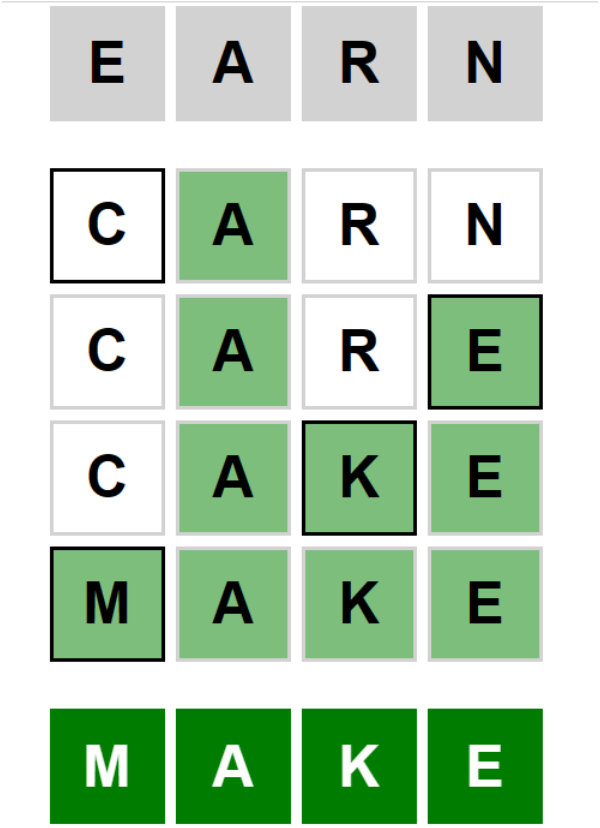
TUGAS KECIL 3
IF2211 Strategi Algoritma
Penyelesaian Permainan *Word Ladder* Menggunakan Algortima UCS, *Greedy Best First Search*, dan A*

DISUSUN OLEH:
13522054 - Benjamin Sihombing

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023

BAB 1
Deskripsi Singkat Program

Pada tugas kecil ini, program akan mencari solusi permainan *word ladder*. Cara Kerja permainan *word ladder* adalah mencari beberapa kata yang menghubungkan *start word* dan *end word* dengan syarat antar kata hanya memiliki satu perbedaan huruf. Berikut ini adalah gambaran permainan *word ladder*,



Program ini akan mencari solusi dengan menggunakan 3 algoritma. Algoritma yang digunakan adalah *Uniform Cost Search* (UCS), *Greedy Best First Search*, dan A*. Algoritma UCS dan A* akan mencari solusi optimal. Solusi optimal adalah solusi yang memiliki jumlah kata paling sedikit dari antara solusi lain.

BAB 2
Analisis dan Implementasi Algoritma

Pada program ini, sebelum dilakukan proses pencarian masukan dari pengguna akan dipastikan terlebih dahulu. Setiap kata harus dipastikan terdapat pada kamus. Setelah kata dipastikan benar, proses pencarian akan dilakukan. Pada program ini, kata akan merepresentasikan sebuah simpul pada graf.

1. **Algoritma *Uniform Cost Search***

   Proses pencarian menggunakan UCS:

   a. Mengecek solusi sudah tercapai (simpul saat ini sama dengan simpul tujuan). Pengecekan solusi (rute) lain tidak perlu dilakukan karena perbedaan huruf antara kata baru dan kata sebelumnya hanya 1.
   b. Jika solusi belum tercapai, simpul baru akan dibangkitkan. Simpul baru yang dibangkitkan harus memiliki perbedaan 1 huruf dengan simpul sebelumnya. Simpul lama tidak lagi menjadi simpul hidup dan simpul baru akan menjadi simpul hidup.
   c. Selanjutnya, proses a akan diulangi dengan simpul yang lain. simpul yang lain tersebut adalah simpul yang memiliki nilai g(n) terkecil (pada umumnya menggunakan g(n) terbesar, namun pada program ini menggunakan g(n) terkecil untuk mempermudah implementasi). Karena *cost* antar kata adalah 1, nilai g(n) yang diambil adalah total kata yang telah dilewati rute untuk mencapai simpul sekarang.
   d. Jika solusi tercapai, pencarian berhenti. Semua kata yang telah dilewati rute untuk mencapai simpul tujuan akan menjadi solusi.

2. **Algoritma *Greedy Best First Search***

   Proses pencarian menggunakan UCS:

   a. Mengecek solusi sudah tercapai (simpul saat ini sama dengan simpul tujuan). Pengecekan solusi (rute) lain tidak perlu dilakukan karena perbedaan huruf antara kata baru dan kata sebelumnya hanya 1.
   b. Jika solusi belum tercapai, simpul baru akan dibangkitkan. Simpul baru yang dibangkitkan harus memiliki perbedaan 1 huruf dengan simpul sebelumnya. Simpul lama tidak lagi menjadi simpul hidup dan simpul baru akan menjadi simpul hidup.
   c. Selanjutnya, proses a akan diulangi dengan simpul lain yang terpilih. simpul yang terpilih tersebut adalah simpul yang memiliki nilai h(n) terkecil (pada umumnya menggunakan h(n) terbesar, namun pada program ini menggunakan h(n) terkecil untuk mempermudah implementasi). Karena *cost* antar kata adalah 1, nilai h(n) yang diambil adalah total perbedaan huruf antara simpul baru (hidup) dan simpul sebelumnya. Simpul baru yang tidak terpilih akan dimatikan (*pruning*).
   d. Jika solusi tercapai, pencarian berhenti. Semua kata yang telah dilewati rute untuk mencapai simpul tujuan akan menjadi solusi.

