LAPORAN TUGAS KECIL I
IF2211 STRATEGI ALGORITMA

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

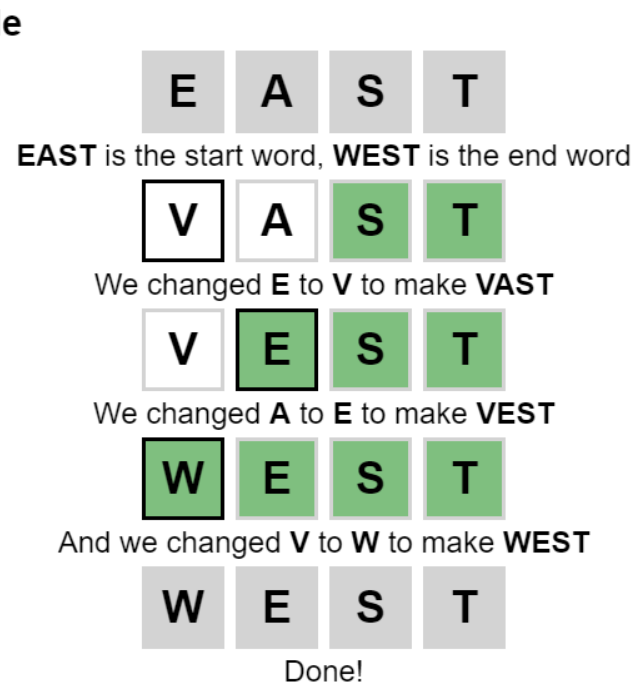
Disusun oleh:
Imam Hanif Mulyarahman 13522030

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Word Ladder adalah salah satu permainan kata yang ditemukan oleh Lewis Carroll pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. 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 mendapat solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah contoh ilustrasi permainan.



Gambar 1. Ilustrasi permainan World Ladder

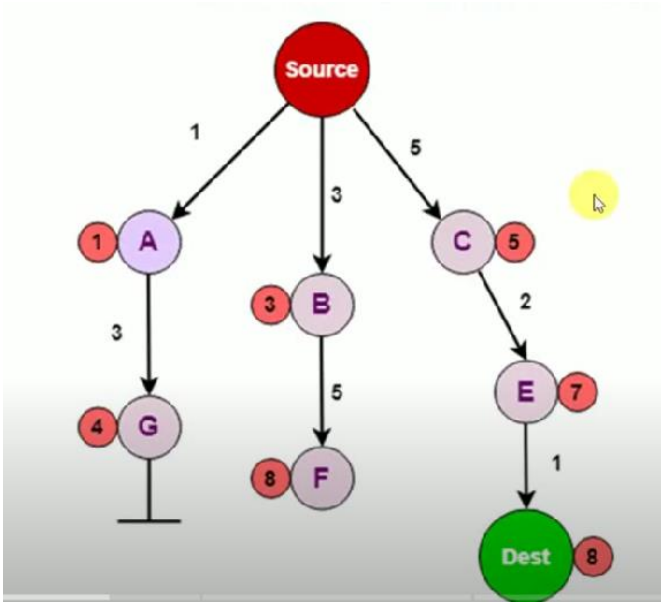
Pada tugas kecil 3 ini, akan dibuat program untuk menyelesaikan permainan world ladder dengan algoritma Uniform Cost Search (UCS), Greedy Best First Search dan A* dalam bahasa Java.

BAB II
LANDASAN TEORI

2.1. Dasar Teori

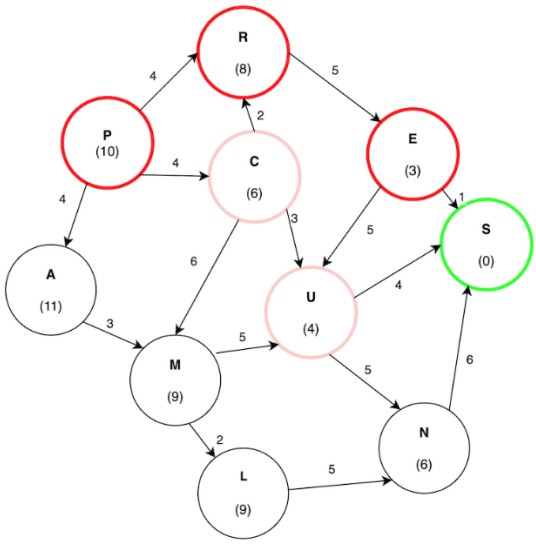
2.1.1. Algoritma Uniform Cost Search (UCS)

Algoritma Uniform Cost Search adalah algoritma yang digunakan untuk mencari rute antara dua simpul pada graf berbobot. Pencarian dilakukan dengan cara menelurusi simpul-simpul dengan biaya yang paling rendah. Pencarian menggunakan algoritma ini akan menghasilkan jalur dengan biaya yang optimal.



