LAPORAN TUGAS KECIL 3
IF2211 Strategi Algoritma Semester II tahun 2023/2024
Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS, Greedy Best First Search, dan A*

Disusun Oleh:

Ariel Herfrison 13522002

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

Daftar Isi

1. Analisis dan Implementasi Algoritma

1.1 Analisis dan Implementasi Route Planning

Route Planning adalah salah satu permasalahan programming di mana algoritma harus dapat mencari rute dari suatu *lokasi awal* menuju *lokasi akhir*. Penentuan rute biasanya dilakukan berdasarkan *cost* yang dibutuhkan untuk mencapai suatu *lokasi n* dari *lokasi awal*. Terdapat dua jenis *cost* yang diperhatikan dalam route planning, yaitu *cost asli*, yang biasanya didefinisikan dengan nilai ***g(n)***, dan *cost heuristic*, yang biasanya didefinisikan dengan nilai ***h(n)***. *Cost asli* adalah "jarak" dari *lokasi awal* ke *lokasi n*, sedangkan *heuristic cost* adalah estimasi "jarak" dari *lokasi n* ke *lokasi akhir*. *Cost asli (**g(n)**)* dan *heuristic cost (**h(n)**)* dapat digabungkan atau digabungkan secara tersendiri untuk menentukan *estimasi total cost* yang biasanya didefiniskan dengan nilai ***f(n)***.

Pada penyelesaian permainan Word Ladder, *cost asli (**g(n)**)* ditentukan berdasarkan jumlah perubahan karakter yang dibutuhkan untuk merubah kata awal (*lokasi awal*) menjadi kata sekarang (*lokasi n*). Sedangkan *cost heuristic (**h(n)**)*, ditentukan berdasarkan jumlah karakter yang berbeda antara kata sekarang dengan kata akhir (*lokasi akhir*). Tetangga dari suatu kata adalah kata lain yang memiliki panjang yang sama dan hanya berbeda satu karakter dengan kata tersebut.

1.1 Analisis dan Implementasi Algoritma UCS

Uniform Cost Search adalah algoritma route planning di mana *estimasi total cost* dari setiap node sama dengan jarak dari *root node* menuju *node sekarang/node n*, tanpa menggunakan *heuristic cost*. Langkah-langkah algoritma UCS secara umum adalah sebagai berikut:

1. Buat himpunan "Simpul Hidup" yang menampung semua simpul yang akan diperiksa beserta *estimasi total cost* mereka. Pada algoritma UCS, *estimasi total cost* ***f(n) = g(n)***.
2. Masukkan Simpul Awal ke Simpul Hidup. *Cost* Simpul Awal = 0.
3. Ambil simpul dalam Simpul Hidup dengan *cost* terendah.
4. Apabila simpul yang sedang diperiksa adalah Simpul Tujuan, hentikan pencarian dan keluarkan rute dari Simpul Awal ke Simpul Tujuan.
5. Cari simpul-simpul tetangga dari simpul yang sedang diperiksa.
6. Apabila simpul tetangga belum diperiksa, masukkan simpul tetangga ke dalam simpul hidup dengan ***cost baru* = g(n) = *cost lama* + 1**.
7. Ulangi langkah 3-6 sampai Simpul Akhir ditemukan.

Seperti yang telah dijelaskan pada subbab sebelumnya, pada penyelesaian permainan Word Ladder, *cost asli (**g(n)**)* adalah jumlah perubahan karakter yang dibutuhkan untuk merubah kata awal (*lokasi awal*) menjadi kata sekarang (*lokasi n*). Karena hanya boleh terdapat 1 perubahan karakter dalam permainan Word Ladder pada setiap tahapnya, *cost baru* pada setiap tahap hanya bertambah 1. Karena *steps = cost*, hal ini membuat UCS secara efektif sama dengan BFS. UCS

pasti akan mencari semua simpul tetangga terlebih dahulu karena mereka pasti memiliki cost yang terkecil.

1.2 Analisis dan Implementasi Algoritma Greedy Best First Search

Greedy Best First Search adalah algoritma route planning di mana *estimasi total cost* dari setiap node sama dengan *heuristic cost*, tanpa menggunakan *cost asli*. Langkah-langkah algoritma Greedy Best First Search secara umum adalah sebagai berikut:

1. Buat himpunan "Simpul Hidup" yang menampung semua simpul yang akan diperiksa beserta *estimasi total cost* mereka. Pada algoritma Greedy Best First Search, *estimasi total cost* ***f(n) = h(n)***.
2. Masukkan Simpul Awal ke Simpul Hidup. *Cost* Simpul Awal = 0.
3. Ambil simpul dalam Simpul Hidup dengan *cost* terendah.
4. Apabila simpul yang sedang diperiksa adalah Simpul Tujuan, hentikan pencarian dan keluarkan rute dari Simpul Awal ke Simpul Tujuan.
5. Cari simpul-simpul tetangga dari simpul yang sedang diperiksa.
6. Masukkan simpul tetangga ke dalam simpul hidup dengan ***cost baru* = h(n) = jumlah karakter yang berbeda antara kata sekarang dengan kata akhir**.
7. Ulangi langkah 3-6 sampai Simpul Akhir ditemukan.

Karena Greedy Best First Search tidak *complete*, Greedy Best First Search tidak menjamin solusi optimal. Algoritma mungkin memilih simpul yang "terlihat dekat" (memiliki perbedaan karakter yang sedikit), tetapi memerlukan rute yang jauh untuk mencapai simpul akhir. Karena *cost* simpul dalam Greedy Best First Search juga tidak akan berubah-ubah (jumlah karakter yang berbeda tidak akan berubah), algoritma Greedy Best First Search juga dapat terjebak dalam local minima.

1.3 Analisis dan Implementasi Algoritma A*

A* adalah algoritma route planning di mana *estimasi total cost* dari setiap node sama dengan jarak dari *root node* menuju *node sekarang/node n*, ditambah *heuristic cost*. Langkah-langkah algoritma A* secara umum adalah sebagai berikut:

1. Buat himpunan "Simpul Hidup" yang menampung semua simpul yang akan diperiksa beserta *estimasi total cost* mereka. Pada algoritma UCS, *estimasi total cost* ***f(n) = g(n)***.
2. Masukkan Simpul Awal ke Simpul Hidup. *Cost* Simpul Awal = 0.
3. Ambil simpul dalam Simpul Hidup dengan *cost* terendah.
4. Apabila simpul yang sedang diperiksa adalah Simpul Tujuan, hentikan pencarian dan keluarkan rute dari Simpul Awal ke Simpul Tujuan.
5. Cari simpul-simpul tetangga dari simpul yang sedang diperiksa.
6. Apabila simpul tetangga belum diperiksa, masukkan simpul tetangga ke dalam simpul hidup dengan ***cost baru* = g(n) + h(n) = *cost lama* + 1 + jumlah karakter yang berbeda antara kata sekarang dengan kata akhir**.

7. Ulangi langkah 3-6 sampai Simpul Akhir ditemukan.

Algoritma A* dapat memberi solusi yang optimal atau *admissable* selama *heuristic cost* selalu "underestimate" *true cost*. Dalam kasus permainan Word Ladder, *true cost* adalah jumlah perubahan karakter yang dibutuhkan untuk merubah kata awal menjadi kata akhir. Karena *heuristic cost* = jumlah karakter yang berbeda, sedangkan pasti dibutuhkan sedemikian jumlah perubahan karakter untuk mencapai kata akhir, maka *heuristic cost* pasti selalu "underestimate" *total cost*. Dengan demikian *heuristic* yang digunakan pada algoritma A* adalah *admissable*.

Dengan penggunaan *heuristic cost* disamping *cost asli*, algoritma A* menjadi lebih efisien dibandingkan dengan algoritma UCS. Hal ini karena algoritma A* akan memeriksa terakhir tetangga yang tidak mendekati kata akhir (jumlah karakter yang sama sedikit). Akibatnya, algoritma A* dapat menemukan kata akhir dengan lebih cepat.

2. Source Code

2.1. Kelas Algorithm

class Algorithm
public static Pair<ArrayList<String>, Integer> UCS(String startWord, String endWord) throws NoSolutionException public static Pair<ArrayList<String>, Integer> GBFS(String startWord, String endWord) throws NoSolutionException public static Pair<ArrayList<String>, Integer> AStar(String startWord, String endWord) throws NoSolutionException

Kelas Algorithm menampung algoritma-algoritma route planning, yaitu dalam method UCS, GBFS, dan AStar. Ketiga method tersebut menerima 2 String sebagai parameter, yaitu kata awal dan kata akhir. Ketiga method mengembalikan ArrayList<String> yang berisi *Path* dari start word ke end word dan Integer yang berisi jumlah node yang dikunjungi. Penjelasan masing-masing method akan dijelaskan pada subbab berikut.

2.1.1 Algoritma UCS

public static Pair<ArrayList<String>, Integer> UCS(String startWord, String endWord) throws NoSolutionException
PriorityQueue<Pair<ArrayList<String>,Integer>> pQueue = new PriorityQueue<>(Pair::compareTo); HashMap<String, Boolean> checked = new HashMap<>(); Boolean found = false;

```
ArrayList<String> startPath = new ArrayList<>();
startPath.add(startWord);

ArrayList<String> finalPath = new ArrayList<>();
pQueue.add(new Pair<ArrayList<String>,Integer>(startPath, 0));
while (!found && !pQueue.isEmpty()) {
   Pair<ArrayList<String>,Integer> currentNode = pQueue.poll();
   String currentWord = currentNode.getKey().get(currentNode.getKey().size()-1);
   Integer currentCost = currentNode.getValue();
   checked.put(currentWord, true);

   if (currentWord.equals(endWord)) {
      found = true;
      finalPath = currentNode.getKey();
      break;
   }

   ArrayList<String> neighbours = Dictionary.getNeighbour(currentWord);
   for (String el : neighbours) {
      if (checked.get(el) == null) {
         ArrayList<String> newPath = new ArrayList<>(currentNode.getKey());
         newPath.add(el);

         pQueue.add(new Pair<ArrayList<String>,Integer>(newPath, currentCost+1));
      }
   }
}

if (!found) {
   throw new NoSolutionException("Tidak ada solusi yang ditemukan", checked.size());
}

return new Pair<ArrayList<String>, Integer>(finalPath, checked.size());
```

Method UCS menerima parameter berupa 2 String yang berisi start word dan end word. Method UCS mengembalikan ArrayList<String> yang berisi Path dari start word ke end word dan Integer yang berisi jumlah node yang dikunjungi. Method UCS menggunakan *priority queue* "pQueue" untuk menentukan urutan node yang akan diperiksa/dikunjungi, serta *map* "checked" untuk mencatat node yang sudah diperiksa/dikunjungi. *Priority Queue* berisi *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* dari kata awal dan Integer berisi *cost*. Semakin kecil nilai Integer/*cost* tersebut, semakin depan posisi *Pair* yang bersangkutan dalam priority queue. Pencarian solusi dilakukan dengan mengiterasi "pQueue" sampai habis atau sampai solusi ditemukan (ditandai dengan Boolean "found"). Pada setiap iterasi, apabila

kata yang sedang diperiksa bukan merupakan *end word*, program akan mencari tetangga dari kata yang sedang diperiksa. Kata yang diperiksa adalah elemen terakhir dari ArrayList<String> yang didapatkan dari *Pair*. Pencarian tetangga dilakukan menggunakan fungsi "Dictionary.getNeigbour(currentWord)". Setiap tetangga yang belum diperiksa kemudian ditambahkan ke dalam priority queue dengan *cost* baru berupa *cost* sekarang + 1. Tetangga ditambahkan dalam bentuk *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* sekarang di-*append* dengan tetangga dan Integer berisi *cost* baru. Setelah pencarian selesai, apabila solusi tidak ditemukan, method melempar/*throw* NoSolutionException yang berisi kalimat exception dan jumlah node yang dikunjungi. Apabila solusi ditemukan, method mengembalikan *path* dan jumlah node yang dikunjungi.

2.1.2 Algoritma Greedy Best First Search

```
public static Pair<ArrayList<String>, Integer> GBFS(String startWord, String endWord)
throws NoSolutionException
```

```
    PriorityQueue<Pair<ArrayList<String>,Integer>> pQueue = new
PriorityQueue<>(Pair::compareTo);
    Boolean found = false;
    Integer numChecked = 0;

    ArrayList<String> startPath = new ArrayList<>();
    startPath.add(startWord);

    ArrayList<String> finalPath = new ArrayList<>();
    pQueue.add(new Pair<ArrayList<String>,Integer>(startPath,
Dictionary.charDifference(startWord, endWord)));
    while (!found && !pQueue.isEmpty()) {
      Pair<ArrayList<String>,Integer> currentNode = pQueue.poll();
      String currentWord = currentNode.getKey().get(currentNode.getKey().size()-1);
      numChecked++;

      if (currentWord.equals(endWord)) {
        found = true;
        finalPath = currentNode.getKey();
        break;
      }

      ArrayList<String> neighbours = Dictionary.getNeighbour(currentWord);
      for (String el : neighbours) {
        ArrayList<String> newPath = new ArrayList<>(currentNode.getKey());
        newPath.add(el);

        pQueue.add(new Pair<ArrayList<String>,Integer>(newPath,
Dictionary.charDifference(el, endWord)));
```

```
        }
    }

    if (!found) {
        throw new NoSolutionException("Tidak ada solusi yang ditemukan", numChecked);
    }

    return new Pair<ArrayList<String>, Integer>(finalPath, numChecked);
```

Method GBFS menerima parameter berupa 2 String yang berisi start word dan end word. Method GBFS mengembalikan ArrayList<String> yang berisi Path dari start word ke end word dan Integer yang berisi jumlah node yang dikunjungi. Method GBFS menggunakan *priority queue* "pQueue" untuk menentukan urutan node yang akan diperiksa/dikunjungi. Tidak seperti method UCS dan AStar, method GBFS tidak mencatat node yang telah diperiksa/dikunjungi sehingga tidak menggunakan map "checked". *Priority Queue* berisi *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* dari kata awal dan Integer berisi *cost*. Semakin kecil nilai Integer/*cost* tersebut, semakin depan posisi *Pair* yang bersangkutan dalam priority queue. Pencarian solusi dilakukan dengan mengiterasi "pQueue" sampai habis atau sampai solusi ditemukan (ditandai dengan Boolean "found"). Pada setiap iterasi, apabila kata yang sedang diperiksa bukan merupakan *end word*, program akan mencari tetangga dari kata yang sedang diperiksa. Kata yang diperiksa adalah elemen terakhir dari ArrayList<String> yang didapatkan dari *Pair*. Pencarian tetangga dilakukan menggunakan fungsi "Dictionary.getNeigbour(currentWord)". Setiap tetangga kemudian akan ditambahkan ke dalam priority queue dengan *cost* baru berupa banyak karakter yang berbeda antara tetangga dengan kata akhir/*end word*. Perhitungan banyak karakter yang berbeda dilakukan dengan fungsi "Dictionary.charDifference(el, endWord)". Tetangga ditambahkan dalam bentuk *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* sekarang di-*append* dengan tetangga dan Integer berisi *cost* baru. Setelah pencarian selesai, apabila solusi tidak ditemukan, method melempar/*throw* NoSolutionException yang berisi kalimat exception dan jumlah node yang dikunjungi. Apabila solusi ditemukan, method mengembalikan *path* dan jumlah node yang dikunjungi.

2.1.3 Algoritma A*

```
public static Pair<ArrayList<String>, Integer> AStar(String startWord, String endWord)
throws NoSolutionException
```

```
    PriorityQueue<Pair<ArrayList<String>,Integer>> pQueue = new
PriorityQueue<>(Pair::compareTo);
    HashMap<String, Boolean> checked = new HashMap<>();
    Boolean found = false;
```

```
        ArrayList<String> startPath = new ArrayList<>();
        startPath.add(startWord);

        ArrayList<String> finalPath = new ArrayList<>();
        pQueue.add(new Pair<ArrayList<String>,Integer>(startPath,
0+Util.charDifference(startWord, endWord)));
        while (!found && !pQueue.isEmpty()) {
            Pair<ArrayList<String>,Integer> currentNode = pQueue.poll();
            String currentWord = currentNode.getKey().get(currentNode.getKey().size()-1);
            Integer currentCost = currentNode.getValue();
            checked.put(currentWord, true);

            if (currentWord.equals(endWord)) {
                found = true;
                finalPath = currentNode.getKey();
                break;
            }

            ArrayList<String> neighbours = Dictionary.getNeighbour(currentWord);
            for (String el : neighbours) {
                if (checked.get(el) == null) {
                    ArrayList<String> newPath = new ArrayList<>(currentNode.getKey());
                    newPath.add(el);

                    pQueue.add(new
Pair<ArrayList<String>,Integer>(newPath,currentCost+1+Util.charDifference(el, endWord)));
                }
            }
        }

        if (!found) {
            throw new NoSolutionException("Tidak ada solusi yang ditemukan", checked.size());
        }

        return new Pair<ArrayList<String>, Integer>(finalPath, checked.size());
```

Method Astar menerima parameter berupa 2 String yang berisi start word dan end word. Method Astar mengembalikan ArrayList<String> yang berisi Path dari start word ke end word dan Integer yang berisi jumlah node yang dikunjungi. Method AStar menggunakan *priority queue* "pQueue" untuk menentukan urutan node yang akan diperiksa/dikunjungi, serta *map* "checked" untuk mencatat node yang sudah diperiksa/dikunjungi. *Priority Queue* berisi *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* dari kata awal dan Integer berisi *cost*. Semakin kecil nilai Integer/*cost* tersebut, semakin depan posisi *Pair* yang bersangkutan dalam priority queue. Pencarian solusi dilakukan dengan mengiterasi "pQueue" sampai habis

atau sampai solusi ditemukan (ditandai dengan Boolean “found”). Pada setiap iterasi, apabila kata yang sedang diperiksa bukan merupakan *end word*, program akan mencari tetangga dari kata yang sedang diperiksa. Kata yang diperiksa adalah elemen terakhir dari ArrayList<String> yang didapatkan dari *Pair*. Pencarian tetangga dilakukan menggunakan fungsi “Dictionary.getNeigbour(currentWord)”. Setiap tetangga yang belum diperiksa kemudian ditambahkan ke dalam priority queue dengan *cost* baru berupa *cost* sekarang + 1 + banyak karakter yang berbeda antara tetangga dengan kata akhir/*end word*. Perhitungan banyak karakter yang berbeda dilakukan dengan fungsi “Dictionary.charDifference(el, endWord)”. Tetangga ditambahkan dalam bentuk *Pair* ArrayList<String> dan Integer. ArrayList<String> berisi *path* sekarang di-*append* dengan tetangga dan Integer berisi *cost* baru. Setelah pencarian selesai, apabila solusi tidak ditemukan, method melempar/*throw* NoSolutionException yang berisi kalimat exception dan jumlah node yang dikunjungi. Apabila solusi ditemukan, method mengembalikan *path* dan jumlah node yang dikunjungi.

2.4 Algoritma Main

class WordLadder

```
private final static Scanner inputScanner;

static {
    inputScanner = new Scanner(System.in);
}
public static void main(String[] args) {
    Scanner inputScanner = new Scanner(System.in);
    Boolean isLoop = true;
    String in = "cli";

    while (isLoop) {
        System.out.print("CLI/GUI? ");
        in = inputScanner.nextLine().split(" ")[0].toLowerCase();

        if (!in.equals("cli") && !in.equals("gui")) {
            System.out.println("Invalid input! Masukkan input CLI/GUI");
        }
        else {
            isLoop = false;
        }
    }

    if (in.equals("cli")) {
        wordLadderCLI();
    } else {
        wordLadderGUI();
    }
```

```
    inputScanner.close();
}

public static void wordLadderCLI() {
    new Dictionary(); // initialize dictionary

    Boolean isLoop = true;
    String startWord = "", endWord = "";
    while (isLoop) {
        System.out.print("Masukkan kata awal: ");
        startWord = inputScanner.nextLine().split(" ")[0].toUpperCase();
        int wordLength = startWord.length();

        System.out.print("Masukkan kata akhir: ");
        endWord = inputScanner.nextLine().split(" ")[0].toUpperCase();

        if (endWord.length() != wordLength) {
            System.out.println("Panjang kata harus sama!");
        }
        else if (!Dictionary.isInDictionary(startWord) || !Dictionary.isInDictionary(endWord)) {
            System.out.println("Kata tidak ada di dalam English Dictionary");
        }
        else {
            isLoop = false;
        }
    }

    System.out.println("Kata awal: " + startWord);
    System.out.println("Kata akhir: " + endWord);

    isLoop = true;
    String pilihanAlgoritma = "";
    while (isLoop) {
        System.out.println("Pilihan algoritma: \n" +
            "1. UCS \n" +
            "2. Greedy Best First Search \n" +
            "3. A* ");

        System.out.print("Masukkan pilihan algoritma: ");
        pilihanAlgoritma = inputScanner.nextLine();

        if (pilihanAlgoritma.equals("UCS")) {
            pilihanAlgoritma = "1";
        }
```

```
        else if (pilihanAlgoritma.equals("Greedy Best First Search")) {
            pilihanAlgoritma = "2";
        }
        else if (pilihanAlgoritma.equals("A*")) {
            pilihanAlgoritma = "3";
        }

        if (pilihanAlgoritma.equals("1") || pilihanAlgoritma.equals("2") ||
pilihanAlgoritma.equals("3"))
        {
            isLoop = false;
        }
        else {
            System.out.println("Input invalid!");
        }
    }

    System.out.println("Pilihan algoritma: " + pilihanAlgoritma);

    Pair<ArrayList<String>, Integer> solusi = new Pair<>(null, null);
    Boolean found = true;
    Integer numChecked = 0;

    long startTime = System.currentTimeMillis();
    System.out.println("Calculating...");
    try {
        if (pilihanAlgoritma.equals("1")){
            solusi = Algorithm.UCS(startWord, endWord);
        } else if (pilihanAlgoritma.equals("2")) {
            solusi = Algorithm.GBFS(startWord, endWord);
        } else {
            solusi = Algorithm.AStar(startWord, endWord);
        }
    }
    catch (NoSolutionException e) {
        System.out.println("Finished calculating");
        System.out.println(e.getMessage());
        if (e.getChecked() != null) {
            numChecked = e.getChecked();
        }
        found = false;
    }
    long endTime = System.currentTimeMillis();

    if (found) {
        System.out.println("Finished calculating");
```

```
        ArrayList<String> path = solusi.getKey();

        System.out.print("Path: ");

        for (int i=0;i<path.size();i++) {
            System.out.print(path.get(i));
            if (i != path.size()-1) {
                System.out.print(" -> ");
            }
        }
        System.out.print(" (" + (path.size()-1) + " steps)\n");

        numChecked = solusi.getValue();
    }

    System.out.println("Jumlah Node yang dikunjungi: " + numChecked);
    System.out.println("Time: " + (endTime-startTime) + "ms");
}

public static void wordLadderGUI() {
    new WordLadderGUI();
}
```

Fungsi Main tertampung dalam kelas WordLadder. Secara singkat, main pertama-tama menerima input apakah program ingin dijalankan secara CLI atau GUI. Apabila program dijalankan secara GUI, program memanggil kelas WordLadderGUI(). Apabila program dijalankan secara CLI, program memanggil method wordLadderCLI(). Pada GUI maupun CLI, program meminta beberapa input berupa kata awal, kata akhir, dan algoritma yang digunakan. Apabila panjang kata berbeda atau kata tidak ada di dalam *dictionary,* maka input akan diulang. Pemeriksaan kata di dalam *dictionary* dilakukan menggunakan fungsi “Dictionary.isInDictionary(word)”. Setelah itu, program akan menjalankan algoritma sembari mencatat waktu. Apabila solusi ditemukan, program akan mencetak *path* beserta banyak *steps* dalam *path*, jumlah node yang dikunjungi, dan waktu yang dibutuhkan. Jika solusi tidak ditemukan, program hanya mencetak jumlah node yang dikunjungi dan waktu yang dibutuhkan.

2.5 Kelas Utilitas

2.5.1 Kelas Dictionary

```
class Dictionary
```

```
    private static HashMap<String, Boolean> wordList;

    static {
        wordList = new HashMap<String, Boolean>();
```

```
    try {
        File wordFile = new File("wordList.txt");
        Scanner readScanner = new Scanner(wordFile);
        System.out.println("Importing dictionary...");
        while (readScanner.hasNextLine()) {
            String word = readScanner.nextLine().toUpperCase();
            wordList.put(word,true);
            // System.out.println(word);
        }
        System.out.println("Finished importing");

        readScanner.close();
    } catch (FileNotFoundException e) {
        System.out.println("File tidak ditemukan! Tolong tempatkan wordList pada /src");
    }

    // System.out.println(wordList);
}

public static Boolean isInDictionary(String word) {
    if (wordList.get(word) == null) {
        return false;
    }

    return true;
}

public static ArrayList<String> getNeighbour(String word) {
    ArrayList<String> neighbours = new ArrayList<String>();
    for (String el : wordList.keySet()) {
        if (charDifference(word, el) == 1) {
            neighbours.add(el);
        }
    }

    return neighbours;
}

static Integer charDifference(String word1, String word2) {
    if (word1.length() != word2.length()) { // if different length
        return -1;
    }

    int ctr = 0;
    for (int i=0; i<word1.length(); i++) {
```

```
            if (word1.charAt(i) != word2.charAt(i)) {
                ctr++;
            }
        }

        return ctr;
    }
```

Kelas Dictionary merupakan kelas utilitas yang berfungsi memanajemen kamus/*dictionary*. Kelas Dictionary bersifat static, terdiri atas map wordList yang menampung kata-kata dalam *dictionary*, dan terdiri atas fungsi isInDictionary(String), getNeighbour(String), dan charDifference(String). Fungsi isInDictionary(String word) berfungsi memeriksa apakah "word" berada dalam "wordList". Fungsi getNeighbour(String word) berfungsi mencari tetangga dari "word", yaitu kata-kata yang hanya berbeda 1 karakter dengan "word". Fungsi getNeighbour(String word) mengembalikan tetangga-tetangga yang tertampung dalam ArrayList<String>. Fungsi charDifference(String word1, String word2) berfungsi mencari banyak karakter yang berbeda antara "word1" dan "word2".

2.5.2 Kelas Pair

```
class Pair<X,Y extends Number> implements Comparable<Pair<X,Y>>
```

```
    private X key;
    private Y value;

    public Pair(X key, Y value) {
        this.key = key;
        this.value = value;
    }

    public X getKey() {
        return this.key;
    }

    public Y getValue() {
        return this.value;
    }

    public int compareTo(Pair<X,Y> p) {
        if (this.getValue().doubleValue() < p.getValue().doubleValue()) {
            return -1;
        }
        else if (this.getValue().doubleValue() > p.getValue().doubleValue()) {
            return 1;
```

```
    }
    else return 0;
}
```

Kelas Pair adalah generic class yang menampung 2 elemen X dan Y. Y merupakan turunan dari *Number* yang digunakan sebagai penanda *cost*. X dapat diambil dari Pair menggunakan fungsi "getKey()" dan Y dapat diambil dari Pair menggunakan fungsi "getValue()". Kelas Pair menurunkan (implements) interface *Comparable* supaya dapat digunakan dalam priority queue. Fungsi compareTo(Pair) berfungsi membandingkan Pair dengan Pair lain supaya dapat dibandingkan dalam priority queue.

2.5.2 Exception Tambahan

```
class NoSolutionException extends Exception

    private Integer numChecked;

    public NoSolutionException(String s, Integer c) {
        super(s);
    }

    public Integer getChecked() {
        return numChecked;
    }
```

NoSolutionException adalah Exception tambahan yang digunakan untuk menandakan apabila solusi tidak ditemukan. NoSolutionException memiliki private variable tambahan yaitu numChecked yang berfungsi menampung jumlah node yang dikunjungi. Ketika NoSolutionException di-*catch*, numChecked, yaitu jumlah node yang dikunjungi, dapat diperoleh menggunakan fungsi "getChecked()".

3. Uji Coba

3.1. BASE → SANE

3.1.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: base
Masukkan kata akhir: sane
Kata awal: BASE
Kata akhir: SANE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Path: BASE -> BANE -> SANE (2 steps)
Jumlah Node yang dikunjungi: 70
Time: 468ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.1.2. GBFS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: base
Masukkan kata akhir: sane
Kata awal: BASE
Kata akhir: SANE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Finished calculating
Path: BASE -> BANE -> SANE (2 steps)
Jumlah Node yang dikunjungi: 3
Time: 47ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.1.3. A*

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: base
Masukkan kata akhir: sane
Kata awal: BASE
Kata akhir: SANE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Path: BASE -> BANE -> SANE (2 steps)
Jumlah Node yang dikunjungi: 11
Time: 108ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.2. MAN → APE

3.2.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: man
Masukkan kata akhir: ape
Kata awal: MAN
Kata akhir: APE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Path: MAN -> WAN -> WYN -> WYE -> AYE -> APE (5 steps)
Jumlah Node yang dikunjungi: 878
Time: 90728ms
```

3.2.2. GBFS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: man
Masukkan kata akhir: ape
Kata awal: MAN
Kata akhir: APE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Finished calculating
Path: MAN -> MAE -> NAE -> SAE -> WAE -> WYE -> AYE -> APE (7 steps)
Jumlah Node yang dikunjungi: 16
Time: 118ms
```

3.2.3. A

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: man
Masukkan kata akhir: ape
Kata awal: MAN
Kata akhir: APE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Path: MAN -> MAE -> WAE -> WYE -> AYE -> APE (5 steps)
Jumlah Node yang dikunjungi: 568
Time: 14017ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.3. PITCH → TENTS

3.3.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pitch
Masukkan kata akhir: tents
Kata awal: PITCH
Kata akhir: TENTS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Path: PITCH -> WITCH -> WINCH -> WENCH -> TENCH -> TENTH -> TENTS (6 steps)
Jumlah Node yang dikunjungi: 161
Time: 3016ms
```

3.3.2. GBFS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pitch
Masukkan kata akhir: tents
Kata awal: PITCH
Kata akhir: TENTS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Finished calculating
Path: PITCH -> PINCH -> WINCH -> WENCH -> TENCH -> TENTH -> TENTS (6 steps)
Jumlah Node yang dikunjungi: 7
Time: 64ms
```

3.3.3. A*

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pitch
Masukkan kata akhir: tents
Kata awal: PITCH
Kata akhir: TENTS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Path: PITCH -> PINCH -> WINCH -> WENCH -> TENCH -> TENTH -> TENTS (6 steps)
Jumlah Node yang dikunjungi: 64
Time: 956ms
```

3.4. PITY → GOOD

3.4.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pity
Masukkan kata akhir: good
Kata awal: PITY
Kata akhir: GOOD
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Path: PITY -> PINY -> PONY -> POND -> POOD -> GOOD (5 steps)
Jumlah Node yang dikunjungi: 2691
Time: 118378ms
```

3.4.2. GBFS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pity
Masukkan kata akhir: good
Kata awal: PITY
Kata akhir: GOOD
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Finished calculating
Path: PITY -> CITY -> CITE -> COTE -> DOTE -> MOTE -> MOVE -> HOVE -> HOKE -> HOWE -> HOWL -> HOWK -> GOWK -> GOOK -> GOOD (14 steps)
Jumlah Node yang dikunjungi: 40
Time: 247ms
```

3.4.3. A*

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: pity
Masukkan kata akhir: good
Kata awal: PITY
Kata akhir: GOOD
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Path: PITY -> PINY -> PONY -> POND -> POOD -> GOOD (5 steps)
Jumlah Node yang dikunjungi: 434
Time: 4715ms
```

3.5. CAESAR → ABACAS

3.5.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: caesar
Masukkan kata akhir: abacas
Kata awal: CAESAR
Kata akhir: ABACAS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Tidak ada solusi yang ditemukan
Jumlah Node yang dikunjungi: 0
Time: 18ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.5.2. GBFS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: caesar
Masukkan kata akhir: abacas
Kata awal: CAESAR
Kata akhir: ABACAS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Finished calculating
Tidak ada solusi yang ditemukan
Jumlah Node yang dikunjungi: 0
Time: 11ms
```

3.5.3. A*

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: caesar
Masukkan kata akhir: abacas
Kata awal: CAESAR
Kata akhir: ABACAS
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Tidak ada solusi yang ditemukan
Jumlah Node yang dikunjungi: 0
Time: 24ms
```

3.6. HAPPY → NIECE

3.6.1. UCS

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: happy
Masukkan kata akhir: niece
Kata awal: HAPPY
Kata akhir: NIECE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 1
Pilihan algoritma: 1
Calculating...
Finished calculating
Path: HAPPY -> HARPY -> HARPS -> HARES -> HIRES -> SIRES -> SINES -> SINGS -> SINGE -> SIEGE -> SIEVE -> NIEVE -> NIECE (12 steps)
Jumlah Node yang dikunjungi: 7184
Time: 346963ms
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002>
```

3.6.2. GBFS

solusi tidak ditemukan

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: happy
Masukkan kata akhir: niece
Kata awal: HAPPY
Kata akhir: NIECE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 2
Pilihan algoritma: 2
Calculating...
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.base/java.util.Arrays.copyOf(Arrays.java:3689)
        at java.base/java.util.ArrayList.toArray(ArrayList.java:401)
        at java.base/java.util.ArrayList.<init>(ArrayList.java:179)
        at Algorithm.GBFS(Algorithm.java:67)
        at WordLadderCLI.main(WordLadderCLI.java:75)
```

3.6.3. A*

```
PS C:\Ariel\Kuliah\MatKul\Semester 3\Strategi Algoritma\Tucil 3\Tucil3_13522002> java -jar ./bin/WordLadderCLI.jar
Importing dictionary...
Finished importing
Masukkan kata awal: happy
Masukkan kata akhir: niece
Kata awal: HAPPY
Kata akhir: NIECE
Pilihan algoritma:
1. UCS
2. Greedy Best First Search
3. A*
Masukkan pilihan algoritma: 3
Pilihan algoritma: 3
Calculating...
Finished calculating
Path: HAPPY -> HARPY -> HARPS -> HARES -> HIRES -> SIRES -> SINES -> SINGS -> SINGE -> SIEGE -> SIEVE -> NIEVE -> NIECE (12 steps)
Jumlah Node yang dikunjungi: 5277
Time: 238527ms
```

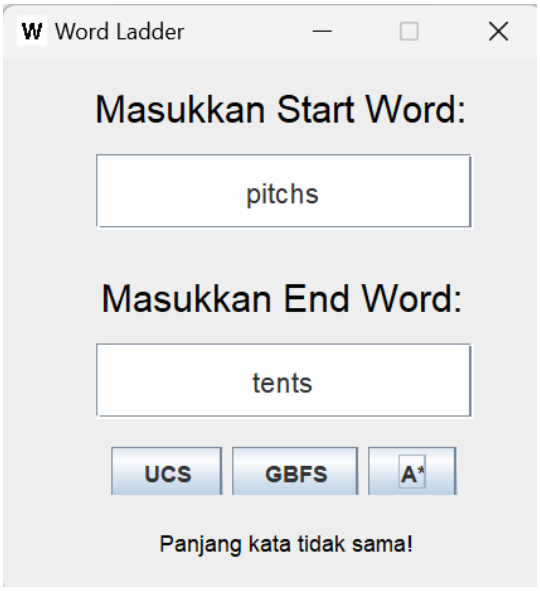
4. Analisis Perbandingan Solusi

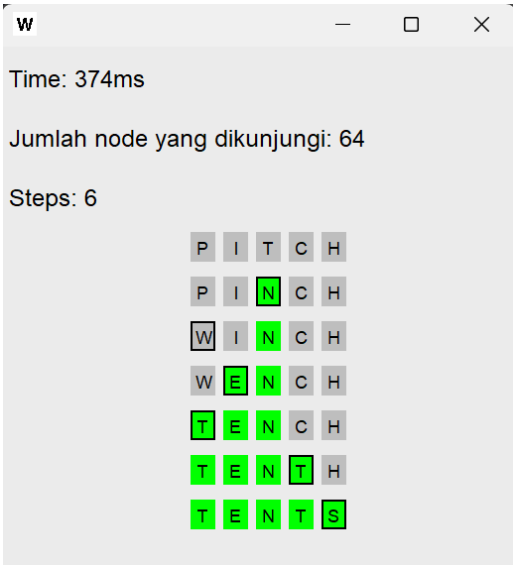
Berdasarkan uji coba 3.1-3.4, dapat dilihat bahwa algoritma A* lebih cepat daripada algoritma UCS dan algoritma GBFS lebih cepat daripada keduanya. Terlihat juga bahwa algoritma A* perlu mengunjungi lebih sedikit simpul daripada algoritma UCS. Hal ini sesuai dengan teori bahwa algoritma A* lebih efisien daripada algoritma UCS. Dalam uji coba 3.2 dan 3.4, dapat dilihat juga bahwa algoritma GBFS belum tentu memberikan solusi yang optimal, meskipun membutuhkan waktu paling sedikit dan jumlah simpul yang harus dikunjunginya paling sedikit. Dalam uji coba 3.6, terlihat juga bahwa algoritma GBFS mungkin tidak dapat menemukan solusi karena terjebak dalam local optima. Terakhir, uji coba 3.5 menunjukkan kasus ketika tidak ada rute menuju simpul tujuan.

Meskipun, algoritma GBFS belum tentu menghasilkan solusi optimal, perlu diperhatikan bahwa algoritma GBFS selalu paling cepat dalam menghasilkan solusi. Hal ini menunjukkan bahwa algoritma GBFS efektif apabila permasalahan tidak memerlukan solusi yang optimal dan jika algoritma tidak terjebak dalam local optima. Uji coba juga menunjukkan bahwa pada umumnya, algoritma A* selalu lebih baik daripada algoritma UCS, selama heuristik yang digunakan *admissable*.

5. Penjelasan Implementasi Bonus

Berikut adalah tampilan dari GUI:





Berikut adalah source code untuk GUI:

```
public class WordLadderGUI implements ActionListener
```

```
    JFrame frame = new JFrame();
    JLabel statusLabel = new JLabel();

    JButton ucsButton;
    JButton gbfsButton;
    JButton astarButton;
```

```
JTextField startWordEntry;
JTextField endWordEntry;

public static void main(String[] args) {
   new WordLadderGUI();
}

WordLadderGUI() {
   // Start Word Panel Setup ======================
   JPanel startWordPanel = new JPanel();
   startWordPanel.setLayout(null);
   // startWordPanel.setBackground(Color.red);
   startWordPanel.setBounds(0, 0, 300, 100);

   JLabel startWordLabel = new JLabel();
   startWordLabel.setText("Masukkan Start Word: ");
   startWordLabel.setFont(new Font("Bebas", Font.PLAIN,20));
   startWordLabel.setForeground(Color.black);
   startWordLabel.setBounds(0,0, 300,50);
   startWordLabel.setHorizontalTextPosition(JLabel.CENTER);
   startWordLabel.setHorizontalAlignment(JLabel.CENTER);

   startWordEntry = new JTextField();
   startWordEntry.setBounds(50,50,200,40);
   startWordEntry.setHorizontalAlignment(JTextField.CENTER);
   startWordEntry.setFont(new Font("Bebas", Font.PLAIN,15));

   startWordPanel.add(startWordLabel);
   startWordPanel.add(startWordEntry);

   // Start Word Panel Setup ======================

   // End Word Panel Setup ========================
   JPanel endWordPanel = new JPanel();
   endWordPanel.setLayout(null);
   // endWordPanel.setBackground(Color.blue);
   endWordPanel.setBounds(0, 100, 300, 100);

   JLabel endWordLabel = new JLabel();
   endWordLabel.setText("Masukkan End Word: ");
   endWordLabel.setFont(new Font("Bebas", Font.PLAIN,20));
   endWordLabel.setForeground(Color.black);
   endWordLabel.setBounds(0,0, 300,50);
   endWordLabel.setHorizontalTextPosition(JLabel.CENTER);
   endWordLabel.setHorizontalAlignment(JLabel.CENTER);
```

```
endWordEntry = new JTextField();
endWordEntry.setBounds(50,50,200,40);
endWordEntry.setHorizontalAlignment(JTextField.CENTER);
endWordEntry.setFont(new Font("Bebas", Font.PLAIN,15));

endWordPanel.add(endWordLabel);
endWordPanel.add(endWordEntry);

// End Word Panel Setup ========================

// Button Panel Setup ============================
JPanel buttonPanel = new JPanel();
buttonPanel.setBounds(0, 200, 300, 30);

ucsButton = new JButton();
ucsButton.setSize(100,30);
ucsButton.setText("UCS");
ucsButton.setAlignmentX(JButton.CENTER);
ucsButton.addActionListener(this);

gbfsButton = new JButton();
gbfsButton.setSize(100,30);
gbfsButton.setText("GBFS");
gbfsButton.setAlignmentX(JButton.CENTER);
gbfsButton.addActionListener(this);

astarButton = new JButton();
astarButton.setSize(100,30);
astarButton.setText("A*");
astarButton.setAlignmentX(JButton.CENTER);
astarButton.addActionListener(this);

buttonPanel.add(ucsButton);
buttonPanel.add(gbfsButton);
buttonPanel.add(astarButton);

// Button Panel Setup ============================

// Status Label Setup ============================
statusLabel = new JLabel();
statusLabel.setText("...");
statusLabel.setFont(new Font("Bebas", Font.PLAIN,12));
statusLabel.setForeground(Color.black);
statusLabel.setBounds(0,230, 300,50);
statusLabel.setHorizontalAlignment(JLabel.CENTER);
// Status Label Setup ============================
```

```
    // Frame setup
    frame.setTitle("Word Ladder");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(null);
    frame.setResizable(false);
    frame.setSize(300,320);
    frame.setVisible(true);

    frame.add(startWordPanel);
    frame.add(endWordPanel);
    frame.add(buttonPanel);
    frame.add(statusLabel);

    ImageIcon image = new ImageIcon("./src/logo.png"); // if not found, ignore
    frame.setIconImage(image.getImage());
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==ucsButton) {
        System.out.println("UCS");

        String startWord = startWordEntry.getText().toUpperCase();
        String endWord = endWordEntry.getText().toUpperCase();

        if (startWord.length() != endWord.length()) {
            statusLabel.setText("Panjang kata tidak sama!");
        }
        else if (!Dictionary.isInDictionary(startWord) || !Dictionary.isInDictionary(endWord)) {
            statusLabel.setText("Kata tidak ada di dalam Dictionary");
        }
        else {
            statusLabel.setText("Calculating...");
            statusLabel.paintImmediately(statusLabel.getVisibleRect());
            Pair<ArrayList<String>, Integer> solusi = new Pair<>(new ArrayList<>(), 0);

            long startTime = System.currentTimeMillis();
            try {
                solusi = Algorithm.UCS(startWord, endWord);
            }
            catch (NoSolutionException ex) {
                System.out.println(ex.getMessage());
            }
            long endTime = System.currentTimeMillis();
```

```
            frame.dispose();
            new SolutionFrame(solusi.getKey(), endTime-startTime, solusi.getValue());
        }
    }
    else if (e.getSource()==gbfsButton) {
        System.out.println("GBFS");

        String startWord = startWordEntry.getText().toUpperCase();
        String endWord = endWordEntry.getText().toUpperCase();

        if (startWord.length() != endWord.length()) {
            statusLabel.setText("Panjang kata tidak sama!");
        }
        else if (!Dictionary.isInDictionary(startWord) || !Dictionary.isInDictionary(endWord)) {
            statusLabel.setText("Kata tidak ada di dalam Dictionary");
        }
        else {
            statusLabel.setText("Calculating...");
            statusLabel.paintImmediately(statusLabel.getVisibleRect());
            Pair<ArrayList<String>, Integer> solusi = new Pair<>(new ArrayList<>(), 0);

            long startTime = System.currentTimeMillis();
            try {
                solusi = Algorithm.GBFS(startWord, endWord);
            }
            catch (NoSolutionException ex) {
                System.out.println(ex.getMessage());
            }
            long endTime = System.currentTimeMillis();

            frame.dispose();
            new SolutionFrame(solusi.getKey(), endTime-startTime, solusi.getValue());
        }
    }
    else if (e.getSource()==astarButton) {
        System.out.println("A*");

        String startWord = startWordEntry.getText().toUpperCase();
        String endWord = endWordEntry.getText().toUpperCase();

        if (startWord.length() != endWord.length()) {
            statusLabel.setText("Panjang kata tidak sama!");
        }
        else if (!Dictionary.isInDictionary(startWord) || !Dictionary.isInDictionary(endWord)) {
            statusLabel.setText("Kata tidak ada di dalam Dictionary");
        }
```

```
        else {
            statusLabel.setText("Calculating...");
            statusLabel.paintImmediately(statusLabel.getVisibleRect());
            Pair<ArrayList<String>, Integer> solusi = new Pair<>(new ArrayList<>(), 0);

            long startTime = System.currentTimeMillis();
            try {
                solusi = Algorithm.AStar(startWord, endWord);
            }
            catch (NoSolutionException ex) {
                System.out.println(ex.getMessage());
            }
            long endTime = System.currentTimeMillis();

            frame.dispose();
            new SolutionFrame(solusi.getKey(), endTime-startTime, solusi.getValue());
        }
    }
}
```

class SolutionFrame

```
public SolutionFrame(ArrayList<String> path, Number time, Integer numChecked) {
    // Info Panel Setup ============================
    JPanel infoPanel = new JPanel();
    infoPanel.setLayout(null);
    infoPanel.setBounds(0, 0, 360, 120);
    // infoPanel.setBackground(Color.red);

    JLabel timeLabel = new JLabel();
    timeLabel.setText("Time: " + time + "ms");
    timeLabel.setFont(new Font("Bebas", Font.PLAIN, 16));
    timeLabel.setForeground(Color.black);
    timeLabel.setBounds(5, 5, 360, 30);

    JLabel numCheckedLabel = new JLabel();
    numCheckedLabel.setText("Jumlah node yang dikunjungi: " + numChecked);
    numCheckedLabel.setFont(new Font("Bebas", Font.PLAIN, 16));
    numCheckedLabel.setForeground(Color.black);
    numCheckedLabel.setBounds(5, 45, 360, 30);

    JLabel numStepsLabel = new JLabel();
    if (path.size() > 0) {
```

```
    numStepsLabel.setText("Steps: " + (path.size()-1));
} else { // not found
    numStepsLabel.setText("Steps: solusi tidak ditemukan");
}
numStepsLabel.setFont(new Font("Bebas", Font.PLAIN, 16));
numStepsLabel.setForeground(Color.black);
numStepsLabel.setBounds(5, 85, 360, 30);

infoPanel.add(timeLabel);
infoPanel.add(numCheckedLabel);
infoPanel.add(numStepsLabel);

// Info Panel Setup ============================

// Steps Panel Setup ===========================
JPanel stepsPanel = new JPanel();
stepsPanel.setBounds(0, 120, 360, 30*path.size());
stepsPanel.setLayout(null);
// stepsPanel.setBackground(Color.blue);

if (path.size() > 0) {
    String prevStep = "";
    String finalStep = path.get(path.size()-1);
    for (int i = 0; i < path.size(); i++) {
        String currentStep = path.get(i);

        JPanel currentStepPanel = new JPanel();
        currentStepPanel.setBounds(0, 30*i, 360, 30);
        currentStepPanel.setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
        // currentStepPanel.setBackground(Color.orange);

        Integer strLength = currentStep.length();
        for (int j=0; j<strLength; j++) { // create box for each char
            JLabel currentStepLabel = new JLabel();
            currentStepLabel.setText(""+currentStep.charAt(j));
            currentStepLabel.setFont(new Font("Bebas", Font.PLAIN, 12));
            currentStepLabel.setForeground(Color.black);
            currentStepLabel.setVerticalAlignment(JLabel.CENTER);
            currentStepLabel.setHorizontalAlignment(JLabel.CENTER);
            currentStepLabel.setPreferredSize(new Dimension(17,20));
            // currentStepLabel.setBorder(border);
            currentStepLabel.setOpaque(true);

            if (currentStep.charAt(j) == finalStep.charAt(j)) {
                currentStepLabel.setBackground(Color.green);
            }
```

```
                else {
                    currentStepLabel.setBackground(Color.lightGray);
                }

                if (i > 0 && currentStep.charAt(j) != prevStep.charAt(j)) {
                    currentStepLabel.setBorder(border);
                }

                currentStepPanel.add(currentStepLabel);
            }

            stepsPanel.add(currentStepPanel);

            prevStep = currentStep;
        }
    }

    // Steps Panel Setup ========================

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(null);
    frame.setSize(360, 180+(30*path.size()));
    frame.setVisible(true);

    frame.add(infoPanel);
    frame.add(stepsPanel);

    ImageIcon image = new ImageIcon("./src/logo.png"); // if not found, ignore
    frame.setIconImage(image.getImage());
}
```

GUI untuk program diimplementasi menggunakan Java Swing. Source code GUI terdiri atas kelas WordLadderGUI dan kelas SolutionFrame. Kelas WordLadderGUI adalah kelas yang mengatur tampilan menu utama yang menerima input kata dan pilihan algoritma. Kelas SolutionFrame adalah kelas yang mengatur tampilan solusi. Pada menu utama, terdapat Entry/TextInputField untuk memasukkan kata awal dan kata akhir, 3 button untuk memilih pilihan algoritma, dan sebuah Text Label di bawah untuk memberi tahu status program. Pada tampilan solusi, ditampilkan waktu, jumlah node yang dikunjungi, jumlah steps rute solusi, serta perubahan kata pada setiap tahap. Constructor SolutionFrame (tampilan solusi) menerima parameter ArrayList<String> path, Number time, dan Integer numChecked, yaitu rute, waktu, dan jumlah node yang dikunjungi yang akan ditampilkan pada tampilan solusi.

Cara kerja pemanggilan algoritma adalah sebagai berikut:

1. WordLadderGUI (main menu) meng-*implement ActionListener* dan menunggu salah satu dari ketiga button ditekan.
2. Ketika salah satu button ditekan, program memanggil algoritma UCS/GBFS/A* sesuai dengan button yang ditekan. Solusi algoritma lalu disimpan ke variabel yang akan dilempar sebagai parameter pembuatan SolutionFrame.
3. WordLadderGUI lalu membuat SolutionFrame dengan parameter berupa solusi yang didapatkan.
4. Solusi ditampilkan.

Pranala Repository

https://github.com/Ariel-HS/Tucil3_13522002.git

Referensi

[Bro Code]. (2020, September 14). Java GUI: Full Course ☕ (FREE) [Video]. Youtube. https://www.youtube.com/watch?v=Kmgo00avvEw&t=1388s

Java Platform SE 8. (n.d.). https://docs.oracle.com/javase/8/docs/api/

Munir, R. (n.d.). Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/

Lampiran

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	✔	