

LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

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



Disusun oleh:

Indraswara Galih Jayanegara

13522119

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I.....	2
DESKRIPSI TUGAS.....	2
1.1 Deskripsi Tugas.....	2
BAB II.....	3
PENJELASAN ALGORITMA.....	3
1. Uniform Cost Search (UCS).....	3
2. Greedy Best First Search.....	4
3. A*.....	4
BAB III.....	5
IMPLEMENTASI DAN ANALISIS.....	5
4.1 Implementasi Algoritma UCS.....	5
4.2 Algoritma Greedy Best First Search.....	6
4.3 Algoritma A*.....	7
4.4 Pengujian.....	8
4.5 Analisis.....	13
4.6 GUI.....	14
BAB IV.....	16
KESIMPULAN DAN SARAN.....	16
5. 1 Kesimpulan.....	16
5.2 Saran.....	16
LAMPIRAN.....	17
DAFTAR PUSTAKA.....	18

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Tugas

Word ladder, juga dikenal sebagai Doublets, adalah permainan kata yang populer di mana pemain harus menghubungkan dua kata, yang disebut sebagai kata awal (start word) dan kata akhir (end word), melalui serangkaian kata yang berbeda satu huruf secara berurutan. Permainan ini diciptakan oleh Lewis Carroll pada tahun 1877. Tujuannya adalah menemukan rantai kata terpendek yang menghubungkan kata awal dengan kata akhir. Kamus kata digunakan untuk memverifikasi apakah sebuah kata valid dalam permainan.

Permainan dimulai dengan dua kata yang memiliki jumlah huruf yang sama. Pemain kemudian mencoba mengganti satu huruf pada kata awal untuk membuat kata baru yang juga harus ada dalam kamus. Langkah ini diulang dengan mengganti satu huruf pada kata baru untuk menciptakan kata-kata lain dalam rantai, hingga mencapai kata akhir.

Kata-kata yang digunakan dalam rantai harus valid dan terdaftar dalam kamus yang digunakan oleh program. Pemain bertujuan untuk menemukan rantai kata terpendek, yang menghubungkan kata awal dengan kata akhir dengan jumlah langkah (atau kata-kata dalam rantai) yang minimal.

Untuk menyelesaikan permainan, seorang pemain harus memiliki pengetahuan yang baik tentang kosakata bahasa yang digunakan. Kamus yang digunakan juga penting untuk memastikan kata-kata yang dihasilkan dalam permainan valid.

Dalam tugas ini, Penulis ditugaskan untuk membuat program yang mencari perubahan kata yang optimal dari kata pertama sampai kata tujuan menggunakan algoritma UCS, *Greedy Best First Search*, dan A*.

BAB II

PENJELASAN ALGORITMA

Pada Permainan ini, setiap langkahnya hanya boleh mengubah satu karakter tiap langkah dan langkah yang dipilih selanjutnya tidak boleh langkah yang sudah pernah dilakukan sebelumnya. Selain itu, banyak sekali kemungkinan yang terjadi dari kombinasi-kombinasi karakter-karakter yang ada sehingga permasalahan ini bisa direpresentasikan dengan graf.

Pemodelan masalah ini pada graf adalah sebagai berikut:

1. Simpul (Node): kata terakhir yang diubah
2. Sisi (Edge): langkah yang dipilih.

Setiap langkah yang dilakukan akan membuat percabangan kata dari setiap karakter yang berbeda. Akan tetapi, perlu diperhatikan kata-kata yang dihasilkan dari perubahan karakter harus valid atau harus ada pada kamus bahasa inggris yang dijadikan acuan. Pada tugas kecil kali ini, akan digunakan tiga algoritma pencarian graf yakni, sebagai berikut:

1. *Uniform Cost Search (UCS)*

Algoritma ini merupakan algoritma yang dijamin optimal. Pada algoritma ini saat sebuah jalur sudah tidak sesuai ketentuan kita bisa semacam *backtracking*, tapi tidak bukan *backtracking* yang seharusnya, hanya memilih jalur yang bisa dilewati. Setiap simpul yang dilewati akan diberikan cost. Definisi cost-nya sendiri biasa dituliskan dengan $g(n)$. Langkah-langkah algoritma ini adalah