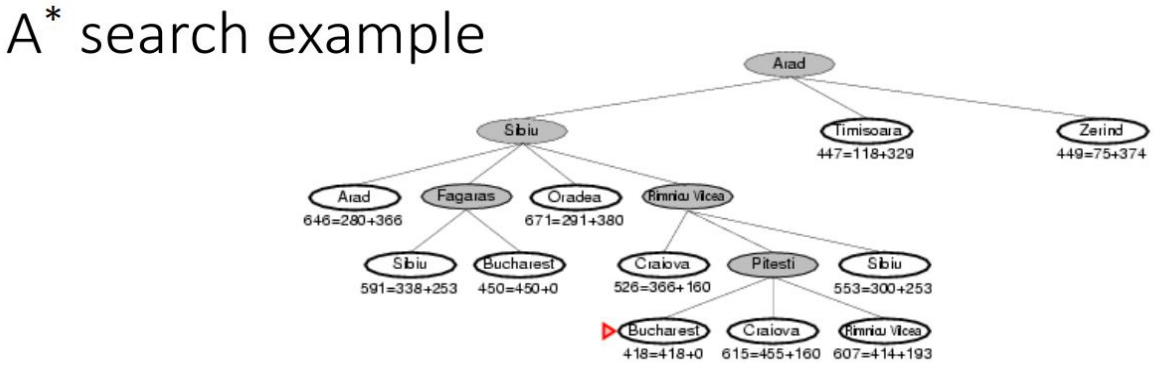
2.1.2. Algoritma Greedy Best First Search

Algoritma Greedy Best First Search adalah algoritma pencarian yang menggunakan nilai fungsi heuristik sebagai pembanding. Algoritma ini mengabaikan bobot pada graf dan hanya menekankan pada perhitungan fungsi heuristik. Algoritma ini belum tentu menghasilkan rute yang optimal secara global karena dapat terjebak pada pengejaran nilai optimal lokal.



2.1.3. Algoritma A*

Algoritma A* adalah algoritma pencarian lintasan yang digunakan untuk menemukan jalur terpendek dari simpul awal ke simpul tujuan dalam graf berbobot. A* menggabungkan teknik dari Uniform Cost Search dengan pendekatan heuristik, yang memungkinkannya untuk menemukan jalur yang optimal lebih efisien daripada Uniform Cost Search, khususnya dalam situasi di mana ruang pencarian besar.



BAB III
ANALISIS PEMECAHAN MASALAH

3.1. Langkah Pemecahan Masalah

Pada awalnya dibuat sebuah kamus bertipe hash set yang berisi semua kata-kata yang dianggap valid oleh program ini. Selain itu, dibuatlah struktur data word yang berisi kata yang bertipe string, cost yang berisi biaya yang bertipe integer, dan prev yang berisi lintasan dari start ke kata saat ini yang bertipe array of string. Selanjutnya permasalahan akan diselesaikan berdasarkan algoritma pencarian yang dipilih. Terdapat tiga algoritma yang tersedia yaitu, Uniform Cost Search, Greedy Best First Search, dan A*. Ketiga algoritma ini memiliki nilai cost yang berbeda-beda. Nilai cost atau nilai f(n) merupakan nilai estimasi total biaya dari simpul aeal ke simpul akhir. Nilai f(n) merupakan hasil penjumlahan dari g(n), yaitu jumlah langkah selama ini yang telah diambil untuk mencapai n, dengan h(n), yaitu estimasi langkah yang diperlukan untuk mencapai end word dari n. Algoritma Uniform Cost Search memiliki nilai cost f(n) = g(n), Algoritma Greedy Best First Search memiliki nilai cost f(n) = h(n), Sedangkan Algoritma A* memiliki nilai cost f(n) = g(n) + h(n). Berikut adalah langkah-langkah penyelesaian setiap algoritma.

3.1.1. Uniform Cost Search (UCS)

1. Pada UCS dibuatlah sebuah Priority Queue yang berisi word dan nilai cost sebagai pembandingnya. Selain itu, dibuat juga sebuah hash set yang berisi kata-kata yang sudah diekspan oleh algoritma.
2. Masukkan start word pada priority queue.
3. Ekspansi elemen pertama priority Queue dengan cara menambahkan semua kata valid yang didapatkan dengan cara mengubah 1 huruf pada elemen tersebut ke dalam priority queue. Ubah nilai costnya sesuai dengan panjang jalur yang sudah dilaluinya atau f(n) = g(n).
4. Lakukan langkah ketiga sampai elemen pertama priority Queue sama dengan end atau priority queue kosong.
5. Jika ditemukan end word dari start word, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.

6. Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari start word dan berakhir di end word. Program akan berakhir dan mengeluarkan pesan "Algoritma Uniform Cost Search (UCS) tidak menemukan jalan dari <start word> ke <end word>".