3. **Algoritma A***

Proses pencarian menggunakan A*:

a. Mengecek solusi sudah tercapai (simpul saat ini sama dengan simpul tujuan). Pengecekan solusi (rute) lain tidak perlu dilakukan karena perbedaan huruf antara kata baru dan kata sebelumnya hanya 1.
b. Jika solusi belum tercapai, simpul baru akan dibangkitkan. Simpul baru yang dibangkitkan harus memiliki perbedaan 1 huruf dengan simpul sebelumnya. Simpul lama tidak lagi menjadi simpul hidup dan simpul baru akan menjadi simpul hidup.
c. Selanjutnya, proses a akan diulangi dengan simpul yang lain. simpul yang lain tersebut adalah simpul yang memiliki nilai g(n) + h(n) terkecil (pada umumnya menggunakan g(n) + h(n) terbesar, namun pada program ini menggunakan g(n) terkecil untuk mempermudah implementasi). Karena *cost* antar kata adalah 1, nilai g(n) yang diambil adalah total kata yang telah dilewati rute untuk mencapai simpul sekarang. Seperti g(n) karena *cost* antar kata adalah 1, nilai h(n) yang diambil adalah total perbedaan huruf antara simpul baru (hidup) dan simpul sebelumnya.
d. Jika solusi tercapai, pencarian berhenti. Semua kata yang telah dilewati rute untuk mencapai simpul tujuan akan menjadi solusi.

4. **Analisis**

Pengambilan g(n) + h(n) sebagai heuristik pada algoritma A* bisa diterima karena g(n) + h(n) memastikan bahwa solusi yang dihasilkan pasti optimal. Nilai g(n) berfungsi untuk memastikan pencarian memilih rute dengan total kata terpendek terlebih dahulu. Sedangkan, nilai g(n) berfungsi untuk memastikan pencarian memilih rute yang simpul ujungnya paling mendekati (mirip) dengan kata tujuan.

Pada kasus ini, algoritma UCS sama dengan algoritma *Breadth First Search*. Hal ini terjadi *cost* antara satu dengan simpul dengan simpul yang lain yang saling terhubung bernilai 1. Hal ini disebabkan oleh aturan *word ladder* yang hanya mengizinkan hanya 1 perbedaan huruf antar kata (simpul). Jadi, setiap simpul baru yang dibangkitkan akan memiliki nilai g(n) yang sama. Hal ini membuat semua simpul baru akan dicek. Perilaku ini akan sama seperti perilaku pada algoritma BFS.

Secara teoritis, algoritma A* akan lebih efisien dibandingkan algoritma UCS. Hal ini disebabkan oleh perilaku UCS yang sama dengan BFS. Seperti yang dijelaskan sebelumnya. Perilaku UCS sama dengan BFS yaitu akan mengecek semua simpul baru. Namun, pada A* karena ada nilai h(n) pencarian akan memprioritaskan rute yang lebih dekat ke solusi.

Secara teoritis, algoritma *Greedy Best First Search* tidak menjamin solusi optimal. Dalam prosesnya, algoritma ini akan memilih 1 rute terbaik sementara dan menghapus rute lain (*pruning*) berdasarkan nilai h(n). Karena hal tersebut, solusi yang dihasilkan belum tentu optimal karena mungkin saja solusi terbaik ada pada rute yang ter-*pruning*.

BAB 3
Source Code Program

Berikut ini adalah kode program yang diimplementasikan menggunakan bahasa Java.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.List;
import java.util.HashMap;
import java.util.Map;

// class perkakas untuk utilitas umum
class Perkakas {
    public static List<String> bacaInput(Scanner scanner) {
        // untuk input kata
        System.out.println(x:"===========INPUT===========");
        System.out.print(s:"Start Word: ");
        String start = scanner.nextLine().toUpperCase();
        System.out.print(s:"End Word: ");
        String end = scanner.nextLine().toUpperCase();
        return Arrays.asList(start, end);
    }

    public static int wordDiff(String s1, String s2) {
        // cari jumlah perbedaan huruf antara 2 kata
        if (s1.length() != s2.length()) {
            return -1;
        }
        int result = 0;
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) != s2.charAt(i)) {
                result++;
            }
        }
        return result;
    }
}
```

```java
// class rute untuk merepresetasikan rute dari node
// dan menyelesaikan permasalahan
class Rute {
    private List<String> nodes;
    private String tujuan;

    public Rute(String target) {
        // ctor
        nodes = new ArrayList<String>();
        tujuan = target;
    }

    public Rute(Rute other) {
        // cctor
        this.nodes = new ArrayList<String>();
        for (String string : other.nodes) {
            this.nodes.add(string);
        }
        this.tujuan = (other.tujuan);
    }

    public List<String> getRute() {
        return nodes;
    }

    public String getLastNode() {
        return nodes.getLast();
    }

    public List<String> searchRute(List<String> dict) {
        // mencari node2 tetangga dari node terakhir di rute
        List<String> result = new ArrayList<String>();
        for (String kata : dict) {
            if (Perkakas.wordDiff(kata, this.getLastNode()) == 1) {
                result.add(kata);
            }
        }
        return result;
    }

    public void addRute(String baru) {
        // menambahkan node baru ke rute
        nodes.add(baru);
    }
```

