LAPORAN TUGAS KECIL III

IF2211 STRATEGI ALGORITMA

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

Dosen:

Ir. Rila Mandala, M. Eng, Ph. D.

Monterico Adrian, S. T., M. T.

Oleh:

13522131 Owen Tobias Sinurat

PROGRAM STUDI TEKNIK INFORMATIKA

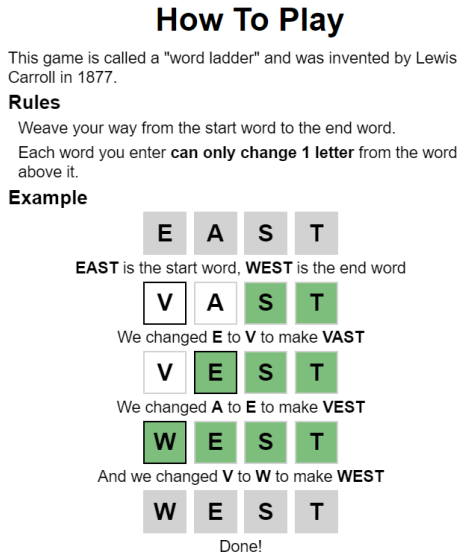
INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

BAB I
DESKRIPSI TUGAS

1.1. Word Ladder

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



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

Permainannya cukup sederhana bukan? Jika belum paham dengan peraturan permainannya, cobalah untuk memainkan permainannya pada link

sumber di atas. Jika sudah paham dengan permainannya, sekarang adalah waktunya kalian untuk membuat sebuah solver permainan tersebut dengan harapan kita dapat menemukan solusi paling optimal untuk menyelesaikan permainan *Word Ladder* ini.

1.2. Spesifikasi Tugas

Berikut adalah spesifikasi tugas yang diberikan:

- Buatlah program dalam bahasa **Java** berbasis **CLI** (*Command Line Interface*) – bonus jika menggunakan GUI – yang dapat menemukan solusi permainan *word ladder* menggunakan algoritma **UCS, *Greedy Best First Search*, dan A***.
- Kata-kata yang dapat dimasukkan harus berbahasa **Inggris**. Cara kalian melakukan validasi sebuah kata dibebaskan, selama kata-kata tersebut benar terdapat pada *dictionary* dan proses validasi tersebut tidak memakan waktu yang terlalu lama.
- Tugas wajib dikerjakan secara individu.
- ***Input* :** Format masukan **dibebaskan**, dengan catatan dijelaskan pada
- README dan laporan. Komponen yang perlu menjadi masukan yaitu. *Start word* dan *end word.*
- Program harus bisa menangani berbagai panjang kata (tidak hanya kata dengan 4 huruf saja seperti Gambar 1)
- Pilihan algoritma yang digunakan (UCS, *Greedy Best First Search,* atau A*)
- ***Output* :**
  Berikut adalah luaran dari program yang diekspektasikan.
  1. *Path* yang dihasilkan dari *start word* ke *end word* (cukup 1 *path* saja)
  2. Banyaknya *node* yang dikunjungi
  3. Waktu eksekusi program
- **Bonus :**
  Pastikan sudah mengerjakan spesifikasi wajib sebelum mengerjakan bonus.
  1. Program dapat berjalan dengan GUI (*Graphical User Interface*) – silakan berkreasi dalam membuat tampilan GUI untuk tucil ini. Untuk kakas GUI dibebaskan asalkan program algoritma UCS, *Greedy Best First Search*, dan A* dalam bahasa Java.

BAB II
ANALISIS DAN IMPLEMENTASI ALGORITMA

2.1 Analisis dan Implementasi Algoritma *Uniform Cost Search (UCS)*

Fungsi UCS menggunakan struktur data graph dan menandai kata yang sudah dikunjungi untuk menghindari kunjungan berulang. Berikut adalah tahapan algoritma UCS yang diimplementasikan:

1. Fungsi menerima parameter berupa kata awal, kata tujuan, dan set kata yang sudah di filter berdasarkan panjang kata awal (karena word ladder hanya menggunakan kata dengan panjang yang sama).
2. Kata awal akan dimasukkan ke dalam openSet yang merupakan Priority Queue terurut berdasarkan ‘harga’-nya. Kata awal akan mempunyai ‘harga’ 0. Harga ini adalah jarak dari kata awal hingga kata ke-n.
3. openSet akan berfungsi sebagai wadah untuk kata-kata yang belum ditelusuri beserta cost-nya, costMap berfungsi menyimpan daftar ‘harga’ setiap kata yang telah ditelusuri, dan parentMap berfungsi menyimpan daftar parent dari setiap kata yang sudah ditelusuri. Kenapa ada costMap kalau openSet menyimpan kata dan cost nya? Alasannya kemudahan retrieval atau pengambilan nilai cost, karena openSet merupakan priority queue, diharuskan iterasi untuk retreival berdasarkan key nya.
4. Setelah konfigurasi awal, fungsi akan mengambil kata dengan prioritas tertinggi dari openSet dan mengecek apakah kata itu adalah kata tujuannya, apabila iya, maka akan langsung dikembalikan dalam bentuk path menggunakan fungsi reconstructPath.

5. Apabila tidak, kata akan dimasukkan ke closedSet untuk menandai bahwa kata sudah ditelusuri.
6. Lalu fungsi akan mengiterasi list neighbor dari kata (hasil perhitungan generateNeighbors). Pada setiap iterasi, akan dicek apakah neighbor sudah ditelusuri, kalau belum, akan dicek apakah neighbor sudah tercatat pada costMap atau cost baru dari kata yang ada pada path sekarang lebih kecil dari cost dari kata yang sama pada path yang mungkin sudah ada pada map costMap. Hal ini memastikan setiap kata menyimpan path terpendeknya, cost terkecilnya, dan dimasukkan ke dalam openSet.
7. Tahapan 4-6 akan diulangi terus menerus hingga semua kata telah ditelusuri.
8. Kalau semua kata sudah ditelusuri dan belum sampai ke kata tujuan, fungsi akan mengembalikan list kosong.

```
public class Pair<K, V> {
    private final K first;
    private final V second;

    public Pair(K first, V second) {
        this.first = first;
        this.second = second;
    }

    public K getFirst() {
        return first;
    }

    public V getSecond() {
        return second;
    }
}

public List<String> uniformCostSearch(String start,
String end, Set<String> wordSet) {
```

```
        Map<String, String> parentMap = new HashMap<>();
// Parent map to reconstruct path
         Map<String, Integer> costMap = new HashMap<>();
// Cost map to store cumulative costs

          PriorityQueue<Pair<String, Integer>> openSet =
new
PriorityQueue<>(Comparator.comparingInt(Pair::getSecond)
);
        Set<String> closedSet = new HashSet<>();

        openSet.offer(new Pair<>(start, 0));
        parentMap.put(start, null);
        costMap.put(start, 0);

        while (!openSet.isEmpty()) {
                    Pair<String, Integer> currentPair =
openSet.poll();
            String currentWord = currentPair.getFirst();
            int currentCost = currentPair.getSecond();

            if (currentWord.equals(end)) {
                // Reconstruct and return the path
                return reconstructPath(parentMap, end);
            }

            closedSet.add(currentWord);

                                for (String neighbor :
generateNeighbors(currentWord, wordSet)) {
                     int newCost = currentCost + 1; //
Assuming uniform cost of 1 for each transition

                    if (!closedSet.contains(neighbor) &&
(!costMap.containsKey(neighbor) || newCost <
costMap.get(neighbor))) {
                                    parentMap.put(neighbor,
currentWord);
                     costMap.put(neighbor, newCost);
```

```
                        openSet.offer(new Pair<>(neighbor,
newCost));
                    }
                }
            }
            // No path found
            return Collections.emptyList();
        }
```

2.2 Analisis dan Implementasi Algoritma *Greedy Best First Search*

Fungsi GBFS menggunakan struktur data graph dan menandai kata yang sudah dikunjungi untuk menghindari kunjungan berulang. Berikut adalah tahapan algoritma GBFS yang diimplementasikan:

9. Fungsi menerima parameter berupa kata awal, kata tujuan, dan set kata yang sudah di filter berdasarkan panjang kata awal (karena word ladder hanya menggunakan kata dengan panjang yang sama).
10. Kata awal akan dimasukkan ke dalam openSet yang merupakan Priority Queue terurut berdasarkan heuristik-nya. Kata awal akan mempunyai heuristik 0.
11. openSet akan berfungsi sebagai wadah untuk kata-kata yang akan ditelusuri dan parentMap berfungsi menyimpan daftar parent dari setiap kata yang sudah ditelusuri.
12. Setelah konfigurasi awal, fungsi akan mengambil kata dengan prioritas tertinggi dari openSet dan mengecek apakah kata itu adalah kata tujuannya, apabila iya, maka akan langsung dikembalikan dalam bentuk path menggunakan fungsi reconstructPath.
13. Apabila tidak, fungsi akan mengiterasi list neighbor dari kata (hasil perhitungan generateNeighbors). Pada setiap iterasi, akan dicek apakah neighbor sudah ditelusuri, kalau belum, neighbor akan ditambahkan ke parentMap dan openSet.

14. Lalu Tahapan 4-6 akan diulangi terus menerus hingga semua kata telah ditelusuri.
15. Kalau semua kata sudah ditelusuri dan belum sampai ke kata tujuan, fungsi akan mengembalikan list kosong.

```
public List<String> greedyBestFirstSearch(String start,
String end, Set<String> wordSet) {
        Map<String, String> parentMap = new HashMap<>();
                  PriorityQueue<String> openSet = new
PriorityQueue<>(Comparator.comparingInt(node ->
heuristic(node, end)));

        openSet.offer(start);
        parentMap.put(start, null);

        while (!openSet.isEmpty()) {
            String current = openSet.poll();
            if (current.equals(end)) {
                // Reconstruct and return the path
                       return reconstructPath(parentMap,
current);
            }
                               for (String neighbor :
generateNeighbors(current, wordSet)) {
                if (!parentMap.containsKey(neighbor)) {
                    parentMap.put(neighbor, current);
                    openSet.offer(neighbor);
                }
            }
        }
        return Collections.emptyList();
    }
```

2.3 Analisis dan Implementasi Algoritma A*

Fungsi aStar menggunakan struktur data graph dan menandai kata yang sudah dikunjungi untuk menghindari kunjungan berulang. Berikut adalah tahapan algoritma A* yang diimplementasikan:

16. Fungsi menerima parameter berupa kata awal, kata tujuan, dan set kata yang sudah di filter berdasarkan panjang kata awal (karena word ladder hanya menggunakan kata dengan panjang yang sama).
17. Kata awal akan dimasukkan ke dalam openSet yang merupakan Priority Queue terurut berdasarkan fScore nya. Kata awal akan mempunyai gScore 0.
18. openSet akan berfungsi sebagai wadah untuk kata-kata yang belum ditelusuri dan closedSet adalah wadah untuk kata-kata yang sudah ditelusuri.
19. Setelah konfigurasi awal, fungsi akan mengambil kata dengan prioritas tertinggi dari openSet dan mengecek apakah kata itu adalah kata tujuannya, apabila iya, maka akan langsung dikembalikan dalam bentuk path menggunakan fungsi reconstructPath.
20. Apabila tidak, kata akan dimasukkan ke closedSet untuk menandai bahwa kata sudah ditelusuri.
21. Lalu fungsi akan mengiterasi list neighbor dari kata (hasil perhitungan generateNeighbors). Pada setiap iterasi, akan dicek apakah neighbor sudah ditelusuri, kalau belum, akan dibandingkan g score tentatif dari kata yang ada pada path sekarang dengan g score dari kata yang sama pada path yang mungkin sudah ada pada map g score. Hal ini memastikan setiap kata menyimpan path terpendeknya, g score dan f score terkecilnya, dan dimasukkan ke dalam openSet.
22. Tahapan 4-6 akan diulangi terus menerus hingga semua kata telah ditelusuri.
23. Kalau semua kata sudah ditelusuri dan belum sampai ke kata tujuan, fungsi akan mengembalikan list kosong.

```
public List<String> aStar(String start, String end,
Set<String> wordSet) {
```

```
        Map<String, String> parentMap = new HashMap<>();
        Map<String, Integer> gScore = new HashMap<>();
        Map<String, Integer> fScore = new HashMap<>();

                    PriorityQueue<String> openSet = new
PriorityQueue<>(Comparator.comparingInt(fScore::get));
        Set<String> closedSet = new HashSet<>();

        openSet.offer(start);
        gScore.put(start, 0);
        fScore.put(start, heuristic(start, end));

        while (!openSet.isEmpty()) {
            String current = openSet.poll();
            if (current.equals(end)) {
                        return reconstructPath(parentMap,
current);
            }
            closedSet.add(current);
                                for (String neighbor :
generateNeighbors(current, wordSet)) {
                if (!closedSet.contains(neighbor)) {
                                    int tentativeGScore =
gScore.getOrDefault(current, Integer.MAX_VALUE) + 1;
                                    if (tentativeGScore <
gScore.getOrDefault(neighbor, Integer.MAX_VALUE)) {
                                        parentMap.put(neighbor,
current);
                                        gScore.put(neighbor,
tentativeGScore);
                                        fScore.put(neighbor,
tentativeGScore + heuristic(neighbor, end));
                            openSet.offer(neighbor);
                        }
                }
            }
        }
        return Collections.emptyList();
    }
```

2.4 *Implementasi Fungsi Umum*

Fungsi heuristic ini digunakan untuk mengkalkulasi heuristik yang digunakan pada algoritma GBFS dan A*. Heuristik dihitung berdasarkan perbedaan huruf pada kedua kata, semakin banyak huruf yang berbeda, semakin tinggi pula heuristiknya.

```
    private int heuristic(String word1, String word2) {
        int diffCount = 0;
        for (int i = 0; i < word1.length(); i++) {
            if (word1.charAt(i) != word2.charAt(i)) {
                diffCount++;
            }
        }
        return diffCount;
    }
```

Fungsi generateNeighbors digunakan untuk membangkitkan children dari suatu node (pada kasus ini sebuah ‘word’). Proses pembangkitannya adalah dengan mengganti setiap huruf pada ‘word’ lalu mengecek apakah kata baru tersebut ada dalam ‘wordSet’. Proses ini lebih cepat dibandingkan mengiterasi setiap kata yang ada pada ‘wordset’ dan mengecek apakah kata tersebut memenuhi syarat children (berbeda hanya 1 huruf dengan ‘word’). Kuncinya berada pada penggunaan wordSet.contains() yang mencari suatu nilai pada sebuah Hash Set dengan kompleksitas waktu O(1), sehingga membuat kompleksitas waktu generateNeighbors adalah T(26*n) yaitu O(n) dengan n adalah panjang ‘word’. Sementara kompleksitas waktu jika mengiterasi ‘wordSet’ adalah O(m*n) dengan m adalah banyak kata dalam ‘wordSet’ dan n adalah panjang ‘word’.

```
private Set<String> generateNeighbors(String word,
Set<String> wordSet) {
```

```
        Set<String> neighbors = new HashSet<>();
        for (int i = 0; i < word.length(); i++) {
            char[] wordArray = word.toCharArray();
            for (char c = 'a'; c <= 'z'; c++) {
                if (c != word.charAt(i)) {
                    wordArray[i] = c;
                                String neighbor = new
String(wordArray);
                    if (wordSet.contains(neighbor)) {
                        neighbors.add(neighbor);
                    }
                }
            }
        }
        return neighbors;
    }
```

Fungsi reconstructPath digunakan untuk membangun path hasil kalkulasi ketiga algoritma. Pembangunan path dilakukan dari ‘end’ yang merupakan node yang sedang ditempati sekarang dan menjadi tujuan akhir dari solusi word ladder. Path dibangun dengan menambahkan parent dari ‘current’ ke dalam list hingga mencapai titik awal yang berarti tidak lagi memiliki parent. Lalu, list akan di reverse untuk mendapatkan path yang berurutan dari titik awal sampai ke titik akhir.

```
// Method to reconstruct the path from the parent map
      private List<String> reconstructPath(Map<String,
String> parentMap, String end) {
        List<String> path = new ArrayList<>();
        String current = end;
        while (current != null) {
            path.add(current);
            current = parentMap.get(current);
        }
        Collections.reverse(path);
        return path;
    }
```

2.5 Implementasi Class Main

Class main digunakan sebagai class utama untuk menjalankan program. Proses eksekusinya adalah class akan mendeklarasikan variabel-variable yang akan dibutuhkan nantinya, membaca file dictionary, menjalankan loop, meminta input, filtrasi dataset, dan memanggil fungsi algoritma, akhirnya mencetak hasil sesuai pengembalian dari fungsi algoritma.

```
import java.util.Map;
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
public class Main {
    public static Scanner input;

    public static void main(String[] args) {
        String start,end, algorithm;
        List<String> path;
        long startTime,endTime,executionTime;
                    WordLadderSolver solver = new
WordLadderSolver();
                              Set<String> wordSet =
readWordsFromFile("dictionary.txt");
        input = new Scanner(System.in);
        while(true){
            System.out.println("\nStart word: ");
            start = input.next().toLowerCase();
            System.out.println("End word: ");
            end = input.next().toLowerCase();
            if(start.length() != end.length()){
                    System.out.println("Words must be of
equal length");
                continue;
                   }else if(!wordSet.contains(start) ||
!wordSet.contains(end)){
```

```
                        System.out.println("Words not in
dictionary");
                continue;
            }
                         Set<String> filteredWordSet =
filterWordsByLength(wordSet, start.length());
            System.out.println("1. UCS");
            System.out.println("2. GBFS");
            System.out.println("3. A*");
            System.out.println("Algorithm: ");
            algorithm = input.next().toLowerCase();
                            if(algorithm.equals("1") ||
algorithm.equals("ucs")){
                         System.out.println("Uniform Cost
Search");
                startTime = System.currentTimeMillis();
                 path = solver.uniformCostSearch(start,
end, filteredWordSet);
                endTime = System.currentTimeMillis();
            }
                        else if(algorithm.equals("2") ||
algorithm.equals("gbfs")){
                  System.out.println("Greedy Best First
Search");
                startTime = System.currentTimeMillis();
                                                path =
solver.greedyBestFirstSearch(start,                end,
filteredWordSet);
                endTime = System.currentTimeMillis();
            }
                        else if(algorithm.equals("3") ||
algorithm.equals("a*")){
                System.out.println("A* Search");
                 startTime = System.currentTimeMillis();
                        path = solver.aStar(start, end,
filteredWordSet);
                 endTime = System.currentTimeMillis();
            }
            else{
                System.out.println("Invalid input");
```

```
                continue;
            }
            executionTime = endTime - startTime;
            System.out.println("\n--- Results ---");
                System.out.println("Execution time: " +
executionTime + " milliseconds");
                  System.out.println("Path length: " +
path.size());
            if (path.isEmpty()) {
                System.out.println("No path found.");
            } else {
                System.out.println("Path found: " + path
+ "");
            }
            System.out.println("--- ****** ---\n");

        }

    }


    // Method to filter words in a set by length
                    public     static     Set<String>
filterWordsByLength(Set<String>       wordSet,       int
targetLength) {
        Set<String> filteredSet = new HashSet<>();
        for (String word : wordSet) {
            if (word.length() == targetLength) {
                filteredSet.add(word);
            }
        }
        return filteredSet;
    }

     // Method to read words from a .txt file and store
them in a set
     public static Set<String> readWordsFromFile(String
filePath) {
        Set<String> wordSet = new HashSet<>();
```

```
                    try (BufferedReader reader = new
BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                // Process each line to extract words
                // Example: split by space
                String[] words = line.split("\\s+");
                Collections.addAll(wordSet, words);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return wordSet;
    }
}
```

2.6 Pertanyaan

- Definisi dari *f(n)* dan *g(n)*, sesuai dengan salindia kuliah.
  - *f(n)* adalah perkiraan 'harga' total dari titik awal ke titik tujuan.
  - *g(n)* adalah 'harga' jalur dari titik awal ke titik n.
- Apakah heuristik yang digunakan pada algoritma A* *admissible*? Jelaskan sesuai definisi *admissible* dari salindia kuliah.
  - Pada kasus word ladder, jika heuristik dihitung berdasarkan jumlah huruf yang berbeda antar kata, maka heuristik bersifat admissible karena h(n) selalu underestimate h*(n) sehingga dapat dipastikan bahwa algoritma A* akan selalu memberikan hasil yang optimal.
- Pada kasus *word ladder*, apakah algoritma UCS sama dengan BFS? (dalam artian urutan *node* yang dibangkitkan dan *path* yang dihasilkan sama)
  - Ya, karena jarak antar node (pada kasus ini kata) yang bertetangga (pada kasus ini yang memiliki 1 huruf berbeda) adalah konstan 1.
- Secara teoritis, apakah algoritma A* lebih efisien dibandingkan dengan algoritma UCS pada kasus *word ladder*?
  - Berdasarkan time dan space complexity, A* lebih efisien karena A* selalu memberikan solusi optimal (heuristik harus admissible). Ditambah UCS pada word ladder pada dasarnya adalah BFS karena jarak antar node (pada kasus ini kata) yang bertetanggaan adalah konstan 1.

- Secara teoritis, apakah algoritma *Greedy Best First Search* menjamin solusi optimal untuk persoalan *word ladder*?
  - Tidak, karena itulah dinamakan Greedy Best First Search, sifatnya sama dengan algoritma Greedy biasa yaitu mengambil langkah optimum lokal dan berharap mendapatkan optimum global. Algoritma GBFS tidak mempertimbangkan sejarah path dari sebuah node, melainkan mengacu hanya pada heuristiknya.

BAB III
PENGUJIAN DAN HASIL

3.1 Pengujian Algoritma

1. Pengujian 1

   Start : whom

   End : when

```
Start word:
whom
End word:
when
1. UCS
2. GBFS
3. A*
Algorithm:
ucs
Uniform Cost Search

--- Results ---
Execution time: 5 milliseconds
Path length: 4
Path found: [whom, whim, whin, when]
--- ****** ---
```

Gambar 1. UCS

```
Start word:
whom
End word:
when
1. UCS
2. GBFS
3. A*
Algorithm:
gbfs
Greedy Best First Search

--- Results ---
Execution time: 1 milliseconds
Path length: 4
Path found: [whom, whim, whin, when]
--- ****** ---
```

Gambar 2. GBFS

```
Start word:
whom
End word:
when
1. UCS
2. GBFS
3. A*
Algorithm:
A*
A* Search

--- Results ---
Execution time: 2 milliseconds
Path length: 4
Path found: [whom, whim, whin, when]
--- ****** ---
```

Gambar 3. A*

2. Pengujian 2

   Start : moist

   End : boils

```
Start word:
moist
End word:
boils
1. UCS
2. GBFS
3. A*
Algorithm:
ucs
Uniform Cost Search

--- Results ---
Execution time: 5 milliseconds
Path length: 7
Path found: [moist, joist, joint, joins, coins, coils, boils]
--- ****** ---
```

Gambar 1. UCS

```
Start word:
moist
End word:
boils
1. UCS
2. GBFS
3. A*
Algorithm:
2
Greedy Best First Search

--- Results ---
Execution time: 5 milliseconds
Path length: 7
Path found: [moist, joist, joint, joins, foins, foils, boils]
--- ****** ---
```

Gambar 2. GBFS

```
Start word:
moist
End word:
boils
1. UCS
2. GBFS
3. A*
Algorithm:
3
A* Search

--- Results ---
Execution time: 3 milliseconds
Path length: 7
Path found: [moist, joist, joint, joins, foins, foils, boils]
--- ****** ---
```

Gambar 3. A*

3. Pengujian 3

   Start : melons

   End : milker

```
Start word:
melons
End word:
milker
1. UCS
2. GBFS
3. A*
Algorithm:
1
Uniform Cost Search

--- Results ---
Execution time: 27 milliseconds
Path length: 17
Path found: [melons, mesons, masons, macons, racons, recons, redons, redoes, redyes, redyed, redded, redden, ridden, midden, milden
, milder, milker]
--- ****** ---
```

Gambar 1. UCS

```
Start word:
melons
End word:
milker
1. UCS
2. GBFS
3. A*
Algorithm:
2
Greedy Best First Search

--- Results ---
Execution time: 1 milliseconds
Path length: 17
Path found: [melons, mesons, masons, macons, racons, recons, redons, redoes, redyes, redyed, redded, redder, ridder, widder, wilder
, milder, milker]
--- ****** ---
```

Gambar 2. GBFS

```
Start word:
melons
End word:
milker
1. UCS
2. GBFS
3. A*
Algorithm:
3
A* Search

--- Results ---
Execution time: 5 milliseconds
Path length: 17
Path found: [melons, mesons, masons, macons, racons, recons, redons, redoes, redyes, redyed, redded, rended, mended, melded, melder
, milder, milker]
--- ****** ---
```

Gambar 3. A*

4. Pengujian 4

   Start : crown

   End : rings

```
Start word:
crown
End word:
rings
1. UCS
2. GBFS
3. A*
Algorithm:
1
Uniform Cost Search

--- Results ---
Execution time: 19 milliseconds
Path length: 9
Path found: [crown, crows, prows, prods, poods, ponds, pongs, pings, rings]
--- ****** ---
```

Gambar 1. UCS

```
Start word:
crown
End word:
rings
1. UCS
2. GBFS
3. A*
Algorithm:
2
Greedy Best First Search

--- Results ---
Execution time: 0 milliseconds
Path length: 13
Path found: [crown, crows, grows, grogs, frogs, flogs, clogs, clons, cions, lions, linns, lings, rings]
--- ****** ---
```

Gambar 2. GBFS

```
Start word:
crown
End word:
rings
1. UCS
2. GBFS
3. A*
Algorithm:
3
A* Search

--- Results ---
Execution time: 1 milliseconds
Path length: 9
Path found: [crown, crows, prows, prods, poods, ponds, pongs, pings, rings]
--- ****** ---
```

Gambar 3. A*

5. Pengujian 5

   Start : manta

   End : scams

```
Start word:
manta
End word:
scams
1. UCS
2. GBFS
3. A*
Algorithm:
1
Uniform Cost Search

--- Results ---
Execution time: 18 milliseconds
Path length: 11
Path found: [manta, menta, yenta, yente, rente, rents, tents, teats, teams, seams, scams]
--- ****** ---
```

Gambar 1. UCS

```
Start word:
manta
End word:
scams
1. UCS
2. GBFS
3. A*
Algorithm:
2
Greedy Best First Search

--- Results ---
Execution time: 0 milliseconds
Path length: 12
Path found: [manta, manna, canna, canny, carny, carns, barns, barms, berms, beams, seams, scams]
--- ****** ---
```

Gambar 2. GBFS

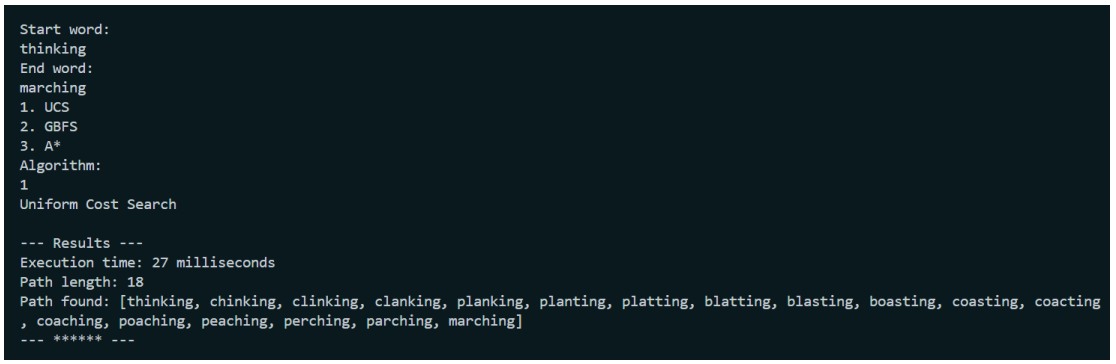
```
Start word:
manta
End word:
scams
1. UCS
2. GBFS
3. A*
Algorithm:
3
A* Search

--- Results ---
Execution time: 1 milliseconds
Path length: 11
Path found: [manta, menta, yenta, yente, rente, rents, bents, beats, beams, seams, scams]
--- ****** ---
```

Gambar 3. A*

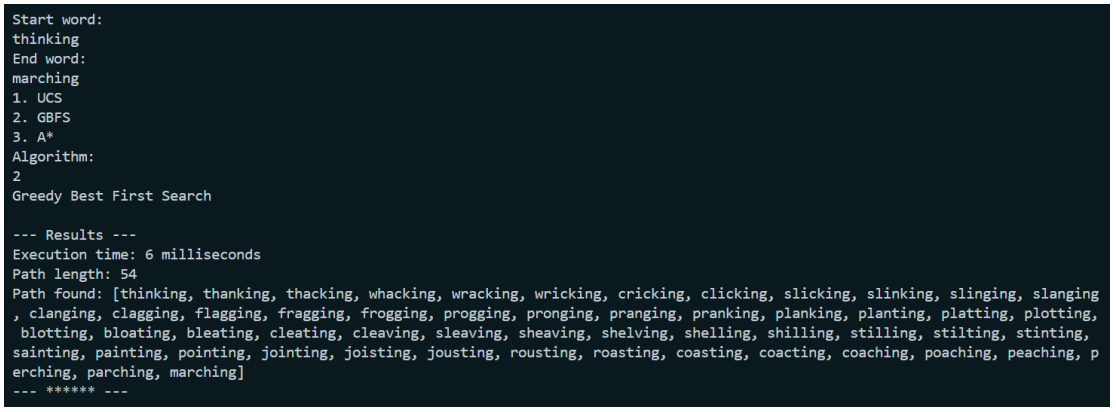
6. Pengujian 6

   Start : thinking

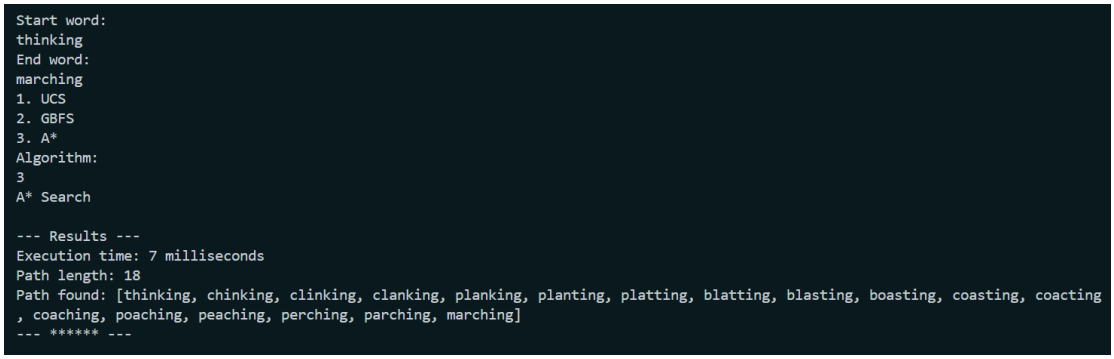
   End : marching



Gambar 1. UCS



Gambar 2. GBFS

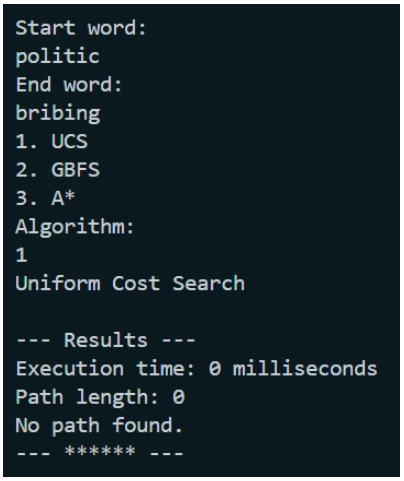


Gambar 3. A*

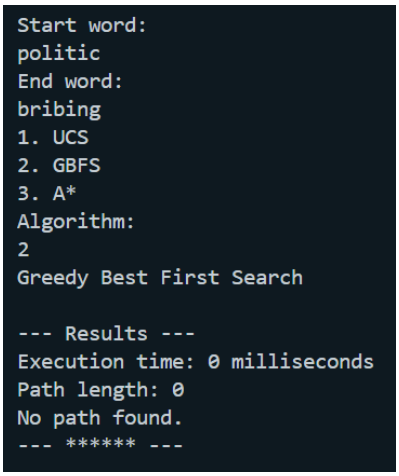
7. Pengujian 7

   Start : politic

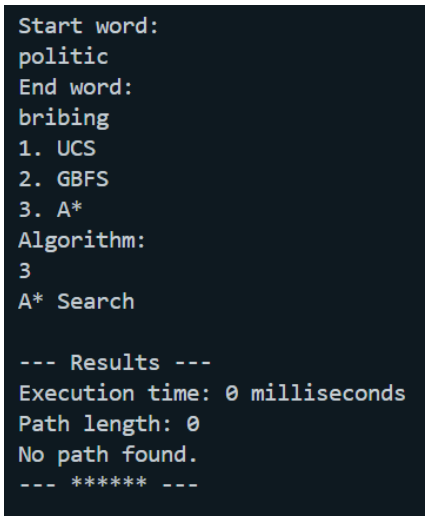
   End : bribing



Gambar 1. UCS



Gambar 2. GBFS



Gambar 3. A*

3.2 Analisis Perbandingan Hasil Pengujian

Berdasarkan hasil pengujian sebanyak 7 test case untuk ketiga algoritma, dapat terlihat jelas perbedaannya. Berikut adalah rincian masing-masing algoritma:

1. UCS

   Optimalitas : Memberikan hasil yang paling optimal karena UCS pada word ladder pada dasarnya adalah BFS disebabkan jarak antar node (atau pada kasus ini kata) yang berhubungan adalah 1.

   Waktu : Memakan waktu paling lama karena mengunjungi semua node.

   Memori : Memakan memori paling banyak karena mengunjungi semua node.

2. GBFS

   Optimalitas : Tidak menjamin solusi optimal.

   Waktu : Memakan waktu paling singkat karena tidak backtrack.

   Memori : Memakan memori paling sedikit karena tidak backtrack sehingga tidak mengunjungi banyak node.

3. A*

   Optimalitas : Hampir selalu menjamin solusi optimal (Saya belum menemukan test case yang membuktikan, tetapi berdasarkan logika, pasti ada kasus dimana UCS memberikan solusi lebih pendek dibandingkan A*).

   Waktu : Memakan waktu yang lebih banyak daripada GBFS, tetapi lebih sedikit daripada UCS.

   Memori : Memakan memori yang lebih banyak daripada GBFS, tetapi lebih sedikit daripada UCS.

LAMPIRAN

6.1 Ketercapaian

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

6.2 GitHub Repository

https://github.com/owenthe10x/Tucil3_13522131