TUGAS KECIL 3

STRATEGI ALGORITMA

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

Disusun Oleh:

Evelyn Yosiana (13522083)

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

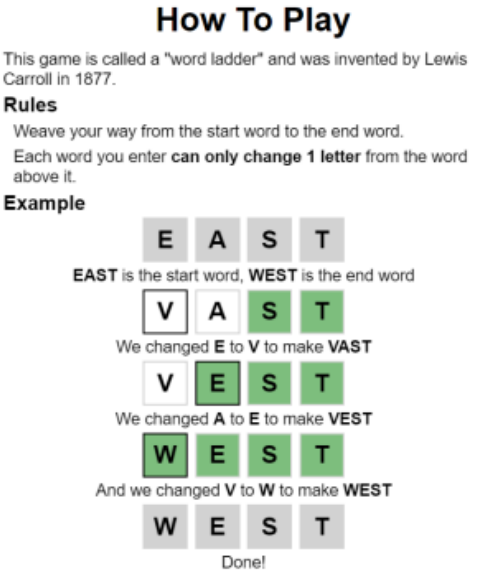
Institut Teknologi Bandung

DAFTAR ISI

BAB 1

Deskripsi Masalah

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 1.1 Ilustrasi dan Peraturan Permainan Word Ladder

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.

BAB 2

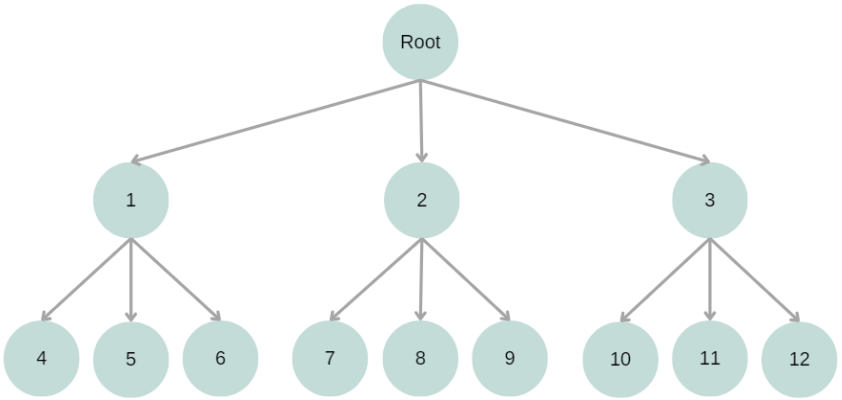
Analisis dan Implementasi

2.1. Analisis dan Implementasi algoritma UCS, Greedy Best First Search, dan A* dalam Pencarian Solusi Word Ladder

2.1.1. Algoritma Uniform Cost Search

Uniform Cost Search (UCS) merupakan salah satu algoritma pencarian yang digunakan untuk mencari jalur terpendek dalam sebuah graf. Algoritma UCS mempertimbangkan biaya terendah dari simpul akar ke setiap simpul lainnya. Dengan menggunakan algoritma UCS, simpul dengan biaya terendah di-expandsi terlebih dahulu, kemudian simpul dengan biaya terendah kedua, ketiga, dan seterusnya sampai mencapai simpul tujuan. Dalam penggunaannya, algoritma UCS mempertimbangkan fungsi g(n). **Fungsi g(n) sendiri menyatakan biaya yang diperlukan untuk mencapai simpul tujuan dari simpul akar**. Algoritma UCS tidak menggunakan heuristik untuk mencapai simpul tujuan.

Pada kasus pencarian solusi Word Ladder, seluruh simpul yang mungkin dicapai dari simpul akar dibangkitkan seluruhnya. Karena tidak menggunakan heuristik, simpul-simpul akan dibangkitkan pasti menggunakan urutan seperti ini.



Berdasarkan gambar di atas, dapat disimpulkan bahwa **pada kasus pencarian solusi Word Ladder, urutan simpul yang dibangkitkan dan path yang dihasilkan oleh algoritma UCS sama dengan BFS.**

Implementasi algoritma UCS pada pencarian solusi Word Ladder kali ini dapat dijelaskan sebagai berikut.

1. Mula-mula, dibuat simpul berisi kata awal dan jalur berisi simpul tersebut.
2. Dari simpul awal, simpul-simpul yang bertetangga dengan simpul awal (simpul dengan kata yang memiliki perbedaan satu huruf dengan simpul awal) dibangkitkan seluruhnya. Biaya dari satu simpul ke simpul lain yaitu 1.
3. Untuk setiap simpul yang dibangkitkan, dibuat duplikat dari jalur awal yang kemudian ditambahkan dengan simpul yang dibangkitkan. Biaya sekarang ditambahkan ke biaya akumulatif jalur tersebut.
4. Seluruh jalur yang telah dibuat dimasukkan ke dalam *priority queue*.
5. Diambil jalur paling depan dari *priority queue* (jalur dengan biaya terkecil), kemudian dilakukan proses langkah nomor 2 sampai 4.
6. Proses nomor 5 diulangi sampai didapatkan simpul tujuan pada elemen pertama *priority queue.*

2.1.2. Algoritma Greedy Breadth-first search

Algoritma Greedy Best-first search (GBFS) merupakan salah satu variasi algoritma pencarian graf dengan prinsip memilih simpul berdasarkan heuristik yang menunjukkan simpul tersebut paling dekat dengan solusi optimal. Algoritma ini mempertimbangkan **h(n) yang merupakan fungsi heuristik**. Dalam pencarian solusi Word Ladder, h(n) merepresentasikan beda huruf antara simpul yang sekarang dengan simpul tujuan. Kelebihan dari algoritma ini terletak pada kecepatan waktu eksekusinya. **Namun algoritma ini tidak menjamin solusi optimal, termasuk pada kasus pencarian solusi Word Ladder.**

Implementasi algoritma GBFS pada pencarian solusi Word Ladder kali ini dapat dijelaskan sebagai berikut.

1. Mula-mula, dibuat simpul berisi kata awal dan jalur berisi simpul tersebut.
2. Dari simpul awal, dicari simpul tetangga dengan heuristik paling rendah. Tambahkan simpul tersebut ke jalur.
3. Lakukan langkah 2 dengan simpul awal merupakan simpul yang sedang di-expand sampai solusi ditemukan. Pada kasus ini, simpul yang sama bisa di-expand lebih dari satu kali.

Implementasi algoritma GBFS sering kali menimbulkan pertanyaan haruskah langkah yang dilakukan backtrack. Namun pada implementasi ini, algoritma GBFS dibuat tidak backtrack, sesuai dengan PPT perkuliahan. Karena itulah pada beberapa kasus algoritma ini tidak menghasilkan solusi di saat kasus tersebut memiliki solusi jika menggunakan dua algoritma yang lain.

2.1.3. Algoritma A*

Algoritma A* merupakan salah satu algoritma pencarian graf untuk menemukan jalur terpendek antara dua simpul dengan menggabungkan prinsip GBFS dan UCS yang telah dijelaskan sebelumnya. Algoritma A* mempertimbangkan fungsi f(n), dimana **f(n) sendiri merupakan penjumlahan dari g(n) dan h(n) yang telah dijelaskan di atas**. Keunggulan dari algoritma ini yaitu adanya jaminan solusi yang ditemukan optimal jika heuristiknya admissible. Admissible artinya biaya estimasi tidak pernah lebih besar daripada biaya sebenarnya.

Dalam kasus pencarian solusi Word Ladder, heuristik yang digunakan dalam algoritma ini admissible. Hal ini dikarenakan tiap pembangkitan simpul baru, perbedaan huruf yang diperbolehkan hanya 1, sehingga jika dua buah kata memiliki perbedaan n huruf, diperlukan minimal n langkah untuk mencapai hasil tujuan. Dengan demikian biaya sebenarnya akan selalu lebih besar atau sama dengan biaya estimasinya.

