

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma



Mohammad Nugraha Eka Prawira K1 13522001

1. Analisis dan Implementasi

1.1. Algoritma Uniform Cost Search (UCS)

UCS adalah algoritma pencarian graf yang mencari jalur dengan biaya terendah dari simpul awal ke simpul tujuan. Algoritma ini mengeksplorasi simpul yang memiliki biaya terendah terlebih dahulu.

Dalam kasus uniform cost search (UCS), berikut adalah penjelasan langkah-langkahnya:

1. Inisialisasi: Mulai dengan menempatkan simpul awal ke dalam antrian prioritas. Biaya dari simpul awal diatur menjadi 0.
2. Looping: Selama antrian prioritas tidak kosong, lakukan langkah-langkah berikut:
 - 2.1. Ambil simpul dengan biaya terendah dari antrian prioritas. Ini akan menjadi simpul saat ini.
 - 2.2. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, selesaikan pencarian.
 - 2.3. Jika tidak, periksa setiap simpul yang terhubung ke simpul saat ini.
 - 2.4. Untuk setiap simpul yang terhubung, periksa apakah biaya untuk mencapai simpul tersebut lebih kecil melalui simpul saat ini daripada biaya yang saat ini diketahui untuk mencapai simpul tersebut. Jika ya, perbarui biaya dan penunjuk jalur, lalu tambahkan simpul tersebut ke dalam antrian prioritas.
3. Penyelesaian: Ketika simpul tujuan ditemukan atau antrian prioritas kosong, pencarian selesai. Jalur dari simpul awal ke simpul tujuan bisa direkonstruksi dari penunjuk jalur yang telah disimpan pada setiap simpul. Jika simpul tidak ditemukan, maka program akan mengeluarkan bahwa jalur tidak ditemukan.
4. $f(n) = g(n)$, yaitu biaya jarak dari simpul start word ke simpul kata saat ini.

1.2. Algoritma Greedy Best First Search (GBFS)

GBFS adalah algoritma pencarian graf yang mencari jalur dengan biaya terendah dari simpul awal ke simpul tujuan. Algoritma ini hanya mempertimbangkan estimasi jarak langsung dari simpul saat ini ke simpul tujuan.

Dalam kasus Greedy Best First Search (GBFS), berikut adalah penjelasan langkah-langkahnya:

1. Inisialisasi: Mulai dengan menempatkan simpul awal ke dalam antrian prioritas. Heuristik digunakan untuk mengukur jarak heuristik dari simpul saat ini ke simpul tujuan. Heuristik yang digunakan adalah jumlah huruf yang berbeda dari kata di simpul awal atau simpul saat ini dengan kata tujuan.
2. Looping: Selama antrian prioritas tidak kosong, lakukan langkah-langkah berikut:

- 2.1. Ambil simpul dengan nilai heuristik terendah dari antrian prioritas. Ini akan menjadi simpul saat ini.
- 2.2. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, selesaikan pencarian.
- 2.3. Jika tidak, periksa setiap simpul yang terhubung ke simpul saat ini.
- 2.4. Untuk setiap simpul yang terhubung, tambahkan mereka ke dalam antrian prioritas.
3. Penyelesaian: Ketika simpul tujuan ditemukan atau antrian prioritas kosong, pencarian selesai. Jalur dari simpul awal ke simpul tujuan bisa direkonstruksi dari penunjuk jalur yang telah disimpan pada setiap simpul. Jika simpul tidak ditemukan, maka program akan mengeluarkan bahwa jalur tidak ditemukan.
4. $f(n) = h(n)$, yaitu biaya perkiraan dari simpul kata saat ini ke simpul kata tujuan.

1.3. Algoritma A* (A-Star)

A* adalah algoritma pencarian graf yang kombinasi dari UCS dan GBFS. Algoritma ini mencari jalur dengan biaya terendah dari simpul awal ke simpul tujuan, mempertimbangkan biaya aktual dari simpul awal ke simpul saat ini serta estimasi jarak langsung dari simpul saat ini ke simpul tujuan.

1. Inisialisasi: Mulai dengan menempatkan simpul awal ke dalam antrian prioritas. Biaya dari simpul awal diatur menjadi 0. Heuristik juga diterapkan pada simpul awal untuk memperkirakan biaya yang tersisa ke simpul tujuan.
2. Looping: Selama antrian prioritas tidak kosong, lakukan langkah-langkah berikut:
 - 2.1. Ambil simpul dengan biaya total terendah (biaya aktual ditambah estimasi biaya yang tersisa) dari antrian prioritas. Ini akan menjadi simpul saat ini.
 - 2.2. Periksa apakah simpul saat ini adalah simpul tujuan. Jika iya, selesaikan pencarian.
 - 2.3. Jika tidak, periksa setiap simpul yang terhubung ke simpul saat ini.
 - 2.4. Untuk setiap simpul yang terhubung, periksa apakah biaya aktual ditambah estimasi biaya yang tersisa lebih kecil dari biaya yang saat ini diketahui untuk mencapai simpul tersebut. Jika ya, perbarui biaya dan penunjuk jalur, lalu tambahkan simpul tersebut ke dalam antrian prioritas.
3. Penyelesaian: Ketika simpul tujuan ditemukan atau antrian prioritas kosong, pencarian selesai. Jalur dari simpul awal ke simpul tujuan bisa direkonstruksi dari penunjuk jalur yang telah disimpan pada setiap simpul. Jika simpul tidak ditemukan, maka program akan mengeluarkan bahwa jalur tidak ditemukan.
4. $f(n) = h(n) + g(n)$, yaitu biaya perkiraan dari simpul kata saat ini ke simpul kata tujuan ditambah biaya jarak dari simpul start word ke simpul kata saat ini.

1.4. Analisis

Heuristik yang digunakan pada algoritma A* admissible jika dan hanya jika estimasi biaya yang diberikan oleh heuristik tidak pernah melebihi biaya sebenarnya untuk mencapai simpul tujuan dari simpul saat ini. Dengan kata lain, heuristik yang admissible adalah yang optimis dan tidak pernah meremehkan biaya untuk mencapai tujuan.

Dalam konteks word ladder, kita dapat menggunakan heuristik yang admissible misalnya jumlah karakter yang berbeda antara kata saat ini dan kata tujuan. Heuristik ini admissible karena tidak pernah melebihi jumlah langkah yang sebenarnya diperlukan untuk mencapai kata tujuan dari kata saat ini.

Dengan memiliki heuristik yang admissible, algoritma A* akan menjamin pencarian jalur optimal, karena itu akan terus mencari jalur yang paling menjanjikan tergantung pada estimasi biaya dari heuristik.

Secara konseptual, UCS (Uniform Cost Search) dan BFS (Breadth-First Search) berbeda dalam cara mereka memprioritaskan simpul saat membangun jalur. UCS memprioritaskan simpul berdasarkan biaya aktual untuk mencapai mereka, sedangkan BFS memprioritaskan simpul berdasarkan kedalaman di mana mereka ditemukan.

Dalam kasus Word Ladder, UCS dan BFS akan mungkin menghasilkan urutan node yang sama. Hal ini karena UCS memberikan nilai cost yang cara kerjanya sama seperti menentukan kedalaman dalam algoritma BFS dan akan mengeksplorasi semua jalur dengan kedalaman yang sama sebelum melanjutkan ke jalur lainnya.

Dalam hal efisiensi, secara teoritis, algoritma A* diharapkan lebih efisien daripada UCS dalam menyelesaikan kasus Word Ladder. Hal ini disebabkan karena A* menggunakan heuristik untuk memandu pencarian, memungkinkannya untuk fokus pada jalur-jalur yang paling menjanjikan dan akhirnya menemukan solusi optimal dengan jumlah langkah yang lebih sedikit dibandingkan dengan UCS.

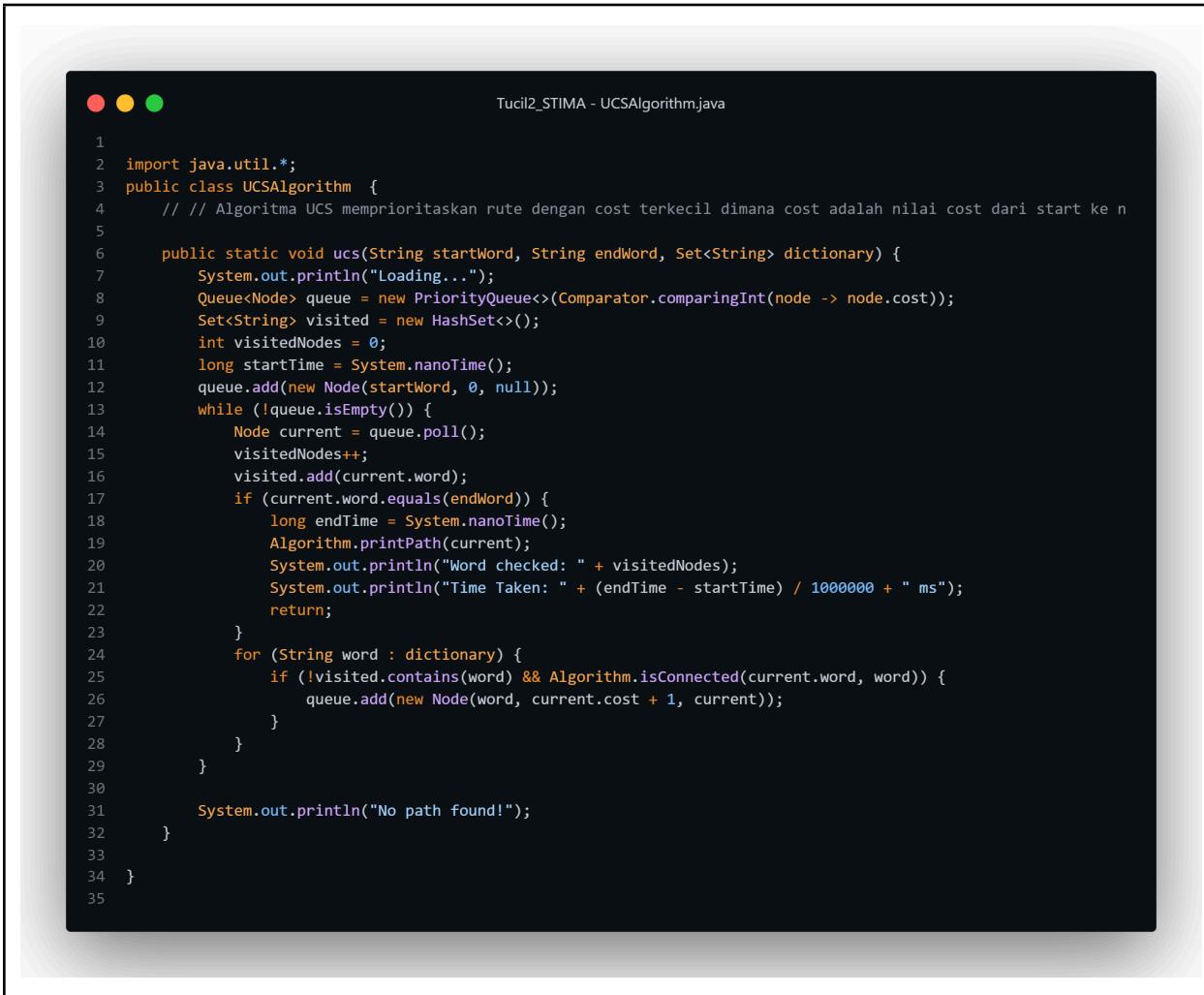
Sedangkan di lain hal, GBFS, secara teoritis tidak menjamin solusi optimal untuk persoalan Word Ladder. GBFS cenderung mengeksplorasi jalur-jalur yang terlihat paling dekat dengan tujuan berdasarkan heuristik yang digunakan, tanpa mempertimbangkan biaya total yang telah dikeluarkan untuk mencapai simpul saat ini. Oleh karena itu, GBFS dapat terjebak dalam jalur yang terlihat optimal tetapi akhirnya tidak optimal secara keseluruhan.

2. Source Code Program

2.1. UCSAlgorithm.java

Kelas ini mengimplementasikan algoritma Uniform Cost Search (UCS) untuk mencari rute terpendek antara dua kata dalam kamus. Metode yang dimiliki adalah:

ucs(String startWord, String endWord, Set<String> dictionary): Metode ini menerima kata awal (startWord), kata akhir (endWord), dan kamus kata-kata yang valid (dictionary). Algoritma UCS kemudian dijalankan untuk mencari rute terpendek dari startWord ke endWord. Metode ini juga mencetak hasil pencarian rute serta waktu eksekusi dan jumlah kata yang diperiksa.



The screenshot shows a terminal window titled "Tucil2_STIMA - UCSAlgorithm.java". The code is a Java program implementing the Uniform Cost Search (UCS) algorithm. It starts by importing java.util.* and defining a public class UCSAlgorithm. The main method, ucs, takes three parameters: startWord, endWord, and a set of strings representing a dictionary. It initializes a priority queue (PriorityQueue<Node>) to store nodes, a visited set (HashSet<String>) to keep track of visited words, and a variable visitedNodes to count the number of nodes visited. It also records the start time. A while loop continues until the queue is empty. Inside the loop, it removes the node with the lowest cost from the queue, increments the visitedNodes counter, and adds the current word to the visited set. If the current word equals the endWord, it prints the path, calculates and prints the execution time, and returns. Otherwise, it iterates through the dictionary. For each word in the dictionary, if it has not been visited and is connected to the current word, a new Node object is created with a cost of 1 more than the current word's cost, and it is added to the queue. If no path is found, it prints "No path found!".

```
1
2 import java.util.*;
3 public class UCSAlgorithm {
4     // // Algoritma UCS memprioritaskan rute dengan cost terkecil dimana cost adalah nilai cost dari start ke n
5
6     public static void ucs(String startWord, String endWord, Set<String> dictionary) {
7         System.out.println("Loading...");
8         Queue<Node> queue = new PriorityQueue<>(Comparator.comparingInt(node -> node.cost));
9         Set<String> visited = new HashSet<>();
10        int visitedNodes = 0;
11        long startTime = System.nanoTime();
12        queue.add(new Node(startWord, 0, null));
13        while (!queue.isEmpty()) {
14            Node current = queue.poll();
15            visitedNodes++;
16            visited.add(current.word);
17            if (current.word.equals(endWord)) {
18                long endTime = System.nanoTime();
19                Algorithm.printPath(current);
20                System.out.println("Word checked: " + visitedNodes);
21                System.out.println("Time Taken: " + (endTime - startTime) / 1000000 + " ms");
22                return;
23            }
24            for (String word : dictionary) {
25                if (!visited.contains(word) && Algorithm.isConnected(current.word, word)) {
26                    queue.add(new Node(word, current.cost + 1, current));
27                }
28            }
29        }
30        System.out.println("No path found!");
31    }
32 }
33
34 }
```

2.2. AstarAlgorithm.java

Kelas ini mengimplementasikan algoritma A* untuk mencari rute terpendek antara dua kata dalam kamus. Metode yang dimiliki adalah:

Astar(String startWord, String endWord, Set<String> dictionary): Metode ini menerima kata awal (startWord), kata akhir (endWord), dan kamus kata-kata yang valid (dictionary). Algoritma A* kemudian dijalankan untuk mencari rute terpendek dari startWord ke endWord. Metode ini juga mencetak hasil pencarian rute serta waktu eksekusi dan jumlah kata yang diperiksa.

```
● ○ ●
1 import java.util.Comparator;
2 import java.util.HashSet;
3 import java.util.PriorityQueue;
4 import java.util.Queue;
5 import java.util.Set;
6
7 public class AstarAlgorithm{
8     // Algoritma A star memprioritaskan rute dengan cost terkecil dimana cost adalah nilai cost dari n ke goal (jumlah huruf yang berbeda dari word n ke word goal) ditambah cost dari start ke n
9     public static void Astar(String startWord, String endWord, Set<String> dictionary){
10         System.out.println("Loading...");
11         Queue<Node> queue = new PriorityQueue<>(Comparator.comparingInt((Node node) -> node.cost));
12         Set<String> visited = new HashSet<>();
13         int visitedNodes = 0;
14         long startTime = System.nanoTime();
15         queue.add(new Node(startWord, Algorithm.GBFScost(endWord, startWord), null));
16         while (!queue.isEmpty()) {
17             Node current = queue.poll();
18             visitedNodes++;
19             visited.add(current.word);
20             if (current.word.equals(endWord)) {
21                 long endTime = System.nanoTime();
22                 Algorithm.printPath(current);
23                 System.out.println("Word checked: " + visitedNodes);
24                 System.out.println("Time taken: " + (endTime - startTime) / 1000000 + " ms");
25                 return;
26             }
27             for (String word : dictionary) {
28                 if (!visited.contains(word) && Algorithm.isConnected(current.word, word)) {
29                     int resultCost = Algorithm.GBFScost(endWord, word);
30                     queue.add(new Node(word, current.cost + 1 + resultCost, current));
31                 }
32             }
33         }
34         System.out.println("No path found!");
35     }
36 }
37
```

2.3. GBFS.java

Kelas ini mengimplementasikan algoritma Greedy Best First Search (GBFS) untuk mencari rute terpendek antara dua kata dalam kamus. Metode yang dimiliki adalah:

greedyBFS(String startWord, String endWord, Set<String> dictionary): Metode ini menerima kata awal (startWord), kata akhir (endWord), dan kamus kata-kata yang valid (dictionary). Algoritma GBFS kemudian dijalankan untuk mencari rute terpendek dari startWord ke endWord. Metode ini juga mencetak hasil pencarian rute serta waktu eksekusi dan jumlah kata yang diperiksa.

```
Tucil2_STIMA - GBFS.java

1 import java.util.Comparator;
2 import java.util.HashSet;
3 import java.util.PriorityQueue;
4 import java.util.Queue;
5 import java.util.Set;
6
7 public class GBFS {
8     // Algoritma greedy BFS memprioritaskan rute dengan cost terkecil dimana cost adalah nilai cost dari n ke goal (jumlah huruf yang berbeda dari word n ke word goal)
9     public static void greedyBFS(String startWord, String endWord, Set<String> dictionary){
10         System.out.println("Loading...");
11         Queue<Node> queue = new PriorityQueue<>(Comparator.comparingInt((Node node) -> node.cost));
12         Set<String> visited = new HashSet<>();
13         int visitedNodes = 0;
14         long startTime = System.nanoTime();
15         queue.add(new Node(startWord, Algorithm.GBFScost(endWord, startWord), null));
16         while (!queue.isEmpty()) {
17             Node current = queue.poll();
18             visitedNodes++;
19             visited.add(current.word);
20             if (current.word.equals(endWord)) {
21                 long endTime = System.nanoTime();
22                 Algorithm.printPath(current);
23                 System.out.println("Word checked: " + visitedNodes);
24                 System.out.println("Time taken: " + (endTime - startTime) / 1000000 + " ms");
25                 return;
26             }
27             for (String word : dictionary) {
28                 if (!visited.contains(word) && Algorithm.isConnected(current.word, word)) {
29                     int resultCost = Algorithm.GBFScost(endWord, word);
30                     queue.add(new Node(word, resultCost, current));
31                 }
32             }
33         }
34         System.out.println("No path found!");
35     }
36 }
37
```

2.4. Node.java

Node.java adalah kelas yang berisi struktur data node yang digunakan selama permainan. Kelas ini memiliki atribut *word* yaitu kata yang terisi dalam node tersebut, *cost* yaitu nilai dari node tersebut, dan *prev* yaitu penunjuk apa node sebelumnya yang akan digunakan dalam pembuatan path nantinya. Method yang berisi dari kelas ini hanyalah sebuah konstruktor.



Tucil2_STIMA - Node.java

```
1 public class Node {  
2     // Class untuk membuat node  
3     String word;  
4     int cost;  
5     Node prev;  
6  
7     Node(String word, int cost, Node prev) {  
8         this.word = word;  
9         this.cost = cost;  
10        this.prev = prev;  
11    }  
12}
```

2.5. Algorithm.java

Kelas ini berisi metode-metode yang digunakan secara umum dalam semua algoritma pencarian. Metode yang dimiliki adalah:

- isConnected(String word1, String word2): Memeriksa apakah dua kata memiliki hubungan yang memenuhi ketentuan permainan, yaitu memiliki panjang yang sama dan hanya memiliki satu perbedaan huruf.
- printPath(Node node): Mencetak rute dari simpul akhir kembali ke simpul awal.
- loadDictionary(String filename): Memuat kamus kata-kata valid dari sebuah file teks.
- readAlphabeticInput(Scanner scanner): Membaca input dari pengguna dan memastikan bahwa input hanya terdiri dari huruf saja.
- GBFScost(String target, String currentWord): Menghitung jumlah perbedaan huruf antara kata tujuan dan kata saat ini.

```
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.HashSet;
6 import java.util.List;
7 import java.util.Queue;
8 import java.util.Scanner;
9 import java.util.Set;
10
11 public class Algorithm{
12     // fungsi untuk mengecheck apakah dua kata memiliki hubungan sesuai dengan ketentuan game yaitu panjang katanya sama dan total perbedaan hurufnya hanya satu
13     public static boolean isConnected(String word1, String word2) {
14         int diffCount = 0;
15         if (word1.length() != word2.length())
16         {
17             return false;
18         }
19         for (int i = 0; i < word1.length(); i++) {
20             if (word1.charAt(i) != word2.charAt(i)) {
21                 diffCount++;
22                 if (diffCount > 1) return false;
23             }
24         }
25         return diffCount == 1;
26     }
27     // fungsi untuk mencetak hasil rute
28     public static void printPath(Node node) {
29         List<Node> path = new ArrayList<>();
30         while (node != null) {
31             path.add(node);
32             node = node.prev;
33         }
34
35         System.out.println("Path: ");
36         for (int i = path.size() - 1; i >= 0; i--) {
37             System.out.print(path.get(i).word);
38             if (i > 0) {
39                 System.out.print(" -> ");
40             }
41         }
42         System.out.println();
43     }
44     // fungsi untuk meload kamus
45     public static Set<String> loadDictionary(String filename) throws IOException {
46         Set<String> dictionary = new HashSet<>();
47         try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
48             String word;
49             while ((word = reader.readLine()) != null) {
50                 dictionary.add(word.toLowerCase());
51             }
52         }
53         return dictionary;
54     }
55     // fungsi untuk memvalidasi input hanya berisi huruf saja
56     public static String readAlphabeticInput(Scanner scanner) {
57         String input = scanner.nextLine().trim().toLowerCase();
58         while (!input.matches("[a-zA-Z]+")) {
59             System.out.println("Alphabet only!");
60             System.out.print("Reinput: ");
61             input = scanner.nextLine().trim().toLowerCase();
62         }
63         return input;
64     }
65     // fungsi untuk menghitung berapa banyak perbedaan huruf dari kata target dan kata yang sedang di check
66     public static int GBFScost(String target, String currentWord){
67         int diffCount = 0;
68         for (int i = 0; i < target.length(); i++) {
69             if (currentWord.charAt(i) != target.charAt(i)) {
70                 diffCount++;
71             }
72         }
73         return diffCount;
74     }
}
```

```
Tucll2_STIMA - Algorithm.java

1  public static void printIntro(){
2      System.out.println("WORD LADDER GAME");
3      System.out.println("-----");
4      System.out.println("-----");
5      System.out.println("-----");
6      System.out.println("-----");
7      System.out.println("-----");
8      System.out.println("-----");
9      System.out.println("-----");
10     System.out.println("-----");
11  }
12  public static void printInput(){
13      System.out.println();
14      System.out.println("INPUT WORD");
15      System.out.println("-----");
16      System.out.println("-----");
17      System.out.println("-----");
18      System.out.println("-----");
19      System.out.println("-----");
20      System.out.println("-----");
21      System.out.println("-----");
22  }
23  public static void printAlgo(String input){
24      System.out.println();
25      if (input.equals("gbfs") || input.equals("2")){
26          System.out.println("-----");
27          System.out.println("-----");
28          System.out.println("-----");
29          System.out.println("-----");
30          System.out.println("-----");
31          System.out.println("-----");
32          System.out.println("-----");
33          System.out.println("-----");
34          System.out.println("-----");
35          System.out.println("-----");
36      }
37      else if (input.equals("ucs") || input.equals("1")){
38          System.out.println("-----");
39          System.out.println("-----");
40          System.out.println("-----");
41          System.out.println("-----");
42          System.out.println("-----");
43          System.out.println("-----");
44          System.out.println("-----");
45          System.out.println("-----");
46          System.out.println("-----");
47      }
48      else{
49          System.out.println("-----");
50          System.out.println("-----");
51          System.out.println("-----");
52          System.out.println("-----");
53          System.out.println("-----");
54          System.out.println("-----");
55          System.out.println("-----");
56          System.out.println("-----");
57          System.out.println("-----");
58      }
59  }
60  }
61 }
62 }
63 }
64 }
```

2.6. Main.java

Main.java adalah inti program yang menghubungkan berbagai kelas lainnya untuk menjalankan gamenya.

```
TucI2_STIMA - Main.java

1 import java.io.IOException;
2 import java.util.Scanner;
3 import java.util.Set;
4 // javac Main.java UCSAlgorithm.java AstarAlgorithm.java GBFS.java Algorithm.java Node.java
5 // Java Main
6 public class Main{
7     static public void main(String[] args) {
8         Scanner scanner = new Scanner(System.in);
9         Algorithm.printIntro();
10        while (true){
11            try {
12                Set<String> dictionary = Algorithm.loadDictionary("words_alpha.txt");
13                System.out.println("Enter start word: ");
14                Algorithm.printInput();
15                System.out.print("Start word: ");
16                String startWord = Algorithm.readAlphabeticInput(scanner);
17
18                System.out.print("End word: ");
19                String endWord = Algorithm.readAlphabeticInput(scanner);
20
21                while (startWord.length() != endWord.length() || (!dictionary.contains(startWord) || !dictionary.contains(endWord)) ) {
22                    if (startWord.length() != endWord.length())
23                        System.out.println("The start and end word must in the same length!");
24                    else
25                        System.out.println("The word is not in the dictionary!");
26                    System.out.print("Start word: ");
27                    startWord = Algorithm.readAlphabeticInput(scanner);
28
29                    System.out.print("End word: ");
30                    endWord = Algorithm.readAlphabeticInput(scanner);
31
32                }
33
34                System.out.println("Choose Algo : ");
35                System.out.print("1. Uniform Cost Search (UCS). \n2. Greedy Best First Search (GBFS). \n3. A-Star (A*). \n");
36                System.out.print("Input: ");
37                String input = scanner.nextLine().trim().toLowerCase();
38                while (!input.equals("ucs") && !input.equals("gbfs") && !input.equals("a") && !input.equals("1") && !input.equals("2") && !input.equals("3")) {
39                    System.out.print("Wrong Input!");
40                    System.out.print("Choose Algo : (UCS,GBFS,A*) : ");
41                    input = scanner.nextLine().trim().toLowerCase();
42                }
43
44                System.out.println("Algorithm.printAlgo(input);
45                if (input.equals("ucs") || input.equals("1")){
46                    UCSAlgorithm_UCS(startWord, endWord, dictionary);
47                } else if (input.equals("gbfs") || input.equals("2")){
48                    GBFS.greedyBFS(startWord, endWord, dictionary);
49                } else {
50                    AstarAlgorithm_Astar(startWord, endWord, dictionary);
51                }
52
53                System.out.println("Do you wish to stop?(y/n) \n");
54                System.out.print("Input: ");
55                String command = scanner.nextLine().trim().toLowerCase();
56                if (command.equals("stop") || command.equals("yes") || command.equals("y")){
57                    break;
58                }
59            } catch (IOException e) {
60                System.err.println("Error loading dictionary: " + e.getMessage());
61            }
62        }
63    }
64
65}
66
67
68
69
70}
71}
72
73
```

3. Hasil Input dan Output

Terdapat 6 variasi input yang sama untuk masing masing algoritma. Variasi kata tersebut dipilih berdasarkan beberapa panjang huruf.Untuk kata 3 huruf terdapat variasi {lie,bed} dan {get,cog}, untuk 4 huruf terdapat {east,west} dan {good,luck}, untuk 5 huruf terdapat {frown,smile}, dan untuk 6 huruf terdapat {incite,willer}. Dimana maksud kurawal adalah {startWord,endWord}

3.1. UCS

WORD LADDER GAME

INPUT WORD

Start word: lie

End word: bed

Choose Algorithm :

- 1. Uniform Cost Search (UCS).
- 2. Greedy Best First Search (GBFS).
- 3. A-Star (A*).

Input: 1

UCS ALGORITHM

Loading...

Path:

lie -> lid -> bid -> bed

Word checked: 1013

Time Taken: 6691 ms

Do you wish to stop?(y/n)

Input: n

INPUT WORD

Start word: get

End word: cog

Choose Algorithm :

- 1. Uniform Cost Search (UCS).
- 2. Greedy Best First Search (GBFS).
- 3. A-Star (A*).

Input: 1

UCS ALGORITHM

Loading...

Path:

get -> got -> cot -> cog

Word checked: 3866

Time Taken: 10161 ms

INPUT WORD

Start word: east

End word: west

Choose Algorithm :

- 1. Uniform Cost Search (UCS).
- 2. Greedy Best First Search (GBFS).
- 3. A-Star (A*).

Input: 1

UCS ALGORITHM

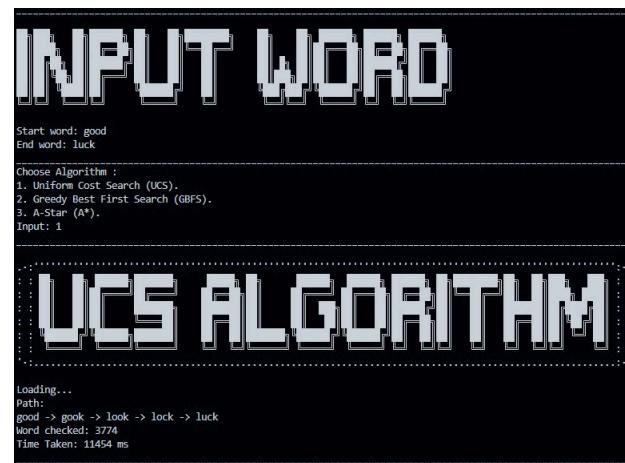
Loading...

Path:

east -> wast -> west

Word checked: 117

Time Taken: 289 ms



3.2. GBFS

INPUT WORD

Start word: lie
End word: bed

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: 2

GBFS ALGORITHM

Loading...
Path:
lie -> bee -> bed
Word checked: 4
Time taken: 28 ms

INPUT WORD

Start word: get
End word: cog

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: gbfs

GBFS ALGORITHM

Loading...
Path:
get -> got -> cot -> cog
Word checked: 4
Time taken: 41 ms

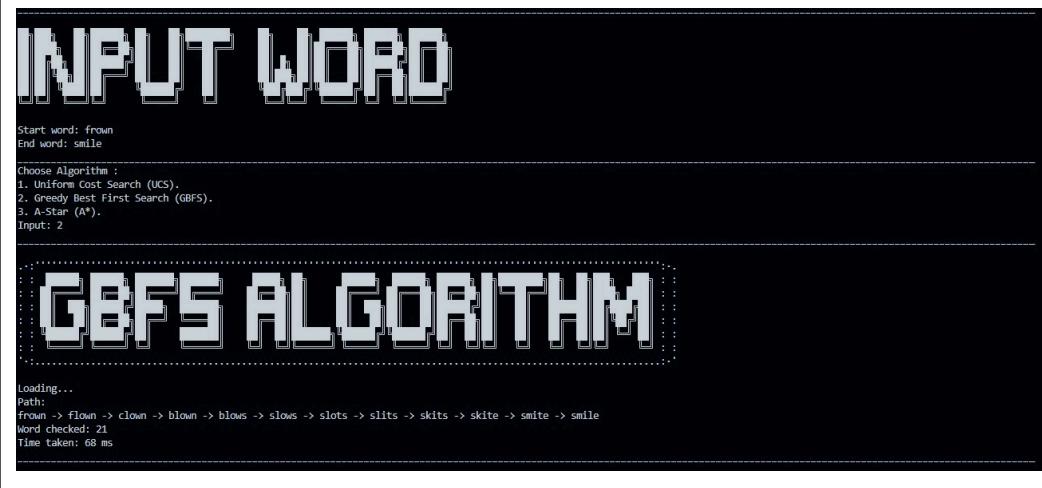
INPUT WORD

Start word: west
End word: g
The start and end word must in the same length!
Start word: east
End word: west

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: 2

GBFS ALGORITHM

Loading...
Path:
east -> wast -> west
Word checked: 3
Time taken: 6 ms



3.3. A-Star

INPUT WORD

Start word: lie
End word: bed

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: 3

A-STAR ALGORITHM

Loading...
Path:
lie -> lid -> led -> bed
Word checked: 24
Time taken: 117 ms

INPUT WORD

Start word: get
End word: cog

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: 3

A-STAR ALGORITHM

Loading...
Path:
get -> got -> cot -> cog
Word checked: 44
Time taken: 213 ms

INPUT WORD

Start word: east
End word: west

Choose Algorithm :
1. Uniform Cost Search (UCS).
2. Greedy Best First Search (GBFS).
3. A-Star (A*).
Input: 3

A-STAR ALGORITHM

Loading...
Path:
east -> west -> west
Word checked: 14
Time taken: 54 ms



4. Analisis dan Perbandingan

Dari segi optimalitas, UCS dikenal karena kemampuannya dalam menemukan solusi optimal dengan mengeksplorasi semua kemungkinan secara merata. Namun, pada kasus tertentu, UCS dapat menjadi lambat jika ruang pencarian sangat besar. Sementara itu, Greedy Best First Search cenderung membuat keputusan berdasarkan heuristik yang paling menjanjikan saat itu,

tanpa mempertimbangkan langkah-langkah di masa depan. Meskipun cepat, pendekatan ini tidak menjamin solusi optimal. A* menawarkan keseimbangan antara kedua pendekatan tersebut dengan menggabungkan biaya sejauh ini (g) dan estimasi biaya ke tujuan (h) untuk membuat keputusan. Ini membuatnya lebih cermat daripada Greedy Best First Search namun lebih efisien daripada UCS.

Dalam hal waktu eksekusi, UCS cenderung membutuhkan waktu yang lebih lama karena mengeksplorasi semua kemungkinan. Di sisi lain, Greedy Best First Search sering kali lebih cepat karena hanya fokus pada langkah terbaik saat ini. Namun, kecepatannya terkadang dikompromikan oleh kemungkinan menghasilkan solusi yang tidak optimal. A* menunjukkan kinerja yang lebih baik dalam hal waktu eksekusi dibandingkan UCS karena kecerdasan tambahan yang diberikan oleh estimasi biaya ke tujuan.

Dari segi memori yang dibutuhkan, UCS cenderung memerlukan memori yang besar karena perlu menyimpan semua node yang dieksplorasi. Greedy Best First Search dapat memerlukan memori yang lebih sedikit karena hanya menyimpan node yang paling menjanjikan, tetapi tidak menutup kemungkinan juga Greedy Best First Search menghasilkan jalur yang lebih panjang karena selalu memilih yang dikira adalah jalur terbaik dan ternyata jalur tersebut tidak optimal. A* juga memerlukan memori yang cukup besar karena perlu menyimpan informasi tentang semua node yang dieksplorasi serta estimasi biaya ke tujuan.

5. Referensi dan Lampiran

[Link github](#)

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

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