

LAPORAN TUGAS KECIL 3 IF2211

STRATEGI ALGORITMA

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



Dosen Pengampu: Dr. Nur Ulfa Maulidevi, S.T, M.Sc

Disusun oleh:

Panji Sri Kuncara Wisma

(13522028)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	1
PENGECEKAN PROGRAM.....	2
DESKRIPSI MASALAH.....	3
A. Analisis Algoritma UCS, Greedy Best First Search, dan A* untuk Menyelesaikan Permainan Word Ladder.....	4
B. Penjelasan Method, Class, dan Source Code Implementasi.....	12
C. Test atau Pengujian Program.....	24
D. Analisis Hasil Pengujian.....	28
E. Asumsi-Asumsi.....	28
F. Repository.....	28

PENGECEKAN PROGRAM

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 program optimal.	✓	
4. Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma <i>Greedy Best First Search</i>	✓	
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	✓	
7. [Bonus]: Program memiliki tampilan GUI		✓

Tabel 1. Tabel Pengecekan Program

DESKRIPSI MASALAH

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.

A. Analisis Algoritma UCS, *Greedy Best First Search*, dan A* untuk Menyelesaikan Permainan Word Ladder

1. Analisis Umum

Untuk menemukan solusi optimal dari permainan Word Ladder dengan algoritma penentuan rute (*Route/Path Planning*), kita perlu menemukan unsur-unsur dari permainan Word Ladder yang bisa dipandang sebagai komponen algoritma penentuan rute (UCS, *Greedy Best First Search*, atau A*). Berikut adalah poin-poin penjelasan mengenai komponen yang dimaksud:

- Simpul secara umum

Kata-kata dalam bahasa Inggris akan dianggap sebagai simpul. Suatu kata akan dianggap valid apabila memenuhi kriteria berikut:

- Kata yang dimaksud memang benar-benar ada pada <https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt>
- Kata tidak mengandung angka atau simbol selain huruf abjad (a - z)

- Simpul ekspan

Misalkan A adalah suatu kata yang valid. Apabila suatu kata A adalah kata yang ingin kita cari kata-kata *intermediate* nya atau kata-kata yang memiliki selisih satu huruf dengan kata A tersebut, maka A adalah kata yang akan kita anggap sebagai simpul ekspan pada permainan Words Ladder.

- Simpul hidup

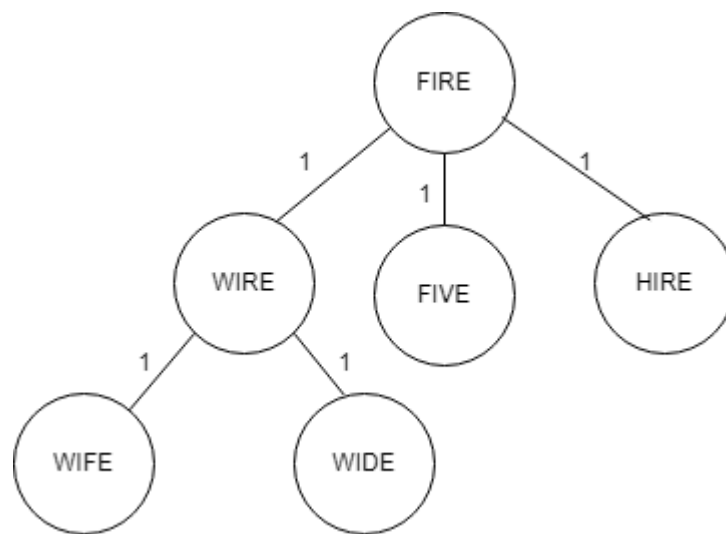
Simpul hidup adalah kata-kata *intermediate* atau kata-kata valid yang memiliki selisih satu huruf dengan kata tertentu yang merupakan simpul ekspan.

- *Root* atau *Start* dan *Goal*

Word start atau kata mulai atau kata yang pertama kali diekspan akan dianggap sebagai *Root*. Kata yang menjadi tujuan akhir dari permainan Word Ladder akan dianggap sebagai *Goal*.

2. Analisis Algoritma UCS atau *Uniform Cost Search*

Solusi optimal pada algoritma UCS adalah solusi dengan total *cost* terkecil. Nilai *cost* yang saya gunakan dalam implementasi adalah banyak huruf yang berubah dari suatu kata ke kata yang lain. Oleh karena itu, *cost* dari dua simpul yang saling berdekatan atau hanya terhubung oleh satu sisi adalah satu. Berdasarkan salindia kuliah, $g(n)$ adalah total *cost* dari *root* menuju simpul n . Dengan kata lain, $g(n)$ adalah 1 dikali dengan banyak lompatan yang dilakukan atau sisi yang dilewati dari *root* menuju n . Berikut adalah ilustrasinya:



Gambar 1. Ilustrasi UCS Sederhana
(Sumber: dokumentasi pribadi penulis)

Keterangan dengan asumsi kata dimulai dari FIRE dan *goal* adalah WIFE:

- $g(\text{WIRE}) = g(\text{FIVE}) = g(\text{HIRE}) = 1$
- $g(\text{WIFE}) = g(\text{WIDE}) = 2$

Berikut adalah langkah-langkah utama yang saya implementasikan untuk menemukan solusi dari permainan Words Ladder dengan algoritma UCS:

1. Hal yang pertama kali dilakukan adalah menginisiasi nilai *cost* menjadi nol
2. Melakukan ekspansi terhadap simpul *root* untuk mendapatkan simpul hidup atau kata-kata yang memiliki perbedaan satu huruf dengan kata pada simpul *root*. Pencarian dilakukan pada referensi *dictionary* utama yang sudah saya jelaskan pada bagian analisis umum.

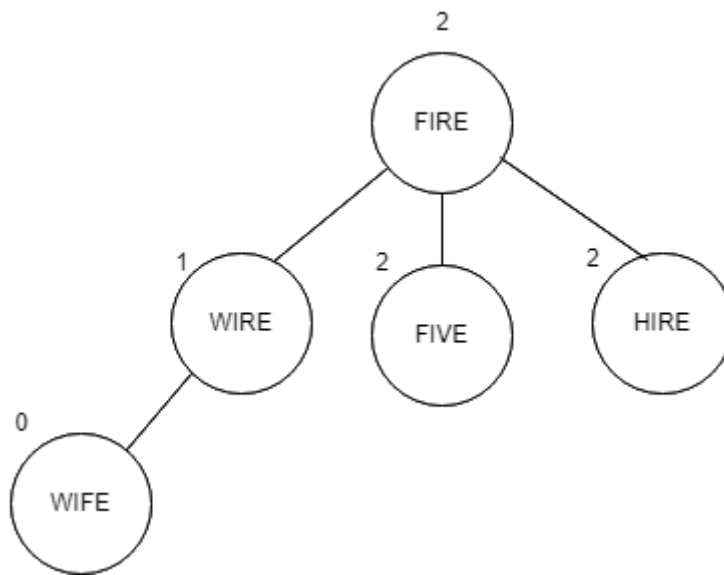
3. Kata-kata yang didapatkan pada langkah nomor 2 disimpan pada suatu list yang menyimpan pasangan String dan Integer. Nilai Integer yang dimaksud disini adalah total *cost* yang dibutuhkan untuk mencapai kata-kata tersebut dari *root*. Dalam kasus ini *cost* adalah nilai *cost* sekarang ditambah dengan 1.
4. Kata dengan nilai *cost* terkecil akan menjadi kata yang diekspan berikutnya.
5. Nilai *cost* diinisiasi menjadi nilai *cost* dari kata yang dimaksud pada langkah 4.
6. Apabila kata yang dimaksud pada langkah 4 belum sama dengan *goal*, maka langkah 2, 3, 4, dan 5 akan diulang kembali dengan nilai *cost* yang sudah diinisiasi dengan nilai baru pada langkah 5. Yang menjadi perbedaan adalah pada langkah 2 berikutnya, kata yang diekspan bukanlah *root* melainkan kata dengan nilai *cost* terkecil pada iterasi sebelumnya seperti yang saya uraikan pada langkah 4. *Path* atau jalur yang dilewati untuk mencapai suatu simpul juga diingat agar bisa dimanfaatkan pada langkah 7.
7. Apabila kata yang dimaksud pada langkah 4 sudah sama dengan *goal*, maka jalur atau langkah - langkah yang perlu dilewati dari *root* untuk menjadi *goal* akan ditampilkan bersama dengan waktu pencarian dan jumlah node yang dilewati.

3. Analisis Algoritma *Greedy Best First Search*

Algoritma *Greedy Best First Search* adalah algoritma yang memanfaatkan nilai *heuristic* untuk mendapatkan solusi yang optimal. Pada permainan Words Ladder, nilai *heuristic* atau $h(n)$ yang digunakan adalah banyaknya perbedaan huruf untuk setiap indeks antara kata pada simpul n dengan kata pada *goal*. Perlu diperhatikan, bahwa lokasi atau indeks dari huruf pada suatu kata sangatlah penting. Huruf yang dibandingkan adalah huruf dengan indeks atau lokasi yang sama. Kita akan menggunakan kata FIVE dan WIFE sebagai contoh. Dalam implementasi saya, kedua kata tersebut memiliki perbedaan dua huruf. Huruf F

pada FIVE dan F pada WIFE tidak dianggap sama karena memiliki lokasi atau indeks yang berbeda.

Nilai *heuristic* atau $h(n)$ dimanfaatkan untuk mengetahui simpul apa yang akan diekspan pada setiap iterasi. Nilai *heuristic* yang saya uraikan sebelumnya seharusnya sudah sesuai dengan salinda kuliah yang menyatakan bahwa $h(n)$ adalah *estimates of cost from n to goal*. Berikut adalah ilustrasinya:



Gambar 2. Ilustrasi *Greedy Best First Search* Sederhana
(Sumber: dokumentasi pribadi penulis)

Keterangan dengan asumsi kata dimulai dari FIRE dan *goal* adalah WIFE:

- $h(\text{WIRE}) = 1$
- $h(\text{FIVE}) = h(\text{HIRE}) = 2$

Berikut adalah langkah-langkah utama yang saya implementasikan untuk menemukan solusi dari permainan Words Ladder dengan algoritma *Greedy Best First Search*:

1. Hal yang pertama dilakukan adalah melakukan ekspansi terhadap simpul *root* untuk mendapatkan simpul hidup atau kata-kata yang memiliki perbedaan satu huruf dengan kata pada simpul *root*. Pencarian dilakukan

pada referensi *dictionary* utama yang sudah saya jelaskan pada bagian analisis umum.

2. Kata-kata yang didapatkan pada langkah nomor 1 disimpan pada suatu list yang menyimpan pasangan String dan Integer. Nilai Integer yang dimaksud disini adalah nilai *heuristic* atau $h(n)$ dari simpul tersebut yang merupakan banyaknya perbedaan huruf untuk setiap indeks antara kata pada simpul tersebut dengan kata pada *goal*.
3. Kata dengan nilai *heuristic* terkecil akan menjadi kata yang di ekspan berikutnya.
4. Apabila kata yang dimaksud pada langkah 3 belum sama dengan *goal*, maka langkah 1, 2, dan 3 akan diulang kembali. Yang menjadi perbedaan adalah pada langkah 1 berikutnya, kata yang diekspan bukanlah *root* melainkan kata dengan nilai *heuristic* terkecil pada iterasi sebelumnya seperti yang saya uraikan pada langkah 3. *Path* atau jalur yang dilewati untuk mencapai suatu simpul juga diingat agar bisa dimanfaatkan pada langkah 5.
5. Apabila kata yang dimaksud pada langkah 3 sudah sama dengan *goal*, maka jalur atau langkah - langkah yang perlu dilewati dari *root* untuk menjadi *goal* akan ditampilkan bersama dengan waktu pencarian dan jumlah node yang dilewati.

3. Analisis Algoritma A*

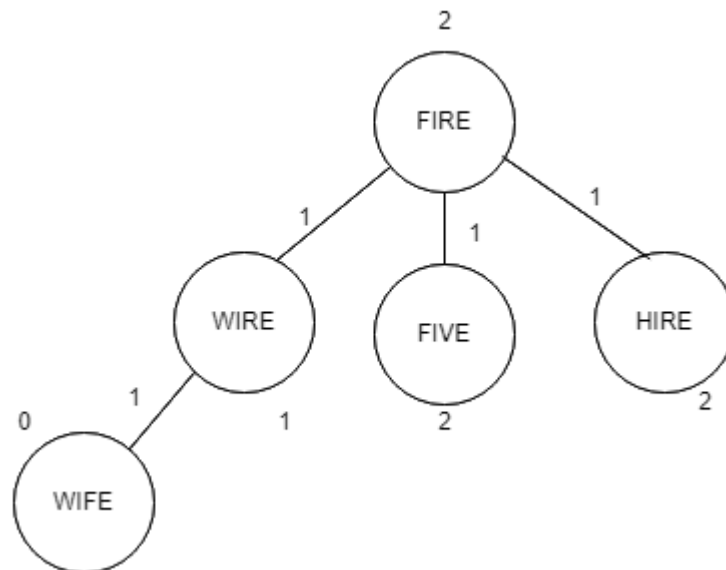
Secara sederhana, algoritma A* adalah kombinasi dari algoritma UCS dan *Greedy Best First Search*. Pada algoritma tersebut, nilai yang menjadi pertimbangan untuk menentukan suatu simpul yang akan diekspan adalah $f(n)$ yang merupakan penjumlahan dari $h(n)$ atau nilai *heuristic* dalam konteks *Greedy Best First Search* dan $g(n)$ atau *cost* dalam konteks UCS dari simpul tersebut. Penjelasan mengenai $h(n)$ dan $g(n)$ sudah saya uraikan pada analisis algoritma UCS dan analisis algoritma *Greedy Best First Search* sehingga tidak akan saya ulang lagi.

Algoritma A* yang digunakan untuk menemukan solusi dari permainan Words Ladder bersifat *admissible*. Berdasarkan salinda kuliah sebuah *heuristic*

$h(n)$ adalah **admissible** jika untuk setiap node n , $h(n) \leq h^*(n)$, dimana $h^*(n)$ adalah *cost* yang sebenarnya untuk mencapai simpul tujuan dari n . Untuk melakukan pembuktian, kita ambil nilai $h(n)$ dan $h^*(n)$ paling minimum dalam kasus permainan Words Ladder. Dengan mengabaikan nilai *heuristic* dari *goal* terhadap dirinya sendiri, maka kita dapat mengetahui jika nilai $h(n)$ terkecil adalah satu. Dengan kata lain, jumlah huruf yang berbeda dari suatu kata pada suatu simpul dengan suatu kata dengan *goal* paling sedikit adalah satu.

Kita juga mengetahui jika nilai $h^*(n)$ yang paling minimum adalah satu. Hal itu karena pada permainan Words Ladder, jarak dua simpul terdekat yang dihubungkan oleh satu sisi adalah bernilai satu. Pada permainan tersebut, nilai tersebut selalu sama untuk setiap dua simpul yang dihubungkan oleh satu sisi sehingga *cost* dari *root* hingga *goal* yang paling kecil adalah satu.

Berdasarkan uraian di atas, dapat dilihat bahwa nilai $h(n)$ paling minimum itu selalu kurang dari sama dengan nilai $h^*(n)$ paling minimum. Oleh karena itu, algoritma A^* yang saya implementasikan adalah bersifat **admissible** dengan terbuktinya $h(n) \leq h^*(n)$ untuk setiap simpul n .



Gambar 3. Ilustrasi A^* Sederhana
(Sumber: dokumentasi pribadi penulis)

Keterangan dengan asumsi kata dimulai dari FIRE dan *goal* adalah WIFE:

- $f(\text{WIRE}) = h(\text{WIRE}) + g(\text{WIRE}) = 1 + 1 = 2$

- $f(\text{FIVE}) = h(\text{FIVE}) + g(\text{FIVE}) = 1 + 2 = 3$
- $f(\text{HIRE}) = h(\text{HIRE}) + g(\text{HIRE}) = 1 + 2 = 3$
-

Berikut adalah langkah-langkah utama yang saya implementasikan untuk menemukan solusi dari permainan Words Ladder dengan algoritma A^* :

1. Hal yang pertama kali dilakukan adalah menginisiasi nilai *cost* menjadi nol
2. Melakukan ekspansi terhadap simpul *root* untuk mendapatkan simpul hidup atau kata-kata yang memiliki perbedaan satu huruf dengan kata pada simpul *root*. Pencarian dilakukan pada referensi *dictionary* utama yang sudah saya jelaskan pada bagian analisis umum.
3. Kata-kata yang didapatkan pada langkah nomor 2 disimpan pada suatu list yang menyimpan pasangan String dan Integer. Nilai Integer yang dimaksud disini adalah $f(n)$ yang merupakan $g(n) + h(n)$. Dengan kata lain, nilai tersebut adalah hasil penjumlahan antara *cost* saat ini ditambah dengan 1 sebagai $g(n)$ ditambah dengan nilai *heuristic* sebagai $h(n)$ dari simpul yang dimaksud.
4. Kata dengan nilai $f(n)$ terkecil akan menjadi kata yang diekspansi berikutnya.
5. Nilai *cost* diinisiasi menjadi nilai *cost* dari kata yang dimaksud pada langkah 4.
6. Apabila kata yang dimaksud pada langkah 4 belum sama dengan *goal*, maka langkah 2, 3, 4, dan 5 akan diulang kembali dengan nilai *cost* yang sudah diinisiasi dengan nilai baru pada langkah 5. Yang menjadi perbedaan adalah pada langkah 2 berikutnya, kata yang diekspansi bukanlah *root* melainkan kata dengan nilai $f(n)$ terkecil pada iterasi sebelumnya seperti yang saya uraikan pada langkah 4. *Path* atau jalur yang dilewati untuk mencapai suatu simpul juga diingat agar bisa dimanfaatkan pada langkah 7.
7. Apabila kata yang dimaksud pada langkah 4 sudah sama dengan *goal*, maka jalur atau langkah - langkah yang perlu dilewati dari *root* untuk menjadi *goal* akan ditampilkan bersama dengan waktu pencarian dan jumlah node yang dilewati.

4. Analisis Mendalam Kasus Words Ladder

Pada Kasus Words Ladder, algoritma UCS memiliki perilaku yang sama dengan algoritma BFS. Pada kasus ini, urutan *node* yang dibangkitkan dan *path* yang dihasilkan oleh UCS itu sama dengan BFS. Hal itu dapat terjadi karena pada kasus Words Ladder, jarak antara dua *node* yang terhubung oleh satu sisi itu selalu sama yaitu satu. Oleh karena itu, *node-node* pada kedalaman x akan di bangkitkan terlebih dahulu sebelum *node-node* pada kedalaman $x+1$. Perilaku tersebut pada algoritma BFS.

Secara teoritis algoritma A* lebih efisien dibandingkan dengan algoritma UCS pada kasus Words Ladder. Hal itu dapat terjadi karena algoritma A* memanfaatkan fungsi *heuristic* untuk memperkirakan jarak (jumlah huruf yang berbeda dengan *goal*) dari *node* saat ini ke *node* tujuan. Fungsi *heuristic* tersebut akan membantu A* untuk fokus pada jalur yang lebih menjanjikan, sehingga mengurangi jumlah *node* yang perlu dijelajahi.

Secara teoritis algoritma *Greedy Best First Search* tidak menjamin solusi optimal secara global untuk persoalan Words Ladder. Hal ini karena ada kemungkinan algoritma *Greedy Best First Search* terjebak pada solusi optimum lokal. Oleh karena algoritma tersebut hanya menggunakan nilai *heuristic* atau $h(n)$ sebagai acuan, maka ada kemungkinan *Greedy Best First Search* akan memilih jalur yang lebih panjang karena memiliki nilai $h(n)$ yang rendah. Pada beberapa kasus mungkin saja untuk mendapatkan solusi optimal secara global, tapi tidak menjamin setiap solusi yang didapat adalah optimal secara global.

B. Penjelasan *Method*, *Class*, dan *Source Code* Implementasi

1. Worddiff.java

```
src > Worddiff.java > ...
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Worddiff {
5      public ArrayList<String> findWordDiff(String word, List<String> dictionary) {
6          ArrayList<String> result = new ArrayList<>();
7
8          for (String entry : dictionary) {
9              if (isOneLetterDifferent(word, entry)) {
10                 result.add(entry);
11             }
12         }
13
14         return result;
15     }
16
17     private boolean isOneLetterDifferent(String word1, String word2) {
18
19         if (word1.length() != word2.length()) {
20             return false;
21         }
22
23         int diffCount = 0;
24         for (int i = 0; i < word1.length(); i++) {
25             if (word1.charAt(i) != word2.charAt(i)) {
26                 diffCount++;
27             }
28             if (diffCount > 1) {
29                 return false;
30             }
31         }
32
33         return diffCount == 1;
34     }
35 }
36
```

Class :

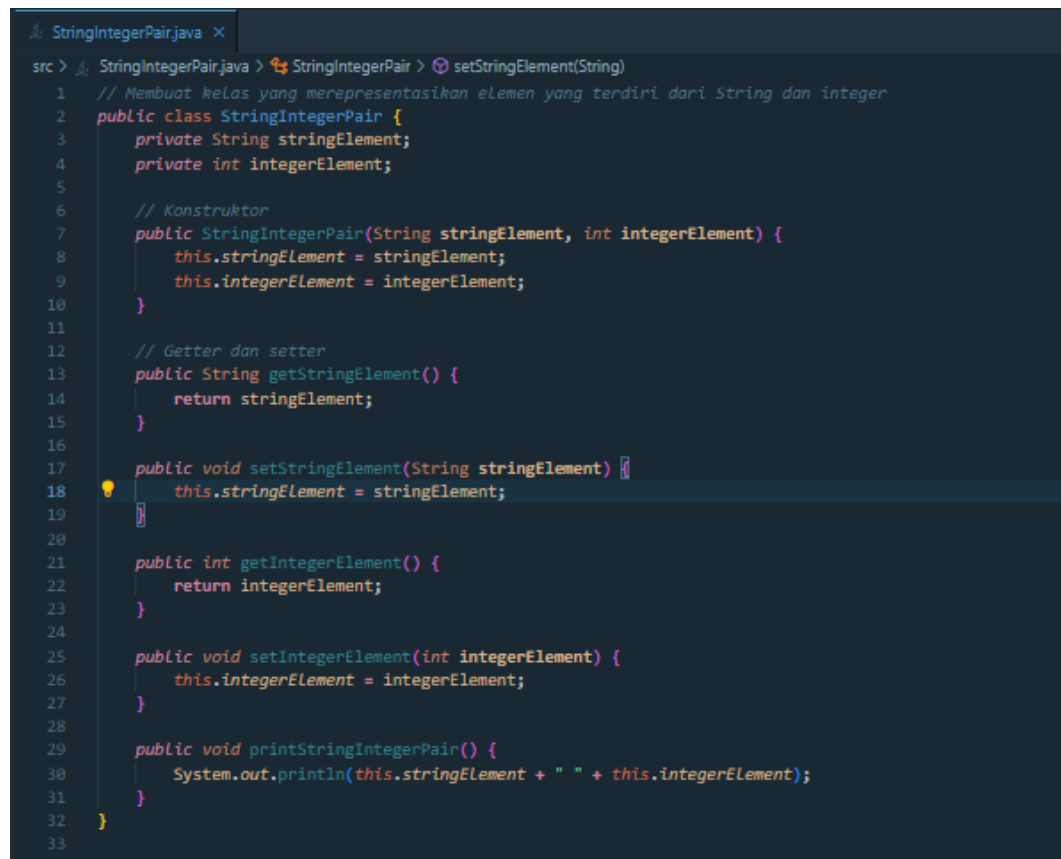
Worddiff adalah *class* yang digunakan untuk membantu untuk melakukan operasi terhadap *dictionary* yang disimpan dalam format `ArrayList<String>`

Method:

Nama <i>Method</i>	Penjelasan
findWordDiff	<i>Method</i> ini digunakan untuk mencari list kata-kata yang memiliki perbedaan satu huruf dengan parameter word pada dictionary dan mengembalikan list yang dihasilkan

isOneLetterDifferent	<i>Method</i> ini mengembalikan nilai true apabila parameter word1 dan word2 memiliki total perbedaan satu huruf sesuai konteks permainan Words Ladder
----------------------	--

2. StringIntegerPair.java



```

src > StringIntegerPair.java > StringIntegerPair > setStringElement(String)
1 // Membuat kelas yang merepresentasikan elemen yang terdiri dari String dan integer
2 public class StringIntegerPair {
3     private String stringElement;
4     private int integerElement;
5
6     // Konstruktor
7     public StringIntegerPair(String stringElement, int integerElement) {
8         this.stringElement = stringElement;
9         this.integerElement = integerElement;
10    }
11
12    // Getter dan setter
13    public String getStringElement() {
14        return stringElement;
15    }
16
17    public void setStringElement(String stringElement) {
18        this.stringElement = stringElement;
19    }
20
21    public int getIntegerElement() {
22        return integerElement;
23    }
24
25    public void setIntegerElement(int integerElement) {
26        this.integerElement = integerElement;
27    }
28
29    public void printStringIntegerPair() {
30        System.out.println(this.stringElement + " " + this.integerElement);
31    }
32 }
33
34

```

Class :

StringIntegerPair adalah *class* yang nantinya akan digunakan sebagai format penyimpanan untuk menyimpan kata dalam tipe data String dan nilai Integer yang bisa berupa $g(n)$, $h(n)$, maupun $f(n)$.

Method:

Nama <i>Method</i>	Penjelasan
StringIntegerPair	Konstruktor
getStringElement	Getter kata
setStringElement	Setter kata

getIntegerElement	Getter nilai Integer
setIntegerElement	Setter nilai Integer
printStringIntegerPair	<i>Method</i> untuk menampilkan kata dan nilai Integer pada terminal

3. SearchAlgorithm.java

```

SearchAlgorithm.java x
src > SearchAlgorithm.java > SearchAlgorithm > word_goal
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public abstract class SearchAlgorithm {
5      protected String word_start;
6      protected String word_goal;
7      protected List<String> dictionary;
8
9      public SearchAlgorithm(String word_start, String word_goal, List<String> dictionary) {
10         this.word_start = word_start.toLowerCase();
11         this.word_goal = word_goal.toLowerCase();
12         this.dictionary = dictionary;
13     }
14
15     public void printWordStartGoal() {
16         System.out.println("Kata Mulai: " + word_start.toUpperCase());
17         System.out.println("Kata Tujuan: " + word_goal.toUpperCase());
18     }
19
20     public abstract void algorithm();
21
22     public abstract void insertInOrder(ArrayList<StringIntegerPair> list, StringIntegerPair newItem);
23
24     public abstract int countLetterDifference(String word1, String word2);
25 }
26

```

Class:

SearchAlgorithm adalah *super class* atau kelas induk dari kelas UCS, A_Star, dan Greedy yang akan menjadi algoritma utama dalam pencarian solusi.

Method:

Nama <i>Method</i>	Penjelasan
SearchAlgorithm	Konstruktor
printWordStartGoal	<i>Method</i> untuk menampilkan kata mulai dan kata tujuan pada terminal
algorithm	Algoritma pencarian utama yang implementasinya ada pada kelas anak
insertInOrder	<i>Method</i> yang digunakan untuk menambahkan nilai dari parameter newItem kedalam list agar tetap terurut membesar berdasarkan nilai

	Integernya.
countLetterDifference	<i>Method</i> ini mengembalikan jumlah perbedaan huruf antara word1 dan word2 dalam integer

4. UCS.java

```

src > UCS.java > UCS > algorithm()
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class UCS extends SearchAlgorithm {
5
6      public UCS(String word_start, String word_goal, List<String> dictionary) {
7          super(word_start, word_goal, dictionary);
8      }
9
10     public void algorithm() {
11         long startTime = System.currentTimeMillis();
12
13         // Uniform Cost Search
14         String currentWord = word_start;
15
16         // Simpul buat di ekspan
17         ArrayList<StringIntegerPair> nodeToExpan = new ArrayList<>();
18
19         // Simpul Simpul ekspansi
20         ArrayList<StringIntegerPair> nodeExpansion = new ArrayList<>();
21
22         nodeToExpan.add(new StringIntegerPair(currentWord, integerElement:0));
23
24         Worddiff wd = new Worddiff();
25
26         int cost = 0;
27
28         while (!currentWord.equals(word_goal)) {
29             ArrayList<String> temp = wd.findWordDiff(currentWord, dictionary);
30
31             for (int i = 0; i < temp.size(); i++) {
32                 List<String> firstWordsList = new ArrayList<>();
33
34                 // kata kata simpul yang pernah diekspan
35                 for (StringIntegerPair element : nodeToExpan) {
36                     String[] firstWords = element.getStringElement().split(regex: " ");
37                     firstWordsList.add(firstWords[0]);
38                 }
39
40                 // kalau belum pernah di ekspan baru boleh jadi calon untuk di ekspan
41                 if (!firstWordsList.contains(temp.get(i))) {
42                     StringIntegerPair newNode = new StringIntegerPair(
43                         temp.get(i) + " " + nodeToExpan.get(nodeToExpan.size() - 1).getStringElement(), cost + 1);
44                     insertInOrder(nodeExpansion, newNode);
45                 }
46             }
47         }

```



```

48         if (nodeExpansion.size() == 0) {
49             System.out.print(s+"\n");
50             System.out.println(x:"");
51             System.out.println(x:"");
52             System.out.println(x:"Tidak ada solusi sesuai referensi kamus dictionary.txt milik oracle");
53             System.out.println(x:"");
54             System.out.println(x:"");
55             System.out.println(x:"");
56             System.exit(status:0);
57             break;
58         }
59
60         StringIntegerPair min = nodeExpansion.remove(index:0);
61         nodeToExpan.add(min);
62
63         currentWord = min.getStringElement().split(regex:" ")[0];
64         cost = min.getIntegerElement();
65
66         Long stopTime = System.currentTimeMillis();
67         Long elapsedTime = stopTime - startTime;
68
69         //buat ngeprint hasilnya
70         String path = nodeToExpan.get(nodeToExpan.size() - 1).getStringElement();
71
72         System.out.print(s+"\n");
73         System.out.println(x:"Solusi: ");
74         System.out.println(x:"-----");
75         for (int i = path.split(regex:" ").length - 1; i >= 0; i--) {
76             System.out.println(path.split(regex:" ")[i].toUpperCase());
77         }
78         System.out.println(x:"-----\n");
79
80         System.out.println("Banyak node yang dikunjungi: " + nodeToExpan.size());
81         System.out.println("Waktu (ms): " + elapsedTime);
82
83
84         public void insertInOrder(ArrayList<StringIntegerPair> list, StringIntegerPair newItem) {
85             int index = 0;
86             // Masuknya biar terurut
87             while (index < list.size() && list.get(index).getIntegerElement() < newItem.getIntegerElement()) {
88                 index++;
89             }
90             list.add(index, newItem);
91         }
92
93         //UCS gak perlu method ini sebenarnya
94         public int countLetterDifference(String word1, String word2) {
95             return 0;
96         }
97     }
98

```

Class:

UCS adalah kelas utama yang digunakan untuk melakukan pencarian dengan menggunakan algoritma UCS.

Method:

Nama Method	Penjelasan
UCS	Konstruktor
algorithm	Algoritma pencarian utama UCS
insertInOrder	Method yang digunakan untuk menambahkan nilai dari parameter newItem kedalam list agar tetap terurut membesar berdasarkan nilai Integernya.

countLetterDifference	<i>Method</i> ini tidak digunakan pada UCS, jadi hanya mengembalikan nilai integer 0
-----------------------	--

5. Greedy.java

```

src > 1 Greedy.java > 2 Greedy > 3 algorithm()
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Greedy extends SearchAlgorithm {
5
6     public Greedy(String word_start, String word_goal, List<String> dictionary) {
7         super(word_start, word_goal, dictionary);
8     }
9
10    public void algorithm() {
11        long startTime = System.currentTimeMillis();
12
13        // // Greedy Best First Search
14        String currentWord = word_start;
15
16        // // Simpul buat di ekspan
17        ArrayList<StringIntegerPair> nodeToExpan = new ArrayList<>();
18
19        // // Simpul Simpul ekspansi
20        ArrayList<StringIntegerPair> nodeExpansion = new ArrayList<>();
21        nodeExpansion.add(new StringIntegerPair(currentWord, integerElement:0));
22
23        WordDiff wd = new WordDiff();
24
25        int heuristic = 0;
26
27        while (!currentWord.equals(word_goal)) {
28            ArrayList<String> temp = wd.findWordDiff(currentWord, dictionary);
29
30
31            for (int i = 0; i < temp.size(); i++) {
32                List<String> firstWordsList = new ArrayList<>();
33
34                // kata kata simpul yang pernah diekspan
35                for (StringIntegerPair element : nodeToExpan) {
36                    String[] firstWords = element.getStringElement().split(regex:" ");
37                    firstWordsList.add(firstWords[0]);
38                }
39
40                // kalau belum pernah di ekspan baru boleh jadi calon untuk di ekspan
41                if (!firstWordsList.contains(temp.get(i))) {
42                    //ini nilai heuristic nya, yaitu kata ini sama goal itu beda berapa huruf
43                    heuristic = countLetterDifference(temp.get(i), word_goal);
44
45                    StringIntegerPair newNode = new StringIntegerPair(
46                        temp.get(i) + " " + nodeToExpan.get(nodeToExpan.size() - 1).getStringElement(), heuristic);
47                    insertInOrder(nodeExpansion, newNode);
48                }
49            }
50        }
51    }

```

```

src > Greedy.java > Greedy > countLetterDifference(String, String)
4 public class Greedy extends SearchAlgorithm {
10     public void algorithm() {
11
12         if (nodeExpansion.size() == 0) {
13             System.out.print(s+"\n");
14             System.out.println(x:"");
15             System.out.println(x:"");
16             System.out.println(x:"Tidak ada solusi sesuai referensi kamus dictionary.txt milik oracle");
17             System.out.println(x:"");
18             System.out.println(x:"");
19             System.exit(status:0);
20             break;
21         }
22
23         StringIntegerPair min = nodeExpansion.remove(index:0);
24         nodeToExpan.add(min);
25
26         currentWord = min.getStringElement().split(regex:" ")[0];
27     }
28
29     long stopTime = System.currentTimeMillis();
30     long elapsedTime = stopTime - startTime;
31
32     //buat ngeprint hasilnya
33     String path = nodeToExpan.get(nodeToExpan.size() - 1).getStringElement();
34
35     System.out.print(s+"\n");
36     System.out.println(x:"Solusi: ");
37     System.out.println(x:"-----");
38     for (int i = path.split(regex:" ").length - 1; i >= 0; i--) {
39         System.out.println(path.split(regex:" ")[i].toUpperCase());
40     }
41     System.out.println(x:"-----\n");
42
43     System.out.println("Banyak node yang dikunjungi: " + nodeToExpan.size());
44     System.out.println("Waktu (ms): " + elapsedTime);
45 }
46
47 public void insertInOrder(ArrayList<StringIntegerPair> list, StringIntegerPair newItem) {
48     int index = 0;
49     // Masuknya biar terurut
50     while (index < list.size() && list.get(index).getIntegerElement() < newItem.getIntegerElement()) {
51         index++;
52     }
53     list.add(index, newItem);
54 }
55
56 public int countLetterDifference(String word1, String word2) {
57     int difference = 0;
58     for (int i = 0; i < word1.length(); i++) {
59         if (word1.charAt(i) != word2.charAt(i)) {
60             difference++;
61         }
62     }
63     return difference;
64 }
65 }

```

Class:

Greedy adalah kelas utama yang digunakan untuk melakukan pencarian dengan menggunakan algoritma *Greedy Best First Search*.

Method:

Nama <i>Method</i>	Penjelasan
Greedy	Konstruktor
algorithm	Algoritma pencarian utama <i>Greedy Best First Search</i>
insertInOrder	<i>Method</i> yang digunakan untuk menambahkan nilai dari parameter newItem kedalam list agar tetap terurut membesar berdasarkan nilai

	Integernya.
countLetterDifference	<i>Method</i> ini mengembalikan jumlah perbedaan huruf antara word1 dan word2 dalam integer

6. A_Star.java

```

SearchAlgorithm.java M Greedy.java M A_Star.java X
src > A_Star.java > A_Star > algorithm()
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class A_Star extends SearchAlgorithm {
5
6      public A_Star(String word_start, String word_goal, List<String> dictionary) {
7          super(word_start, word_goal, dictionary);
8      }
9
10     public void algorithm() {
11         long startTime = System.currentTimeMillis();
12
13         // A* Search
14         String currentWord = word_start;
15
16         // Nodes to expand
17         ArrayList<StringIntegerPair> nodeToExpan = new ArrayList<>();
18
19         // Expanded nodes
20         ArrayList<StringIntegerPair> nodeExpansion = new ArrayList<>();
21
22         nodeToExpan.add(new StringIntegerPair(currentWord, integerElement:0));
23
24         WordDiff wd = new WordDiff();
25
26         int cost = 0;
27
28         while (!currentWord.equals(word_goal)) {
29             ArrayList<String> temp = wd.findWordDiff(currentWord, dictionary);
30
31             for (int i = 0; i < temp.size(); i++) {
32                 List<String> firstWordsList = new ArrayList<>();
33
34                 // kata kata simpul yang pernah diekspan
35                 for (StringIntegerPair element : nodeToExpan) {
36                     String[] firstWords = element.getStringElement().split(regex:" ");
37                     firstWordsList.add(firstWords[0]);
38                 }
39
40                 // kalau belum pernah di ekspan baru boleh jadi calon untuk di ekspan
41                 if (!firstWordsList.contains(temp.get(i))) {
42
43                     int heuristic = countLetterDifference(temp.get(i), word_goal);
44
45                     StringIntegerPair newNode = new StringIntegerPair(
46                         temp.get(i) + " " + nodeToExpan.get(nodeToExpan.size() - 1).getStringElement(),
47                         heuristic + cost+1);
48                     insertInOrder(nodeExpansion, newNode);
49                 }
50             }
51         }

```

```

src > A_Star.java > A_Star > algorithm()
4 public class A_Star extends SearchAlgorithm {
10     public void algorithm() {
52         if (nodeExpansion.size() == 0) {
53             System.out.println("\n");
54             System.out.println(x:"***");
55             System.out.println(x:"***");
56             System.out.println(x:"Tidak ada solusi sesuai referensi kamus dictionary.txt milik oracle");
57             System.out.println(x:"***");
58             System.out.println(x:"***");
59             System.out.println(x:"***");
60             System.exit(status:0);
61             break;
62         }
63         StringIntegerPair min = nodeExpansion.remove(index:0);
64         nodeToExpan.add(min);
65
66         currentWord = min.getStringElement().split(regex:" ")[0];
67
68         // System.out.println(currentWord);
69
70         cost = min.getIntegerElement();
71     }
72
73     long stopTime = System.currentTimeMillis();
74     long elapsedTime = stopTime - startTime;
75     //buat ngeprint hasilnya
76     String path = nodeToExpan.get(nodeToExpan.size() - 1).getStringElement();
77
78     System.out.print(s:"\n");
79     System.out.println(x:"Solusi: ");
80     System.out.println(x:"-----");
81     for (int i = path.split(regex:" ").length - 1; i >= 0; i--) {
82         System.out.println(path.split(regex:" ")[i].toUpperCase());
83     }
84     System.out.println(x:"-----\n");
85
86     System.out.println("Banyak node yang dikunjungi: " + nodeToExpan.size());
87     System.out.println("Waktu (ms): " + elapsedTime);
88 }
89
90 public void insertInOrder(ArrayList<StringIntegerPair> list, StringIntegerPair newItem) {
91     int index = 0;
92     // Masuknya biar terurut
93     while (index < list.size() && list.get(index).getIntegerElement() < newItem.getIntegerElement()) {
94         index++;
95     }
96     list.add(index, newItem);
97 }
98
99 public int countLetterDifference(String word1, String word2) {
100     int difference = 0;
101     for (int i = 0; i < word1.length(); i++) {
102         if (word1.charAt(i) != word2.charAt(i)) {
103             difference++;
104         }
105     }
106     return difference;
107 }

```

Class:

A_Star adalah kelas utama yang digunakan untuk melakukan pencarian dengan menggunakan algoritma A*.

Method:

Nama Method	Penjelasan
A_Star	Konstruktor
algorithm	Algoritma pencarian utama A*
insertInOrder	Method yang digunakan untuk menambahkan nilai dari parameter newItem kedalam list agar tetap terurut membesar berdasarkan nilai Integernya.

countLetterDifference	<i>Method</i> ini mengembalikan jumlah perbedaan huruf antara word1 dan word2 dalam integer
-----------------------	---

7. Main.java

```

src > Main.java > Main > main(String[])
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.Scanner;
6
7  public class Main {
8
9      private static final String WORDS_FILE_PATH = "dictionary.txt";
10
11      public static void main(String[] args) {
12
13          //bisa keren
14          System.out.println(x:"-----");
15          System.out.println(x:"|          Selamat Datang di Program    |");
16          System.out.println(x:"|          TUCIL 3 STIMA              |");
17          System.out.println(x:"|          13522028                  |");
18          System.out.println(x:"-----");
19
20          Scanner scanner = new Scanner(System.in);
21
22          List<String> dictionary = loadDictionary();
23
24          String start, goal;
25          do {
26              // Input kata awal
27              System.out.print(s:"Ketikkan Kata Mulai: ");
28              start = scanner.nextLine().trim().toLowerCase();
29
30              // Input kata tujuan
31              System.out.print(s:"Ketikkan Kata Tujuan: ");
32              goal = scanner.nextLine().trim().toLowerCase();
33
34              // Validasi panjang kata
35              if (start.length() != goal.length()) {
36                  System.out.print(s:"\n");
37                  System.out.println(x:"***");
38                  System.out.println(x:"Peringatan : Kata awal dan kata tujuan harus memiliki panjang yang sama.");
39                  System.out.println(x:"***");
40                  System.out.println(x:"");
41                  System.out.print(s:"\n");
42                  continue;
43              }
44
45              // Validasi hanya huruf abjad
46              if (!start.matches(regex:"[a-z]*") || !goal.matches(regex:"[a-z]*")) {
47                  System.out.print(s:"\n");
48                  System.out.println(x:"***");
49                  System.out.println(x:"Peringatan : Kata hanya boleh terdiri dari huruf abjad.");
50                  System.out.println(x:"***");
51                  System.out.println(x:"");
52                  System.out.print(s:"\n");
53                  continue;
54              }
55          }
56
57

```

```

SearchAlgorithm.java M Greedy.java M A_Star.java Main.java X
src > Main.java > Main > main(String[])
7 public class Main {
11     public static void main(String[] args) {
12         // Validasi kata ada di dalam dictionary
13         if (!dictionary.contains(start) || !dictionary.contains(goal)) {
14             System.out.print(s:"\n");
15             System.out.println(x:"***");
16             System.out.println(x:"Peringatan : Salah satu atau kedua kata masukkan tidak ditemukan di kamus.");
17             System.out.println(x:"***");
18             System.out.println(x:"***");
19             System.out.print(s:"\n");
20             continue;
21         }
22         break;
23     } while (true);
24
25     int choice;
26     // Pilihan algoritma
27     do {
28         System.out.println(x:"Pilih Algoritma:");
29         System.out.println(x:"1. UCS (Uniform Cost Search)");
30         System.out.println(x:"2. A*");
31         System.out.println(x:"3. Greedy Best First Search");
32         System.out.print(s:"Masukkan pilihanmu (1/2/3): ");
33         choice = scanner.nextInt();
34         scanner.nextLine();
35
36         if (!(choice == 1 || choice == 2 || choice == 3)) {
37             System.out.print(s:"\n");
38             System.out.println(x:"***");
39             System.out.println(x:"Pilihan tidak valid. Silakan coba lagi.");
40             System.out.println(x:"***");
41             System.out.println(x:"***");
42             System.out.print(s:"\n");
43             continue;
44         }
45         break;
46     } while (true);
47
48     SearchAlgorithm searchAlgorithm = null;
49
50     switch (choice) {
51         case 1:
52             searchAlgorithm = new UCS(start, goal, dictionary);
53             break;
54         case 2:
55             searchAlgorithm = new A_Star(start, goal, dictionary);
56             break;
57         case 3:
58             searchAlgorithm = new Greedy(start, goal, dictionary);
59             break;
60         default:
61             System.out.print(s:"Ada Masalah Pada Pilihan");
62     }
63 }

```

```

src > Main.java > Main > main(String[])
7 public class Main {
11     public static void main(String[] args) {
112         default:
113             System.out.println(x:"Ada Masalah Pada Pilihan");
114             System.out.println(x:"Keluar...");
115             System.exit(status:0);
116         }
117
118         // Menjalankan algoritma
119         if (searchAlgorithm != null) {
120             searchAlgorithm.printWordStartGoal();
121             System.out.println(x:"Loading...");
122             searchAlgorithm.algoritma();
123         }
124
125         scanner.close();
126     }
127
128     // Method untuk kata-kata dari file words_alpha.txt ke dalam ArrayList
129     private static List<String> loadDictionary() {
130         List<String> dictionary = new ArrayList<>();
131         try {
132             //absolute path src
133             String srcDirPath = new File(pathname:"src").getAbsolutePath();
134
135             // System.out.println(srcDirPath);
136             // Mengecek apakah sistem operasi yang digunakan adalah Windows
137             boolean isWindows = System.getProperty(key:"os.name").toLowerCase().startsWith(prefix:"windows");
138
139             //Asumsi
140             // Jika di Windows, maka pemisah jalur file "\\".
141             // Jika di Linux atau MSI, maka pemisah jalur file "/".
142             String fileSeparator = isWindows ? "\\" : "/";
143
144             // Menggabungkan path src dengan nama file menggunakan pemisah yang tepat
145             String fullPath = srcDirPath + fileSeparator + WORDS_FILE_PATH;
146
147             Scanner fileScanner = new Scanner(new File(fullPath));
148             while (fileScanner.hasNextLine()) {
149                 dictionary.add(fileScanner.nextLine().trim());
150             }
151             fileScanner.close();
152         } catch (FileNotFoundException e) {
153             System.out.print(s:"\n");
154             System.out.println(x:"");
155             System.out.println(x:"");
156             System.out.println(x:"");
157             System.out.println(x:"File words_alpha.txt tidak ditemukan.");
158             System.out.println(x:"Tolong file words_alpha.txt nya jangan di hapus atau dipindahkan ya kakak asisten yang terhormat.");
159             System.out.println(x:"Terimakasih :)");
160             System.out.println(x:"");
161             System.out.println(x:"");
162             System.out.print(s:"\n");
163             e.printStackTrace();
164             System.exit(status:0);
165         }
166         return dictionary;
167     }

```

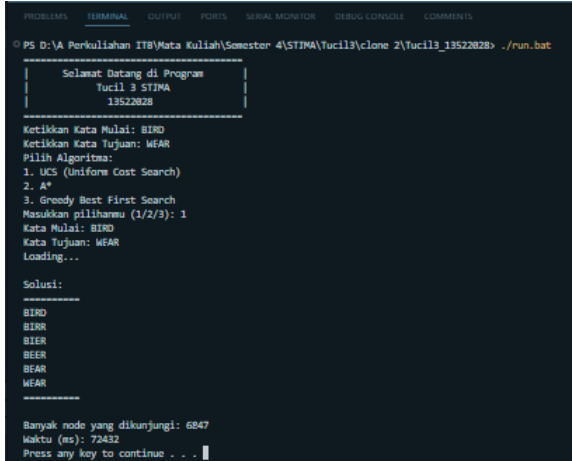
Class:

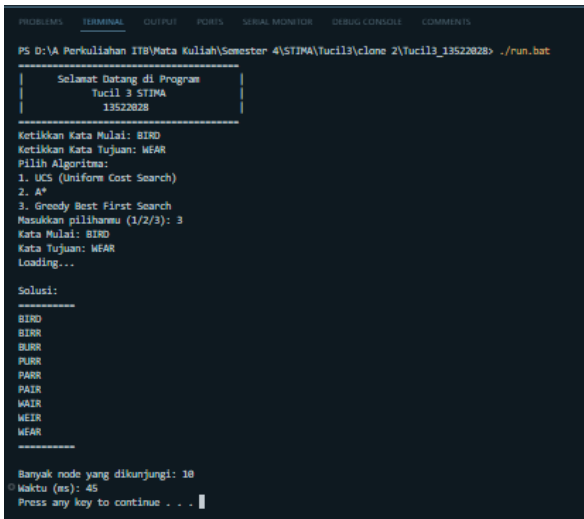
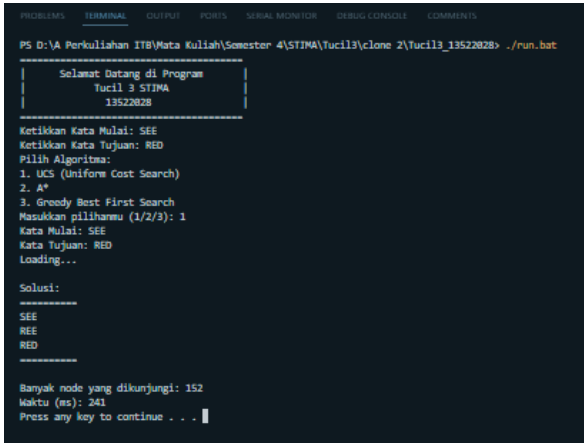
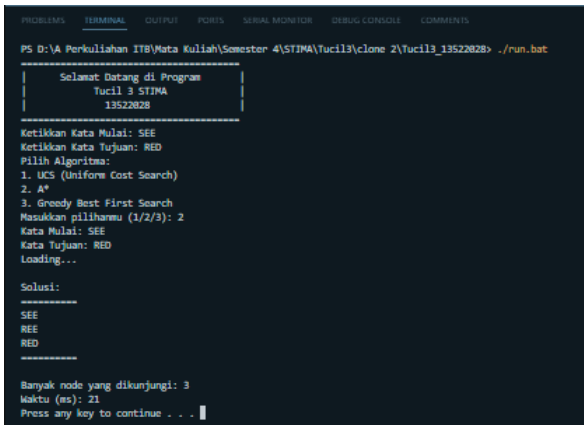
Main adalah kelas utama yang menyatukan semua kelas yang sudah dijelaskan di atas untuk diimplementasikan.

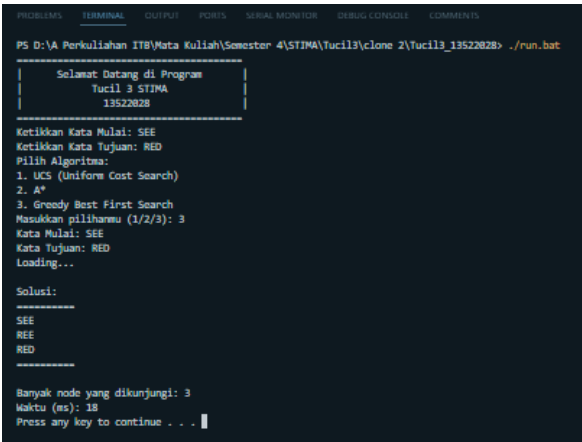
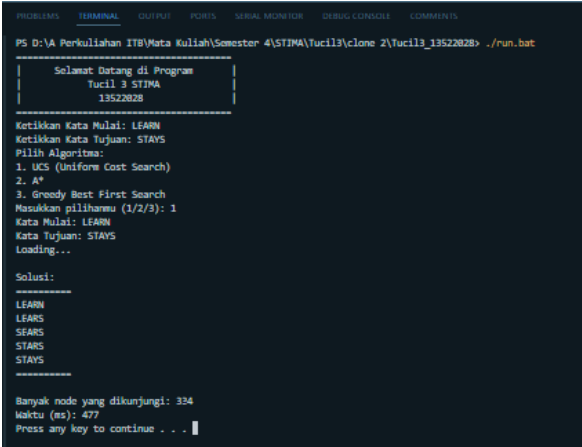
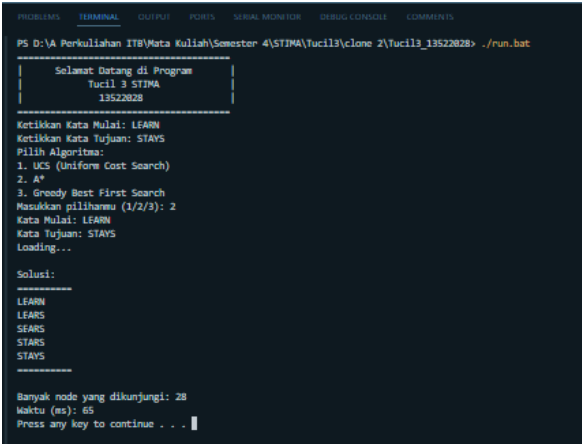
Method:

Nama <i>Method</i>	Penjelasan
main	<i>Method</i> yang dieksekusi pertama kali ketika program dijalankan
loadDictionary	<i>Method</i> yang digunakan untuk memuat kata-kata pada <i>dictionary</i> dan menyimpan kedalam sebuah List<String>

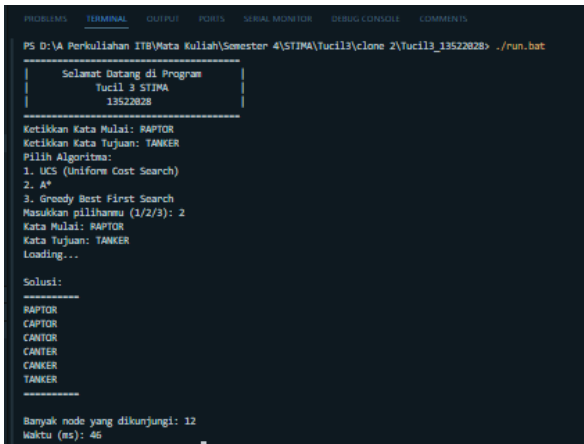
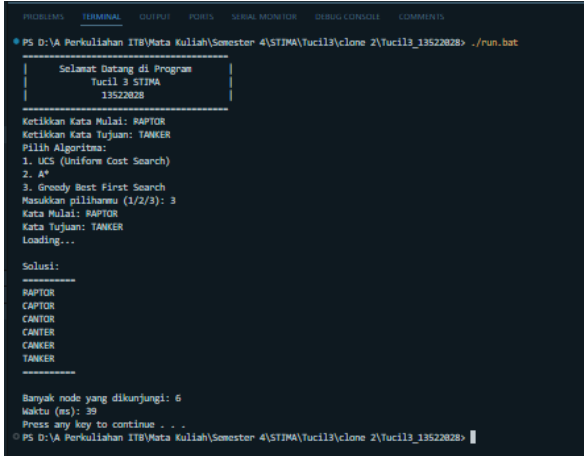
C. Test atau Pengujian Program

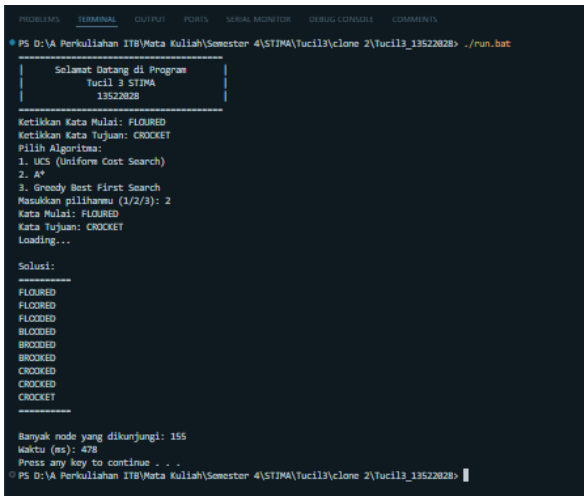
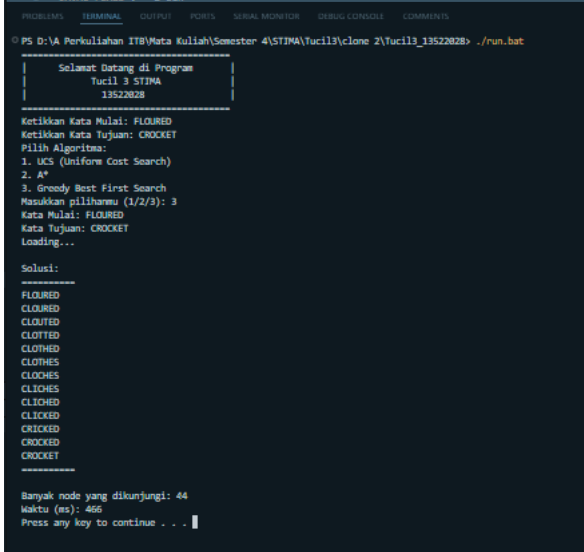
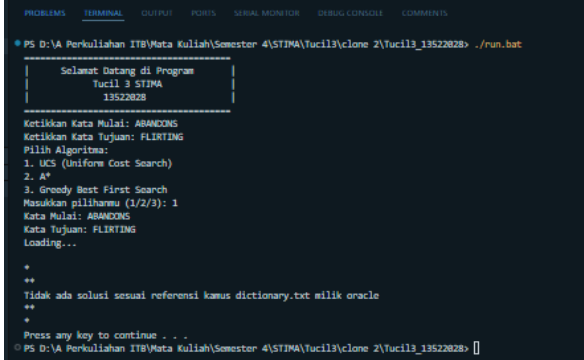
No.	Input	Output
1.	test1.txt kata mulai : BIRD kata tujuan : WEAR	<p>-UCS</p>  <p>Banyak node yang dikunjungi: 6847 Waktu (ms): 72432</p> <p>-A*</p>  <p>Banyak node yang dikunjungi: 145 Waktu (ms): 237</p> <p>-Greedy Best First Search</p>

		 <p>Banyak node yang dikunjungi: 10 Waktu (ms): 45</p>
2.	<p>test2.txt</p> <p>kata mulai : SEE kata tujuan : RED</p>	<p>-UCS</p>  <p>Banyak node yang dikunjungi: 152 Waktu (ms): 241</p> <p>-A*</p>  <p>Banyak node yang dikunjungi: 3</p>

		<p>Waktu (ms): 21</p> <p><i>-Greedy Best First Search</i></p>  <p>Banyak node yang dikunjungi: 3 Waktu (ms): 18</p>
3.	<p>test3.txt</p> <p>kata mulai : LEARN</p> <p>kata tujuan : STAYS</p>	<p><i>-UCS</i></p>  <p>Banyak node yang dikunjungi: 334 Waktu (ms): 477</p> <p><i>-A*</i></p> 

		<p>Banyak node yang dikunjungi: 28 Waktu (ms): 65</p> <p><i>-Greedy Best First Search</i></p>  <p>Banyak node yang dikunjungi: 5 Waktu (ms): 23</p>
4.	<p>test4.txt</p> <p>kata mulai : RAPTOR kata tujuan : TANKER</p>	<p><i>-UCS</i></p>  <p>Banyak node yang dikunjungi: 78 Waktu (ms): 165</p> <p><i>-A*</i></p>

		 <p>Banyak node yang dikunjungi: 12 Waktu (ms): 46</p> <p><i>-Greedy Best First Search</i></p>  <p>Banyak node yang dikunjungi: 6 Waktu (ms): 39</p>
5	<p>test5.txt</p> <p>kata mulai : FLOURED kata tujuan : CROCKET</p>	<p><i>-UCS</i></p>  <p>Banyak node yang dikunjungi: 737 Waktu (ms): 2508</p> <p><i>-A*</i></p>

		 <p>Banyak node yang dikunjungi: 155 Waktu (ms): 478 <i>-Greedy Best First Search</i></p>
		 <p>Banyak node yang dikunjungi: 44 Waktu (ms): 466</p>
6	<p>test6.txt</p> <p>kata mulai : ABANDONS</p> <p>kata tujuan : FLIRTING</p>	<p><i>-UCS</i></p>  <p><i>-A*</i></p>

		 <p style="text-align: center;"><i>-Greedy Best First Search</i></p> 
--	--	--

Tabel 2. Input dan Output Program

D. Analisis Hasil Pengujian

1. Optimalitas Solusi

Dari hasil pengujian, tampaknya algoritma A* dan UCS memberikan solusi yang optimal pada setiap kasus uji. *Greedy Best First Search* memberikan solusi yang optimal pada beberapa kasus uji contohnya adalah untuk kasus test4. Ada juga *test case* yang tidak menghasilkan solusi di algoritma manapun seperti pada kasus test6. Hal tersebut dapat terjadi karena ada kemungkinan ada kata X yang bahkan tidak memiliki satu kata valid yang memiliki perbedaan satu huruf dengan X pada *dictionary*.

2. Waktu Eksekusi

Greedy Best First Search cenderung menjadi yang paling efisien dalam hal waktu eksekusi, diikuti oleh A* , dan UCS yang memiliki waktu eksekusi lebih lama.

3. Memori

Berdasarkan pengujian, UCS membutuhkan memori yang lebih besar dibandingkan dengan A* dan *Greedy Best First Search* karena harus membangkitkan lebih banyak *node*. Sementara itu, A* dan *Greedy Best First Search* membutuhkan memori yang lebih sedikit karena hanya membangkitkan *node* yang paling menjanjikan. Dengan catatan tambahan, *Greedy Best First Search* lebih efisien dalam hal memori dibandingkan A* pada kasus permainan Words Ladder.

E. Asumsi-Asumsi

1. Berikut adalah *dictionary* utama yang digunakan sebagai referensi:

<https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt>

2. Kata tidak mengandung angka atau simbol selain huruf abjad (a - z)

3. Berikut adalah *environment* java yang saya gunakan:

- WSL2
openjdk 21.0.2 2024-01-16
OpenJDK Runtime Environment (build 21.0.2+13-Ubuntu-122.04.1)
OpenJDK 64-Bit Server VM (build 21.0.2+13-Ubuntu-122.04.1, mixed mode, sharing)
- Windows
java 22.0.1 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)

F. Repository

Link Repository dari Tugas Kecil 03 IF2211 Strategi Algoritma Panji Sri Kuncara Wisma:

https://github.com/PanjiSri/Tucil3_13522028.git