LAPORAN TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

PENYELESAIAN PERMAINAN *WORD LADDER* MENGGUNAKAN ALORITMA UCS, *GREEDY BEST FIRST SEARCH*, DAN A*

Oleh:

Hayya Zuhailii Kinasih 13522102

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I
ANALISIS DAN IMPLEMENTASI ALGORITMA

1.1. Implementasi Algoritma *Uniform Cost Search* (UCS)

Uniform Cost Search (UCS) adalah algoritma pencarian yang mirip dengan BFS/*Breadth First Search*. Perbedaan utamanya adalah UCS merupakan *informed search* yang memiliki informasi biaya berpindah dari satu simpul ke simpul lainnya. Oleh karena itu, simpul yang dikunjungi pada algoritma UCS diurutkan berdasarkan jaraknya/biayanya ke simpul awal, atau dikenal dengan notasi g(n). Algoritma ini memiliki kelengkapan dan dapat menemukan solusi yang optimal.

Langkah-langkah implementasi algoritma dengan UCS adalah sebagai berikut:

1. Melakukan inisiasi *priority queue* dengan komparator nilai g(n), *set visited*, dan *hashmap parent*. Kata awal dimasukkan ke dalam *priority queue* dan *parent*-nya diisi sebagai "ROOT". Nilai g(n) kata awal dijadikan 0.
2. Kata pertama dalam *priority queue* dijadikan simpul ekspan, dan dicatat bahwa kata tersebut telah diekspansi dalam *set visited*.
3. Jika kata tersebut adalah kata yang ingin dituju, pencarian berakhir.
4. Jika tidak, melakukan iterasi terhadap kata-kata yang berbeda tepat satu karakter dengan kata tersebut. Jika kata yang didapat belum diekspansi, nilai *parent*-nya adalah kata yang sebelumnya, nilai g(n)-nya adalah nilai g(n) induknya ditambah 1, dan kata tersebut dimasukkan ke dalam *priority queue*.
5. Mengulangi langkah 2-4 hingga kata akhir ditemukan atau *priority queue* kosong.

1.2 Implementasi Algoritma *Greedy Best First Search*

Greedy Best First Search (GBFS) adalah algoritma pencarian yang mempertimbangkan jarak suatu simpul ke simpul tujuan. Saat pencarian, jarak tersebut tidak diketahui secara pasti, melainkan menggunakan heuristik yang memperkirakan jaraknya, atau dikenal dengan notasi h(n). Heuristik yang dapat digunakan adalah yang tidak mengoverestimasi jarak simpul ke simpul tujuan. Algoritma ini tidak memiliki kelengkapan dan bisa tidak menemukan solusi yang optimal, namun karena menggunakan fungsi heuristik, algoritma ini dapat menemukan solusi dengan cepat.

Langkah-langkah implementasi algoritma dengan GBFS adalah sebagai berikut:

1. Melakukan inisiasi *priority queue* dengan komparator nilai h(n), *set visited*, dan *hashmap parent*. Kata awal dimasukkan ke dalam *priority queue* dan *parent*-nya diisi sebagai "ROOT". Nilai h(n) kata awal dihitung.
2. Kata pertama dalam *priority queue* dijadikan simpul ekspan, dan dicatat bahwa kata tersebut telah diekspansi dalam *set visited.*
3. Jika kata tersebut adalah kata yang ingin dituju, pencarian berakhir.

4. Jika tidak, melakukan iterasi terhadap kata-kata yang berbeda tepat satu karakter dengan kata tersebut. Jika kata yang didapat belum diekspansi, nilai *parent*-nya adalah kata yang sebelumnya, nilai h(n)-nya dihitung, dan kata tersebut dimasukkan ke dalam *priority queue*.
5. Mengulangi langkah 2-4 hingga kata akhir ditemukan atau *priority queue* kosong.

1.3 Implementasi Algoritma A*

Algoritma A* adalah penggabungan dari kedua algoritma yang telah disebutkan. Algoritma ini mengimplementasikan g(n) dari UCS dan h(n) dari GBFS untuk menemukan solusi yang optimal dengan arahan lebih menuju simpul tujuan. Algoritma ini memiliki kelengkapan dan akan menemukan solusi yang optimal dalam waktu yang lebih singkat daripada UCS karena pencarian lebih terarah.

Langkah-langkah implementasi algoritma dengan A* adalah sebagai berikut:

1. Melakukan inisiasi *priority queue* dengan komparator nilai f(n), *set visited*, dan *hashmap parent*. Kata awal dimasukkan ke dalam *priority queue* dan *parent*-nya diisi sebagai "ROOT". Nilai g(n) kata awal dijadikan 0 dan nilai h(n) kata awal dihitung.
2. Kata pertama dalam *priority queue* dijadikan simpul ekspan, dan dicatat bahwa kata tersebut telah diekspansi dalam *set visited.*
3. Jika kata tersebut adalah kata yang ingin dituju, pencarian berakhir.
4. Jika tidak, melakukan iterasi terhadap kata-kata yang berbeda tepat satu karakter dengan kata tersebut. Jika kata yang didapat belum diekspansi, nilai *parent*-nya adalah kata yang sebelumnya, nilai g(n)-nya adalah nilai g(n) induknya ditambah 1, nilai h(n)-nya dihitung, dan kata tersebut dimasukkan ke dalam *priority queue*.
5. Mengulangi langkah 2-4 hingga kata akhir ditemukan atau *priority queue* kosong.

BAB 2
SOURCE CODE PROGRAM

2.1 Class Main

Kelas ini berguna untuk melakukan validasi input, memanggil inisiasi daftar kata dalam kamus, dan memanggil fungsi untuk melakukan pencarian. Kelas ini tidak memiliki atribut dan hanya memiliki satu metode, yaitu *main*. Metode tersebut mengambil parameter berupa *Command Line Arguments* dengan format:

```
java Main <kata awal> <kata akhir> <jenis algoritma>
```

```
public static void main(String[] args) {
        String startWord = args[0].toLowerCase();
        String endWord = args[1].toLowerCase();
        String algorithm = args[2].toLowerCase();
        if (startWord.length() != endWord.length()) {
            System.out.println("Start and end words must have the
same length.");
            return;
        }

        String filepath = String.format("words//%dletter.txt",
startWord.length());
        Words words = new Words(filepath);

        if (!words.hasWord(startWord)) {
            System.out.println("Start word does not exist in
dictionary.");
        } else if (!words.hasWord(endWord)) {
            System.out.println("End word does not exist in
dictionary.");
        } else {
            long start = System.currentTimeMillis();
            if (algorithm.equals("astar")) {
                Astar astar = new Astar(startWord, endWord,
words);
                astar.search();
                System.out.printf("Time elapsed: %dms\n",
```

```
System.currentTimeMillis() - start);
            } else if (algorithm.equals("ucs")) {
                UCS ucs = new UCS(startWord, endWord, words);
                ucs.search();
                System.out.printf("Time elapsed: %dms\n",
System.currentTimeMillis() - start);
            } else if (algorithm.equals("gbfs")) {
                GBFS gbfs = new GBFS(startWord, endWord, words);
                gbfs.search();
                System.out.printf("Time elapsed: %dms\n",
System.currentTimeMillis() - start);
            } else {
                System.out.println("Algorithm input invalid.");
            }
        }
    }
```

2.2 Class Words

Kelas ini berisi *Hash Map* dengan *key* adalah suatu kata dan *value* adalah daftar kata yang berbeda satu karakter dengan kata tersebut. Data tersebut didapatkan dengan membaca *text file* yang sesuai dengan panjang kata yang ingin dicari. *Text file* tersebut berisi kata dan kata-kata yang bertetangga dengannya.

Metode Words(String filepath)

```
public Words(String filepath) {
        try {
            wordList = new HashMap<String, List<String>>();
            File file = new File(filepath);
            Scanner s1 = new Scanner(file);

            while (s1.hasNextLine()) {
                String word = s1.nextLine();
                String neighborStr = s1.nextLine();
                List<String> neighbors = new ArrayList<String>();
                if (!neighborStr.isEmpty()) {
                    Scanner s2 = new Scanner(neighborStr);
                    s2.useDelimiter(" ");
```

```
                    while (s2.hasNext()) {
                        neighbors.add(s2.next());
                    }
                    s2.close();
                }
                // Word w = new Word(word, neighbors);
                wordList.put(word, neighbors);
            }
            s1.close();
        } catch (FileNotFoundException e) {
            System.err.println("Dictionary only contains words up
to 8 characters in length.");
        }

    }
```

Metode getNeighbors(String word)

```
public List<String> getNeighbors(String word) {
        return wordList.get(word);
    }
```

Metode hasWord(String word)

```
public boolean hasWord(String word) {
        return wordList.containsKey(word);
    }
```

2.3 Class Word

Kelas ini berisi data dari suatu kata saat sedang dilakukan pencarian. Atribut yang dimiliki kelas ini adalah *name* yang berupa kata itu sendiri, g yang bernilai g(n), dan h yang bernilai h(n). Metode yang dimiliki kelas ini adalah konstruktor yang mengambil argumen *name* dan *calculateH* yang mengkalkulasi nilai h(n) dari kata tersebut.

Metode Word(String name)

```
public Word(String name) {
        this.name = name;
        this.g = 0;
        this.h = 0;
```

```
    }
```

Metode calculateH(String end)

```
public void calculateH(String end) {
        int res = 0;
        for (int i=0; i<this.name.length(); i++) {
            if (this.name.charAt(i) != end.charAt(i)) {
                res++;
            }
        }
        this.h = res;
    }
```

2.4 Class UCS

Kelas ini memiliki atribut berupa kata awal, kata tujuan, daftar kata, dan data induk dari suatu kata, serta mengandung metode konstruktor, *printPath* untuk menampilkan hasil pencarian, dan *search* untuk melakukan pencarian.

Metode UCS(String start, String end, Words words)

```
public UCS(String start, String end, Words words) {
        this.start = start;
        this.end = end;
        this.words = words;
    }
```

Metode printPath(Word word)

```
public void printPath(Word word) {
        Word current = word;
        List<String> path = new ArrayList<String>();
        while (!parent.get(current).name.equals("ROOT")) {
            path.add(0, current.name);
            current = parent.get(current);
        }
        path.add(0, current.name);
        System.out.printf("Path: %d words\n", path.size()-1);
        System.out.println(path);
    }
```

Metode search()

```
public void search() {
        Word root = new Word("ROOT");
        Word startWord = new Word(start);

        PriorityQueue<Word> pq = new PriorityQueue<>(new
compareG());
        pq.add(startWord);

        parent = new HashMap<Word, Word>();
        parent.put(startWord, root);

        Set<String> visited = new HashSet<String>();

        while (!pq.isEmpty()) {
            Word current = pq.poll();
            visited.add(current.name);
            if (current.name.equals(end)) {
                this.printPath(current);
                System.out.printf("Words visited: %d\n",
visited.size());
                Runtime runtime = Runtime.getRuntime();
                runtime.gc();
                System.out.print("Memory used: ");
                System.out.println(runtime.totalMemory() -
runtime.freeMemory());
                break;
            }
            for (String child : words.getNeighbors(current.name))
{
                if (!visited.contains(child)) {
                    Word next = new Word(child);
                    next.g = current.g + 1;
                    parent.put(next, current);
                    pq.add(next);
                }
            }
        }
```

```
        if (pq.isEmpty()) {
            System.out.println("Path not found.");
        }
    }
```

2.5 Class CompareG

Kelas ini adalah komparator yang digunakan dalam algoritma UCS.

```
class CompareG implements Comparator<Word> {
    public int compare(Word a, Word b) {
        return a.g - b.g;
    }
}
```

2.6 Class GBFS

Kelas ini memiliki atribut berupa kata awal, kata tujuan, daftar kata, dan data induk dari suatu kata, serta mengandung metode konstruktor, *printPath* untuk menampilkan hasil pencarian, dan *search* untuk melakukan pencarian.

Metode GBFS(String start, String end, Words words)

```
public GBFS(String start, String end, Words words) {
        this.start = start;
        this.end = end;
        this.words = words;
    }
```

Metode printPath(Word word)

```
public void printPath(Word word) {
        Word current = word;
        List<String> path = new ArrayList<String>();
        while (!parent.get(current).name.equals("ROOT")) {
            path.add(0, current.name);
            current = parent.get(current);
        }
        path.add(0, current.name);
```

```
        System.out.printf("Path: %d words\n", path.size()-1);
        System.out.println(path);
    }
```

Metode search()

```
public void search() {
        Word root = new Word("ROOT");
        Word startWord = new Word(start);
        startWord.calculateH(end);

        PriorityQueue<Word> pq = new PriorityQueue<>(new
CompareH());
        pq.add(startWord);

        parent = new HashMap<Word, Word>();
        parent.put(startWord, root);

        Set<String> visited = new HashSet<String>();

        while (!pq.isEmpty()) {
            Word current = pq.poll();
            visited.add(current.name);
            if (current.name.equals(end)) {
                this.printPath(current);
                System.out.printf("Words visited: %d\n",
visited.size());
                Runtime runtime = Runtime.getRuntime();
                runtime.gc();
                System.out.print("Memory used: ");
                System.out.println(runtime.totalMemory() -
runtime.freeMemory());
                break;
            }
            for (String child : words.getNeighbors(current.name))
{
                if (!visited.contains(child)) {
                    Word next = new Word(child);
                    next.calculateH(end);
                    parent.put(next, current);
```

```
                    pq.add(next);
                }
            }
        }

        if (pq.isEmpty()) {
            System.out.println("Path not found.");
        }
    }
```

2.7 Class CompareH

Kelas ini adalah komparator yang digunakan dalam algoritma GBFS.

```
class CompareH implements Comparator<Word> {
    public int compare(Word a, Word b) {
        return a.h - b.h;
    }
}
```

2.8 Class Astar

Kelas ini memiliki atribut berupa kata awal, kata tujuan, daftar kata, dan data induk dari suatu kata, serta mengandung metode konstruktor, *printPath* untuk menampilkan hasil pencarian, dan *search* untuk melakukan pencarian.

Metode GBFS(String start, String end, Words words)

```
public Astar(String start, String end, Words words) {
        this.start = start;
        this.end = end;
        this.words = words;
    }
```

Metode printPath(Word word)

```
public void printPath(Word word) {
        Word current = word;
        List<String> path = new ArrayList<String>();
        while (!parent.get(current).name.equals("ROOT")) {
```

```
            path.add(0, current.name);
            current = parent.get(current);
        }
        path.add(0, current.name);
        System.out.printf("Path: %d words\n", path.size()-1);
        System.out.println(path);
    }
```

Metode search()

```
public void search() {
        Word root = new Word("ROOT");
        Word startWord = new Word(start);
        startWord.calculateH(end);

        PriorityQueue<Word> pq = new PriorityQueue<>(new
CompareF());
        pq.add(startWord);

        parent = new HashMap<Word, Word>();
        parent.put(startWord, root);

        Set<String> visited = new HashSet<String>();

        while (!pq.isEmpty()) {
            Word current = pq.poll();
            visited.add(current.name);
            if (current.name.equals(end)) {
                this.printPath(current);
                System.out.printf("Words visited: %d\n",
visited.size());
                Runtime runtime = Runtime.getRuntime();
                runtime.gc();
                System.out.print("Memory used: ");
                System.out.println(runtime.totalMemory() -
runtime.freeMemory());
                break;
            }
            for (String child : words.getNeighbors(current.name))
{
```

```
                if (!visited.contains(child)) {
                    Word next = new Word(child);
                    next.g = current.g + 1;
                    next.calculateH(end);
                    parent.put(next, current);
                    pq.add(next);
                }
            }
        }

        if (pq.isEmpty()) {
            System.out.println("Path not found.");
        }
    }
```

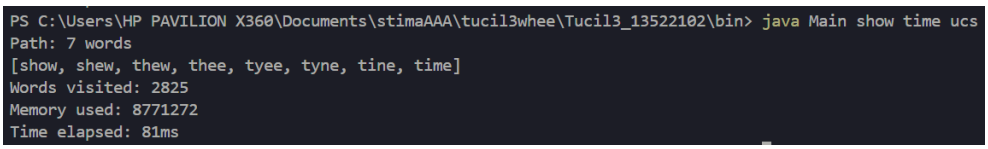
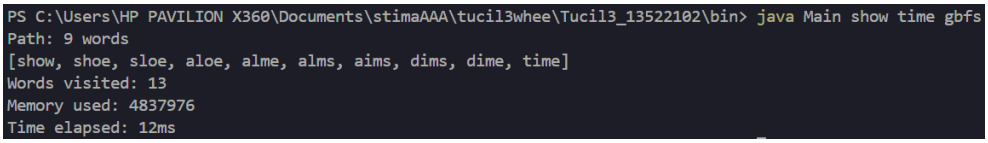
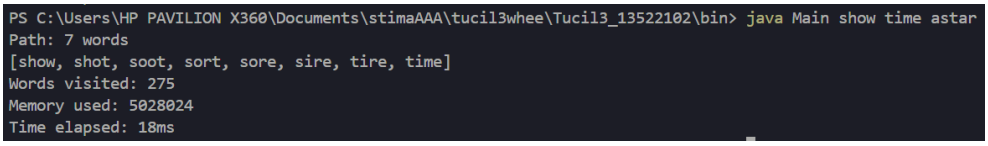
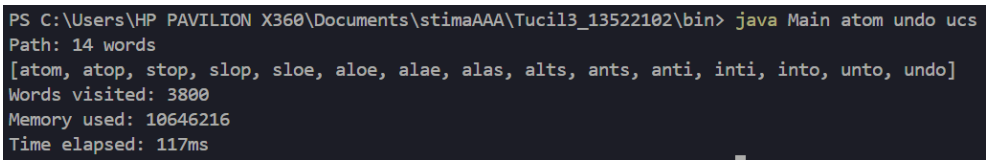
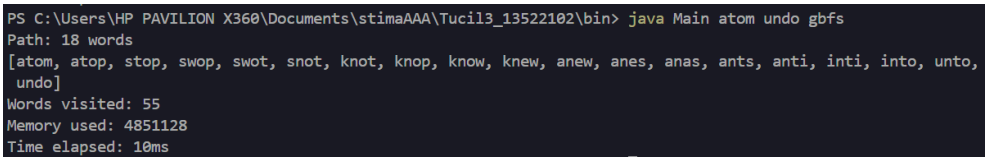
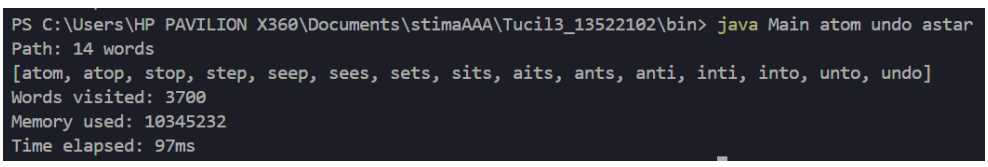
2.9 Class CompareF

Kelas ini adalah komparator yang digunakan dalam algoritma Astar.

```
class CompareF implements Comparator<Word> {
    public int compare(Word a, Word b) {
        return a.g + a.h - b.g - b.h;
    }
}
```

BAB 3
PENGUJIAN

3.1 Hasil Pengujian

<table>
<tr><th>No</th><th>Input</th><th>Hasil</th></tr>
<tr><td rowspan="3">1</td><td rowspan="3">Start: show
End: time</td><td>UCS
 
</td></tr>
<tr><td>GBFS
 
</td></tr>
<tr><td>A*
 
</td></tr>
<tr><td rowspan="3">2</td><td rowspan="3">Start: atom
End: undo</td><td>UCS
 
</td></tr>
<tr><td>GBFS
 
</td></tr>
<tr><td>A*
 
</td></tr>
</table>

3	Start: orange End: purply	UCS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main orange purply ucs Path: 24 words [orange, orangs, prangs, pranks, tranks, thanks, shanks, sharks, shirks, shirrs, shiers, shyers, sayers, savers, havers, hovers, hovels, hotels, motels, morels, sorels, sorely, surely, purely, purply] Words visited: 8367 Memory used: 16572840 Time elapsed: 151ms GBFS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main orange purply gbfs Path: 60 words [orange, orangy, orangs, prangs, pranks, planks, plunks, plucks, placks, places, placet, placed, peaced, peaked, perked, parked, parged, purged, purger, burger, burgee, burgle, bungle, jungle, jungly, jingly, tingly, tinily, tinkly, tinkle, tickle, pickle, sickle, sickly, fickly, fickle, nickle, mickle, mackle, rackle, hackle, heckle, deckle, decile, defile, refile, refill, refall, befall, befell, bedell, bedels, betels, botels, motels, morels, sorels, sorely, surely, purely, purply] Words visited: 2232 Memory used: 7958320 Time elapsed: 41ms A* PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main orange purply astar Path: 24 words [orange, orangs, wrangs, whangs, changs, chants, charts, charrs, chirrs, shirrs, shiers, shyers, sayers, savers, havers, hovers, hovels, hotels, motels, morels, sorels, sorely, surely, purely, purply] Words visited: 7640 Memory used: 12189616 Time elapsed: 121ms
4	Start: gimlets End: treeing	UCS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main gimlets treeing ucs Path: 24 words [gimlets, giglets, wiglets, willets, wallets, ballets, ballers, bailers, bailees, bailies, dailies, doilies, dollies, collies, collins, codlins, codling, godling, godding, goading, grading, graying, greying, greeing, treeing] Words visited: 6208 Memory used: 31300944 Time elapsed: 341ms GBFS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main gimlets treeing gbfs Path: 80 words [gimlets, giglets, wiglets, willets, willers, tillers, tilters, tinters, tenters, teeters, tetters, petters, setters, sitters, sisters, sinters, winters, winners, winkers, wingers, zingers, singers, sinkers, pinkers, punkers, punters, putters, puttees, putties, tutties, tatties, tattier, tastier, vastier, pastier, pestier, peskier, peakier, peckier, pockier, rockier, rookier, rookies, bookies, boobies, bobbies, bobbins, bobbing, bibbing, ribbing, rubbing, tubbing, tabbing, tabling, tailing, toiling, toiting, totting, tatting, tasting, testing, tenting, tensing, teasing, tearing, searing, staring, storing, stowing, stewing, shewing, chewing, crewing, creping, craping, crating, prating, praying, preying, preeing, treeing] Words visited: 672 Memory used: 8222784 Time elapsed: 32ms A* PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main gimlets treeing astar Path: 24 words [gimlets, giglets, wiglets, willets, billets, billers, ballers, bailers, bailees, bailies, dailies, doilies, dollies, collies, collins, codlins, codling, coaling, goaling, goading, grading, graying, greying, greeing, treeing] Words visited: 4076 Memory used: 10669424 Time elapsed: 152ms
5	Start:	UCS

	quirking End: wrathing	PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main quirking wrathing ucs Path: 31 words [quirking, quirting, quilting, quilling, quelling, duelling, dwelling, swelling, spelling, spalling, stalling, starling, starting, scarting, scatting, slatting, blatting, blasting, boasting, coasting, coacting, coaching, couching, mouching, mouthing, southing, soothing, toothing, trothing, trithing, writhing, wrathing] Words visited: 861 Memory used: 8100592 Time elapsed: 32ms GBFS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main quirking wrathing gbfs Path: 49 words [quirking, quirting, quieting, quilting, quilling, quelling, fuelling, duelling, dwelling, swelling, swilling, twilling, trilling, trolling, drolling, drooling, drooping, trooping, trouping, tromping, tramping, trapping, wrapping, whapping, chapping, clapping, clasping, clashing, slashing, swashing, swathing, scathing, scatting, slatting, blatting, blasting, boasting, coasting, coacting, coaching, couching, mouching, mouthing, southing, soothing, toothing, trothing, trithing, writhing, wrathing] Words visited: 390 Memory used: 7718256 Time elapsed: 26ms A* PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main quirking wrathing astar Path: 31 words [quirking, quirting, quilting, quilling, quelling, duelling, dwelling, swelling, spelling, spalling, sparling, sparring, scarring, scarting, scatting, slatting, blatting, blasting, boasting, coasting, coacting, coaching, couching, mouching, mouthing, southing, soothing, toothing, trothing, trithing, writhing, wrathing] Words visited: 836 Memory used: 7863384 Time elapsed: 33ms
6	Start: atlases End: cabaret	UCS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main atlases cabaret ucs Path: 52 words [atlases, anlases, anlaces, unlaces, unlades, unladed, unfaded, unfaked, uncaked, uncakes, uncases, uneases, ureases, creases, cresses, crosses, crosser, crasser, crasher, brasher, brasier, brakier, beakier, peakier, peckier, pickier, dickier, dickies, hickies, hackies, hackles, heckles, deckles, deciles, defiles, refiles, refiled, reviled, reveled, raveled, ravened, havened, havered, wavered, watered, catered, capered, tapered, tabered, tabored, taboret, tabaret, cabaret] Words visited: 7648 Memory used: 92241176 Time elapsed: 1395ms GBFS PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main atlases cabaret gbfs Path: 180 words [atlases, anlases, anlaces, unlaces, unlaced, unladed, unjaded, unfaded, unfazed, unrazed, unrated, unbated, unbased, uncased, uncases, uneases, ureases, creases, creased, created, cleated, cleared, clearer, cleaver, cleaves, sleaves, sheaves, sheaved, sheared, speared, speered, sheered, cheered, cheesed, cheeses, chesses, chasses, chassed, chasmed, charmed, charted, chanted, chantey, chanter, chaster, coaster, coasted, coapted, coacted, coached, couched, coughed, toughed, tougher, toughen, roughen, rougher, roughed, soughed, southed, souther, soother, sootier, zootier, rootier, rookier, rookies, cookies, coolies, collies, cullies, cullied, sullied, sallied, sallier, pallier, palmier, balmier, barmier, barnier, barrier, marrier, marlier, manlier, mangier, tangier, tangler, tangled, dangled, daggled, daggles, haggles, higgles, wiggles, wiggler, wiggled, niggled, niggles, miggles, mingles, tingles, tinkles, winkles, wintles, wintled, winkled, windled, windles, widdles, widdled, riddled, ruddled, ruddles, rundles, runkles, rankles, rankled, runkled, ruckled, suckled, suckles, ruckles, huckles, heckles, keckles, deckles, deciles, decides, decider, derider, deriver, derives, derived, derided, decided, decoded, demoded, demoted, denoted, denotes, demotes, demoses, demises, semises, remises, revises, revives, reviver, revived, revised, reviser, reviler, reviles, retiles, retires, refires, refixes, refixed, remixed, remised, demised, devised, deviled, develed, reveled, raveled, ravened, havened, havered, wavered, watered, catered, capered, caperer, taperer, tapered, tabered, tabored, taboret, tabaret, cabaret] Words visited: 4390 Memory used: 9103336 Time elapsed: 59ms A*

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

```
PS C:\Users\HP PAVILION X360\Documents\stimaAAA\tucil3whee\Tucil3_13522102\bin> java Main atlases cabaret as
tar
Path: 52 words
[atlases, anlases, anlaces, unlaces, unlades, unladed, unfaded, unfaked, uncaked, uncased, uncases, uneases,
 ureases, creases, cresses, tresses, trasses, trashes, brashes, brasher, brasier, brakier, beakier, peakier,
 peckier, pickier, dickier, dickies, hickies, hackies, hackles, heckles, deckles, deciles, defiles, refiles,
 reviles, reviled, reveled, raveled, ravened, havened, havered, wavered, watered, catered, capered, tapered,
 tabered, tabored, taboret, tabaret, cabaret]
Words visited: 7595
Memory used: 19752208
Time elapsed: 224ms
```

</td></tr>
<tr><td>7</td><td>Start: starling
End: soothing</td><td>UCS

```
PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main starling soothing ucs
Path: 15 words
[starling, starting, scarting, scatting, slatting, blatting, blasting, boasting, coasting, coacting, coachin
g, couching, mouching, mouthing, southing, soothing]
Words visited: 816
Memory used: 7903512
Time elapsed: 27ms
```

GBFS

```
PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main starling soothing gbfs
Path: 43 words
[starling, snarling, sparling, sparring, starring, stirring, skirring, skirting, spirting, sporting, spoutin
g, snouting, snooting, snooling, stooling, stooking, stocking, shocking, chocking, checking, chucking, cluck
ing, plucking, plunking, plinking, prinking, printing, pointing, jointing, joisting, jousting, rousting, roa
sting, coasting, coapting, coacting, coaching, poaching, pooching, mooching, mouching, mouthing, southing, s
oothing]
Words visited: 464
Memory used: 7723096
Time elapsed: 20ms
```

A*

```
PS C:\Users\HP PAVILION X360\Documents\stimaAAA\Tucil3_13522102\bin> java Main starling soothing astar
Path: 15 words
[starling, starting, scarting, scatting, slatting, blatting, blasting, boasting, coasting, coacting, coachin
g, couching, coughing, soughing, southing, soothing]
Words visited: 716
Memory used: 7854352
Time elapsed: 26ms
```

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

3.2 Analisis

Berikut adalah tabel yang merangkum hasil pengujian.

No	Panjang kata	Panjang solusi optimal	UCS	GBFS	A*
1	4	7	8771272 bytes 81 ms	4837976 bytes 12 ms	5028024 bytes 18 ms
2	4	14	10646216 bytes 117 ms	4851128 bytes 10 ms	10345232 bytes 97 ms
3	6	24	16572840 bytes 151 ms	7958320 bytes 41 ms	12189616 bytes 121 ms
4	7	24	31300944 bytes	8222784 bytes	10669424 bytes

			341 ms	32 ms	152 ms
5	8	31	8100592 bytes 32 ms	7718256 bytes 26 ms	7863384 bytes 33 ms
6	7	52	92241176 bytes 1395 ms	9103336 bytes 59 ms	19752208 bytes 224 ms
7	8	15	7903512 bytes 27 ms	7723096 bytes 20 ms	7854352 bytes 26 ms

Berdasarkan pasangan-pasangan test case yang memiliki panjang kata yang sama, terlihat bahwa GBFS tidak terlalu terpengaruh oleh panjang solusi optimal jika dibandingkan dengan algoritma yang lain. GBFS juga menemukan solusi dengan lebih cepat dan menggunakan paling sedikit memori. Namun, berdasarkan hasil uji, GBFS tidak menemukan solusi yang optimal.

Selain itu, *branching factor* adalah sesuatu yang sangat mempengaruhi hasil dari UCS dan A*, terlihat pada test case 4 dan 6 dimana kata dengan panjang 7 huruf memiliki *branching factor* yang tinggi, sehingga seiring bertambah kedalaman solusi optimal, semakin meningkat ruang dan waktu yang digunakan kedua algoritma tersebut. Dibandingkan dengan test case 5 dan 8, dimana kata dengan panjang huruf 8 memiliki *branching factor* yang kecil karena banyak kata yang *aloof* alias tidak memiliki simpul lain yang terhubung. Pada test case tersebut, besar ruang dan waktu yang digunakan meningkat dengan lebih lambat, bahkan tidak berbeda jauh dengan GBFS.

Satu “anomali” dapat ditemukan pada test case 3 dan 4, dimana waktu dan ruang yang digunakan algoritma UCS meningkat, namun waktu GBFS menurun dan ruang A* juga menurun. Ketiga algoritma ini pada dasarnya sangat mirip. Satu-satunya perbedaan di antaranya adalah cara ketiga algoritma menentukan simpul mana yang paling optimal untuk ditelusuri berikutnya. Algoritma GBFS dan A* menerapkan fungsi heuristik yang sangat mengarahkan pencarian, sehingga penemuan solusi bisa jauh lebih cepat. Itulah yang kemungkinan terjadi pada test case tersebut.

LAMPIRAN

Repository Github
https://github.com/hayyazk/Tucil3_13522102.git

Checklist Pengerjaan

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	☐	☑