Pada Kasus permainan Word Ladder urutan simpul pada queue sesuai dengan nilai g(n), yaitu jumlah langkah dari simpul awal ke simpul tersebut. karena setiap simpul yang bertetangga hanya berjarak satu, maka pencarian solusi menggunakan UCS akan sama dengan BFS. Hal ini disebabkan karena bobot dari jarak setiap simpul yang bertetangga sama sehingga pencarian solusi ini akan menghasilkan alur kerja yang sama dengan BFS.

3.1.2. Greedy Best First Search

1. Pada Greedy Best First Search dibuatlah sebuah Priority Queue yang berisi word dan nilai cost sebagai pembandingnya. Selain itu, dibuat juga sebuah hash set yang berisi kata-kata yang sudah diekspan oleh algoritma.
2. Masukkan start word pada priority queue.
3. Ekspansi elemen pertama priority Queue dengan cara menambahkan maksimal satu kata valid dengan nilai heuristik terendah yang didapatkan dengan cara mengubah 1 huruf pada elemen tersebut ke dalam priority queue. Ubah nilai costnya sesuai dengan jumlah perbedaan huruf dengan end word (nilai heuristik) atau f(n) = h(n).
4. Lakukan langkah ketiga sampai elemen pertama priority Queue sama dengan end atau priority queue kosong.
5. Jika ditemukan end word dari start word, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.
6. Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari start word dan berakhir di end word. Program akan berakhir dan mengeluarkan pesan "Algoritma Greedy Best First Search tidak menemukan jalan dari <start word> ke <end word>".

Algoritma Greedy Best First Search tidak menjamin akan mendapatkan solusi optimal. Hal ini dapat disebabkan oleh beberapa faktor. Faktor pertama karena algoritma ini hanya mengandalkan nilai heuristik yang merupakan nilai estimasi

tanpa mempertimbangkan langkah yang sudah ditempuh. Faktor lainnya karena algoritma ini hanya menelusuri satu jalur dengan heuristik terkecil tanpa adanya backtracking yang belum tentu menghasilkan solusi yang optimal secara global.

3.1.3. A*

1. Pada A* dibuatlah sebuah Priority Queue yang berisi word dan nilai cost sebagai pembandingnya. Selain itu, dibuat juga sebuah hash set yang berisi kata-kata yang sudah diekspan oleh algoritma.
2. Masukkan start word pada priority queue.
3. Ekspansi elemen pertama priority Queue dengan cara menambahkan semua kata valid yang didapatkan dengan cara mengubah 1 huruf pada elemen tersebut ke dalam priority queue. Ubah nilai costnya sesuai dengan panjang jalur yang sudah dilaluinya ditambah jumlah perbedaan huruf dengan end word atau f(n) = g(n) + h(n).
4. Lakukan langkah ketiga sampai elemen pertama priority Queue sama dengan end atau priority queue kosong.
5. Jika ditemukan end word dari start word, program akan mengeluarkan rute yang diambil, jumlah node yang dikunjungi, dan waktu eksekusi program.
6. Jika queue kosong, hal ini berarti bahwa tidak ada rute yang berawal dari start word dan berakhir di end word. Program akan berakhir dan mengeluarkan pesan “Algoritma A* tidak menemukan jalan dari <start word> ke <end word>”.

Alur program A* mirip dengan alur program pada UCS. Perbedaannya hanya terletak pada nilai cost yang dihasilkan. Nilai cost pada UCS adalah f(n) = g(n) sedangkan pada A* adalah f(n) = g(n) + h(n). Nilai heuteristik pada program ini admissible karena tidak pernah meng-overestimate dari nilai yang sesungguhnya. Jumlah perbedaan huruf suatu kata dengan end word selalu lebih kecil atau sama dengan jumlah langkah sebenarnya yang harus dilakukan untuk mencapai end word dari kata tersebut. Karena nilai h(n) yang admissible, maka bisa dipastikan bahwa algoritma A* akan mendapatkan solusi yang optimal.

Secara teoritis, algoritma A* lebih efisien dibandingkan dengan algoritma UCS. Pada algoritma UCS hanya menggunakan nilai jumlah langkah dari simpul awal untuk menentukan jalur. Sedangakan algoritma A* menggunakan bantuan nilai

heuristik untuk memilih simpul yang akan diekspansi berikutnya. Nilai h(n) akan membantu A* memilih simpul yang lebih baik yang mengakibatkan jumlah simpul yang diekspansi menjadi lebih sedikit dibandingkan dengan UCS.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Program

Terdapat 6 Kelas yang dibuat untuk program ini yaitu Word, BacaKamus, UCS, GreedyBFS, AStar, dan Main.

4.1.1. Word

Kelas Word berisi informasi kata, biaya, dan path yang dilalui dari start ke kata tersebut.

Metode	Deskripsi
public Word (String word, int cost, ArrayList<String> prev)	Konstuktor kelas Word
public String getWord()	Mengembalikan kata saat ini
public int getCost()	Mengembalikan nilai cost
public ArrayList<String> getPrev()	Mengembalikan path dari start ke kata sebelum kata ini
public void displayPath()	Menuliskan path dari start ke kata saat ini
public boolean isValid (BacaKamus kamus)	Mengembalikan true apabila Word merupakan kata yang berada di kamus yamg tersedia

Word.java

```
import java.util.ArrayList;

class Word {
  private String word;
  private int cost;
  private ArrayList<String> prev;

  public Word(String word, int cost, ArrayList<String> prev){
    this.word = word;
    this.cost = cost;
    this.prev = prev;
```

```
  }

  public int getCost() {
    return cost;
  }

  public ArrayList<String> getPrev() {
    return prev;
  }

  public String getWord() {
    return word;
  }

  public void addPrev(String kata){
    prev.add(prev.size(), kata);
  }

  public void displayPath(){
    for (int i = 0; i < prev.size(); i++) {
        System.out.print(prev.get(i)+ " -> ");
    }
    System.out.println(word);
  }

  public boolean isValid(BacaKamus kamus){
    return kamus.getKamus().contains(word);
  }
}
```

4.1.2. BacaKamus

BacaKamus adalah kelas yang menangani pembacaan file kamus.txt dan menyediakan hash set berisi daftar kata yang valid.

Metode	Deskripsi
public BacaKamus()	Konstuktor kelas BacaKamus. Mengisi kamus dengan kata yang tersedia pada kamus.txt

public String getKamus()	Mengembalikan kamus

BacaKamus.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;

class BacaKamus {
    private HashSet<String> kamus;

    public BacaKamus(){
        String fileName = "kamus.txt";
        HashSet<String> lines = new HashSet<>(); // Array untuk menyimpan setiap
baris

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                lines.add(line); // Tambahkan setiap baris ke dalam array
            }
        } catch (IOException e) {
            System.err.println("Error reading the file: " + e.getMessage());
        }
        kamus = lines;
    }

    public HashSet<String> getKamus() {
        return kamus;
    }
}
```

4.1.3. UCS

UCS adalah kelas yang melakukan algoritma Uniform cost search. Kelas ini berisi tipe data priority queue yang menangani pencarian path dan hash set untuk menyimpan simpul yang sudah diekspan.

Metode	Deskripsi
public UCS()	Konstuktor kelas UCS.

public int calculateCost(Word word)	Menghitung biaya pada UCS.
public void addQueue (Word word)	Menambahkan anggota priority queue.
public void ubahKata (Word word, BacaKamus kamus)	Mencari kumpulan kata selanjutnya dan menambahkannya ke priority queue.
public static void mainUCS (String awal, String goal, BacaKamus kamus)	Fungsi main yang menjalankan algoritma UCS.

UCS.java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

public class UCS {
  private PriorityQueue<Word> prioQueue;
  private HashSet<String> simpulEkspan;

  public UCS(){
    prioQueue = new PriorityQueue<Word>(Comparator.comparing(Word::getCost));
    simpulEkspan = new HashSet<String>();
  }

  public int calculateCost(Word word){
    return word.getCost()+1;
  }

  public void addQueue(Word word){
    prioQueue.add(word);
  }

  public void ubahKata(Word word, BacaKamus kamus){
    simpulEkspan.add(word.getWord());
    ArrayList<String> tempprev = new ArrayList<>(word.getPrev());
    tempprev.add(word.getWord());
    for (int i = 0; i < word.getWord().length(); i++) {
      for (char c = 'a'; c <= 'z'; c++) {
        StringBuilder tempbuilder = new StringBuilder(word.getWord());
        tempbuilder.setCharAt(i, c);
        Word temp = new Word(tempbuilder.toString(), calculateCost(word),
tempprev);
```

```
        if (temp.isValid(kamus) &&
!simpulEkspan.contains(tempbuilder.toString())) {
          addQueue(temp);
        }
      }
    }
  }

  public static void mainUCS(String awal, String goal, BacaKamus kamus){
    boolean isKetemu = false;
    Word start = new Word(awal, 0, new ArrayList<String>());
    Word end = new Word(goal, 0, new ArrayList<String>());
    UCS ucs = new UCS();
    ucs.addQueue(start);
    while (!isKetemu && !ucs.prioQueue.isEmpty()) {
        start = ucs.prioQueue.poll();
        ucs.ubahKata(start, kamus);
        if (start.getWord().equals(end.getWord())) {
          isKetemu = true;
        }
    }
    if (isKetemu) {
      start.displayPath();
      System.out.println("Panjang Path : " + start.getPrev().size());
    } else {
        System.out.println("Algoritma Uniform cost Search tidak menemukan jalan
dari " + awal + " ke " + goal + ".");
    }
    System.out.println("Jumlah node yang diperiksa : " +
ucs.simpulEkspan.size());
  }
}
```

4.1.4. GreedyBFS

GreedyBFS adalah kelas yang melakukan algoritma Greedy Best First search. Kelas ini berisi tipe data priority queue yang menangani pencarian path dan hash set untuk menyimpan simpul yang sudah diekspan.

Metode	Deskripsi
public GreedyBFS()	Konstuktor kelas GreedyBFS.

public int calculateHeuristik (String now, Word goal)	Menghitung fungsi heuristik pada GreedyBFS.
public void addQueue (Word word)	Menambahkan anggota priority queue.
public void ubahKata (Word word, BacaKamus kamus)	Mencari kata dengan nilai heuristik terendah dan menambahkannya ke priority queue.
public static void mainGreedyBFS (String awal, String goal, BacaKamus kamus)	Fungsi main yang menjalankan algoritma Greedy Best First Search.

GreedyBFS.java

```java
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

public class GreedyBFS {
  private PriorityQueue<Word> prioQueue;
  private HashSet<String> simpulEkspan;

  public GreedyBFS(){
    prioQueue = new PriorityQueue<Word>(Comparator.comparing(Word::getCost));
    simpulEkspan = new HashSet<String>();
  }

  public int calculateHeuristik(String next , Word goal){
    char[] tempNext = next.toCharArray();
    char[] tempGoal = goal.getWord().toCharArray();
    int count = 0;
    for (int i = 0; i < tempNext.length; i++) {
        if (tempNext[i] != tempGoal[i]) {
            count++;
        }
    }
    return count;
  }

  public void addQueue(Word word){
    prioQueue.add(word);
  }
```

```
  public void ubahKata(Word word, Word goal, BacaKamus kamus){
    simpulEkspan.add(word.getWord());
    ArrayList<String> tempprev = new ArrayList<>(word.getPrev());
    tempprev.add(word.getWord());
    Word kandidatGBFS = null;
    for (int i = 0; i < word.getWord().length(); i++) {
      for (char c = 'a'; c <= 'z'; c++) {
        StringBuilder tempbuilder = new StringBuilder(word.getWord());
        tempbuilder.setCharAt(i, c);
        Word temp = new Word(tempbuilder.toString(),
calculateHeuristik(tempbuilder.toString(), goal), tempprev);
        if (temp.isValid(kamus) &&
!simpulEkspan.contains(tempbuilder.toString())) {
          if (kandidatGBFS == null) {
            kandidatGBFS = temp;
          } else {
            if (kandidatGBFS.getCost() > temp.getCost()) {
                kandidatGBFS = temp;
            }
          }
        }
      }
    }
    if (kandidatGBFS != null) {
        addQueue(kandidatGBFS);
    }
  }

  public static void mainGreedyBFS(String awal, String goal, BacaKamus kamus){
    boolean isKetemu = false;
    Word start = new Word(awal, 0, new ArrayList<String>());
    Word end = new Word(goal, 0, new ArrayList<String>());
    GreedyBFS greedyBFS = new GreedyBFS();
    greedyBFS.addQueue(start);
    while (!isKetemu && !greedyBFS.prioQueue.isEmpty()) {
        start = greedyBFS.prioQueue.poll();
        greedyBFS.ubahKata(start, end, kamus);
        if (start.getWord().equals(end.getWord())) {
          isKetemu = true;
        }
    }
    if (isKetemu) {
        start.displayPath();
        System.out.println("Panjang Path : " + start.getPrev().size());
    } else {
```

```
        System.out.println("Algoritma Greedy Best First Search tidak menemukan
jalan dari " + awal + " ke " + goal + ".");
    }
    System.out.println("Jumlah node yang diperiksa : " +
greedyBFS.simpulEkspan.size());
  }
}
```

4.1.5. AStar

AStar adalah kelas yang melakukan algoritma A*. Kelas ini berisi tipe data priority queue yang menangani pencarian path dan hash set untuk menyimpan simpul yang sudah diekspan.

Metode	Deskripsi
public AStar()	Konstuktor kelas AStar.
public int calculateEstimedCost (Word now, String next, Word goal)	Menghitung nilai f(n) = g(n) + h(n) untuk algoritma A*
public void addQueue (Word word)	Menambahkan anggota priority queue.
public void ubahKata (Word word, BacaKamus kamus)	Mencari kumpulan kata selanjutnya dan menambahkannya ke priority queue.
public static void mainA*a (String awal, String goal, BacaKamus kamus)	Fungsi main yang menjalankan algoritma Greedy Best First Search.

AStar.java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.PriorityQueue;

public class AStar {
  private PriorityQueue<Word> prioQueue;
  private HashSet<String> simpulEkspan;

  public AStar(){
    prioQueue = new PriorityQueue<Word>(Comparator.comparing(Word::getCost));
    simpulEkspan = new HashSet<String>();
  }
```

```
  public int calculateEstimedCost(Word now, String next, Word goal){
    char[] tempNext = next.toCharArray();
    char[] tempGoal = goal.getWord().toCharArray();
    int count = 0;
    for (int i = 0; i < tempNext.length; i++) {
        if (tempNext[i] != tempGoal[i]) {
            count++;
        }
    }
    return now.getCost() + 1 + count;
  }

  public void addQueue(Word word){
    prioQueue.add(word);
  }

  public void ubahKata(Word word, Word goal, BacaKamus kamus){
    simpulEkspan.add(word.getWord());
    ArrayList<String> tempprev = new ArrayList<>(word.getPrev());
    tempprev.add(word.getWord());
    for (int i = 0; i < word.getWord().length(); i++) {
      for (char c = 'a'; c <= 'z'; c++) {
        StringBuilder tempbuilder = new StringBuilder(word.getWord());
        tempbuilder.setCharAt(i, c);
        Word temp = new Word(tempbuilder.toString(), calculateEstimedCost(word,
tempbuilder.toString(), goal), tempprev);
        if (temp.isValid(kamus) &&
!simpulEkspan.contains(tempbuilder.toString())) {
          addQueue(temp);
        }
      }
    }
  }

  public static void mainAStar(String awal, String goal, BacaKamus kamus){
    boolean isKetemu = false;
    Word start = new Word(awal, 0, new ArrayList<String>());
    Word end = new Word(goal, 0, new ArrayList<String>());
    AStar aStar = new AStar();
    aStar.addQueue(start);
    while (!isKetemu && !aStar.prioQueue.isEmpty()) {
        start = aStar.prioQueue.poll();
        aStar.ubahKata(start, end, kamus);
        if (start.getWord().equals(end.getWord())) {
          isKetemu = true;
```

```
        }
    }
    if (isKetemu) {
      start.displayPath();
      System.out.println("Panjang Path : " + start.getPrev().size());
    } else {
        System.out.println("Algoritma A* tidak menemukan jalan dari " + awal + "
ke " + goal + ".");
    }
    System.out.println("Jumlah node yang diperiksa : " +
aStar.simpulEkspan.size());
  }
}
```

4.1.6. Main

Main adalah kelas untuk menjalankan program utama.

Metode	Deskripsi
public static void main(String[] args)	Fungsi utama untuk menjalankan program.

Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        BacaKamus kamus = new BacaKamus();
        Scanner sc = new Scanner(System.in);
        String start;
        String goal;
        do {
            do {
                System.out.println("Masukan kata awal :");
                System.out.print(">> ");
                start = sc.nextLine().toLowerCase();
                if (!kamus.getKamus().contains(start)) {
                    System.out.println("Kata masukan tidak ada dalam kamus");
                }

            } while (!kamus.getKamus().contains(start));

            do {
                System.out.println("Masukan kata tujuan :");
```

```
                System.out.print(">> ");
                goal = sc.nextLine().toLowerCase();
                if (!kamus.getKamus().contains(goal)) {
                    System.out.println("Kata masukan tidak ada dalam kamus");
                }

            } while (!kamus.getKamus().contains(goal));

            if (start.length() != goal.length()) {
                System.out.println("Kata awal dan akhir tidak memiliki panjang
yang sama\n");
            }

        } while (start.length() != goal.length());

        String pilihan;
        long startTime;
        long endTime;
        do {
            System.out.println("Pilihlah Algoritma yang diinginkan : ");
            System.out.println("1. Algoritma Uniform Search Cost (UCS)");
            System.out.println("2. Algoritma Greedy Best First Search (Greedy
BFS)");
            System.out.println("3. Algoritma A*");

            System.out.print(">> ");
            pilihan = sc.nextLine();
            System.out.println();
            startTime = System.currentTimeMillis();
            if (pilihan.equals("1")) {
                UCS.mainUCS(start, goal, kamus);
            } else if (pilihan.equals("2")) {
                GreedyBFS.mainGreedyBFS(start, goal, kamus);
            } else if (pilihan.equals("3")) {
                AStar.mainAStar(start, goal, kamus);
            } else {
                System.out.println("Pilihan tidak Valid");
            }

        } while (pilihan.equals("1") &&  pilihan.equals("2") &&
pilihan.equals("3"));

    endTime = System.currentTimeMillis();
    System.out.println("Runtime : " + (endTime-startTime) + " ms\n");
    sc.close();
```

```
    }
}
```

4.2. Hasil Pengujian

Terdapat enam testcase yang akan diujikan para program kali ini. Berikut rincian testcasenya.

Masukan	Keluaran

Start Word : Earn
End Word : Make
	UCS
	`Masukan kata awal :`
	`>> earn`
	`Masukan kata tujuan :`
	`>> make`
	`Pilihlah Algoritma yang diinginkan :`
	`1. Algoritma Uniform Search Cost (UCS)`
	`2. Algoritma Greedy Best First Search (Greedy BFS)`
	`3. Algoritma A*`
	`>> 1`
	
	`earn -> earl -> marl -> mare -> make`
	`Panjang Path : 4`
	`Jumlah node yang diperiksa : 645`
	`Runtime : 120 ms`
	Greedy BFS
	`Masukan kata awal :`
	`>> earn`
	`Masukan kata tujuan :`
	`>> make`
	`Pilihlah Algoritma yang diinginkan :`
	`1. Algoritma Uniform Search Cost (UCS)`
	`2. Algoritma Greedy Best First Search (Greedy BFS)`
	`3. Algoritma A*`
	`>> 2`
	
	`earn -> barn -> bare -> mare -> make`
	`Panjang Path : 4`
	`Jumlah node yang diperiksa : 5`
	`Runtime : 25 ms`
	A*

	Masukan kata awal : >> earn Masukan kata tujuan : >> make Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 3 earn -> darn -> dare -> mare -> make Panjang Path : 4 Jumlah node yang diperiksa : 86 Runtime : 65 ms
Start Word : Sun End Word : Bad	UCS Masukan kata awal : >> sun Masukan kata tujuan : >> bad Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 1 sun -> bun -> ban -> bad Panjang Path : 3 Jumlah node yang diperiksa : 358 Runtime : 74 ms Greedy BFS Masukan kata awal : >> sun Masukan kata tujuan : >> bad Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 2 sun -> bun -> ban -> bad Panjang Path : 3 Jumlah node yang diperiksa : 4 Runtime : 23 ms A*

<table>
<tr><td></td><td>

```
Masukan kata awal :
>> sun
Masukan kata tujuan :
>> bad
Pilihlah Algoritma yang diinginkan :
1. Algoritma Uniform Search Cost (UCS)
2. Algoritma Greedy Best First Search (Greedy BFS)
3. Algoritma A*
>> 3

sun -> bun -> bud -> bad
Panjang Path : 3
Jumlah node yang diperiksa : 27
Runtime : 32 ms
```

</td></tr>
<tr><td>Start Word : Frown
End Word : Smile</td><td>UCS

```
Masukan kata awal :
>> frown
Masukan kata tujuan :
>> smile
Pilihlah Algoritma yang diinginkan :
1. Algoritma Uniform Search Cost (UCS)
2. Algoritma Greedy Best First Search (Greedy BFS)
3. Algoritma A*
>> 1

frown -> frows -> flows -> slows -> slots -> spots -> spits -> spite -> spile -> smile
Panjang Path : 9
Jumlah node yang diperiksa : 4199
Runtime : 1632 ms
```


Greedy BFS

```
Masukan kata awal :
>> frown
Masukan kata tujuan :
>> smile
Pilihlah Algoritma yang diinginkan :
1. Algoritma Uniform Search Cost (UCS)
2. Algoritma Greedy Best First Search (Greedy BFS)
3. Algoritma A*
>> 2

frown -> brown -> crown -> drown -> grown -> groan -> groin -> grain -> brain -> drain -> train -> twain -> swain -> slain
-> stain -> stein -> skein -> skean -> spean -> speak -> sneak -> steak -> steek -> smeek -> smerk -> smirk -> shirk -> shi
re -> spire -> spile -> smile
Panjang Path : 30
Jumlah node yang diperiksa : 31
Runtime : 44 ms
```

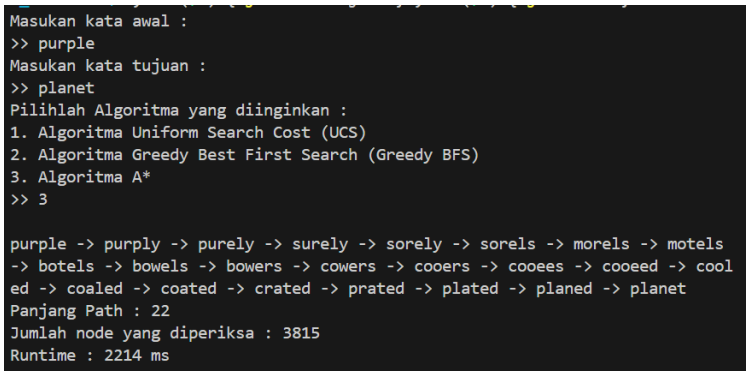

A*

```
Masukan kata awal :
>> frown
Masukan kata tujuan :
>> smile
Pilihlah Algoritma yang diinginkan :
1. Algoritma Uniform Search Cost (UCS)
2. Algoritma Greedy Best First Search (Greedy BFS)
3. Algoritma A*
>> 3

frown -> frows -> flows -> slows -> slops -> slips -> slipe -> stipe -> stile -> smile
Panjang Path : 9
Jumlah node yang diperiksa : 839
Runtime : 398 ms
```

</td></tr>
<tr><td>Start Word : Lied
End Word : Fact</td><td>UCS</td></tr>
</table>

	Masukan kata awal : >> lied Masukan kata tujuan : >> fact Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 1 lied -> hied -> haed -> hard -> hart -> fart -> fact Panjang Path : 6 Jumlah node yang diperiksa : 3192 Runtime : 989 ms Greedy BFS Masukan kata awal : >> lied Masukan kata tujuan : >> fact Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 2 lied -> died -> diet -> deet -> feet -> fret -> frat -> feat -> fiat -> flat -> flit -> frit -> brit -> bait -> gait -> wai t -> waft -> daft -> haft -> raft -> rant -> cant -> hant -> pant -> pact -> fact Panjang Path : 25 Jumlah node yang diperiksa : 26 Runtime : 38 ms A* Masukan kata awal : >> frown Masukan kata tujuan : >> smile Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 3 frown -> frows -> flows -> slows -> slops -> slips -> slipe -> stipe -> stile -> smile Panjang Path : 9 Jumlah node yang diperiksa : 839 Runtime : 477 ms
Start Word : Purple End Word : Planet	UCS Masukan kata awal : >> purple Masukan kata tujuan : >> planet Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 1 purple -> purply -> purely -> surely -> sorely -> sorels -> morels -> motels -> hotels -> hovels -> hovers -> covers -> cooers -> cooees -> cooeed -> cooled - > coaled -> coated -> crated -> prated -> plated -> planed -> planet Panjang Path : 22 Jumlah node yang diperiksa : 7026 Runtime : 3667 ms Greedy BFS

	Masukan kata awal : >> purple Masukan kata tujuan : >> planet Pilihlah Algoritma yang diinginkan : 1. Algoritma Uniform Search Cost (UCS) 2. Algoritma Greedy Best First Search (Greedy BFS) 3. Algoritma A* >> 2 Algoritma Greedy Best First Search tidak menemukan jalan dari purple ke planet. Jumlah node yang diperiksa : 2 Runtime : 38 ms A* 

4.3. Analisis Hasil Pengujian

Berdasarkan Hasil pengujian beberapa test case yang ada pada tabel diatas, dapat diambil kesimpulan bahwa Algoritma Uniform Cost Search dan A* selalu mendapatkan solusi yang optimal. Sedangkan, pada algoritma Greedy Best First Search tidak ada jaminan solusi optimal, bahkan bisa saja tidak mendapatkan solusi. Hal ini disebabkan sifat pencarian Greedy Best First Search yang tidak lengkap dibandingkan UCS dan A* yang lengkap.

Algoritma yang paling cepat diantara ketiga algoritma tersebut adalah Greedy Best First Search, disusul oleh A*, dan terakhir UCS. Hal ini disebabkan fungsi heuteristik yang membuat pemilihan simpul yang lebih baik sehingga memiliki waktu eksekusi yang lebih singkat. Bisa dilihat bahwa perbandingan jumlah node yang dikunjungi berbanding lurus dengan waktu eksekusi. Terdapat pengecualian pada kasus 6 yang tidak memiliki solusi. Pada kasus 6 algoritma A* lebih lambat dibandingkan UCS karena jumlah node yang diperiksa sama, akan tetapi A* lebih lambat karena pembuatan fungsi heuristik.

Algoritma Greedy Breach First Search memakan memori yang lebih sedikit dibandingkan algoritma UCS dan A*. Hal ini disebabkan Greedy Best First Search hanya memasukkan satu node dengan heuteristik terendah pada setiap iterasi ke dalam queue. Sedangkan UCS dan A* memasukkan semua kemungkinan node ke dalam queue. Hal ini membuat penggunaan memori UCS dan A* lebih boros dibandingkan Greedy Best First Search.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Solusi dari permainan Word Ladder dapat dicari menggunakan algoritma Uniform Cost Search, Greedy Best First Search, dan A*. Dari ketiga algoritma tersebut, algoritma A* adalah algoritma yang paling baik disebabkan memiliki waktu yang paling singkat untuk menemukan solusi yang optimal dengan syarat memiliki heuristik yang admissible. Algoritma A* menggabungkan keuntungan dari algoritma Greedy BFS dan UCS untuk menghasilkan algoritma yang lebih baik.

LAMPIRAN

Repository

https://github.com/HanifIHM/Tucil3_13522030

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