Dalam implementasinya, A* mempertimbangkan biaya heuristik yang memungkinkan algoritma tersebut lebih efisien dalam mencari solusi. Di sisi lain, algoritma UCS tidak mempertimbangkan heuristik dan dapat mengarah pada ekspansi yang kurang efisien. **Karena itulah, pada kasus Word Ladder, algoritma A* sering kali lebih efisien dibandingkan dengan algoritma UCS.**

Implementasi algoritma A* pada pencarian solusi Word Ladder kali ini dapat dijelaskan sebagai berikut.

1. Mula-mula, dibuat simpul berisi kata awal dan jalur berisi simpul tersebut.
2. Dari simpul awal, simpul-simpul yang bertetangga dengan simpul awal (simpul dengan kata yang memiliki perbedaan satu huruf dengan simpul awal) dibangkitkan seluruhnya. Biaya dari satu simpul ke simpul lain yaitu perbedaan huruf simpul sekarang dengan simpul tujuan ditambah dengan 1 (g(n)).
3. Untuk setiap simpul yang dibangkitkan, dibuat duplikat dari jalur awal yang kemudian ditambahkan dengan simpul yang dibangkitkan. Biaya sekarang ditambahkan ke biaya akumulatif jalur tersebut.
4. Seluruh jalur yang telah dibuat dimasukkan ke dalam *priority queue*.
5. Diambil jalur paling depan dari *priority queue* (jalur dengan biaya terkecil), kemudian dilakukan proses langkah nomor 2 sampai 4.
6. Proses nomor 5 diulangi sampai didapatkan simpul tujuan pada elemen pertama *priority queue.*

BAB 3

Source Code

3.1. Kelas Node

Kelas Node mengimplementasikan suatu jalur dari solusi atau kandidat solusi. Kelas Node memiliki atribut:

- Cost (integer), untuk menyimpan f(n), g(n), ataupun kombinasi keduanya secara akumulatif;
- List (list of string), untuk menyimpan jalur dari kandidat solusi;
- Target (string), untuk menyimpan kata target.

Kelas Node memiliki beberapa method, antara lain sebagai berikut.

- getCost(), memberikan return atribut cost (integer);
- addGn(), menambahkan cost dengan satu, digunakan untuk algoritma UCS dan A*.
- addHn(int cost), menambahkan atribut cost dengan cost pada parameter, digunakan untuk algoritma Greedy dan A*.
- getList(), mengembalikan list of string yang berisi kumpulan kata yang membentuk jalur ke kata tujuan.
- addList(String word), menambahkan word pada parameter ke dalam atribut list di bagian akhir.
- print(), menampilkan list of string ke layar.
- getSize(), mengembalikan ukuran dari atribut list.
- isFound(String word), mengecek apakah kata pada parameter sudah terdapat pada atribut list atau belum. Mengembalikan nilai true jika sudah ada dan nilai false jika belum ada.

Berikut adalah *source code* dari kelas Node.

```java
public class Node {
    private int cost;
    private List<String> list;
    private String target;

    // Ctor
    public Node(String word, String target){
        this.list = new ArrayList<String>();
        list.add(word);
        this.cost = 0;
        this.target = target;
    }

    // Cctor
    public Node(Node other){
        this.list = new ArrayList<String>();
        this.cost = other.cost;
        for (String word : other.getList()){
            this.list.add(word);
        }
        this.target = other.target;
    }

    // Return cost
    public int getCost(){
        return this.cost;
    }

    // Cost += 1
    public void addGn(){
        this.cost++;
    }

    // Cost += input cost
    public void addHn(int cost){
        this.cost = cost;
    }
```

```java
    // Return list of node
    public List<String> getList(){
        return this.list;
    }

    // Tambahin word ke list
    public void addList(String word){
        this.list.add(word);
    }

    // Print
    public void print(){
        if (!this.list.isEmpty()){
            for (String word : list){
                System.out.println(word);
            }
        }
        else{
            System.out.println(x:"No Solution!");
        }
    }

    // Return size of list
    public int getSize(){
        return this.list.size();
    }

    // Cek word udah ada di list belom, kalo udah return true
    public boolean isFound(String word){
        for (String s : this.list){
            if (s.equals(word)){
                return true;
            }
        }
        return false;
    }
}
```

3.2. Kelas PQueue

Kelas PQueue mengimplementasikan *prioriry queue* berisi node dengan prioritas atribut cost pada kelas Node. Berikut atribut dari kelas PQueue.

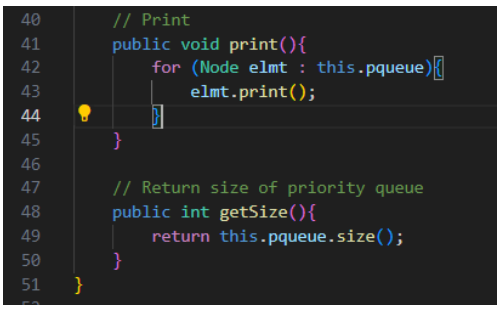
- Pqueue (list), merupakan list berisi kumpulan node dengan node yang memiliki cost terkecil berada pada indeks paling kecil.

Kelas PQueue memiliki beberapa method, antara lain sebagai berikut.

- add(Node n), memasukkan node n ke dalam list sesuai dengan urutan cost nodenya.
- getFirstElmt(), mengembalikan node dalam list dengan indeks ke-0 kemudian menghapus node tersebut dalam list. Jika list kosong maka method akan menge-*throw exception*.
- getFirstElmtWithoutRemove(), mengembalikan node dalam list dengan indeks ke-0 tanpa menghapus node tersebut dalam list. Jika list kosong maka method akan menge-*throw exception*.
- print(), memanggil method print() pada setiap node pada list.
- getSize(), mengembalikan ukuran list.

Berikut adalah *source code* dari kelas PQueue.

```
3   public class PQueue {
4       private List<Node> pqueue;
5
6       // Ctor
7       public PQueue(){
8           this.pqueue = new ArrayList<Node>();
9       }
10
11      // Nambahin node ke priority queue, yang costnya kecil taruh depan
12      public void add(Node n) {
13          pqueue.add(n);
14          Collections.sort(pqueue, Comparator.comparingInt(Node::getCost));
15      }
16
17      // Return first elemen which is node dengan cost paling kecil + remove node tsb
18      public Node getFirstElmt() throws NoSuchElementException {
19          if (this.pqueue.isEmpty()) {
20              throw new NoSuchElementException(s:"Oh no! Solution not found T_T");
21          }
22          else{
23              Node n = this.pqueue.get(index:0);
24              this.pqueue.remove(index:0);
25              return n;
26          }
27      }
28
29      // Return first elemen which is node dengan cost paling kecil tanpa diremove
30      public Node getFirstElmtWithoutRemove() throws NoSuchElementException {
31          if (this.pqueue.isEmpty()) {
32              throw new NoSuchElementException(s:"Oh no! Solution not found T_T");
33          }
34          else{
35              Node n = this.pqueue.get(index:0);
36              return n;
37          }
38      }
```



```
40      // Print
41      public void print(){
42          for (Node elmt : this.pqueue){
43              elmt.print();
44          }
45      }
46
47      // Return size of priority queue
48      public int getSize(){
49          return this.pqueue.size();
50      }
51  }
```

.

3.3. Kelas Method

Kelas Method mengimplementasikan method-method yang ada hubungannya dengan daftar kata pada kamus. Kelas Method memiliki atribut sebagai berikut.

- MapOfArray (map dengan key integer dan value list of string), berisi daftar kata dalam bahasa Inggris dimana key dari map merupakan panjang dari kata.
- wordExist (list of string), berisi kata-kata yang sudah pernah dikunjungi. Kata-kata ini disimpan agar tidak dikunjungi lagi yang dapat menyebabkan *infinite loop.*
- LIST_OF_CHAR berisi list huruf dari A sampai Z.

Kelas Method memiliki beberapa method, antara lain sebagai berikut.

