dan memberikan solusi yang tidak optimal untuk kata-kata yang lebih panjang, algoritma *greedy* juga gagal memberikan solusi untuk test case 4 dan 5. Sehingga didapatkan bahwa dari segi optimalitas, algoritma UCS dan A* unggul jika dibandingkan dengan algoritma *Greedy Best First Search*.

Jika dilihat dari segi waktu eksekusi program (*runtime*), algoritma UCS memiliki *runtime* yang jauh lebih besar jika dibandingkan dengan algoritma *Greedy Best First Search* dan A*. Hal ini disebabkan karena algoritma UCS tidak mempertimbangkan nilai heuristik untuk menentukan urutan ekspansi simpul, yang menyebabkan algoritma UCS melakukan ekspansi terhadap jumlah simpul yang jauh lebih banyak jika dibandingkan dengan algoritma *Greedy Best First Search* dan A*.

Penggunaan memori algoritma UCS dan A* lebih besar jika dibandingkan dengan algoritma *Greedy Best First Search*, seperti yang dapat dilihat pada gambar 4.2.19, 4.2.20, dan 4.2.21. Karena algoritma *Greedy Best First Search* hanya memilih satu simpul dengan nilai heuristik paling rendah untuk setiap iterasinya, algoritma ini memiliki konsumsi memori

yang rendah. Sedangkan algoritma UCS dan A* menyimpan seluruh simpul yang memenuhi syarat kedalam queue, yang menyebabkan queue untuk menyimpan simpul dengan jumlah yang sangat besar dan mengkonsumsi banyak memori.

BAB V
KESIMPULAN

5.1. Kesimpulan

Algoritma *Uniform Cost Search* (UCS), *Greedy Best First Search*, dan A* semua dapat digunakan untuk mencari solusi dalam permainan *Word Ladder*. Meskipun begitu, berdasarkan data yang didapat dari hasil pengujian, dalam konteks permainan *Word Ladder*, dapat dilihat bahwa algoritma A* merupakan algoritma yang terbaik. Algoritma A* memiliki waktu eksekusi program yang sangat cepat, setara dengan algoritma *greedy*, selalu menghasilkan solusi yang optimal, dan memiliki konsumsi memori yang lebih kecil jika dibandingkan dengan algoritma UCS.

LAMPIRAN

Repository github tugas ini dapat diakses melalui pranala berikut:

https://github.com/DieroA/Tucil3_13522056.git

Poin	Ya	Tidak
1. Program bisa 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		✓

DAFTAR PUSTAKA

https://github.com/dwyl/english-words/tree/master (file dictionary words.txt berasal dari pranala tersebut)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm