

# **LAPORAN TUGAS KECIL 3**

## **IF2211 STRATEGI ALGORITMA**

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

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 2  
(dua) Tahun Akademik 2023/2024.



Disusun oleh:  
Melati Anggraini                    13522035

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

## **DAFTAR ISI**

DAFTAR ISI	1
BAB I APLIKASI DAN IMPLEMENTASI DALAM ALGORITMA UCS, Greedy Best First Search, dan A*	2
BAB II SOURCE CODE	11
BAB III PENGUJIAN	24
BAB VI BONUS	28
LAMPIRAN	31

## BAB I

### APLIKASI DAN IMPLEMENTASI DALAM ALGORITMA UCS, Greedy Best First Search, dan A\*

#### A. Cara Menggunakan Program

1. Jalankan **javac Main.java** kemudian **java Main** di terminal.
2. Pilih salah satu metode untuk menjalankan program yaitu menggunakan cli atau gui.
3. Komponen yang menjadi masukan yaitu :
  1. **startWord** : kata awal
  2. **endWord** : kata akhir
4. Untuk metode CLI, program akan **memvalidasi apakah ingin menutup program atau tidak**. Sedangkan untuk metode GUI silakan ctrl+C di terminal atau tekan button silang(X) di pojok kanan atas.

#### B. Analisis dan Implementasi Algoritma UCS dalam pembuatan World Ladder Solver

UCS atau Uniform Cost Search adalah algoritma yang mencari rute tanpa informasi menggunakan cost terendah untuk menemukan rute dari node sumber ke node tujuan. Algoritma ini beroperasi di sekitar ruang pencarian berbobot terarah untuk berpindah dari node awal ke salah satu node akhir dengan cost minimum.

Dalam implementasi nilai cost antar node diasumsikan konstan dan sama dengan 1 karena setiap langkah hanya boleh memiliki satu huruf yang berbeda. Sehingga untuk menentukan nilai cost minimum adalah dengan cara mengurutkan menurut posisi(dari kiri ke kanan) serta abjad node-node tetangga yang akan dikunjungi. Hal ini dilakukan agar urutan pencarian lebih terstruktur.

Langkah-langkah implementasi algoritma UCS dalam pemecahan World Ladder :

1. Masukan node root ke dalam priority queue dengan mempertimbangkan biaya saat ini. Implementasinya, masukkan node awal ke dalam priority queue queueUCS dengan cost awal 0. Setiap queue memiliki `final_score` sebagai  $g(n)$  yang merepresentasikan cost total saat ini dari node awal ke node tersebut.
2. Ambil elemen dari prioritas tertinggi menggunakan `poll()` method.
3. Cek apakah node tersebut sama dengan goal atau tidak. Jika node saat ini sama dengan node tujuan, maka pencarian selesai.
4. Jika node saat ini bukan node tujuan, periksa node-node tetangga dari node tersebut dan masukkan node tetangga yang belum dieksplorasi ke dalam priority queue dengan mempertimbangkan cost.

5. Ulangi langkah b-d sampai ditemukan node tujuan atau priority queue kosong.
6. Jika ditemukan node tujuan, rekonstruksi path dari node awal ke node tujuan menggunakan informasi path yang disimpan dalam pencarian.
7. Jika priority queue kosong dan tidak ada path yang ditemukan , kembalikan null.

Penggunaan algoritma BFS dengan UCS dalam pemecahan permainan World Ladder mungkin sama hanya untuk kasus ini. Keduanya menggunakan pendekatan pencarian graf melebar, UCS akan memprioritaskan node berdasarkan cost untuk mencapai node tersebut ( $g(n)$ ), sedangkan BFS tidak dan hanya mempertimbangkan kedalaman dari node saat ini. Penulis menyebutkan mungkin karena terdapat kesamaan yang unik dalam pemecahan permasalahan ini, untuk mencapai ke path selanjutnya pasti nilai costnya bertambah satu yang mencerminkan kedalaman yang sama. Namun, perlu diingat untuk implementasi yang lebih umum, UCS akan memberikan prioritas berbeda pada node berdasarkan nilai cost, sementara BFS hanya mempertimbangkan kedalaman dari node saja.

### **C. Aplikasi Algoritma Greedy Best First Search dalam pembuatan World Ladder Solver**

Greedy Best-First Search (GBFS) adalah algoritma pencarian yang berusaha menemukan jalur yang paling menjanjikan dari titik awal ke tujuan. Algoritma ini memprioritaskan jalur yang tampaknya paling menjanjikan, tanpa memperhatikan apakah jalur tersebut benar-benar merupakan jalur terpendek. GBFS menggunakan fungsi heuristik( $h(n)$ ) yaitu fungsi untuk menghitung cost dari node(n) ke tujuan yang dipakai berdasarkan kondisi node saat itu sehingga perhitungan costnya bersifat heuristic dan tidak aktual.

Definisi

Langkah-langkah implementasi GBFS dalam pemecahan masalah :

1. Inisialisasi rawPath dengan startWord.
2. Selama belum mencapai endWord:
  - Pilih kata dengan prioritas heuristic terendah yang belum dieksplorasi. Fungsi heuristic(currentWord, endWord) menghitung jumlah karakter yang berbeda antara currentWord dan endWord. Semakin sedikit karakter yang berbeda, semakin dekat kata saat ini dengan kata akhir. Fungsi ini mengasumsikan bahwa setiap karakter yang berbeda dapat diubah dalam satu langkah(misalnya, mengganti satu huruf)
  - Tandai kata tersebut sebagai dieksplorasi.
  - Cari tetangga yang berbeda satu huruf dengan kata saat ini.
  - Urutkan tetangga berdasarkan abjad dan prioritas posisi.
  - Tambahkan tetangga yang belum dieksplorasi ke rawPath.

3. Rekonstruksi jalur dari endWord ke startWord.
4. Hasilkan jalur dan jumlah node yang dieksplorasi.

Secara teoritis, GBFS tidak menjamin solusi optimal untuk persoalan World Ladder. Algoritma ini hanya mempertimbangkan heuristik dan tidak mempertimbangkan panjang jalur sebenarnya. Dalam beberapa kasus, GBFS dapat menghasilkan jalur yang lebih panjang daripada solusi optimal dan bahkan tidak menghasilkan solusi karena terjebak lokal optima.

#### **D. Aplikasi Algoritma Greedy Best First Search dalam pembuatan World Ladder Solver**

Algoritma A\* adalah salah satu algoritma yang digunakan dalam mencari rute dan graf traversal. Pada setiap langkah, algoritma A\* memilih node berdasarkan  $f\text{-value}(f(n))$  dimana  $f\text{-value}(f(n))$  adalah  $g\text{-value}$  ditambah  $h\text{-value}(h(n))$ .  $G\text{-value}(g(n))$  merupakan total weight dari matriks ketetanggaan dan  $h\text{-value}(h(n))$  dimana adalah nilai heuristik atau nilai straight line dari node awal ke node tersebut.

Definisi  $f(n)$  dan  $g(n)$  :

- Dalam A\*,  $(f(n))$  adalah fungsi evaluasi total yang menggabungkan cost sejauh ini  $((g(n)))$  dan estimasi cost tersisa  $((h(n)))$ .
- $(g(n))$  adalah cost aktual yang dikeluarkan untuk mencapai node  $(n)$ .
- $(h(n))$  adalah estimasi cost dari node  $(n)$  ke tujuan.

Langkah-langkah implementasi Algoritma A\* dalam penemuan solusi World Ladder :

1. Inisialisasi priorityQueue(queueAStar) dengan elemen pertama berisi nilai heuristic dari startWord. Selain itu juga inisialisasi rawPath untuk melacak jalur dan costFar untuk melacak cost yang digunakan sejauh ini.
2. Iterasi, selama prioritas tidak kosong :
  - a. Selama antrian prioritas tidak kosong:
    - Ambil elemen dengan prioritas terendah dari antrian prioritas (berdasarkan nilai  $f(n)$ ).
    - Tandai kata saat ini sebagai dieksplorasi.
    - Cari tetangga yang berbeda satu huruf dengan kata saat ini.
    - Hitung biaya baru untuk mencapai tetangga ini (selalu 1 karena perbedaan hanya satu huruf).
    - Jika tetangga belum pernah dikunjungi atau biaya lebih rendah, perbarui informasi dan tambahkan ke antrian prioritas.
    - Rekonstruksi jalur dari endWord ke startWord.
3. Hasil

- Jika tidak ada path yang ditemukan, kembalikan null.
- Jika ada, kembalikan jalur dan jumlah node yang dieksplorasi.

Heuristik yang hanya bergantung pada biaya (seperti yang digunakan dalam algoritma A\* pada kasus Word Ladder) adalah heuristik yang admissible. Ini berarti heuristik tersebut tidak pernah memperkirakan biaya mencapai tujuan dengan lebih tinggi daripada biaya sebenarnya dari titik saat ini ke tujuan. Dengan demikian, algoritma A\* dengan heuristik admissible dapat menemukan solusi optimal dengan efisien.

Secara teoritis, algoritma A\* lebih efisien daripada algoritma UCS (Uniform-Cost-Search) dalam kasus Word Ladder. Hal ini dikarenakan A\* menggunakan heuristik untuk memprioritaskan ekspansi node yang lebih mungkin mengarah ke solusi. Ini memungkinkan A\* untuk menghindari ekspansi node yang tidak relevan dan fokus pada path yang potensial. Selain itu, Optimalitas yang diberikan A\* menjamin solusi optimal jika heuristiknya admissible. Dalam Word Ladder, heuristik membantu menemukan solusi lebih cepat daripada UCS. Terakhir, kompleksitas A\* biasanya lebih efisien karena mengurangi jumlah node yang dieksplorasi dengan menggunakan heuristik.

## E. Analisis Perbandingan Solusi UCS, Greedy Best First Search, dan A\*

Start-End Word	Output	
	Algoritma	Keterangan
snake-world	<pre>Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1 Word Ladder Path: snake snare scare scars sears seals weels weeld woeld world Banyak node yang dikunjungi: 3794 Waktu Eksekusi: 25,69200000 detik</pre>	[UCS]Solusi optimal, waktu eksekusi paling lambat(25,7 detik)

	<pre> Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  snake stake stale shale whale while white whits waits waills walls weils weals weald woald world world  Banyak node yang dikunjungi: 48 Waktu Eksekusi: 0,41500000 detik </pre>	[GBFS]Solusi tidak optimal(panjang path 16), waktu eksekusi tercepat(0,41 detik)
	<pre> Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  snake shake shame shams seams seals seals weals weald woald world world  Banyak node yang dikunjungi: 359 Waktu Eksekusi: 1,46800000 detik </pre>	[A*]Solusi optimal, waktu eksekusi cenderung cepat(1,47 detik)
love-your	<pre> Masukkan kata awal (start word): love Masukkan kata akhir (end word): your Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  love lore lord loud lour your your  Banyak node yang dikunjungi: 2220 Waktu Eksekusi: 14,55500000 detik </pre>	[UCS]Solusi optimal, waktu eksekusi paling lambat(25,7 detik)
	<pre> Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  love move moue roue roux doux dour your your  Banyak node yang dikunjungi: 8 Waktu Eksekusi: 0,06500000 detik </pre>	[GBFS]Solusi tidak optimal(panjang path 16), waktu eksekusi tercepat(0,41 detik)
	<pre> Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  love lose lost lout lour your your  Banyak node yang dikunjungi: 61 Waktu Eksekusi: 0,50400000 detik </pre>	[A*]Solusi optimal, waktu eksekusi cenderung cepat(1,47 detik)

head-tail	<pre> Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 1034 Waktu Eksekusi: 6,97500000 detik </pre>	<p>[UCS] Solusi optimal, waktu eksekusi paling lambat(6,9 detik)</p>
	<pre> Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 5 Waktu Eksekusi: 0,03300000 detik </pre>	<p>[GBFS] Solusi optimal , waktu eksekusi tercepat(0,033 detik)</p>
	<pre> Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 6 Waktu Eksekusi: 0,04600000 detik </pre>	<p>[A*] Solusi optimal, waktu eksekusi cenderung cepat(0,046 detik)</p>
folder-family	<pre> Masukkan kata awal (start word): folder Masukkan kata akhir (end word): family Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  folder bolder balder balker barker darker darkey darkly dankly lankly lanely lameley gameley gamily family Banyak node yang dikunjungi: 7783 Waktu Eksekusi: 59,46500000 detik </pre>	<p>[UCS] Solusi optimal, waktu eksekusi paling lambat(59,46 detik)</p>
	<pre> lamely gamely gamily family Banyak node yang dikunjungi: 97 Waktu Eksekusi: 0,82000000 detik </pre>	<p>[GBFS] Solusi tidak optimal(panjang path 36), waktu eksekusi tercepat(0,82 detik), memori</p>

	<pre> Masukkan kata awal (start word): folder Masukkan kata akhir (end word): family Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  folder bolder baider balker barker darker darkey darkly dankly lankly lanely lamely gameiy gamily family Banyak node yang dikunjungi: 4282 Waktu Eksekusi: 22,5700000 detik </pre>	[A*]Solusi optimal, waktu eksekusi cenderung lama(22,57 detik)
zero-five	<pre> Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 843 Waktu Eksekusi: 5,53200000 detik </pre>	[UCS]Solusi optimal, waktu eksekusi paling lambat(5,53 detik)
	<pre> Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 6 Waktu Eksekusi: 0,06600000 detik </pre>	[GBFS]Solusi optimal, waktu eksekusi tercepat(0,06 detik)
	<pre> Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 13 Waktu Eksekusi: 0,14200000 detik </pre>	[A*]Solusi optimal, waktu eksekusi cenderung cepat(0,142 detik)

Tabel 3.1 Perbandingan Hasil Ketiga Algoritma

Kesimpulan data tabel :

1. Uniform-Cost Search (UCS) :
  - **Optimalitas** : Solusi Optimal
  - **Waktu Eksekusi** : Paling lambat (25,7 detik)
2. GBFS :
  - **Optimalitas** : Solusi tidak Optimal bisa dilihat dari tabel kolom 2 yang berwarna merah untuk algoritma GBFS sendiri.
  - **Waktu Eksekusi** : Tercepat (0,41 detik)
3. A\* :
  - **Optimalitas** : Solusi optimal
  - **Waktu Eksekusi** : Cenderung cepat (0,142 detik)

- **Optimalitas** : Solusi Optimal
- **Waktu Eksekusi** : Cenderung cepat(1,47 detik)

Berdasarkan data Tabel 3.1 UCS memberikan solusi optimal , tetapi memiliki waktu eksekusi yang paling lambat. GBFS memiliki waktu eksekusi yang sangat cepat, tetapi solusi yang dihasilkan tidak selalu optimal, hanya kasus-kasus tertentu saja bisa optimal seperti kondisi ditemukannya path dalam penelusuran awal yang optimal. Sedangkan, A\* menggabungkan keuntungan UCS dan GBFS dengan memberi solusi yang optimal dan waktu eksekusi yang cenderung cepat.

### Analisis Penggunaan Memori

#### 1. Algoritma UCS

- Struktur Data yang Digunakan:
  - a. rawPath : Map<String, String> yang melacak path dari startWord ke endWord.
  - b. costFar : Map<String, Double> yang menyimpan cost sejauh ini untuk mencapai setiap data
  - c. Explored : Set<String> menyimpan kata-kata yang dieksplorasi
  - d. Neighbors : List<String> berisi node-node tetangga(anak) yang berbeda satu huruf dengan kata saat ini
- Penggunaan Memori

**Keuntungan :** Penggunaan memori rawPath, costSoFar dan explored relatif efisien karena hanya menyimpan kata-kata yang dieksplorasi serta neighbors juga menyimpan node tetangga yang relevan untuk pencarian saat ini.

**Kekurangan :** Penggunaan memori bisa meningkat jika kamus kata sangat besar, struktur data rawPath dan explored memerlukan alokasi memori tambahan untuk setiap kata, jumlah kata yang dieksplorasi exploredNodes juga mempengaruhi penggunaan memori. Sebenarnya List bisa menjadi solusi, akan tetapi untuk pencarian List harus melakukan iterasi, memakan waktu, sehingga digunakan Map agar pencarian node-node anak bisa lebih cepat.

#### 2. Analisis Penggunaan Memori Algoritma GBFS

Perbedaan penggunaan memori hanya terletak pada penggunaan costFar, karena algoritma tidak berdasarkan cost dalam perhitungan nya sehingga tidak diperlukan struktur data ini.

#### 3. Analisis Penggunaan Memori Algoritma A\*

Dalam algoritma ini, terdapat penambahan struktur data priorityQueue (queueAStar) yang digunakan untuk mengatur ekspansi node.

Secara umum, dari ketiga deskripsi struktur data diatas adalah semua algoritma memiliki penggunaan memori yang wajar dan tidak menunjukkan boros memori. Dalam konteks World Ladder ketiganya digunakan dengan baik tanpa khawatir boros memori.

## BAB II

### SOURCE CODE

#### 4.1 Word Ladder Solver dengan Algoritma UCS



```
1 // Method untuk menyelesaikan Word Ladder menggunakan algoritma UCS
2 public List<Object> solveUCS(String startWord, String endWord, English dictionary) {
3     // Inisialisasi PriorityQueue untuk UCS
4     PriorityQueue<QueueElementUCS> queueUCS = new PriorityQueue<>();
5     QueueElementUCS node = new QueueElementUCS(0, startWord);
6     queueUCS.add(node);
7     // Inisialisasi path dan cost
8     Map<String, String> rawPath = new HashMap<>();
9     Map<String, Double> costSoFar = new HashMap<>();
10    costSoFar.put(startWord, 0.0);
11    Set<String> explored = new HashSet<>();
12    int exploredNodes = 0;
13    while (!queueUCS.isEmpty()) {
14        // ambil simpul dengan cost minimum dari priorityQueue
15        QueueElementUCS currentElement = queueUCS.poll();
16        String currentWord = currentElement.word;
17        exploredNodes++;
18        // Jika kata saat ini adalah kata akhir, maka pencarian selesai
19        if (currentWord.equals(endWord)) {
20            break;
21        }
22        // Tandai kata saat ini sebagai sudah dieksplorasi
23        explored.add(currentWord);
24        // Cari node-node tetangga yang berbeda satu huruf dengan kata(node) saat ini
25        List<String> neighbors = findNeighbors(currentWord, dictionary);
26        String finalCurrentWord = currentWord;
27        neighbors.sort(new Comparator<String>() {
28            @Override
29            public int compare(String word1, String word2) {
30                // Prioritaskan berdasarkan urutan abjad
31                int compareAlpha = word1.compareTo(word2);
32                if (compareAlpha != 0) {
33                    return compareAlpha;
34                } else {
35                    // Jika urutan abjad sama, prioritas posisi dari kiri ke kanan
36                    return finalCurrentWord.indexOf(word1.charAt(0)) - finalCurrentWord.indexOf(word2.charAt(0));
37                }
38            }
39        });
40        for (String neighbor : neighbors) {
41            if (!explored.contains(neighbor)) {
42                // update cost, selalu 1 karena perbedaan hanya satu huruf
43                double newCost = costSoFar.get(currentWord) + 1;
44
45                // Jika tetangga belum pernah dikunjungi atau biaya baru lebih rendah, perbarui informasi
46                if (!costSoFar.containsKey(neighbor) || newCost < costSoFar.get(neighbor)) {
47                    costSoFar.put(neighbor, newCost);
48                    double priority = newCost;
49                    QueueElementUCS newElement = new QueueElementUCS(priority, neighbor);
50                    queueUCS.add(newElement);
51                    rawPath.put(neighbor, currentWord);
52                }
53            }
54        }
55    }
}
```

```
● ● ●  
1 // Jika tidak ada path yang ditemukan, return null  
2 if (!costSoFar.containsKey(endWord)) {  
3     List<Object> result = new ArrayList<>();  
4     result.add(null);  
5     result.add(exploredNodes);  
6     return result;  
7 }  
8 // Rekonstruksi path dari startWord ke endWord  
9 List<String> path = new ArrayList<>();  
10 String word = endWord;  
11 while (!word.equals(startWord)) {  
12     path.add(0, word);  
13     word = rawPath.get(word);  
14 }  
15 // Jika path ditemukan, tambahkan startWord ke path  
16 path.add(0, startWord);  
17 // Return path dan banyak node yang dikunjungi  
18 List<Object> result = new ArrayList<>();  
19 result.add(path);  
20 result.add(exploredNodes);  
21 return result;  
22 }
```

```
● ● ●  
1 // Kelas untuk merepresentasikan elemen dalam antrian prioritas untuk algoritma UCS  
2 static class QueueElementUCS implements Comparable<QueueElementUCS> {  
3     double final_score;  
4     String word;  
5     public QueueElementUCS(double final_score, String word) {  
6         this.final_score = final_score;  
7         this.word = word;  
8     }  
9     @Override  
10    public int compareTo(QueueElementUCS other) {  
11        return Double.compare(this.final_score, other.final_score);  
12    }  
13 }
```

```
1 // Metode untuk menghitung jumlah perbedaan antara dua kata
2 private int countDifferences(String word1, String word2) {
3     int count = 0;
4     for (int i = 0; i < word1.length(); i++) {
5         if (word1.charAt(i) != word2.charAt(i)) {
6             count++;
7         }
8     }
9     return count;
10 }
```

```
1 // Metode untuk mencari node-node tetangga yang berbeda satu huruf dengan kata tertentu
2 private List<String> findNeighbors(String word, English dictionary) {
3     List<String> neighbors = new ArrayList<>();
4     // Iterasi melalui semua kata dalam kamus
5     for (String candidate : dictionary.getWords()) {
6         // Jika panjangnya sama dengan kata saat ini dan berbeda satu huruf, tambahkan sebagai tetangga
7         if (candidate.length() == word.length() && countDifferences(candidate, word) == 1) {
8             neighbors.add(candidate);
9         }
10    }
11    return neighbors;
12 }
```

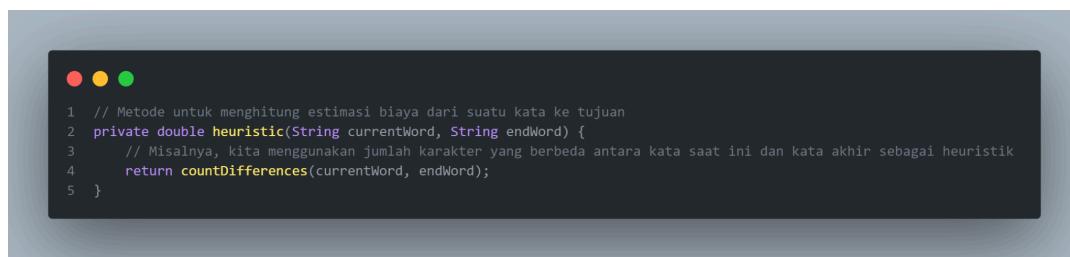
Method - method yang digunakan dalam kelas ini :

1. Metode **findNeighbors**:
  - a. Metode ini mencari node-node tetangga(anak) yang berbeda satu huruf dengan kata tertentu.
  - b. Iterasi melalui semua kata dalam kamus dan memeriksa apakah panjangnya sama dengan kata saat ini dan berbeda satu huruf.
  - c. Jika ya, kata tersebut ditambahkan sebagai tetangga.
  - d. Metode ini memanfaatkan kelas English yang mengandung daftar kata-kata dalam kamus.
2. Metode **countDifferences**:
  - a. Metode ini menghitung jumlah perbedaan antara dua kata.
  - b. Mengiterasi melalui karakter-karakter kata dan membandingkannya.
  - c. Jumlah karakter yang berbeda menjadi hasil perhitungan.
3. Kelas **QueueElementUCS**:

- a. Kelas ini merepresentasikan elemen dalam antrian prioritas untuk algoritma Uniform Cost Search (UCS).
  - b. Memiliki atribut `final_score` (biaya total) dan `word` (kata yang sedang dievaluasi).
  - c. Mengimplementasikan metode `compareTo` untuk membandingkan elemen berdasarkan biaya total.
4. Metode `solveUCS`:
- a. Metode ini menyelesaikan Word Ladder menggunakan algoritma UCS.
  - b. Inisialisasi antrian prioritas (PriorityQueue) dengan simpul awal.
  - c. Selama antrian tidak kosong, ambil simpul dengan biaya terendah.
  - d. Cari tetangga-tetangga yang belum dieksplorasi dan perbarui biaya.
  - e. Jika kata saat ini adalah kata akhir, pencarian selesai.

## 4.2 Word Ladder Solver dengan Algoritma *Greedy Best First Search*

```
1  public List<Object> solveGreedyBestFirstSearch(String startWord, String endWord, English dictionary) {
2      // Inisialisasi path
3      Map<String, String> rawPath = new HashMap<>();
4      Set<String> explored = new HashSet<>();
5      int exploredNodes = 0;
6
7      rawPath.put(startWord, null);
8
9      while(true){
10         String currentWord = null;
11         exploredNodes++;
12         double minPriority = Double.MAX_VALUE;
13         // Pilih node dengan prioritas minimum yang belum dieksplorasi
14         for(String word : rawPath.keySet()){
15             if(!explored.contains(word)){
16                 double priority = heuristic(word, endWord);
17                 if(priority < minPriority){
18                     minPriority = priority;
19                     currentWord = word;
20                 }
21             }
22         }
23
24         // Jika kata saat ini adalah kata akhir, maka pencarian selesai
25         if(currentWord == null){
26             List<Object> result = new ArrayList<>();
27             result.add(null);
28             result.add(exploredNodes);
29             return result;
30         }
31         if (currentWord != null && currentWord.equals(endWord)) {
32             break;
33         }
34
35         // Tandai kata saat ini sebagai sudah dieksplorasi
36         explored.add(currentWord);
37
38         // Cari node-node tetangga yang berbeda satu huruf dengan kata saat ini
39         List<String> neighbors = findNeighbors(currentWord, dictionary);
40         // Urutkan tetangga berdasarkan abjad dan prioritas posisi
41         neighbors.sort(new Comparator<String>() {
42             @Override
43             public int compare(String word1, String word2) {
44                 return word1.compareTo(word2);
45             }
46         });
47
48         for (String neighbor : neighbors) {
49             if (!explored.contains(neighbor)) {
50                 // Jika node tetangga belum pernah dikunjungi, tambahkan ke path
51                 if (!rawPath.containsKey(neighbor)) {
52                     rawPath.put(neighbor, currentWord);
53                 }
54             }
55         }
56     }
57
58     // Rekonstruksi path dari startWord ke endWord
59     List<String> path = new ArrayList<>();
60     String word = endWord;
61     while (!word.equals(startWord)) {
62         path.add(0, word);
63         word = rawPath.get(word);
64     }
65     rawPath.clear();
66
67     // Jika path ditemukan, tambahkan startWord ke path
68     if (!path.get(0).equals(startWord)) {
69         path.add(0, startWord);
70     }
71     // Tambahan endWord ke path
72     List<Object> result = new ArrayList<>();
73     result.add(path);
74     result.add(exploredNodes);
75     return result;
76 }
```



```
1 // Metode untuk menghitung estimasi biaya dari suatu kata ke tujuan
2 private double heuristic(String currentWord, String endWord) {
3     // Misalnya, kita menggunakan jumlah karakter yang berbeda antara kata saat ini dan kata akhir sebagai heuristik
4     return countDifferences(currentWord, endWord);
5 }
```

Method-Method yang digunakan dalam kelas ini :

1. Metode **heuristic**:
  - a. Metode ini menghitung estimasi biaya dari suatu kata ke tujuan (misalnya, kata akhir).
  - b. Dalam implementasi ini, kita menggunakan jumlah karakter yang berbeda antara kata saat ini dan kata akhir sebagai heuristik.
2. Metode **findNeighbors**:
  - a. Metode ini mencari node-node tetangga(anak) yang berbeda satu huruf dengan kata tertentu.
  - b. Iterasi melalui semua kata dalam kamus dan memeriksa apakah panjangnya sama dengan kata saat ini dan berbeda satu huruf.
  - c. Jika ya, kata tersebut ditambahkan sebagai tetangga.
  - d. Metode ini memanfaatkan kelas English yang mengandung daftar kata-kata dalam kamus.
3. Metode **solveGreedyBestFirstSearch**:
  - a. Metode ini menyelesaikan Word Ladder menggunakan algoritma GBFS.
  - b. Inisialisasi antrian Map (rawPath) yang menyimpan path yang dilalui, explored, dan exploredNodes.
  - c. Selama antrian tidak kosong, ambil simpul dengan biaya terendah.
  - d. Cari tetangga-tetangga yang belum dieksplorasi dan perbarui biaya.
  - e. Jika kata saat ini adalah kata akhir, pencarian selesai.

### 4.3 Word Ladder Solver dengan Algoritma A\*



```
1 // Metode untuk menyelesaikan Word Ladder menggunakan algoritma A*
2 public List<Object> solveAstar(String startWord, String endWord, English dictionary) {
3     // Inisialisasi antrian prioritas untuk A*
4     PriorityQueue<QueueElementGreedyBFS> queueAStar = new PriorityQueue<>();
5     QueueElementGreedyBFS node = new QueueElementGreedyBFS(heuristic(startWord, endWord), startWord);
6     queueAStar.add(node);
7     // Inisialisasi path dan cost
8     Map<String, String> rawPath = new HashMap<>();
9     Map<String, Double> costSoFar = new HashMap<>();
10    costSoFar.put(startWord, 0.0);
11    Set<String> explored = new HashSet<>();
12    int exploredNodes = 0;
13    while (!queueAStar.isEmpty()) {
14        QueueElementGreedyBFS currentElement = queueAStar.poll(); // Ambil node dengan prioritas terendah dari antrian prioritas
15        String currentWord = currentElement.word;
16        exploredNodes++;
17        // Jika kata saat ini adalah kata akhir, maka pencarian selesai
18        if (currentWord.equals(endWord)) {
19            break;
20        }
21        // Tandai kata saat ini sebagai sudah dieksplorasi
22        explored.add(currentWord);
23        // Cari node-node tetangga yang berbeda satu huruf dengan kata saat ini
24        List<String> neighbors = findNeighbors(currentWord, dictionary);
25        neighbors.sort(new Comparator<String>() {
26            @Override
27            public int compare(String word1, String word2) {
28                return word1.compareTo(word2);
29            }
30        });
31        for (String neighbor : neighbors) {
32            if (!explored.contains(neighbor)) {
33                // Hitung cost untuk mencapai tetangga ini
34                double newCost = costSoFar.get(currentWord) + 1; // cost selalu 1 karena perbedaan hanya satu huruf
35
36                // Jika node tetangga belum pernah dikunjungi atau cost lebih rendah, perbarui informasi
37                if (!costSoFar.containsKey(neighbor) || newCost < costSoFar.get(neighbor)) {
38                    costSoFar.put(neighbor, newCost);
39                    double priority = newCost + heuristic(neighbor, endWord);
40                    QueueElementGreedyBFS newElement = new QueueElementGreedyBFS(priority, neighbor);
41                    queueAStar.add(newElement);
42                    rawPath.put(neighbor, currentWord);
43                }
44            }
45        }
46    }
47    // Jika tidak ada path yang ditemukan, kembalikan null
48    if (!costSoFar.containsKey(endWord)) {
49        List<Object> result = new ArrayList<>();
50        result.add(null);
51        result.add(exploredNodes);
52        return result;
53    }
54    // Rekonstruksi path dari startWord ke endWord
55    List<String> path = new ArrayList<>();
56    String word = endWord;
57    while (!word.equals(startWord)) {
58        path.add(0, word);
59        word = rawPath.get(word);
60    }
61    path.add(0, startWord);
62    List<Object> result = new ArrayList<>();
63    result.add(path);
64    result.add(exploredNodes);
65    return result;
66 }
```



```
1 static class QueueElementGreedyAStar implements Comparable<QueueElementGreedyAStar> {
2     double heuristic_score;
3     String word;
4     public QueueElementGreedyAStar(double heuristic_score, String word) {
5         this.heuristic_score = heuristic_score;
6         this.word = word;
7     }
8     @Override
9     public int compareTo(QueueElementGreedyAStar other) {
10        return Double.compare(this.heuristic_score, other.heuristic_score);
11    }
12 }
```

1. Metode `solveAStar`: Metode ini menyelesaikan Word Ladder menggunakan algoritma A\*.
2. Metode `QueueElementGreedyAStar` :
  - a. Kelas ini menggabungkan skor heuristik (estimasi biaya) dengan kata (node) yang akan dieksplorasi.
  - b. Konstruktor menginisialisasi objek `QueueElementGreedyAStar` dengan dua parameter yaitu `heuristic_score` yaitu biaya estimasi dari kata saat ini ke tujuan (misalnya, dari `startWord` ke `endWord`). Dan `word` yaitu kata aktual (node) yang terkait dengan elemen ini.

## 4.4 Kelas Output

```
1 public class Output {
2     public static final String ANSI_PURPLE = "\u001B[35m";
3     public static final String ANSI_RESET = "\u001B[0m";
4     // Method to print Word Ladder path to JTextArea
5     public static void printWordLadder(List<String> path, JTextArea outputTextArea) {
6         outputTextArea.setFont(new Font("Poppins", Font.PLAIN, 16));
7         if (path == null) {
8             outputTextArea.append("\nPath tidak ditemukan.\n");
9         } else {
10            outputTextArea.append("\nWord Ladder Path: ");
11            if (path.isEmpty()) {
12                outputTextArea.append("Path tidak ditemukan..\n");
13            } else {
14                for (String word : path) {
15                    if(path.indexOf(word) == path.size()-1) {
16                        outputTextArea.append(word);
17                    } else {
18                        outputTextArea.append(word + "-->");
19                    }
20                }
21            }
22        }
23        outputTextArea.append("\nPanjang path : " + path.size() + "\n");
24    }
25    // Method to print Word Ladder path using CLI
26    public static void printWordLadderCLI(List<String> path) {
27        if (path == null) {
28            System.out.println("\nPath tidak ditemukan..\n");
29        } else {
30            System.out.println("\nWord Ladder Path:\n");
31            if (path.isEmpty()) {
32                System.out.println("Path tidak ditemukan..\n");
33            } else {
34                for (String word : path) {
35                    System.out.println(word);
36                }
37            }
38        }
39    }
40    // Methid to print the Banyak node yang dikunjungi using CLI
41    public static void printStepCountCLI(int exploredNodes) {
42        System.out.println("Banyak node yang dikunjungi: " + exploredNodes);
43    }
44    // Method to print the Banyak node yang dikunjungi to JTextArea
45    public static void printStepCount(int exploredNodes, JTextArea outputTextArea) {
46        outputTextArea.setFont(new Font("Poppins", Font.PLAIN, 16));
47        outputTextArea.append("Banyak node yang dikunjungi: " + exploredNodes + "\n");
48    }
49
50    // Method to print Waktu Eksekusi to JTextArea
51    public static void printExecutionTime(long startTime, long endTime, JTextArea outputTextArea) {
52        outputTextArea.setFont(new Font("Poppins", Font.PLAIN, 16));
53        double executionTime = (double) (endTime - startTime) / 1000.0;
54        String formattedExecutionTime = String.format("%.8f", (double) executionTime);
55        outputTextArea.append("Waktu Eksekusi: " + formattedExecutionTime + " detik\n");
56    }
57
58    // Method to print Waktu Eksekusi using CLI
59    public static void printExecutionTimeCLI(long startTime, long endTime) {
60        double executionTime = (double) (endTime - startTime) / 1000.0;
61        String formattedExecutionTime = String.format("%.8f", (double) executionTime);
62        System.out.println("Waktu Eksekusi: " + formattedExecutionTime + " detik\n");
63    }
64 }
```

Metode dalam kelas Output memiliki tujuan tertentu untuk mengelola output dan informasi terkait jalur kata (Word Ladder) serta penghitungan node yang dikunjungi.

1. Method `printWordLadder`:

- Metode ini digunakan untuk mencetak jalur kata (Word Ladder) ke dalam komponen JTextArea.
- Jika jalur tidak ditemukan (path == null), akan mencetak pesan “Path tidak ditemukan.”
- Jika jalur kosong (path.isEmpty()), akan mencetak pesan “Path tidak ditemukan...”
- Jika jalur ada, akan mencetak kata-kata jalur dengan tanda “->” di antara mereka.

2. Method `printWordLadderCLI`:

- Metode ini digunakan untuk mencetak jalur kata (Word Ladder) ke konsol (CLI).
- Sama seperti `printWordLadder`, namun output ditampilkan di konsol.

3. Method `printStepCount`:

- Metode ini mencetak jumlah node yang dikunjungi ke dalam komponen JTextArea.
- Menggunakan font “Poppins” dengan ukuran 16.
- Menampilkan pesan “Banyak node yang dikunjungi: [jumlah]” di area teks.

4. Method `printStepCountCLI`:

- Metode ini mencetak jumlah node yang dikunjungi ke konsol (CLI).
- Menampilkan pesan “Banyak node yang dikunjungi: [jumlah]” di konsol.

5. Method `printExecutionTime`:

- Metode ini mencetak waktu eksekusi ke dalam komponen JTextArea.
- Menggunakan font “Poppins” dengan ukuran 16.
- Menghitung waktu eksekusi dari startTime hingga endTime dan menampilkan dalam detik.

6. Method `printExecutionTimeCLI`:

- Metode ini mencetak waktu eksekusi ke konsol (CLI).
- Menghitung waktu eksekusi dari startTime hingga endTime dan menampilkan dalam detik.

## 4.5 Kelas English

```
1  public class English {
2      private HashSet<String> words;
3      public English() {
4          words = new HashSet<>();
5          loadWordsFromFile("dictionary.txt");
6      }
7      private void loadWordsFromFile(String filename) {
8          try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
9              String line;
10             while ((line = reader.readLine()) != null) {
11                 words.add(line.trim().toLowerCase());
12             }
13         } catch (IOException e) {
14             System.err.println("Error loading words from file: " + e.getMessage());
15         }
16     }
17     public boolean isValidWord(String word) {
18         return words.contains(word.toLowerCase());
19     }
20     // Getter method for the words field
21     public HashSet<String> getWords() {
22         return words;
23     }
24 }
25
```

Metode dalam kelas English memiliki tujuan untuk mengelola kamus kata-kata dalam bahasa Inggris.

1. Konstruktor English():
  - Metode ini merupakan konstruktor kelas English.
  - Ketika objek English dibuat, konstruktor ini akan dijalankan.
  - Pada konstruktor ini, kita melakukan inisialisasi kamus kata-kata dengan memuat kata-kata dari file “dictionary.txt”.
2. Metode loadWordsFromFile(String filename):
  - Metode ini bertanggung jawab untuk memuat kata-kata dari file teks (dalam hal ini, “dictionary.txt”) ke dalam kamus.
  - Langkah-langkahnya adalah sebagai berikut:
    - Membuka file dengan nama yang diberikan (filename).
    - Membaca setiap baris dari file.
    - Menghapus spasi di awal dan akhir baris (trim) dan mengubah semua huruf menjadi huruf kecil (lowercase).
    - Menambahkan kata yang telah diolah ke dalam kamus (words).
3. Metode isValidWord(String word):

- Metode ini digunakan untuk memeriksa apakah suatu kata valid (ada dalam kamus) atau tidak.

- Mengembalikan true jika kata ada dalam kamus, dan false jika tidak.

4. Getter method getWords():

- Metode ini mengembalikan seluruh kata yang ada dalam kamus.

- Digunakan untuk mengakses kamus kata-kata dari luar kelas.

#### 4.6 Main.java

Kelas ini akan menghandle seluruh proses yang ada di program. Terdiri dari method RunGUI() yang digunakan untuk menjalankan program dengan memakai GUI, runCLI() yang digunakan untuk menjalankan program dengan memakai CLI, dan NextOROut() yang digunakan untuk menjalankan program untuk memvalidasi apakah pengguna ingin keluar atau melanjutkan permainan.

```

1  public class Main {
2      public static final String ANSI_RED = "\u001B[31m";
3      public static final String ANSI_RESET = "\u001B[0m";
4      public static final String ANSI_PURPLE = "\u001B[35m";
5      public static final String ANSI_GREEN = "\u001B[32m";
6      public static void main(String[] args) {
7          Scanner scanner = new Scanner(System.in);
8          System.out.println("Choose input method: (1 = CLI , 2 = GUI , 3 = Exit)");
9          if (scanner.hasNextInt()) {
10              int inputChoice = scanner.nextInt();
11              if (inputChoice == 1) {
12                  // Input via CLI
13                  runCLI();
14              } else if (inputChoice == 2) {
15                  // Input via GUI
16                  RunGUI();
17              } else if (inputChoice == 3) {
18                  return;
19              } else {
20                  System.out.println("Invalid input method choice.");
21                  main(args);
22              }
23          } else {
24              String invalidInput = scanner.next(); // Ambil input yang tidak valid
25              System.out.println("Invalid input method choice: " + invalidInput + ". Please enter a valid integer.");
26              main(args);
27          }
28      }
29
30      public static void runCLI() {
31          Scanner scanner = new Scanner(System.in);
32          English dictionary = new English();
33          WordLadderSolver wordLadderSolver = new WordLadderSolver();
34
35          String startWord;
36          System.out.print(ANSI_GREEN+"*****");
37          System.out.print(" Selamat bermain di program Word Ladder Solver! *****");
38          System.out.print(ANSI_RESET);
39
40          System.out.print("Masukkan kata awal (start word): ");
41          startWord = scanner.nextLine().toLowerCase();
42
43          // Validasi kata awal
44          while (!dictionary.isValidWord(startWord) || startWord.split("\s+").length > 1) {
45              if (!dictionary.isValidWord(startWord)) {
46                  System.out.println(ANSI_RED + "Kata awal tidak valid." + ANSI_RESET);
47              } else if (startWord.split("\s+").length > 1) {
48                  System.out.println(ANSI_RED + "Tolong masukkan hanya satu kata." + ANSI_RESET);
49              }
50              System.out.print("Masukkan kata awal (start word): ");
51              startWord = scanner.nextLine().toLowerCase();
52          }
53
54          String endWord;
55          System.out.print("Masukkan kata akhir (end word): ");
56          endWord = scanner.nextLine().toLowerCase();
57
58          // Validasi kata akhir
59          while (!dictionary.isValidWord(endWord) || !endWord.length() != startWord.length() || endWord.split("\s+").length > 1) {
60              if (!dictionary.isValidWord(endWord)) {
61                  System.out.println(ANSI_RED + "Kata akhir tidak valid." + ANSI_RESET);
62              } else if (endWord.length() != startWord.length()) {
63                  System.out.println(ANSI_RED + "Tolong masukkan kata yang memiliki panjang yang sama dengan kata awal." + ANSI_RESET);
64              }
65              System.out.print("Masukkan kata akhir (end word): ");
66              endWord = scanner.nextLine().toLowerCase();
67          }
68
69          System.out.print("Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): ");
70          int choice = scanner.nextInt();
71          scanner.nextLine(); // Membersihkan karakter newline
72
73          List<Object> path = null;
74          long startTime = System.currentTimeMillis();
75          switch (choice) {
76              case 1:
77                  path = wordLadderSolver.solveUCS(startWord, endWord, dictionary);
78                  break;
79              case 2:
80                  path = wordLadderSolver.solveGreedyBestFirstSearch(startWord, endWord, dictionary);
81                  break;
82              case 3:
83                  path = wordLadderSolver.solveAStar(startWord, endWord, dictionary);
84                  break;
85              default:
86                  System.out.println("Pilihan algoritma tidak valid.");
87                  return;
88          }
89          long endTime = System.currentTimeMillis();
90
91          @SuppressWarnings("unchecked")
92          List<String> pathList = (List<String>) path.get(0);
93          int exploredNodes = (Int) path.get(1);
94
95          Output.printWordLadderCLI(pathList);
96          Output.printStepCountCLI(exploredNodes); // Kurangi 1 karena jumlah langkah sama dengan jumlah node dikunjungi minus 1
97          Output.printExecutionTimeCLI(startTime, endTime);
98
99          NextOrOut();
100
101      public static void NextOrOut(){
102          System.out.println("Apakah Anda ingin keluar? (Y/N): ");
103          Scanner scanner = new Scanner(System.in);
104          String exitChoice = scanner.nextLine().toUpperCase();
105          if (exitChoice.equals("Y")) {
106              System.out.println(ANSI_GREEN+"Terima kasih telah menggunakan program Word Ladder Solver!" + ANSI_RESET);
107              return;
108          } else if (exitChoice.equals("N")) {
109              runCLI();
110          } else {
111              System.out.println(ANSI_RED+"Pilihan tidak valid." + ANSI_RESET);
112          }
113      }
114
115      public static void RunGUI() { // Ubah "RunGUI()" menjadi "RunGUI()"
116          SwingUtilities.invokeLater(() -> {
117              RunGUI gui = new RunGUI();
118              gui.setVisible(true);
119          });
120      }
121  }
122 }
```

### BAB III

#### PENGUJIAN

No	startWord-endWord	Output
1.	[UCS]snake-world	<pre>Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path: snake snare scare scars sears seals weals weald woald world Banyak node yang dikunjungi: 3794 Waktu Eksekusi: 25,69200000 detik</pre>
2.	[GBFS]snake-world	<pre>Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path: snake isnare stale shale whale while white whits waists walls walls wells weals weald woald world Banyak node yang dikunjungi: 48 Waktu Eksekusi: 0,41500000 detik</pre>
3.	[A*]snake-world	<pre>Masukkan kata awal (start word): snake Masukkan kata akhir (end word): world Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path: snake shake shame shams seamp seals weals weald woald world Banyak node yang dikunjungi: 359 Waktu Eksekusi: 1,46800000 detik</pre>
4.	[UCS]love-your	<pre>Masukkan kata awal (start word): love Masukkan kata akhir (end word): your Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path: love lone load loud lour your Banyak node yang dikunjungi: 2220 Waktu Eksekusi: 14,55500000 detik</pre>
5.	[GBFS]love-your	<pre>Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path: love move move rouve rouxe doux dour your Banyak node yang dikunjungi: 8 Waktu Eksekusi: 0,06500000 detik</pre>
6.	[A*]love-your	<pre>Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path: love lose lost lost lour lour your Banyak node yang dikunjungi: 61 Waktu Eksekusi: 0,50400000 detik</pre>

7.	[UCS]head-tail	<pre>Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 1034 Waktu Eksekusi: 6,97500000 detik</pre>
8.	[GBFS]head-tail	<pre>Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 5 Waktu Eksekusi: 0,03300000 detik</pre>
9.	[A*]head-tail	<pre>Masukkan kata awal (start word): head Masukkan kata akhir (end word): tail Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  head heal heil hail tail Banyak node yang dikunjungi: 6 Waktu Eksekusi: 0,04600000 detik</pre>
10.	[UCS]folder-family	<pre>Masukkan kata awal (start word): folder Masukkan kata akhir (end word): family Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  folder bolder balder balker barker darker darkey darkly dankly lankly lanely lamely gamedy gamily family Banyak node yang dikunjungi: 7783 Waktu Eksekusi: 59,46500000 detik</pre>
11.	[GBFS]folder-family	<pre>lamely gamedy gamily family Banyak node yang dikunjungi: 97 Waktu Eksekusi: 0,82000000 detik</pre>

12.	[A*]folder-family	<pre>Masukkan kata awal (start word): folder Masukkan kata akhir (end word): family Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  folder bolder balder balker barker darker darkey darkly lankly lanely lamely gamely gamily family Banyak node yang dikunjungi: 4282 Waktu Eksekusi: 22,57100000 detik</pre>	
13.	[UCS]zero-five	<pre>Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 843 Waktu Eksekusi: 5,53200000 detik</pre>	
14.	[GBFS]zero-five	<pre>Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 6 Waktu Eksekusi: 0,06600000 detik</pre>	
15.	[A*]zero-five	<pre>Masukkan kata awal (start word): zero Masukkan kata akhir (end word): five Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Word Ladder Path:  zero cero cere cire fire five Banyak node yang dikunjungi: 13 Waktu Eksekusi: 0,14200000 detik</pre>	
16.	[UCS]warfare-abettal	<pre>Masukkan kata awal (start word): warfare Masukkan kata akhir (end word): abettal Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 1  Path tidak ditemukan..  Banyak node yang dikunjungi: 2 Waktu Eksekusi: 0,01600000 detik</pre>	

17.	[GBFS]warfare-abettal	<pre>Masukkan kata awal (start word): warfare Masukkan kata akhir (end word): abettal Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 2  Path tidak ditemukan..  Banyak node yang dikunjungi: 3 Waktu Eksekusi: 0,02200000 detik</pre>
18.	[A*]warfare-abettal	<pre>Masukkan kata awal (start word): warfare Masukkan kata akhir (end word): abettal Pilih algoritma (1 = UCS, 2 = Greedy Best First Search, 3 = A*): 3  Path tidak ditemukan..  Banyak node yang dikunjungi: 2 Waktu Eksekusi: 0,00600000 detik</pre>

Tabel 3.1 pengujian program UCS

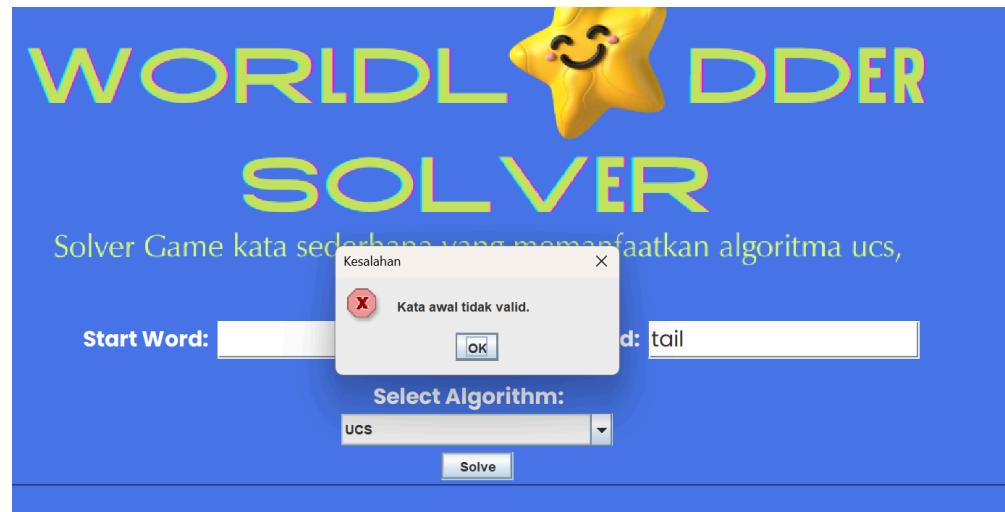
## BAB VI BONUS

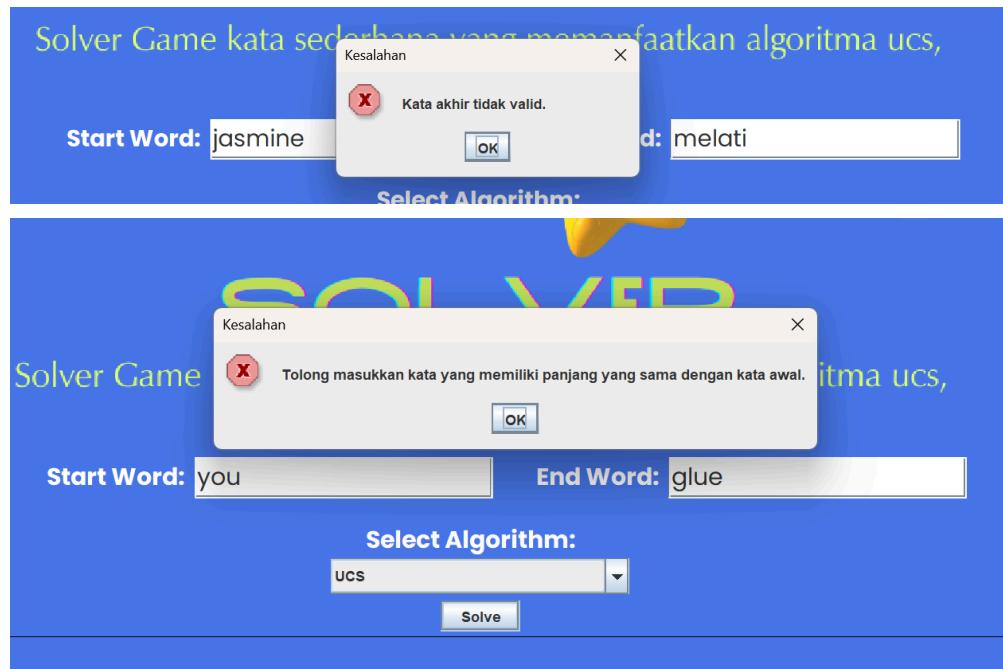
GUI dalam bonus ini menggunakan Java Swing. Pemilihan Java Swing sendiri karena implementasinya lebih mudah dalam program java. Implementasi bonus dibangun dalam kelas RunGUI() yang nantinya dihubungkan ke main.



Berikut fitur-fitur yang terdapat dalam bonus :

1. Fitur input masukan startWord dan endWord yang berfungsi untuk menerima masukan 2 kata yang akan dicari pathnya.
2. Fitur select algorithm berupa dropdown pilihan algoritma UCS GBFS, dan A\*
3. Button solve yang diklik setelah memasukkan kedua input
4. Pop up input tidak valid meliputi startWord dan endWord yang tidak di kamus beserta startWord dan endWord yang panjangnya sama.





- Box Result yang menampilkan result dari pencarian yang telah dilakukan, box ini bisa discroll jika pencarian melebihi satu. Jika tidak ingin men-scroll pengguna dapat menghapus manual result sebelumnya

Penjelasan implementasi kode :

Dalam konstruktor RunGUI(), kita mengatur properti dasar untuk jendela GUI. Ini termasuk mengatur ukuran jendela, mengatur tata letak (layout), dan menambahkan komponen-komponen GUI seperti tombol, teks, dan area teks. Selain itu, kita menambahkan penanganan acara (event handling) untuk tombol “Solve” agar dapat mengeksekusi algoritma yang sesuai berdasarkan pilihan pengguna. Hasil dari algoritma ditampilkan di area teks yang disediakan.

Berikut penjelasan algoritma yang terdapat dalam Method solveWordLadder() :

- Validasi Input: Metode ini pertama-tama memvalidasi masukan pengguna, seperti memeriksa apakah kata-kata awal dan akhir valid, memiliki panjang yang sama, dan hanya terdiri dari satu kata
- Pemanggilan Algoritma: Berdasarkan pilihan algoritma yang dipilih oleh pengguna dari algorithmComboBox, metode ini memanggil metode yang sesuai untuk menyelesaikan masalah Word Ladder.
- Penanganan Output: Setelah mendapatkan jalur dan informasi lainnya dari algoritma, metode ini menampilkan hasilnya di outputTextArea. Ini termasuk menampilkan jalur kata, jumlah node yang dieksplorasi, dan waktu eksekusi.

Berikut implementasi kode(tangkapan layar hanya sebagian) :



```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import javax.swing.border.EmptyBorder;
6 import java.util.List;
7
8 public class RunGUI extends JFrame {
9     private static final String ANSI_PURPLE = "\u001B[35m";
10    private static final String ANSI_RESET = "\u001B[0m";
11    private JTextField startTextField;
12    private JTextField endTextField;
13    private JComboBox<String> algorithmComboBox;
14    private JTextArea outputTextArea;
15
16    public RunGUI() {
17        // setTitle("Word Ladder Solver");
18        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19        setSize(600, 400);
20        setLayout(new BorderLayout());
21        // Set background color
22        Color backgroundColor = Color.decode("#4877E9");
23        getContentPane().setBackground(backgroundColor);
24
25        JLabel logoLabel = new JLabel(new ImageIcon("assets/logo.png"));
26        add(logoLabel, BorderLayout.WEST);
27
28        JPanel inputPanel = new JPanel(new GridBagLayout());
29        GridBagConstraints gbc = new GridBagConstraints();
30        gbc.gridx = 0;
31        gbc.gridy = 0;
32        gbc.anchor = GridBagConstraints.CENTER;
33        gbc.insets = new Insets(0, 20, 0, 0); // Padding bottom 20
34        inputPanel.setBackground(Color.decode("#4877E9"));
35
36        JLabel startLabel = new JLabel("Start Word: ");
37        startLabel.setFont(new Font("Poppins", Font.BOLD, 20));
38        startLabel.setForeground(Color.WHITE);
39        gbc.gridy = 1;
40        gbc.insets = new Insets(0, 0, 5, 0); // Padding bottom 10
41        inputPanel.add(startLabel, gbc);
42
43        startTextField = new JTextField();
44        startTextField.setPreferredSize(new Dimension(250, 35));
45        startTextField.setFont(new Font("Poppins", Font.PLAIN, 20));
46        gbc.gridx = 1;
47        gbc.insets = new Insets(0, 10, 10, 10); // Padding left 20, padding bottom 10
48        inputPanel.add(startTextField, gbc);

```



```

1  private void solveWordLadder() {
2      outputTextArea.setText("");
3      String startWord = startTextField.getText().toLowerCase();
4      String endWord = endTextField.getText().toLowerCase();
5      int choice = algorithmComboBox.getSelectedIndex() + 1;
6
7      English dictionary = new English();
8      if (!dictionary.isValidWord(startWord)) {
9          // Kata tidak valid, tampilkan pesan kesalahan
10         SwingUtilities.invokeLater(() -> {
11             JOptionPane.showMessageDialog(this, "Kata awal tidak valid.", "Kesalahan", JOptionPane.ERROR_MESSAGE);
12         });
13         return; // Keluar dari metode jika kata tidak valid
14     } else if (startWord.split("\\s+").length > 1) {
15         // Kata tidak valid, tampilkan pesan kesalahan
16         SwingUtilities.invokeLater(() -> {
17             JOptionPane.showMessageDialog(this, "Tolong masukkan hanya satu kata.", "Kesalahan", JOptionPane.ERROR_MESSAGE);
18         });
19         return; // Keluar dari metode jika kata tidak valid
20     }
21
22     if (!dictionary.isValidWord(endWord)) {
23         // Kata tidak va

```

## LAMPIRAN

Repository: [https://github.com/mlatia/Tucil3\\_13522035/](https://github.com/mlatia/Tucil3_13522035/)

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma UCS	✓	
3. Solusi yang diberikan pada Algoritma UCS optimal.	✓	
4. Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma Greedy Best First Search	✓	
5. Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma A*	✓	
6. Solusi yang diberikan pada algoritma A* optimal	✓	
<b>7. [Bonus]: Program memiliki tampilan GUI</b>	✓	