- isUsed(String word), mengecek apakah suatu kata sudah pernah dikunjungi sebelumnya. Mengembalikan nilai true apabila kata tersebut sudah pernah dikunjungi,
- isExist(String word), mengecek apakah kata pada parameter ada pada kamus. Mengembalikan nilai true jika kata tersebut tersedia di dalam kamus (words.txt).
- calculateCost(String start, String target, int length), menghitung perbedaan huruf dari kata awal dan kata akhir kemudian mengembalikannya dengan tipe integer.
- isMatch(String start, String target, int length), mengecek apakah dua kata sama atau tidak. Mengembalikan nilai true apabila kedua kata sama.
- findChildUCS(Node ori, int length, String target, List<Node> list), mencari seluruh simpul yang dapat dibangkitkan dari simpul ori dengan algoritma UCS kemudian menyimpannya dalam list. Mengembalikan list berisi kumpulan jalur simpul.

- findChildGreedy(Node ori, int length, String target, List<Node> list), mencari seluruh simpul yang dapat dibangkitkan dari simpul ori dengan algoritma Greedy BFS kemudian menyimpannya dalam list. Mengembalikan list berisi kumpulan jalur simpul.
- findChildAStar(Node ori, int length, String target, List<Node> list), mencari seluruh simpul yang dapat dibangkitkan dari simpul ori dengan algoritma A* kemudian menyimpannya dalam list. Mengembalikan list berisi kumpulan jalur simpul.
- findMinimumNode(List<Node> list), mengembalikan simpul dengan cost terkecil di dalam list.

Berikut adalah *source code* dari kelas Method.

```java
public class Method {
    // Nyimpen word
    static Map<Integer, List<String>> mapOfArray = initialize();

    // Nyimpen word yang udah pernah dipake
    static List<String> wordExist = new ArrayList<String>();

    // List char dari A sampe Z
    private static final List<Character> LIST_OF_CHAR = new ArrayList<>();

    // Generate huruf dari a sampe z, simpen di LIST_OF_CHAR
    static {
        for (char c = 'A'; c <= 'Z'; c++) {
            LIST_OF_CHAR.add(c);
        }
    }

    // Cek wordnya udah pernah dipake belum, return true kalo udah pernah dipake
    private static boolean isUsed(String word){
        for (String s : wordExist){
            if (s.equals(word)){
                return true;
            }
        }
        return false;
    }
```

```java
// Read word dari txt, simpen di map (key: length, value: list of word)
private static final Map<Integer, List<String>> initialize(){
    Map<Integer, List<String>> mapOfArray = new HashMap<Integer, List<String>>();
    String filePath = "words.txt";
    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String word;
        while ((word = br.readLine()) != null) {
            int wordLength = word.length();
            if (!mapOfArray.containsKey(wordLength)){
                mapOfArray.put(wordLength, new ArrayList<String>());
            }
            mapOfArray.get(wordLength).add(word.toUpperCase());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return mapOfArray;
}

// Cek word ada di dictionary (words.txt) ato ga, return true kalo ada
public static boolean isExist(String word, int length) {
    List<String> words = mapOfArray.get(length);
    if (words == null) {
        return false;
    }
    for (String w : words) {
        if (w.equals(word)) {
            return true;
        }
    }
    return false;
}
```

```java
// Ngitung beda karakter word yang lagi dicek sama targetnya
public static int calculateCost(String start, String target, int length){
    int count = 0;
    for (int i = 0; i < length; i++){
        if (start.charAt(i) != target.charAt(i)){
            count++;
        }
    }
    return count;
}

// Cek wordnya udah sama kaya target belom, kalo udah return true
public static boolean isMatch(String start, String target, int length){
    return calculateCost(start, target, length) == 0;
}
```

```java
// Nyari semua word yang bisa dikunjungi dari node ori, simpen hasilnya di list, tiap step diitung costnya 1
public static int findChildUCS(Node ori, int length, String target, List<Node> list){
    int visited = 0;
    for (int i = 0; i < length; i++){
        for (char c : LIST_OF_CHAR){

            List <String> listOri = ori.getList();
            String lastWord = listOri.get(listOri.size()-1);

            // Iterasi perhuruf
            String wordTemp = lastWord.substring(beginIndex:0, i) + c + lastWord.substring(i + 1);
            char currChar = lastWord.charAt(i);

            if (Method.isExist(wordTemp, length) && currChar != c && !isUsed(wordTemp)){

                visited++;

                Node foundChild = new Node(ori);
                if (!foundChild.isFound(wordTemp)){
                    wordExist.add(wordTemp);
                    foundChild.addList( wordTemp);
                    foundChild.addGn();
                    list.add(foundChild);
                }
            }
        }
    }
    return visited;
}
```

```java
// Nyari semua word yang bisa dikunjungi dari node ori, simpen hasilnya di list, tiap step diitung costnya sesuai beda karakter sama targetnya
public static int findChildGreedy(Node ori, int length, String target, List<Node> list){
    int visited = 0;
    for (int i = 0; i < length; i++){
        for (char c : LIST_OF_CHAR){
            List <String> listOri = ori.getList();
            String lastWord = listOri.get(listOri.size()-1);
            String wordTemp = lastWord.substring(beginIndex:0, i) + c + lastWord.substring(i + 1);
            char currChar = lastWord.charAt(i);

            // Cek wordnya ada di dictionary ato engga
            if (Method.isExist(wordTemp, length) && currChar != c && !isUsed(wordTemp)){

                visited++;

                // foundChild = CCtor dari node ori
                Node foundChild = new Node(ori);
                if (!foundChild.isFound(wordTemp)){

                    // Itung cost
                    int cost = Method.calculateCost(wordTemp, target, wordTemp.length());

                    // wordExist: list word yang udah pernah dikunjungi
                    wordExist.add(wordTemp);

                    // Add word baru ke node
                    foundChild.addList( wordTemp);
                    foundChild.addHn(cost);

                    // Add node ke list
                    list.add(foundChild);
                }
            }
        }
    }
    return visited;
}
```

```java
// Nyari semua word yang bisa dikunjungi dari node ori, simpen hasilnya di list, tiap step diitung costnya sesuai beda karakter sama targetnya + 1
public static int findChildAStar(Node ori, int length, String target, List<Node> list){
    int visited = 0;
    for (int i = 0; i < length; i++){
        for (char c : LIST_OF_CHAR){
            List <String> listOri = ori.getList();
            String lastWord = listOri.get(listOri.size()-1);
            String wordTemp = lastWord.substring(beginIndex:0, i) + c + lastWord.substring(i + 1);
            char currChar = lastWord.charAt(i);

            // Cek wordnya ada di dictionary ato engga
            if (Method.isExist(wordTemp, length) && currChar != c && !isUsed(wordTemp)){

                visited++;

                // foundChild = CCtor dari node ori
                Node foundChild = new Node(ori);
                if (!foundChild.isFound(wordTemp)){
                    int cost = Method.calculateCost(wordTemp, target, wordTemp.length());

                    // wordExist: list word yang udah pernah dikunjungi
                    wordExist.add(wordTemp);

                    // Add word baru ke node
                    foundChild.addList( wordTemp);
                    foundChild.addHn(cost);
                    foundChild.addGn();

                    // Add node ke list
                    list.add(foundChild);
                }
            }
        }
    }
    return visited;
}
```

```
186     // Cari word paling deket sama target di list, dipake buat greedy
187     public static Node findMinimumNode(List<Node> list){
188         int minCost = list.get(index:0).getCost();
189         Node choosenNode = list.get(index:0);
190         for (Node n : list){
191             if (n.getCost() < minCost){
192                 minCost = n.getCost();
193                 choosenNode = n;
194             }
195         }
196         return choosenNode;
197     }
198 }
```

3.4. Kelas Solver

Kelas Solver mengimplementasikan algoritma solusi untuk tiap pilihan metode (UCS, Greedy BFS, dan A*). Berikut atribut dari kelas PQueue.

- Length (int) merupakan panjang dari kata awal dan kata target.
- Start (string) merupakan kata awal yang akan menjadi simpul akar.
- Target (string) merupakan kata tujuan yang dicari.
- Pqueue (PQueue) merupakan *priority queue* dengan spesifikasi yang telah dijelaskan di atas pada bagian Kelas PQueue.
- timeExecution (double) untuk menyimpan lama waktu eksekusi.
- memoryUsed (double) untuk menyimpan memory yang dibutuhkan selama eksekusi.
- visitedWords (int) untuk menyimpan banyak simpul yang dikunjungi.

Kelas PQueue memiliki beberapa method, antara lain sebagai berikut.

- solveUCS(), berisi algoritma untuk mendapatkan solusi dengan metode UCS.
- solveGreedy(), berisi algoritma untuk mendapatkan solusi dengan metode Greedy BFS.
- solveAStar(), berisi algoritma untuk mendapatkan solusi dengan metode A*.
- getTimeExecution(), mengembalikan lama waktu eksekusi dengan mengurangkan waktu setelah eksekusi dengan waktu sebelum eksekusi secara *real time*.
- getMemoryUsed(), mengembalikan besar memori yang digunakan selama eksekusi dengan menghitung memori setelah eksekusi dikurangkan dengan memori sebelum eksekusi.

- getVisitedWords(), mengembalikan banyak simpul yang dikunjungi.

Berikut adalah *source code* dari kelas Solver.

```
3   public class Solver {
4       private int length;
5       private String start;
6       private String target;
7       private PQueue pqueue;
8       private double timeExecution; // dalam ms
9       private double memoryUsed; // dalam mb
10      private int visitedWords;
11
12      // Ctor
13      public Solver(int length, String start, String target){
14          this.length = length;
15          this.start = start;
16          this.target = target;
17          this.pqueue = new PQueue();
18          this.timeExecution = 0;
19          this.memoryUsed = 0;
20          this.visitedWords = 0;
21      }
```

```
187     // Return time execution dalam ms
188     public double getTimeExecution(){
189         return this.timeExecution;
190     }
191
192     // Return memory used dalam mb
193     public double getMemoryUsed(){
194         return this.memoryUsed;
195     }
196
197     public int getVisitedWords(){
198         return this.visitedWords;
199     }
200
201 }
```

```java
    // Solve dengan metode UCS, return node
    public Node solveUCS(){

        // Time & memory
        long startTime = System.currentTimeMillis();
        Runtime runtime = Runtime.getRuntime();
        long memoryBefore = runtime.totalMemory() - runtime.freeMemory();

        Node first = new Node(this.start, this.target);
        List<Node> listTemp = new ArrayList<Node>();

        // Cari semua child
        int visited;
        visited = Method.findChildUCS(first, this.length, this.target, listTemp);
        this.visitedWords += visited;

        // Masukin ke priority queue
        for (Node solution : listTemp){
            pqueue.add(solution);
        }

        Node firstElmtPQueue = pqueue.getFirstElmtWithoutRemove();
        String word = firstElmtPQueue.getList().get(firstElmtPQueue.getList().size() - 1);


        while (!Method.isMatch(word, this.target, this.length)){

            List<Node> listNode = new ArrayList<Node>();

            // Ambil first + remove
            firstElmtPQueue = pqueue.getFirstElmt();

            // Find child dari elemen pertama
            Node ntemp = new Node(firstElmtPQueue);
            visited = Method.findChildUCS(ntemp, this.length, this.target, listNode);
            this.visitedWords += visited;

            // Masukin hasilnya
            for (Node solution : listNode){
                pqueue.add(solution);
            }

            firstElmtPQueue = pqueue.getFirstElmtWithoutRemove();

            word = firstElmtPQueue.getList().get(firstElmtPQueue.getList().size() - 1);
        }

        // Time & memory
        long endTime = System.currentTimeMillis();
        this.timeExecution = endTime - startTime;
        long memoryAfter = runtime.totalMemory() - runtime.freeMemory();
        long memoryUsage = memoryAfter - memoryBefore;
        this.memoryUsed = memoryUsage / (1024.0 * 1024.0);

        // Return
        return pqueue.getFirstElmtWithoutRemove();
    }
```

```java
    // Solve dengan metode Greedy, return node
    public Node solveGreedy(){

        // Time & memory
        long startTime = System.currentTimeMillis();
        Runtime runtime = Runtime.getRuntime();
        long memoryBefore = runtime.totalMemory() - runtime.freeMemory();

        Node first = new Node(this.start, this.target);

        // Tambahin cost di awal
        int initialCost = Method.calculateCost(start, target, length);
        first.addHn(initialCost);

        List<Node> listTemp = new ArrayList<Node>();

        // Cari semua child
        int visited;
        visited = Method.findChildGreedy(first, this.length, this.target, listTemp);
        this.visitedWords += visited;

        // Cari node dengan cost minimal
        Node choosen = Method.findMinimumNode(listTemp);

        String word = choosen.getList().get(choosen.getList().size() - 1);

        while (!Method.isMatch(word, this.target, this.length)){

            List<Node> listNode = new ArrayList<Node>();
            visited = Method.findChildGreedy(choosen, this.length, this.target, listNode);
            this.visitedWords++;

            choosen = Method.findMinimumNode(listNode);

            word = choosen.getList().get(choosen.getList().size() - 1);
        }

        // Time & memory
        long endTime = System.currentTimeMillis();
        this.timeExecution = endTime - startTime;
        long memoryAfter = runtime.totalMemory() - runtime.freeMemory();
        long memoryUsage = memoryAfter - memoryBefore;
        this.memoryUsed = memoryUsage / (1024.0 * 1024.0);

        // Return
        return choosen;
    }

```

```java
    // Solve dengan metode A*, return node
    public Node solveAStar(){

        // Time
        long startTime = System.currentTimeMillis();
        Runtime runtime = Runtime.getRuntime();
        long memoryBefore = runtime.totalMemory() - runtime.freeMemory();

        Node first = new Node(this.start, this.target);
        List<Node> listTemp = new ArrayList<Node>();

        // Cari semua child
        int visited;
        visited = Method.findChildAStar(first, this.length, this.target, listTemp);
        this.visitedWords += visited;

        // Masukin ke priority queue
        for (Node solution : listTemp){
            pqueue.add(solution);
        }

        Node firstElmtPQueue = pqueue.getFirstElmtWithoutRemove();
        String word = firstElmtPQueue.getList().get(firstElmtPQueue.getList().size() - 1);


        while (!Method.isMatch(word, this.target, this.length)){

            List<Node> listNode = new ArrayList<Node>();

            // Ambil first + remove
            firstElmtPQueue = pqueue.getFirstElmt();

            // Find child dari elemen pertama
            Node ntemp = new Node(firstElmtPQueue);
            visited = Method.findChildAStar(ntemp, this.length, this.target, listNode);
            this.visitedWords += visited;

            // Masukin hasilnya
            for (Node solution : listNode){
                pqueue.add(solution);
            }

            firstElmtPQueue = pqueue.getFirstElmtWithoutRemove();

            word = firstElmtPQueue.getList().get(firstElmtPQueue.getList().size() - 1);
        }

        // Time & memory
        long endTime = System.currentTimeMillis();
        this.timeExecution = endTime - startTime;
        long memoryAfter = runtime.totalMemory() - runtime.freeMemory();
        long memoryUsage = memoryAfter - memoryBefore;
        this.memoryUsed = memoryUsage / (1024.0 * 1024.0);

        //Return
        return pqueue.getFirstElmtWithoutRemove();
    }
```

3.5. Kelas Main

Kelas Main bertanggung jawab sebagai kelas utama untuk mengeksekusi program dengan *command line interface* (CLI), termasuk menjembatani antara kelas-kelas berisi algoritma dengan input dari pengguna dan output yang ditampilkan ke pengguna. Dalam kelas ini, input dari pengguna divalidasi (panjang kata awal dan akhir harus sama, kata awal dan akhir harus ada dalam kamus yang sudah tersedia). Program akan meminta input ulang dari pengguna sampai semua input valid. Setelah di-*run* program akan *terminated*. Kelas Main sendiri tidak memiliki atribut dan method.

Berikut implementasi dari kelas Main.

```
5   public class Main {
        Run | Debug
6       public static void main(String[] args) {
7
8           Scanner scanner = new Scanner(System.in);
9           System.out.println(x:"Welcome to Word Ladder Solver!");
10          System.out.println(x:"Make sure you enter valid words in English!");
11
12          String start;
13          String goal;
14
15          do {
16              System.out.print(s:"Enter start word: ");
17              start = scanner.next().toUpperCase();
18              System.out.print(s:"Enter goal word: ");
19              goal = scanner.next().toUpperCase();
20
21              if (start.length() != goal.length()) {
22                  System.out.println(x:"Start and goal words must have the same length.");
23              } else if (!Method.isExist(start, start.length()) || !Method.isExist(goal,
24                      goal.length())) {
25                  System.out.println(x:"One or both words do not exist in the dictionary.");
26              }
27          } while (start.length() != goal.length() ||
28                  !Method.isExist(start, start.length()) || !Method.isExist(goal,
29                          goal.length()));
30
31          int method = 0;
32          boolean validMethod = false;
33
```

```
        while (!validMethod) {
            try {
                System.out.println();
                System.out.println(x:"Method: ");
                System.out.println(x:"1. UCS");
                System.out.println(x:"2. Greedy");
                System.out.println(x:"3. A*");
                System.out.println(x:"Just enter the number!");
                System.out.print(s:"Enter method: ");
                method = scanner.nextInt();

                if (method >= 1 && method <= 3) {
                    validMethod = true;
                } else {
                    System.out.println(x:"Invalid input! Please enter a number between 1 and 3.");
                }
            } catch (java.util.InputMismatchException e) {
                System.out.println(x:"Invalid input! Please enter a number.");
                scanner.next();
            }
        }
        scanner.close();

        // Solve

        if (start.equals(goal)){
            System.out.println();
            System.out.println(x:"You have found the answer!");
            System.out.println();
            return;
        }
```

```
        Solver solver = new Solver(start.length(), start, goal);
        Node nodeRes = new Node(start, goal);

        try{
            if (method == 1) { // UCS
                nodeRes = solver.solveUCS();
            }

            else if (method == 2) { // Greedy
                nodeRes = solver.solveGreedy();
            }

            else { // A Star
                nodeRes = solver.solveAStar();
            }

            System.out.println();
            System.out.println(x:"Result: ");
            nodeRes.print();

            System.out.println("Execution time: " + String.format(format:"%.2f", solver.getTimeExecution()) + " ms");
            System.out.println("Memory used: " + String.format(format:"%.2f", solver.getMemoryUsed()) + " mb");
            System.out.println("Node visited: " + solver.getVisitedWords() + " nodes");
        } catch(NoSuchElementException e){
            System.out.println();
            System.out.println(e.getMessage());
        }

        System.out.println();
        System.out.println(x:"Have a nice day!");
        System.out.println();

    }
}
```

3.6. Kelas Gui

Kelas Gui bertanggung jawab terhadap graphical user interface (GUI) untuk menerima input dari pengguna dan menampilkan output kepada penguna. Kelas ini tidak memiliki atribut, namun memiliki beberapa method antara lain sebagai berikut.

- Gui(), untuk meng-*setup* tampilan GUI, termasuk textfield untuk kata awal dan kata tujuan, dropdown untuk pilihan algoritma, tombol untuk memulai eksekusi program, serta scrollable area untuk menampilkan output.
- Main, untuk mengeksekusi program.

Sama seperti kelas Main, kelas ini juga menjembatani kelas-kelas algoritma dengan input dan output pengguna. Kelas ini juga memvalidasi input dari pengguna. Berbeda dengan kelas Main, kelas ini memungkinkan pengguna untuk mencari solusi berkali kali dalam sekali *run*. Ketika program sedang berjalan, tombol untuk mengeksekusi program akan di-*disabled* untuk menghindari adanya *bug*.

Berikut implementasi dari kelas Gui.

```
161     public static void main(String[] args) {
162         SwingUtilities.invokeLater(new Runnable() {
163             public void run() {
164                 new Gui();
165             }
166         });
167     }
168 }
```

```java
public class Gui extends JFrame {
    private JTextField startTextField;
    private JTextField goalTextField;
    private JComboBox<String> algorithmChoosen;
    private JTextArea outputTextArea;
    private JButton startButton;

    public Gui() {

        // Setup Frame
        setTitle(title:"Word Ladder");
        setSize(width:400, height:450);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(rows:6, cols:1));
        getContentPane().setBackground(Color.BLACK);
        setLocationRelativeTo(c:null);

        // Title, subtitle
        JLabel titleLabel = new JLabel(text:"Welcome to Word Ladder");
        JLabel subtitleLabel = new JLabel(text:"by Evelyn Yosiana 13522083");
        titleLabel.setFont(new Font(name:"Arial", Font.BOLD, size:20));
        titleLabel.setForeground(Color.WHITE);
        subtitleLabel.setFont(new Font(name:"Arial", Font.BOLD, size:12));
        subtitleLabel.setForeground(Color.WHITE);
        JPanel titlePanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        titlePanel.add(titleLabel);
        titlePanel.add(subtitleLabel);
        titlePanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
        titlePanel.setBackground(Color.BLACK);
        add(titlePanel);
```

```java
        // Label start
        JLabel startLabel = new JLabel(text:"Start:");
        startLabel.setFont(new Font(name:"Arial", Font.BOLD, size:14));
        startLabel.setForeground(Color.WHITE);
        startTextField = new JTextField();
        startTextField.setPreferredSize(new Dimension(width:300, height:30));
        startTextField.setBackground(Color.BLACK);
        startTextField.setForeground(Color.WHITE);
        JPanel startPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        startPanel.add(startLabel);
        startPanel.add(startTextField);
        startPanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
        startPanel.setBackground(Color.BLACK);
        add(startPanel);

        // Label goal
        JLabel goalLabel = new JLabel(text:"Goal:");
        goalLabel.setFont(new Font(name:"Arial", Font.BOLD, size:14));
        goalLabel.setForeground(Color.WHITE);
        goalTextField = new JTextField();
        goalTextField.setPreferredSize(new Dimension(width:300, height:30));
        goalTextField.setBackground(Color.BLACK);
        goalTextField.setForeground(Color.WHITE);
        JPanel goalPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        goalPanel.add(goalLabel);
        goalPanel.add(goalTextField);
        goalPanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
        goalPanel.setBackground(Color.BLACK);
        add(goalPanel);
```

```
        // Dropdown algorithm choice
        JLabel algorithmLabel = new JLabel(text:"Select Algorithm:");
        algorithmLabel.setFont(new Font(name:"Arial", Font.BOLD, size:14));
        algorithmLabel.setForeground(Color.WHITE);
        String[] algorithms = { "UCS", "Greedy", "AStar" };
        algorithmChoosen = new JComboBox<>(algorithms);
        algorithmChoosen.setFont(new Font(name:"Arial", Font.PLAIN, size:12));
        algorithmChoosen.setBackground(Color.BLACK);
        algorithmChoosen.setForeground(Color.WHITE);
        JPanel algorithmPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        algorithmPanel.add(algorithmLabel);
        algorithmPanel.add(algorithmChoosen);
        algorithmPanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
        algorithmPanel.setBackground(Color.BLACK);
        add(algorithmPanel);

        // Result frame
        outputTextArea = new JTextArea(rows:10, columns:30);
        outputTextArea.setEditable(b:false);
        outputTextArea.setBackground(Color.BLACK);
        outputTextArea.setForeground(Color.WHITE);
        JScrollPane scrollPane = new JScrollPane(outputTextArea);

        Border lineBorder = BorderFactory.createLineBorder(Color.WHITE);
        Border matteBorder = BorderFactory.createMatteBorder(top:0, left:20, bottom:5, right:20, Color.BLACK);
        Border compoundBorder = BorderFactory.createCompoundBorder(matteBorder, lineBorder);
        scrollPane.setBorder(compoundBorder);
        scrollPane.setBackground(Color.BLACK);
        add(scrollPane);
```

```
        // Button
        startButton = new JButton(text:"Find");
        startButton.setFont(new Font(name:"Arial", Font.BOLD, size:14));
        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                startButton.setEnabled(b:false);

                Method.wordExist.clear();
                outputTextArea.setText(t:"");

                String start = startTextField.getText();
                String goal = goalTextField.getText();
                String method = (String) algorithmChoosen.getSelectedItem();

                if (start.equals(goal)) {
                    outputTextArea.append("You have found the answer!" + "\n");
                    return;
                }

                Solver solver = new Solver(start.length(), start, goal);
                Node nodeRes = new Node(start, goal);

                try {
                    if (method.equals(anObject:"UCS")) {
                        nodeRes = solver.solveUCS();
                    } else if (method.equals(anObject:"Greedy")) {
                        nodeRes = solver.solveGreedy();
                    } else {
                        nodeRes = solver.solveAStar();
                    }

                    outputTextArea.append("Result: " + "\n");
                    for (String s : nodeRes.getList()) {
                        outputTextArea.append(s);
                        outputTextArea.append(str:"\n");
                    }

                    outputTextArea.append("Execution time: " + String.format(format:"%.2f", solver.getTimeExecution()) + " ms" + "\n");
                    outputTextArea.append("Memory used: " + String.format(format:"%.2f", solver.getMemoryUsed()) + " mb" + "\n");
                    outputTextArea.append("Node visited: " + solver.getVisitedWords() + " nodes" + "\n");

                } catch (NoSuchElementException ex) {
                    outputTextArea.append(ex.getMessage() + "\n");
                }

                startButton.setEnabled(b:true);
                startTextField.setText(t:"");
                goalTextField.setText(t:"");
                startTextField.setText(start);
                goalTextField.setText(goal);
                algorithmChoosen.setSelectedItem(method);
            }
        });
        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        buttonPanel.add(startButton);
        buttonPanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
        buttonPanel.setBackground(Color.BLACK);
        add(buttonPanel);

        setResizable(resizable:false);
        setVisible(b:true);
    }
```