```java
class Rute {
    public int gn() {
        // mengembalikan panjang rute
        return nodes.size();
    }

    public int hn() {
        // mengembalikan jumlah perbedaan huruf antara node terakhir dengan tujuan
        return Perkakas.wordDiff(nodes.getLast(), tujuan);
    }

    public void printRute() {
        // print rute
        for (String kata : nodes) {
            System.out.println(kata);
        }
    }

    @Override
    public String toString() {
        // print rute untuk println
        return "Rute: \n" + String.join(delimiter:"\n", nodes) + "\n" + "Panjang rute: " + nodes.size();
    }

    // @Override
    public int compareUCS(Rute other) {
        // untuk menentukan prioritas ketika UCS
        if (Integer.compare(this.gn(), other.gn()) == 0) {
            return Integer.compare(this.hn(), other.hn());
        } else {
            return Integer.compare(this.gn(), other.gn());
        }
    }

    public int compareGBFS(Rute other) {
        // untuk menentukan prioritas ketika GBFS
        if (Integer.compare(this.hn(), other.hn()) == 0) {
            return Integer.compare(this.gn(), other.gn());
        } else {
            return Integer.compare(this.hn(), other.hn());
        }
    }

    public int compareAStar(Rute other) {
        // untuk menentukan prioritas ketika AStar
        return Integer.compare(this.gn() + this.hn(), other.gn() + other.hn());
    }
```

```java
class Rute {
    // Algoritma searching
    public static Rute UCS(String start, String target, List<String> dict) {
        // algoritma UCS
        // inisialisasi
        Rute r = new Rute(target);
        r.addRute(start);
        PriorityQueue<Rute> pq = new PriorityQueue<Rute>((r1, r2) -> r1.compareUCS(r2));
        pq.add(r);
        Map<String, Boolean> visited = new HashMap<String, Boolean>();
        boolean stop = false;
        // proses
        while (!stop) {
            Rute temp = pq.poll();
            visited.put(temp.getLastNode(), value:true);
            if (temp.getLastNode().equals(target)) { // solusi tercapai?
                stop = true;
                System.out.println("Jumlah node yang dikunjungi: " + pq.size());
                return temp;
            } else {
                List<String> newNodes = temp.searchRute(dict); // ambil node baru
                for (String kata : newNodes) { // kata adalah node
                    Rute newNode = new Rute(temp); // duplikasi rute
                    if (!(visited.get(kata) != null)) { // node belum pernah divisit
                        newNode.addRute(kata); // tambah node baru
                        pq.add(newNode); // tambah rute baru
                    }
                }
            }
        }
        return null;
    }
```

```java
class Rute {
    public static Rute GBFS(String start, String target, List<String> dict) {
        // algoritma GBFS
        // inisialisasi
        Rute r = new Rute(target);
        r.addRute(start);
        PriorityQueue<Rute> pq = new PriorityQueue<Rute>((r1, r2) -> r1.compareGBFS(r2));
        pq.add(r);
        Map<String, Boolean> visited = new HashMap<String, Boolean>();
        boolean stop = false;
        // proses
        while (!stop) {
            Rute temp = pq.poll();
            if (temp == null) {
                System.out.println(x:"Tidak ada solusi");
                return null;
            }
            visited.put(temp.getLastNode(), value:true);
            while (!pq.isEmpty()) {
                pq.poll();
            }
            if (temp.getLastNode().equals(target)) { // solusi tercapai?
                System.out.println("Jumlah node yang dikunjungi: " + pq.size());
                stop = true;
                return temp;
            } else {
                List<String> newNodes = temp.searchRute(dict); // ambil node baru
                for (String kata : newNodes) { // kata adalah node
                    Rute newNode = new Rute(temp); // duplikasi rute
                    if (!(visited.get(kata) != null)) { // node belum pernah divisit
                        newNode.addRute(kata); // tambah node baru
                        pq.add(newNode); // tambah rute baru
                    }
                }
            }
        }
        return null;
    }
```

```java
class Rute {
    public static Rute AStar(String start, String target, List<String> dict) {
        // algoritma AStar
        // inisialisasi
        Rute r = new Rute(target);
        r.addRute(start);
        PriorityQueue<Rute> pq = new PriorityQueue<Rute>((r1, r2) -> r1.compareAStar(r2));
        pq.add(r);
        Map<String, Boolean> visited = new HashMap<String, Boolean>();
        boolean stop = false;
        // proses
        while (!stop) {
            Rute temp = pq.poll();
            visited.put(temp.getLastNode(), value:true);
            if (temp.getLastNode().equals(target)) { // solusi tercapai?
                System.out.println("Jumlah node yang dikunjungi: " + pq.size());
                stop = true;
                return temp;
            } else {
                List<String> newNodes = temp.searchRute(dict); // ambil node baru
                for (String kata : newNodes) { // kata adalah node
                    Rute newNode = new Rute(temp); // duplikasi rute
                    if (!(visited.get(kata) != null)) { // node belum pernah divisit
                        newNode.addRute(kata); // tambah node baru
                        pq.add(newNode); // tambah rute baru
                    }
                }
            }
        }

        return null;
    }

}
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // proses baca kamus
        List<String> dict = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(fileName:"../src/words.txt"))) {
            String word;
            while ((word = reader.readLine()) != null) {
                dict.add(word.toUpperCase());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Input
        Scanner scanner = new Scanner(System.in);
        List<String> input = Perkakas.bacaInput(scanner);
        String start = input.get(index:0);
        String end = input.get(index:1);

        // validasi input
        while (!dict.contains(start) || !dict.contains(end) || start.length() != end.length()) {
            if (!dict.contains(start) || !dict.contains(end)) {
                System.out.println(x:"Input kata salah! Pastikan setiap kata sudah benar!");
                input = Perkakas.bacaInput(scanner);
                start = input.get(index:0);
                end = input.get(index:1);
            } else {
                System.out.println(x:"Panjang kata tidak sama! Pastikan kedua kata memiliki panjang yang sama!");
                input = Perkakas.bacaInput(scanner);
                start = input.get(index:0);
                end = input.get(index:1);

            }
        }
```

```java
public class Main {
    public static void main(String[] args) {
        // mereduksi kamus
        int sz = start.length();
        dict.stream().filter(s -> s.length() == sz);
        // System.out.println("sz: " + dict.size());

        // penentuan metode
        boolean salah = true;
        String metode = "";
        while (salah) {
            System.out.println(x:"==========================");
            System.out.println(x:"Metode");
            System.out.println(x:"1. Uniform Cost Search");
            System.out.println(x:"2. Greedy Best First Search");
            System.out.println(x:"3. A*");
            System.out.println(x:"==========================");
            System.out.println(x:"Pilih metode: ");
            metode = scanner.nextLine();
            if (!metode.equals(anObject:"1") && !metode.equals(anObject:"2") && !metode.equals(anObject:"3")) {
                System.out.println(x:"Input metode salah!");
            } else {
                salah = false;
            }
        }
        // eksekusi
        System.out.println(x:"==========HASIL==========");
        System.out.println(start + " -> " + end);
        long startTime = System.currentTimeMillis();
        if (metode.equals(anObject:"1")) {
            System.out.println(x:"Uniform Cost Search");
            System.out.println(Rute.UCS(start, end, dict));
        } else if (metode.equals(anObject:"2")) {
            System.out.println(x:"Greedy Best First Search");
            System.out.println(Rute.GBFS(start, end, dict));
        } else {
            System.out.println(x:"A*");
            System.out.println(Rute.AStar(start, end, dict));
        }
        long endTime = System.currentTimeMillis();
        long time = endTime - startTime;
        System.out.println("Waktu eksekusi: " + time + "ms");

        scanner.close();
    }
}
```

BAB 4
Uji Coba dan Analisis

1. Uji Coba

Berikut ini adalah kasus uji coba yang digunakan serta output yang dihasilkan dari program:

- Uji Coba 1

<table>
<tr><td colspan="3">root >> base</td></tr>
<tr><td>UCS</td><td>GBFS</td><td>A*</td></tr>
<tr>
<td>

```
==========INPUT==========
Start Word: root
End Word: base
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
1
==========HASIL==========
ROOT -> BASE
Uniform Cost Search
Jumlah node yang dikunjungi: 1396
Rute:
ROOT
BOOT
BORT
BORE
BARE
BASE
Panjang rute: 6
Waktu eksekusi: 2833ms
```

</td>
<td>

```
==========INPUT==========
Start Word: root
End Word: base
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
2
==========HASIL==========
ROOT -> BASE
Greedy Best First Search
Jumlah node yang dikunjungi: 19
Rute:
ROOT
BOOT
BLOT
BLAT
BLAE
BLUE
BLUB
BLAB
BLAH
BLAM
BEAM
BEAD
BEAK
BEAN
BEAR
BEAT
BEST
BAST
BASE
Panjang rute: 19
Waktu eksekusi: 41ms
```

</td>
<td>

