

Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

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



Disusun Oleh
Fabian Radenta Bangun 13522105

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Semester IV - 2024

A. Algoritma *Uniform Cost Search* (UCS) dalam Menemukan Solusi Permainan Word Ladder

Dalam melakukan pencarian dengan menggunakan metode UCS ada 2 elemen penting, yaitu simpul yang sedang ditinjau (*current node*) dan senarai simpul hidup. Algoritma UCS akan menempatkan *root node* sebagai *current node* di awal pencarian. Sementara senarai simpul hidup masih kosong. Kemudian dari *current node* akan dicari *child node*-nya dan dimasukkan ke dalam senarai simpul hidup. Urutan simpul di dalam senarai diurutkan berdasarkan *cost* paling kecil hingga paling besar. Kemudian pemeriksaan dilakukan lagi dengan nilai *current node* adalah *node* yang berada pada urutan paling depan di senarai simpul hidup. Langkah tersebut terus diulangi hingga *current node* sama dengan *goal node*.

Pada kasus permainan Word Ladder, dapat diimplementasikan algoritma UCS untuk menemukan solusi. *Current node* pada kasus ini adalah *word* yang sedang ditinjau. Senarai simpul hidup dapat diterapkan dengan menggunakan *Priority Queue* dengan mengkomparasikan *cost*-nya, yaitu jumlah huruf yang sudah diubah terhadap kata awal pada setiap langkah pencarian. Langkah penyelesaian pada program yang dibuat dapat diuraikan sebagai berikut :

1. Tempatkan kata awal pencarian sebagai *current node* yang berisi kata awal dan *cost* 0 (karena belum ada perubahan huruf).
2. Jika nilai kata pada *current node* sama dengan kata akhir yang dituju maka pencarian diakhiri dan solusi ditemukan.
3. Jika tidak, iterasi setiap karakter pada kata awal dengan abjad *a-z* dan periksa apakah kata yang dihasilkan valid. Misal : hit diperiksa setiap hi*, h*t, *it dengan * melambangkan iterasi abjad *a-z*.
4. Kata yang valid disimpan sebagai *child node* dari *current node* pada senarai simpul hidup dengan *cost* yang paling kecil berada didepan (pada implementasinya akan banyak simpul yang memiliki *cost* sama).
5. Ambil elemen *node* paling depan pada senarai simpul hidup dan jadikan ia sebagai *current node*. Lakukan pencarian ulang hingga solusi ditemukan atau elemen pada senarai simpul hidup habis dan solusi tidak ditemukan.

B. Algoritma *Greedy Best First Search* dalam Menemukan Solusi Permainan

Word Ladder

Pencarian solusi dengan metode *Greedy Best First Search* dilakukan dengan mengutamakan simpul yang jaraknya paling dekat dengan simpul tujuan (*goal node*) tanpa mempertimbangkan hal lain. Jarak dari *node n* menuju ke *goal node* dinyatakan dalam fungsi evaluasi $h(n)$. $h(n)$ menyatakan jarak garis lurus dari *node n* menuju ke *goal node*. Pencarian dimulai dari simpul akar (*root*) dinyatakan sebagai *current node*. Lalu *Root* akan dicari *child node* nya. Kemudian dari *child node* tersebut dicari *node* yang nilai fungsi $h(n)$ -nya paling kecil (jaraknya menuju *goal node* paling dekat). Kemudian dilakukan lagi pemeriksaan ulang dengan *node* tersebut dijadikan *current node*.

Pada kasus permainan Word Ladder dapat diimplementasikan algoritma *Greedy Best First Search* untuk menemukan solusi. *Current node* pada kasus ini adalah *word* yang sedang ditinjau dan fungsi $h(n)$ menyatakan jumlah perbedaan karakter pada kata yang sedang ditinjau terhadap kata tujuan. Pada program yang dibuat, penyelesaian masalah Word Ladder dengan algoritma *Greedy Best First Search* menggunakan pendekatan rekursif dengan langkah sebagai berikut :

1. Tempatkan kata awal pencarian sebagai *current node*.
2. Jika kata yang sedang ditinjau (nilai kata dari *current node*) sama dengan kata akhir yang dituju maka solusi ditemukan dan pencarian dihentikan.
3. Jika kata yang sedang ditinjau sudah pernah ditinjau sebelumnya maka pencarian gagal (pencarian menemukan *local optimum*). Solusi tidak ditemukan dan pencarian diakhiri.
4. Jika selain dua kondisi diatas maka pencarian diulangi dari *step 1* dengan *current node* adalah *child node* dari *current node* sebelumnya yang memiliki nilai $h(n)$ paling rendah.

C. Algoritma A* dalam Menemukan Solusi Permainan Word Ladder

Pencarian solusi dengan algoritma A* menggabungkan keunggulan *Greedy Best First Search* dengan UCS. Prinsip pencarinya kurang lebih sama dengan prinsip UCS. Namun, perbedaannya adalah nilai yang dibawa setiap *node* pada UCS untuk diurutkan pada senarai simpul hidup hanya mempertimbangkan *cost* yang telah dilewati hingga bisa sampai di *current node*. Sementara pada A* juga dipertimbangkan jarak dari *node n* menuju ke *goal node*. Bobot yang dibawa setiap *node* dalam senarai hidup pada algoritma A* adalah hasil penjumlahan dari $f(n)$, yaitu *cost* dari *root* menuju ke *current node n* dan $h(n)$, yaitu jarak dari *current node n*

menuju *goal node*. Sehingga bobot setiap node pada senarai hidup dapat dinyatakan dengan $\text{bobot} = f(n) + h(n)$. Langkah penyelesaian pada program yang dibuat dapat diuraikan sebagai berikut :

1. Tempatkan kata awal pencarian sebagai *current node* yang berisi kata awal dan *cost* 0 (karena belum ada perubahan huruf).
2. Jika nilai kata pada *current node* sama dengan kata akhir yang dituju maka pencarian diakhiri dan solusi ditemukan.
3. Jika tidak, iterasi setiap karakter pada kata awal dengan abjad *a-z* dan periksa apakah kata yang dihasilkan valid. Misal : hit diperiksa setiap hi^* , $h*t$, $*it$ dengan * melambangkan iterasi abjad *a-z*.
4. Kata yang valid disimpan sebagai *child node* dari *current node* pada senarai simpul hidup dengan *cost* yang paling kecil berada didepan (pada implementasinya akan banyak simpul yang memiliki *cost* sama).
5. Ambil elemen *node* paling depan pada senarai simpul hidup dan jadikan ia sebagai *current node*. Lakukan pencarian ulang hingga solusi ditemukan atau elemen pada senarai simpul hidup habis dan solusi tidak ditemukan.

D. **Source Code Program**

1. File Algoritma UCS (UCS.java)

Pada file ini dilakukan pencarian solusi word ladder dengan metode UCS.

```

1 import java.util.ArrayList;
2 import java.util.HashSet;
3 import java.util.PriorityQueue;
4
5 public class UCS {
6     public static void findWordLadderSolution(String startWord, String endWord, HashSet<String> dictionary) {
7         PriorityQueue<Node> pq = new PriorityQueue<>(new UCSNodeComparator());
8         HashSet<String> visitedNode = new HashSet<>();
9
10        Node startNode = new Node(startWord, new ArrayList<>());
11        startNode.currentNodePath.add(startWord);
12        pq.offer(startNode);
13        boolean found = false;
14
15        while (!pq.isEmpty()) {
16            Node current = pq.poll();
17
18            if (current.currentNodeWord.equals(endWord)) {
19                System.out.println("\nPath ditemukan : " + current.currentNodePath);
20                System.out.println("Jumlah node dikunjungi : " + visitedNode.size());
21                found = true;
22                break;
23            }
24
25            visitedNode.add(current.currentNodeWord);
26
27            for (String neighbor : getNeighbors(current.currentNodeWord, dictionary)) {
28                if (!visitedNode.contains(neighbor)) {
29                    Node neighborNode = new Node(neighbor, new ArrayList<>(current.currentNodePath));
30                    neighborNode.currentNodePath.add(neighbor);
31                    pq.offer(neighborNode);
32                }
33            }
34        }
35
36        if (!found) {
37            System.out.println("\nPencarian tidak ditemukan.\n");
38        }
39    }
40
41
42    private static ArrayList<String> getNeighbors(String word, HashSet<String> dictionary) {
43        ArrayList<String> neighbors = new ArrayList<>();
44        for (int i = 0; i < word.length(); i++) {
45            for (char c = 'a'; c <= 'z'; c++) {
46                StringBuilder neighborBuilder = new StringBuilder(word);
47                neighborBuilder.setCharAt(i, c);
48                String neighbor = neighborBuilder.toString();
49                if (!neighbor.equals(word) && dictionary.contains(neighbor)) {
50                    neighbors.add(neighbor);
51                }
52            }
53        }
54        return neighbors;
55    }
56}

```

2. File Algoritma *Greedy Best First Search* (GreedyBFS.java)

Pada file ini dilakukan pencarian solusi word ladder dengan metode Greedy Best First Search.

```

1 import java.util.*;
2
3 public class GreedyBFS {
4     private static ArrayList<String> visitedNode = new ArrayList<>();
5
6     private static int heuristic(String word, String endWord){
7         int res = 0;
8         for (int i=0; i<word.length(); i++){
9             if (word.charAt(i) != endWord.charAt(i)){
10                 res++;
11             }
12         }
13         return res;
14     }
15
16     private static String getMinimumDistance(String currentWord, String endWord, HashSet<String> dictionary) {
17         ArrayList<String> neighbors = new ArrayList<>();
18         int currentMin = 999;
19         String res = currentWord;
20         for (int i = 0; i < currentWord.length(); i++) {
21             for (char c = 'a'; c <= 'z'; c++) {
22                 StringBuilder neighborBuilder = new StringBuilder(currentWord);
23                 neighborBuilder.setCharAt(i, c);
24                 String neighbor = neighborBuilder.toString();
25                 if (!neighbor.equals(currentWord) && dictionary.contains(neighbor)) {
26                     neighbors.add(neighbor);
27                 }
28             }
29
30             for (String w : neighbors){
31                 if (heuristic(w, endWord) < currentMin){
32                     currentMin = heuristic(w, endWord);
33                     res = w;
34                 }
35             }
36         }
37         return res;
38     }
39
40     public static void findWordLadderSolution(String currentWord, String endWord, HashSet<String> dictionary){
41         if (currentWord.equals(endWord)){
42             visitedNode.add(currentWord);
43             System.out.println("\nPath ditemukan : " + visitedNode);
44             System.out.println("Jumlah node dikunjungi : " + visitedNode.size());
45             return;
46         }
47         else if (visitedNode.contains(currentWord)) {
48             System.out.println("\nHasil pencarian tidak ditemukan\n");
49             return;
50         }
51         else {
52             visitedNode.add(currentWord);
53             findWordLadderSolution(getMinimumDistance(currentWord, endWord, dictionary), endWord, dictionary);
54         }
55     }
56 }

```

3. File Algoritma A* (AStar.java)

Pada file ini dilakukan pencarian solusi word ladder dengan metode A*.

```

1 import java.util.ArrayList;
2 import java.util.HashSet;
3 import java.util.PriorityQueue;
4
5 public class AStar {
6     public static void findWordLadderSolution(String startWord, String endWord, HashSet<String> dictionary){
7         PriorityQueue<Node> pq = new PriorityQueue<>(new AStarNodeComparator(endWord));
8         HashSet<String> visitedNode = new HashSet<>();
9
10        Node startNode = new Node(startWord, new ArrayList<>());
11        startNode.currentNodePath.add(startWord);
12        pq.offer(startNode);
13        boolean found = false;
14
15        while (!pq.isEmpty()) {
16            Node current = pq.poll();
17
18            if (current.currentNodeWord.equals(endWord)) {
19                System.out.println("\nPath ditemukan : " + current.currentNodePath);
20                System.out.println("Jumlah node dikunjungi : " + visitedNode.size());
21                return;
22            }
23
24            visitedNode.add(current.currentNodeWord);
25
26            for (String neighbor : getNeighbors(current.currentNodeWord, dictionary)) {
27                if (!visitedNode.contains(neighbor)) {
28                    Node neighborNode = new Node(neighbor, new ArrayList<>(current.currentNodePath));
29                    neighborNode.currentNodePath.add(neighbor);
30                    pq.offer(neighborNode);
31                }
32            }
33        }
34        if (!found){
35            System.out.println("\nPencarian tidak ditemukan.\n");
36        }
37    }
38
39    private static ArrayList<String> getNeighbors(String word, HashSet<String> dictionary) {
40        ArrayList<String> neighbors = new ArrayList<>();
41        for (int i = 0; i < word.length(); i++) {
42            for (char c = 'a'; c <= 'z'; c++) {
43                StringBuilder neighborBuilder = new StringBuilder(word);
44                neighborBuilder.setCharAt(i, c);
45                String neighbor = neighborBuilder.toString();
46                if (!neighbor.equals(word) && dictionary.contains(neighbor)) {
47                    neighbors.add(neighbor);
48                }
49            }
50        }
51        return neighbors;
52    }
53 }

```

4. File Kelas Node (Node.java)

Pada file ini dideklarasikan kelas Node yang memiliki atribut currentNodeWord bertipe *string* untuk menyimpan nilai kata dalam simpul dan atribut currentNodePath yang bertipe *arraylist of string* untuk menyimpan path dari simpul.

```

1 import java.util.ArrayList;
2
3 public class Node {
4     public String currentNodeWord;
5     public ArrayList<String> currentNodePath;
6
7     public Node(String currentNodeWord, ArrayList<String> currentNodePath){
8         this.currentNodePath = currentNodePath;
9         this.currentNodeWord = currentNodeWord;
10    }
11 }
12 }
```

5. File UCS Comparator (UCSNodeComparator.java)

File ini ditulis bertujuan agar *priority queue* yang digunakan dalam algoritma UCS dapat mengurutkan simpul berdasarkan *cost* yang paling rendah.

```

1 import java.util.*;
2
3 public class UCSNodeComparator implements Comparator<Node>{
4
5     @Override
6     public int compare(Node n1, Node n2) {
7         if (n1.currentNodePath.size() > n2.currentNodePath.size()){
8             return 1;
9         } else if (n1.currentNodePath.size() < n2.currentNodePath.size()){
10            return -1;
11        } else {
12            return 0;
13        }
14    }
15 }
```

6. File A* Comparator (AStarNodeComparator.java)

File ini ditulis bertujuan agar *priority queue* yang digunakan dalam algoritma A* dapat mengurutkan simpul berdasarkan (*cost + heuristic*) yang paling rendah.

```

1 import java.util.*;
2
3 public class AStarNodeComparator implements Comparator<Node>{
4
5     private String endWord;
6
7     public AStarNodeComparator(String endWord){
8         this.endWord = endWord;
9     }
10
11     public int getHeuristic(String word){
12         int res = 0;
13         for (int i=0; i<word.length(); i++){
14             if (word.charAt(i) != endWord.charAt(i)){
15                 res++;
16             }
17         }
18         return res;
19     }
20     @Override
21     public int compare(Node n1, Node n2) int AStarNodeComparator.getHeuristic(String word)
22     if ((n1.currentNodePath.size() + getHeuristic(n1.currentNodeWord)) > (n2.currentNodePath.size() + getHeuristic(n2.currentNodeWord))){
23         return 1;
24     } else if (n1.currentNodePath.size() < n2.currentNodePath.size()){
25         return -1;
26     } else {
27         return 0;
28     }
29 }
30 }
```

7. File Program Utama (Main.java)

```
1 import java.util.*;
2
3 public class Main {
4     private static Integer inputMethod;
5     private static String startWord;
6     private static String endWord;
7     private static HashSet<String> wordArray;
8
9     Run | Debug
10    public static void main(String[] args) {
11
12        // 1. DEKLARASI AWAL
13        Scanner scanner = new Scanner(System.in);
14        Utils.hii();
15
16        // 2. INPUT START WORD DAN END WORD
17        while (true){
18            System.out.print("Masukkan start word : ");
19            startWord = scanner.nextLine();
20            startWord = startWord.toLowerCase();
21            System.out.print("Masukkan end word : ");
22            endWord = scanner.nextLine();
23            endWord = endWord.toLowerCase();
24            wordArray = Utils.makeListOfWords(startWord);
25
26            if (startWord.length()==endWord.length() && wordArray.contains(startWord) && wordArray.contains(endWord)){
27                break;
28            }
29
30            // output error input
31            if (wordArray.contains(startWord)){
32                System.out.println("\nTolong masukkan start word dengan benar.\n");
33            } else if (wordArray.contains(endWord)){
34                System.out.println("\nTolong masukkan end word dengan benar.\n");
35            } else {
36                System.out.println("\nPanjang start word dengan end word berbeda.\n");
37            }
38
39        // 3. INPUT METODE
40        Utils.showMethodMenu();
41        while (true){
42            System.out.print("Masukkan pilihan metode : ");
43            if (scanner.hasNextInt()){
44                inputMethod = scanner.nextInt();
45                if (inputMethod>=1 && inputMethod<=3){
46                    break;
47                } else {
48                    System.out.println("\nTolong masukkan inputMethod dengan benar.\n");
49                    scanner.nextLine();
50                }
51            } else {
52                System.out.println("\nTolong masukkan inputMethod dengan benar.\n");
53                scanner.nextLine();
54            }
55        }
56
57        // 4. PANGGIL ALGORITMA
58        long startTime = System.currentTimeMillis();
59
60        if (inputMethod==1){
61            UCS.findWordLadderSolution(startWord, endWord, wordArray);
62        } else if (inputMethod==2){
63            GreedyBFS.findWordLadderSolution(startWord, endWord, wordArray);
64        } else {
65            AStar.findWordLadderSolution(startWord, endWord, wordArray);
66        }
67
68        long endTime = System.currentTimeMillis();
69
70        // 5. OUTPUT HASIL DAN END
71        scanner.close();
72        System.out.println("Waktu eksekusi : " + (endTime-startTime) + "ms\n");
73        Utils.thanks();
74    }
75 }
```

8. File Dictionary (words.txt)

File ini berisi seluruh kata bahasa inggris yang dianggap valid (menggunakan *dictionary* yang direkomendasikan asisten).

```
1 aa
2 aah
3 aahed
4 aahing
5 aahs
6 aal
7 aalii
8 aaliis
9 aals
10 aardvark
11 aardwolf
12 aargh
13 aarrgh
14 aarrghh
15 aas
16 aasvogel
17 ab
18 aba
19 abaca
20 abacas
21 abaci
22 aback
23 abacus
24 abacuses
25 abafit
26 abaka
27 abakas
28 abalone
29 abalones
30 abamp
31 abampere
32 abamps
33 abandon
34 abandons
35 abapical
36 abas
37 abase
38 abased
39 abasedly
40 abaser
41 abasers
42 abases
43 abash
44 abashed
45 abashes
46 abashing
47 abasia
48 abasias
49 abasing
50 abatable
51 abate
52 abated
53 abater
54 abaters
55 abates
56 abating
57 abatis
58 abatizes
59 abator
60 abators
61 abattis
62 abattoir
63 abaxial
64 abaxile
65 abba
66 abbaclies
67 abbacy
68 abbas
69 abbatial
70 abbe
71 abbes
72 abbess
73 abbesses
74 abbey
75 abbey
76 abbot
77 abbocy
78 abbots
79 abdicate
80 abdomen
81 abdomens
82 abdomina
```

E. Program Testing

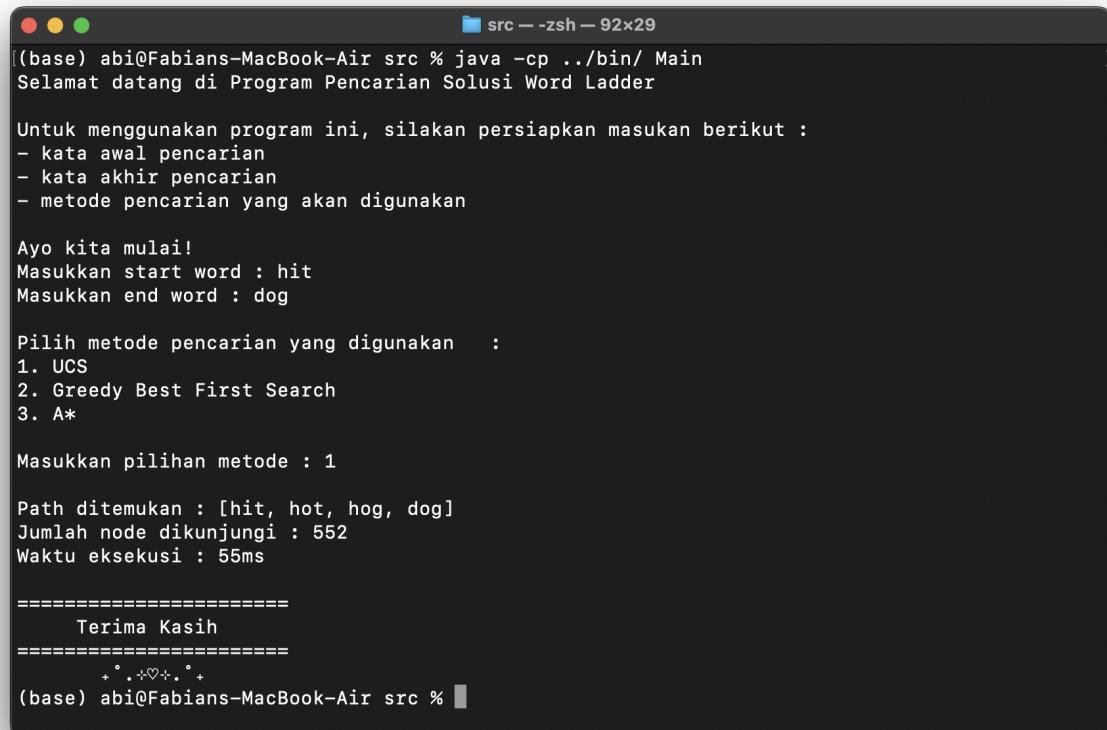
1. Algoritma UCS

a. Percobaan 1

Data

Start Word : hit

End Word : dog



```
src -- zsh - 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : hit
Masukkan end word : dog

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 1

Path ditemukan : [hit, hot, hog, dog]
Jumlah node dikunjungi : 552
Waktu eksekusi : 55ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

b. Percobaan 2

Data

Start Word : engine

End Word : sponge

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : engine
Masukkan end word : sponge

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 1

Pencarian tidak ditemukan.

Waktu eksekusi : 7ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

c. Percobaan 3

Data

Start Word : purple

End Word : yellow

```
src -- zsh -- 92x29
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : purple
Masukkan end word : yellow

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 1

Path ditemukan : [purple, purply, purely, surely, sorely, sorels, morels, motels, botels, be
tels, bevels, levels, levees, levies, bevies, belies, belles, belled, balled, ballet, ballot
, hallot, hallow, fallow, fellow, yellow]
Jumlah node dikunjungi : 7866
Waktu eksekusi : 748ms

=====
Terima Kasih
=====
+ ° . °♥° . ° +
(base) abi@Fabians-MacBook-Air src %
```

d. Percobaan 4

Data

Start Word : soft

End Word : ware

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : soft
Masukkan end word : ware

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 1

Path ditemukan : [soft, sort, wort, wart, ware]
Jumlah node dikunjungi : 973
Waktu eksekusi : 73ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

e. Percobaan 5

Data

Start Word : wolf

End Word : false

```
src - java -cp .. /bin / Main - 92x29
(base) abi@Fabians-MacBook-Air src % java -cp .. /bin / Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

Masukkan start word : wol
```

f. Percobaan 6

Data

Start Word : dozen

End Word : beach

```
src -- zsh -- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : dozen
Masukkan end word : beach

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 1

Path ditemukan : [dozen, dozes, doles, toles, tores, torcs, torch, touch, teuch, teach, beach]
Jumlah node dikunjungi : 6240
Waktu eksekusi : 545ms

=====
Terima Kasih
=====
+ ^ . . ^ + . +
(base) abi@Fabians-MacBook-Air src %
```

2. Algoritma *Greedy Best First Search*

a. Percobaan 1

Data

Start Word : hit

End Word : dog

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : hit
Masukkan end word : dog

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 2

Path ditemukan : [hit, dit, dot, dog]
Jumlah node dikunjungi : 4
Waktu eksekusi : 13ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

b. Percobaan 2

Data

Start Word : engine

End Word : sponge

```
src -- zsh -- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : engine
Masukkan end word : sponge

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan metode : 2

Hasil pencarian tidak ditemukan

Waktu eksekusi : 2ms

=====
Terima Kasih
=====
+ °.~♡~.°+
(base) abi@Fabians-MacBook-Air src %
```

c. Percobaan 3

Data

Start Word : purple

End Word : yellow

```
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : purple
Masukkan end word : yellow

Pilih metode pencarian yang digunakan  :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 2

Hasil pencarian tidak ditemukan

Waktu eksekusi : 3ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

d. Percobaan 4

Data

Start Word : soft

End Word : ware

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : soft
Masukkan end word : ware

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 2

Path ditemukan : [soft, sort, wort, wart, ware]
Jumlah node dikunjungi : 5
Waktu eksekusi : 14ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

e. Percobaan 5

Data

Start Word : wolf

End Word : false

```
src - java -cp .. /bin/ Main - 92x29
(base) abi@Fabians-MacBook-Air src % java -cp .. /bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

Masukkan start word : █
```

f. Percobaan 6

Data

Start Word : dozen

End Word : beach

```
src -- zsh -- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : dozen
Masukkan end word : beach

Pilih metode pencarian yang digunakan  :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 2

Hasil pencarian tidak ditemukan

Waktu eksekusi : 4ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

3. Algoritma A*

a. Percobaan 1

Data

Start Word : hit

End Word : dog

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : hit
Masukkan end word : dog

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 3

Path ditemukan : [hit, dit, din, don, dol, dog]
Jumlah node dikunjungi : 103
Waktu eksekusi : 27ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

b. Percobaan 2

Data

Start Word : engine

End Word : sponge

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : engine
Masukkan end word : sponge

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 3

Pencarian tidak ditemukan.

Waktu eksekusi : 5ms

=====
Terima Kasih
=====
+ ° . ° ° + . ° +
(base) abi@Fabians-MacBook-Air src %
```

c. Percobaan 3

Data

Start Word : purple

End Word : yellow

```
src -- zsh -- 92x29
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : purple
Masukkan end word : yellow

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 3

Path ditemukan : [purple, purply, purely, surely, sorely, sorels, morels, motels, botels, be
tels, bevels, levels, levees, levies, bevies, belies, belied, belled, balled, ballet, ballot
, hallot, hallow, mallow, mellow, yellow]
Jumlah node dikunjungi : 3365
Waktu eksekusi : 217ms

=====
Terima Kasih
=====
+ ° . °♥° . ° +
(base) abi@Fabians-MacBook-Air src %
```

d. Percobaan 4

Data

Start Word : soft

End Word : ware

```
src --zsh-- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp ..//bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : soft
Masukkan end word : ware

Pilih metode pencarian yang digunakan   :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 3

Path ditemukan : [soft, sort, bort, born, bore, bare, ware]
Jumlah node dikunjungi : 275
Waktu eksekusi : 39ms

=====
Terima Kasih
=====
+ ° .~°~+ .°+
(base) abi@Fabians-MacBook-Air src %
```

e. Percobaan 5

Data

Start Word : wolf

End Word : false

```
src - java -cp .. /bin/ Main - 92x29
(base) abi@Fabians-MacBook-Air src % java -cp .. /bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

Masukkan start word : wolf
Masukkan end word : false

Tolong masukkan start word dengan benar.

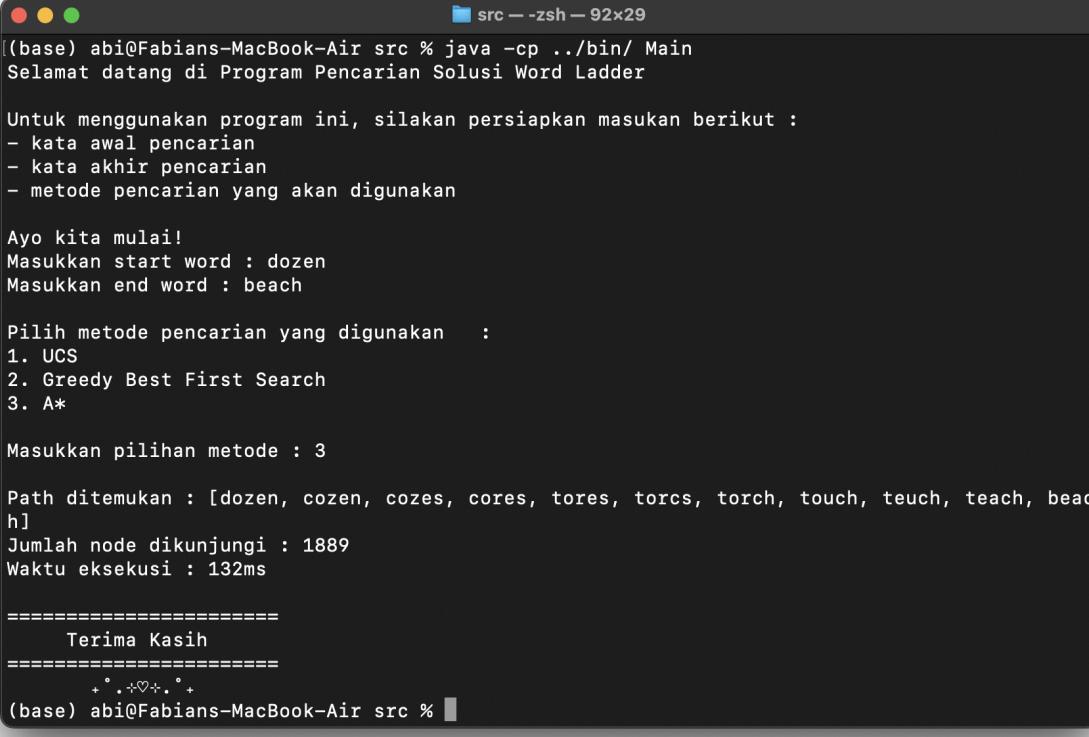
Masukkan start word : 
```

f. Percobaan 6

Data

Start Word : dozen

End Word : beach



```
src -- zsh -- 92x29
(base) abi@Fabians-MacBook-Air src % java -cp .../bin/ Main
Selamat datang di Program Pencarian Solusi Word Ladder

Untuk menggunakan program ini, silakan persiapkan masukan berikut :
- kata awal pencarian
- kata akhir pencarian
- metode pencarian yang akan digunakan

Ayo kita mulai!
Masukkan start word : dozen
Masukkan end word : beach

Pilih metode pencarian yang digunakan  :
1. UCS
2. Greedy Best First Search
3. A*

Masukkan pilihan metode : 3

Path ditemukan : [dozen, cozen, cozes, cores, tores, torcs, torch, touch, teuch, teach, beach]
Jumlah node dikunjungi : 1889
Waktu eksekusi : 132ms

=====
Terima Kasih
=====
+ ^ .-.-^ .-+
(base) abi@Fabians-MacBook-Air src %
```

F. Perbandingan Metode dalam Mencari Solusi

Algoritma penyelesaian permainan Word Ladder dengan metode UCS pasti dapat menemukan solusi (jika solusi ada) karena program melakukan pencarian dengan komplit. Pencarian UCS cenderung menggunakan waktu eksekusi lebih lama dibandingkan algoritma yang lain dan jumlah simpul yang dikunjungi lebih banyak. Namun, dapat dipastikan bahwa solusi yang dihasilkan dari algoritma ini optimal, sebab algoritma ini melakukan pengurutan dengan mengutamakan simpul yang memiliki *cost* paling kecil.

Algoritma *Greedy Best First Search* memiliki waktu eksekusi yang paling rendah, sebab algoritma ini hanya meninjau masing-masing satu *node* pada tiap simpul (simpul yang diambil adalah yang memiliki jarak paling dekat dengan simpul tujuan). Solusi yang dihasilkan algoritma *Greedy Best First Search* tidak pasti optimal dan pencarinya tidak komplit. Sehingga ada kemungkinan solusi tidak ditemukan padahal solusi tersebut ada. Hal ini terjadi karena algoritma ini hanya mempertimbangkan simpul mana yang jaraknya paling dekat dengan simpul tujuan

pada “saat itu”. Namun, algoritma ini memiliki kelebihan dalam hal waktu eksekusi program. Waktu eksekusi program yang cenderung lebih kecil dibandingkan dengan UCS dan A*.

Algoritma A* menggabungkan kelebihan UCS dan *Greedy Best First Search*. Pencarian algoritma A* bersifat komplit. Namun, solusi yang dihasilkan tidak pasti optimal. Waktu eksekusi yang digunakan A* cenderung lebih cepat dibandingkan UCS tetapi lebih lambat dari *Greedy Best First Search*.

Dari penjelasan masing-masing algoritma diatas, dapat disimpulkan bahwa setiap metode pencarian memiliki kelebihan dan kekurangan masing-masing. UCS mengunggulkan solusinya yang optimal dan komplit. Namun, disisi lain UCS juga memiliki waktu eksekusi yang cenderung lambat. *Greedy Best First Search* mengunggulkan waktu eksekusinya yang cenderung cepat. Namun, ia tidak melakukan pencarian secara komplit dan memunculkan kemungkinan terjadi kondisi dimana solusi tidak ditemukan padahal solusinya ada. Algoritma A* juga memiliki kelebihan waktu eksekusi yang lebih cepat dan pencarian yang komplit. Namun, ternyata solusi yang ia hasilkan tidak optimal. Tidak dapat ditentukan algoritma mana yang terbaik karena penggunaan algoritma ditentukan oleh prioritas kepentingan pengguna.

G. Lampiran

Pranala Github : https://github.com/fabianradenta/Tucil3_13522105

Poin	Ya	Tidak
Program berhasil dijalankan.	✓	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma UCS	✓	
Solusi yang diberikan pada algoritma UCS optimal	✓	

Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma <i>Greedy Best First Search</i>	✓	
Program dapat menemukan rangkaian kata dari <i>start word</i> ke <i>end word</i> sesuai aturan permainan dengan algoritma A*	✓	
Solusi yang diberikan pada algoritma A* optimal		✓
[Bonus]: Program memiliki tampilan GUI		✓