Laporan Tugas Kecil 3
IF2211 Strategi Algoritma
World Ladder Solver
Semester II Tahun 2023/2024

Disusun Oleh:

Muhammad Althariq Fairuz 13522027

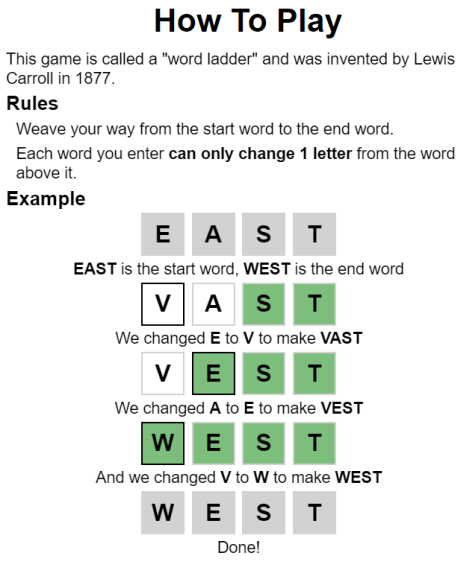
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

DAFTAR ISI

BAB 1

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.



Gambar 1.1 Ilustrasi dan Peraturan Permainan *Word Ladder*
(Sumber: https://wordwormdormdork.com/)

BAB 2

Landasan Teori

2.1 Algoritma Uniform-Cost Search

Algoritma Uniform-Cost Search adalah bentuk lanjutan dari algoritma BFS (Breadth-First Search) dengan menggunakan *priority queue* dengan node yang memiliki nilai g(n) (*cost*) paling kecil menjadi prioritas utama (f(n) = g(n)). Pada tugas kali ini, cost ditentukan berdasarkan kedalaman dari suatu node sehingga algoritma UCS yang diimplementasikan pada tugas kali ini mirip dengan BFS karena pada akhirnya semua node akan ditraversal secara per-level. Algoritma ini pasti menemukan solusi yang efektif. Namun, algoritma ini juga memakan waktu paling lama karena ia mengunjungi semua node pada tiap level.

2.2 Algoritma Greedy Best-First Search

Algoritma Greedy Best-First Search Adalah algoritma pencarian yang menggunakan fungsi heuristik untuk memandu pencariannya (f(n) = h(n)). Fungsi heuristik ini memberikan perkiraan tentang seberapa dekat suatu node dengan tujuan. GBFS bekerja dengan cara memilih node yang paling menjanjikan pada setiap langkahnya. Pada tugas kali ini, nilai h(n) (heuristik) dihitung berdasarkan seberapa banyak perbedaan huruf dari word ke-n dengan word tujuan (node tujuan). Node yang memiliki nilai heuristik terkecil akan menjadi prioritas dikunjungi. Kelemahan dari algoritma ini adalah solusi yang diperoleh belum tentu minimum global sehingga solusi yang dihasilkan belum tentu efektif seperti halnya Algoritma Greedy biasa.

2.3 Algoritma A*

Algoritma A* adalah algoritma pencarian yang merupakan gabungan dari UCS dan GBFS. A* menggunakan fungsi heuristik pada GBFS dan juga mempertimbangkan biaya jalur yang telah dilalui pada UCS. Oleh karena itu, A* dapat menemukan jalur terpendek dengan lebih efisien dan cepat dibandingkan UCS dan GBFS karena algoritma ini menggabungkan kelebihan dari dua algoritma sebelumnya. Seperti yang telah dijelaskan sebelumnya, cost dari tiap node ditentukan dari levelnya dan nilai heuristik dari tiap node diperoleh dari jumlah perbedaan antara node tersebut dengan node tujuan (f(n) = h(n) + g(n)). Secara teori, algoritma ini lebih sangkil dibandingkan algoritma UCS dan GBFS karena algoritma A* mempertimbangkan dua faktor penting, yaitu cost terendah serta word yang memiliki perbedaan jumlah huruf terkecil dengan word tujuan. Nilai heuristik yang digunakan admissible karena cost sebenarnya untuk mencapai ke word tujuan sama dengan jumlah perbedaan huruf yang ada pada suatu word/node ke-n dengan word tujuan.