```
==========INPUT==========
Start Word: root
End Word: base
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
3
==========HASIL==========
ROOT -> BASE
A*
Jumlah node yang dikunjungi: 30
Rute:
ROOT
BOOT
BOOS
BOSS
BASS
BASE
Panjang rute: 6
Waktu eksekusi: 45ms
```

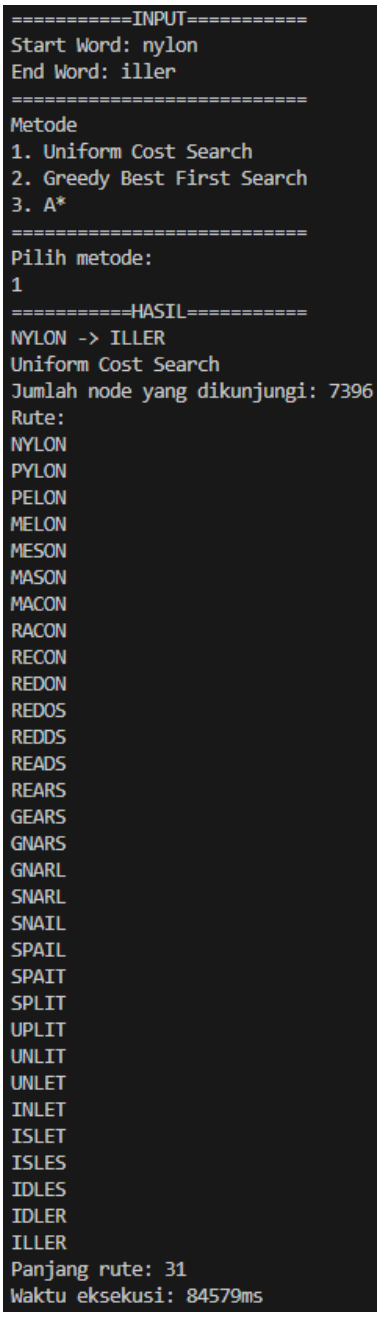
</td>
</tr>
</table>

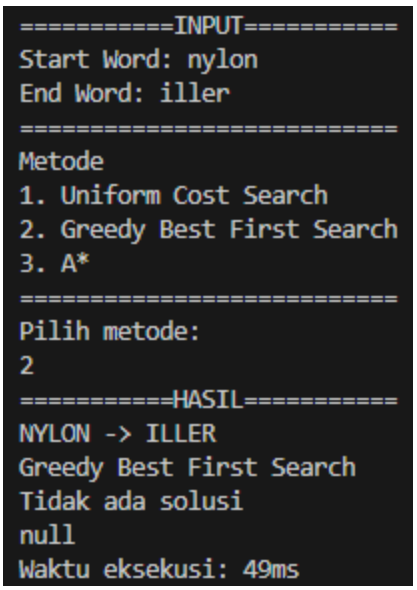
- Uji Coba 2

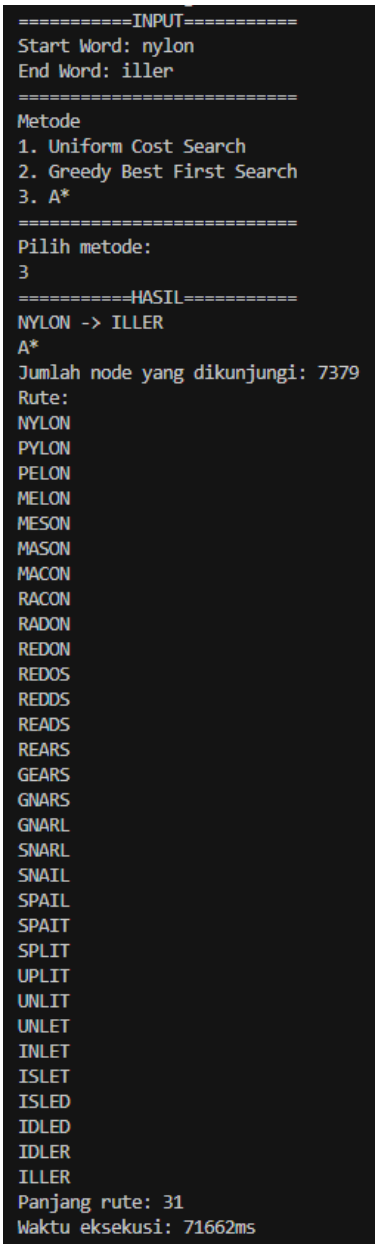
take >> give		
UCS	GBFS	A*
==========INPUT========== Start Word: take End Word: give ========== Metode 1. Uniform Cost Search 2. Greedy Best First Search 3. A* ========== Pilih metode: 1 ==========HASIL========== TAKE -> GIVE Uniform Cost Search Jumlah node yang dikunjungi: 767 Rute: TAKE WAKE WAVE GAVE GIVE Panjang rute: 5 Waktu eksekusi: 1360ms	==========INPUT========== Start Word: take End Word: give ========== Metode 1. Uniform Cost Search 2. Greedy Best First Search 3. A* ========== Pilih metode: 2 ==========HASIL========== TAKE -> GIVE Greedy Best First Search Jumlah node yang dikunjungi: 10 Rute: TAKE TIKE BIKE BICE BIDE AIDE EIDE HIDE HIVE GIVE Panjang rute: 10 Waktu eksekusi: 24ms	==========INPUT========== Start Word: take End Word: give ========== Metode 1. Uniform Cost Search 2. Greedy Best First Search 3. A* ========== Pilih metode: 3 ==========HASIL========== TAKE -> GIVE A* Jumlah node yang dikunjungi: 25 Rute: TAKE RAKE RAVE GAVE GIVE Panjang rute: 5 Waktu eksekusi: 55ms

- Uji Coba 3

nylon >> iller		
UCS	GBFS	A*







- Uji Coba 4

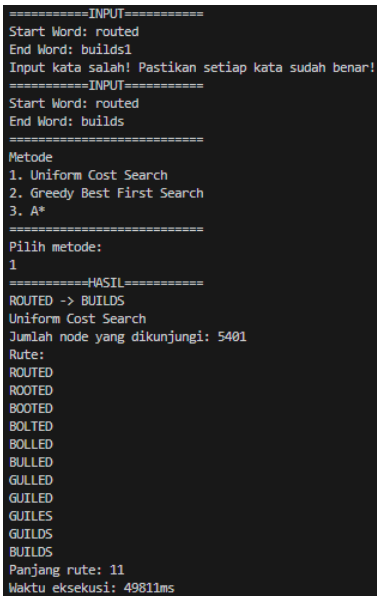
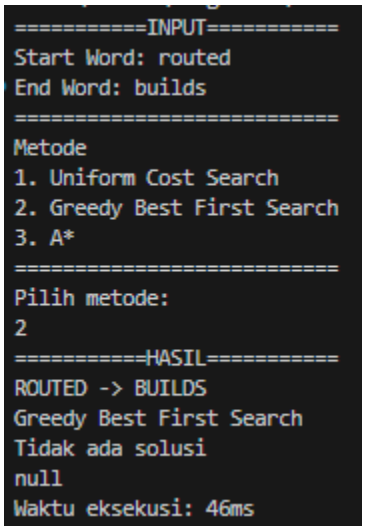
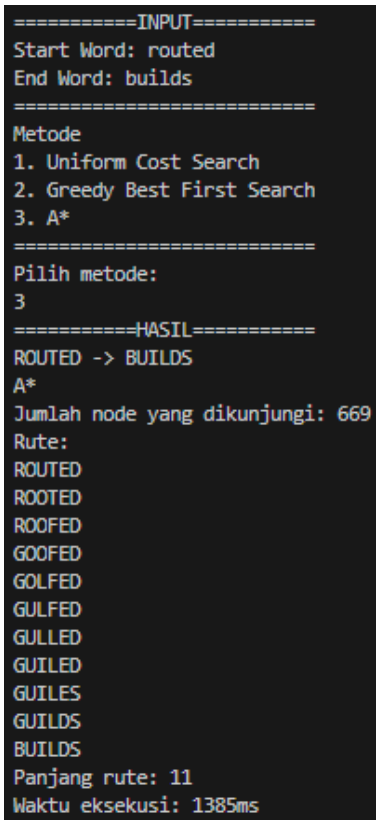
boyish >> painch		
UCS	GBFS	A*

```
==========INPUT==========
Start Word: boyish
End Word: painch
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
1
==========HASIL==========
BOYISH -> PAINCH
Uniform Cost Search
Jumlah node yang dikunjungi: 8403
Rute:
BOYISH
TOYISH
TONISH
MONISH
MOPISH
POPISH
POLISH
PALISH
PARISH
PARIAH
PARIAN
PARTAN
TARTAN
TARTAR
TARTER
CARTER
CARTES
CARSES
CORSES
HORSES
HORSTS
HOISTS
JOISTS
JOINTS
POINTS
POINDS
POUNDS
MOUNDS
MOUNTS
COUNTS
COUNTY
BOUNTY
BOUNCY
JOUNCY
JOUNCE
JAUNCE
LAUNCE
LAUNCH
PAUNCH
PAINCH
Panjang rute: 40
Waktu eksekusi: 92949ms
```

```
==========INPUT==========
Start Word: boyish
End Word: painch
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
2
==========HASIL==========
BOYISH -> PAINCH
Greedy Best First Search
Tidak ada solusi
null
Waktu eksekusi: 48ms
```

```
==========INPUT==========
Start Word: boyish
End Word: painch
=========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
=========================
Pilih metode:
3
==========HASIL==========
BOYISH -> PAINCH
A*
Jumlah node yang dikunjungi: 8378
Rute:
BOYISH
TOYISH
TONISH
MONISH
MOPISH
POPISH
POLISH
PALISH
PARISH
PARIAH
PARIAN
PARTAN
TARTAN
TARTAR
TARTER
CARTER
CARTES
CARSES
CORSES
HORSES
HOISES
HOISTS
JOISTS
JOINTS
POINTS
POINDS
POUNDS
FOUNDS
FOUNTS
COUNTS
COUNTY
BOUNTY
BOUNCY
JOUNCY
JOUNCE
JAUNCE
LAUNCE
LAUNCH
PAUNCH
PAINCH
Panjang rute: 40
Waktu eksekusi: 78826ms
```

- Uji Coba 5

routed >> builds		
UCS	GBFS	A*
		

- Uji Coba 6

atlases >> cabaret		
UCS	GBFS	A*

```
==========HASIL==========
ATLASES -> CABARET
Uniform Cost Search
Jumlah node yang dikunjungi: 7648
Rute:
ATLASES
ANLASES
ANLACES
UNLACES
UNLACED
UNLADED
UNFADED
UNFAKED
UNCAKED
UNCAKES
UNCASES
UNEASES
UREASES
CREASES
CRESSES
TRESSES
TRASSES
BRASSES
BRASHES
BRASHER
BRASIER
BRAKIER
BEAKIER
PEAKIER
PECKIER
PICKIER
DICKIER
DICKIES
HICKIES
HACKIES
HACKLES
HECKLES
DECKLES
DECILES
DEFILES
DEFILED
DEVILED
REVILED
REVELED
RAVELED
RAVENED
HAVENED
HAVERED
WAVERED
WATERED
CATERED
CAPERED
TAPERED
TABERED
TABORED
TABORET
TABARET
CABARET
Panjang rute: 53
Waktu eksekusi: 1683742ms
```

```
==========INPUT==========
Start Word: atlases
End Word: cabaret
==========================
Metode
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
==========================
Pilih metode:
2
==========HASIL==========
ATLASES -> CABARET
Greedy Best First Search
Tidak ada solusi
null
Waktu eksekusi: 40ms
```

```
==========HASIL==========
ATLASES -> CABARET
A*
Jumlah node yang dikunjungi: 7595
Rute:
ATLASES
ANLASES
ANLACES
UNLACES
UNLADES
UNLADED
UNFADED
UNFAKED
UNCAKED
UNCASED
UNCASES
UNEASES
UREASES
CREASES
CRESSES
TRESSES
TRASSES
TRASHES
BRASHES
BRASHER
BRASIER
BRAKIER
BEAKIER
PEAKIER
PECKIER
PICKIER
DICKIER
DICKIES
HICKIES
HACKIES
HACKLES
HECKLES
DECKLES
DECILES
DEFILES
REFILES
REVILES
REVILED
REVELED
RAVELED
RAVENED
HAVENED
HAVERED
WAVERED
WATERED
CATERED
CAPERED
TAPERED
TABERED
TABORED
TABORET
TABARET
CABARET
Panjang rute: 53
Waktu eksekusi: 176528ms
```

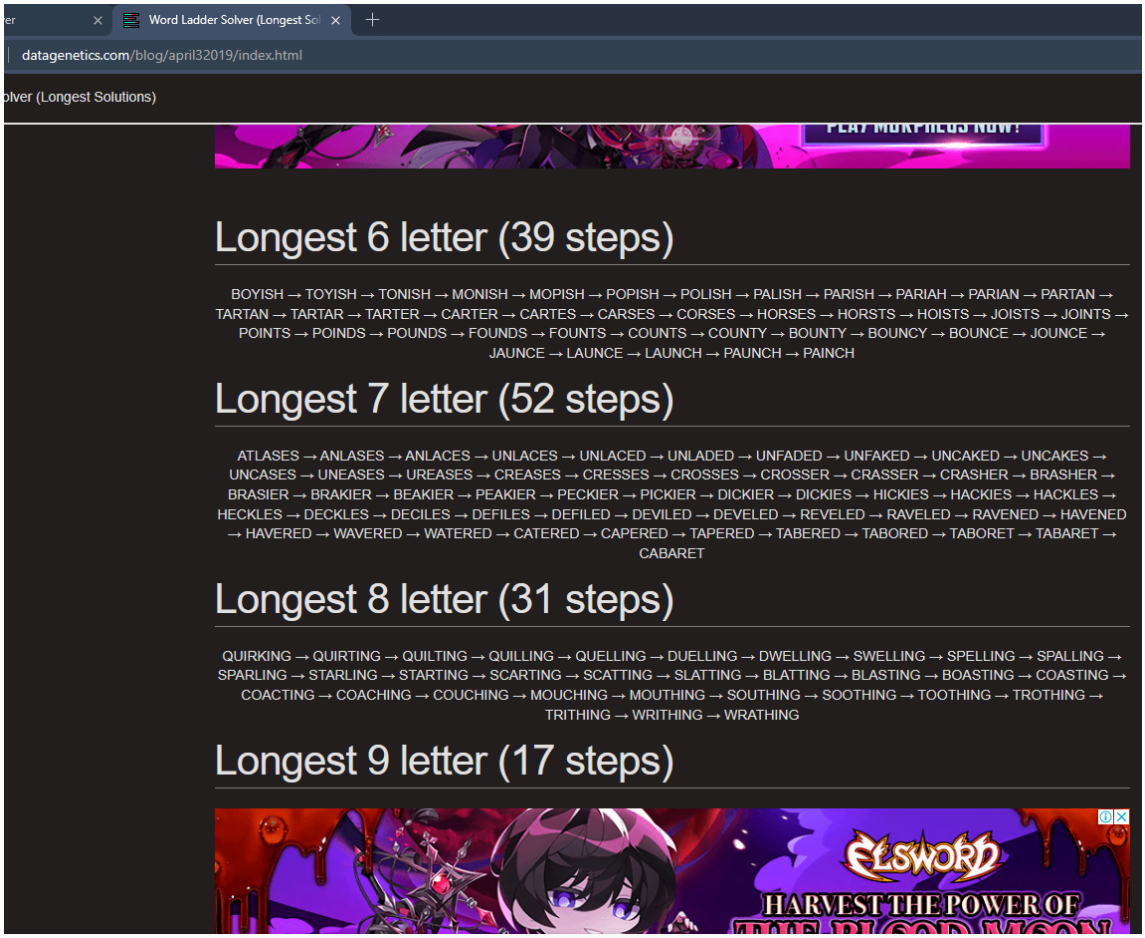
2. Analisis

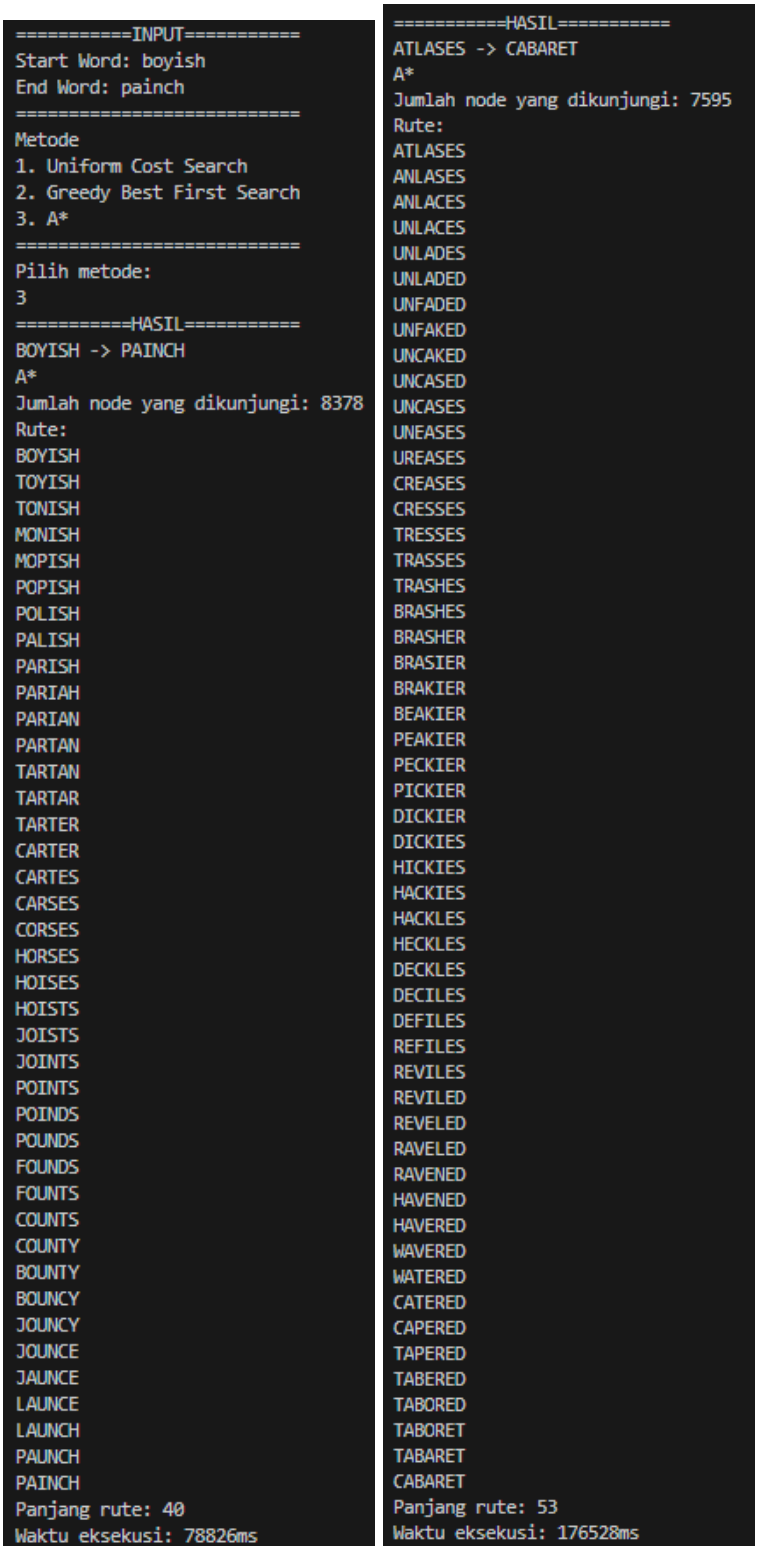
Dari hasil uji coba, dapat disimpulkan beberapa hal:

a. Algoritma UCS dan A* selalu menghasilkan solusi yang optimal (rute terpendek). Hal ini bisa dilihat dari perbandingan solusi dengan GBFS maupun dengan solver di internet. Berikut beberapa solver yang digunakan sebagai pembanding.

- Word Ladder Solver (ceptimus.co.uk)
- Word Ladder Solver (Longest Solutions) (datagenetics.com)

Bukti bisa dilihat dari gambar berikut (rute mungkin tidak sama karena perbedaan pengurutan berdasarkan abjad)





b. Waktu eksekusi terbaik dimiliki oleh *Greedy Best First Search* dan waktu eksekusi terburuk dimiliki oleh UCS. Hal ini terjadi karena GBFS melakukan

pruning. Bahkan karena *pruning*, GBFS tidak mendapatkan solusi. UCS memiliki waktu eksekusi yang terburuk karena algoritma ini melakukan pengecekan terhadap semua simpul yang baru dibangkitkan. Sedangkan A*, melakukan pemilih terlebih dahulu dengan prioritas tertentu. Bukti bisa dilihat pada tabel Uji Coba 1 - 6.

c. Penggunaan memori paling sedikit dimiliki oleh GBFS. Hal ini sama seperti kasus waktu eksekusi. Karena adanya *pruning*, jumlah simpul yang ditelusuri lebih sedikit sehingga memori yang digunakan sedikit. Begitu juga dengan UCS dan A*, UCS melakukan penelusuran simpul yang lebih banyak sehingga menggunakan memori yang paling banyak dibandingkan dengan algoritma lain. Bukti bisa dilihat pada tabel Uji Coba 1 - 6.
d. GBFS bisa menghasilkan solusi yang tidak optimal dan bahkan tidak menghasilkan solusi. Bukti bisa dilihat pada tabel Uji Coba 1 - 6.

BAB 5
Pranala

Link repository dari Tugas Kecil 3 IF 2211 Strategi Algoritma Benjamin Sihombing, https://github.com/Bbennn/Tucil3_13522054.

BAB 6
Lampiran

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 GUI		☑