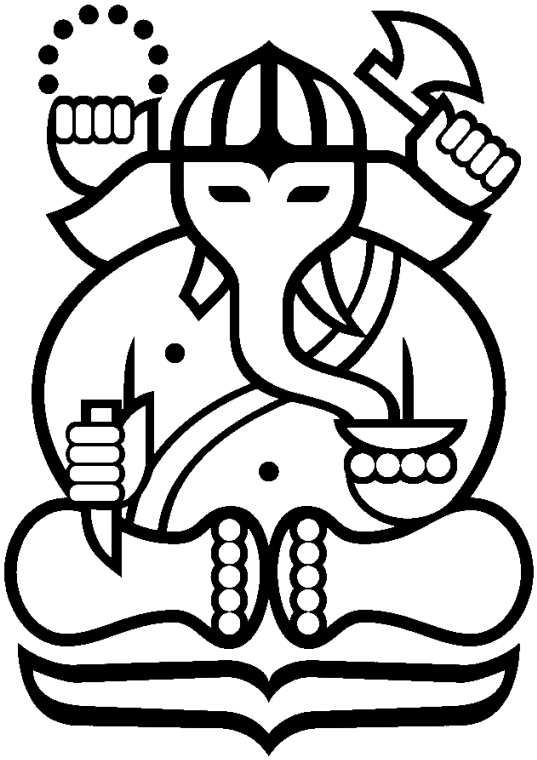
Laporan Tugas Kecil 3

IF2211 Strategi Algoritma

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

Semester II tahun 2023/2024



Disusun Oleh:

Rayendra Althaf Taraka Noor (13522107)

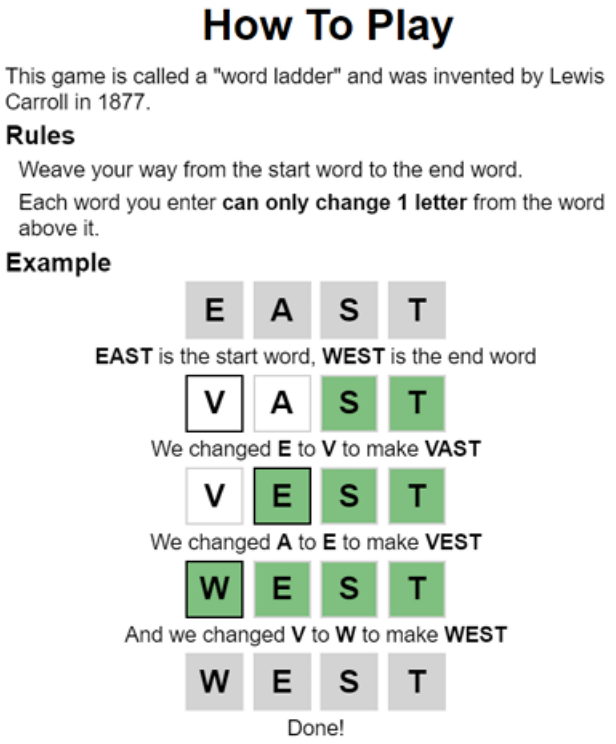
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I
DESKRIPSI TUGAS

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/)

Permainannya cukup sederhana bukan? Jika belum paham dengan peraturan permainannya, cobalah untuk memainkan permainannya pada link sumber di atas. Jika sudah paham dengan permainannya, sekarang adalah waktunya kalian untuk membuat sebuah solver permainan tersebut dengan harapan kita dapat menemukan solusi paling optimal untuk menyelesaikan permainan Word Ladder ini.

BAB II

LANDASAN TEORI

2.1. Algoritma Uniform Cost Search (UCS)

Uniform Cost Search (UCS) adalah algoritma *uninformed search* yang digunakan untuk menemukan rute dengan jalur terpendek dalam sebuah graf. Tujuan dari algoritma ini adalah untuk mencari jalur dengan biaya terendah dari titik awal ke titik tujuan.

Prinsip utama dari UCS adalah mengunjungi simpul yang memiliki biaya terendah terlebih dahulu pada setiap langkahnya. Secara garis besar, Algoritma ini mirip dengan algoritma *Breadth-First Search* (BFS). Yang membedakannya dengan BFS adalah pada algoritma ini graf tempat UCS digunakan dapat berupa graf berbobot. Pada akhir algoritma ini, dipastikan bahwa akan didapatkan suatu solusi yang memiliki biaya terendah (solusi optimal).Untuk lebih detailnya berikut adalah langkah-langkah utama dari algoritma UCS :

1. Masukkan simpul awal ke dalam *priority queue* dengan biaya nol.
2. Ambil simpul dengan biaya terendah dari *priority queue*.
3. Jika simpul tersebut adalah titik tujuan, selesaikan pencarian.
4. Jika bukan, tambahkan semua tetangga dari simpul tersebut ke dalam *priority queue* dengan memperhitungkan biaya dari titik awal ke simpul tersebut.
5. Ulangi langkah-langkah di atas (selain langkah pertama) hingga menemukan solusi atau *priority queue* kosong.

Kelebihan UCS adalah solusi yang didapatkannya akan selalu optimal karena mengeksplorasi semua jalur dengan biaya terendah terlebih dahulu. Namun, dikarenakan eksplosrasi tersebut, UCS memiliki waktu penyelesaian yang relatif lebih lambat dibandingkan dengan algoritma pencarian rute lainnya.

2.2. Algoritma Greedy-First Search (GBFS)

Greedy Best-First Search (GBFS) adalah algoritma *informed search* untuk menemukan rute dengan jalur terpendek dari suatu simpul awal ke suatu simpul tujuan dalam sebuah graf. Prinsip utama dalam algoritma ini ialah memilih simpul yang paling dekat dengan simpul tujuan berdasarkan suatu perhitungan heuristik yang dipilih. Lebih detilnya, berikut merupakan langkah-langkah utama Algoritma GBFS:

1. Masukkan ke dalam sebuah antrian.
2. Selama antrian tidak kosong, pilih simpul yang menurut perkiraan heuristik paling dekat dengan simpul tujuan.
3. Periksa apakah simpul yang dipilih adalah simpul tujuan. Jika iya, selesaikan pencarian.
4. Jika bukan, periksa semua tetangga dari simpul yang dipilih dan tambahkan mereka ke dalam antrian.
5. Ulangi langkah-langkah di atas (selain langkah pertama) hingga menemukan simpul tujuan atau antrian kosong.

Karena pendekatan yang digunakannya, GBFS memiliki keunggulan dalam kecepatan yang disebabkan oleh pemilihannya yang hanya mempertimbangkan heuristik. Walaupun begitu, Algoritma ini seringkali memberikan solusi yang tidak optimal dikarenakan heuristik yang kurang tepat atau mungkin peta pencarian itu sendiri yang tidak cocok untuk diperikirakan dengan suatu heuristik. Pada kasus tertentu, solusi bahkan tidak ditemukan karena terjebak pada *local minima*.

2.3. Algoritma A*

A* (dibaca A-star) adalah algoritma *informed search* lain yang digunakan untuk menemukan rute dengan jalur terpendek dari suatu simpul awal ke suatu simpul tujuan dalam sebuah graf. Prinsip utama A* adalah mengeksplorasi jalur dengan biaya terendah yang merupakan kombinasi dari biaya sejauh ini dari titik awal ke suatu simpul dan estimasi biaya yang tersisa

(yang diperkirakan oleh fungsi heuristik) dari simpul tersebut ke titik tujuan. Lebih jelasnya berikut merupakan langkah-langkah utama dari algoritma A*:

1. Masukkan simpul awal ke dalam *priority queue*.
2. Selama *priority queue* tidak kosong, pilih simpul dengan biaya terendah (biaya dari titik awal ditambah estimasi biaya ke simpul tujuan) untuk mencapai simpul tujuan.
3. Jika simpul tersebut merupakan simpul tujuan, selesaikan pencarian.
4. Jika bukan, periksa semua tetangga dari simpul yang dipilih dan tambahkan mereka ke dalam *priority queue* dengan memberikan setiap simpul tersebut sebuat biaya.
5. Ulangi langkah-langkah di atas (selain langkah pertama) hingga menemukan simpul tujuan atau *priority queue* kosong.

A* merupakan pencarian yang efisien dengan memanfaatkan heuristik sambil tetap berusaha menjaga optimalitas solusi. Algoritma ini bisa dikatakan sebagai algoritma yang berusaha menyeimbangkan antara pendapatan solusi yang cukup optimal dengan waktu pencarian yang relatif singkat.

BAB III

ANALISIS DAN IMPLEMENTASI

3.1 Analisis Solusi Permainan Word Ladder dengan Algoritma Pencarian Graf

Pada tugas kecil ini, word ladder solver akan dibuat menggunakan tiga algoritma yang sudah dijelaskan pada bab sebelumnya, yakni UCS, GBFS, dan A*. Simpul yang berhubungan pada masalah ini adalah simpul yang mewakili kata yang memiliki satu perbedaan huruf. Sebagai contoh perbedaan huruf ketiga dari kata *brake* dan *broke* atau perbedaan huruf kedua dari kata *raw* dan *row.* Karena langkah untuk melangkah dari sebuah node simpul ke simpul lain yang bertetanggan memiliki ongkos perjalanan yang sama, semua sisi yang menghubungkan dua simpul diberi nilai yang seragam yakni 1.

Pada algoritma UCS yang dibuat pada permasalahan ini, g(n) yang memiliki arti jarak suatu simpul dari titik awal dan akan berperan sebagai f(n) (fungsi evaluasi) dari fungsi ini akan selalu bernilai satu lebih tinggi dari simpul yang mengunjunginya. Fungsi evaluasi tersebut kemudian akan menjadi nilai yang menentukan pengurutan pada *priority queue*. Dikarenakan hal tersebut algoritma ini akan memiliki urutan pengunjungan yang sama persis dengan BFS.

Di sisi lain, untuk implementasi algoritma GBFS, f(n) yang memiliki arti fungsi evaluasi akan ditentukan dari sebuah fungsi heuristik, h(n). Perkiraan jarak suatu simpul dari simpul lain, h(n), pada implementasi ini akan dihitung berdasarkan jumlah huruf yang berbeda berada di urutan yang sama. Sebagai contoh, kata *chair* dan *chore* memiliki jarak 3 karena terdapat perbedaan pada huruf ketiga, keempat, dan kelima. Pada algoritma ini juga akan digunakan *priority queue* dengan nilai penentu fungsi heuristik tersebut. Selain itu, agar sesuai dengan GBFS, pada setiap pengambilan elemen di *queue, queue* akan dikosongkan sehingga setiap pengambilannya hanya dilakukan sekali untuk setiap kedalaman. Hal ini

menyebabkan solusi yang ditemukannya tidak selalu optimal, bahkan tidak jarang juga algortima ini tidak menemukan solusi karena terjebak pada *local minima.*

Terakhir, Algoritma A*. Fungsi evaluasinya dihitung dengan menggabungkan jarak dari simpul awal dan perkiraan langkah menuju simpul seperti menggabungkan dua algoritma sebelumnya. Penggabungan dua fungsi tersebut juga bisa dibuktikan sebagai fungsi perkiraan yang akan selalu bernilai sama atau lebih kecil dibandingkan dengan jarak sesungguhnya *(admissible)* karena perubahan maksimum dalam satu langkah adalah dengan merubah satu huruf dan mungkin tidak bisa langsung ke huruf tujuan. Dengan penggabungan ini pemilihan simpul akan menjadi lebih baik sehingga bisa didapatkan dengan pengunjungan simpul yang lebih sedikit sehingga waktu eksekusi secara teori akan lebih cepat jika dibandingkan dengan penyelesaian algoritma UCS.

3.2 Implementasi pada Program dengan Bahasa Java

Pada program ini algoritma utamanya dibagi menjadi 5 file java, 1 file java pemuat dictionary yaitu EnglishWorlds.java, tiga file java penghitung nilai f(n) yakni UCS.java, GBFS.java, dan Astar.java, dan 1 file java penjelajah graf yaitu WordLadderSolver.java. Selain itu, ada juga file java sebagai kelas dari node yang bernama SearchNode.java dan file java berisi program utama yaitu Program.java. Berikut isi dari kelas-kelas tersebut:

```java
// EnglishWorlds.java

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

class EnglishWords{
    private HashSet<String> data;


    EnglishWords(String filePath){
        HashSet<String> result = new HashSet<>();
```

```
        try(BufferedReader r = new BufferedReader(new
FileReader(filePath))){
            String word = r.readLine();
            while(word != null){
                result.add(word);
                word = r.readLine();
            }
        }
        catch(IOException e){
            e.getStackTrace();
        }
        data = new HashSet<>(result);
    }

    public boolean isWordValid(String s){
        return data.contains(s);
    }

    public ArrayList<String> getNearby(String s){
        ArrayList<String> nearby = new ArrayList<>();
        for(int i = 0; i < s.length(); i++){
            char[] cCopy = s.toCharArray();
            for(char c = 'a'; c <= 'z'; c++){
                if(c == s.charAt(i))
                    continue;
                cCopy[i] = c;
                String sCopy = new String(cCopy);
                if(!isWordValid(sCopy))
                    continue;
                nearby.add(sCopy);
            }
        }
        return nearby;
    }

}
```

```
// UCS.java

public class UCS {
    public static int getDistance(int currentPathLength){
        return currentPathLength + 1;
    }
}
```

```java
// GBFS.java

public class GBFS {
    public static int getDistance(String currentString, String
targetString){
        int distanceCounter = 0;
        for(int i = 0; i < currentString.length(); i++){
            if(currentString.charAt(i) != targetString.charAt(i))
                distanceCounter ++;
        }
        return distanceCounter;
    }
}
```

```java
// Astar.java

public class Astar {
    public static int getDistance(int currentPathLength, String
currentString, String targetString){
        int distanceCounter = 0;
        for(int i = 0; i < currentString.length(); i++){
            if(currentString.charAt(i) != targetString.charAt(i))
                distanceCounter ++;
        }
        return currentPathLength + 1 + distanceCounter;
    }
}
```

```java
// WordLadderSolver.java

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

public class WordLadderSolver {
    private static PriorityQueue<SearchNode> pq;
    private static HashSet<String> visited;
    private static HashMap<String, String> parents;
    private static ArrayList<String> resultPath;
    private static String target, start;
    private static EnglishWords englishWords;
    private static int nodeVisited;
```

```
    public static ArrayList<String> WordLadderSolve(String start, String
target, EnglishWords englishWords, String chosenAlgorithm)throws Exception{
        prepare(start, target, englishWords);
        try{
            return solve(chosenAlgorithm);
        }
        catch(Exception e){
            throw e;
        }
    }

    private static void prepare(String _start, String _target, EnglishWords
_englishWords){
        target = _target;
        start = _start;

        pq = new PriorityQueue<>();
        resultPath = new ArrayList<String>();
        pq.add(new SearchNode(_start, _start, 0));
        visited = new HashSet<>();
        parents = new HashMap<>();
        englishWords = _englishWords;
        nodeVisited = 0;
    }


    private static ArrayList<String> solve(String chosenAlgorithm) throws
Exception{
        SearchNode currentNode;
        ArrayList<String> nearbyWords;
        int newPathValue;
        while(!pq.isEmpty()){
            currentNode = pq.remove();

            // skip if visited
            if(visited.contains(currentNode.getWord()))
            continue;

            // clear queue if using gbfs
            if(chosenAlgorithm.equalsIgnoreCase("gbfs"))
            pq.clear();

            // add to parent value to currentWord
            if(currentNode.getParent() != null)
            parents.put(currentNode.getWord(), currentNode.getParent());

            // add to visited
            visited.add(currentNode.getWord());
```

```
            nodeVisited+=1;

            // return if target word is found
            if(currentNode.getWord().equalsIgnoreCase(target)){
                traceBackParents(currentNode.getWord());
                return resultPath;
            }

            // visit all nearby words
            nearbyWords = englishWords.getNearby(currentNode.getWord());
            for(String nearbyWord : nearbyWords){
                // skip if visited
                if(visited.contains(nearbyWord))
                    continue;

                // measure new node value depends on chosen Algorithm
                if(chosenAlgorithm.equalsIgnoreCase("ucs")){
                    newPathValue =
UCS.getDistance(currentNode.getPathLength());
                }
                else if(chosenAlgorithm.equalsIgnoreCase("gbfs")){
                    newPathValue = GBFS.getDistance(nearbyWord, target);
                }
                else{
                    assert chosenAlgorithm.equalsIgnoreCase("astar");
                    newPathValue =
Astar.getDistance(currentNode.getPathLength(), nearbyWord, target);
                }

                // add nearbyNode to queue
                pq.add(new SearchNode (nearbyWord, currentNode.getWord(),
newPathValue));
            }
        }
        throw new Exception("Target is unreachable");
    }

    private static void traceBackParents(String currentWord){
        // System.out.println(parents);
        resultPath.add(0,currentWord);
        if(!currentWord.equalsIgnoreCase(start))
            traceBackParents(parents.get(currentWord));
    }

    public static int getNodeVisited(){
        return nodeVisited;
    }
}
```

```java
// SearchNode.java

public class SearchNode implements Comparable<SearchNode> {
    private String word;
    private String parent;
    private int pathLength;

    SearchNode(String word, String parent, int pathLength){
        this.word = word;
        this.parent = parent;
        this.pathLength = pathLength;
    }

    public String getWord(){
        return word;
    }
    public int getPathLength(){
        return pathLength;
    }
    public String getParent(){
        return new String(parent);
    }

    public int compareTo(SearchNode other) {
        return Integer.compare(pathLength, other.pathLength);
    }
}
```

```java
// Program.java

import java.util.*;
import java.lang.String;

public class Program {

    private static ArrayList<String> wordPath;
    private static String start, finish, chosenAlgorithm;
    private static EnglishWords englishWords;
    public static long startTime ;
    public static void main(String[] args) {

        englishWords = new EnglishWords("src/dictionary.txt");

        takeInput();
        solve();
```

```
    }

    public static void solve() {
        System.out.println("\n========\"Result\"========");
        try{
            startTime = System.nanoTime();
            wordPath = WordLadderSolver.WordLadderSolve(start, finish,
englishWords, chosenAlgorithm);
            System.out.printf("Path :\n", wordPath.size());
            for(String word : wordPath){
                System.out.println(word);
            }
            System.out.printf("Path length: %d\n", wordPath.size());
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
        finally{
            double processMillisecondTime = (System.nanoTime()-
startTime)/1e6;
            System.out.printf("Process time(Millisecond): %f\n",
processMillisecondTime);
            int nodeVisited = WordLadderSolver.getNodeVisited();
            System.out.printf("Node visited: %d\n", nodeVisited);
        }
    }

    private static void takeInput(){
        boolean str1Valid, str2Valid, str3Valid;
        Scanner scanner = new Scanner(System.in);
        str1Valid = str2Valid = str3Valid = false;

        System.out.println("==\"Word Ladder Solver\"==");

        while(!str1Valid){
            System.out.printf("Start word: ");
            start = scanner.nextLine().trim().toLowerCase();
            if(englishWords.isWordValid(start)){
                str1Valid = true;
            }
            else{
                System.out.println("Start word is not a valid english
word.");
            }
        }

        while(!str2Valid){
```

```
            System.out.printf("Target word: ");
            finish = scanner.nextLine().trim().toLowerCase();
            if(englishWords.isWordValid(finish)){
                if(start.length() == finish.length()){
                    str2Valid = true;
                }
                else{
                    System.out.println("Input invalid, target word has
different length with start word.");
                }
            }
            else{
                System.out.println("Target word is not a valid english
word.");
            }
        }

        while(!str3Valid){
            System.out.println("Available algorithm:\n-GBFS\n-AStar\n-UCS");
            System.out.printf("Select algorithm: ");
            chosenAlgorithm = scanner.nextLine().trim();
            chosenAlgorithm.toLowerCase();
            if(chosenAlgorithm.equalsIgnoreCase("ucs") ||
chosenAlgorithm.equalsIgnoreCase("gbfs") ||
            chosenAlgorithm.equalsIgnoreCase("astar")){
                str3Valid = true;
            }
            else{
                System.out.println("Start word is not a valid algorithm.");
            }
        }
        scanner.close();
    }
}
```

BAB IV

PENGUJIAN PROGRAM

4.1 Hasil Pengujian

- Pengujian ke-1:

Start Word	chair
Target Word	choir

UCS	==="Word Ladder Solver"== Start word: chair Target word: choir Available algorithm: -GBFS -AStar -UCS Select algorithm: ucs ========"Result"======== Path : chair choir Path length: 2 Process time(Millisecond): 2.841100 Node visited: 2	
GBFS	==="Word Ladder Solver"== Start word: chair Target word: choir Available algorithm: -GBFS -AStar -UCS Select algorithm: gbfs ========"Result"======== Path : chair choir Path length: 2 Process time(Millisecond): 3.188600 Node visited: 2	
A*	==="Word Ladder Solver"== Start word: chair Target word: choir Available algorithm: -GBFS -AStar -UCS Select algorithm: astar ========"Result"======== Path : chair choir Path length: 2 Process time(Millisecond): 4.579900 Node visited: 2	

- Pengujian ke-2:

Start Word	chair
Target Word	table

UCS	==="Word Ladder Solver"== Start word: chair Target word: table Available algorithm: -GBFS -AStar -UCS Select algorithm: ucs ========"Result"======== Path : chair charr chars chats coats costs casts carts carls carle cable table Path length: 12 Process time(Millisecond): 148.195900 Node visited: 6153
GBFS	==="Word Ladder Solver"== Start word: chair Target word: table Available algorithm: -GBFS -AStar -UCS Select algorithm: gbfs ========"Result"======== Target is unreachable Process time(Millisecond): 2.435900 Node visited: 2
A*	==="Word Ladder Solver"== Start word: chair Target word: table Available algorithm: -GBFS -AStar -UCS Select algorithm: astar ========"Result"======== Path : chair charr chare chase cease pease perse parse parle carle cable table Path length: 12 Process time(Millisecond): 67.809800 Node visited: 2298

- Pengujian ke-3:

Start Word	war
Target Word	rot
UCS	==="Word Ladder Solver"== Start word: war Target word: rot Available algorithm: -GBFS -AStar -UCS Select algorithm: ucs ========"Result"======== Path : war wat rat rot Path length: 4 Process time(Millisecond): 16.476200 Node visited: 454
GBFS	==="Word Ladder Solver"== Start word: war Target word: rot Available algorithm: -GBFS -AStar -UCS Select algorithm: gbfs ========"Result"======== Path : war wat rat rot Path length: 4 Process time(Millisecond): 3.877400 Node visited: 4
A*	==="Word Ladder Solver"== Start word: war Target word: rot Available algorithm: -GBFS -AStar -UCS Select algorithm: astar ========"Result"======== Path : war wat rat rot Path length: 4 Process time(Millisecond): 5.519600 Node visited: 40

- Pengujian ke-4:

Start Word	atlases
Target Word	cabaret
UCS	Path length: 53 Process time(Millisecond): 218.948100 Node visited: 7648
GBFS	========"Result"======== Target is unreachable Process time(Millisecond): 2.442000 Node visited: 10
A*	Path length: 53 Process time(Millisecond): 200.660800 Node visited: 7623

- Pengujian ke-5:

Start Word	laptop
Target Word	mousse
UCS	==Word Ladder Solver"== Start word: laptop Target word: mousse Available algorithm: -GBFS -AStar -UCS Select algorithm: ucs ========"Result"======== Target is unreachable Process time(Millisecond): 2.026200 Node visited: 1
GBFS	==Word Ladder Solver"== Start word: laptop Target word: mousse Available algorithm: -GBFS -AStar -UCS Select algorithm: gbfs ========"Result"======== Target is unreachable Process time(Millisecond): 2.009200 Node visited: 1

A*	==="Word Ladder Solver"== Start word: laptop Target word: mousse Available algorithm: -GBFS -AStar -UCS Select algorithm: astar ========"Result"======== Target is unreachable Process time(Millisecond): 2.093400 Node visited: 1	

- Pengujian ke-6:

Start Word	weird	
Target Word	beard	
UCS	==="Word Ladder Solver"== Start word: weird Target word: beard Available algorithm: -GBFS -AStar -UCS Select algorithm: ucs ========"Result"======== Path : weird weirs wears bears beard Path length: 5 Process time(Millisecond): 6.505300 Node visited: 73	
GBFS	==="Word Ladder Solver"== Start word: weird Target word: beard Available algorithm: -GBFS -AStar -UCS Select algorithm: gbfs ========"Result"======== Path : weird weirs wears bears beard Path length: 5 Process time(Millisecond): 4.132600 Node visited: 5	

A*	``` =="Word Ladder Solver"== Start word: weird Target word: beard Available algorithm: -GBFS -AStar -UCS Select algorithm: astar ========"Result"======== Path : weird weirs wears bears beard Path length: 5 Process time(Millisecond): 4.333100 Node visited: 17 ```	

4.2 Analisis Hasil Pengujian

Berdasarkan hasil pengujian terlihat bahwa algoritma UCS dan A* akan selalu menemukan hasil yang optimal. Dalam kecepatannya A* terlihat cukup lebih baik dibandingkan UCS, hal ini dikarenakan pemilihan simpulnya yang digabungkan dengan perhitungan heuristik. GBFS juga terlihat lebih cepat dibandingkan keduanya tetapi dibarengi dengan sebuah kekurangan yaitu kemungkinan bahwa solusi yang didapat tidak optimal bahkan bisa juga tidak mendapatkan solusi karena terjebak di *local optima*. Kita juga bisa mengamati memori yang digunakan melalu penghitungan node yang dikunjungi. Urutannya sama, dari yang paling hemat memori ada GBFS, diikuti oleh A* dan terakhir UCS.

LAMPIRAN

Link Repository: https://github.com/RayNoor0/Tucil3_13522107/tree/main

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		✓

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf