

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma  
Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS,  
Greedy Best First Search, dan A\*



Disusun oleh:  
Muhamad Rifki Virziadeili Harisman 13522120

## Algoritma

Pada tucil kali ini akan dibuat penyelesaian permainan word ladder menggunakan algoritma UCS, GBFS, dan A\*. Secara garis besar, ketiga algoritma ini memiliki kemiripan dalam implementasinya. Maka dari itu berikut adalah rangkuman dari langkah-langkah implementasinya,

1. Pembentukan graph dan menyimpannya di dalam file txt
2. Validasi input dan target
3. Lakukan pencarian input menuju target

Untuk meminimalisir waktu pencarian yang begitu lama ketika langsung mencarinya pada dictionary (kamus kata bahasa inggris), graph dari setiap panjang suku kata dibentuk terlebih dahulu dari dictionary. Jadi akan terbentuk file-file txt yang berisikan informasi graph yang siap pakai. Proses pembentukan graph disertai dengan proses validasi string mana saja yang memenuhi ketentuan (Harus memiliki panjang kata yang sama, dari satu node ke node lain harus memiliki perbedaan karakter maksimal 1, tidak boleh mengandung non-alfabetik). Kemudian, pada saat kita akan mencari path dari **start** menuju **target**, kita bisa langsung memilih file txt mana yang harus digunakan sesuai dengan panjang dari suku kata tersebut.

Ketika sudah memilih file txt yang akan dijadikan database kata, muat graph agar mudah digunakan. Validasi **start** dan **target** dilakukan dengan cara mencari kata tersebut pada file txt terpilih. Jika tidak ditemukan maka akan langsung menghentikan eksekusi karena kata yang dimasukan tidak valid (tidak terdapat pada dictionary). Ketika ditemukan, maka node akan dimasukkan ke priority queue.

**Priority queue** adalah tempat menyimpan node pencarian. Node akan terurut dari kecil-besar menurut dari cost node tersebut. Cost dari tiap-tiap node akan ditentukan sesuai dengan algoritma yang akan digunakan.

Penelusuran node akan dilakukan dengan menelusuri node dengan cost terkecil pada **priority queue**. Selama proses penelusuran, setiap node yang pernah dijelajahi akan disimpan ke dalam **parentMap** dan list **visited**. **parentMap** adalah tempat menyimpan informasi parent dari node. Jika **currentNode** bukan **targetNode** maka lanjutkan pencarian. Tambahkan ke priority queue **childNode** yang ada pada node tersebut beserta cost masing-masing node yang telah dihitung terlebih dahulu. Jika node tersebut adalah node target, maka hentikan penelusuran. Jika sampai antrian kosong tidak menemukan target. Artinya solusi tidak ditemukan.

Lakukan pencarian rute terbalik dari **currentNode** ke **startNode**. Pencarian ini akan memanfaatkan **parentMap**. Simpan hasil tersebut ke dalam **List<Node>** dan **reverse**. Hasil path berhasil dibentuk.

## UCS (Uniform Cost Search)

### 1. Skema penentuan cost:

Cost pada UCS direpresentasikan sebagai jarak tempuh dari **currentNode** ke **startNode**. Untuk memulai, node pertama yaitu **startNode** akan diberi  $\text{cost}=0$ . **childNode** dari **startNode** akan diberi nilai  $\text{cost} = 1$ . Pada pencarian berikutnya, setiap kali kita akan memasukkan node ke dalam priority queue, maka cost akan berisi cost dari current +1.

### 2. Kelebihan:

Algoritma ini akan menghasilkan rute pencarian yang optimal. Definisi optimal pada kasus ini adalah rute yang dihasilkan memiliki cost atau jarak yang paling minimum.

### 3. Kekurangan:

Meskipun menghasilkan rute optimal, algoritma ini akan menguras waktu pencarian lebih lama dibanding GBFS ataupun A\*. Hal ini disebabkan oleh pencarian yang dilakukan oleh UCS biasanya lebih banyak daripada algoritma GBFS ataupun A\*.

### 4. UCS vs BFS:

Pada kasus pencarian solusi permainan word ladder, algoritma UCS dan BFS memiliki kesamaan dari urutan pengunjungan node yang akan ditelusuri. Hal ini disebabkan oleh cost yang digunakan adalah bertambah 1 setiap kedalaman. Ini menyebabkan pencarian akan terurut seperti BFS. Akibat dari kesamaan rute penelusuran, maka rute yang dihasilkan pun akan sama.

## GBFS (Greedy Best First Search)

### 1. Skema penentuan cost:

Cost pada GBFS direpresentasikan sebagai jumlah perbedaan karakter pada kata **currentNode** dengan kata dari **targetNode**. Setiap kali kita akan menambahkan node ke priority queue, cost akan dihitung terlebih dahulu. Artinya, priority queue akan terurut kecil-besar sesuai dengan jumlah karakter yang berbeda. Alasan menggunakan skema cost seperti itu adalah sebagai berikut, apabila kita memilih node dengan jumlah perbedaan karakter sesedikit mungkin, maka akan semakin mudah kita menemukan target yang kita cari. Ini akan menghasilkan pencarian yang lebih cepat dari algoritma sebelumnya yaitu ucs.

### 2. Kelebihan:

Algoritma ini akan menghasilkan rute pencarian dengan waktu yang sangat cepat dibanding algoritma UCS ataupun A\* seperti yang telah dijelaskan sebelumnya.

### **3. Kekurangan:**

Rute hasil pencarian menggunakan algoritma ini biasanya akan lebih panjang dibanding dengan algoritma lainnya. Hal ini disebabkan oleh pencarian pada rute ini tidak menelusuri secara menyeluruh, sehingga kadangkali menelusuri node yang kurang efektif.

### **4. Keoptimalan solusi:**

Secara teori algoritma ini tidak menjamin solusi yang dihasilkan adalah solusi optimal. GBFS memang akan selalu memilih pilihan node yang paling optimal berdasarkan greedy yang telah ditentukan (maksimum lokal). Akan tetapi, kita tidak bisa menjamin apakah maksimum lokal tersebut akan membawa ke solusi akhir yang optimal. Pada kasus nyata juga rute pencarian yang dihasilkan memang tidak selalu optimal. Ketidakeoptimalan ini bisa disebabkan oleh pemilihan node yang menurut greedy sudah paling optimal, justru menutup kesempatan rute lain untuk mencari solusi.

## **A\* algorithm**

### **1. Skema penentuan cost (Is heuristic admissible ?)**

Definisi suatu heuristic itu admissible ketika  $h(n)$  pada node tersebut selalu kurang dari  $h(n)$  solusi. Artinya selama pencarian nilai cost selalu lebih rendah dari cost pada saat menemukan solusi. Pada kasus ini cost ditentukan dengan menjumlahkan (jarak+beda karakter). Ini seperti menjumlahkan cost UCS dan cost GBFS. Dengan mencari hasil penjumlahan paling minimum kita akan mengimbangi keoptimalan UCS dan kecepatan GBFS. Skema penentuan cost seperti ini termasuk ke dalam “heuristic admissible”. Hal ini bisa dilihat bahwa cost pada saat pencarian tidak akan melebihi cost akhir (target) yang akan kita tuju.

### **2. Kelebihan**

Dengan mengimbangi kelambatan UCS, tetapi tetap menghasilkan rute optimal, dan ketidakeoptimalan rute GBFS, tetapi tetap lebih cepat, A\* dapat menghasilkan rute yang optimal serta dengan waktu yang relatif lebih cepat dibandingkan dengan UCS,

### **3. A\* vs UCS**

Secara teori A\* lebih efisien dari UCS. Hal ini dikarenakan A\* memilih cost dengan mempertimbangkan jarak dan perbedaan karakter. Sehingga akan mengimbangi UCS dan GBFS. Alhasil akan menghasilkan rute yang optimal dengan waktu yang relatif lebih cepat seperti yang telah dijelaskan sebelumnya.

## Source Code

Kelas	Gambar
<p>Kelas Node</p> <p>Node adalah kelas yang dibentuk untuk menampung node pencarian. Berisikan attribut parent, word, dan list connected</p>	 <pre>import java.util.List; import java.util.ArrayList; public class Node {     private Node parent;     private String word;     private List&lt;Node&gt; connected;      public Node(String word) {         this.word = word;         this.connected = new ArrayList&lt;&gt;();     }      public void addConnected(Node node) {         this.connected.add(node);     }      // Metode untuk menetapkan parent dari node     public void setParent(Node parent) {         this.parent = parent;     }      // Metode untuk mendapatkan parent dari node     public Node getParent() {         return parent;     }      public String getWord(){         return word;     }      public List&lt;Node&gt; getConnected(){         return connected;     } }</pre>

## Kelas graph

Graph adalah kelas yang dibuat sebagai struktur data utama dalam pencarian rute. Berisikan kumpulan pasangan string dan node.

```
public class Graph {
    Map<String, Node> nodes;

    // constructor
    public Graph() {
        this.nodes = new HashMap<>();
    }

    // add connection two word and put in graph
    public void add(String word1, String word2) {
        Node node1 = this.nodes.getDefault(word1, new Node(word1));
        Node node2 = this.nodes.getDefault(word2, new Node(word2));
        node1.addConnected(node2);
        node2.addConnected(node1);
        node2.setParent(node1);
        this.nodes.put(word1, node1);
        this.nodes.put(word2, node2);
    }

    // getter nodes
    public Node get(String word) {
        return this.nodes.get(word);
    }

    // save graph to txt file in csv format
    public void saveToCSV(String filename) {
        try (FileWriter writer = new FileWriter(filename)) {
            for (Map.Entry<String, Node> entry : nodes.entrySet()) {
                String word = entry.getKey();
                Node node = entry.getValue();
                for (Node connectedNode : node.getConnected()) {
                    String line = word + "," + connectedNode.getWord() + "\n";
                    writer.write(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Load graph from txt
    public static Graph readFromCSV(String filename) {
        Graph graph = new Graph();

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(",");
                String word1 = parts[0];
                String word2 = parts[1];

                graph.add(word1, word2);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return graph;
    }
}
```

```
// build graph from dictionary
public static Graph buildGraph(int length, List<String> dict){
    Graph graph = new Graph();
    ExecutorService executor = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());

    for (String word : dict) {
        executor.execute(() -> {
            if (word.length() == length) {
                for (int i = 0; i < length; i++) {
                    for (int alp = 0; alp < 26; alp++) {
                        String current = word.substring(0, i) + (char) (97 + alp) + word.substring(i + 1);
                        if (dict.contains(current) && !current.equalsIgnoreCase(word)) {
                            graph.add(word, current);
                            System.out.println(word);
                        }
                    }
                }
            }
        });
    }

    executor.shutdown();
    while (!executor.isTerminated()) {
        // tunggu semua proses konkuren selesai
    }
    return graph;
}

// print graph for checking
public void printGraph() {
    for (Map.Entry<String, Node> entry : nodes.entrySet()) {
        String word = entry.getKey();
        Node node = entry.getValue();
        System.out.print(word + " is connected to: ");
        for (Node connectedNode : node.getConnected()) {
            System.out.print(connectedNode.getWord() + " ");
        }
        System.out.println();
    }
}
```

## Kelas NodeCostPair

Sebetulnya kelas ini hanya menghubungkan node dengan cost. Node ini yang akan disimpan pada priority queue nantinya.

```
// Kelas untuk menyimpan pasangan node dan cost
public class NodeCostPair implements Comparable<NodeCostPair> {
    private Node node;
    private int cost;

    // constructor
    public NodeCostPair(Node node, int cost) {
        this.node = node;
        this.cost = cost;
    }

    // getter node
    public Node getNode(){
        return node;
    }

    // getter cost
    public int getCost(){
        return cost;
    }

    // setter cost
    public void setCost(int cost){
        this.cost = cost;
    }

    // Implementasi metode compareTo untuk membandingkan pasangan berdasarkan cost
    @Override
    public int compareTo(NodeCostPair other) {
        return Integer.compare(this.cost, other.cost);
    }
}
```

## Kelas Algorithm

Ini adalah kelas yang menjadi algoritma utama dalam pencarian rute. Terdapat algoritma pencarian rute beserta algoritma penentuan cost dari masing masing algoritma.

```
import java.util.*;

public class Algorithm {
    private List<Node> visited;
    private PriorityQueue<NodeCostPair> pQueue;
    private String name;

    // constructor
    public Algorithm(String name){
        pQueue = new PriorityQueue<>();
        visited = new ArrayList<>();
        this.name = name;
    }

    // calculate cost UCS
    public int costUCS(NodeCostPair parent, boolean isFirst){
        if(isFirst){
            return 1;
        }else{
            return parent.getCost()+1;
        }
    }

    // calculate cost GBFS
    public int costGBFS(Node currentNode, String target, boolean isFirst){
        if(isFirst){
            return currentNode.getWord().length();
        }else{
            int dif=0;
            for (int i = 0; i < currentNode.getWord().length(); i++) {
                if(currentNode.getWord().charAt(i)!=target.charAt(i)){
                    dif++;
                }
            }
            return dif;
        }
    }

    // calculate cost
    public int calculateCost(Node currentNode, NodeCostPair parent, Node neighbor, String target, boolean isFirst){
        if(name.equalsIgnoreCase("gbfs")){
            return costGBFS(neighbor, target, isFirst);
        }else if(name.equalsIgnoreCase("ucs")){
            return costUCS(parent, isFirst);
        }else{
            return costUCS(parent, isFirst) + costGBFS(currentNode, target, isFirst);
        }
    }
}
```

```

// Algoritma pencarian rute
public List<String> routeSearch(Graph g, String start, String target){
    Node startNode = g.get(start);
    Node targetNode = g.get(target);
    Map<Node, Node> parentMap = new HashMap<>();
    if (startNode == null || targetNode == null) {
        return null;
    }
    visited.add(startNode);
    for (Node node : startNode.getConnected()) {
        int cost = calculateCost(node, null, node, target, true);
        pQueue.add(new NodeCostPair(node, cost));
        parentMap.put(node, startNode);
    }
    while(!pQueue.isEmpty()){
        NodeCostPair current = pQueue.poll();
        // System.err.println("current: "+ current.getNode().getWord()+ ", cost: "+ current.getCost());
        if(visited.contains(current.getNode())){
            continue;
        }
        visited.add(current.getNode());
        if(target.equals(current.getNode().getWord())){
            List<String> res= getPathList(buildPath(parentMap, targetNode));
            return res;
        }
        for (Node node : current.getNode().getConnected()) {
            int cost = calculateCost(current.getNode(), current, node, target, false);
            pQueue.add(new NodeCostPair(node, cost));
            if(!visited.contains(node) && parentMap.get(node)==null){
                parentMap.put(node, current.getNode());
            }
        }
    }
    return null;
}

// Metode untuk membangun jalur dari titik awal ke titik target
public List<Node> buildPath(Map<Node, Node> parentMap, Node target) {
    List<Node> path = new ArrayList<>();
    Node current = target;

    while (current != null) {
        path.add(current);
        current = parentMap.get(current);
    }

    Collections.reverse(path);
    return path;
}

public List<String> getPathList(List<Node> path){
    List<String> pathList = new ArrayList<>();
    for (Node node : path) {
        pathList.add(node.getWord());
    }
    return pathList;
}

public List<String> getVisitedList(){
    List<String> visitedList = new ArrayList<>();
    for (Node node : visited) {
        visitedList.add(node.getWord());
    }
    return visitedList;
}

public void addVisitedList(String word){
    visited.add(new Node(word));
}
}

```



## Kelas Main

Kelas ini menjadi pembungkus utama program. Dalam kelas ini kita akan melakukan permulaan program dengan memilih scrap data atau run program.

Scrap adalah skema pembentukan graph dari dictionary. Terdapat dua dictionary (dari rekomendasi asisten dan riset pribadi)

Run adalah menjalankan program sesuai dengan instruksi di spesifikasi tugas besar

```
public class Main{
    public static Scanner scanner = new Scanner(System.in);

    public static String getAlgo(String i){
        String res;
        if(i.equalsIgnoreCase("1")){
            res = "ucs";
        }else if(i.equalsIgnoreCase("2")){
            res = "gbfs";
        }else if(i.equalsIgnoreCase("3")){
            res = "A*";
        }else{
            res = null;
        }
        return res;
    }

    public static void output(FileWriter writer, List<String> res, Algorithm game, long startTime){
        System.out.println("-----Jawaban-----");
        System.out.println("Rute: " + res);
        System.out.println("Banyak step: " + (res.size()-1));
        System.out.println("Banyak kata dikunjungi: " + game.getVisitedList().size());
        System.out.println("Durasi pencarian: " + (System.currentTimeMillis()-startTime) + "ms");
        System.out.println("-----");
        try{
            writer.write("-----Jawaban-----\n");
            writer.write("Rute: " + res + "\n");
            writer.write("Banyak step: " + (res.size()-1) + "\n");
            writer.write("Banyak kata dikunjungi: " + game.getVisitedList().size() + "\n");
            writer.write("Durasi pencarian: " + (System.currentTimeMillis()-startTime) + "ms" + "\n");
            writer.write("-----\n");
            writer.write("\n");
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

```

public static void run(){
    String algo, lagi, start, target, path, filename, dict;
    System.out.print("output file : ");
    filename = scanner.nextLine();
    System.out.println("dictionary : ");
    System.out.println("1. Asistent");
    System.out.println("2. Pribadi");
    System.out.print("pilih: ");
    dict = scanner.nextLine();
    if(dict.equalsIgnoreCase("1")){
        dict = "assistent";
    }else{
        dict = "prib";
    }
    try(FileWriter writer = new FileWriter("../test/"+filename+".txt")){
        do{
            boolean onlyAlphabetic1, onlyAlphabetic2;
            do{
                System.out.print("Start : ");
                start = scanner.nextLine();
                System.out.print("Target : ");
                target = scanner.nextLine();

                // Mengecek apakah string hanya mengandung karakter alfabet
                onlyAlphabetic1 = Pattern.matches("[a-zA-Z]+", start);
                onlyAlphabetic2 = Pattern.matches("[a-zA-Z]+", target);
                if(start.length() != target.length() || (!onlyAlphabetic1 || !onlyAlphabetic2)){
                    System.out.println("Masukkan string dengan panjang yang sesuai");
                }
            }while(start.length() != target.length() || (!onlyAlphabetic1 || !onlyAlphabetic2));
            writer.write("Start : " + start + "\n");
            writer.write("Target : " + target + "\n");

            // Load graph
            path = "../dict/"+dict+"/"+start.length() + ".txt";
            Graph graph = Graph.readFromCSV(path);

            // mainkan
            System.out.println("-----");
            System.out.println("Tersedia 3 Algoritma : ");
            System.out.println("1. UCS (Uniform Cost Search)");
            System.out.println("2. GBFS (Greedy Best-First Search)");
            System.out.println("3. A* (Uniform Cost Search)");
            do{
                System.out.print("Pilih: ");
                algo = scanner.nextLine();
                algo = getAlgo(algo);
                if(algo == null){
                    System.out.println("Pilih algoritma yang tersedia!");
                }
            }while(algo == null);
            writer.write("Algoritma : " + algo + "\n");
            System.out.println("-----");

            if(!start.equalsIgnoreCase(target)){
                Algorithm game = new Algorithm(algo);
                long startTime = System.currentTimeMillis();
                List<String> res = game.routeSearch(graph, start, target);
                if(res != null){
                    output(writer, res, game, startTime);
                }else{
                    System.out.println("-----Pencarian tidak ditemukan-----");
                    System.out.println("Durasi pencarian : " + (System.currentTimeMillis() - startTime) + "ms");
                    System.out.println("-----");
                }
            }else{
                long startTime = System.currentTimeMillis();
                List<String> res = new ArrayList<>();
                res.add(start);
                Algorithm game = new Algorithm(algo);
                game.addVisitedList(start);
                output(writer, res, game, startTime);
            }

            System.out.print("Lagi? (y/n): ");
            lagi = scanner.nextLine();
        }while(lagi.equalsIgnoreCase("y"));
    } catch (IOException e){
        e.printStackTrace();
    }
    System.out.println("Terima Kasih sudah bermain...");
}

```

```

public static void scrap(){
    Integer num;
    String dict;
    System.out.print("Jumlah suku kata: ");
    num = Integer.parseInt(scanner.nextLine());

    System.out.println("dictionary : ");
    System.out.println("1. Asistent");
    System.out.println("2. Pribadi");
    System.out.print("pilih (1 atau 2): ");
    dict = scanner.nextLine();
    if(dict.equalsIgnoreCase("1")){
        dict = "assistent";
    }else{
        dict = "prib";
    }

    List<String> lines = ReadFile.toList("../dict/"+dict+"/Dict.txt");
    Graph graph = Graph.buildGraph(num, lines);
    graph.saveToCSV("../assets/Dict/"+num+".txt");
}

public static void main(String[] args) {

    boolean valid = true;
    do{
        System.out.print("Masukkan perintah (Scrap atau Run): ");
        String inputString = scanner.nextLine();

        if(inputString.equalsIgnoreCase("Scrap")){
            long startTime = System.currentTimeMillis();
            scrap();
            System.err.println(System.currentTimeMillis()-startTime);
        }else if(inputString.equalsIgnoreCase("Run")){
            run();
        }else{
            System.out.println("Masukkan perintah yang valid!");
            System.out.println("");
            valid=false;
        }
    }while(!valid);

}
}

```

## Kelas ReadFile

Kelas ini bertugas untuk proses membaca file dari txt

```

public class ReadFile {

    public static List<String> toList(String path){
        BufferedReader reader = null;
        List<String> lines = new ArrayList<>();
        try {
            // Membuat objek FileReader
            FileReader fileReader = new FileReader(path);

            // Membuat objek BufferedReader
            reader = new BufferedReader(fileReader);

            String line;
            // Membaca file baris per baris dan menyimpannya dalam List
            while ((line = reader.readLine()) != null) {
                lines.add(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // Tutup BufferedReader setelah selesai membaca file
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return lines;
    }
}
}

```


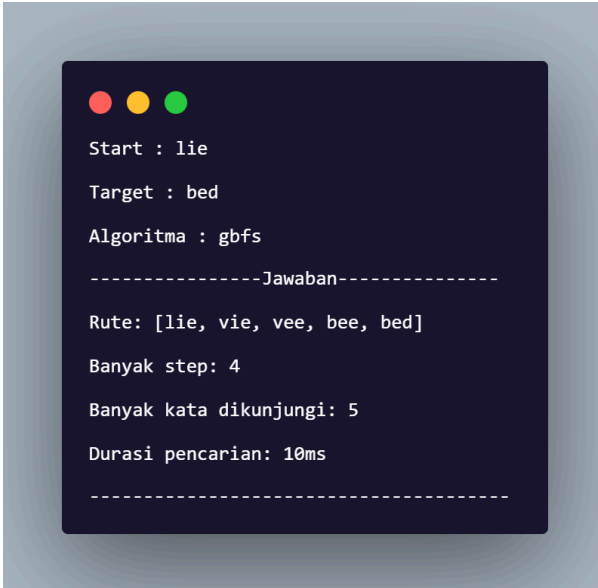
## Kelas Art



Berisi string dari ascii generator dan hasil modifikasi pribadi. Ini akan ditampilkan di awal program

[illegible]

## Hasil Uji

Untuk mempermudah dan mempercantik pengambilan hasil uji, saya menyimpan hasil program ke dalam folder test/demo.txt. Kemudian hasil tersebut saya *screenshot* menggunakan *codesnap*. Jika Asisten merasa keberatan dengan hal tersebut, dapat dibuktikan dengan menjalankan programnya secara langsung.


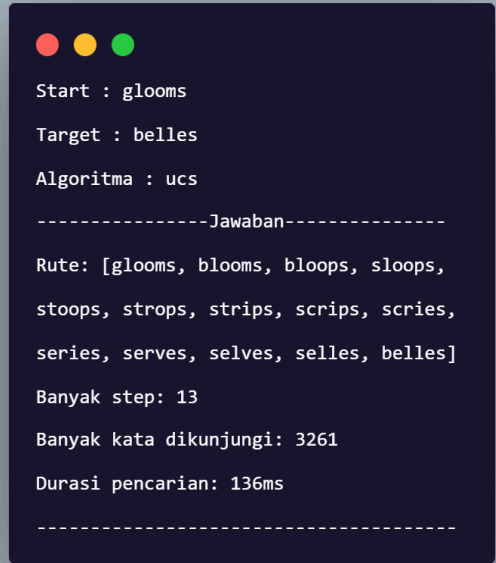
Uji	Algorithm	Gambar
1	UCS	 A screenshot of a terminal window with a dark background and light green text. The output shows the start and target words, the algorithm used (ucs), the path found (lie, lee, bee, bed), the number of steps (3), the number of words visited (286), and the search duration (72ms). The path is enclosed in brackets and the steps are listed as a sequence of words.
	GBFS	 A screenshot of a terminal window with a dark background and light green text. The output shows the start and target words, the algorithm used (gbfs), the path found (lie, vie, vee, bee, bed), the number of steps (4), the number of words visited (5), and the search duration (10ms). The path is enclosed in brackets and the steps are listed as a sequence of words.

	A*	 <pre>Start : lie Target : bed Algoritma : A* -----Jawaban----- Rute: [lie, lee, bee, bed] Banyak step: 3 Banyak kata dikunjungi: 163 Durasi pencarian: 22ms -----</pre>
2	UCS	 <pre>Start : loch Target : ness Algoritma : ucs -----Jawaban----- Rute: [loch, loth, lots, lets, nets, nes s] Banyak step: 5 Banyak kata dikunjungi: 2012 Durasi pencarian: 141ms -----</pre>

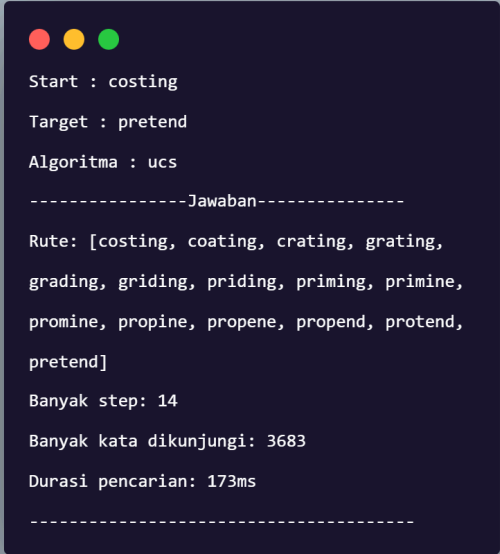

	<p>GBFS</p>	 <pre>Start : loch Target : ness Algoritma : gbfs -----Jawaban----- Rute: [loch, lech, pech, peccs, pets, nets, ness] Banyak step: 6 Banyak kata dikunjungi: 11 Durasi pencarian: 5ms -----</pre>
	<p>A*</p>	 <pre>Start : loch Target : ness Algoritma : A* -----Jawaban----- Rute: [loch, loth, lots, lets, less, nes s] Banyak step: 5 Banyak kata dikunjungi: 561 Durasi pencarian: 22ms -----</pre>

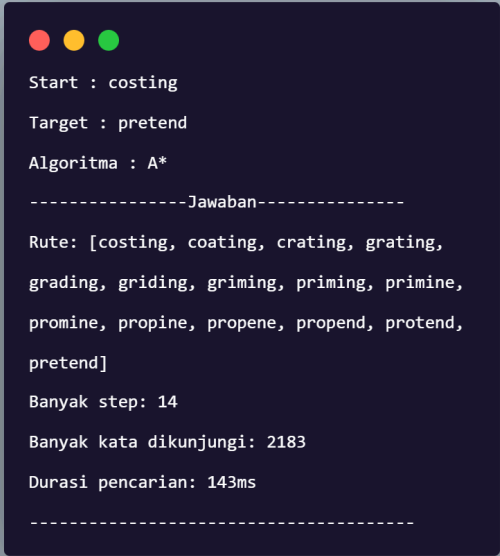

3	UCS	 <pre>Start : start Target : break Algoritma : ucs -----Jawaban----- Rute: [start, stark, shark, shack, whack, wrack, wreck, wreak, break] Banyak step: 8 Banyak kata dikunjungi: 3655 Durasi pencarian: 191ms -----</pre>
	GBFS	 <pre>Start : start Target : break Algoritma : gbfs -----Jawaban----- Rute: [start, stark, stork, stock, stick, spick, speck, speak, steak, steal, sheal, wheal, wheat, cheat, cleat, bleat, bleak, break] Banyak step: 17 Banyak kata dikunjungi: 39 Durasi pencarian: 4ms -----</pre>



	A*	 <pre> Start : start Target : break Algoritma : A* -----Jawaban----- Rute: [start, stark, shark, shack, whack, wrack, wreck, wreak, break] Banyak step: 8 Banyak kata dikunjungi: 1551 Durasi pencarian: 46ms ----- </pre>
4	UCS	 <pre> Start : glooms Target : belles Algoritma : ucs -----Jawaban----- Rute: [glooms, blooms, bloops, sloops, stoops, strops, strips, scrips, sries, series, serves, selves, selles, belles] Banyak step: 13 Banyak kata dikunjungi: 3261 Durasi pencarian: 136ms ----- </pre>

	<p>GBFS</p>	<div><div><div></div><div></div><div></div></div><div>Start : glooms</div><div>Target : belles</div><div>Algoritma : gbfs</div><div>-----Jawaban-----</div><div>Rute: [glooms, blooms, brooms, brooks, brocks, bricks, brinks, brines, brides, bredes, breves, beeves, peeves, pelves, selves, selles, belles]</div><div>Banyak step: 16</div><div>Banyak kata dikunjungi: 30</div><div>Durasi pencarian: 14ms</div><div>-----</div></div>
	<p>A*</p>	<div><div><div></div><div></div><div></div></div><div>Start : glooms</div><div>Target : belles</div><div>Algoritma : A*</div><div>-----Jawaban-----</div><div>Rute: [glooms, blooms, bloops, sloops, stoops, strops, strips, scrips, scries, series, serves, selves, selles, belles]</div><div>Banyak step: 13</div><div>Banyak kata dikunjungi: 2408</div><div>Durasi pencarian: 95ms</div><div>-----</div></div>

5	UCS	 <pre> Start : costing Target : pretend Algoritma : ucs -----Jawaban----- Rute: [costing, coating, crating, grating, grading, griding, priding, priming, primine, promine, propine, propene, propend, pretend, pretend] Banyak step: 14 Banyak kata dikunjugi: 3683 Durasi pencarian: 173ms ----- </pre>
	GBFS	 <pre> Start : costing Target : pretend Algoritma : gbfs -----Jawaban----- Rute: [costing, coating, crating, grating, gracing, tracing, tricing, pricing, priming, primine, promine, propine, propene, propend, protend, pretend] Banyak step: 15 Banyak kata dikunjugi: 738 Durasi pencarian: 15ms ----- </pre>

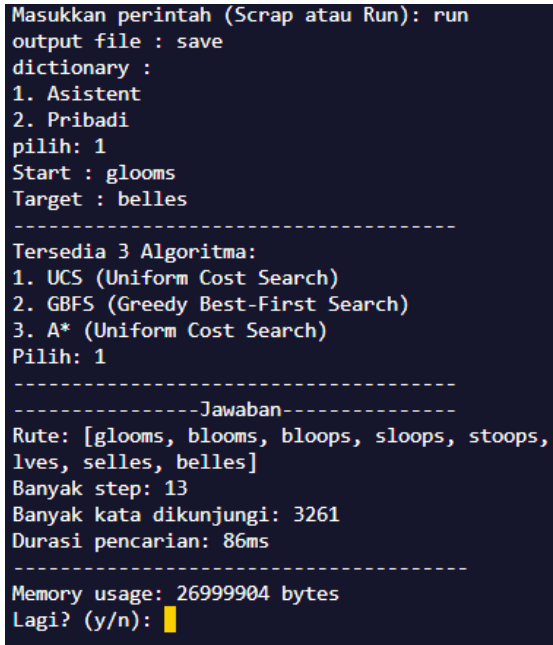
	A*	 <pre> Start : costing Target : pretend Algoritma : A* -----Jawaban----- Rute: [costing, coating, crating, grating, grading, griding, griming, priming, primine, promine, propine, propene, propend, pretend, pretend] Banyak step: 14 Banyak kata dikunjugi: 2183 Durasi pencarian: 143ms ----- </pre>
6	UCS	 <pre> Start : chanTERS Target : vaunTERS Algoritma : ucs -----Jawaban----- Rute: [chanTERS, chunTERS, shunTERS, saunTERS, vaunTERS] Banyak step: 4 Banyak kata dikunjugi: 61 Durasi pencarian: 8ms ----- </pre>

	<p>GBFS</p>	 <pre>Start : chanters Target : vaunters Algoritma : gbfs -----Jawaban----- Rute: [chanters, chatters, chitters, critters, fritters, flitters, slitters, spitters, sputters, shutters, shunters, saunters, vaunters] Banyak step: 12 Banyak kata dikunjungi: 37 Durasi pencarian: 3ms -----</pre>
	<p>A*</p>	 <pre>Start : chanters Target : vaunters Algoritma : A* -----Jawaban----- Rute: [chanters, chunters, shunters, saunters, vaunters] Banyak step: 4 Banyak kata dikunjungi: 30 Durasi pencarian: 2ms -----</pre>

## Analisis

Berdasarkan hasil analisis secara teoritis dan praktikal didapatkan beberapa kesimpulan sebagai berikut:

1. Dari segi keoptimalan solusi UCS dan A\* dijamin mendapatkan solusi optimal. Seperti yang sudah dijelaskan sebelumnya dan beberapa contoh yang sudah dilihat pada hasil uji. Sementara, GBFS tidak menjamin akan selalu menghasilkan solusi yang optimal.
2. Dari segi kecepatan eksekusi GBFS memiliki kecenderungan waktu eksekusi lebih singkat dari UCS maupun A\*. Pada kasus ini UCS bisa jadi memiliki waktu eksekusi paling lama dibanding GBFS maupun A\*.
3. Dari segi kebutuhan memori UCS memiliki kecenderungan untuk membutuhkan lebih banyak memori dibandingkan GBFS dan A\*. Hal ini disebabkan UCS pada kasus ini akan menelusuri secara BFS sehingga membutuhkan ruang yang lebih untuk menyimpan node pada priority queue. Sebaliknya, GBFS memiliki kecenderungan untuk lebih sedikit menggunakan memori. Hal ini disebabkan GBFS akan menelusuri node yang paling rendah cost nya sehingga akan banyak memotong pencarian node lainnya.
4. Berikut adalah *benchmark* dari setiap algoritma dengan uji yang sama

Algoritma	Gambar
UCS Panjang rute : 13 Waktu eksekusi: 86ms Penggunaan memori $\approx$ 25.75 MB	 <pre>Masukkan perintah (Scrap atau Run): run output file : save dictionary : 1. Asistent 2. Pribadi pilih: 1 Start : glooms Target : belles ----- Tersedia 3 Algoritma: 1. UCS (Uniform Cost Search) 2. GBFS (Greedy Best-First Search) 3. A* (Uniform Cost Search) Pilih: 1 ----- -----Jawaban----- Rute: [glooms, blooms, bloops, sloops, stoops, lves, selles, belles] Banyak step: 13 Banyak kata dikunjungi: 3261 Durasi pencarian: 86ms ----- Memory usage: 26999904 bytes Lagi? (y/n):</pre>

<p>GBFS</p> <p>Panjang rute: 16</p> <p>Waktu eksekusi 9ms</p> <p>Penggunaan memori <math>\approx</math> 23.14 MB</p>	<pre> Masukkan perintah (Scrap atau Run): run output file : save dictionary : 1. Asistent 2. Pribadi pilih: 1 Start : glooms Target : belles ----- Tersedia 3 Algoritma: 1. UCS (Uniform Cost Search) 2. GBFS (Greedy Best-First Search) 3. A* (Uniform Cost Search) Pilih: 2 ----- -----Jawaban----- Rute: [glooms, blooms, brooms, brooks, brocks, eves, peeves, pelves, selves, selles, belles] Banyak step: 16 Banyak kata dikunjungi: 30 Durasi pencarian: 9ms ----- Memory usage: 24264064 bytes Lagi? (y/n): </pre>
<p>A*</p> <p>Panjang rute: 13</p> <p>Waktu eksekusi: 59ms</p> <p>Penggunaan memori <math>\approx</math> 24.16 MB</p>	<pre> Masukkan perintah (Scrap atau Run): run output file : save dictionary : 1. Asistent 2. Pribadi pilih: 1 Start : glooms Target : belles ----- Tersedia 3 Algoritma: 1. UCS (Uniform Cost Search) 2. GBFS (Greedy Best-First Search) 3. A* (Uniform Cost Search) Pilih: 3 ----- -----Jawaban----- Rute: [glooms, blooms, bloops, sloops, sto lves, selles, belles] Banyak step: 13 Banyak kata dikunjungi: 2408 Durasi pencarian: 59ms ----- Memory usage: 25331712 bytes Lagi? (y/n): </pre>

Link Repository Github: [Click here](#)

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. <b>[Bonus]:</b> Program memiliki tampilan GUI		✓