

LAPORAN TUGAS KECIL III
IF2211 STRATEGI ALGORITMA
PENYELESAIAN PERMAINAN WORD LADDER
MENGGUNAKAN ALGORITMA UCS, GREEDY BEST FIRST
SEARCH, DAN A*



Disusun oleh:

Thea Josephine Halim

13522012

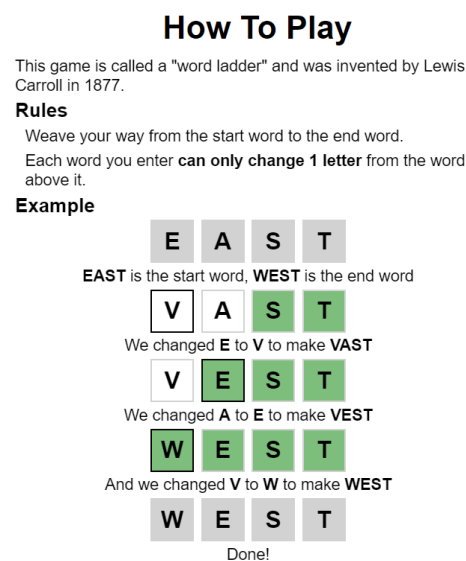
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

DAFTAR ISI.....	2
A. Deskripsi Persoalan.....	3
B. Konsep/Ide dan Algoritma.....	3
1. Uniform Cost Search (UCS).....	3
2. Greedy Best First Search (Greedy BFS).....	4
3. A* Algorithm.....	4
C. Source Code.....	4
1. Main Program (CLI).....	4
2. Algorithm.....	6
3. Dictionary.....	7
4. Result.....	7
5. Fungsi Lain.....	8
6. GUI.....	9
D. Contoh Output.....	12
1. UCS.....	12
2. Greedy Best First Search.....	18
3. A-star (A*).....	25
4. Invalid Input.....	31
5. GUI.....	32
E. Analisis.....	33
F. Checklist.....	38
G. Links.....	39

A. Deskripsi Persoalan

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/>)

B. Konsep/Ide dan Algoritma

Pada persoalan kali ini, kita akan menggunakan 3 macam algoritma: UCS, A*, dan Greedy BFS. Ketiganya akan memilih node dengan cost terkecil, tetapi dengan cara penghitungan cost yang berbeda-beda sehingga mempengaruhi pemilihan pathnya. Sebuah cost di antara node dapat kita wakikan sebagai jumlah step untuk mencapai goal sehingga jarak setiap node pastilah satu. Nilai heuristik sebuah node ke goal node dihitung menggunakan konsep *hamming distance*.

1. Uniform Cost Search (UCS)

Algoritma ini pada dasarnya memiliki konsep yang sama seperti BFS, tetapi dengan memperhitungkan *cost* pada setiap pergerakannya. Pada UCS perhitungan cost untuk memilih jalur dilakukan dengan mencari $g(n)$ yaitu total *actual cost* dari node

awal hingga node yang sekarang dicek. Perhitungan berdasarkan *actual cost* ini memberikan jaminan bahwa solusi yang didapatkan optimal (*cost minimum*). Akan disediakan sebuah priority queue untuk menyimpan daftar simpul hidup. Mulai dari simpul pertama, akan dilakukan ekspansi node tetangga dan di-*enqueue* pada antrian simpul hidup. Nodes akan dimasukkan ke antrian berdasarkan *cost* aktualnya, dimulai dari *cost* terkecil hingga terbesar. Kemudian node dengan *cost* terkecil akan di-*dequeue* untuk diperiksa apakah nilainya sama dengan yang kita cari, jika tidak, akan dilakukan ekspansi kembali. Hal ini dilakukan berulang-ulang hingga sebuah solusi ditemukan atau antrian menjadi kosong (tidak ditemukan solusi). Apabila node akhir sudah ditemukan, akan dilakukan *backtracking* untuk melacak path yang telah dilalui hingga mencapai node akhir tersebut.

2. Greedy Best First Search (Greedy BFS)

Algoritma Greedy Best First Search, selanjutnya akan disebut GBFS sama seperti UCS, tetapi dengan priority queue simpul hidup diurutkan berdasarkan *cost heuristic*. *Cost heuristic* atau sering disebut dengan $h(n)$ adalah perkiraan *cost* dari node yang sekarang dicek hingga ke node tujuan. Penggunaan *heuristic cost* akan menyebabkan area pencarian semakin dekat ke node tujuan, sehingga secara teoritis akan meminimalkan jumlah node yang dibangkitkan dan mempercepat waktu untuk menemukan sebuah solusi. Akan tetapi, seperti orang yang serakah, GBFS akan mengambil semua path yang tampak pendek yang justru bukanlah salah satu solusi yang baik.

3. A* Algorithm

Algoritma A* (A-star) adalah kombinasi dari kedua algoritma sebelumnya. Pada A*, nilai *cost* akan berdasarkan nilai heuristik dan aktual $f(n) = g(n) + h(n)$, sehingga secara teori akan menjadikannya teroptimal. Nilai heuristik membantunya memilih jalan yang terefektif dan mengurangi jumlah node ekspan, sedangkan nilai aktual akan memastikan solusi yang didapat tetap optimal.

C. Source Code

1. Main Program (CLI)

Tabel 3.1.1 Main Program

Kode	Keterangan
<pre> 1 public static void main(String[] args){ 2 boolean stop = false; 3 while (!stop) { 4 System.out.println("Welcome to Word Ladder Solver ^^"); 5 Dict wordDictionary = new Dict("Dictionary.txt"); 6 Scanner scanner = new Scanner(System.in); 7 String start; 8 String finish; 9 while (true) { 10 System.out.println("-----"); 11 System.out.println("Enter the start and finish word"); 12 System.out.print("Start: "); 13 start = scanner.next().toUpperCase(); 14 System.out.print("Finish: "); 15 finish = scanner.next().toUpperCase(); 16 // testing in here 17 // List<String> neighbors = Ucs.generateNeighbors(start,wordDictionary); 18 // neighbors.forEach(System.out::println); 19 if (start.length()==finish.length()){ 20 if (start.equals(finish)){ 21 System.out.println("Both cannot be the same word"); 22 } 23 else if (!wordDictionary.containsWord(finish)&&!wordDictionary.containsWord(start)){ 24 System.out.println("Both words do not exist in the dictionary."); 25 } 26 else if (!wordDictionary.containsWord(start)){ 27 System.out.println("Start word does not exist in the dictionary."); 28 } 29 else if (!wordDictionary.containsWord(finish)){ 30 System.out.println("Finish word does not exist in the dictionary."); 31 } 32 else{ 33 break; 34 } 35 } 36 else{ 37 System.out.println("Both needs to be the same length."); 38 } 39 } 40 System.out.println(""); 41 while (true) { 42 System.out.println("Choose your algorithm to solve:"); 43 System.out.println("1. Uniform Cost Search"); 44 System.out.println("2. Greedy Best First Search"); 45 System.out.println("3. A*"); 46 System.out.println("Enter '0' to quit."); 47 System.out.print(">>> "); 48 String input = scanner.next(); 49 if(input.equals("1") input.equals("2") input.equals("3")){ 50 Result res; 51 if (input.equals("1")){ 52 System.out.println("You have chosen UCS algo!"); 53 res = UCS.findUCS(start, finish, wordDictionary); 54 printOutSol(res.getResultlist()); 55 System.out.println("Num of nodes checked: " + res.getnumofcheckednodes()); 56 System.out.println("Algorithm execution time: " + res.getexexecutiontime() + " seconds"); 57 break; 58 } 59 else if (input.equals("2")){ 60 System.out.println("You have chosen Greedy algo!"); 61 res = Greedy.findGreedy(start, finish, wordDictionary); 62 printOutSol(res.getResultlist()); 63 System.out.println("Num of nodes checked: " + res.getnumofcheckednodes()); 64 System.out.println("Algorithm execution time: " + res.getexexecutiontime() + " seconds"); 65 break; 66 } 67 else if (input.equals("3")){ 68 System.out.println("You have chosen A* algo!"); 69 res = AStar.findAStar(start, finish, wordDictionary); 70 printOutSol(res.getResultlist()); 71 System.out.println("Num of nodes checked: " + res.getnumofcheckednodes()); 72 System.out.println("Algorithm execution time: " + res.getexexecutiontime() + " seconds"); 73 break; 74 } 75 } 76 else if (input.equals("0")){ 77 System.out.println("Quitting..."); 78 break; 79 } 80 else{ 81 System.out.println("Wrong input. Retrying..."); 82 continue; 83 } 84 } 85 } 86 } 87 while (true){ 88 System.out.println("Do you want to search for another (Y/N)?"); 89 System.out.print(">>> "); 90 String ans = scanner.next(); 91 if (ans.equals("N")){ 92 stop = true; 93 System.out.println("Goodbye!"); 94 scanner.close(); 95 break; 96 } 97 else if (ans.equals("Y")){ 98 break; 99 } 100 else if (!ans.equals("Y")){ 101 System.out.println("Wrong input"); 102 } 103 } 104 } 105 }</pre>	<p>Algoritma main untuk CLI</p>

<pre> 1 public static void printOutSol(List<String> solution){ 2 if (solution != null) { 3 System.out.println("Solution found: "); 4 solution.forEach(System.out::println); 5 System.out.println("The shortest solution length: " + solution.size()); 6 } else { 7 System.out.println("No solution found."); 8 } 9 } </pre>	Print semua node dari solusi yang ditemukan
---	---

2. Algorithm

Tabel 3.2.1 Kode Semua Algoritma

Kode	Keterangan
<pre> 1 public static Result findPath(String start, String end, List dictionary, String algorithmtype){ 2 long startTime = System.nanoTime(); // Record start time 3 Result r1; 4 PriorityQueue<Node> queue = new PriorityQueue<>((Comparator.comparingInt(node -> node.getCost()))); 5 Set<String> visited = new HashSet<>(); 6 int numofcheckednodes = 0; 7 queue.offer(new Node(start, null, 0)); 8 while (!queue.isEmpty()) { 9 Node currentNode = queue.poll(); 10 numofcheckednodes++; 11 12 // found 13 if ((currentNode.getWord().equals(end)){ 14 long endTime = System.nanoTime(); // Record end time 15 double elapsedTimeInSeconds = (endTime - startTime) / 1e9; // Calculate elapsed time in seconds 16 r1 = new Result(functions.backtrackPath(currentNode),numofcheckednodes,elapsedTimeInSeconds); 17 return r1; 18 } 19 20 // not found continue, add to queue from currentNode 21 for (String neighbor : functions.generateNeighbors(currentNode.getWord(),dictionary)) { 22 if (!visited.contains(neighbor)) { 23 visited.add(neighbor); 24 int cost = 0; 25 if(algorithmtype.equals("GBFS")){ 26 cost = functions.hamming_distance(neighbor,end) ; 27 } 28 else if (algorithmtype.equals("UCS")){ 29 cost = currentNode.getCost() + 1; 30 } 31 else if (algorithmtype.equals("Astar")){ 32 cost = functions.hamming_distance(neighbor,end) + currentNode.getCost() + 1; 33 } 34 queue.offer(new Node(neighbor, currentNode, cost)); 35 } 36 } 37 } 38 long endTime = System.nanoTime(); // Record end time 39 double elapsedTimeInSeconds = (endTime - startTime) / 1e9; // Calculate elapsed time in seconds 40 r1 = new Result(null, numofcheckednodes, elapsedTimeInSeconds); 41 return r1; 42 } </pre>	Ketiga algoritma dijadikan sebagai satu fungsi karena cara kerja yang hampir sama, dengan cost yang berbeda berdasarkan parameter tipe algoritma.

3. Dictionary

Tabel 3.3.1 Kode Kelas Dictionary

Kode	Keterangan
<pre> 1 public class Dict { 2 private Set<String> dictionary; 3 public Dict(String filename){ 4 dictionary = new HashSet<>(); 5 loadWords(filename); 6 } 7 private void loadWords(String filename){ 8 try (BufferedReader br = new BufferedReader(new FileReader(filename))) { 9 String line; 10 while ((line = br.readLine()) != null) { 11 dictionary.add(line.trim()); 12 } 13 } catch (IOException e) { 14 e.printStackTrace(); 15 } 16 } 17 public boolean containsWord(String word) { 18 return dictionary.contains(word.toUpperCase()); 19 } 20 }</pre>	<p>Kelas Dictionary memiliki atribut set of words yang digunakan untuk validasi apakah sebuah kata terdapat pada dictionary. Fungsi <i>loadWords</i> akan membaca semua isi dictionary.txt yang tersedia dan memasukkan semuanya kedalam set dictionary.</p>

4. Result

Tabel 3.4.1 Kode Kelas Result

Kode	Keterangan
<pre> 1 public class Result { 2 private List<String> resultlist; 3 private int numofcheckednodes; 4 private double executiontime; 5 public Result(List<String> resultlist, int numofcheckednodes, double elapsedTimeInSeconds){ 6 this.resultlist = resultlist; 7 this.numofcheckednodes = numofcheckednodes; 8 this.executiontime = elapsedTimeInSeconds; 9 } 10 11 public int getnumofcheckednodes() { 12 return numofcheckednodes; 13 } 14 15 public double getexecutiontime() { 16 return executiontime; 17 } 18 19 public List<String> getResultlist() { 20 return resultlist; 21 } 22 } 23 }</pre>	<p>Kelas Result terdiri dari List of words yang menjadi solusi, jumlah node yang dicek (<i>numofcheckednodes</i>), dan waktu eksekusi algoritma (<i>executiontime</i>).</p>

5. Fungsi Lain

Tabel 3.5.1 Fungsi lain


Kode	Keterangan
<pre> 1 public static List<String> generateNeighbors(String word, Dict dictionary){ 2 List<String> neighbors = new ArrayList<>(); 3 char[] charArray = word.toCharArray(); 4 for (int i = 0; i < charArray.length; i++) { 5 char originalChar = charArray[i]; 6 for (char c = 'a'; c <= 'z'; c++) { 7 if (c != originalChar) { 8 charArray[i] = c; 9 String neighbor = new String(charArray).toUpperCase(); 10 11 // Check if the neighbor is in the dictionary and not equal to the original word 12 if (dictionary.containsWord(neighbor) && !(neighbor.toUpperCase().equals(word.toUpperCase()))) { 13 neighbors.add(neighbor); 14 } 15 } 16 } 17 charArray[i] = originalChar; 18 } 19 return neighbors; 20 } 21 </pre>	<p>Fungsi <i>generateNeighbors</i> untuk menghasilkan list of strings kata-kata yang bisa dicapai dari sebuah kata.</p> <p>Parameter kata akan diganti hurufnya satu per satu mulai dari slot pertama hingga terakhir sambil melakukan pengecekan dictionary untuk kevalidan kata. Apabila kata tersebut valid, kata tersebut akan ditambahkan ke dalam list neighbors.</p>
<pre> 1 public static void printPrioQueue(PriorityQueue<Node> x){ 2 for (Node node : x) { 3 System.out.println(node.getWord().toString()); 4 } 5 } </pre>	<p>Fungsi <i>printPrioQueue</i> akan mencetak seluruh isi queue untuk membantu debugging dan cek queue simpul hidup</p>
<pre> 1 public static List<String> backtrackPath(Node node){ 2 List<String> result = new ArrayList<>(); 3 Node currentNode = node; 4 while (currentNode!=null) { 5 result.add(0,currentNode.getWord()); 6 currentNode = currentNode.getParent(); 7 } 8 return result; 9 } </pre>	<p>Untuk melakukan “backtrack” atau membangun solusi path yang ditemukan dengan cara mengakses atribut parent sambil menambahkan hasil result path hingga atribut parent adalah null (kata pertama).</p>

 <pre> 1 public static int hamming_distance(String word, String goal){ 2 int dist = 0; 3 for (int i = 0; i<word.length(); i++){ 4 if (word.charAt(i) != goal.charAt(i)) { 5 dist++; 6 } 7 } 8 return dist; 9 } </pre>	<p>Fungsi <i>hamming_distance</i> untuk menghitung jarak heuristik dari currentnode ke node akhir berdasarkan perbedaan huruf.</p>
--	--

6. GUI

Untuk implementasi GUI kali ini akan digunakan Java Swing, toolkit GUI di Java. Pertama-tama akan dibuat sebuah window dengan ukuran 400x420 dan lokasi diset tidak ditentukan atau null. Kemudian dengan menggunakan komponen JTextField dan JLabel, akan dibuat sebuah form sederhana yang menerima input kata dari user. Pada kelas ini juga akan diinisialisasi dahulu dictionary dengan menginstansiasi *wordDictionary*. Tombol *findButton* untuk pencarian akan diset untuk memanggil fungsi *findWordLadder*. Fungsi tersebut akan melakukan pengecekan kalau-kalau terjadi salah input atau *edge cases*, lalu menampilkan pop up pilihan dengan tiga tombol pilihan algoritma. Sesuai dengan algoritma pilihan user, fungsi *findPath* dari kelas Algorithm akan dipanggil. GUI akan menampilkan hasil solusi yang *scrollable* dan detailnya pada label bawah window.

Tabel 3.6.1 GUI

Kode	Keterangan
 <pre> 1 import javax.swing.*; 2 import java.awt.*; 3 import java.awt.event.*; 4 5 public class WordLadderGUI extends JFrame { 6 private JTextField startField, endField; 7 private JButton findButton; 8 private JList<String> wordList; 9 private DefaultListModel<String> listModel; 10 private JLabel resultLabel; 11 private Dict wordDictionary; 12 13 public WordLadderGUI() { 14 setTitle("Word Ladder Solver :3"); 15 setSize(400, 420); 16 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); 17 setLocationRelativeTo(null); 18 19 20 JPanel inputPanel = new JPanel(); 21 inputPanel.setLayout(new GridLayout(3, 2)); 22 23 JLabel startLabel = new JLabel("Start Word:"); 24 startField = new JTextField(); 25 JLabel endLabel = new JLabel("End Word:"); 26 endField = new JTextField(); 27 findButton = new JButton("Let's Go!!"); 28 findButton.addActionListener(new ActionListener() { 29 public void actionPerformed(ActionEvent e) { 30 findWordLadder(); 31 } 32 }); 33 34 inputPanel.add(startLabel); 35 inputPanel.add(startField); 36 inputPanel.add(endLabel); 37 inputPanel.add(endField); 38 39 listModel = new DefaultListModel<>(); 40 wordList = new JList<>(listModel); 41 42 JScrollPane scrollPane = new JScrollPane(wordList); 43 44 resultLabel = new JLabel(); 45 resultLabel.setHorizontalAlignment(SwingConstants.CENTER); 46 47 getContentPane().setLayout(null); 48 49 inputPanel.setBounds(10, 10, 350, 90); 50 scrollPane.setBounds(10, 110, 350, 200); 51 resultLabel.setBounds(10, 320, 350, 50); 52 findButton.setBounds(250, 320, 110, 30); 53 54 getContentPane().add(inputPanel); 55 getContentPane().add(scrollPane); 56 getContentPane().add(resultLabel); 57 getContentPane().add(findButton); 58 59 wordDictionary = new Dict("Dictionary.txt"); 60 } 61 </pre>	<p>Mengatur tampilan GUI awal dan melakukan pembacaan dictionary</p>

```

1 private void findWordLadder() {
2     String start = startField.getText().trim().toUpperCase();
3     String end = endField.getText().trim().toUpperCase();
4     long res;
5     long startTime = System.nanoTime();
6     // Exceptions
7     if (start.length() != end.length()) {
8         JOptionPane.showMessageDialog(this, "Both words need to be the same length.",
9             "Error", JOptionPane.ERROR_MESSAGE);
10        return;
11    }
12    if (start.equals(end)) {
13        JOptionPane.showMessageDialog(this, "Start and end words cannot be the same.",
14            "Error", JOptionPane.ERROR_MESSAGE);
15        return;
16    }
17    if (!wordDictionary.containsWord(start) && !wordDictionary.containsWord(end)) {
18        JOptionPane.showMessageDialog(this, "Both words do not exist in the dictionary.",
19            "Error", JOptionPane.ERROR_MESSAGE);
20        return;
21    }
22    if (!wordDictionary.containsWord(start)) {
23        JOptionPane.showMessageDialog(this, "Start word does not exist in the dictionary.",
24            "Error", JOptionPane.ERROR_MESSAGE);
25        return;
26    }
27    if (!wordDictionary.containsWord(end)) {
28        JOptionPane.showMessageDialog(this, "End word does not exist in the dictionary.",
29            "Error", JOptionPane.ERROR_MESSAGE);
30        return;
31    }
32
33    String[] options = {"Uniform Cost Search", "Greedy Best First Search", "A*"};
34    int selectedOption = JOptionPane.showOptionDialog(this, "Choose algorithm to solve:",
35        "Algorithm Selection",
36        JOptionPane.DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
37    String algorithmName;
38    switch (selectedOption) {
39        case 0:
40            res = Algorithm.findPath(start, end, wordDictionary, "UCS");
41            algorithmName = "Uniform Cost Search";
42            break;
43        case 1:
44            res = Algorithm.findPath(start, end, wordDictionary, "GBFS");
45            algorithmName = "Greedy Best First Search";
46            break;
47        case 2:
48            res = Algorithm.findPath(start, end, wordDictionary, "Astar");
49            algorithmName = "A*";
50            break;
51        default:
52            return;
53    }
54    long endTime = System.nanoTime();
55    double elapsedTimeInSeconds = (endTime - startTime) / 1e9;
56    // Print res
57    if (res.getResultList() != null) {
58        listModel.clear();
59        for (String word : res.getResultList()) {
60            listModel.addElement(word);
61        }
62        // Set result label text
63        resultLabel.setBounds(0, 320, 250, 50);
64        resultLabel.setText("<html><div style='text-align: left;'>" + "Length: " + res.getResultList().size()
65            + "<br>Nodes checked: " + res.getnumofcheckednodes() + "<br>Execution time: "
66            + elapsedTimeInSeconds + " seconds</div></html>");
67    } else {
68        JOptionPane.showMessageDialog(this, "No solution found for " + algorithmName + ". \nNodes checked: "
69            + res.getnumofcheckednodes() + ", \nExecution time: " + elapsedTimeInSeconds + " seconds",
70            "Error", JOptionPane.ERROR_MESSAGE);
71    }
72 }
73

```

Fungsi *findWordLadder* melakukan pengecekan exceptions dari input yang dimasukkan. Memanggil algoritma pencarian solusi sesuai dengan button yang ditekan user lalu menampilkan hasilnya. Jika terjadi exception akan memunculkan pop up error, sedangkan tidak ditemukannya solusi akan menampilkan pop up tidak ada solusi dan detailnya.

```

1 public static void main(String[] args) {
2     SwingUtilities.invokeLater(new Runnable() {
3         public void run() {
4             try {
5                 UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
6             } catch (Exception e) {
7                 e.printStackTrace();
8             }
9         }
10
11         WordLadderGUI gui = new WordLadderGUI();
12         gui.setVisible(true);
13     });
14 }

```

Memanggil GUI yang telah dibuat dan mengatur tampilan, dan membuat jendela GUI muncul.

D. Contoh Output

Catatan: Akan digunakan input test case yang sama pada ketiga algoritma (kecuali invalid input) untuk perbandingan.

1. UCS

Tabel 4.1.1 Uji Coba UCS

No.	Masukan	Keluaran
1.	Start: HEAD Finish: TAIL	<pre>----- Enter the start and finish word Start: HEAD Finish: TAIL Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: HEAD HEAL HEIL HAIL TAIL The shortest solution length: 5 Num of nodes checked: 1351 Algorithm execution time: 0.11179 seconds</pre>
2.	Start: ATE Finish: BAT	<pre>----- Enter the start and finish word Start: ATE Finish: BAT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: ATE ATT AIT BIT BAT The shortest solution length: 5 Num of nodes checked: 454 Algorithm execution time: 0.0085076 seconds</pre>

3.	Start: JELLY Finish: SUPER	<pre> ----- Enter the start and finish word Start: JELLY Finish: SUPER Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: JELLY JELLS TELLS TELES TULES TUBES TUBER SUBER SUPER The shortest solution length: 9 Num of nodes checked: 4711 Algorithm execution time: 0.1630977 seconds </pre>
4.	Start: TALENT Finish: FALCON	<pre> ----- Enter the start and finish word Start: TALENT Finish: FALCON Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! No solution found. Num of nodes checked: 1 Algorithm execution time: 1.257E-4 seconds </pre>

5.	Start: FARMING Finish: FARMERS	<pre> Enter the start and finish word Start: FARMING Finish: FARMERS Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: FARMING FARTING MARTING MARTINS MATTINS LATTINS LATTENS PATTENS PATTERS POTTERS PORTERS PORKERS FORKERS FORMERS FARMERS The shortest solution length: 15 Num of nodes checked: 3948 Algorithm execution time: 0.1456031 seconds </pre>
6.	Start: REACT Finish: FLASK	<pre> ----- Enter the start and finish word Start: REACT Finish: FLASK Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: REACT REACH ROACH COACH CLACH CLASH FLASH FLASK The shortest solution length: 8 Num of nodes checked: 589 Algorithm execution time: 0.0147071 seconds </pre>

7.	Start: FLASK Finish: REACT	<pre> ----- Enter the start and finish word Start: FLASK Finish: REACT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: FLASK FLACK CLACK CLACH COACH ROACH REACH REACT The shortest solution length: 8 Num of nodes checked: 2353 Algorithm execution time: 0.0952973 seconds </pre>
8.	Start: APE Finish: MAN	<pre> ----- Enter the start and finish word Start: APE Finish: MAN Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: APE OPE OPT OAT MAT MAN The shortest solution length: 6 Num of nodes checked: 645 Algorithm execution time: 0.0110038 seconds </pre>

9.	Start: CHARGE Finish: COMEDO	<pre>----- Enter the start and finish word Start: CHARGE Finish: COMEDO Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 1 You have chosen UCS algo! Solution found: CHARGE CHANGE CHANGS CHANTS CHINTS CHINES CHINED COINED COINER CONNER CONGER CONGES CONIES CONINS CONING HONING HOMING HOMINY HOMILY HOMELY COMELY COMEDY COMEDO The shortest solution length: 23 Num of nodes checked: 8713 Algorithm execution time: 0.2684817 seconds</pre>
----	---------------------------------	--

10.	Start: ATLASES Finish: CABARET	<pre> >> 1 You have chosen UCS algo! Solution found: ATLASES ANLASES ANLACES UNLACES UNLACED UNLADED UNFADED UNFAKED UNCAKED UNCAGES UNCASES UNEASES UREASES CREASES CRESSES TRESSES TRASSES TRASHES BRASHES BRASHER BRASIER BRAKIER BEAKIER PEAKIER PECKIER PICKIER DICKIER DICKIES HICKIES HACKIES HACKLES HECKLES DECKLES DECILES DEFILES DEFILED REFILED REVILED REVELED RAVELED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED TAPERED TABERED TABORED TABORET TABARET CABARET The shortest solution length: 53 Num of nodes checked: 7973 Algorithm execution time: 0.2611142 seconds </pre>
-----	-----------------------------------	---

11.	Start: QUIRKING Finish: WRATHING	<pre> You have chosen UCS algo! Solution found: QUIRKING QUIRTING QUILTING QUILLING QUELLING DUELLING DWELLING SWELLING SHELLING SHEALING SHOALING SHOOLING SHOOTING SHOTTING SLOTING SLATING BLATING BLASTING BOASTING COASTING COACTING COACHING COUCHING MOUCHING MOUTHING SOUTHING SOOTHING TOOTHING TROTHING TRITHING WRITHING WRATHING The shortest solution length: 32 Num of nodes checked: 881 Algorithm execution time: 0.0385831 seconds </pre>
-----	-------------------------------------	---

2. Greedy Best First Search

Tabel 4.2.1 Uji Coba Greedy Best First Search

No.	Masukan	Keluaran
1.	Start: HEAD Finish: TAIL	<pre> Enter the start and finish word Start: HEAD Finish: TAIL Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: HEAD HEAL HEIL HAIL TAIL The shortest solution length: 5 Num of nodes checked: 6 Algorithm execution time: 0.0606741 seconds </pre>

2.	Start: ATE Finish: BAT	<pre> ----- Enter the start and finish word Start: ATE Finish: BAT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: ATE ATT APT OPT OAT BAT The shortest solution length: 6 Num of nodes checked: 9 Algorithm execution time: 7.698E-4 seconds </pre>
3.	Start: JELLY Finish: SUPER	<pre> ----- Enter the start and finish word Start: JELLY Finish: SUPER Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: JELLY BELLY BULLY SULLY SURLY SURFY SURFS SURAS SORAS SORES SORER SURER SUPER The shortest solution length: 13 Num of nodes checked: 35 Algorithm execution time: 0.0051161 seconds </pre>
4.	Start: TALENT Finish: FALCON	<pre> ----- Enter the start and finish word Start: TALENT Finish: FALCON Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! No solution found. Num of nodes checked: 1 Algorithm execution time: 1.764E-4 seconds </pre>

5.	Start: FARMING Finish: FARMERS	<pre> >> 2 You have chosen Greedy algo! Solution found: FARMING HARMING HARMINS HARPINS HARPIES HARRIES PARRIES PERRIES FERRIES FERLIES FELLIES FELLOES FELLOWS FALLOWS WALLOWS WALLOPS GALLOPS GALLONS BALLONS BALLOTS BALLETS BALLERS FALLERS FALTERS FILTERS FILMERS FIRMERS FARMERS The shortest solution length: 28 Num of nodes checked: 72 Algorithm execution time: 0.0051164 seconds </pre>
6.	Start: REACT Finish: FLASK	<pre> ----- Enter the start and finish word Start: REACT Finish: FLASK Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: REACT REACH LEACH LEASH LEASE FEASE FEAST BEAST BLAST CLAST CLASH FLASH FLASK The shortest solution length: 13 Num of nodes checked: 22 Algorithm execution time: 8.814E-4 seconds </pre>

7.	Start: FLASK Finish: REACT	<pre> ----- Enter the start and finish word Start: FLASK Finish: REACT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: FLASK FLACK CLACK CLACH COACH ROACH REACH REACT The shortest solution length: 8 Num of nodes checked: 13 Algorithm execution time: 4.113E-4 seconds </pre>
8.	Start: APE Finish: MAN	<pre> ----- Enter the start and finish word Start: APE Finish: MAN Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 2 You have chosen Greedy algo! Solution found: APE OPE OPT OAT MAT MAN The shortest solution length: 6 Num of nodes checked: 7 Algorithm execution time: 2.087E-4 seconds </pre>

9.	Start: CHARGE Finish: COMEDO	<pre> You have chosen Greedy algo! Solution found: CHARGE CHANGE CHANGS CHANTS CHINTS CHINES CHINED COINED COTLED COOLED COOEED COOEES COOERS CODERS CODENS CODONS COLONS COLINS CONINS CONING COMING HOMING HOMINY HOMILY HOMELY COMELY COMEDY COMEDO The shortest solution length: 28 Num of nodes checked: 251 Algorithm execution time: 0.0109798 seconds </pre>
10.	Start: ATLASES Finish: CABARET	<pre> You have chosen Greedy algo! Solution found: ATLASES ANLASES ANLACES UNLACES UNLACED UNLADED UNFADED UNFAKED UNBAKED UNBASED UNCASED UNCASES UNEASES UREASES CREASES CREASED CREAKED CROAKED CLOAKED CLONKED CLUNKED CLUNKER CLINKER CLICKER CLICKED CLICHED </pre>

		<div>CLICHES CLOCHES COOCHES COACHES COACHED COACTED COASTED COASTER CHASTER CHARTER CHARMER CHARMED CHASMED CHASSED CLASSED CLASSER CLASHER PLASHER PLASTER PLANTER PLANTED SLANTED SCANTED SCANTER SCATTER SCUTTER SPUTTER</div> <div>SPOTTER SPOTTED SPATTED SPATHED SWATHED SWATHER SLATHER SLATIER PLATIER PEATIER PEAKIER PERKIER PORKIER CORKIER COCKIER ROCKIER ROOKIER ROOKIES COOKIES COOLIES COLLIES CULLIES CULLIED SULLIED SALLIED TALLIED TALLIER</div>
--	--	---

		<div>TALKIER TACKIER TACKLER CACKLER CACKLES HACKLES HECKLES DECKLES DECILES DEFILES DEFILER DEFINER DEFINED REFINED REFIRED RETIRED RETIRES RETINES RATINES RAVINES RAVINED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED TAPERED TABERED</div> <div>TABORED TABORET TABARET CABARET The shortest solution length: 114 Num of nodes checked: 3441 Algorithm execution time: 0.1458964 seconds</div>
--	--	--

11.	Start: QUIRKING Finish: WRATHING	<pre> >> 2 You have chosen Greedy algo! Solution found: QUIRKING QUIRTING QUILTING QUILLING QUELLING DUELLING DWELLING SWELLING SWILLING SWIRLING SKIRLING SKIRTING SPIRTING SPITTING SPATTING SLATTING BLATTING BLASTING BOASTING COASTING COACTING COACHING POACHING POUCHING MOUCHING MOUTHING SOUTHING SOOTHING TOOTHING TROTHING TRITHING WRITHING WRATHING The shortest solution length: 33 Num of nodes checked: 384 Algorithm execution time: 0.0127192 seconds Do you want to search for another (Y/N)? </pre>
-----	-------------------------------------	---

3. A-star (A*)

Tabel 4.3.1 Uji Coba A*

No.	Masukan	Keluaran
1.	Start: HEAD Finish: TAIL	<pre> ----- Enter the start and finish word Start: HEAD Finish: TAIL Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: HEAD HEAL HEIL HAIL TAIL The shortest solution length: 5 Num of nodes checked: 76 Algorithm execution time: 0.0018247 seconds Do you want to search for another (Y/N)? </pre>

2.	Start: ATE Finish: BAT	<pre> ----- Enter the start and finish word Start: ATE Finish: BAT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: ATE ATT AIT BIT BAT The shortest solution length: 5 Num of nodes checked: 80 Algorithm execution time: 0.0013714 seconds </pre>
3.	Start: JELLY Finish: SUPER	<pre> ----- Enter the start and finish word Start: JELLY Finish: SUPER Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: JELLY JELLS TELLS TELES TULES TUBES TUBER SUBER SUPER The shortest solution length: 9 Num of nodes checked: 1751 Algorithm execution time: 0.0811939 seconds Do you want to search for another (Y/N)? </pre>
4.	Start: TALENT Finish: FALCON	<pre> ----- Enter the start and finish word Start: TALENT Finish: FALCON Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! No solution found. Num of nodes checked: 1 Algorithm execution time: 1.579E-4 seconds </pre>

5.	Start: FARMING Finish: FARMERS	<pre> ----- Enter the start and finish word Start: FARMING Finish: FARMERS Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: FARMING FARTING MARTING MARTINS MATTINS LATTINS LATTENS BATTENS BATTERS BARTERS DARTERS DARNERS WARNERS WARMERS FARMERS The shortest solution length: 15 Num of nodes checked: 2148 Algorithm execution time: 0.0768088 seconds </pre>
6.	Start: REACT Finish: FLASK	<pre> ----- Enter the start and finish word Start: REACT Finish: FLASK Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: REACT REACH ROACH COACH CLACH CLACK FLACK FLASK The shortest solution length: 8 Num of nodes checked: 106 Algorithm execution time: 0.0031731 seconds </pre>

7.	Start: FLASK Finish: REACT	<pre> ----- Enter the start and finish word Start: FLASK Finish: REACT Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: FLASK FLACK PLACK PLACE PEACE PEACH REACH REACT The shortest solution length: 8 Num of nodes checked: 259 Algorithm execution time: 0.0059803 seconds </pre>
8.	Start: APE Finish: MAN	<pre> ----- Enter the start and finish word Start: APE Finish: MAN Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: APE OPE OPT OAT MAT MAN The shortest solution length: 6 Num of nodes checked: 276 Algorithm execution time: 0.0039853 seconds </pre>

9.	Start: CHARGE Finish: COMEDO	<pre>Enter the start and finish word Start: CHARGE Finish: COMEDO Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 3 You have chosen A* algo! Solution found: CHARGE CHANGE CHANGS CHANTS CHINTS CHINES CHINED COINED COINER CONNER CONGER CONGES CONIES CONINS CONING COMING HOMING HOMINY HOMILY HOMELY COMELY COMEDY COMEDO The shortest solution length: 23 Num of nodes checked: 7747 Algorithm execution time: 0.3098787 seconds</pre>
----	---------------------------------	--

10.	Start: ATLASES Finish: CABARET	<pre> >> 3 You have chosen A* algo! Solution found: ATLASES ANLASES ANLACES UNLACES UNLACED UNLADED UNFADED UNFAKED UNCAKED UNCASED UNCASES UNEASES UREASES CREASES CRESSES CROSSES CROSSER CROSIER CROZIER CRAZIER BRAZIER BRAKIER BEAKIER PEAKIER PECKIER PICKIER DICKIER DICKIES HICKIES HACKIES HACKLES HECKLES DECKLES DECILES DEFILES DEFILED DEVILED DEVELED REVELED RAVELED RAVENED HAVENED HAVERED WAVERED WATERED CATERED CAPERED TAPERED TABERED TABORED TABORET TABARET CABARET The shortest solution length: 53 Num of nodes checked: 7947 Algorithm execution time: 0.2036996 seconds </pre>
-----	-----------------------------------	---

11.	Start: QUIRKING Finish: WRATHING	<pre> 773 You have chosen A* algo! Solution found: QUIRKING QUIRTING QUILTING QUILLING QUELLING DUELLING DWELLING SWELLING SPELLING SPALLING SPARLING SPARRING SCARRING SCARTING SCATTING SLATTING BLATTING BLASTING BOASTING COASTING COACTING COACHING COUCHING MOUCHING MOUTHING SOUTHING SOOTHING TOOTHING TROTHING TRITHING WRITHING WRATHING The shortest solution length: 32 Num of nodes checked: 873 Algorithm execution time: 0.0334711 seconds </pre>
-----	-------------------------------------	--

4. Invalid Input

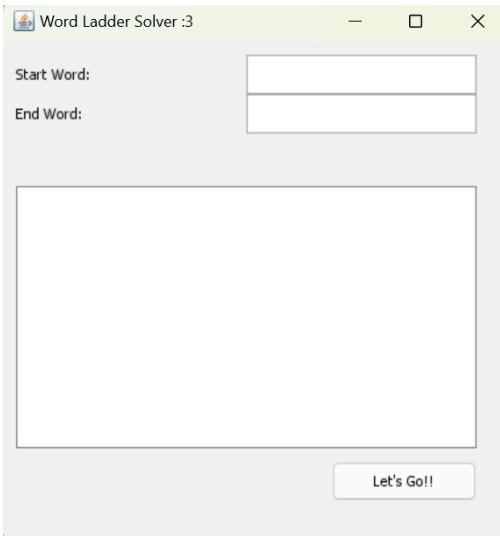
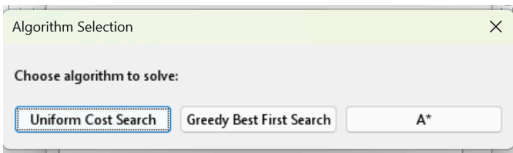
Tabel 4.4.1 Uji Coba Invalid Input

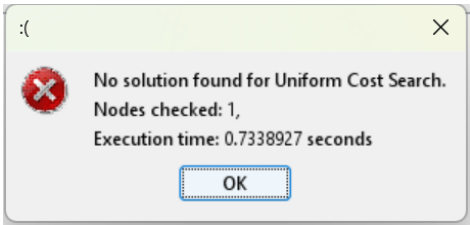
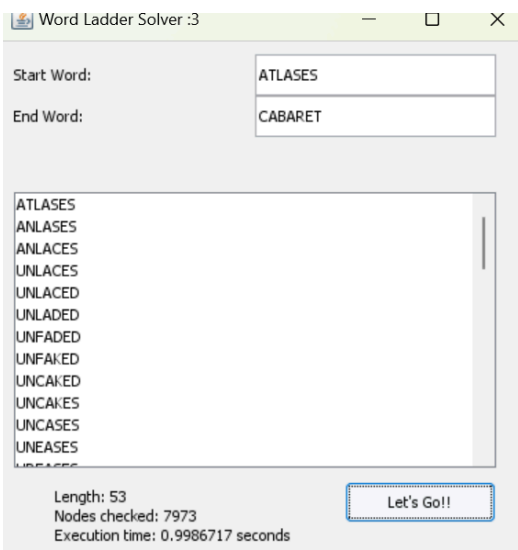
<pre> Do you want to search for another (Y/N)? >> 1 Wrong input Choose your algorithm to solve: 1. Uniform Cost Search 2. Greedy Best First Search 3. A* Enter `0` to quit. >> 4 Wrong input. Retrying... ----- Enter the start and finish word Start: STIMA Finish: COOL Both needs to be the same length. ----- </pre>	<p>Seluruh testing invalid input</p> <ul style="list-style-type: none"> • Opsi • Panjang berbeda • Input tidak ada di Dict
--	---

<pre> ----- Enter the start and finish word Start: TUBES Finish: STRES Finish word does not exist in the dictionary. ----- </pre>	
<pre> ----- Enter the start and finish word Start: AAAAAA Finish: STRESS Start word does not exist in the dictionary. ----- </pre>	
<pre> ----- Enter the start and finish word Start: KAPAN Finish: LIBUR Both words do not exist in the dictionary. ----- </pre>	

5. GUI

Tabel 4.5.1 Contoh Tampilan GUI

	Tampilan awal Word Ladder Solver
	Pop up box untuk memilih algoritma

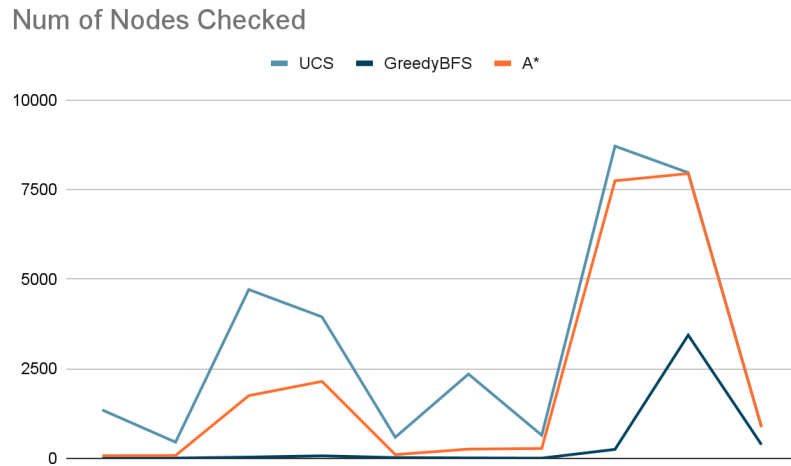
	Solusi tidak ditemukan
	Ditemukannya solusi dan detail akan tercetak di label bawah

E. Analisis

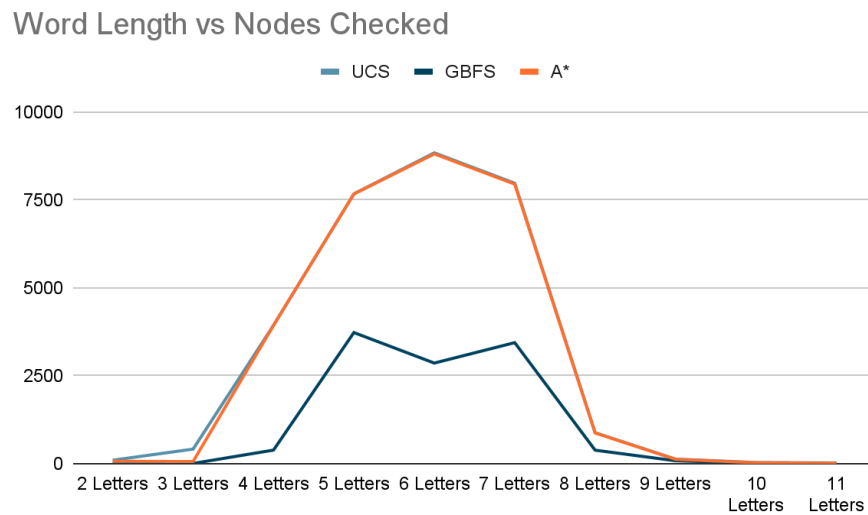
Jika diperhatikan lebih lagi, pada algoritma UCS, kita seolah-olah melakukannya dengan cara BFS. Secara teknis UCS memang mirip dengan BFS, menggunakan konsep queue simpul hidup dan melakukan dequeue setiap iterasinya hingga sebuah solusi ditemukan atau queue menjadi kosong. Pada persoalan ini kita menetapkan jarak antar node adalah 1, didapat dari jumlah step perubahan huruf yang dibutuhkan, yang dalam permainan ini hanya dibolehkan 1x. Maka dari itu, pada kasus ini secara teknis UCS akan mirip dengan BFS, cara pembangkitan node maupun jalur yang dihasilkan akan sama. Cost antar node menjadi tidak penting lagi, sebab urutan simpul hidup akan selalu sama, node baru akan selalu di-*generate* di belakang queue, sebab cost yang konsisten semakin naik jumlahnya. Selanjutnya algoritma akan dijalankan sama seperti BFS, node akan di-*dequeue* lalu akan ditumbuhkan node tetangganya dan menambahkan costnya dengan 1, serta setting parentnya dengan *currentnode*.

Tabel 5.1.1 Banyak nodes yang dicek

Masukan	UCS	Greedy BFS	A*
Start: HEAD Finish: TAIL	1351	6	76
Start: ATE Finish: BAT	454	9	80
Start: JELLY Finish: SUPER	4711	35	1751
Start: TALENT Finish: FALCON	1	1	1
Start: FARMING Finish: FARMERS	3948	72	2148
Start: REACT Finish: FLASK	589	22	106
Start: FLASK Finish: REACT	2353	13	259
Start: APE Finish: MAN	645	7	276
Start: CHARGE Finish: COMEDO	8713	251	7747
Start: ATLASES Finish: CABARET	7973	3441	7947
Start: QUIRKING Finish: WRATHING	881	384	873
Rata-rata	3161	424	2126



Gambar 5.1.1 Line Graph Banyak Nodes Dicek



Gambar 5.1.2 Line Graph Perbandingan Panjang Kata dan Jumlah Nodes
(Catatan: Test case diambil dari [DataGenetics Word Ladder Solver \(Longest Solutions\)](#))

Tabel 5.1.2 Waktu Eksekusi Algoritma (s)

Masukan	UCS	Greedy BFS	A*
Start: HEAD Finish: TAIL	0.0306589	0.0606741	0.0018247
Start: ATE Finish: BAT	0.0068532	7.698E-4	0.0013714
Start: JELLY Finish: SUPER	0.1276795	0.0051161	0.0811939

Start: TALENT Finish: FALCON	0.0010479	1.764E-4	1.579E-4
Start: FARMING Finish: FARMERS	0.121516	0.0051164	0.0768088
Start: REACT Finish: FLASK	0.0138845	8.814E-4	0.0031731
Start: FLASK Finish: REACT	0.0681262	4.113E-4	0.0059803
Start: APE Finish: MAN	0.0731367	2.087E-4	0.0039853
Start: CHARGE Finish: COMEDO	0.2518857	0.0109798	0.3098787
Start: ATLASES Finish: CABARET	0.2675815	0.1458964	0.2036996
Start: QUIRKING Finish: WRATHING	0.0385831	0.0127192	0.0334711
Rata-rata	0.09099574545	0.02208632727	0.06559498182

Berdasarkan tabel 5.1.1 dan gambar 5.1.1, kita mendapat informasi bahwa algoritma GBFS melakukan pengecekan nodes yang sedikit berbanding jauh dari kedua algoritma lainnya. Algoritma UCS dan A* melakukan pengecekan nodes dengan jumlah yang hampir-hampir sama, tetapi A* selalu melakukan pengecekan yang lebih sedikit daripada UCS. Hal ini terjadi akibat pengecekan GBFS yang ke dalam, tidak meluas seperti UCS. Algoritma GBFS adalah pencarian *informed search* yang berusaha meminimalkan nodes yang diekspansi. Seperti yang sudah disebutkan pada dasar teori algoritma, GBFS melakukan minimisasi dengan memprioritaskan node dengan cost heuristik terendah. Fungsi heuristik mencari nodes yang kemungkinan mendekatkan ke goal node, mengestimasi node yang membutuhkan nilai cost menuju goal yang paling sedikit sehingga goal node cepat ditemukan. Memprioritaskan node dengan *chances* menuju ke goal lebih tinggi mengakibatkan penelusuran menuju lokasi pencarian yang lebih menjanjikan.

Teori ini didukung dengan hasil algoritma UCS yang merupakan *uninformed search* yang mengekskan berdasarkan *path cost*, yaitu *cost* dari awal hingga node yang sekarang dicek. Nodes diekspan berdasarkan *cost* aktual dan tidak

mempertimbangkan kedekatannya ke goal node. Hal ini mengakibatkan pencarian dipastikan menjadi optimal, menemukan solusi dengan cost terkecil, yang dalam kasus ini adalah solusi dengan step minimal. Algoritma A* adalah bisa dibilang adalah kombinasi dari UCS dan GBFS. A* menggunakan perhitungan $f(n) = g(n) + h(n)$ dengan $g(n)$ adalah *actual cost* dan $h(n)$ sebagai *heuristic cost*. Hal ini menjadikan A* secara teoritis selalu memberikan jalur teroptimal sembari berusaha meminimalkan jumlah node yang diekspansi. Akan tetapi, perlu diingat juga bahwa A* akan menjadi optimal apabila heuristik yang digunakan *admissible*.

Analisis jumlah node ekspansi ini membuktikan bahwa pencarian dengan $g(n)$ menjamin keoptimalan solusi, sedangkan pencarian dengan $h(n)$ tidak menjamin keoptimalan solusi seperti yang terjadi pada GBFS. Berdasarkan percobaan yang kita lakukan, GBFS tidak selalu memberikan alur terpendek, menjadikannya tidak optimal. Walaupun begitu, GBFS dapat menemukan solusi dengan node ekspansi yang paling sedikit, diikuti dengan A* pada urutan kedua dan UCS pada urutan ketiga dalam hal jumlah node ekspansi minimum.

Jumlah node ekspansi ini juga berkaitan dengan perbandingan penggunaan memori dari setiap algoritma. Membangkitkan node artinya menyimpan informasi tambahan dan terus menerus berkembang seiring ekspansi berjalan. Hal ini menjadikan GBFS sebagai algoritma dengan penggunaan memori minimum untuk mencapai sebuah solusi. Berbanding lurus dengan jumlah node yang diekspansi, waktu algoritma GBFS jauh lebih cepat dari dua algoritma lainnya, sedangkan algoritma UCS dan A* memiliki selisih kecil dalam hal perbedaan kecepatan algoritma dengan algoritma A* menempati urutan kedua.

Gambar 5.1.2 merupakan hasil kumpulan data jumlah node yang diekspansi dalam suatu test case terpanjang pada panjang kata tertentu. Berdasarkan gambar tersebut jumlah node yang dicek mengalami kenaikan tertinggi pada kata dengan 5-7 huruf sebab terdapat banyak kata dalam jumlah itu, sehingga pengecekan UCS dan A* akan semakin meluas pada setiap pengecekan, menambah jumlah node yang dicek. Semakin panjang kata tersebut, semakin sedikit ekspansinya sehingga range pencarian semakin sempit.

Perlu ditambahkan juga, fungsi heuristik yang digunakan dalam permainan ini adalah konsep *hamming distance*. *Hamming distance* adalah jumlah huruf yang

berbeda antara kedua kata dalam slot huruf yang sama. Heuristik yang digunakan dalam permainan ini termasuk dalam kategori *admissible* sebab nilainya tidak pernah melebihi nilai aktual untuk mencapai goal. Hal ini disebabkan oleh perbedaan *cost* antar node adalah 1, menjadikan *cost* aktual terminimum adalah sebanyak huruf kata tersebut. Nilai maksimal dari heuristik ini pula juga sebanyak huruf dalam kata tersebut, menjadikan konsep *hamming distance* ini *admissible*. Dalam kasus ini, *hamming distance* akan selalu menjadi *admissible* selama kedua kata (awal dan akhir) memiliki panjang yang sama dan diterapkannya peraturan perubahan 1 huruf setiap langkahnya.

Berdasarkan seluruh analisis yang telah dilakukan, dapat kita simpulkan bahwa algoritma A* menjadi yang teroptimal dalam permainan ini. Hal tersebut disebabkan oleh penggunaan konsep perhitungan cost dengan mempertimbangkan nilai heuristik yang memperpendek jalur dan *cost* aktual yang memberikan jaminan optimal. Algoritma UCS juga memberikan solusi yang optimal, tetapi sedikit lebih lambat daripada A*. Algoritma GBFS mampu memberikan solusi dengan waktu yang sangat singkat, tetapi beberapa solusi mungkin tidak optimal karena terjebak dalam lokal optima.

F. Checklist

Tabel 7.1. Checklist

Poin	Ya	Tidak
Program berhasil dijalankan	v	
Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS	v	
Solusi yang diberikan pada algoritma UCS optimal	v	
Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search	v	
Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan	v	

algoritma A*		
Solusi yang diberikan pada algoritma A* optimal	v	
[Bonus]: Program memiliki tampilan GUI	v	

G. Links

1. Repository:

https://github.com/pandaandsushi/Tucil3_13522012

2. References:

[1] R. Munir, "Route-Planning-Bagian1," Institut Teknologi Bandung, 2024.

[Online]. Available: [Penentuan Rute \(Route/Path Planning\)](#)

[2] R. Munir, "Route-Planning-Bagian2," Institut Teknologi Bandung, 2024.

[Online]. Available: [Penentuan Rute \(Route/Path Planning\)](#)

[3] McLoone, J. (2012, January 11). *The Longest Word Ladder Puzzle Ever*.

[The Longest Word Ladder Puzzle Ever—Wolfram Blog](#)

[4] StackOverflow. (2012, April 14). "Need help creating a word ladder

between many one letter different pairs of words (java)". [Need help creating a](#)

[word ladder between many one letter different pairs of words \(java\) - Stack](#)

[Overflow](#)

[5] DataGenetics. [DataGenetics Word Ladder Solver \(Longest Solutions\)](#)