BAB 3

Implementasi Program

3.1 Implementasi Algoritma Uniform-Cost Search

UCS.java

```java
package src;
import java.util.*;

public class UCS {
    // Uniform Cost Search algorithm is based on the cost (depth of the node) in this case
    public static List<String> findLadderUCS(String start, String end, Set<String> dictionary) {
        PriorityQueue<NodeUCS> queue = new PriorityQueue<>(Comparator.comparingInt(a -> a.getCost()));
        queue.add(new src.NodeUCS(start, 0, null));

        // Save the cost of each word
        Map<String, Integer> costs = new HashMap<>();
        costs.put(start, 0);
        int totalNodeVisited = 0;

        while (!queue.isEmpty()) {
            NodeUCS node = queue.poll();
            totalNodeVisited++;
            String word = node.getWord();

            // Check whether the word is the end word
            if (word.equals(end)) {
                List<String> path = new ArrayList<>();
                while (node != null) {
                    path.add(node.getWord());
                    node = node.getParent();
                }
                System.out.println("===== Total nodes visited: " + totalNodeVisited + " ======");
                Collections.reverse(path);
                return path;
            }

            // Generate all possible words from the current word
            for (int i = 0; i < word.length(); i++) {
                // Change one letter at a time
                char[] chars = word.toCharArray();
                for (char c = 'a'; c <= 'z'; c++) {
                    chars[i] = c;
                    String newWord = new String(chars);
                    if (dictionary.contains(newWord)) {
                        int newCost = costs.get(word) + 1;
                        // Check whether the new word is not in the costs map or the new cost is less than the previous cost
                        if (!costs.containsKey(newWord) || newCost < costs.get(newWord)) {
                            queue.add(new NodeUCS(newWord, newCost, node));
                            costs.put(newWord, newCost);
                        }
                    }
                }
            }
        }
        return null;
    }
}

```

Snipped

Gambar 3.1 Class UCS

1. **Class UCS**

   Class ini berisi algoritma UCS yang menggunakan Priority Queue of Node yang mengurutkan node berdasarkan costnya. Semakin kecil costnya maka ia semakin diprioritaskan, implementasinya mirip BFS karena cost disini ditinjau dari kedalaman suatu node.

3.2 Implementasi Algoritma Greedy Best-First Search

GBFS.java

```java
package src;

import java.util.*;

public class GBFS {

    public static List<String> findLadderGBFS(String start, String end, Set<String> dictionary) {
        // Greedy Best First Search algorithm is based on the heuristic value
        PriorityQueue<NodeGBFS> queue = new PriorityQueue<>(Comparator.comparingInt(a -> Heuristic.heuristic(a.getWord(), end)));
        queue.add(new src.NodeGBFS(start, null));

        Set<String> visited = new HashSet<>();

        int totalNodeVisited = 0;

        while (!queue.isEmpty()) {
            NodeGBFS node = queue.poll();
            totalNodeVisited++;
            String word = node.getWord();
            visited.add(word);

            // Check whether the word is the end word
            if (word.equals(end)) {
                List<String> path = new ArrayList<>();
                while (node != null) {
                    path.add(node.getWord());
                    node = node.getParent();
                }
                System.out.println("===== Total nodes visited: " + totalNodeVisited + " ======");
                Collections.reverse(path);
                return path;
            }

            // Generate all possible words from the current word
            for (int i = 0; i < word.length(); i++) {
                // Change one letter at a time
                char[] chars = word.toCharArray();
                for (char c = 'a'; c <= 'z'; c++) {
                    chars[i] = c;
                    String newWord = new String(chars);
                    // Check whether the new word is in the dictionary and not visited
                    if (dictionary.contains(newWord) && !visited.contains(newWord)) {
                        queue.add(new NodeGBFS(newWord, node));
                    }
                }
            }
        }
        return null;
    }
}
```

Snipped

Gambar 3.2 Class GBFS

1. **Class GBFS**

   Class ini berisi algoritma GBFS yang mengeksplor/mengunjungi node berdasarkan heuristiknya. Nilai heuristik terkecil akan dikunjungi/diambil terlebih dahulu, semua node dipastikan hanya dikunjungi sekali sehingga diperlukan Set untuk mencatat node yang telah dikunjungi.

3.3 Implementasi Algoritma A*

AStar.java

```java
package src;
import java.util.*;

public class AStar {

    // A* algorithm is based on the cost and its heuristic value (cost + heuristic)
    public static List<String> findLadderAStar(String start, String end, Set<String> dictionary) {
        PriorityQueue<NodeAStar> queue = new PriorityQueue<>(Comparator.comparingInt(a -> a.getCost() + a.getHeuristic()));
        queue.add(new src.NodeAStar(start, null, 0, Heuristic.heuristic(start, end)));

        // Save the cost and heuristic value of each word
        Map<String, Integer> costs = new HashMap<>();
        Set<String> visited = new HashSet<>();
        costs.put(start, 0);
        int totalNodeVisited = 0;

        while (!queue.isEmpty()) {
            NodeAStar node = queue.poll();
            totalNodeVisited++;
            String word = node.getWord();
            visited.add(word);

            // Check whether the word is the end word
            if (word.equals(end)) {
                List<String> path = new ArrayList<>();
                while (node != null) {
                    path.add(node.getWord());
                    node = node.getParent();
                }
                System.out.println("===== Total nodes visited: " + totalNodeVisited + " ======");
                Collections.reverse(path);
                return path;
            }

            // Generate all possible words from the current word
            for (int i = 0; i < word.length(); i++) {
                // Change one letter at a time
                char[] chars = word.toCharArray();
                for (char c = 'a'; c <= 'z'; c++) {
                    chars[i] = c;
                    String newWord = new String(chars);
                    // Check whether the new word is in the dictionary and not visited
                    if (dictionary.contains(newWord)  && !visited.contains(newWord)) {
                        int newCost = costs.get(word) + 1;
                        // Check whether the new word is not in the costs map or the new cost is less than the previous cost
                        if (!costs.containsKey(newWord) || newCost < costs.get(newWord)) {
                            queue.add(new NodeAStar(newWord,node, newCost, Heuristic.heuristic(newWord, end)));
                            costs.put(newWord, newCost);
                        }
                    }
                }
            }
        }
        return null;
    }
}
```

Snipped

Gambar 3.3 Class AStar

1. Class AStar

Class ini berisi algoritma A*yang menggunakan Priority Queue of Node yang mengurutkan node berdasarkan heuristiknya serta costnya. Semakin kecil jumlah cost dan heuristiknya maka ia semakin diprioritaskan, semua node dipastikan hanya dikunjungi sekali sehingga diperlukan Set untuk mencatat node yang telah dikunjungi dan solusi yang dihasilkan dijamin efektif.

3.4 Implementasi Heuristic

Heuristic.java

```java
package src;

public class Heuristic {
        // Heuristic function is based on the number of different characters between two words
        public static int heuristic(String start, String end){
            int diff = 0;
            for (int i = 0 ; i < start.length(); i++){
                if (start.charAt(i) != end.charAt(i)){
                    diff++;
                }
            }
            return diff;
        }
}
```

Snipped

Gambar 3.4 Class Heuristic

1. **Class Heuristic**

   Class ini berisi algoritma pencarian nilai heuristik, yaitu berdasarkan seberapa banyak perbedaan antara word ke-n dengan word tujuan.

3.5 Implementasi Node

Node.java

```java
package src;

public abstract class Node {

    protected String word;

    // Constructor
    public Node(String word) {
        this.word = word;
    }

    public abstract String getWord();
}

```

Snipped

Gambar 3.5 Class Node

NodeUCS.java

```java
package src;

public class NodeUCS extends Node {

    protected NodeUCS parent;
    protected Integer cost;
    // Constructor
    public NodeUCS(String word, Integer cost,  NodeUCS parent) {
        super(word);
        this.parent = parent;
        this.cost = cost;
    }

    public String getWord() {
        return word;
    }

    public NodeUCS getParent (){
        return parent;
    }

    public Integer getCost() {
        return cost;
    }

    public void setCost(Integer cost) {
        this.cost = cost;
    }
}

```

Snipped

Gambar 3.6 Class NodeUCS

NodeGBFS.java

```java
package src;

public class NodeGBFS extends Node{

    protected NodeGBFS parent;
    // Constructor
    public NodeGBFS(String word,  NodeGBFS parent) {
        super(word);
        this.parent = parent;
    }

    public String getWord() {
        return word;
    }

    public NodeGBFS getParent() {
        return parent;
    }

}

```

Snipped

Gambar 3.7 Class NodeGBFS

Gambar 3.8 Class NodeAStar

1. **Class Node serta Turunannya**

   Class ini berisi algoritma struktur data Node yang digunakan pada algoritma tertentu sesuai dengan namanya masing-masing dengan Node sebagai abstract base class (ABC) yang wajib diturunkan

3.6 Implementasi GBFS Tanpa Backtrack

NoBacktrackGBFS.java

```java
package src;

import java.util.*;

public class NoBacktrackGBFS {
        // Greedy Best First Search algorithm is based on the heuristic value
        public static List<String> findLadderNoBacktrackGBFS(String start, String end, Set<String> dictionary) {
        NodeGBFS node = new NodeGBFS(start, null);
        Set<String> visited = new HashSet<>();
        int totalNodeVisited = 0;

        while (true) {
            totalNodeVisited++;
            String word = node.getWord();
            visited.add(word);

            // Check whether the word is the end word
            if (word.equals(end)) {
                List<String> path = new ArrayList<>();
                while (node != null) {
                    path.add(node.getWord());
                    node = node.getParent();
                }
                System.out.println("===== Total nodes visited: " + totalNodeVisited + " ======");
                Collections.reverse(path);
                return path;
            }

            // Generate all possible words from the current word and select the one with the smallest heuristic value
            NodeGBFS nextNode = null;
            int minHeuristic = Integer.MAX_VALUE;
            for (int i = 0; i < word.length(); i++) {
                // Change one letter at a time
                char[] chars = word.toCharArray();
                for (char c = 'a'; c <= 'z'; c++) {
                    chars[i] = c;
                    String newWord = new String(chars);
                    // Check whether the new word is in the dictionary and not visited
                    if (dictionary.contains(newWord) && !visited.contains(newWord)) {
                        int heuristic = Heuristic.heuristic(newWord, end);
                        // Check whether the heuristic value is less than the previous heuristic value
                        if (heuristic < minHeuristic) {
                            minHeuristic = heuristic;
                            nextNode = new NodeGBFS(newWord, node);
                        }
                    }
                }
            }

            if (nextNode == null) {
                // No path found
                return null;
            }
            node = nextNode;
        }
    }
}
```

Snipped

1. Class NoBacktrackGBFS

Class ini berisi algoritma GBFS. Namun, tidak ada backtrack disini karena tidak ada priority queue, program akan mengambil nilai heuristik terkecil pada tiap level dan mengeksplor node/word tersebut. Jika solusi tidak ditemukan, program akan mengembalikan null.

BAB 4

Analisis dan Pengujian

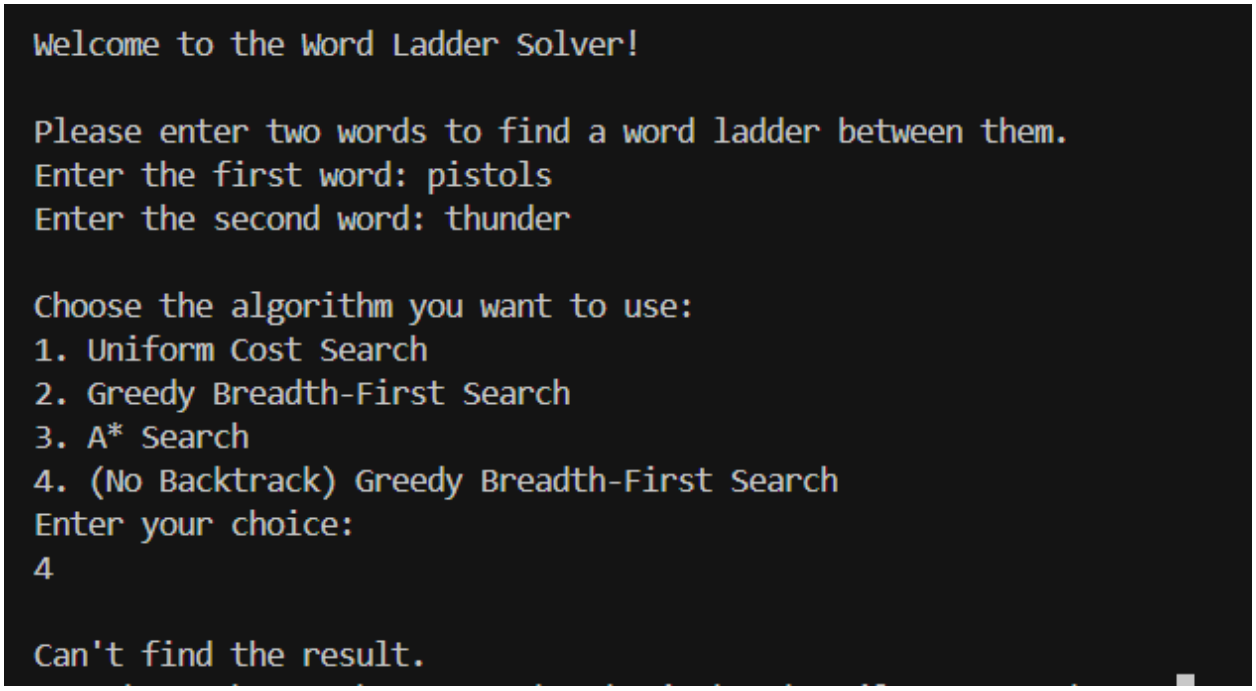
4.1 TC1

Start word: Pistols
End word: Thunder

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: pistols
Enter the second word: thunder

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 7660 ======
1. pistols
2. pistils
3. pastils
4. pasties
5. patties
6. putties
7. puttier
8. pettier
9. peatier
10. platier
11. platter
12. plutter
13. clutter
14. clunter
15. chunter
16. chunder
17. thunder
Steps taken: 16
Time taken: 263 ms
Total Memory (KB): 516096
Free Memory (KB): 443833
Used Memory (KB): 72262
================================================
```

```
Please enter two words to find a word ladder between them.
Enter the first word: pistols
Enter the second word: thunder

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 204 ======
1. pistols
2. pistils
3. pastils
4. pasties
5. pastier
6. tastier
7. tattier
8. tattler
9. rattler
10. rattier
11. battier
12. cattier
13. cartier
14. carrier
15. tarrier
16. terrier
17. tearier
18. learier
19. leavier
20. leafier
21. leadier
22. headier
23. beadier
24. beakier
25. brakier
26. brazier
27. brasier
28. brasher
29. brusher
30. blusher
31. bluster
32. blunter
33. blunder
34. bhunder
35. thunder
Steps taken: 34
Time taken: 33 ms
Total Memory (KB): 516096
Free Memory (KB): 448919
Used Memory (KB): 67176
================================================
PS C:\Users\Lenovo\Documents\ITB\Stima\tes\Tucil3_13522027\src>
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: 3
Enter the second word: e

The two words must have the same length and must be valid english words.
Enter the first word: pistols
Enter the second word: thunder

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 3180 ======
1. pistols
2. pistils
3. pastils
4. pasties
5. patties
6. potties
7. pottier
8. pettier
9. peatier
10. platier
11. platter
12. plutter
13. clutter
14. clunter
15. chunter
16. chunder
17. thunder
Steps taken: 16
Time taken: 191 ms
Total Memory (KB): 516096
Free Memory (KB): 450698
Used Memory (KB): 65397
================================================
```



4.2 TC 2

Start word: kill
End word: jail

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: kill
Enter the second word: jail

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 999 ======
1. kill
2. gill
3. gall
4. gail
5. jail
Steps taken: 4
Time taken: 56 ms
Total Memory (KB): 516096
Free Memory (KB): 445754
Used Memory (KB): 70341
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: kill
Enter the second word: jail

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 16 ======
1. kill
2. jill
3. jell
4. jeel
5. jerl
6. jarl
7. jail
Steps taken: 6
Time taken: 14 ms
Total Memory (KB): 516096
Free Memory (KB): 448593
Used Memory (KB): 67502
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: kill
Enter the second word: jail

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 24 ======
1. kill
2. bill
3. ball
4. bail
5. jail
Steps taken: 4
Time taken: 17 ms
Total Memory (KB): 516096
Free Memory (KB): 450413
Used Memory (KB): 65682
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: kill
Enter the second word: jail

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
4

===== Total nodes visited: 9 ======
1. kill
2. jill
3. jell
4. joll
5. joel
6. jeel
7. jerl
8. jarl
9. jail
Steps taken: 8
Time taken: 9 ms
Total Memory (KB): 516096
Free Memory (KB): 447295
Used Memory (KB): 68800
================================================
```

4.3 TC 3

Start word: red
End word: car

```
Please enter two words to find a word ladder between them.
Enter the first word: red
Enter the second word: car

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 551 ======
1. red
2. rad
3. cad
4. car
Steps taken: 3
Time taken: 26 ms
Total Memory (KB): 516096
Free Memory (KB): 447117
Used Memory (KB): 68978
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: red
Enter the second word: car

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 4 ======
1. red
2. rad
3. cad
4. car
Steps taken: 3
Time taken: 13 ms
Total Memory (KB): 516096
Free Memory (KB): 447154
Used Memory (KB): 68941
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: red
Enter the second word: car

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 4 ======
1. red
2. rad
3. cad
4. car
Steps taken: 3
Time taken: 15 ms
Total Memory (KB): 516096
Free Memory (KB): 451425
Used Memory (KB): 64670
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: red
Enter the second word: car

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
4

===== Total nodes visited: 4 ======
1. red
2. rad
3. cad
4. car
Steps taken: 3
Time taken: 9 ms
Total Memory (KB): 516096
Free Memory (KB): 447691
Used Memory (KB): 68404
================================================
```

4.4 TC 4

Start word: check
End word: black

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: check
Enter the second word: black

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 216 ======
1. check
2. cleck
3. clack
4. black
Steps taken: 3
Time taken: 25 ms
Total Memory (KB): 516096
Free Memory (KB): 448763
Used Memory (KB): 67332
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: check
Enter the second word: black

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 4 ======
1. check
2. cleck
3. bleck
4. black
Steps taken: 3
Time taken: 13 ms
Total Memory (KB): 516096
Free Memory (KB): 451055
Used Memory (KB): 65040
================================================
```

```
The two words must have the same length and must be valid english words.
Enter the first word: check
Enter the second word: black

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 6 ======
1. check
2. cleck
3. clack
4. black
Steps taken: 3
Time taken: 13 ms
Total Memory (KB): 516096
Free Memory (KB): 450340
Used Memory (KB): 65755
================================================
```

```
Please enter two words to find a word ladder between them.
Enter the first word: check
Enter the second word: black

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
4

===== Total nodes visited: 4 ======
1. check
2. cleck
3. bleck
4. black
Steps taken: 3
Time taken: 9 ms
Total Memory (KB): 516096
Free Memory (KB): 451360
Used Memory (KB): 64735
================================================
```

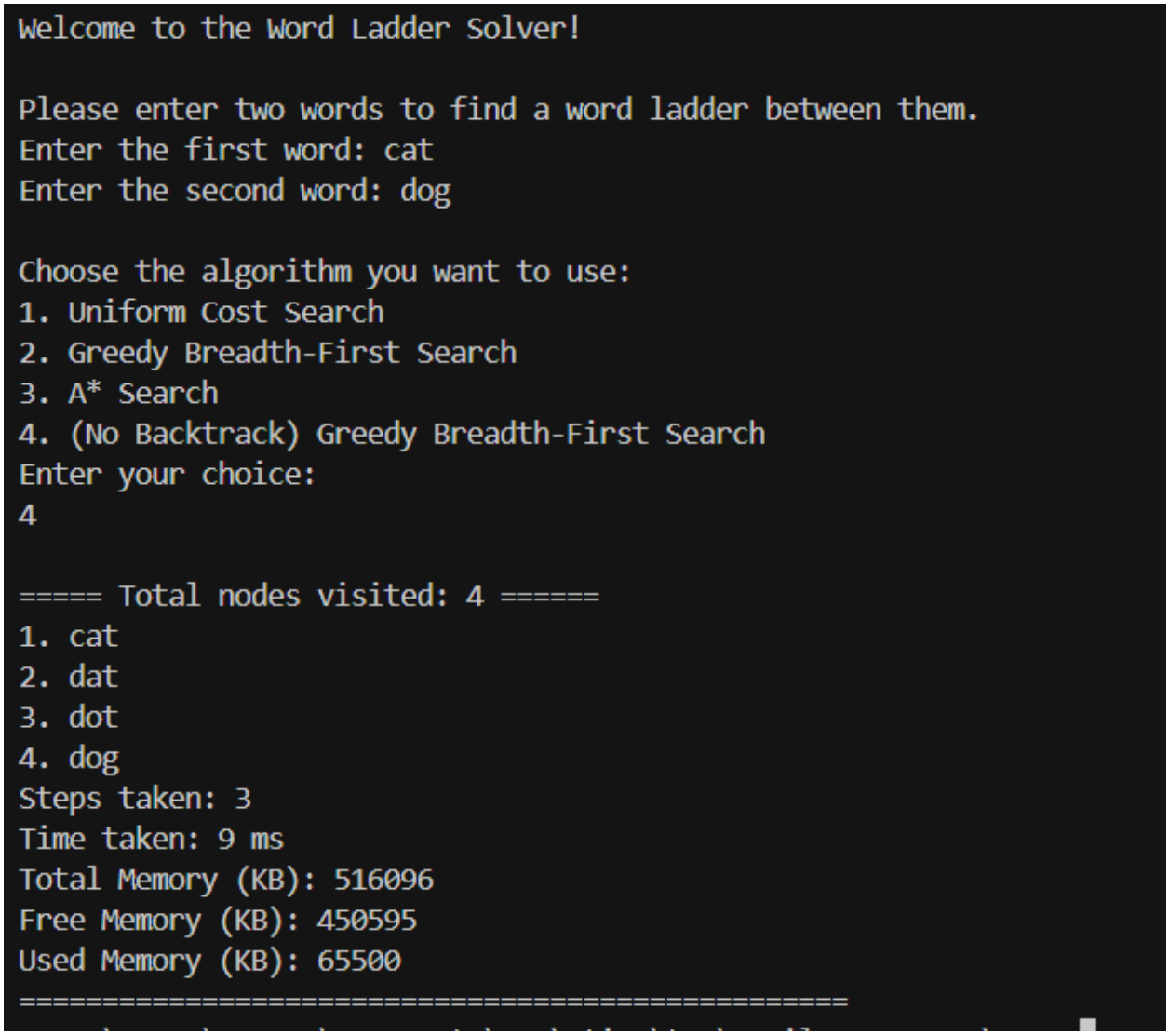
4.5 TC 5

Start word: cat
End word: dog

```
Please enter two words to find a word ladder between them.
Enter the first word: cat
Enter the second word: dog

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 1198 ======
1. cat
2. dat
3. dot
4. dog
Steps taken: 3
Time taken: 42 ms
Total Memory (KB): 516096
Free Memory (KB): 446155
Used Memory (KB): 69940
================================================
```

```
Please enter two words to find a word ladder between them.
Enter the first word: cat
Enter the second word: dog

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 4 ======
1. cat
2. dat
3. dot
4. dog
Steps taken: 3
Time taken: 12 ms
Total Memory (KB): 516096
Free Memory (KB): 451087
Used Memory (KB): 65008
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: cat
Enter the second word: dog

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 8 ======
1. cat
2. dat
3. dot
4. dog
Steps taken: 3
Time taken: 15 ms
Total Memory (KB): 516096
Free Memory (KB): 447959
Used Memory (KB): 68136
================================================
```



4.6 TC 6

Start word: king
End word: rock

```
The two words must have the same length and must be valid english words.
Enter the first word: king
Enter the second word: rock

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
1

===== Total nodes visited: 3083 ======
1. king
2. kink
3. kick
4. rick
5. rock
Steps taken: 4
Time taken: 94 ms
Total Memory (KB): 516096
Free Memory (KB): 470113
Used Memory (KB): 45982
================================================
```

```
The two words must have the same length and must be valid english words.
Enter the first word: king
Enter the second word: rock

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
2

===== Total nodes visited: 6 ======
1. king
2. ring
3. rink
4. rick
5. rock
Steps taken: 4
Time taken: 14 ms
Total Memory (KB): 516096
Free Memory (KB): 450171
Used Memory (KB): 65924
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: king
Enter the second word: rock

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
3

===== Total nodes visited: 9 ======
1. king
2. ring
3. rink
4. rick
5. rock
Steps taken: 4
Time taken: 14 ms
Total Memory (KB): 516096
Free Memory (KB): 473738
Used Memory (KB): 42357
================================================
```

```
Welcome to the Word Ladder Solver!

Please enter two words to find a word ladder between them.
Enter the first word: king
Enter the second word: rock

Choose the algorithm you want to use:
1. Uniform Cost Search
2. Greedy Breadth-First Search
3. A* Search
4. (No Backtrack) Greedy Breadth-First Search
Enter your choice:
4

===== Total nodes visited: 7 ======
1. king
2. ring
3. rong
4. rond
5. road
6. roak
7. rock
Steps taken: 6
Time taken: 11 ms
Total Memory (KB): 516096
Free Memory (KB): 450717
Used Memory (KB): 65378
================================================
```

Berdasarkan hasil pengujian sebelumnya, dapat disimpulkan bahwa algoritma UCS menghasilkan output yang dijamin efektif karena ia menelusuri seluruh word yang mungkin pada tiap levelnya (seperti BFS) sehingga waktu yang dibutuhkan juga lebih lama dan memori yang digunakan juga besar karena ia harus menyimpan informasi words yang diekspan di tiap levelnya pada priority queue selagi word tersebut ada dalam dictionary dan belum dikunjungi. Sedangkan algoritma GBFS lebih cepat dibandingkan dua algoritma lainnya karena ia memprioritaskan heuristik/jarak terdekat dari word ke-n dengan word tujuan. Namun, solusi yang dihasilkan belum tentu efektif/bisa saja bukan minimum global, memori yang digunakan juga lebih sedikit karena nodes yang dikunjungi cenderung lebih sedikit sehingga informasi yang disimpan tidak begitu banyak. Algoritma A* merupakan perpaduan dua algoritma sebelumnya sehingga kekurangan dari dua algoritma sebelumnya bisa tertutupi, seperti keefektifan UCS dalam mencari solusi serta kecepatan GBFS dalam mencari solusi sehingga solusi yang dihasilkan oleh Algoritma A* dijamin minimum global dan efisien karena ia memprioritaskan node dengan nilai f(n) terkecil terlebih dahulu dan pemakaian memori yang

relatif lebih kecil dibandingkan algoritma UCS dan sedikit lebih banyak dibandingkan algoritma GBFS karena node yang dikunjunginya berada diantara GBFS dan UCS.

Lampiran

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

Daftar Pustaka

Munir, Rinaldi. "Algoritma Route Planning - Bagian 1." Institut Teknologi Bandung, 2021

Route-Planning-Bagian1-2021.pdf (itb.ac.id)

Munir, Rinaldi. "Algoritma Route Planning- Bagian 2." Institut Teknologi Bandung, 2021

Route-Planning-Bagian2-2021.pdf (itb.ac.id)

Link Repository GitHub
AlthariqFairuz/Tucil3_13522027 (github.com)