BAB 4

Testing

UCS	Greedy BFS	A*
Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: ABY Enter goal word: MAR Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 1 Result: ABY ABS AAS MAS MAR Execution time: 164.00 ms Memory used: 4.24 mb Node visited: 720 nodes Have a nice day!	Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: ABY Enter goal word: MAR Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 2 Result: ABY ANY ANA ABA AGA AHA SHA WHA WHO MHO MOO MOR MAR Execution time: 32.00 ms Memory used: 1.32 mb Node visited: 13 nodes Have a nice day!	Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: ABY Enter goal word: MAR Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 3 Result: ABY ABS AAS MAS MAR Execution time: 107.00 ms Memory used: 1.32 mb Node visited: 250 nodes Have a nice day!
Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: BOTH Enter goal word: HOLE Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 1 Result: BOTH DOTH DOTE DOLE HOLE Execution time: 1036.00 ms Memory used: -6.91 mb Node visited: 1754 nodes Have a nice day!	Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: BOTH Enter goal word: HOLE Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 2 Result: BOTH DOTH DOTE DOLE HOLE Execution time: 27.00 ms Memory used: 0.92 mb Node visited: 5 nodes Have a nice day!	Welcome to Word Ladder Solver! Make sure you enter valid words in English! Enter start word: BOTH Enter goal word: HOLE Method: 1. UCS 2. Greedy 3. A* Just enter the number! Enter method: 3 Result: BOTH DOTH DOTE DOLE HOLE Execution time: 321.00 ms Memory used: 3.32 mb Node visited: 697 nodes Have a nice day!

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: APPLE
Enter goal word: MANGO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 1

Result:
APPLE
AMPLE
AMOLE
ANOLE
ANILE
ANISE
ARISE
PRISE
PAISE
PARSE
MARSE
MANSE
MANGE
MANGO
Execution time: 8274.00 ms
Memory used: 12.30 mb
Node visited: 3020 nodes

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: APPLE
Enter goal word: MANGO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 2

Oh no! Solution not found T_T

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: APPLE
Enter goal word: MANGO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 3

Result:
APPLE
AMPLE
AMOLE
ANOLE
ANILE
ANISE
ARISE
PRISE
PAISE
PARSE
MARSE
MANSE
MANGE
MANGO
Execution time: 3483.00 ms
Memory used: 7495.79 mb
Node visited: 770 nodes

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CROWN
Enter goal word: JEWEL

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 1

Result:
CROWN
CROWS
CHOWS
SHOWS
SHOES
SHOED
SHRED
SERED
SEWED
JEWED
JEWEL
Execution time: 15070.00 ms
Memory used: 67.48 mb
Node visited: 6864 nodes

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CROWN
Enter goal word: JEWEL

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 2

Oh no! Solution not found T_T

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CROWN
Enter goal word: JEWEL

Method:
1. UCS
2. Greedy
3. A*

Result:
CROWN
CROWS
CHOWS
SHOWS
SHOES
SHOED
SHRED
SERED
SEWED
JEWED
JEWEL
Execution time: 7303.00 ms
Memory used: 13470.26 mb
Node visited: 4793 nodes

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: NUMBER
Enter goal word: BANANA

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 1

Oh no! Solution not found T_T

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: NUMBER
Enter goal word: BANANA

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 2

Oh no! Solution not found T_T

Have a nice day!
```

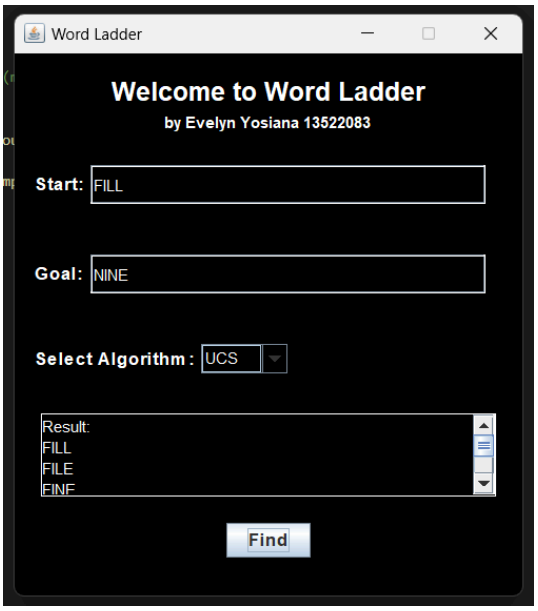
```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: NUMBER
Enter goal word: BANANA

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 3

Oh no! Solution not found T_T

Have a nice day!
```

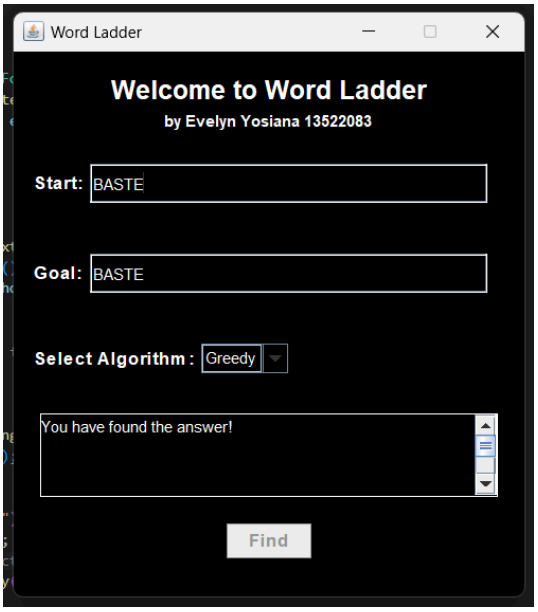
```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CHARGE
Enter goal word: COMEDO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 1

Result:
CHARGE
CHANGE
CHANGS
CHANTS
CHINTS
CHINES
CHINED
COINED
CONNED
CONNER
CONGER
CONGES
CONIES
CONINS
CONING
HONING
HOMING
HOMINY
HOMILY
HOMELY
COMELY
COMEDY
COMEDO
Execution time: 39807.00 ms
Memory used: 131.38 mb
Node visited: 8360 nodes

Have a nice day!
```

```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CHARGE
Enter goal word: COMEDO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 2

Oh no! Solution not found T_T

Have a nice day!
```

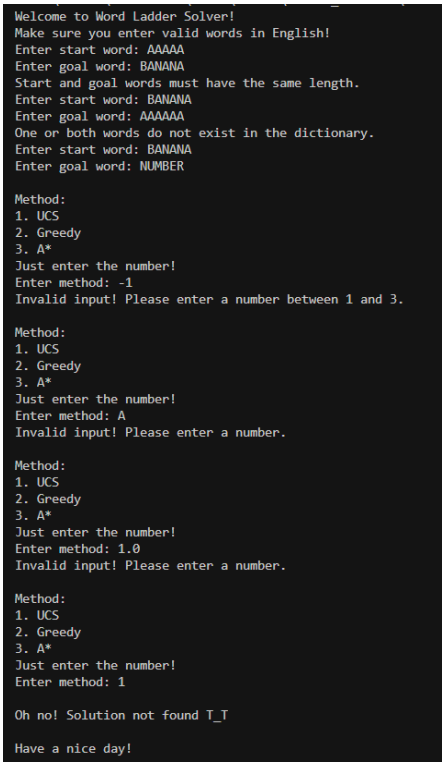
```
Welcome to Word Ladder Solver!
Make sure you enter valid words in English!
Enter start word: CHARGE
Enter goal word: COMEDO

Method:
1. UCS
2. Greedy
3. A*
Just enter the number!
Enter method: 3

Result:
CHARGE
CHANGE
CHANGS
CHANTS
CHINTS
CHINES
CHINED
COINED
COILED
COOLED
COOEED
COOEES
COOERS
COVERS
COVENS
COVINS
COVING
COMING
HOMING
HOMINY
HOMILY
HOMELY
COMELY
COMEDY
COMEDO
Execution time: 4105.00 ms
Memory used: 9543.81 mb
Node visited: 926 nodes

Have a nice day!
```

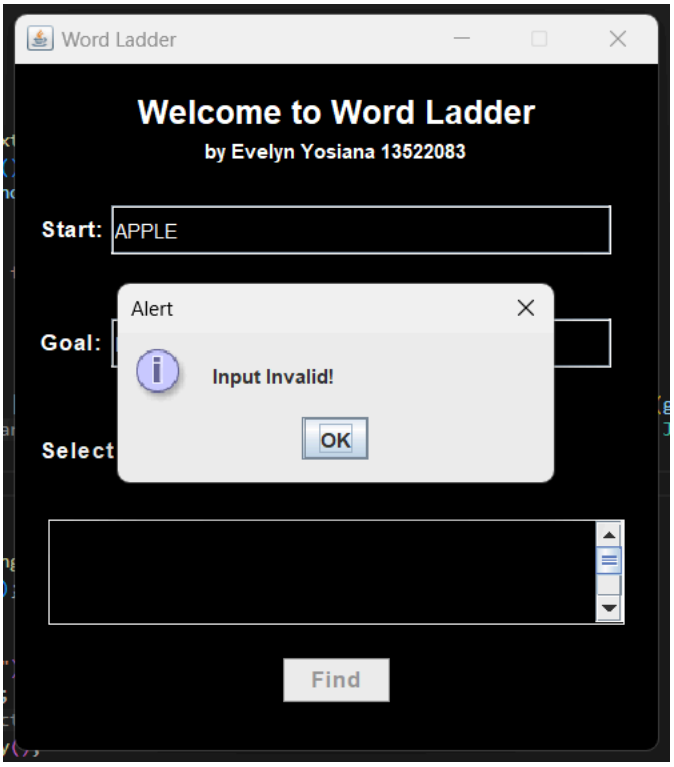
Tampilan GUI:





Pengujian validasi:



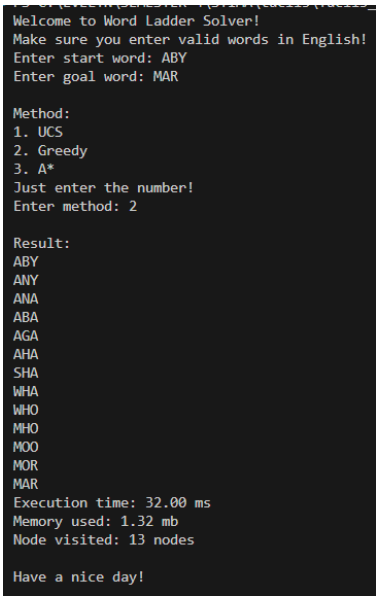


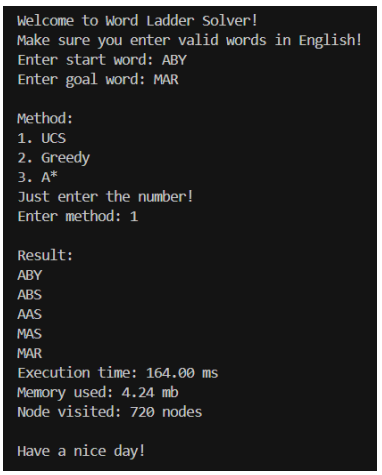
BAB 5

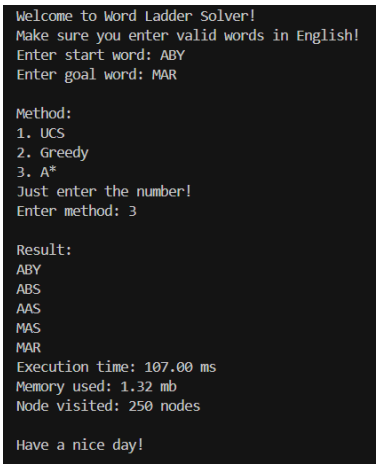
Pembahasan

Berdasarkan hasil pengujian, urutan kompleksitas waktu GBFS > A* > UCS. Sedangkan urutan kompleksitas ruang GBFS > UCS > A*. Urutan optimalitasnya sebagai berikut A* > UCS > GBFS.

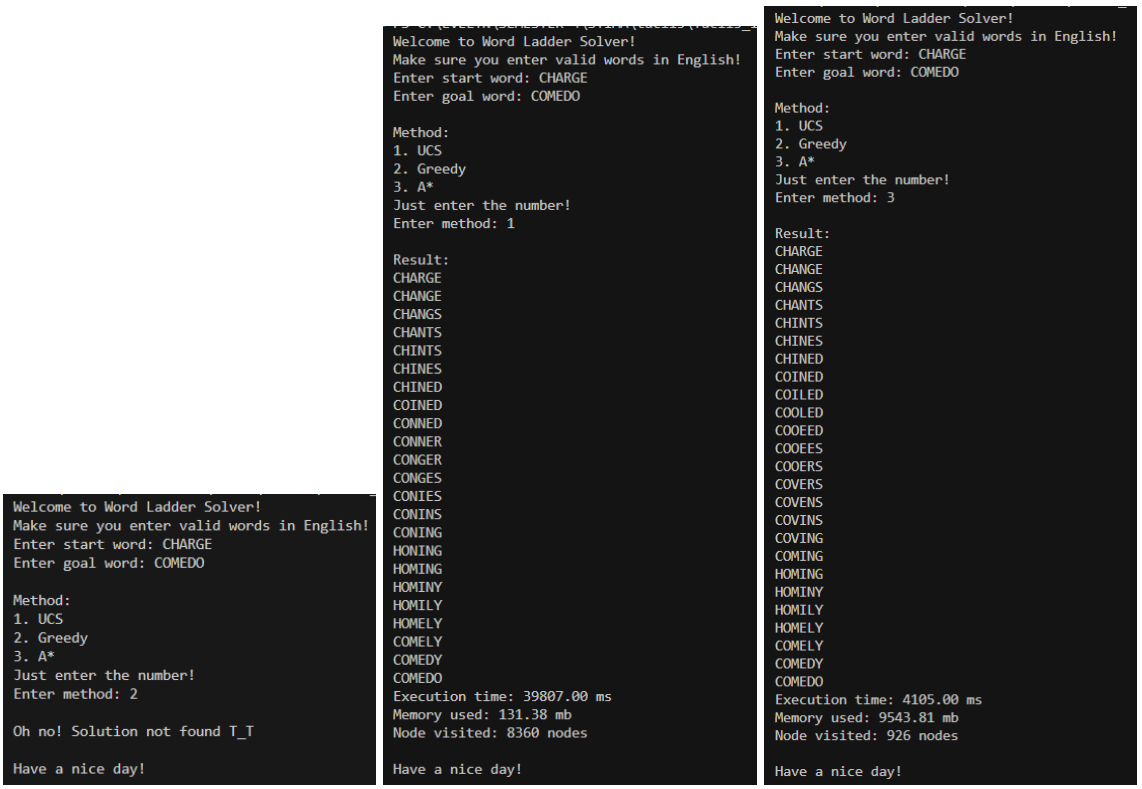
GBFS menjadi algoritma yang paling cepat dan efisien kompleksitas ruang karena simpul yang di-expand hanya simpul dengan heuristik terkecil. Algoritma ini tidak mengekspan simpul yang lain. Namun karena hal itu juga solusi yang dihasilkan tidak selalu optimal. Misalnya pada kasus kata “ABY” ke “MAR”. Algoritma ini menghasilkan solusi dengan 13 simpul di saat algoritma lainnya menghasilkan hanya 5 simpul. Namun waktu eksekusi dan kompleksitas ruangnya tetap menjadi yang paling minimum.







Bahkan terkadang algoritma ini tidak menghasilkan solusi di saat algoritma lain menghasilkan solusi. Misalnya pada kasus “CHARGE” ke “KOMEDO”. Hal ini dikarenakan pada pengerjaan kali ini, algoritma GBFS dibuat tidak backtrack ketika telah sampai ke simpul terakhir yang sudah tidak dapat di-expand dan belum menemukan solusi.



Algoritma A* lebih cepat dibandingkan dengan algoritma UCS karena dengan menggunakan algoritma ini, simpul-simpul yang dibangkitkan menuju pada hasil optimal dengan menggunakan kombinasi dari g(n) dan h(n). Artinya algoritma ini memperhitungkan biaya heuristiknya juga. Pada beberapa kasus, kompleksitas ruang dari algoritma ini sangat besar. Hal ini bergantung pada method perhitungan memori yang telah disediakan oleh java.

Algoritma UCS sering kali menjadi algoritma dengan eksekusi waktu yang lama dikarenakan semua simpul yang berpotensi di-expand akan di-expand tanpa memperdulikan urutan heuristiknya. Pada kasus ini, hasil dari algoritma ini sering kali sama dengan hasil dengan algoritma A*.

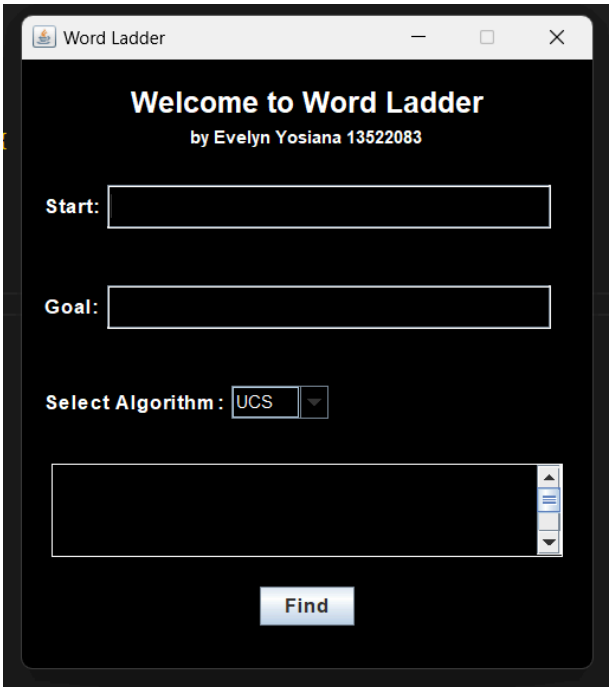
Pada beberapa kasus, memori yang digunakan selama perhitungan bernilai negatif. Hal ini dapat terjadi karena adanya garbage collector. Memori yang digunakan dihitung dari memori setelah program selesai dieksekusi dikurangkan dengan memori sebelum program dieksekusi. Di antara waktu tersebut, mungkin saja memori dibersihkan oleh garbage collector dimana ukuran memori yang dibersihkan ini lebih besar daripada memori asli yang digunakan selama proses pencarian. Hal inilah yang menyebabkan perhitungan memori menjadi negatif.

BAB 6

Penjelasan Bonus

Pada pengerjaan tugas kecil kali ini, bonus yang dikerjakan adalah GUI. *Source code* dari program GUI sudah dilampirkan pada bab 3. GUI dibuat dengan menggunakan Swing Java. GUI dapat menerima input dari pengguna. Input yang dimasukkan oleh pengguna akan divalidasi terlebih dahulu sebelum diproses. Validasi yang dilakukan yaitu validasi kata yang dimasukkan ada pada kamus atau tidak, serta panjang dari kedua kata harus sama. Saat pemrosesan berlangsung, tombol "Find!" akan di-*disabled* untuk menghindari adanya bug karena proses bertabrakan. Setelah proses selesai, hasilnya akan ditampilkan di daerah yang dapat di-*scroll*. Setelah itu tombol "Find!" dapat diklik kembali dan pengguna dapat memproses kata kembali.

Berikut tampilan dari GUI yang telah dibuat.



Cara menjalankan GUI yaitu dengan masuk ke folder bin kemudian masukkan "java -jar gui.jar" pada CLI. Namun terkadang file jar ini sedikit bermasalah. Jika terjadi masalah pada file jar, pengguna dapat menjalankannya dari file src dengan memasukkan "javac Gui.java" kemudian "java Gui".

BAB 7

Kesimpulan

Program yang dibuat berjalan dengan baik. Program dapat menerima dan memvalidasi input dari CLI maupun GUI. Berdasarkan hasil pengamatan saat pengujian, didapatkan bahwa:

Urutan kompleksitas waktu: GBFS > A* > UCS.
Urutan kompleksitas ruang: GBFS > UCS > A*
Urutan optimalitas solusi: A* > UCS > GBFS.

Saran untuk pengerjaan tugas berikutnya berdasarkan hasil pada tugas ini yaitu sebisa mungkin, rancang kelas-kelas yang diperlukan dengan mata sebelum proses *coding*. Hal ini bertujuan agar tidak ada kejadian tanggung jawab kelas tiba-tiba berubah sedangkan nama kelas tidak berubah sehingga nama kelas menjadi kurang intuitif. Contohnya pada pengerjaan ini, kelas Node yang secara intuitif merepresentasikan sebuah simpul malah merepresentasikan kumpulan simpul.

LAMPIRAN

Link github repository: https://github.com/evelynnn04/Tucil3_13522083

Keberhasilan program:

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	✓	

DAFTAR PUSTAKA

GeeksforGeeks. (2022, February 15). Introduction to Java Swing. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-java-swing/

Java T Point. (n.d.). Java Swing Tutorial - javatpoint. Www.javatpoint.com. https://www.javatpoint.com/java-swing

Munir, Rinaldi. 2024. Penentuan Rute (Route/Path Planning) Bagian 1. Bandung: Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

Munir, Rinaldi. 2024. Penentuan Rute (Route/Path Planning) Bagian 2. Bandung: Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf