# LAPORAN TUGAS KECIL 3
# IF2211 STRATEGI ALGORITMA


Disusun Oleh :
M Athaullah Daffa Kusuma M (13522044)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

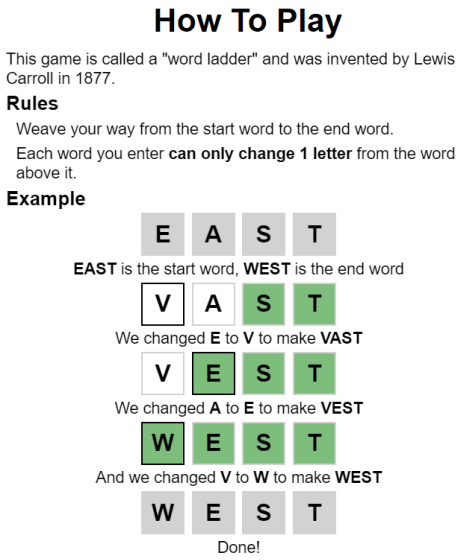
# Daftar Isi

**Daftar Isi** ..... **2**
**Bab 1 : Deskripsi Masalah** ..... **3**
1.1. Word Ladder ..... 3
**Bab 2 : Analisis Landasan Teori** ..... **4**
2.1. Algoritma UCS ..... 4
2.2. Algoritma GBFS ..... 4
2.3. Algoritma A* ..... 5
**Bab 3 : Implementasi** ..... **6**
3.1. NodeGraf.java ..... 6
3.2. Init.java ..... 6
3.3. UCS.Java ..... 8
3.4. GBFS.Java ..... 9
3.5. AStar.Java ..... 11
3.6. Main.Java ..... 12
**Bab 4 : Uji Coba** ..... **14**
4.1. Test Case 1 – Word dengan length 3 ..... 14
4.2. Test Case 2 – Word dengan length 5 ..... 15
4.3. Test Case 3 – Word dengan length 7 ..... 16
4.4. Test Case 4 – Tidak ditemukan solusi ..... 17
4.5. Test Case 5 – Word dengan length 4 ..... 17
4.6. Test Case 6 – Word dengan length 6 ..... 18
4.6. Test Case 7 – Word dengan length 5 ..... 19
**Bab 5 : Analisis Hasil Uji Coba** ..... **21**
**Lampiran** ..... **23**
Link Repository ..... 23
Sheets Poin ..... 23

# Bab 1 : Deskripsi Masalah

## 1.1. Word Ladder

*Word ladder* (juga dikenal sebagai *Doublets, word-links, change-the-word puzzles, paragrams, laddergrams*, atau *word golf*) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. *Word ladder* ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai *start word* dan *end word*. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara *start word* dan *end word*. Banyaknya huruf pada *start word* dan *end word* selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.




**Gambar 1.** Ilustrasi dan Peraturan Permainan *Word Ladder*
(Sumber: https://wordwormdormdork.com/)

# Bab 2 : Analisis Landasan Teori

## 2.1. Algoritma UCS

UCS (Uniform Cost Search) merupakan salah satu algoritma *Route/Path Planning* yang berguna untuk mendapatkan jarak terdekat dari start ke destinasi. Algoritma ini pasti mengembalikan solusi yang optimal, yaitu solusi paling dekat. Algoritma ini pada dasarnya seperti BFS, akan tetapi queue nya dibuat menjadi priority queue dimana yang diprioritaskan adalah g(n) terendah, dimana g(n) merupakan jarak dari simpul start ke simpul yang sedang ditinjau. Implementasi UCS di permainan word ladder adalah dengan membuat setiap word menjadi sebuah simpul, dan sisi menunjukkan bahwa simpul bertetangga, yaitu kedua kata mempunyai panjang yang sama dan hanya memiliki beda 1 karakter. Sementara g(n) nya adalah jarak simpul yang ditinjau ke simpul start (jika simpul yang ditinjau merupakan children dari simpul start, maka jaraknya adalah 1). Di kasus ini, dikarenakan jarak dari simpul start ke simpul lain seragam, yaitu 1,2,3,dst, algoritma UCS dan BFS sama dalam segi urutan node yang dibangkitkan dan path yang dihasilkan. Berikut merupakan langkah-langkah dari UCS:

1. Siapkan sebuah priorityqueue kosong yang diurutkan berdasarkan jarak simpul start ke simpul tersebut,
2. Masukkan simpul start ke priorityqueue,
3. Ambil elemen pertama di priorityqueue kemudian cari simpul tetangganya, jika simpul tetangganya belum pernah dicek, masukkan ke dalam priority queue (urutkan sesuai constraint priority queue),
4. Ulangi langkah 3 sampai priorityqueue kosong atau ditemukan simpul yang sama dengan simpul destinasi,
5. Jika priorityqueue kosong, maka tidak ditemukan jalan apapun yang dapat dilewati dari simpul start ke simpul destinasi.

## 2.2. Algoritma GBFS

GBFS (Greedy Best-First Search) merupakan salah satu algoritma *Route/Path Planning* untuk mencari jalan (*path*) dari simpul start ke simpul destinasi. Algoritma ini belum tentu menghasilkan solusi yang optimal, tetapi algoritma ini mempunyai keunggulan dimana kecepatan eksekusinya yang cukup cepat. Algoritma ini mengembalikan salah satu *path* yang bisa dilewati dari simpul start ke simpul destinasi. Algoritma ini juga mirip seperti UCS, dimana UCS memanfaatkan priority queue yang terurut berdasarkan g(n) (jarak dari simpul start ke simpul yang sedang ditinjau). Akan tetapi, algoritma ini memanfaatkan priority queue yang terurut berdasarkan h(n), yaitu perkiraan jarak dari simpul yang sedang ditinjau ke simpul destinasi. Implementasi GBFS di permainan word ladder adalah sama seperti UCS, namun untuk menghitung h(n) nya adalah seberapa banyak karakter di simpul start yang berbeda dan berindeks sama dengan karakter di simpul destinasi. Berikut merupakan langkah-langkah dari GBFS:

1. Siapkan sebuah priority queue kosong yang diurutkan berdasarkan jarak simpul tersebut ke simpul destinasi
2. Masukkan simpul start ke priority queue
3. Ambil elemen pertama dari priority queue kemudian cari simpul tetangganya, jika simpul tetangganya belum pernah dicek, masukkan ke priority queue
4. Ulangi langkah 3 sampai queue kosong atau ditemukan simpul tetangga yang merupakan simpul destinasi
5. Jika priority queue kosong, maka tidak ditemukan solusi yang memenuhi.

Secara teoritis, algoritma GBFS tidak menjamin menghasilkan solusi optimal di word ladder ini. Karena ia akan selalu memilih h(n) terendah terlebih dahulu untuk diiterasikan terlebih dahulu.

## 2.3. Algoritma A*

A* merupakan algoritma terakhir *Route/Path Planning* untuk mencari jalan (*path*) dari simpul start ke simpul destinasi. Algoritma ini dipastikan selalu menghasilkan solusi yang optimal. Ide dari A* berasal dari UCS yang selalu menghasilkan solusi optimal, tetapi ingin mengurangi memanjangkan path yang sudah terlalu jauh dari destinasi. Algoritma ini mengembalikan path terpendek dari simpul start ke simpul destinasi. Algoritma ini bisa dibilang kombinasi dari UCS dan GBFS, dimana A* memanfaatkan priority queue yang terurut berdasarkan g(n)+h(n) (sama seperti di UCS dan GBFS) terkecil yang diprioritaskan. Implementasi A* dalam Word ladder adalah sama seperti di GBFS dan UCS. Berikut merupakan langkah-langkah dari A* kurang lebih sama seperti UCS dan GBFS:

1. Siapkan sebuah priority queue kosong yang diurutkan berdasarkan g(n)+h(n)
2. Masukkan simpul start ke priority queue
3. Ambil elemen pertama dari priority queue kemudian cari simpul tetangganya, jika simpul tetangganya belum pernah dicek, masukkan ke priority queue
4. Ulangi langkah 3 sampai queue kosong atau ditemukan simpul tetangga yang merupakan simpul destinasi
5. Jika priority queue kosong, maka tidak ditemukan solusi yang memenuhi.

Heuristik (h(n)) pada algoritma ini di kasus word ladder admissible karena h(n) selalu lebih rendah dari h*(n) (sebagai contoh h(n) dari nills ke filla adalah 2 akan tetapi h*(n) nya adalah 3). Secara teoritis, algoritma A* lebih efisien dari algoritma UCS di word ladder ini, karena A* mengurangi simpul yang dikunjungi, ia tidak akan mengunjungi simpul yang sudah terlalu jauh dari simpul start dan simpul destinasi.

# Bab 3 : Implementasi

## 3.1. NodeGraf.java

Source code berikut merupakan source code paling dasar yang mendefinisikan struktur data dari simpul. Simpul menyimpan ID ia sendiri yang otomatis diisi sesuai dengan urutan simpul yang dibuat, dan menyimpan ID dari parentnya. Selain itu, ia menyimpan g(n) sebagai Depth dan h(n) sebagai heuristic. Depth dan Heuristic disimpan untuk mengurangi waktu eksekusi program nantinya. Dibuat method getter primitif dari atribut-atribut class ini.

```
class NodeGraf{
    private Integer Kode;
    private Integer Parent;
    private Integer Depth;
    private Integer Heuristic;
    private String Simpul;
    private static Integer jumlahNode = 0;

    public NodeGraf(Integer Parent, Integer Depth, String Isinya,
String Destinasi){
        this.Parent = Parent; this.Simpul = Isinya; this.Depth =
Depth; this.Heuristic = Init.getDistanceToFinish(Isinya,
Destinasi);
        this.Kode = ++jumlahNode;
    }
    public Integer getParent(){
        return this.Parent;
    }
    public Integer getKode(){
        return this.Kode;
    }
    public Integer getDepth(){
        return this.Depth;
    }
    public Integer getHeuristic(){
        return this.Heuristic;
    }
    public String getSimpul(){
        return this.Simpul;
    }
}
```

## 3.2. Init.java

Source code berikut merupakan source code yang berfungsi untuk menginisiasi progarm (membaca dictionary dari dict.txt) dan menyimpan fungsi2 general yang dapat diakses oleh semua program lainnya. Dictionary disimpan dalam bentuk Set/HashSet untuk menghindari duplikat dan dapat mengurangi waktu eksekusi sedikit dibanding menggunakan ArrayList.

```
import java.util.HashSet;
import java.util.List;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

class Init{
    public static HashSet<String> Dictionary;

    public static void initialize(){
        Init.Dictionary = new HashSet<>();
        try {
            File objFile = new File("../src/dict.txt");
            Scanner reader = new Scanner(objFile);
            while (reader.hasNextLine())
Init.Dictionary.add(reader.nextLine());
            reader.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
    public static boolean hasNeighbour(String arg){
        String tempstr;
        for (int i = 0; i < arg.length(); i++){
            for (char charFit :
"abcdefghijklmnopqrstuvwxyz".toCharArray()){
                tempstr = arg.substring(0, i) + charFit +
arg.substring(i+1);
                if (Init.Dictionary.contains(tempstr) &&
!tempstr.equals(arg)) return true;
            }
        } return false;
    }
    public static Integer getDistanceToFinish(String arg1, String
arg2){
        if (arg1.length() != arg2.length()) return 9999;
        else{
            int dist = arg1.length();
            for (int i = 0; i < arg1.length(); i++) if
(arg1.substring(i, i+1).equals(arg2.substring(i, i+1))) dist--;
            return dist;
        }
    }
```

```
    public static void solveHelper(List<NodeGraf> tempvar){
        NodeGraf jawabannya = tempvar.get(tempvar.size()-1);
        String Answer = ""; boolean found; Integer itr, steps = 0;
        while(jawabannya.getParent() != 0){
            steps++;
            Answer = "->" + jawabannya.getSimpul() + Answer;
            found = false; itr = 0;
            while (!found && itr < tempvar.size()){
                if (tempvar.get(itr).getKode() ==
jawabannya.getParent()){
                    jawabannya = tempvar.get(itr); found = true;
                } else itr++;
            }
        } Answer = jawabannya.getSimpul() + Answer;
        System.out.println("Jumlah steps : " + steps);
        System.out.println(Answer);
    }
}
```

Beberapa method dari class tersebut yaitu hasNeighbour yang menghasilkan true jika sebuah word mempunyai word tetangga minimal 1. Lalu ada method getDistanceToFinish yang menghitung jarak heuristic dari kedua word (menghitung berapa karakter yang mempunyai indeks sama namun berbeda di kedua word). Lalu ada method solveHelper yang berfungsi untuk mengeluarkan output hasil dari algoritma.

## 3.3. UCS.Java

Source code berikut merupakan source code yang mengimplementasikan ide dari UCS di word ladder.

```
import java.util.ArrayList;
import java.util.Set;
import java.util.HashSet;
import java.util.List;

class UCS {
    public static List<NodeGraf> addPriorQueue(List<NodeGraf>
queue, NodeGraf arg2){
        List<NodeGraf> tempSet = new ArrayList<>(queue);
        boolean found = false; int itr = 0;
        while (!found && itr < tempSet.size()){
            if (arg2.getDepth() < tempSet.get(itr).getDepth())
found = true;
            else itr++;
        } tempSet.add(itr, arg2);
        return tempSet;
    }
```

```
    public static void solve(String startWord, String destWord){
        Integer visited = 0;

        NodeGraf leluhur = new NodeGraf(0, 0, startWord, destWord);

        List<NodeGraf> tempvar = new ArrayList<>();
tempvar.add(leluhur);
        List<NodeGraf> queue = new ArrayList<>();
queue.add(leluhur);
        Set<String> visitedDict = new HashSet<>();
visitedDict.add(leluhur.getSimpul());

        while (!queue.isEmpty()){
            visited++;
            NodeGraf temp = queue.get(0); queue.remove(0);

            for (int i = 0; i < temp.getSimpul().length(); i++){
                for (char charFit :
"abcdefghijklmnopqrstuvwxyz".toCharArray()){
                    String tempstr = temp.getSimpul().substring(0,
i) + charFit + temp.getSimpul().substring(i+1);
                    if (Init.Dictionary.contains(tempstr) &&
!visitedDict.contains(tempstr)){ // berarti simpul bertetangga
(cuma beda 1 huruf)
                        visitedDict.add(tempstr);
                        NodeGraf cucu = new
NodeGraf(temp.getKode(), temp.getDepth()+1, tempstr, destWord);
                        tempvar.add(cucu);
                        if (tempstr.equals(destWord)){
                            System.out.println("Visited jml : " +
visited);
                            Init.solveHelper(tempvar); return;
                        } else queue = UCS.addPriorQueue(queue,
cucu);
                    }
                }
            }
        } System.out.println("Tidak ditemukan jalan menuju
Destinasi.");
    }
}
```

## 3.4. GBFS.Java

Source code berikut merupakan source code yang mengimplementasikan ide dari GBFS di word ladder.

```
import java.util.List;
import java.util.Set;
import java.util.ArrayList;
import java.util.HashSet;

class GBFS {
    public static List<NodeGraf> addPriorQueue(List<NodeGraf>
queue, NodeGraf arg2, String destinasi){
        List<NodeGraf> tempSet = new ArrayList<>(queue);
        boolean found = false; int itr = 0;
        while (!found && itr < tempSet.size()){
            if (arg2.getHeuristic() <
tempSet.get(itr).getHeuristic()) found = true;
            else itr++;
        } tempSet.add(itr, arg2);
        return tempSet;
    }

    public static void solve(String startWord, String destWord){
        Integer visited = 0;

        NodeGraf leluhur = new NodeGraf(0, 0, startWord, destWord);

        List<NodeGraf> tempvar = new ArrayList<>();
tempvar.add(leluhur);
        List<NodeGraf> queue = new ArrayList<>();
queue.add(leluhur);
        Set<String> visitedDict = new HashSet<>();
visitedDict.add(leluhur.getSimpul());

        while (!queue.isEmpty()){
            visited++;
            NodeGraf temp = queue.get(0); queue.remove(0);

            for (int i = 0; i < temp.getSimpul().length(); i++){
                for (char charFit :
"abcdefghijklmnopqrstuvwxyz".toCharArray()){
                    String tempstr = temp.getSimpul().substring(0,
i) + charFit + temp.getSimpul().substring(i+1);
                    if (Init.Dictionary.contains(tempstr) &&
!visitedDict.contains(tempstr)){ // berarti simpul bertetangga
(cuma beda 1 huruf)
                        visitedDict.add(tempstr);
                        NodeGraf cucu = new
NodeGraf(temp.getKode(), temp.getDepth()+1, tempstr, destWord);
                        tempvar.add(cucu);
                        if (tempstr.equals(destWord)){
                            System.out.println("Visited jml : " +
visited);
                            Init.solveHelper(tempvar); return;
                        } else queue = GBFS.addPriorQueue(queue,
```

```
cucu, destWord);
                    }
                }
            }
        } System.out.println("Tidak ditemukan jalan menuju
Destinasi.");
    }
}
```

## 3.5. AStar.Java

Source code berikut merupakan source code yang mengimplementasikan ide dari A* di word ladder.

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

class AStar {
    public static List<NodeGraf> addPriorQueue(List<NodeGraf>
queue, NodeGraf arg2, String destinasi){
        List<NodeGraf> tempSet = new ArrayList<>(queue);
        boolean found = false; int itr = 0, fCurr = arg2.getDepth()
+ arg2.getDepth();
        while (!found && itr < tempSet.size()){
            if (fCurr < tempSet.get(itr).getDepth() +
tempSet.get(itr).getHeuristic()) found = true;
            else itr++;
        } tempSet.add(itr, arg2);
        return tempSet;
    }

    public static void solve(String startWord, String destWord){
        Integer visited = 0;

        NodeGraf leluhur = new NodeGraf(0, 0, startWord, destWord);

        List<NodeGraf> tempvar = new ArrayList<>();
tempvar.add(leluhur);
        List<NodeGraf> queue = new ArrayList<>();
queue.add(leluhur);
        Set<String> visitedDict = new HashSet<>();
visitedDict.add(leluhur.getSimpul());

        while (!queue.isEmpty()){
            visited++;
```

```
            NodeGraf temp = queue.get(0); queue.remove(0);

            for (int i = 0; i < temp.getSimpul().length(); i++){
                for (char charFit :
"abcdefghijklmnopqrstuvwxyz".toCharArray()){
                    String tempstr = temp.getSimpul().substring(0,
i) + charFit + temp.getSimpul().substring(i+1);
                    if (Init.Dictionary.contains(tempstr) &&
!visitedDict.contains(tempstr)){ // berarti simpul bertetangga
(cuma beda 1 huruf)
                        visitedDict.add(tempstr);
                        NodeGraf cucu = new
NodeGraf(temp.getKode(), temp.getDepth()+1, tempstr, destWord);
                        tempvar.add(cucu);
                        if (tempstr.equals(destWord)){
                            System.out.println("Visited jml : " +
visited);
                            Init.solveHelper(tempvar); return;
                        } else queue = AStar.addPriorQueue(queue,
cucu, destWord);
                    }
                }
            }
        } System.out.println("Tidak ditemukan jalan menuju
Destinasi.");
    }
}
```

## 3.6. Main.Java

Source code berikut merupakan source code yang mengimplementasikan main interface dari implementasi file lainnya.

```
import java.util.Scanner;

class Main{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Init.initialize();

        System.out.print("Masukkan String Awal : "); String
startWord = sc.nextLine();
        System.out.print("Masukkan String Akhir : "); String
destWord = sc.nextLine();
        while (startWord.length() != destWord.length() ||
startWord.equals(destWord) || !Init.Dictionary.contains(startWord)
|| !Init.Dictionary.contains(destWord) ||
```

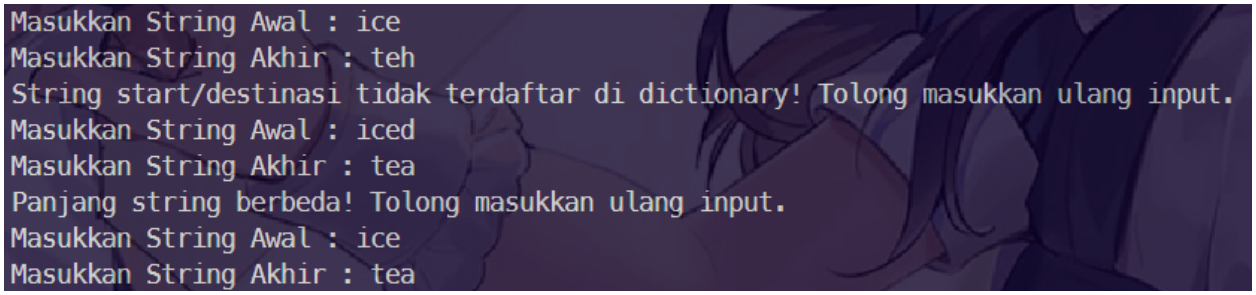
```
!Init.hasNeighbour(destWord) || !Init.hasNeighbour(startWord)){
            if (startWord.equals(destWord))
System.out.println("String start dan destinasi sama! Tolong
masukkan ulang input.");
            else if (!Init.Dictionary.contains(startWord) ||
!Init.Dictionary.contains(destWord)) System.out.println("String
start/destinasi tidak terdaftar di dictionary! Tolong masukkan
ulang input.");
            else if (startWord.length() != destWord.length())
System.out.println("Panjang string berbeda! Tolong masukkan ulang
input.");
            else System.out.println("String start/destinasi tidak
mempunyai tetangga! Tolong masukkan ulang input.");
            System.out.print("Masukkan String Awal : "); startWord
= sc.nextLine();
            System.out.print("Masukkan String Akhir : "); destWord
= sc.nextLine();
        }

        System.out.println("Masukkan Algoritma yang hendak
dipilih.");
        System.out.println("1. UCS\n2. GBFS\n3. A*");
        System.out.print("Input pilihan (1/2/3) : "); Integer algo
= sc.nextInt();
        while (algo <= 0 && algo >= 4) {
            System.out.println("Input tidak valid! Tolong masukkan
ulang input.");
            System.out.print("Input pilihan (1/2/3) : "); algo =
sc.nextInt();
        }

        long startTime = System.nanoTime();
        if (algo == 1) UCS.solve(startWord, destWord);
        else if (algo == 2) GBFS.solve(startWord, destWord);
        else AStar.solve(startWord, destWord);
        System.out.println("Waktu dibutuhkan : " +
(System.nanoTime()-startTime)/1000000 + "ms");

        sc.close();
    }
}
```

Nantinya program akan meminta 2 argument dari pengguna, yaitu String start dan String destinasi. Lalu program akan meng-*loop* langkah input jika pengguna memasukkan start dan destinasi yang sama, panjang String start dan panjang String berbeda, String start atau destinasi tidak terdaftar di Dictionary, atau String start atau destinasi tidak mempunyai tetangga. Lalu pengguna memasukkan algoritma yang dipilih, lalu program memanggil fungsi solve dari algoritma yang dipilih. Setelah selesai dan mendapat solusinya, program juga memberi lama waktu yang diperlukan Algoritma tersebut untuk mendapat solusi.

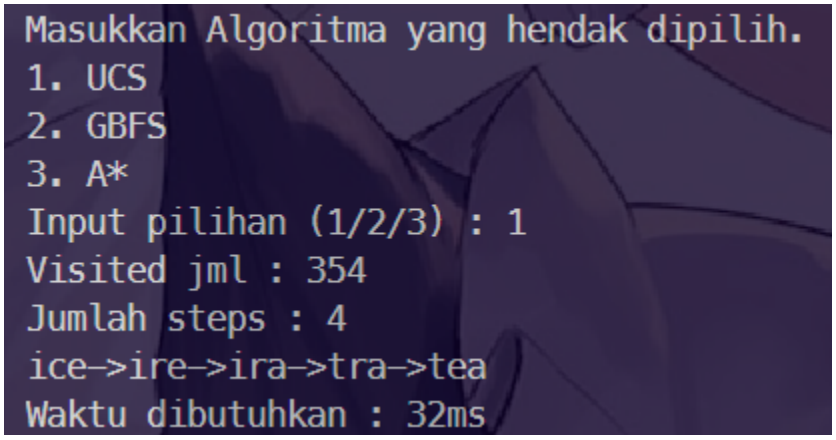
# Bab 4 : Uji Coba

## 4.1. Test Case 1 – Word dengan length 3

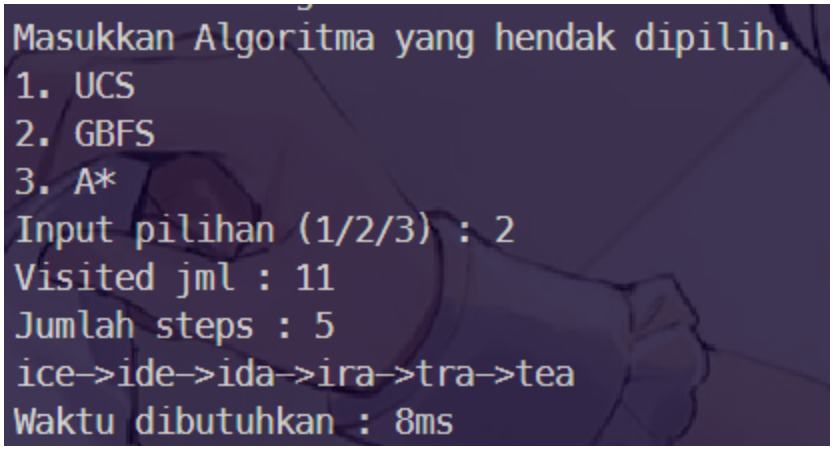
Dengan input berupa (serta tambahan output error untuk kasus word tidak ada di dictionary dan length word tidak sama)



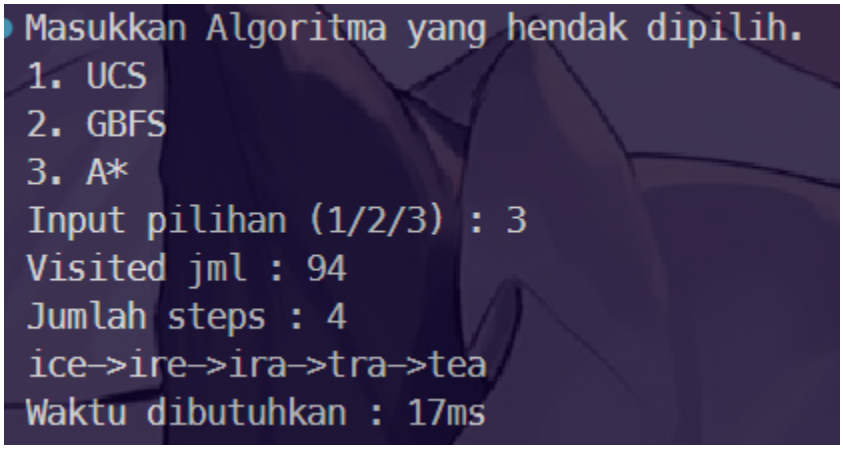

Didapat solusi algoritma UCS berupa




Juga solusi algoritma GBFS berupa

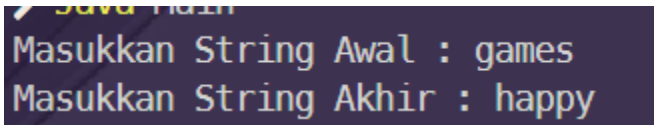



Dan solusi algoritma A* berupa

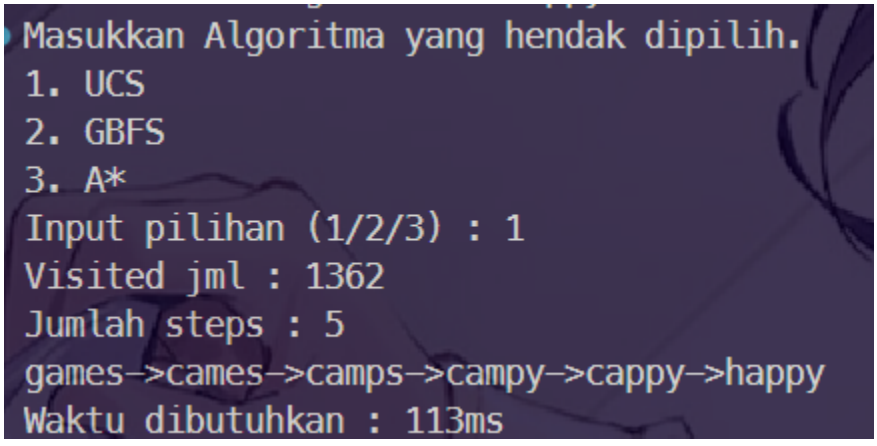



## 4.2. Test Case 2 – Word dengan length 5

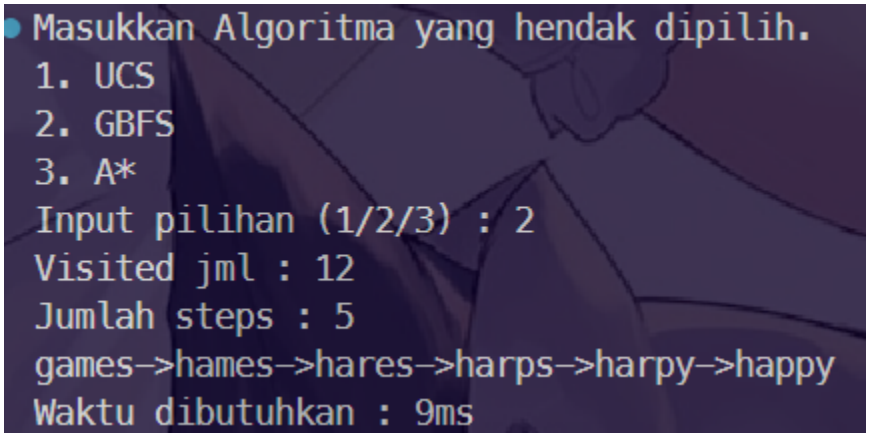
Dengan input berupa



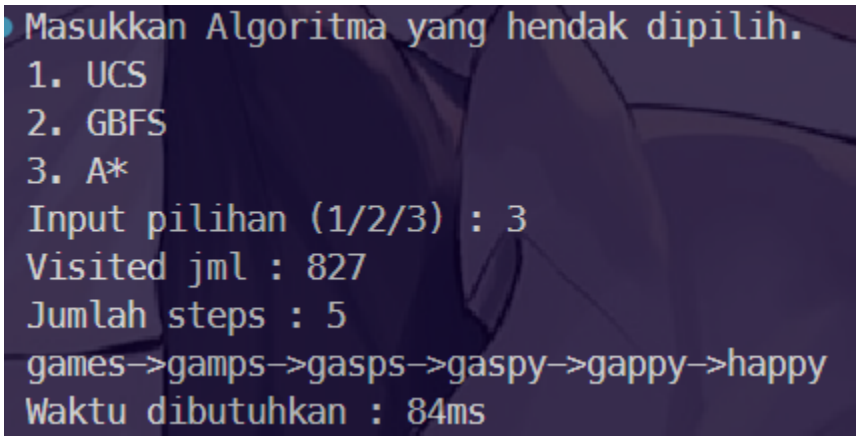

Didapat solusi algoritma UCS sebagai berikut



Juga solusi algoritma GBFS sebagai berikut

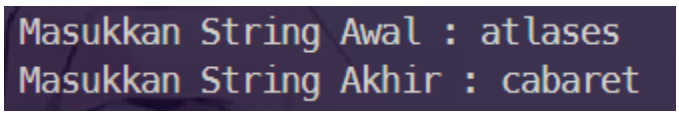



Dan solusi algoritma A* sebagai berikut




## 4.3. Test Case 3 – Word dengan length 7

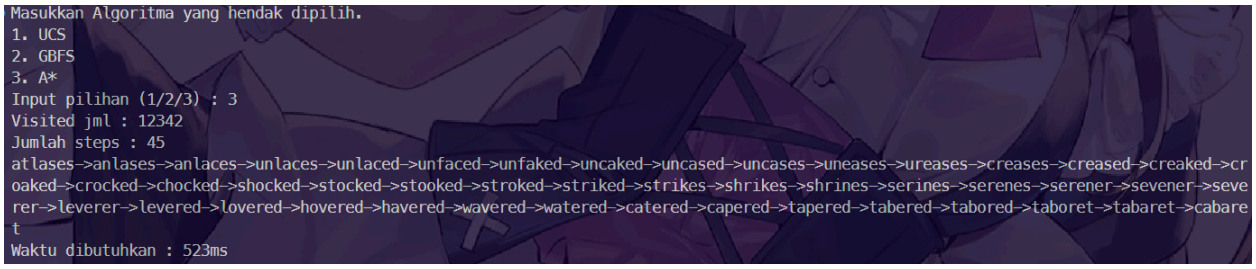
Dengan input berupa




Didapat solusi algoritma UCS sebagai berikut

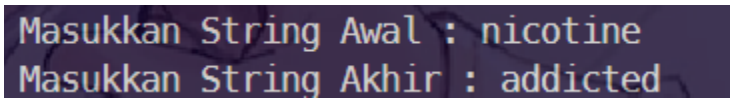

Juga solusi algoritma GBFS sebagai berikut


Dan solusi algoritma A* sebagai berikut

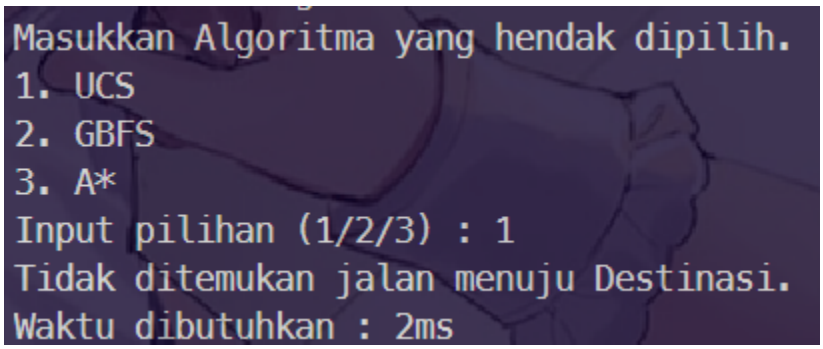



## 4.4. Test Case 4 – Tidak ditemukan solusi

Dengan input berupa

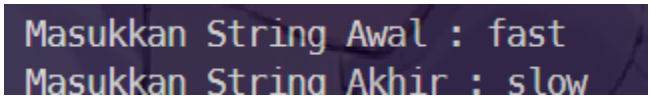



Didapat solusi algoritma UCS sebagai berikut (UCS merupakan algoritma paling efektif, jiak UCS tidak didapat solusi, maka yang lainnya juga tidak mendapat solusi)

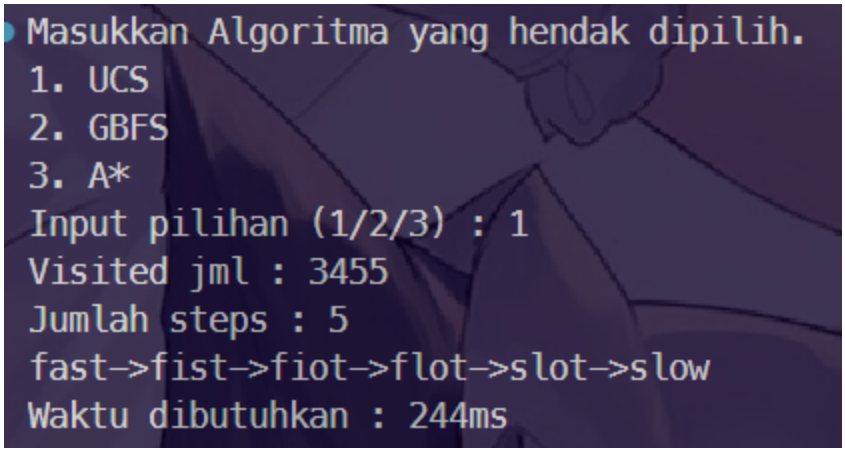



## 4.5. Test Case 5 – Word dengan length 4

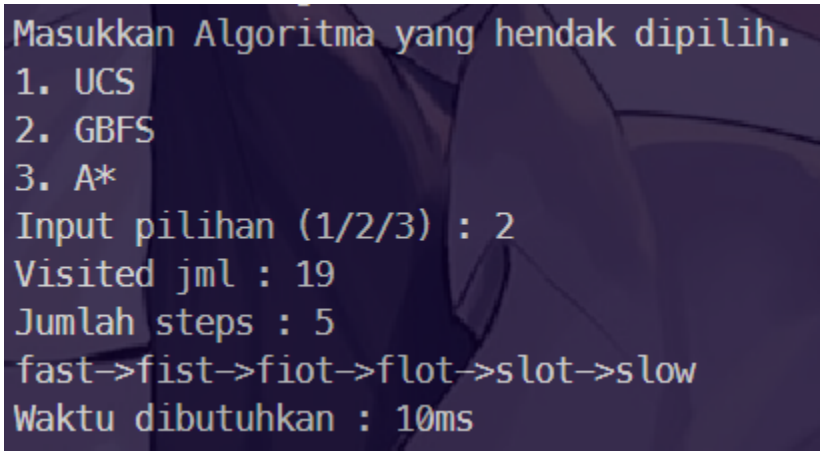
Dengan input berupa



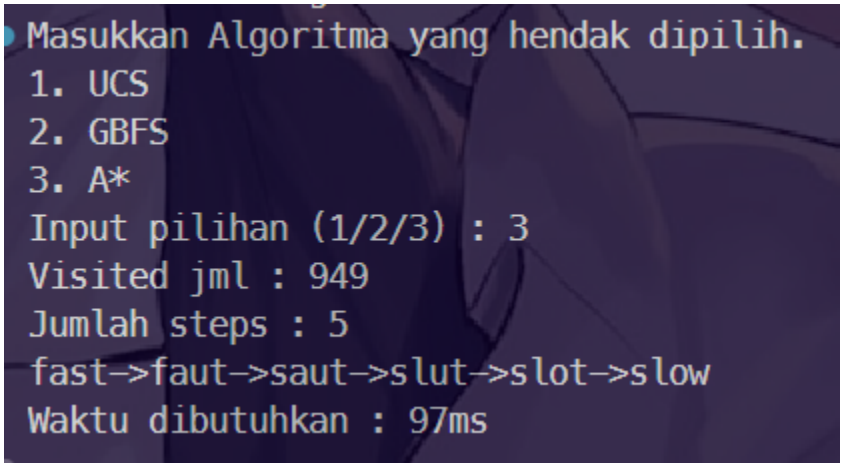

Didapat solusi algoritma UCS sebagai berikut




Juga solusi algoritma GBFS sebagai berikut




Dan solusi algoritma A* sebagai berikut




## 4.6. Test Case 6 – Word dengan length 6

Dengan input berupa

```
Masukkan String Awal : username
Masukkan String Akhir : password
String start/destinasi tidak terdaftar di dictionary! Tolong masukkan ulang input.
Masukkan String Awal : password
Masukkan String Akhir : database
String start/destinasi tidak mempunyai tetangga! Tolong masukkan ulang input.
Masukkan String Awal : dating
Masukkan String Akhir : married
Panjang string berbeda! Tolong masukkan ulang input.
Masukkan String Awal : dating
Masukkan String Akhir : breaks
```

Didapat solusi algoritma UCS sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 1
Visited jml : 14589
Jumlah steps : 13
dating->sating->sabing->sabins->sabirs->sabers->sayers->shyers->sheers->cheers->cheeks->creeks->breeks->breaks
Waktu dibutuhkan : 741ms
```

Juga solusi algoritma GBFS sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 2
Visited jml : 163
Jumlah steps : 30
dating->bating->baring->barong->barons->borons->bosons->bosoms->besoms->besots->besets->berets->burets->bursts->burses->busses->bosses->b
ouses->boules->brules->brumes->bromes->broses->brises->brisks->bricks->brocks->crocks->croaks->creaks->breaks
Waktu dibutuhkan : 26ms
```

Dan solusi algoritma A* sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 3
Visited jml : 12211
Jumlah steps : 13
dating->sating->satins->sabins->sabirs->sabers->sayers->shyers->sheers->cheers->cheeks->creeks->breeks->breaks
Waktu dibutuhkan : 651ms
```

## 4.6. Test Case 7 – Word dengan length 5

Dengan input berupa

```
Masukkan String Awal : human
Masukkan String Akhir : alien
```

Didapat solusi algoritma UCS sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 1
Visited jml : 9211
Jumlah steps : 10
human->ruman->rumen->lumen->limen->limer->aimer->aider->alder->alden->alien
Waktu dibutuhkan : 527ms
```

Juga solusi algoritma GBFS sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 2
Visited jml : 163
Jumlah steps : 19
human->ruman->rumen->lumen->limen->liven->leven->leden->laden->faden->frden->freen->green->gleen->glean->elean->ellan->allan->allen->alie
n
Waktu dibutuhkan : 25ms
```

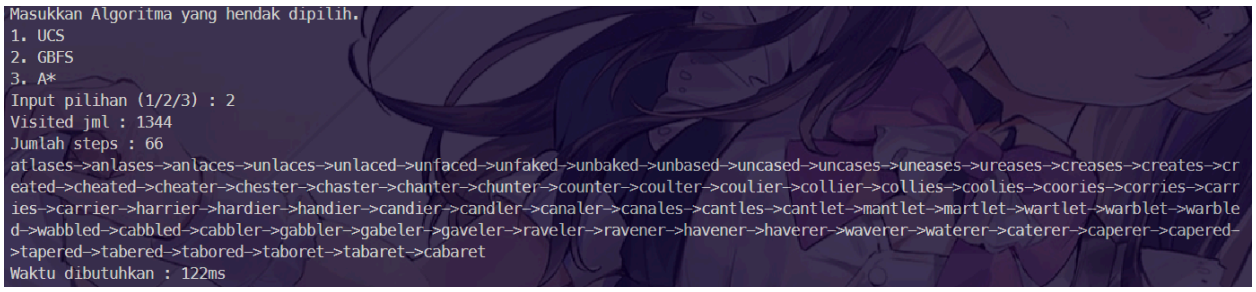
Dan solusi algoritma A* sebagai berikut

```
Masukkan Algoritma yang hendak dipilih.
1. UCS
2. GBFS
3. A*
Input pilihan (1/2/3) : 3
Visited jml : 8477
Jumlah steps : 10
human->ruman->rumen->lumen->limen->limer->aimer->aider->alder->alden->alien
Waktu dibutuhkan : 484ms
```

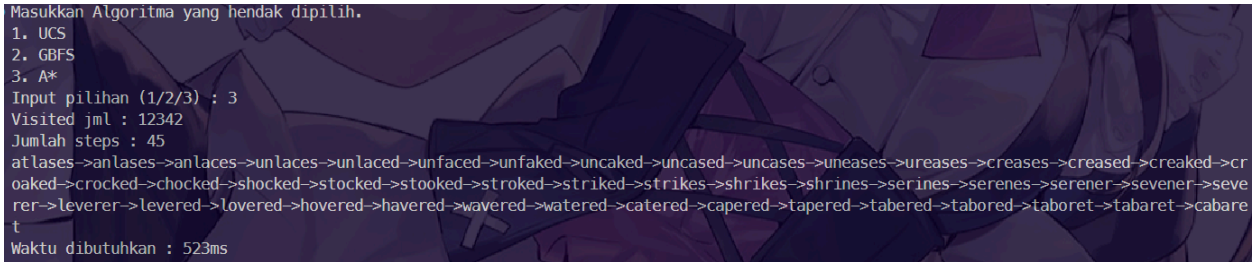
# Bab 5 : Analisis Hasil Uji Coba

Dari 7 percobaan dari masing-masing algoritma, didapat bahwa masing-masing algoritma mempunyai kelebihannya masing-masing. Algoritma UCS pasti selalu mendapat solusi yang optimal (bisa dibilang UCS merupakan versi brute force dari route planning), Algoritma GBFS pasti selalu mendapat waktu eksekusi terendah dengan solusi yang belum tentu optimal, dan Algoritma A* mengambil solusi optimal dari UCS dengan waktu eksekusi yang lebih cepat dari UCS. Berikut merupakan bukti pendukung dengan menggunakan test case 3.


Didapat solusi paling optimal dari Algoritma UCS tetapi lambat




Didapat solusi tercepat dengan Algoritma UCS tetapi tidak optimal




Didapat solusi optimal yang lebih cepat dari Algoritma UCS

Selain waktu eksekusi, memori yang dibutuhkan oleh Algoritma GBFS juga sangat unggul dibanding keduanya, hal ini ditandai dari node yang di-*visit* oleh GBFS jauh lebih rendah dari UCS dan A*. Selain itu Algoritma A* juga unggul di memori jika dibandingkan dengan UCS, hal ini juga sesuai dengan tujuan A* yaitu mengulangi UCS namun tidak mengiterasi lebih lanjut simpul yang sudah terlalu jauh dari simpul start dan simpul destinasi.

# Lampiran

## Link Repository

https://github.com/NameLessAth/Tucil3_13522044

## Sheets Poin

| 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 | | ✓ |