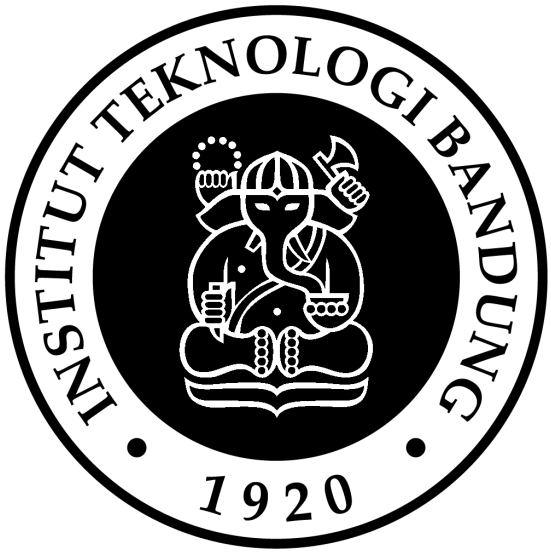
Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II tahun 2023/2024

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



Disusun Oleh :
Muhammad Dzaki Arta 13522149

Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Daftar Isi

Bab I
Teori Dasar

1.1 Permainan Word Ladder

Word ladder (juga dikenal sebagai Doublets, word-links, teka-teki ubah-kata, paragrams, laddergrams, atau word golf) adalah permainan kata yang diciptakan oleh Lewis Carroll. Teka-teki word ladder dimulai dengan dua kata, dan untuk memecahkan teka-teki tersebut seseorang harus menemukan rangkaian kata lain untuk menghubungkan kedua kata tersebut, di mana dua kata berdekatan (yaitu, kata-kata dalam langkah-langkah berurutan) berbeda satu huruf.

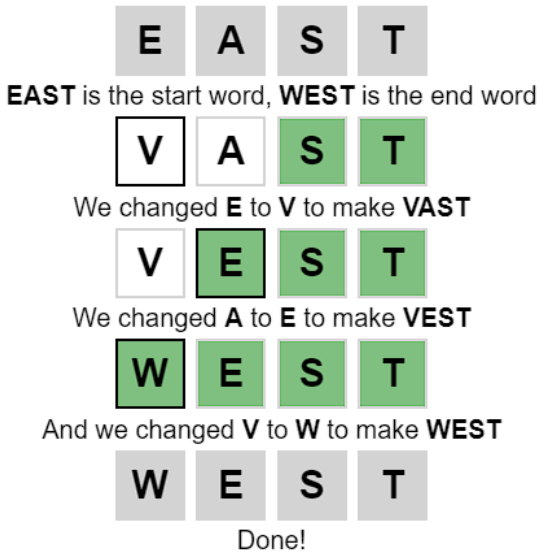
Lewis Carroll mengatakan bahwa dia menciptakan permainan tersebut pada Natal tahun 1877. Carroll merancang permainan kata untuk Julia dan Ethel Arnold. Catatan pertama tentang permainan tersebut dalam diari Carroll adalah pada tanggal 12 Maret 1878, yang awalnya dia sebut "Word-links", dan dijelaskan sebagai permainan untuk dua pemain. Carroll menerbitkan serangkaian teka-teki word ladder dan solusinya, yang kemudian dia sebut "Doublets", di majalah Vanity Fair, dimulai dari edisi 29 Maret 1879. Tahun itu juga, permainan itu dibuat menjadi buku, diterbitkan oleh Macmillan and Co.

J. E. Surrick dan L. M. Conant menerbitkan sebuah buku Laddergrams dari teka-teki semacam itu pada tahun 1927.

Vladimir Nabokov menyebut permainan tersebut dengan nama "word golf" dalam novel Pale Fire, di mana narator mengatakan 'beberapa catatan saya adalah: benci—cinta dalam tiga langkah, gadis—pria dalam empat, dan hidup—mati dalam lima (dengan "pinjam" di tengah).'

Permainan itu dihidupkan kembali di Australia pada tahun 1990-an oleh The Canberra Times dengan nama "Stepword".

Word ladder sering muncul dalam teka-teki silang New York Times.



Gambar 1.1 Permainan Word Ladder

1.2 Algoritma Uniform Cost Search

Uniform Cost Search merupakan salah satu algoritma penjelajahan graf berbobot dengan informasi. Algoritma ini dapat digunakan untuk menentukan jalur dengan harga terendah dari simpul akar ke simpul target (solusi optimal). Uniform Cost Search bekerja dengan cara mempertimbangkan semua jalur yang mungkin dari simpul akar ke simpul target dengan memilih jalur dengan harga terendah pada setiap iterasinya dengan memanfaatkan struktur data PriorityQueue. Algoritma ini akan terus melakukan iterasi selama masih ada jalur dengan harga yang lebih rendah walau simpul target telah ditemukan atau sampai PriorityQueue kosong jika simpul target belum juga ditemukan. Harga yang dimiliki simpul n pada graf menurut algoritma ini adalah bobot total dari setiap simpul yang dilewati dari akar sampai ke simpul n yaitu g(n).

1.3 Algoritma Greedy Best First Search

Greedy Best Search merupakan salah satu algoritma penjelajahan graf berbobot dengan informasi. Algoritma ini dapat digunakan untuk menentukan jalur dari simpul akar ke simpul target dengan cukup cepat karena menggunakan konsep Greedy. Uniform Cost Search bekerja dengan cara mempertimbangkan semua jalur yang mungkin dari simpul current ke simpul next dan akan melakukan perluasan pencarian di simpul next dengan harga terendah, setiap simpul yang dijelajah akan dimasukkan ke PriorityQueue yang terurut berdasarkan harga tiap simpul. Harga yang dimiliki simpul n pada graf menurut algoritma ini adalah estimasi harga dari simpul n ke simpul target yang dihitung menggunakan fungsi heuristik yaitu h(n). Algoritma ini akan berhenti melakukan pencarian ketika simpul target ditemukan atau sampai PriorityQueue kosong jika simpul target belum juga ditemukan.

1.4 Algoritma A*

A* (dibaca A Star) merupakan salah satu algoritma penjelajahan graf berbobot dengan informasi. Algoritma ini dapat digunakan untuk menentukan jalur dengan harga terendah dari simpul akar ke simpul target (solusi optimal). A* bekerja dengan cara mempertimbangkan semua jalur yang mungkin dari simpul akar ke simpul target dengan memilih jalur dengan harga terendah pada setiap iterasinya dengan memanfaatkan struktur data PriorityQueue. Algoritma ini akan terus melakukan iterasi selama masih ada jalur dengan harga yang lebih rendah walau simpul target telah ditemukan atau sampai PriorityQueue kosong jika simpul target belum juga ditemukan. Harga yang dimiliki simpul n pada graf menurut algoritma ini adalah f(n) yaitu bobot total dari setiap simpul yang dilewati dari akar sampai ke simpul n yaitu g(n) ditambah dengan estimasi harga dari simpul n ke simpul target yang dihitung menggunakan fungsi heuristik yaitu h(n).

BAB II
Implementasi Algoritma

2.1 Kelas Node

Kelas Node yang dipakai memiliki atribut word yang bertipe String atribut ini digunakan untuk menyimpan kata, atribut len yang bertipe Integer atribut ini digunakan untuk mengetahui panjang kata, atribut value bertipe int atribut ini digunakan untuk menjadi tolak ukur antara Node lain, atribut age bertipe int atribut ini menjadi tolak ukur untuk menghitung perbandingan Node yang memiliki value yang sama sehingga Node yang pertama muncul (age terendah) akan menjadi pilihannya, atribut parent dengan tipe data Node atribut ini digunakan untuk mengetahui parent dari Node tersebut

```
public class Node implements Comparable<Node> {
    private String word;
    private int len;
    private int value;
    private int age;
    private Node parent;
```

Gambar 2.1 Atribut Kelas Node

Kelas Node memiliki beberapa metode getter seperti getWord(), getLen(), getValue(), getAge(), getParent(). Kelas method juga memiliki beberapa metode setter seperti setWord(), setParent(), setValue(), addValue(), addParent().

```
// Getter
public String getWord() {
    return word;
}
public int getLen() {
    return len;
}
public int getValue() {
    return value;
}

public int getAge() {
    return age;
}

public Node getParent() {
    return parent;
}
```

```
// Setter
public void setWord(String word) {
    this.word = word;
    this.len = word.length();
}
public void setParent(Node parent) {
    this.parent = parent;
}
public void setValue(int value) {
    this.value = value;
}

// Menambahkan value
public void addValue() {
    this.value ++;
}

// Menambah age
public void addAge() {
    this.age ++;
}
```

Gambar 2.2 Metode getter dan setter pada kelas Node

Kelas Node juga memiliki metode tambahan yaitu metode isParentEmpty() metode ini untuk mengecek apakah Node tersebut tidak memiliki parent. Metode compareTo(Node node)

metode ini me override metode yang berasal dari kelas Comparable metode ini digunakan untuk perbandingan antar 2 node nantinya berdasarkan value dan age

```java
// Apakah parent kosong
public void isParentEmpty() {
    if (this.parent == null) {
        System.out.println(x:"Parent kosong");
    } else {
        System.out.println(x:"Parent tidak kosong");
    }
}

@Override
public int compareTo(Node node) {
   if (this.value < node.value) {
       return -1;
    } else if (this.value > node.value) {
        return 1;
    } else if (this.age < node.age) {
        return -1;
    }else if(this.age > node.age){
        return 1;
    }else{
        return 0;
     }
}
```

Gambar 2.3 Metode tambahan kelas Node

2.2 Kelas UCS

Kelas UCS memiliki 1 fungsi yaitu find, fungsi ini akan menerima firstword yang bertipe string, lastword bertipe string, dan words bertipe list of string untuk digunakan sebagai dictionary dan akan menghasilkan ArrayList of object.

Pada implementasinya value yang dihitung merupakan jarak antar node yang bernilai 1 sehingga karena setiap iterasi f(n) akan bertambah 1 algoritma ini akan sama dengan BFS.

Program akan melakukan membuat simpul firstword dan akan melakukan pencarian dari simpul firstword dan mencari kata dengan f(n) terendah menggunakan kelas WordUtil dengan fungsi findWordsUCS dan disimpan pada PriorityQueue bernama queue. Kata yang didapat juga akan disimpan di current. Program akan melakukan iterasi sampai mendapatkan kata akhir (current == lastword). Setiap iterasi word yang ada pada current akan di *remove* dari queue dan list of words yang digunakan sebagai acuan *dictionary* sehingga tidak akan ada kondisi infinite loop atau node lain yang mengulang pencarian sehingga lebih efisien.

```java
public class UCS {

    public ArrayList<Object> find(String firstword, String lastword, List<String> words) {
        WordUtil util = new WordUtil();
        List<String> wordsCopy = new ArrayList<>(words);
        PriorityQueue<Node> queue = new PriorityQueue<Node>();
        Node node = new Node(firstword, value:0,age:1, parent:null);

        queue.add(node);

        Node result = new Node(word:"", value:0, age:0, parent:null);

        int nodeCount = 0;
        // Algoritma UCS
        Long start = System.currentTimeMillis();
        while (!queue.isEmpty()) {
            Node current = queue.remove();
            wordsCopy.remove(current.getWord());
            nodeCount++;
            if (current.getWord().equals(lastword)) {
                result = current;
                break;
            }
            int val = current.getValue() + 1;
            ArrayList<Node> nextWords = util.findWordsUCS(wordsCopy, current.getWord(), val, current);
            queue.addAll(nextWords);
        }
        if (queue.isEmpty()){
            System.out.println(x:"Tidak ditemukan jalan");
            return null;
        }
```

Gambar 2.4 Kelas UCS I

```java
        Long end = System.currentTimeMillis();
        Long time = end - start;

        System.out.println("Waktu eksekusi: " + time + " ms");
        float seconds = (time / 1000F);
        System.out.println("Waktu eksekusi : " + seconds + " s");
        System.out.println("Jumlah node yang dikunjungi: " + nodeCount);

        // print hasil
        ArrayList<String> resultWords = new ArrayList<>();
        Node temp = result;
        while (temp != null) {
            resultWords.add(temp.getWord());
            temp = temp.getParent();
        }

        for (int i = resultWords.size() - 1; i >= 0; i--) {
            if (i == 0) {
                System.out.print(resultWords.get(i));
            } else {
                System.out.print(resultWords.get(i) + " -> ");
            }
        }

        ArrayList<Object> resultList = new ArrayList<>();
        resultList.add(result);
        resultList.add(time);
        resultList.add(nodeCount);
        return resultList;


    }
}
```

Gambar 2.5 Kelas UCS II

2.3 Kelas GreedyBFS

Kelas GreedyBFS memiliki 1 fungsi yaitu find, fungsi ini akan menerima firstword yang bertipe string, lastword bertipe string, dan words bertipe list of string untuk digunakan sebagai dictionary dan akan menghasilkan ArrayList of object.

Pada implementasinya bobot yang dihitung untuk h(n) merupakan banyak kata yang berbeda dengan kata tujuan (lastword). Sehingga program ini akan terus mencari kata yang paling mirip dengan kata tujuannya

Program akan melakukan membuat simpul firstword dan akan melakukan pencarian dari simpul firstword dan mencari kata dengan h(n) terendah menggunakan kelas WordUtil dengan fungsi findWordGreedyBFS dan disimpan pada PriorityQueue bernama tempQueue. Kata yang didapat juga akan disimpan di current. Program akan melakukan iterasi sampai mendapatkan kata akhir (current == lastword). Setiap iterasi word yang ada pada current akan di *remove* dari tempQueue dan list of words yang digunakan sebagai acuan *dictionary* sehingga tidak akan ada kondisi infinite loop.

Algoritma GreedyBFS tidak bisa *backtrack* sehingga akan lebih sering menemui jalan buntu atau tidak ada solusi.

```
public class GreedyBFS {
    public ArrayList<Object> find(String firstword, String lastword, List<String> words) {
        WordUtil util = new WordUtil();
        List<String> wordsCopy = new ArrayList<>(words);

        Node root = new Node(firstword, value:0, age:1, parent:null); //value pada node pertama tidak digunakan


        // Algoritma Greedy BFS
        int nodeCount = 1;
        Long start = System.currentTimeMillis();
        PriorityQueue<Node> tempQueue = new PriorityQueue<Node>();
        tempQueue = util.findWordGreedyBFS(wordsCopy, firstword, root, lastword);
        Node current = tempQueue.poll();
        nodeCount += tempQueue.size();
        while (!tempQueue.isEmpty() && !current.getWord().equals(lastword)){
            // System.out.println(current.getWord());
            wordsCopy.remove(current.getWord());

            tempQueue = util.findWordGreedyBFS(wordsCopy, current.getWord(), current, lastword);
            current = tempQueue.poll();
            nodeCount += tempQueue.size();
        }
        if (tempQueue.isEmpty()){
            System.out.println(x:"Tidak ditemukan jalan");
            return null;
        }

        Long end = System.currentTimeMillis();
        Long time = end - start;
```

Gambar 2.6 Kelas GreedyBFS I

```
        Long end = System.currentTimeMillis();
        Long time = end - start;

        System.out.println("Waktu eksekusi: " + time + " ms");
        float seconds = (time / 1000F);
        System.out.println("Waktu eksekusi : " + seconds + " s");
        System.out.println("Jumlah node yang dikunjungi: " + nodeCount);

        // print hasil
        ArrayList<String> resultWords = new ArrayList<>();
        Node temp = current;
        while (temp != null) {
            resultWords.add(temp.getWord());
            temp = temp.getParent();
        }

        for (int i = resultWords.size() - 1; i >= 0; i--) {
            if (i == 0) {
                System.out.print(resultWords.get(i));
            } else {
                System.out.print(resultWords.get(i) + " -> ");
            }
        }

        ArrayList<Object> resultList = new ArrayList<>();
        resultList.add(current);
        resultList.add(time);
        resultList.add(nodeCount);
        return resultList;

    }
}
```

Gambar 2.7 Kelas GreedyBFS II

2.4 Kelas AStar

Kelas AStar memiliki 1 fungsi yaitu find, fungsi ini akan menerima firstword yang bertipe string, lastword bertipe string, dan words bertipe list of string untuk digunakan sebagai dictionary dan akan menghasilkan ArrayList of object.

Pada implementasinya bobot yang dihitung untuk g(n) merupakan banyak kata yang berbeda dengan kata tujuan (lastword) ditambah besar jarak yang disini berarti 1. Sehingga program ini akan terus mencari kata yang paling mirip dengan kata tujuannya ditambah dengan jarak antar kata tersebut

Program akan melakukan membuat simpul firstword dan akan melakukan pencarian dari simpul firstword dan mencari kata dengan h(n) terendah menggunakan kelas WordUtil dengan fungsi findWordsAStar dan disimpan pada PriorityQueue bernama queue. Kata yang didapat juga akan disimpan di current. Program akan melakukan iterasi sampai mendapatkan kata akhir (current == lastword). Setiap iterasi word yang ada pada current akan di *remove* dari queue dan list of words yang digunakan sebagai acuan *dictionary* sehingga tidak akan ada kondisi infinite loop. ALgoritma AStar seperti algoritma GreedyBFS dengan tambahan bisa reverse dan menambah value antar kelas

```java
public class AStar {
    public ArrayList<Object> find(String firstword, String lastword, List<String> words){
        WordUtil util = new WordUtil();
        List<String> wordsCopy = new ArrayList<>(words);
        PriorityQueue<Node> queue = new PriorityQueue<Node>();
        Node node = new Node(firstword, value:0, age:1, parent:null);

        queue.add(node);

        Node result = new Node(word:"", value:0, age:0, parent:null);

        int nodeCount = 0;
        // Algoritma A*
        Long start = System.currentTimeMillis();
        while (!queue.isEmpty()) {
            Node current = queue.remove();
            wordsCopy.remove(current.getWord());
            nodeCount++;
            if (current.getWord().equals(lastword)) {
                result = current;
                break;
            }
            int val = current.getValue() + 1;
            ArrayList<Node> nextWords = util.findWordsAStar(wordsCopy, current.getWord(), val, current, lastword);
            queue.addAll(nextWords);
        }
```

Gambar 2.8 Kelas AStar I

```java
        Long end = System.currentTimeMillis();
        Long time = end - start;

        System.out.println("Waktu eksekusi: " + time + " ms");
        float seconds = (time / 1000F);
        System.out.println("Waktu eksekusi : " + seconds + " s");
        System.out.println("Jumlah node yang dikunjungi: " + nodeCount);

        // print hasil
        ArrayList<String> resultWords = new ArrayList<>();
        Node temp = result;
        while (temp != null) {
            resultWords.add(temp.getWord());
            temp = temp.getParent();
        }
        if (resultWords.isEmpty()){
            System.out.println(x:"Tidak ditemukan jalan");
            return null;
        }

        for (int i = resultWords.size() - 1; i >= 0; i--) {
            if (i == 0) {
                System.out.print(resultWords.get(i));
            } else {
                System.out.print(resultWords.get(i) + " -> ");
            }
        }

        ArrayList<Object> resultList = new ArrayList<>();
        resultList.add(result);
        resultList.add(time);
        resultList.add(nodeCount);
        return resultList;
    }
}
```

Gambar 2.9 Kelas AStar II

2.5 Kelas WordUtil

Kelas word util berisi 3 fungsi yaitu findWordsUCS yang mengimplementasikan algoritma UCS dalam me-*expand* sebuah *node*. Fungsi ini akan menerima list of string words

yang menjadi acuan kata dari dictionary, string word yang merupakan kata yang ingin di expand, int value untuk set value setiap kata yang di expand, serta Node parent sebagai parent dari setiap kata yang di expand.

findWordsAStar yang mengimplementasikan algoritma AStar dalam me-*expand* sebuah *node.* ungsi ini akan menerima list of string words yang menjadi acuan kata dari dictionary, string word yang merupakan kata yang ingin di expand, int value untuk set value setiap kata yang di expand, Node parent sebagai parent dari setiap kata yang di expand, serta string lastword untuk menghitung jumlah huruf yang berbeda dengan kata tujuan (lastword).

findWordGreedyBFS yang mengimplementasikan algoritma GreedyBFS dalam me-*expand* sebuah *node.* ungsi ini akan menerima list of string words yang menjadi acuan kata dari dictionary, string word yang merupakan kata yang ingin di expand, int value untuk set value setiap kata yang di expand, Node parent sebagai parent dari setiap kata yang di expand, serta string lastword untuk menghitung jumlah huruf yang berbeda dengan kata tujuan (lastword).

```java
public class WordUtil {
    public ArrayList<Node> findWordsUCS(List<String> words, String word, int value, Node parent) {
        ArrayList<Node> result = new ArrayList<>();
        int len = word.length();
        HashSet<String> wordSet = new HashSet<>(words); // Membuat HashSet untuk pencarian yang lebih cepat
        int age = 0;
        for (int i = 0; i < len; i++) {
            char originalChar = word.charAt(i);
            for (char c = 'a'; c <= 'z'; c++) {
                if (c == originalChar) {
                    continue;
                }
                StringBuilder sb = new StringBuilder(word);
                sb.setCharAt(i, c);
                String newWord = sb.toString();
                if (wordSet.contains(newWord)) {
                    age++;
                    Node node = new Node(newWord, value, age,parent);
                    result.add(node);
                }
            }
        }
        return result;
    }
```

Gambar 2.10 fungsi findWordsUCS

```java
public ArrayList<Node> findWordsAStar(List<String> words, String word, int value, Node parent, String lastword) {
    ArrayList<Node> result = new ArrayList<>();
    int len = word.length();
    HashSet<String> wordSet = new HashSet<>(words); // Membuat HashSet untuk pencarian yang lebih cepat
    int age = 0;
    for (int i = 0; i < len; i++) {
        char originalChar = word.charAt(i);
        for (char c = 'a'; c <= 'z'; c++) {
            if (c == originalChar) {
                continue;
            }
            StringBuilder sb = new StringBuilder(word);
            sb.setCharAt(i, c);
            String newWord = sb.toString();
            if (wordSet.contains(newWord)) {
                age++;
                Node node = new Node(newWord, value, age, parent);
                // find heuristic
                int heuristic = lastword.length() - 1;
                for (int j = 0; j < len; j++) {
                    if (newWord.charAt(j) == lastword.charAt(j)) {
                        heuristic--;
                    }
                }
                node.setValue(heuristic + value);
                result.add(node);
            }
        }
    }
    return result;
}
```

Gambar 2.11 fungsi findWordsAStar

```java
    public PriorityQueue<Node> findWordGreedyBFS(List<String> words, String word, Node parent, String lastword) {
        PriorityQueue<Node> resultQueue = new PriorityQueue<Node>();
        int len = word.length();
        ArrayList<String> wordSet = new ArrayList<String>(words); // Membuat HashSet untuk pencarian yang lebih cepat
        int age = 0;
        for (int i = 0; i < len; i++) {
            char originalChar = word.charAt(i);
            for (char c = 'a'; c <= 'z'; c++) {
                if (c == originalChar) {
                    continue;
                }
                StringBuilder sb = new StringBuilder(word);
                sb.setCharAt(i, c);
                String newWord = sb.toString();
                if (wordSet.contains(newWord)) {
                    age++;
                    Node node = new Node(newWord, value:0, age, parent);
                    // find heuristic
                    int heuristic = lastword.length() - 1;
                    for (int j = 0; j < len; j++) {
                        if (newWord.charAt(j) == lastword.charAt(j)) {
                            heuristic--;
                        }
                    }
                    node.setValue(heuristic);
                    resultQueue.add(node);
                }
            }
        }
        return resultQueue;
    }
}
```

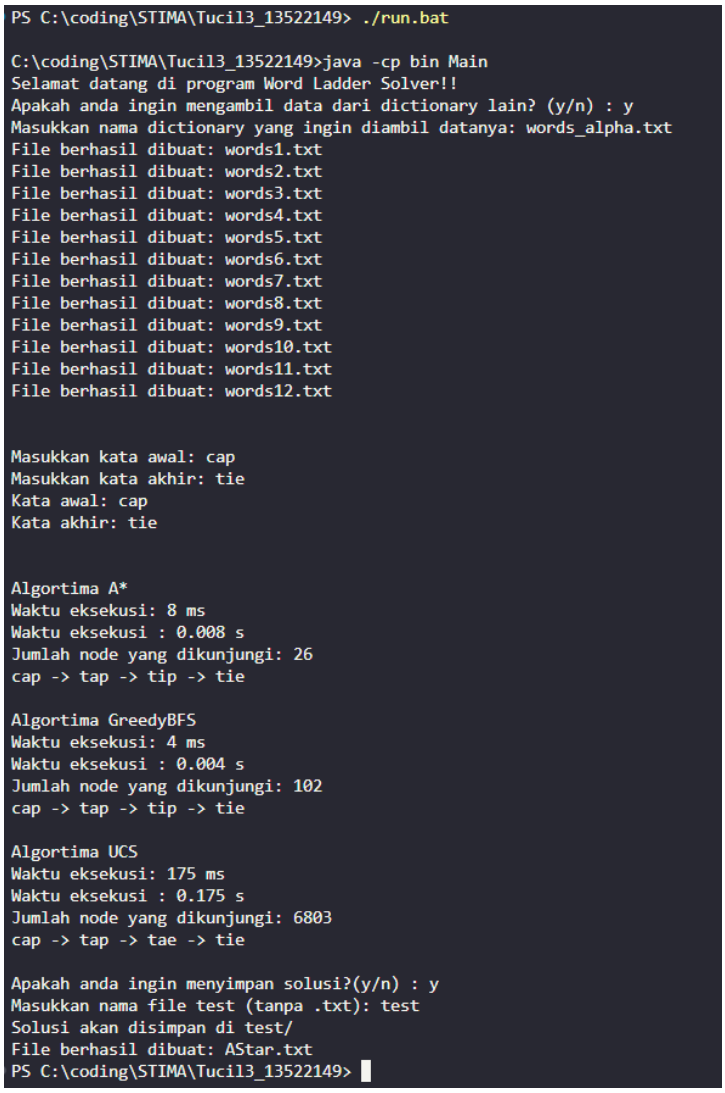
Gambar 2.12 fungsi findWordGreedyBFS

2.6 Kelas Scraping

Kelas ini digunakan untuk me scrape data dictionary menjadi beberapa file txt yang berisi kata kata yang dipisahkan berdasarkan panjang katanya. Sayangnya kelas ini hanya bisa membuat file txt dengan maksimal panjang 12 huruf. File dictionary yang ingin di *scraping* harus berada di **Tucil3_13522149/src/Data/** dan bersifat **txt**

2.7 Kelas Main

Kelas ini merupakan kelas main yang menyatukan seluruh algoritma menjadi sebuah program. Setelah program dijalankan program akan menanyakan apakah ingin mengambil data dari dictionary lain. Jika user menjawab ya maka user disuruh untuk memasukan nama file dictionarinya. Jika user salah memasukan nama maka file yang terbuat akan kosong. Setelah itu program akan menanyakan kata awal dan akhir. Kemudian program akan menampilkan hasilnya. Setelah itu program akan menanyakan apakah user ingin menyimpan filenya? File yang disimpan akan berada di **Tucil3_13522149/test.**



Gambar 2.13 Tampilan CLI

BAB III
Cara Menjalankan Program

3.1 *Requirement*

Program ini dibuat menggunakan bahasa Java 21.0.2. Sehingga sangat disarankan untuk menginstall versi tersebut agar program dapat berjalan sesuai dengan yang diharapkan.

3.2 Cara Menjalankan Program

1. Clone Repository

```
git clone https://github.com/TuanOnta/Tucil3_13522149
```

2. Change directory ke root

```
cd Tucil3_13522149
```

3. Kompilasi program

```
// untuk sistem operasi Windows
./compile.bat
// untuk sistem operasi Linux atau WSL
./compile.sh
```

4. Menjalankan program

```
// untuk sistem operasi Windows
./run.bat
// untuk sistem operasi Linux atau WSL
./run.sh
```

BAB IV
Uji Coba

4.1 TestCase 1

```
Masukkan kata awal: earn
Masukkan kata akhir: make
Kata awal: earn
Kata akhir: make


Algortima A*
Waktu eksekusi: 51 ms
Waktu eksekusi : 0.051 s
Jumlah node yang dikunjungi: 160
Memori yang digunakan: 1720216 bytes
earn -> barn -> bare -> bake -> make

Algortima GreedyBFS
Waktu eksekusi: 11 ms
Waktu eksekusi : 0.011 s
Jumlah node yang dikunjungi: 117
Memori yang digunakan: 1534264 bytes
earn -> barn -> bare -> mare -> make

Algortima UCS
Waktu eksekusi: 1287 ms
Waktu eksekusi : 1.287 s
Jumlah node yang dikunjungi: 9124
Memori yang digunakan: 6710616 bytes
earn -> larn -> lare -> lake -> make
```

Gambar 4.1 Test case 1

4.2 Test Case 2

```
Algortima A*
Waktu eksekusi: 35 ms
Waktu eksekusi : 0.035 s
Jumlah node yang dikunjungi: 101
Memori yang digunakan: 1608064 bytes
rest -> reit -> roit -> root -> room

Algortima GreedyBFS
Waktu eksekusi: 11 ms
Waktu eksekusi : 0.011 s
Jumlah node yang dikunjungi: 70
Memori yang digunakan: 1532704 bytes
rest -> rist -> riot -> root -> room

Algortima UCS
Waktu eksekusi: 2560 ms
Waktu eksekusi : 2.56 s
Jumlah node yang dikunjungi: 25790
Memori yang digunakan: 10918416 bytes
rest -> reit -> roit -> root -> room
```

Gambar 4.2 Test case 2

4.3 Test Case 3

```
Masukkan kata awal: jack
Masukkan kata akhir: jill
Kata awal: jack
Kata akhir: jill


Algortima A*
Waktu eksekusi: 70 ms
Waktu eksekusi : 0.07 s
Jumlah node yang dikunjungi: 310
Memori yang digunakan: 1766640 bytes
jack -> jark -> jarl -> jerl -> jell -> jill

Algortima GreedyBFS
Waktu eksekusi: 36 ms
Waktu eksekusi : 0.036 s
Jumlah node yang dikunjungi: 231
Memori yang digunakan: 1534768 bytes
jack -> jock -> jack -> juck -> junk -> jink -> jina -> jiva -> jive -> jibe -> jibb -> jibi -> jiri -> jiti -> biti -> liti -> siti -> titi -> viti -> vili -> vill -> jill

Algortima UCS
Waktu eksekusi: 3577 ms
Waktu eksekusi : 3.577 s
Jumlah node yang dikunjungi: 62649
Memori yang digunakan: 1506232 bytes
jack -> fack -> falk -> fall -> fill -> jill
```

Gambar 4.3 Test case 3

4.4 Test Case 4

```
Masukkan kata awal: drum
Masukkan kata akhir: beat
Kata awal: drum
Kata akhir: beat


Algortima A*
Waktu eksekusi: 12 ms
Waktu eksekusi : 0.012 s
Jumlah node yang dikunjungi: 21
Memori yang digunakan: 1534680 bytes
drum -> dram -> drat -> brat -> beat

Algortima GreedyBFS
Waktu eksekusi: 13 ms
Waktu eksekusi : 0.013 s
Jumlah node yang dikunjungi: 44
Memori yang digunakan: 1531616 bytes
drum -> brum -> bram -> beam -> beat

Algortima UCS
Waktu eksekusi: 120 ms
Waktu eksekusi : 0.12 s
Jumlah node yang dikunjungi: 871
Memori yang digunakan: 2023768 bytes
drum -> brum -> brut -> brat -> beat
```

Gambar 4.4 Test case 4

4.5 Test Case 5

```
Masukkan kata awal: lute
Masukkan kata akhir: harp
Kata awal: lute
Kata akhir: harp


Algortima A*
Waktu eksekusi: 28 ms
Waktu eksekusi : 0.028 s
Jumlah node yang dikunjungi: 70
Memori yang digunakan: 1623672 bytes
lute -> late -> hate -> hare -> harp

Algortima GreedyBFS
Waktu eksekusi: 11 ms
Waktu eksekusi : 0.011 s
Jumlah node yang dikunjungi: 116
Memori yang digunakan: 1533896 bytes
lute -> late -> hate -> hare -> harp

Algortima UCS
Waktu eksekusi: 1853 ms
Waktu eksekusi : 1.853 s
Jumlah node yang dikunjungi: 24032
Memori yang digunakan: 10157984 bytes
lute -> late -> hate -> hare -> harp
```

Gambar 4.5 Test Case 5

4.6 Test Case 6

```
Masukkan kata awal: pile
Masukkan kata akhir: heap
Kata awal: pile
Kata akhir: heap


Algortima A*
Waktu eksekusi: 22 ms
Waktu eksekusi : 0.022 s
Jumlah node yang dikunjungi: 53
Memori yang digunakan: 1596328 bytes
pile -> pele -> hele -> help -> heap

Algortima GreedyBFS
Waktu eksekusi: 12 ms
Waktu eksekusi : 0.012 s
Jumlah node yang dikunjungi: 84
Memori yang digunakan: 1531656 bytes
pile -> hile -> hele -> help -> heap

Algortima UCS
Waktu eksekusi: 1560 ms
Waktu eksekusi : 1.56 s
Jumlah node yang dikunjungi: 17046
Memori yang digunakan: 8532416 bytes
pile -> pele -> hele -> help -> heap
```

Gambar 4.6 Test case 6

Bab V
Analisis Hasil Percobaan dan Kesimpulan

Dari hasil yang didapat dapat dilihat bahwa panjang *path* yang dihasilkan dari algoritma Uniform Cost Search dan A* memiliki panjang yang sama. Sedangkan hasil yang didapat dari Greedy Best First Search cenderung lebih panjang. Hal ini sesuai dengan teori yaitu algoritma Uniform Cost Search dan A* dapat menghasilkan solusi yang optima.

Berdasarkan jumlah simpul yang dikunjungi, algoritma Greedy Best First Search cenderung menang dengan jumlah paling sedikit, diikuti dengan A*, dan terakhir Uniform Cost Search. Hal serupa juga berlaku berbanding lurus terhadap memori yang dipakai dan waktu eksekusi yang diperlukan. Hal tersebut dapat terjadi karena algoritma Greedy Best First Search hanya memilih simpul berikutnya berdasarkan local optima yaitu simpul berikutnya yang paling murah, tanpa memperdulikan dampak sesudahnya sehingga simpul yang dijelajah lebih sedikit. Berbeda dengan algoritma Uniform Cost Search dan A* yang perlu mempertimbangkan seluruh jalur dengan fungsi heuristiknya masing-masing. Di antara Uniform Cost Search dan A*, algoritma yang kedua lebih unggul karena memiliki fungsi heuristik yang lebih baik sehingga dapat dengan lebih cepat menemukan solusi yang optimal.

Kesimpulan yang dapat diambil adalah algoritma A* merupakan algoritma yang paling baik. Jika ingin menemukan solusi yang lebih cepat dapat menggunakan algoritma Greedy Best First Search tetapi terdapat kemungkinan tidak mendapat solusi.

Referensi

KANTINIT (2023). Uniform Cost Search: Cara Kerja dan Kelebihannya. Diakses pada 6 Mei 2024 dari https://kantinit.com/algoritma/uniform-cost-search-cara-kerja-dan-kelebihannya/#A_Konsep_dasar_Uniform_Cost_Search

Kumar, Sahil & Yang, Christine (2023). Greedy Best-First Search. Diakses pada 6 Mei 2024 dari https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search

javatpoint.com. A* Search Algorithm in Artificial Intelligence. Diakses pada 7 Mei 2024 dari https://www.javatpoint.com/ai-informed-search-algorithms

Maulidevi, Nur Ulfa. Penentuan Rute (Route/Path Planning) bagian 1. Diakses pada 7 Mei 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf

Maulidevi, Nur Ulfa. Penentuan Rute (Route/Path Planning) bagian 1. Diakses pada 5 Mei 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

LAMPIRAN

Pranala Repository GitHub

https://github.com/TuanOnta/Tucil3_13522149

Pranala kamus yang digunakan

https://github.com/dwyl/english-words/blob/master/words_alpha.txt

Check Box

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