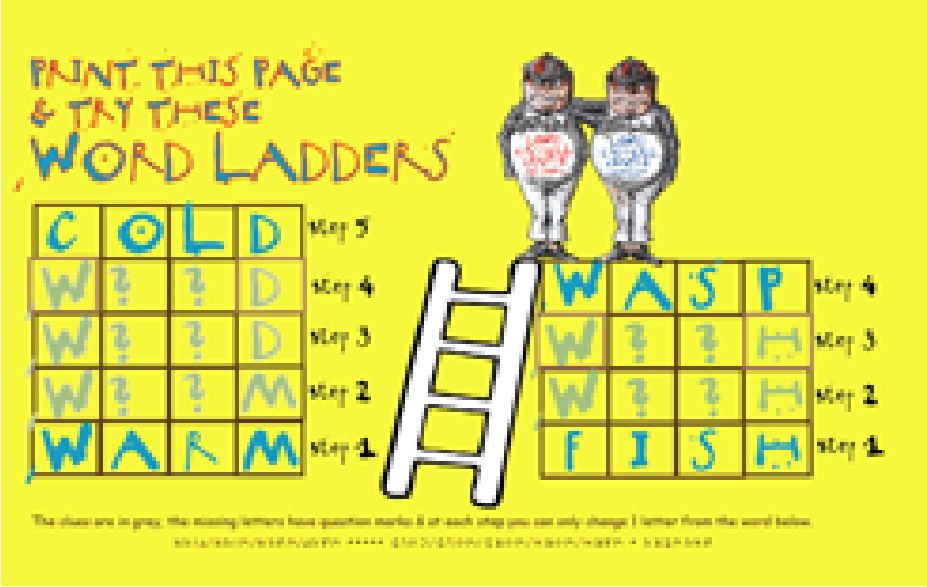
LAPORAN TUGAS BESAR

IF2211 Strategi Algoritma

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



Dipersiapkan oleh:

Pradipta Rafa Mahesa 13522162

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

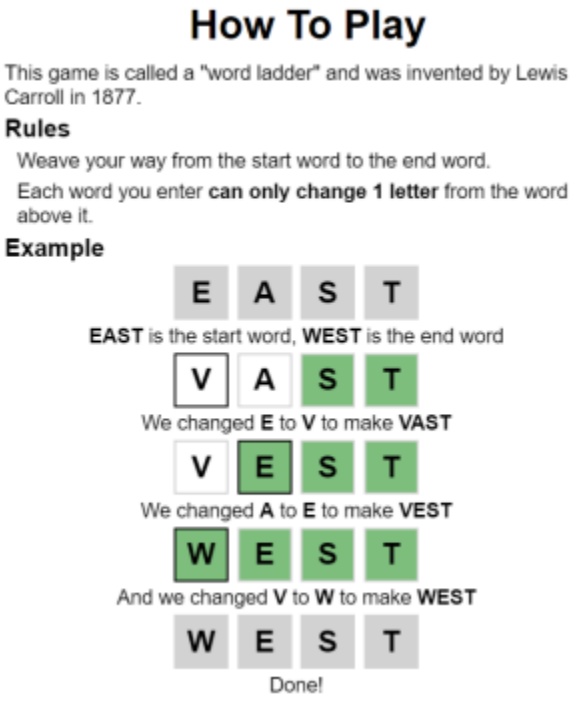
Jl. Ganesha 10, Bandung 4013

Daftar Isi

BAB I

Deskripsi Tugas

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.



Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder
(Sumber: https://wordwormdormdork.com/)

Spesifikasi Tugas Kecil 3

- Buatlah program 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*)

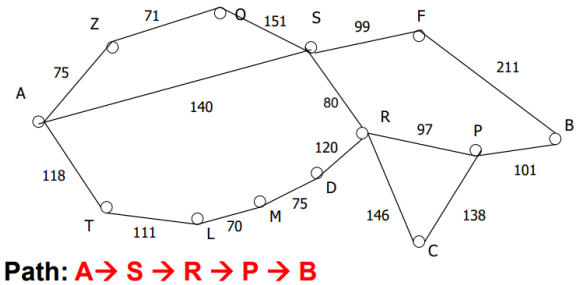
- Output :

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 2

Analisis Algoritma

3.1. Algoritma UCS



Path: A→ S → R → P → B
Path-cost = 418 → optimal solution

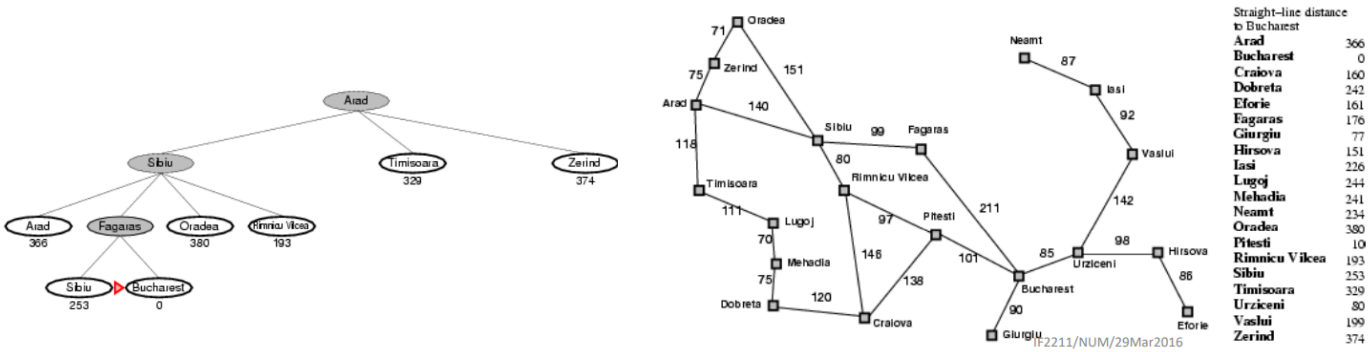
IF2211/NUM/2

Simpul-E	Simpul Hidup
A	Z_{A-75}, T_{A-118}, S_{A-140}
Z_{A-75}	T_{A-118}, S_{A-140}, O_{AZ-146}
T_{A-118}	S_{A-140}, O_{AZ-146}, L_{AT-229}
S_{A-140}	O_{AZ-146}, R_{AS-220}, L_{AT-229}, F_{AS-239}, O_{AS-291}
O_{AZ-146}	R_{AS-220}, L_{AT-229}, F_{AS-239}, O_{AS-291}
R_{AS-220}	L_{AT-229}, F_{AS-239}, O_{AS-291}, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$
L_{AT-229}	F_{AS-239}, O_{AS-291}, $M_{ATL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$
F_{AS-239}	O_{AS-291}, $M_{ATL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$, $B_{ASF-450}$
O_{AS-291}	$M_{ATL-299}$, $P_{ASR-317}$, $D_{ASR-340}$, $C_{ASR-366}$, $B_{ASF-450}$
$M_{ATL-299}$	$P_{ASR-317}$, $D_{ASR-340}$, $D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASF-450}$
$P_{ASR-317}$	$D_{ASR-340}$, $D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$D_{ASR-340}$	$D_{ATLM-364}$, $C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$D_{ATLM-364}$	$C_{ASR-366}$, $B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$C_{ASR-366}$	$B_{ASRP-418}$, $C_{ASRP-455}$, $B_{ASF-450}$
$B_{ASRP-418}$	Solusi ketemu

Algoritma UCS atau Uniform Cost Search adalah algoritma penentuan rute non heuristik yang menggunakan solusi optimal dengan memanfaatkan sebuah fungsi untuk menentukan biaya dari root ke sebuah node (g(n)). Algoritma ini bekerja dengan cara menghitung biaya yang diperlukan untuk menempuh suatu titik tetangga lalu memasukkannya ke dalam sebuah Priority Queue dimana biaya yang lebih rendah memiliki prioritas yang lebih tinggi. Lalu titik dengan prioritas paling tinggi akan dikunjungi dan di cek apakah titik tersebut merupakan tujuannya, apabila iya selesai tetapi apabila tidak maka akan dicek biaya tetangga-tetangga titik tersebut dan dimasukkan ke dalam PrioQueuenya. Hal tersebut diulang hingga ditemukan titik targetnya.

Untuk implementasi algoritma UCS pada tugas kali ini saya memanfaatkan algoritma Hamming Distance untuk menentukan biaya dari suatu titik ke titik lainnya. Namun karena terdapat batasan dari gamenya sendiri yang hanya memperbolehkan titik selanjutnya hanya berbeda 1 huruf dengan node sebelumnya, algoritma UCS akan lebih terlihat seperti algoritma BFS. Perbedaan algoritma UCS dan BFS sebenarnya hanya bahwa titik pada UCS memiliki biaya dan urutan pengecekkan titik didasarkan pada biaya tersebut. Tetapi karena batasan sebelumnya biaya dari suatu titik ke titik lainnya sudah pasti hanya 1, maka tidak akan terurut berdasarkan biaya tetapi lebih ke terurut berdasarkan pertama kali di cek, jadi seperti queue biasa.

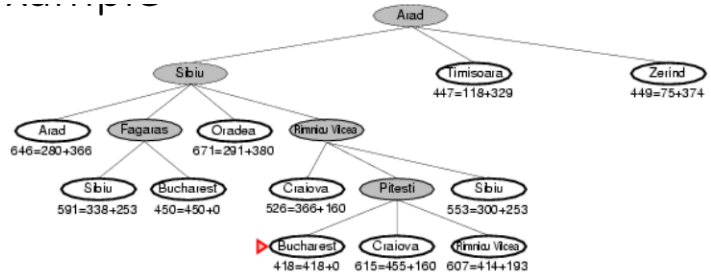
3.2. Algoritma GBFS

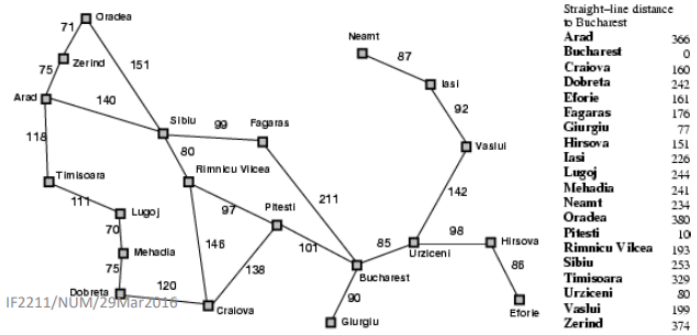


Algoritma GBFS (Greedy Best First Search) adalah algoritma penentuan rute heuristik yang memanfaatkan fungsi perhitungan biaya dari suatu titik ke titik tujuan (h(n)) untuk menentukan prioritas suatu titik. Algoritma GBFS hanya mengambil tetangga dengan prioritas paling tinggi dan mengabaikan tetangga lainya lalu menambahkannya ke dalam list berisi rute yang dilalui dengan syarat suatu titik tidak dapat dikunjungi lebih dari sekali walaupun biayanya paling rendah dan tidak dapat backtrack. Pengunjungan titik dilakukan hingga ketemu titik tujuan atau sudah tidak ada tetangga yang dapat dikunjungi.

Implementasi Algoritma GBFS pada tugas kali ini juga memanfaatkan Hamming Distance untuk menentukan biaya dari suatu titik. Biaya tersebut dihitung berdasarkan jarak antara titik yang ingin dikunjungi dan titik tujuan akhir. Jarak yang paling dekat akan menghasilkan biaya paling kecil sehingga memiliki prioritas paling tinggi. Namun karena keterbatasan dari kamus dan tidak bolehnya backtrack, kadang-kadang algoritma GBFS tidak menghasilkan solusi optimal atau bahkan tidak menemukan solusi sama sekali. Hal tersebut terjadi karena selain berdasarkan biaya, prioritas dari suatu titik juga ditentukan berdasarkan urutan ditemukan suatu titiknya yang pada kasus ini berdasarkan posisi kata tersebut pada kamus. Dalam kamus mungkin terdapat beberapa kata yang biaya untuk dikunjunginya sama tetapi kata untuk solusi optimalnya tidak dikunjungi pertama kali, sehingga Algoritma GBFSnya mengabaikannya untuk saat itu dan malah melanjutkan ke kata yang mestinya tidak terdapat dalam solusi optimal.

3.3. *Algoritma A**





Algoritma A* adalah algoritma penentuan rute yang memanfaatkan 2 fungsi (g(n) + h(n)) yaitu fungsi menentukan biaya untuk mencapai suatu titik dari titik awal (g(n)) dan fungsi untuk menentukan biaya estimasi dari suatu titik ke titik tujuannya (h(n)). Estimasi total dari kedua fungsi tersebut adalah fungsi yang digunakan oleh algoritma a* untuk menentukan biaya dari suatu titik (f(n)). Algoritma A* juga menggunakan PrioQueue untuk menentukan urutan pengunjungan node dengan biaya total yang diperoleh dari fungsi f(n) sebagai penentu prioritasnya (lebih kecil biaya , prioritas lebih tinggi). Secara Teori karena algoritma memanfaatkan aspek dari kedua algoritma UCS dan GBFS, A* dapat berjalan dengan efisiensi lebih tinggi daripada UCS dan memiliki tingkat konsistensi solusi optimal yang sama dengan UCS. Hal tersebut terjadi sebagian besar karena penggunaan Priority Queue untuk menemukan urutan pengunjungan node yang paling optimal dibandingkan dengan UCS yang menggunakan urutan FIFO (first in first out).

Pada Tugas Besar kali ini saya memanfaatkan Hamming Distance untuk fungsi perhitungan jarak algoritma A* dengan jarak totalnya adalah jarak kata awal ke kata saat ini ditambah jarak kata saat ini ke kata akhir. Lalu node kata tersebut akan dimasukkan ke PrioQueue berdasarkan prioritas jaraknya.

BAB 3

Source Code

4.1. Class DictInit

Kelas DictInit adalah kelas yang berfungsi sebagai menginisialisasi Kamus yang akan digunakan

No	Kode	Deskripsi
1	`public static Map<Integer, List<String>> init(String dict){` `Map<Integer, List<String>> wordLists = new HashMap<Integer, List<String>>();` `String filename = "";` `switch(dict){` `case "Dictionary Asisten":` `filename = "src\\Dict_Asisten.txt";` `break;` `case "Github Dictionary":` `filename = "src\\WordList.txt";` `break;` `case "Basic Dictionary 2":` `filename = "src\\WordList2.txt";` `break;` `case "Collins Scrabble Words (2019)":` `filename = "src\\Collins Scrabble Words (2019).txt";` `break;` `}` `try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {` `String line;` `while ((line = reader.readLine()) != null) {` `line = line.trim();` `int length = line.length();` `List<String> wordList = wordLists.getOrDefault(length, new ArrayList<String>());` `wordList.add(line);` `wordLists.put(length, wordList);` `}` `} catch (IOException e) {` `e.printStackTrace();` `}` `return wordLists;`	Method init berfungsi untuk menginisialisasi map of dengan keynya adalah panjang kata (integer) dan valuenya list of string berupa list dari kata dengan panjang tertentu yang terdapat dalam kamus pilihan.

4.2. Class StringNode dan NodeComparator

Kelas StringNode merupakan suatu kelas yang memiliki 2 atribut yaitu list dan cost. List adalah path yang dilalui untuk mencapai node tersebut. Cost adalah nilai yang diberikan kepada node tersebut berupa integer. Kelas ini digunakan untuk algoritma A* sebagai element yang dimasukkan ke dalam PrioQueuenya. Kelas NodeComparator merupakan kelas yang meng implement method untuk membandingkan 2 StringNode berdasarkan costnya.

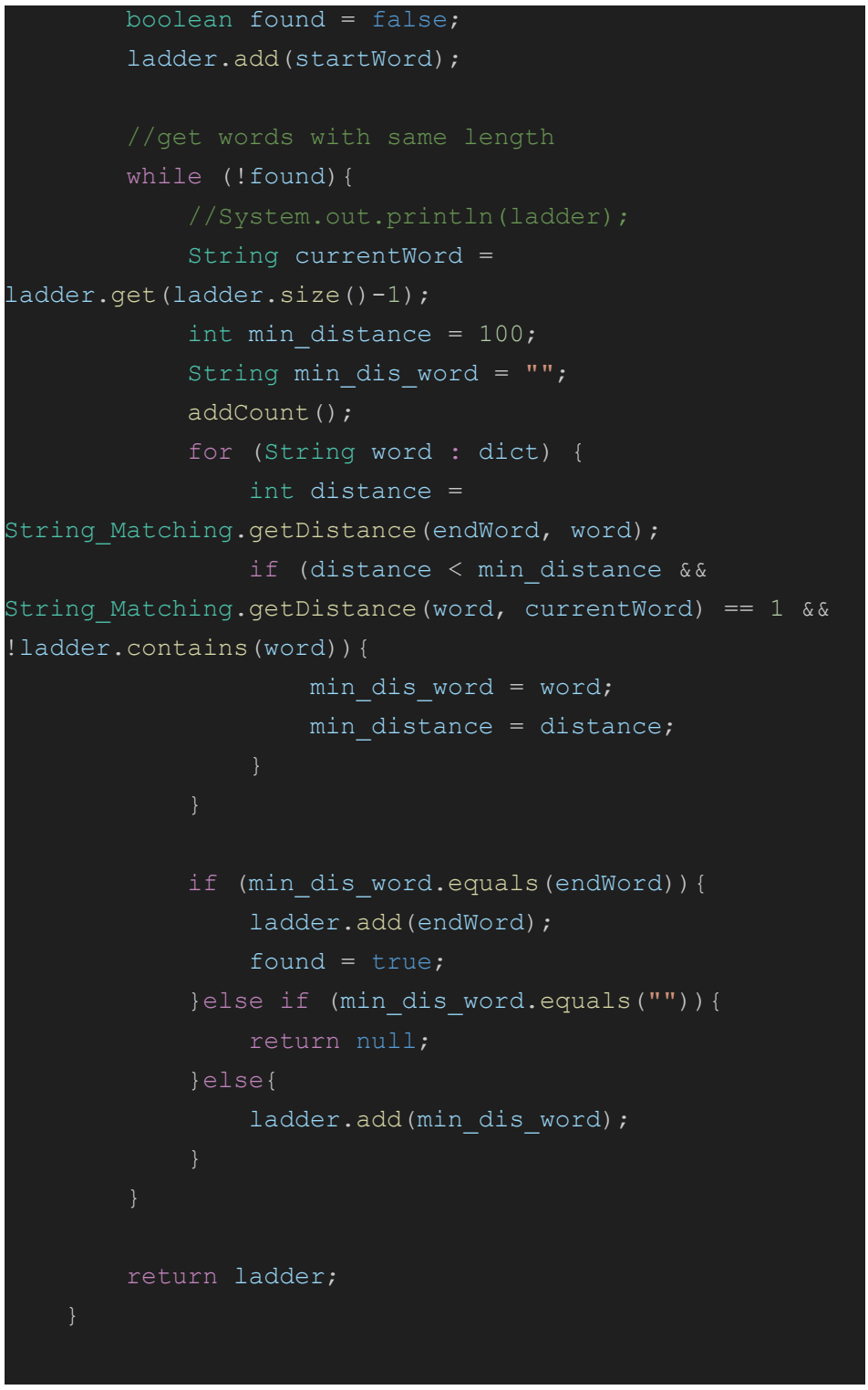
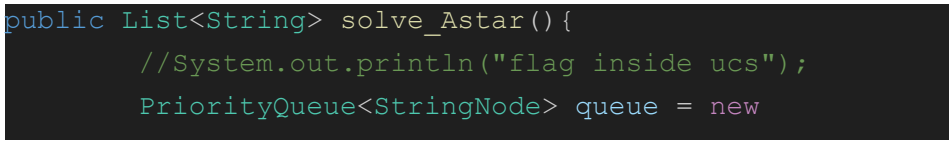
No	Kode	Deskripsi
1	`public static Map<Integer, List<String>> init(String dict){public StringNode(List<String> list, int cost){ this.list=list; this.cost = cost; }`	Method Constructor
2	`public List<String> getList(){ return this.list; }`	Method untuk mengembalikan List dari string yang terdapat dalam suatu String Node
3	`public int getCost(){ return this.cost; }`	Method untuk mengembalikan Cost dari suatu StringNode
4	`public class NodeComparator implements Comparator<StringNode> { @Override public int compare(StringNode o1, StringNode o2) { return Integer.compare(o1.getCost(), o2.getCost()); } }`	Method compare untuk membandingkan suatu StringNode dengan StringNode lainnya. Digunakan untuk pengurutan PrioQueue

4.3. Class Solver

Kelas Solver merupakan kelas yang memiliki 4 atribut yaitu startWord, endWord, count, dan dict dan method-method algoritma mencari solusi (UCS, GBFS, A*). startWord adalah kata awal dan endWord adalah kata tujuan. Count adalah jumlah node yang dikunjungi dan dict adalah list of string yang digunakan sebagai Dictionary.

No	Kode	Deskripsi
1	`public Solver(String startWord,String endWord,List<String> dict){` `this.startWord = startWord.toLowerCase();` `this.endWord = endWord.toLowerCase();` `this.count = 0;` `this.dict = dict;` `}`	Method Constructor
2	`public void addCount(){` `this.count++;` `}`	Method untuk menambah count
3	`public Integer getCount(){` `return count;` `}`	Method untuk mengembalikan Count
4	`public static int getDistance(String string1,String string2){ //Menggunakan Hamming Distance` `int value = 0;` `for (int i = 0; i<string1.length();i++){` `if (string1.charAt(i) != string2.charAt(i)){` `value++;` `}` `}` `return value;` `}`	Method untuk mencari nilai jarak antara 2 string menggunakan Hamming Distance
5	`public List<String> solve_UCS(){` `//System.out.println("flag inside ucs");` `Queue<List<String>> queue = new`	Method mencari solusi dengan algoritma UCS

	LinkedList<List<String>>(); List<String> ladder = new ArrayList<String>(); ladder.add(startWord); queue.offer(ladder); //System.out.println("is queue empty "+queue.isEmpty()); Map<String, Boolean> visited = new HashMap<>(); while (!queue.isEmpty()){ addCount(); List<String> currentList = queue.remove(); //System.out.println(currentList); String currentWord = currentList.get(currentList.size()-1); visited.put(currentWord,true); for (String word : dict) { if (!currentList.contains(word) && String_Matching.getDistance(word,currentWord) == 1 && !visited.containsKey(word)){ List<String> temp = new ArrayList<String>(currentList); temp.add(word); queue.offer(temp); if (word.equals(endWord)){ //System.out.println("found flag"); return temp; } } } } return null; }	
6	public List<String> solve_Greedy(){ List<String> ladder = new ArrayList<String>();	Method mencari solusi dengan algoritma Greedy Best First Search

		
7		Method mencari solusi dengan algoritma A*

```
PriorityQueue<>(new NodeComparator());
        List<String> ladder = new ArrayList<String>();
        ladder.add(startWord);
        StringNode start = new
StringNode(ladder,String_Matching.getDistance(startWord,
endWord));
        queue.add(start);
        //System.out.println("is queue empty
"+queue.isEmpty());
        Map<String, Boolean> visited = new HashMap<>();
        while (!queue.isEmpty()){
            addCount();
            StringNode currentNode = queue.remove();
            List<String> currentList =
currentNode.getList();
            Integer currentcost = currentNode.getCost();
            //System.out.println(currentList);
            String currentWord =
currentList.get(currentList.size()-1);
            visited.put(currentWord,true);
            for (String word : dict) {
                if (!currentList.contains(word) &&
String_Matching.getDistance(word,currentWord) == 1 &&
!visited.containsKey(word)){
                    List<String> temp_list = new
ArrayList<String>(currentList);
                    temp_list.add(word);
                    Integer temp_cost = currentcost +
String_Matching.getDistance(word, endWord);
                    queue.add(new
StringNode(temp_list,temp_cost));
                    if (word.equals(endWord)){
                        //System.out.println("found
flag");
                        return temp_list;
                    }
                }

            }
```

	<pre>} return null; }</pre>	

4.4. Class Main

Kelas Main adalah kelas utama tempat berjalannya program

No	Kode	Deskripsi
1	<pre>public static String[] InputWords(Scanner scanner, Map<Integer, List<String>> dicts){ String[] words = {"",""}; System.out.print("Enter start word: "); words[0] = scanner.nextLine().toLowerCase(); System.out.print("Enter target word: "); words[1] = scanner.nextLine().toLowerCase(); if (words[0].length() != words[1].length()) { throw new IllegalArgumentException(new Throwable("DifferenLengthError")); }else if(words[0].equals(words[1])){ throw new IllegalArgumentException(new Throwable("SameWordError")); } List<String> dict = dicts.get(words[0].length()); if(!dict.contains(words[0])){ throw new IllegalArgumentException(new Throwable("StartNotInDictError")); }else if (!dict.contains(words[1])){ throw new IllegalArgumentException(new</pre>	Method untuk menerima inputan 2 kata sebagai startword dan targetword.

	`Throwable("TargetNotInDictError"));` `        }` `        return words;` `    }`	
2	`public static void change_dict(){` `        //System.out.println("Current Dictionary : " + Dictionary);` `        switch (Dictionary){` `            case "Github Dictionary":` `                Dictionary = "Collins Scrabble Words (2019)";` `                break;` `            case "Collins Scrabble Words (2019)":` `                Dictionary = "Basic Dictionary 2";` `                break;` `            case "Basic Dictionary 2":` `                Dictionary = "Dictionary Asisten";` `                break;` `            case "Dictionary Asisten":` `                Dictionary = "Github Dictionary";` `                break;` `        }` `        System.out.println("Changed into : "+Dictionary);` `    }`	Method untuk mengubah dictionary yang digunakan
3	`public static void Solve(int Algorithm, String startWord, String targetWord,Map<Integer, List<String>> dicts){` `        List<String> dict = dicts.get(startWord.length());` `        List<String> path = new ArrayList<String>();` `        Solver solution = new`	Method tempat memanggil fungsi dari kelas solver dan menampilkan jawaban.

	`` Solver(startWord,targetWord,dict); long startTime = System.currentTimeMillis(); switch(Algorithm){ case 1: path = solution.solve_UCS(); break; case 2: path = solution.solve_Greedy(); break; case 3: path = solution.solve_Astar(); break; } long endTime = System.currentTimeMillis(); if (path != null && !path.isEmpty()){ System.out.println("Path :"); int i = 1; for (String word : path) { System.out.println(i++ +"). "+word); } }else{ System.out.println("Not found"); } System.out.println("Nodes Passed:" + solution.getCount()); System.out.println("Runtime: " + (endTime-startTime) + " milliseconds"); } ```	
4	``` public static void main(String[] args){ Map<Integer, List<String>> dicts = DictInit.init(Dictionary); String[] words = {"",""}; boolean end = false; Scanner scanner = new Scanner(System.in); System.out.println("-------------------------"); ```	Main loop tempat program berjalan

```
        System.out.println("--- Word Ladder Solver ---");
        System.out.println("--------------------------");
        System.out.println("");
        System.out.println("---------- INFO ----------");
        System.out.println("Start Word : "+words[0]);
        System.out.println("Target Word : "+words[1]);
        System.out.println("Current Dictionary : "+Dictionary);
        System.out.println("");
        System.out.println("-------- COMMANDS --------");
        System.out.println("1). DICT (change dictionary)");
        System.out.println("2). WORD (change start Word and target Word)");
        System.out.println("3). UCS (Solve using UCS algorithm)");
        System.out.println("4). GREEDY (Solve using Greedy Best First Search
Algorithm)");
        System.out.println("5). A* (Solve using A* Algorithm)");
        System.out.println("6). END (End Program)");
        System.out.println("7). INFO (Displays Current Start,Target, and
Dictionary)");
        System.out.println("8). HELP (Displays This Commands List)\n");

        while (!end){
            System.out.print(">>> ");
            String input;
            input = scanner.nextLine().toUpperCase();
            switch (input){
                case "DICT":
                    change_dict();
                    dicts = DictInit.init(Dictionary);
                    break;
                case "WORD":
                    try{
                        words = InputWords(scanner,dicts);
                    }catch (IllegalArgumentException e) {
                        switch(e.getCause().getMessage()){
                            case "DifferenLengthError":
                                System.out.println("ERROR! , Start and Target
must be same length\n");
                                break;
                            case "SameWordError":
                                System.out.println("ERROR! , Start Word must
be different from Target\n");
                                break;
                            case "StartNotInDictError":
                                System.out.println("ERROR! , Start Word not in
dictionary\n");
                                break;
                            case "TargetNotInDictError":
                                System.out.println("ERROR! , Target Word not
in dictionary\n");
                                break;
                            default :
                                System.out.println("Error!");
                                break;
                        }
                    }
```

```
                    break;
                case "UCS":
                    if (words[0].equals("")){
                        System.out.println("Mohon input Start Word dan Target
Word terlebih dahulu\n");
                    }
                    else{
                        Solve(1, words[0], words[1],dicts);
                    }
                    break;
                case "GREEDY":
                    if (words[0].equals("")){
                        System.out.println("Mohon input Start Word dan Target
Word terlebih dahulu\n");
                    }
                    else{
                        Solve(2, words[0], words[1],dicts);
                    }
                    break;
                case "A*":
                    if (words[0].equals("")){
                        System.out.println("Mohon input Start Word dan Target
Word terlebih dahulu\n");
                    }
                    else{
                        Solve(3, words[0], words[1],dicts);
                    }
                    break;
                case "END":
                    end = true;
                    System.out.println("CLOSING PROGRAM");
                    break;
                case "INFO":
                    System.out.println("");
                    System.out.println("---------- INFO ----------");
                    System.out.println("Start Word : "+words[0]);
                    System.out.println("Target Word : "+words[1]);
                    System.out.println("Current Dictionary : "+Dictionary);
                    System.out.println("");
                    break;
                case "HELP":
                    System.out.println("");
                    System.out.println("-------- COMMANDS --------");
                    System.out.println("1). DICT (change dictionary)");
                    System.out.println("2). WORD (change start Word and target
Word)");
                    System.out.println("3). UCS (Solve using UCS algorithm)");
                    System.out.println("4). GREEDY (Solve using Greedy Best
First Search Algorithm)");
                    System.out.println("5). A* (Solve using A* Algorithm)");
                    System.out.println("6). END (End Program)");
                    System.out.println("7). INFO (Displays Current
Start,Target, and Dictionary)");
                    System.out.println("8). HELP (Displays This Commands
List)");
```

```
                    System.out.println("");
                    break;
                default:
                    System.out.println("That command does not exists");
                    System.out.println("Try HELP to see commands list");
                    System.out.println("");
                    break;
            }

        }
        scanner.close();

    }
```

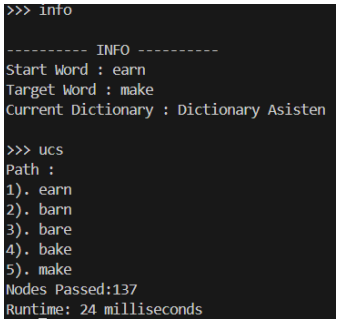
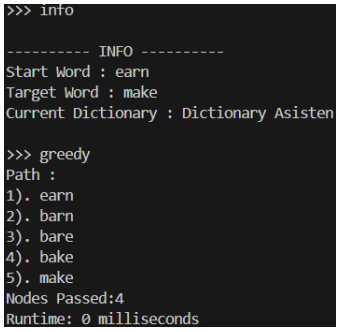
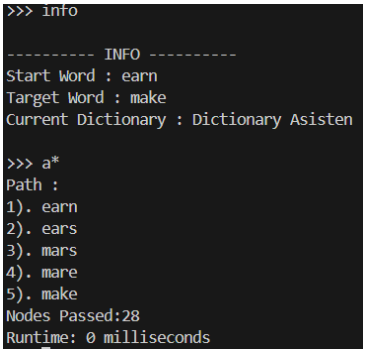
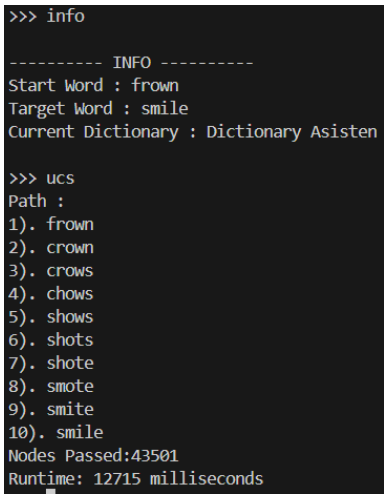
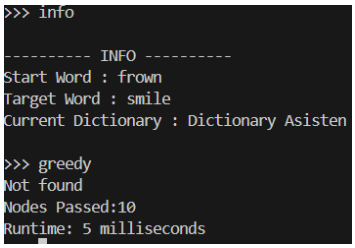
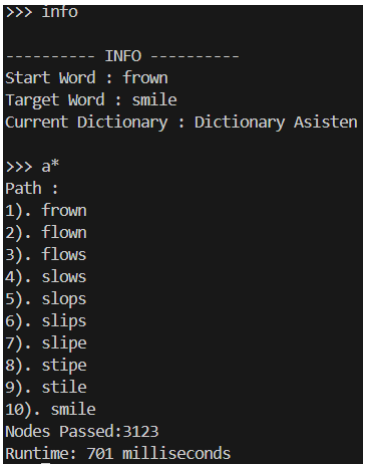
BAB 4

Test Case

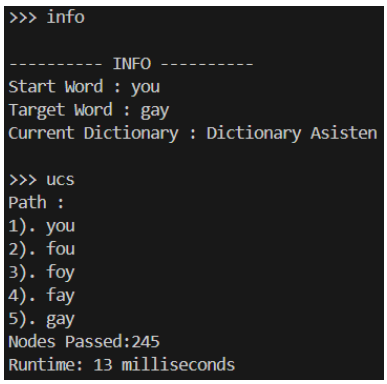
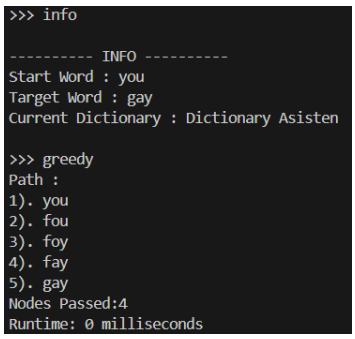
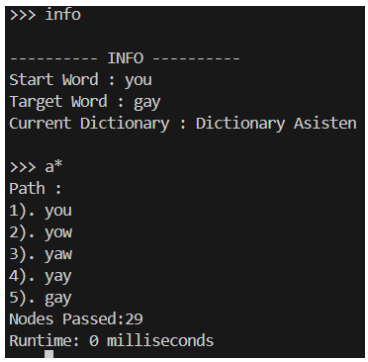
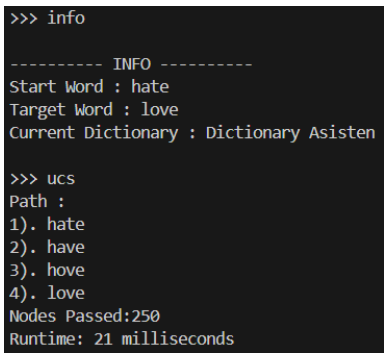
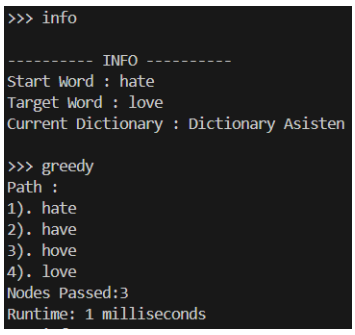
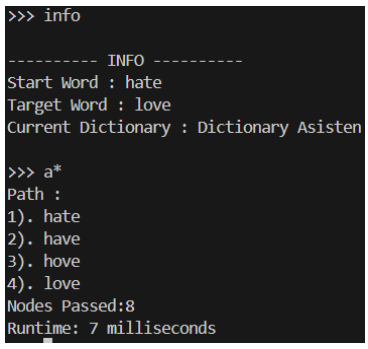
4.1. Test Case

Test Case dilakukan dengan Kamus Asisten yang diberikan asisten melalui QnA.

Tabel 4.1.1 tabel screenshot test case

No	Word	UCS	GBFS	A*
1	earn -> make			
2	frown -> smile			

3	ladder -> miners	>>> info ---------- INFO ---------- Start Word : ladder Target Word : miners Current Dictionary : Dictionary Asisten >>> ucs Path : 1). ladder 2). lander 3). lancer 4). lances 5). rances 6). ranees 7). rakees 8). rakers 9). lakers 10). likers 11). liners 12). miners Nodes Passed:115111 Runtime: 66619 milliseconds	>>> info ---------- INFO ---------- Start Word : ladder Target Word : miners Current Dictionary : Dictionary Asisten >>> greedy Not found Nodes Passed:27 Runtime: 2 milliseconds	>>> info ---------- INFO ---------- Start Word : ladder Target Word : miners Current Dictionary : Dictionary Asisten >>> a* Path : 1). ladder 2). lander 3). lancer 4). lances 5). rances 6). ranees 7). razees 8). razers 9). mazers 10). maters 11). miters 12). miners Nodes Passed:15568 Runtime: 7149 milliseconds
4	blacks -> nagger	>>> info ---------- INFO ---------- Start Word : blacks Target Word : nagger Current Dictionary : Dictionary Asisten >>> ucs Path : 1). blacks 2). slacks 3). slicks 4). slices 5). saices 6). sauces 7). saucer 8). sauger 9). sagger 10). nagger Nodes Passed:24466 Runtime: 12099 milliseconds	>>> info ---------- INFO ---------- Start Word : blacks Target Word : nagger Current Dictionary : Dictionary Asisten >>> greedy Path : 1). blacks 2). blanks 3). blinks 4). blinds 5). blends 6). bleeds 7). bleeps 8). sleeps 9). sleeks 10). cleeks 11). cheeks 12). checks 13). chicks 14). chicos 15). chinos 16). chines 17). chides 18). chider 19). cheder 20). chewer 21). chawer 22). chafer 23). chaser 24). chased 25). ceased 26). ceases 27). censes 28). censer 29). center 30). canter 31). banter 32). banger 33). bagger 34). nagger Nodes Passed:33 Runtime: 0 milliseconds	>>> info ---------- INFO ---------- Start Word : blacks Target Word : nagger Current Dictionary : Dictionary Asisten >>> a* Path : 1). blacks 2). slacks 3). slicks 4). slices 5). saices 6). sauces 7). saucer 8). sauger 9). sagger 10). nagger Nodes Passed:1745 Runtime: 722 milliseconds

5	you -> gay			
6	hate -> love			

4.2. Analisis Test Case

Tabel 4.2.1 tabel data dari test case

No	Banyak Node pada Solusi			Banyak Node dikunjungi			Runtime(ms)		
	UCS	GBFS	A*	UCS	GBFS	A*	UCS	GBFS	A*
1	5	5	5	137	4	28	24	0	0
2	10	-	10	43501	10	3123	12715	5	701
3	12	-	12	115111	27	15568	66619	2	7149
4	10	34	10	24466	33	1745	12099	0	722
5	5	5	5	245	4	29	13	0	0
6	4	4	4	250	3	8	21	1	7

a. Banyak Node pada Solusi

Berdasarkan data banyak node pada solusi dari setiap algoritma dapat dilihat bahwa algoritma UCS dan A* sudah dapat mendapatkan solusi yang optimal dengan konsisten walaupun mungkin hasil akhirnya ada kata yang berbeda. Selain itu dapat dilihat juga bahwa untuk algoritma GBFS masih terdapat kasus dimana solusinya tidak optimal (test case 4) bahkan ada yang tidak ketemu solusinya (test case 2&3). Hal tersebut sesuai dengan pembahasan pada Analisis Algoritma. Hal unik lainnya adalah apabila solusi GBFS optimal maka solusi UCS pasti sama dengan solusi GBFS.

b. Banyak Node dikunjungi

Berdasarkan data banyak node dikunjungi dari setiap algoritma dapat dilihat beberapa hal. Pertama algoritma GBFS selalu memiliki jumlah node dikunjungi yang paling sedikit, hal ini terjadi karena algoritma GBFS hanya mengecek 1 node per depth maka dari itu banyak node dikunjungi bagi algoritma GBFS sebenarnya adalah kedalaman yang ditempuh oleh algoritma tersebut. Dari data juga dapat terlihat bahwa banyak node dikunjungi oleh GBFS selalu sama dengan banyak node pada solusi dikurangi 1 karena 1 tersebut adalah node awal.

Berdasarkan data dapat dilihat bahwa banyak node yang dikunjungi oleh A* jauh lebih sedikit dibandingkan dengan UCS bahkan bisa sampai 10x lebih sedikit. Hal tersebut berarti algoritma A* menggunakan memory yang lebih sedikit dibandingkan dengan algoritma UCS.

c. Runtime

Berdasarkan data dapat dilihat bahwa waktu eksekusi paling cepat sering kali diperoleh oleh algoritma GBFS. Hal tersebut dikarenakan banyak node yang dikunjungi GBFS jauh lebih sedikit dibandingkan UCS dan A*. Selain itu dapat dilihat juga perbandingan antara UCS dan A* dimana A* secara rata-rata memiliki waktu eksekusi yang jauh lebih rendah dibandingkan UCS dan bahkan hampir menyaingi GBFS. Hal tersebut terjadi karena jumlah node yang dikunjungi A* juga lebih sedikit dari UCS.

d. Kesimpulan

Secara keseluruhan algoritma A* adalah algoritma yang paling optimal, efisien, dan konsisten dari segi solusi, memori, dan waktu eksekusi jika dibandingkan dengan kedua algoritma lainnya. Namun pada kasus tertentu dimana algoritma GBFS bisa mendapatkan solusi yang optimal, algoritma GBFS akan jauh lebih efisien dalam segi memori dan waktu eksekusi dibandingkan UCS dan A*. Hanya saja tingkat konsistensi dalam menemukan solusi optimal tersebut rendah. Algoritma UCS kurang efisien dalam memori dan waktu eksekusi tetapi solusinya sudah terjamin konsistensi kebenarannya.

Lampiran

Repository : Github

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

Daftar Pustaka

Munir, Rinaldi. “Penentuan rute (Route/Path Planning).” *Informatika*, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf. Accessed 5 May 2024.

Munir, Rinaldi. “Penentuan rute (Route/Path Planning).” *Informatika*, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf. Accessed 5 May 2024.

https://github.com/dwyl/english-words