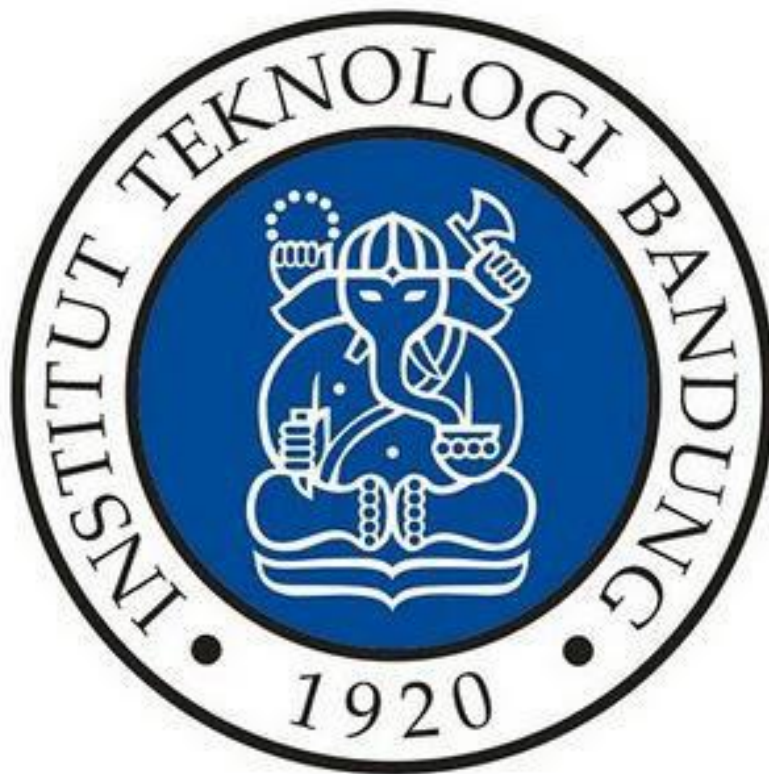


Laporan Tugas Kecil 3 IF2211 Strategi Algoritma Semester IV tahun 2023/2024

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



Disusun oleh:

Eduardus Alvito Kristiadi (13522004)

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

BAB I 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. Ilustrasi serta aturan permainan adalah tercantum pada link: <https://wordwormdormdork.com/> Spesifikasi program adalah sebagai berikut:

- Program dibuat dalam bahasa Java berbasis CLI (Command Line Interface) – bonus jika menggunakan GUI – yang dapat menemukan solusi permainan word ladder menggunakan algoritma UCS, Greedy Best First Search, dan A*.
- Kata-kata yang dapat dimasukkan harus berbahasa Inggris. Cara kalian melakukan validasi sebuah kata dibebaskan, selama kata-kata tersebut benar terdapat pada dictionary dan proses validasi tersebut tidak memakan waktu yang terlalu lama.
- Tugas wajib dikerjakan secara individu.
- Input :
Format masukan dibebaskan, dengan catatan dijelaskan pada README dan laporan. Komponen yang perlu menjadi masukan yaitu.
 1. Start word dan end word. Program harus bisa menangani berbagai panjang kata (tidak hanya kata dengan 4 huruf saja seperti Gambar 1).
 2. Pilihan algoritma yang digunakan (UCS, Greedy Best First Search, atau A*)
- Output :
Berikut adalah luaran dari program yang diekspektasikan.
 1. Path yang dihasilkan dari start word ke end word (cukup 1 path saja)

2. Banyaknya node yang dikunjungi
3. Waktu eksekusi program

BAB II

ALGORITMA UCS, DAN A* SECARA UMUM

2.1 Algoritma Uniform Cost Search (UCS)

Algoritma Uniform Cost Search (UCS) adalah algoritma pencarian lintasan terpendek di antara dua simpul pada sebuah graf atau peta dengan menghitung biaya(*cost*) terkecil untuk mencapai simpul tujuan dari simpul awal. Langkah-langkah algoritma UCS adalah sebagai berikut:

1. Tentukan simpul awal dan simpul tujuan.
2. Buat sebuah *priority queue* yang akan digunakan untuk menyimpan simpul-simpul yang akan dikunjungi beserta dengan lintasan yang digunakan untuk mencapai simpul tersebut dan biayanya.
3. Selama *priority queue* tidak kosong, kunjungi simpul yang memiliki lintasan dengan biaya terendah.
4. Periksa apakah simpul yang sedang dikunjungi adalah simpul tujuan atau bukan. Jika ya, kembalikan lintasan tersebut.
5. Jika simpul yang sedang dikunjungi (*v*) bukan simpul tujuan dan belum pernah dikunjungi sebelumnya, tambahkan tetangga-tetangganya ke dalam *queue* yang telah dibuat. Lintasan dari masing-masing tetangga adalah lintasan yang ditempuh untuk mencapai simpul *v* ditambahkan dengan dirinya sendiri. *Cost* dihitung dari *cost* untuk mencapai simpul *v* ditambah dengan *cost* dari *v* ke tetangga tersebut.
6. Ulangi langkah 3-5 hingga *priority queue* kosong atau ditemukan solusi. Jika *priority queue* kosong, artinya tidak ada lintasan dari simpul awal ke simpul tujuan.

2.2 Algoritma A*

Algoritma A* adalah algoritma pencarian jalur atau *pathfinding* yang digunakan untuk menemukan jalur terpendek antara dua simpul atau titik pada sebuah graf atau peta. Dalam masalah

yang telah dijelaskan pada Bab I, langkah-langkah algoritma A* adalah sebagai berikut:

1. Tentukan simpul asal dan simpul tujuan.
2. Cari terlebih dahulu nilai heuristik tiap simpul untuk algoritma A*, yang dalam persoalan ini berupa jarak tegak lurus dari simpul ke simpul tujuan, simpan jarak tersebut sebagai atribut simpul.
3. Buat sebuah *priority queue* yang akan digunakan untuk menyimpan simpul-simpul yang akan dikunjungi beserta lintasan yang dilalui untuk mencapai simpul tersebut dan biayanya (dalam hal ini biaya adalah jarak), serta prediksi biaya untuk mencapai simpul tujuan dari simpul saat ini. *Priority queue* akan terurut menaik berdasarkan prediksi biaya yang diperlukan untuk mencapai simpul tujuan dari simpul saat ini, prediksi biaya diperoleh dari biaya sebenarnya yang telah terakumulasi hingga mencapai simpul saat ini ditambah dengan nilai heuristik simpul tersebut.
4. Tambahkan simpul asal ke dalam *priority queue*.
5. Selama *priority queue* tidak kosong, kunjungi simpul pertama pada *priority queue* untuk diperiksa. Periksa apakah simpul yang sedang dikunjungi adalah simpul tujuan atau bukan. Jika, ya kembalikan biaya tempuh sebenarnya dari simpul tersebut sebagai jawaban.

6. Jika simpul yang sedang dikunjungi (v) bukan simpul tujuan, periksa apakah tetangganya sudah pernah dikunjungi atau dilewati dalam lintasan untuk mencapai v , jika belum maka tambahkan tetangganya ke dalam *priority queue* yang telah dibuat. Lintasan darimasing-masing tetangga adalah lintasan yang ditempuh untuk mencapai simpul v ditambahkan dengan dirinya sendiri (tetangga itu sendiri). *Cost* tetangga adalah *cost* untuk mencapai simpul v ditambah dengan *cost* dari v ke tetangga tersebut. Prediksi biaya dihitung dari *cost* dirinya sendiri ditambah dengan nilai heuristik dirinya sendiri.
7. Ulangi langkah 5-7 hingga *priority queue* kosong atau ditemukan solusi. Jika *priority queue* kosong, artinya tidak ada lintasan dari simpul awal ke simpul tujuan.

2.3 Algoritma GBFS

Algoritma Greedy Best-First Search (GBFS) adalah metode pencarian yang menggunakan prinsip kegirangan (*greediness*) untuk mencapai solusi. Algoritma ini merupakan salah satu bentuk algoritma pencarian berinformasi yang mengandalkan heuristik atau pengetahuan tambahan mengenai masalah untuk memperkirakan jarak dari keadaan saat ini ke tujuan akhir. Tujuannya adalah untuk menemukan jalur terpendek atau solusi tercepat dari suatu titik awal ke titik tujuan dengan mengikuti jalur yang tampak paling menjanjikan pada setiap langkahnya. Terdapat fungsi heuristik dalam algoritma ini. Fungsi heuristik adalah kunci dari algoritma pencarian berinformasi seperti GBFS. Fungsi ini mengestimasi biaya (atau jarak) terkecil dari sebuah node ke tujuan akhir. Penentuan fungsi heuristik yang baik sangat penting karena kualitas dan efektivitas pencarian sangat bergantung pada akurasi perkiraan ini.

2.4 $f(n)$ dan $g(n)$

2.5 Apakah heuristik yang digunakan pada algoritma A^* *admissible*?

2.6 Pada kasus *word ladder*, apakah algoritma UCS sama dengan BFS?

Dalam kasus ini, penggunaan UCS mirip dengan BFS karena tidak ada pembobotan khusus dalam setiap langkah yang diambil atau dalam kasus ini yaitu tiap node yang dilewati. Pada dasarnya BFS merupakan kasus khusus dari UCS di mana BFS memiliki bobot yang seragam yaitu 1. Dalam kasus ini pun begitu, tiap node yang dilewati mempunyai bobot 1.

2.7 Secara teoritis, apakah algoritma A^* lebih efisien dibandingkan dengan algoritma UCS pada kasus *word ladder*?

Iya, A^* lebih efisien karena dia membandingkan bobot node dari jarak menuju kata tujuan dan jarak dari node awal sehingga perhitungannya lebih efisien saat program dijalankan. A^* tidak perlu melewati node sebanyak UCS.

2.8 Secara teoritis, apakah algoritma *Greedy Best First Search* menjamin solusi optimal untuk persoalan *word ladder*?

Tidak, karena bisa saja GBFS terjebak dalam Solusi optimum local yang dalam hal ini juga bisa terjadi saat tidak ada kata ditengah-tengah antara dua node. Jadi tidak menjamin optimum global.

BAB III

IMPLEMENTASI PROGRAM

DENGAN BAHASA PEMROGRAMAN JAVA

Algoritma pada Bab II diimplementasikan dengan menggunakan bahasa Java. *Source code* dari program dapat diakses pada https://github.com/Edvardesu/Tucil3_13522004

FORMAT MASUKAN:

1. Masukkan startword terlebih dahulu dengan huruf kecil semua (case sensitive)
2. Tekan enter
3. Masukkan targetword dengan huruf kecil semua (case sensitive)
4. Tekan enter
5. Masukkan algoritma yang diinginkan (UCS atau GBFS atau A*) - case sensitive
6. Tekan enter
7. Akan keluar hasil

Source code diimplementasikan secara modular dengan beberapa file yang merepresentasikan algoritma tertentu juga main.py yang berperan sebagai program utama.

1. UCS.java

```
import java.util.*;
import java.io.*;
```

Codeium: Refactor | Explain

```
public class UCS {
    private String start;
    private String target;
    private Set<String> dict;

    private int visitedNode;

    public UCS(String start, String target, Set<String> dict) {
        this.start = start;
        this.target = target;
        this.dict = dict;
        this.visitedNode = 0;
    }
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public List<String> findNeighbors(String word, Set<String> dictionary) {

    List<String> neighbors = new ArrayList<>();
    char[] chars = word.toCharArray();

    for (int i = 0; i < chars.length; i++) {
        char oldChar = chars[i];
        for (char c = 'a'; c <= 'z'; c++) {
            if (c == oldChar) {
                continue;
            }
            chars[i] = c;
            String newWord = new String(chars);
            if (dictionary.contains(newWord)) {
                neighbors.add(newWord);
            }
        }
        chars[i] = oldChar;
    }
    return neighbors;
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public List<String> getResult(Set<String> dictionary, String start, String end, int visitedNode) {
    PriorityQueue<WordStep> pq = new PriorityQueue<>(Comparator.comparingInt(ws -> ws.cost));
    Map<String, String> wordFrom = new HashMap<>();
    Set<String> visited = new HashSet<>();

    pq.add(new WordStepUCS(start, cost:0));
    wordFrom.put(start, value:null);
}
```


Codeium: Refactor | Explain | Generate Javadoc | X

```
public List<String> getResult(Set<String> dictionary, String start, String end, int visitedNode) {
    PriorityQueue<WordStep> pq = new PriorityQueue<>((Comparator.comparingInt(ws -> ws.cost)));
    Map<String, String> wordFrom = new HashMap<>();
    Set<String> visited = new HashSet<>();

    pq.add(new WordStepUCS(start, cost:0));
    wordFrom.put(start, value:null);

    while (!pq.isEmpty()) {
        WordStep current = pq.poll();
        this.visitedNode++;

        if (current.word.equals(end)) {
            return reconstructPath(wordFrom, end);
        }

        if (!visited.contains(current.word)) {
            visited.add(current.word);
            for (String neighbor : findNeighbors(current.word, dictionary)) {
                if (!visited.contains(neighbor)) {
                    pq.add(new WordStepUCS(neighbor, current.cost + 1));
                    wordFrom.put(neighbor, current.word);
                }
            }
        }
    }

    return Collections.emptyList(); // tidak ketemu path
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public List<String> reconstructPath(Map<String, String> parent, String end) {
    List<String> path = new ArrayList<>();
    for (String i = end; i != null; i = parent.get(i)) {
        path.add(i);
    }
    Collections.reverse(path);
    return path;
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public void mainUCS() {
    long startTime = System.nanoTime();
    List<String> path = getResult(dict, start, target, visitedNode);
    long endTime = System.nanoTime();
}
```

```
    return Collections.emptyList(); // tidak ketemu path
}

Codeium: Refactor | Explain | Generate Javadoc | X
public List<String> reconstructPath(Map<String, String> parent, String end) {
    List<String> path = new ArrayList<>();
    for (String i = end; i != null; i = parent.get(i)) {
        path.add(i);
    }
    Collections.reverse(path);
    return path;
}

Codeium: Refactor | Explain | Generate Javadoc | X
public void mainUCS() {
    Long startTime = System.nanoTime();
    List<String> path = getResult(dict, start, target, visitedNode);
    Long endTime = System.nanoTime();
    Long executionTime
        = (endTime - startTime) / 1000000;

    if (path != null) {
        for (String res : path) {
            System.out.println(res);
        }
    } else {
        System.out.println(x:"No path found.");
    }
    System.out.println("Banyak node dikunjungi : " + this.visitedNode);
    System.out.println("Execution time : " + executionTime + "ms");
}
```

2. GBFS.java

```
import java.util.Scanner;
import java.util.*;
import java.io.File;
```

Codeium: Refactor | Explain

```
public class GBFS {
    private String start;
    private String target;
    private List<String> dict;

    public GBFS(String start, String target, List<String> dict) {
        this.start = start;
        this.target = target;
        this.dict = dict;
    }
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public static char getCharFromString(String str, int index) {
    return str.charAt(index);
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public static String setCharFromString(String str, int index, char ch) {
    StringBuffer string = new StringBuffer(str);
    string.setCharAt(index, ch);
    String result = new String(string);
    return result;
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
public static List<String> findNeighbors(String word, List<String> dictionary) {
    List<String> neighbors = new ArrayList<>();
    for (String dictWord : dictionary) {
        if (dictWord.length() != word.length()) {
            continue;
        }
        int diffCount = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) != dictWord.charAt(i)) {
                diffCount++;
            }
            if (diffCount > 1) {
                break;
            }
        }
        if (diffCount == 1) {
            neighbors.add(dictWord);
        }
    }
}
```

```

        break;
    }
}
if (diffCount == 1) {
    neighbors.add(dictWord);
}
}
return neighbors;
}

```

Codeium: Refactor | Explain | Generate Javadoc | X

```

public static PriorQueue countDif(String word, List<String> lstr) {
    PriorQueue pq = new PriorQueue();
    for (String str : lstr) {
        int diffCount = 0;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) != str.charAt(i)) {
                diffCount++;
            }
        }
        pq.enqueue(str, diffCount);
    }
    return pq;
}

```

Codeium: Refactor | Explain | Generate Javadoc | X

```

public static void makeResult(PriorQueue pq, List<String> lstr) {
    String res = pq.getTopVal();
    if (res != null) {
        lstr.add(res);
    }
}

```

Codeium: Refactor | Explain | Generate Javadoc | X

```

public void mainGBFS()
{
    long startTime = System.nanoTime();
    Set<String> visited = new HashSet<>();
    boolean found= true;

    visited.add(start);

    List<String> resList = new ArrayList<>();
    resList.add(start);
}

```

```

    }

    Long startTime = System.nanoTime();
    Set<String> visited = new HashSet<>();
    boolean found = true;

    visited.add(start);

    List<String> resList = new ArrayList<>();
    resList.add(start);

    while (!resList.get(resList.size() - 1).equals(target)) {
        start = resList.get(resList.size() - 1);
        List<String> neighbors = findNeighbors(start, dict);
        neighbors.removeIf(visited::contains); // Remove already visited neighbors
        if (neighbors.isEmpty()) {
            System.out.println(x:"No path found.");
            found = false;
            break;
        }

        PriorityQueue pqueue = countDif(target, neighbors);
        String nextNode = pqueue.getTopVal();
        if (nextNode == null || visited.contains(nextNode)) {
            System.out.println(x:"No further progress can be made.");
            found = false;
            break;
        }

        visited.add(nextNode);
        resList.add(nextNode);
    }

    Long endTime = System.nanoTime();
    Long executionTime = (endTime - startTime) / 1000000;

    System.out.println(x:"RESULT LIST:");
    if (found) {
        for (String res : resList) {
            System.out.println(res);
        }
    }

    System.out.println("Visited nodes: " + visited.size());
    System.out.println("Execution time : " + executionTime + "ms");
}

```

3. Astar.java

```
import java.util.*;
```



Codeium: Refactor | Explain

```
public class Astar {  
    private String start;  
    private String target;  
    private Set<String> dict;  
  
    private int visitedNode;  
  
    public Astar(String start, String target, Set<String> dict) {  
        this.start = start;  
        this.target = target;  
        this.dict = dict;  
        this.visitedNode = 0;  
    }  
}
```

Codeium: Refactor | Explain | Generate Javadoc | X

```
private static int heuristic(String current, String target) {  
    int mismatchCount = 0;  
    for (int i = 0; i < current.length(); i++) {  
        if (current.charAt(i) != target.charAt(i)) {  
            mismatchCount++;  
        }  
    }  
    return mismatchCount;  
}
```

// Generate all valid transformations of a word by changing one letter at a time

Codeium: Refactor | Explain | X

```
private static List<String> findNeighbors(String word, Set<String> dictionary) {  
    List<String> neighbors = new ArrayList<>();  
    char[] chars = word.toCharArray();  
    for (int i = 0; i < word.length(); i++) {  
        char oldChar = chars[i];  
        for (char c = 'a'; c <= 'z'; c++) {  
            if (c != oldChar) {  
                chars[i] = c;  
                String newWord = new String(chars);  
                if (dictionary.contains(newWord)) {  
                    neighbors.add(newWord);  
                }  
            }  
        }  
        chars[i] = oldChar;  
    }  
    return neighbors;  
}
```

```

    }

    // A* Algorithm
    Codeium: Refactor | Explain | X
    public List<String> getResult(String start, String target, Set<String> dictionary, int visitedNode) {
        PriorityQueue<WordStepAstar> frontier = new PriorityQueue<>((Comparator.comparingInt(ws -> ws.getTotalCost())));
        Map<String, Integer> costSoFar = new HashMap<>();
        frontier.add(new WordStepAstar(start, cost:0, heuristic(start, target), parent:null));
        costSoFar.put(start, value:0);

        while (!frontier.isEmpty()) {
            WordStepAstar current = frontier.poll();
            this.visitedNode++;

            if (current.getWord().equals(target)) {
                return makePath(current);
            }

            for (String neighbor : findNeighbors(current.getWord(), dictionary)) {
                int newCost = current.getCost() + 1; // Each step costs 1
                if (!costSoFar.containsKey(neighbor) || newCost < costSoFar.get(neighbor)) {
                    int priority = newCost + heuristic(neighbor, target);
                    frontier.add(new WordStepAstar(neighbor, newCost, priority, current));
                    costSoFar.put(neighbor, newCost);
                }
            }
        }

        return null; // No path found
    }
}

Codeium: Refactor | Explain | Generate Javadoc | X
private static List<String> makePath(WordStepAstar target) {
    LinkedList<String> path = new LinkedList<>();
    WordStepAstar step = target;
    while (step != null) {
        path.addFirst(step.getWord());
        step = step.getPred();
    }
    return path;
}

Codeium: Refactor | Explain | Generate Javadoc | X
public void mainAstar() {
    Long startTime = System.nanoTime();
    List<String> result = getResult(start, target, dict, visitedNode);
    Long endTime = System.nanoTime();
    Long executionTime

```

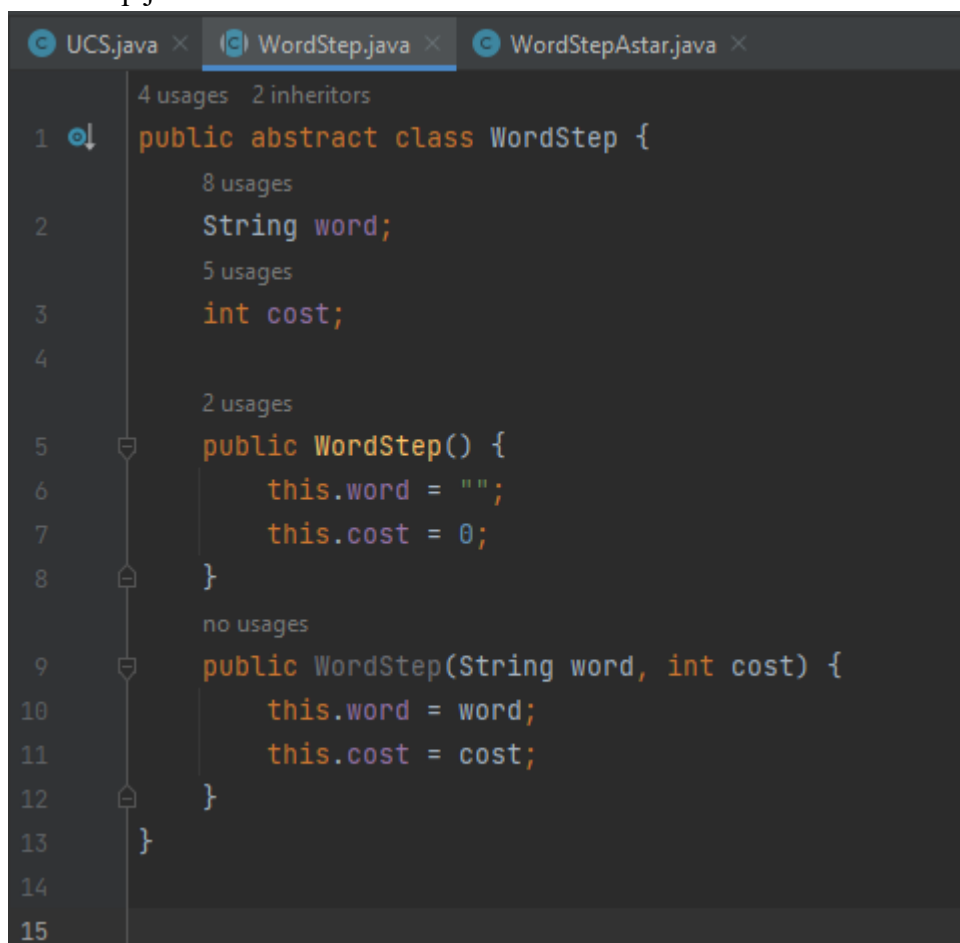
```

Codeium: Refactor | Explain | Generate Javadoc | X
public void mainAstar() {
    Long startTime = System.nanoTime();
    List<String> result = getResult(start, target, dict, visitedNode);
    Long endTime = System.nanoTime();
    Long executionTime
        = (endTime - startTime) / 1000000;

    if (result != null) {
        for (String res : result) {
            System.out.println(res);
        }
    } else {
        System.out.println(x:"No path found.");
    }
    System.out.println("Banyak node dikunjungi : " + this.visitedNode);
    System.out.println("Execution time : " + executionTime + "ms");
}
}

```

4. WordStep.java



```

1  public abstract class WordStep {
2      String word;
3      int cost;
4
5      public WordStep() {
6          this.word = "";
7          this.cost = 0;
8      }
9
10     public WordStep(String word, int cost) {
11         this.word = word;
12         this.cost = cost;
13     }
14 }
15

```

5. WordStepAstar.java


```
9 usages
1 public class WordStepAstar extends WordStep {
2     2 usages
3     private String word;
4     2 usages
5     private int cost; // g(x): Cost from start
6     2 usages
7     private int totalCost; // f(x): g(x) + h(x)
8     2 usages
9     WordStepAstar parent;
10
11     2 usages
12     public WordStepAstar(String word, int cost, int totalCost, WordStepAstar parent) {
13         this.word = word;
14         this.cost = cost;
15         this.totalCost = totalCost;
16         this.parent = parent;
17     }
18
19     1 usage
20     public WordStepAstar getPred() { return this.parent; }
21
22     3 usages
23     public String getWord() { return this.word; }
24
25     1 usage
26     public int getTotalCost() { return this.totalCost; }
27
28     1 usage
29     public int getCost() { return this.cost; }
30 }
31
```

6. WordStepUCS.java

```
2 usages
1 public class WordStepUCS extends WordStep {
2     2 usages
3     public WordStepUCS(String word, int cost) {
4         this.word = word;
5         this.cost = cost;
6     }
7 }
```

7. Main.java

```

1  import java.util.List;
2  import java.util.*;
3  import java.util.Scanner;
4  import java.io.File;
5
6  Codeium: Refactor | Explain
7  public class Main {
8      Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | X
9      // pass the path to the file as a parameter
10     File file = new File(
11         pathname:"words_alpha.txt");
12     Scanner sc = new Scanner(file);
13
14     List<String> dictionaryL = new ArrayList<>();
15     Set<String> dictionaryS = new HashSet<>();
16     while (sc.hasNextLine()) {
17         String line = sc.nextLine().trim();
18         if (!line.isEmpty()) {
19             dictionaryL.add(line);
20             dictionaryS.add(line);
21         }
22     }
23     sc.close(); // Close the file scanner
24
25     System.out.println(x:"Enter Start");
26     Scanner myObj1 = new Scanner(System.in);
27     String start = myObj1.nextLine().trim();
28     if (!dictionaryL.contains(start)) {
29         System.out.println(x:"Start word not in dictionary");
30         return;
31     }
32     if (!dictionaryS.contains(start)) {
33         System.out.println(x:"Start word not in dictionary");
34         return;
35     }
36
37     System.out.println(x:"Enter Target");
38     Scanner myObj2 = new Scanner(System.in);
39     String target = myObj2.nextLine().trim();
40
41     if (!dictionaryL.contains(target)) {
42         System.out.println(x:"Target word not in dictionary");
43         return;
44     }
45     if (!dictionaryS.contains(target)) {
46         System.out.println(x:"Target word not in dictionary");
47         return;
48     }

```

```

public class Main {
    public static void main(String[] args) throws Exception {
        System.out.println(x:"Target word not in dictionary");
        return;
    }
    if (!dictionaryS.contains(target)) {
        System.out.println(x:"Target word not in dictionary");
        return;
    }

    if (start.length() != target.length()) {
        System.out.println(x:"Start and target words must be of the same length.");
    }

    if (start.equals(target)) {
        System.out.println(x:"Start and target word sudah sama");
        return;
    }

    UCS ucs = new UCS(start, target, dictionaryS);
    GBFS gbfs = new GBFS(start, target, dictionaryL);
    Astar astar = new Astar(start, target, dictionaryS);

    // List<String> path = UCS.getResult(dictionaryS, start, target);

    System.out.println(x:"Pick your algorithm!!!");
    System.out.println(x:"=== Type in UCS / GBFS / A* ===");
    Scanner nyar = new Scanner(System.in);
    String option = nyar.nextLine().trim();
    myObj1.close();
    myObj2.close();
    nyar.close();
    if (option.equals(anObject:"UCS")) {
        ucs.mainUCS();
    } else if (option.equals(anObject:"GBFS")) {
        gbfs.mainGBFS();
    } else if (option.equals(anObject:"A*")) {
        astar.mainAstar();
    }
}
}

```

BAB IV EKSPERIMEN

1. Pengujian 1

Start: Earn

Target: Make

Hasil Pengujian

UCS:

```
"D:\EDUARD\IT\IntelliJ IDEA Commu... Main x
Enter Start
earn
Enter Target
make
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
earn
barn
baru
barf
warf
wark
warl
warm
warp
wars
wart
wary
gary
gars
jars
jarl
karl
marl
parl
paal
paas
baas
babs
bads
bags
bais
bals
bams
bans
bass
bats
cats
cats

marl
parl
paal
paas
baas
babs
bads
bags
bais
bals
bams
bans
bass
bats
cats
fats
hats
kats
lats
labs
lacs
lace
lade
lake
lame
fame
fawe
fane
fate
face
fade
fake
jake
make
Banyak node dikunjungi : 3083
Execution time : 30ms
Process finished with exit code 0
```

GBFS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community Edition 20
Enter Start
earn
Enter Target
make
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
RESULT LIST:
earn
yarn
yare
mare
make
Visited nodes: 5
Execution time : 53ms

Process finished with exit code 0
|

```

A*:

```

"D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
earn
Enter Target
make
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
earn
tarn
tare
take
make
Banyak node dikunjungi : 18
Execution time : 12ms

Process finished with exit code 0

```

2. Pengujian 2

Start: mist

Target: pope

Hasil Pengujian

UCS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
mist
Enter Target
pope
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
mist
misy
misc
mise
mose
mosk
moss
mass
masa
mala
mela
mola
cola
dola
hola
hold
hole
jole
pole
poke
pome
pone
pope
Banyak node dikunjungi : 5568
Execution time : 52ms

Process finished with exit code 0

```

GBFS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community Edition
Enter Start
mist
Enter Target
pope
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
RESULT LIST:
mist
pist
post
pose
pope
Visited nodes: 5
Execution time : 44ms

Process finished with exit code 0
|

```

A*:

```

"D:\EDUARD\IT\IntelliJ IDEA Community E
Enter Start
mist
Enter Target
pope
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
mist
pist
pise
pose
pope
Banyak node dikunjungi : 9
Execution time : 5ms

Process finished with exit code 0
|

```

3. Pengujian 3

Start: blue

Target: bird

Hasil Pengujian

UCS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community E
Enter Start
blue
Enter Target
bird
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
blue
clue
cove
code
cade
bade
made
make
rake
bake
bale
dale
hale
pale
pala
paba
paca
paga
saga
sara
bara
baba
baga
bala
bana
banc
band
bind
bird
Banyak node dikunjungi : 12599
Execution time : 82ms

```

GBFS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community Edit
Enter Start
blue
Enter Target
bird
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
RESULT LIST:
blue
blur
bour
boud
bord
bird
Visited nodes: 6
Execution time : 47ms

Process finished with exit code 0

```

A*:

```

"D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
blue
Enter Target
bird
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
blue
blur
bour
boud
bord
bird
Banyak node dikunjungi : 42
Execution time : 5ms

Process finished with exit code 0

```

4. Pengujian 4

Start: bride

Target: thumb

Hasil Pengujian

UCS:


```
"D:\EDUARD\IT\IntelliJ IDEA Community E
Enter Start
bride
Enter Target
thumb
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
bride
brite
brise
bribe
brike
brine
briny
bliny
blimy
blimp
flimp
flump
clump
chump
thump
thumb
Banyak node dikunjungi : 14875
Execution time : 122ms

Process finished with exit code 0
|
```

GBFS:

```
"D:\EDUARD\IT\IntelliJ IDEA Community Ed
Enter Start
bride
Enter Target
thumb
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
No path found.
RESULT LIST:
Visited nodes: 17
Execution time : 93ms

Process finished with exit code 0
```

A*:

```
"D:\EDUARD\IT\IntelliJ IDEA Communit
Enter Start
bride
Enter Target
thumb
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
bride
gride
grime
grume
grump
trump
thump
thumb
Banyak node dikunjungi : 124
Execution time : 8ms

Process finished with exit code 0
```

5. Pengujian 5

Start: plant

Target: brake

Hasil Pengujian

UCS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
plant
Enter Target
brake
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
plant
platt
plait
slait
slart
blart
clart
clast
blast
boast
beast
brast
bract
brace
brake
Banyak node dikunjungi : 3900
Execution time : 47ms

Process finished with exit code 0

```

GBFS:

```

"D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
plant
Enter Target
brake
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
RESULT LIST:
plant
plane
slane
slake
blake
brake
Visited nodes: 6
Execution time : 108ms

Process finished with exit code 0

```

A*:

```
"D:\EDUARD\IT\IntelliJ IDEA Community Editi
Enter Start
plant
Enter Target
brake
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
plant
plane
flane
flake
blake
brake
Banyak node dikunjungi : 27
Execution time : 5ms

Process finished with exit code 0
```

6. Pengujian 6

Start: bridge

Target: church

Hasil Pengujian

UCS:

```
"D:\EDUARD\IT\IntelliJ IDEA Community Editi
Enter Start
bridge
Enter Target
church
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
UCS
Banyak node dikunjungi : 48768
Execution time : 385ms

Process finished with exit code 0
```

GBFS:

```

D:\EDUARD\IT\IntelliJ IDEA Community Edi
Enter Start
bridge
Enter Target
church
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
GBFS
No path found.
RESULT LIST:
Visited nodes: 4
Execution time : 50ms

Process finished with exit code 0

```

A*:

```

D:\EDUARD\IT\IntelliJ IDEA Community
Enter Start
bridge
Enter Target
church
Pick your algorithm!!!
=== Type in UCS / GBFS / A* ===
A*
No path found.
Banyak node dikunjungi : 20579
Execution time : 375ms

Process finished with exit code 0

```

OPTIMALITAS

Untuk masalah optimalitas, didapat bahwa A* lah yang paling optimal karena durasinya paling cepat dan memiliki probabilitas lebih tinggi untuk mendapatkan hasil akhir (path found). A star juga memiliki jumlah kata dalam path lebih sedikit dan lebih akurat sehingga meningkatkan optimalitas.

WAKTU EKSEKUSI

Untuk waktu eksekusi, A* lah yang seringkali paing cepat. Hal ini dikarenakan A*

menempuh jumlah kata dalam path yang lebih sedikit. Bisa dilihat pula dari hasil pengujian, bahwa mayoritas A* lah yang paling cepat dalam execution time.

MEMORI

Untuk penggunaan memori, yang paling unggul ialah Greedy sebab Greedy memiliki jumlah node dikunjungi lebih sedikit dibanding yang lain. Hal ini pula yang menyebabkan Greedy terkadang tidak mendapatkan hasil akhir (no path found).

BAB V

KESIMPULAN DAN KOMENTAR

5.1 Kesimpulan

Algoritma UCS, GBFS, serta A* sangat berguna dalam metode pencarian jalur (pathfinding). Ketiga algoritma tersebut dapat dirasakan kegunaanya dalam tugas besar ini yaitu Word Ladder. Dalam Word Ladder terdapat perbedaan antara ketiga algoritma tersebut dalam hal $g(x)$ dan $h(x)$. Masing-masing memiliki kelebihan dan kekurangan.

5.2 Komentar

Pengerjaan tugas besar ini terlalu terburu-buru dan sangat mendekati deadline. Dapat disarankan untuk ke depannya lebih memanfaatkan waktu dengan baik agar tugas besar menjadi lebih sempurna hasilnya.

BAB VI LAMPIRAN

Link Repository: https://github.com/Edvardesu/Tucil3_13522004

Tabel Tugas:

| 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 <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 | | √ |