1. Buat sebuah priority queue untuk menyimpan jalur yang dilalui dan set untuk memeriksa apakah sebuah kata sudah pernah di-visit atau belum.
2. Masukkan kata awal, dan berikan cost-nya sebesar 0.
3. Pastikan priority queue memiliki isi. Jika tidak, eksekusi selesai.
4. Bangkitkan sebuah simpul yang memiliki perbedaan satu karakter, periksa apakah simpul yang dibangkitkan sudah pernah ada pada set, jika sudah biarkan dan kembali ke poin 3, jika belum masukkan ke dalam set dan lanjutkan ke poin berikutnya.
5. Periksa apakah simpul yang dihasilkan merupakan kata akhir, jika sudah kembalikan jalur yang menuju dari kata awal ke kata akhir. Jika belum lanjut poin ke-6.
6. berikan cost sebesar $\text{current cost} + 1$ ($g(n)$) pada simpul, lalu masukkan ke dalam priority queue.
7. ulangi dari poin 3.

Algoritma ini memiliki kelemahan yaitu memiliki total kunjungan yang lebih banyak dari algoritma-algoritma lainnya.

2. *Greedy Best First Search*

Algoritma ini tidak selalu memberikan solusi yang optimal. Berbeda dari UCS algoritma ini menentukan cost-nya dengan heuristic atau bisa disebut $h(n)$. Untuk $h(n)$ adalah perbedaan karakter dari setiap kata dengan kata yang akan dituju. Contohnya, *fire* menuju ke *food* terdapat 3 perbedaan huruf maka *cost* dari *fire* adalah 3.

Langkah-langkah dari algoritma ini adalah:

1. Sediakan *priority queue* untuk menyimpan setiap *node* yang dikunjungi. Selain, itu sediakan set untuk mengecek apakah kata yang dikunjungi sudah pernah dikunjungi atau belum.
2. Masukkan kata awal dan *cost* awalnya dengan $h(n)$.
3. Lakukan ekspansi pada *node* saat ini.
4. Setiap *node* yang dikunjungi diberikan *cost* sesuai dengan heuristic yang telah ditentukan.
5. Pilih *node* yang paling kecil untuk dikunjungi.
6. Apabila tidak ada lagi yang bisa dikunjungi maka hentikan pencarian karena hasil tidak ditemukan.

3. A^*

Algoritma ini merupakan gabungan dari *Greedy Best First Search* dan UCS. Penentuan *cost*-nya berdasarkan $f(n) = g(n) + h(n)$ atau *cost* dari UCS digabung dengan *cost* dari *Greedy Best First Search*. Langkah algoritma yang dilakukan hampir sama dengan UCS hanya berbeda pada penentuan *cost*-nya Berikut adalah langkah-langkah yang dilakukan:

1. Buat sebuah *priority queue* untuk menyimpan jalur yang dilalui dan set untuk memeriksa apakah sebuah kata sudah pernah di-visit atau belum.
2. Masukkan kata awal, dan berikan *cost*-nya sebesar 0.
3. Pastikan *priority queue* memiliki isi. Jika tidak, eksekusi selesai.
4. Bangkitkan sebuah simpul yang memiliki perbedaan satu karakter, periksa apakah simpul yang dibangkitkan sudah pernah ada pada set, jika sudah biarkan dan kembali ke poin 3, jika belum masukkan ke dalam set dan lanjutkan ke poin berikutnya.
5. Periksa apakah simpul yang dihasilkan merupakan kata akhir, jika sudah kembalikan jalur yang menuju dari kata awal ke kata akhir. Jika belum lanjut poin ke-6.
6. berikan *cost* sebesar $\text{currCost} + 1 (g(n)) + h(n)$ pada simpul, lalu masukkan ke dalam *priority queue*.
7. ulangi dari poin 3.

BAB III

IMPLEMENTASI DAN ANALISIS

Pada bagian ini saya membuat *node* yang isi-isinya adalah String kata, *cost*, dan *parent* dari *node* tersebut. Sehingga saat hasil ditemukan dari sebuah *node* saya bisa menelusuri *path*-nya berdasarkan *parent* dari *node* tersebut. Sebagian besar implementasi pada UCS, *Greedy Best First Search*, dan A* memiliki kode yang sama perbedaannya hanya pada pemberian *cost* saya. UCS menggunakan $g(n)$, *Greedy Best First Search* menggunakan $h(n)$, dan A* menggunakan $f(n) = g(n) + h(n)$.

4.1 Implementasi Algoritma UCS

Seperti yang bisa dilihat UCS yang saya buat menggunakan *Priority Queue* untuk menyimpan *node* yang dikunjungi dan diekspansi. Selain itu, saya juga memakai Set untuk *visited words*. Sebagian besar program yang dibuat sama seperti yang ada pada langkah-langkah algoritma. *Cost* yang diberikan pada setiap simpul adalah $\text{currentCost} + 1$

```
1 public class UCS extends Algorithm{
2     public SearchResult process(String startWord, String endWord, List<String> dictionary) {
3         PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparingInt(Node -> Node.getCost()));
4         ArrayList<String> visitedWords = new ArrayList<>();
5
6         priorityQueue.add(new Node(startWord, 0, null));
7
8         while (!priorityQueue.isEmpty()) {
9             Node currNode = priorityQueue.poll();
10            String currWord = currNode.getWord();
11            int currCost = currNode.getCost();
12            visitedWords.add(currWord);
13            dictionary.remove(currWord);
14            List<String> nextWords = getValidWords(currWord, dictionary);
15            for (String nextWord : nextWords) {
16                int newCost = currCost + 1;
17                if(!visitedWords.contains(nextWord)){
18                    visitedWords.add(nextWord);
19                    dictionary.remove(nextWord);
20                    Node next = new Node(nextWord, newCost, currNode);
21                    priorityQueue.add(next);
22                    if(nextWord.equals(endWord)){
23
24                        return new SearchResult(next.reconstructPath(), visitedWords.size());
25                    }
26                }
27            }
28        }
29
30        return new SearchResult();
31    }
32 }
```

4.2 Algoritma Greedy Best First Search

Seperti yang bisa dilihat *Greedy Best First Search* yang saya buat menggunakan *Priority Queue* untuk menyimpan *node* yang dikunjungi dan diekspansi. Selain itu, saya juga memakai *Set* untuk *visited words*. Sebagian besar program yang dibuat sama seperti yang ada pada langkah-langkah algoritma. Penentuan *Cost* dari simpul adalah dengan menggunakan fungsi *heuristic()*.

```
1 public class GreedyBest extends Algorithm{
2     public SearchResult process(String startWord, String endWord, List<String> dictionary){
3         PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparingInt(Node::getCost));
4         Set<String> visitedWords = new HashSet<>();
5
6         priorityQueue.add(new Node(startWord, heuristic(startWord, endWord), null));
7
8         while(!priorityQueue.isEmpty()){
9             Node currNode = priorityQueue.poll();
10            String currWord = currNode.getWord();
11            visitedWords.add(currWord);
12            dictionary.remove(currWord);
13            List<String> nextWords = getValidWords(currWord, dictionary);
14            for(String nextWord : nextWords){
15                if(!visitedWords.contains(nextWord)){
16                    visitedWords.add(nextWord);
17                    dictionary.remove(nextWord);
18                    Node next = new Node(nextWord, heuristic(nextWord, endWord), currNode);
19                    priorityQueue.add(next);
20                    if (nextWord.equals(endWord)) {
21                        return new SearchResult(next.reconstructPath(), visitedWords.size());
22                    }
23                }
24            }
25        }
26        return new SearchResult();
27    }
28
29    public int heuristic(String word, String endWord){
30        int distance = 0;
31        for(int i = 0; i < word.length(); i++){
32            if(word.charAt(i) != endWord.charAt(i)){
33                distance += 1;
34            }
35        }
36        return distance;
37    }
38 }
```

4.3 Algoritma A*

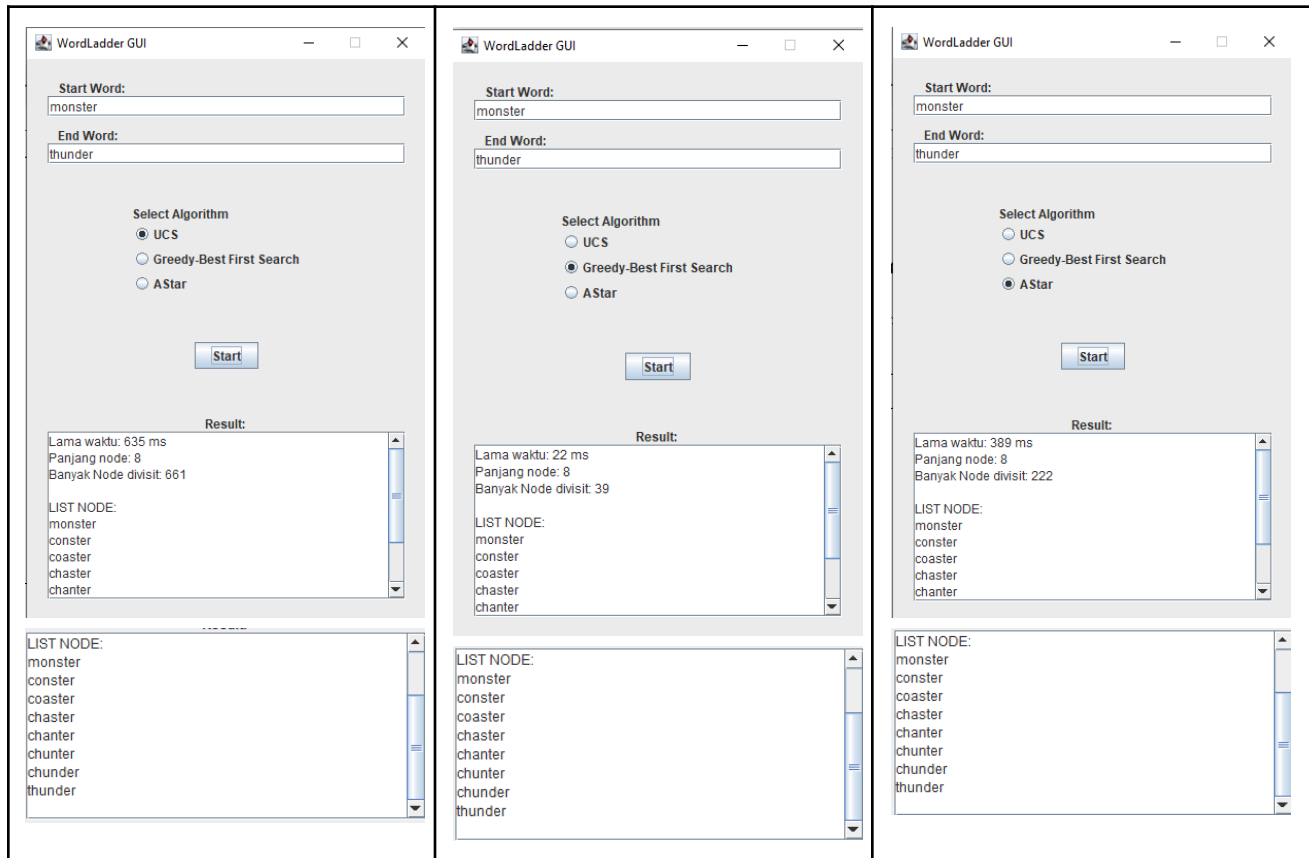
Seperti yang bisa dilihat A* yang saya buat menggunakan *Priority Queue* untuk menyimpan *node* yang dikunjungi dan diekspansi. Selain itu, saya juga memakai Set untuk *visited words*. Sebagian besar program yang dibuat sama seperti yang ada pada langkah-langkah algoritma. Penentuan *Cost* adalah dengan menggunakan $\text{currentCost} + 1 + \text{heuristic}()$.

```
1 public class Astar extends Algorithm{
2     public SearchResult process(String startWord, String endWord, List<String> dictionary){
3         PriorityQueue<Node> priorityQueue = new PriorityQueue<>(Comparator.comparing(Node::getCost));
4         Set<String> visitedWords = new HashSet<>();
5
6         GreedyBest gbfs = new GreedyBest();
7         priorityQueue.add(new Node(startWord, gbfs.heuristic(startWord, endWord), null));
8         while(!priorityQueue.isEmpty()){
9             Node currNode = priorityQueue.poll();
10            String currWord = currNode.getWord();
11            int currCost = currNode.getCost();
12            visitedWords.add(currWord);
13            dictionary.remove(currWord);
14            List<String> nextWords = getValidWords(currWord, dictionary);
15            for(String nextWord : nextWords){
16                int newCost = currCost + 1 + gbfs.heuristic(nextWord, endWord);
17                if(!visitedWords.contains(nextWord)) {
18                    visitedWords.add(nextWord);
19                    dictionary.remove(nextWord);
20                    Node next = new Node(nextWord, newCost, currNode);
21                    priorityQueue.add(next);
22                    if(nextWord.equals(endWord)) {
23                        return new SearchResult(next.reconstructPath(), visitedWords.size());
24                    }
25                }
26            }
27        }
28        return new SearchResult();
29    }
30 }
31
```


4.4 Pengujian

Berikut merupakan test-case yang dilakukan menggunakan kamus yang ada pada link berikut (<https://github.com/dwyl/english-words>):

1. monster -> thunder



2. cell -> atom

The image displays three sequential screenshots of the WordLadder GUI, showing the search process from 'cell' to 'atom' using different algorithms. Each window has a title bar 'WordLadder GUI' and standard window controls.

Window 1 (Left): UCS Algorithm

- Start Word: cell
- End Word: atom
- Select Algorithm: ☒ UCS, ☐ Greedy-Best First Search, ☐ AStar
- Start button
- Result section:
 - Lama waktu: 977 ms
 - Panjang node: 8
 - Banyak Node divisit: 13123
 - LIST NODE:
 - cell
 - coll
 - colp
 - coop
 - clap

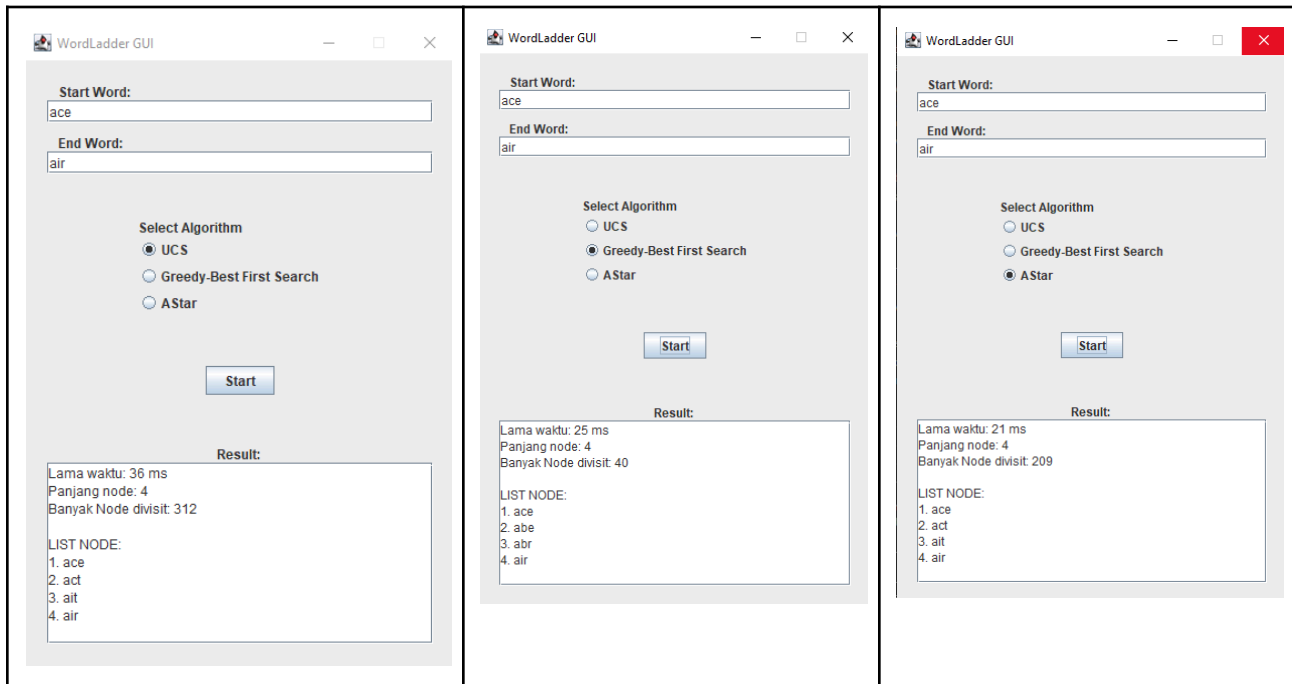
Window 2 (Middle): Greedy-Best First Search Algorithm

- Start Word: cell
- End Word: atom
- Select Algorithm: ☐ UCS, ☒ Greedy-Best First Search, ☐ AStar
- Start button
- Result section:
 - Lama waktu: 16 ms
 - Panjang node: 17
 - Banyak Node divisit: 198
 - LIST NODE:
 - cell
 - bell
 - bull
 - sull
 - sulu

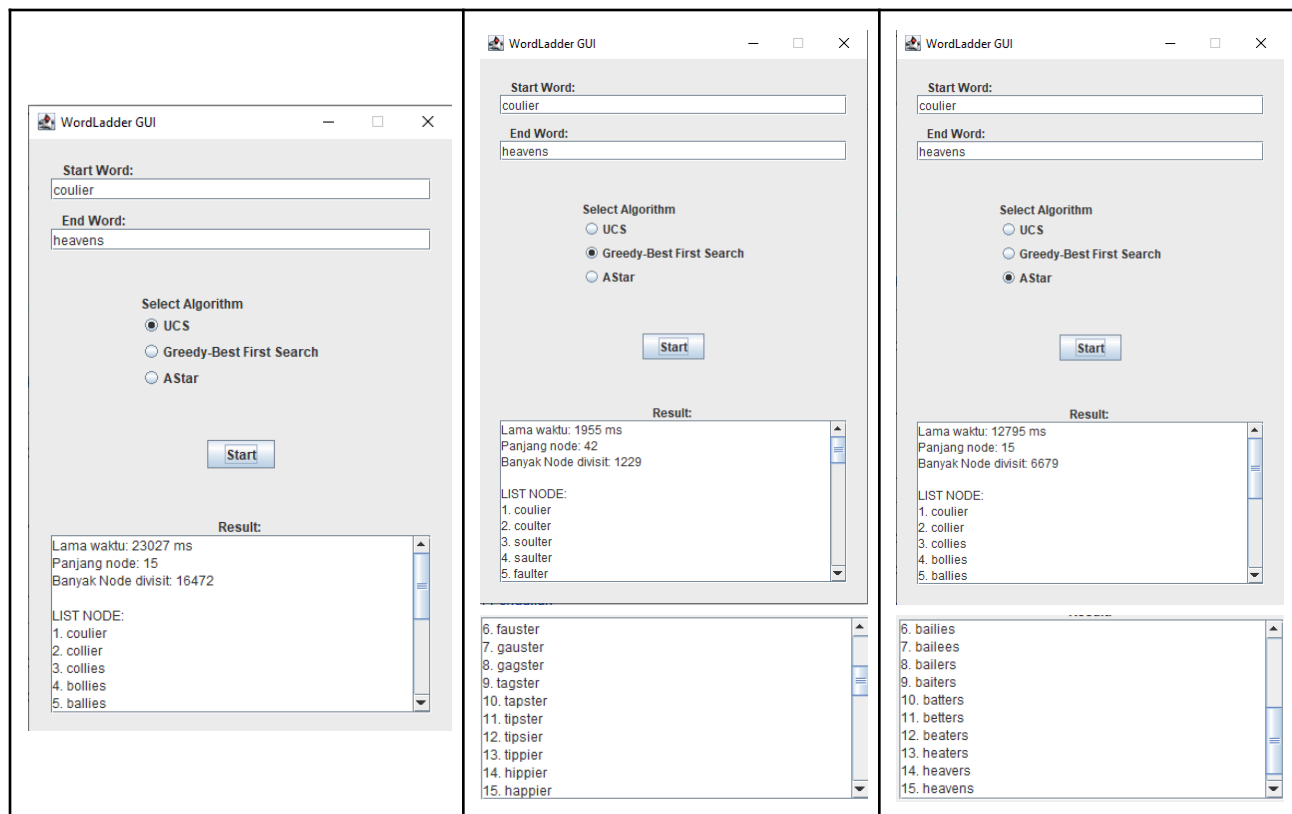
Window 3 (Right): AStar Algorithm

- Start Word: cell
- End Word: atom
- Select Algorithm: ☐ UCS, ☐ Greedy-Best First Search, ☒ AStar
- Start button
- Result section:
 - Lama waktu: 395 ms
 - Panjang node: 8
 - Banyak Node divisit: 5837
 - LIST NODE:
 - cell
 - coll
 - cool
 - coom
 - woom

3. ace -> air



4. coulier -> heavens

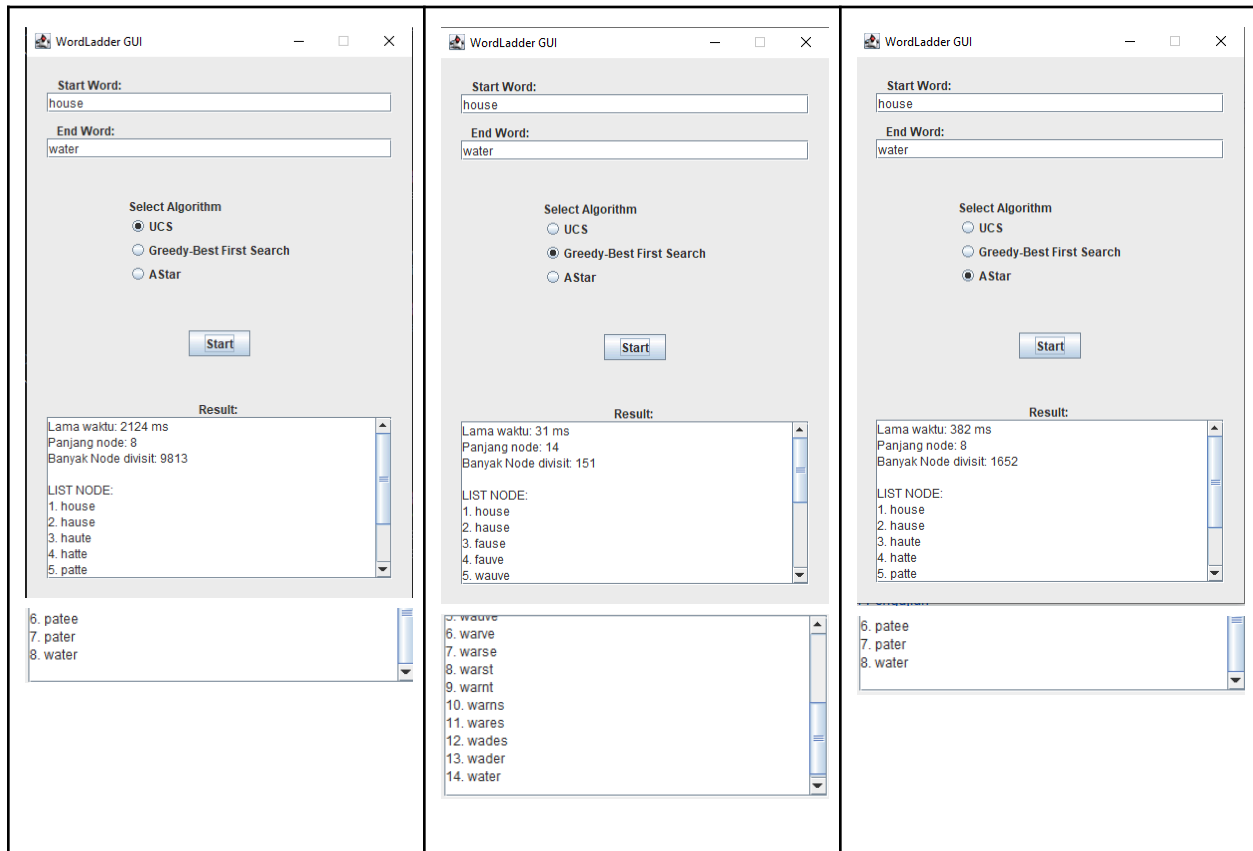


6. bailies 7. bailees 8. bailers 9. ballers 10. fallers 11. fellers 12. hellers 13. healers 14. heavers 15. heavens	16. harpier 17. harpies 18. harpins 19. harping 20. harding 21. herding 22. heading 23. heating 24. henting 25. hanting 26. hanking 27. hawking 28. hawkins 29. hawkies 30. hackies 31. hackees 32. hackers 33. hankers 34. hunkers 35. hunters 34. hunkers 35. hunters 36. hurters 37. hurlers 38. hullers 39. hellers 40. healers 41. heavers 42. heavens	
--	---	--

5. eat -> run

<p>WordLadder GUI</p> <p>Start Word: eat</p> <p>End Word: run</p> <p>Select Algorithm <input checked="" type="radio"/> UCS <input type="radio"/> Greedy-Best First Search <input type="radio"/> AStar </p> <p>Start</p> <p>Result:</p> <p>Lama waktu: 17 ms Panjang node: 4 Banyak Node divisit: 963</p> <p>LIST NODE: 1. eat 2. ean 3. ran 4. run </p>	<p>WordLadder GUI</p> <p>Start Word: eat</p> <p>End Word: run</p> <p>Select Algorithm <input type="radio"/> UCS <input checked="" type="radio"/> Greedy-Best First Search <input type="radio"/> AStar </p> <p>Start</p> <p>Result:</p> <p>Lama waktu: 5 ms Panjang node: 4 Banyak Node divisit: 71</p> <p>LIST NODE: 1. eat 2. ean 3. ran 4. run </p>	<p>WordLadder GUI</p> <p>Start Word: eat</p> <p>End Word: run</p> <p>Select Algorithm <input type="radio"/> UCS <input type="radio"/> Greedy-Best First Search <input checked="" type="radio"/> AStar </p> <p>Start</p> <p>Result:</p> <p>Lama waktu: 10 ms Panjang node: 4 Banyak Node divisit: 472</p> <p>LIST NODE: 1. eat 2. rat 3. rut 4. run </p>
--	--	--

6. house -> water



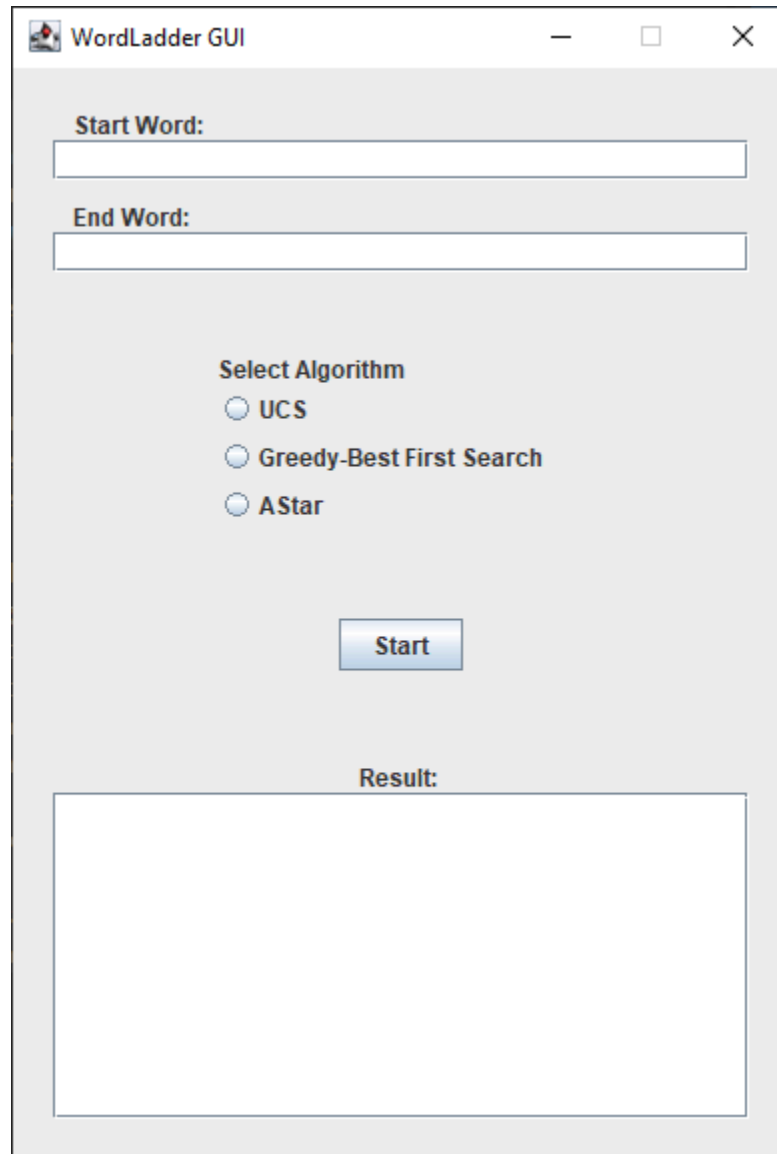
4.5 Analisis

Algoritma	Panjang Node	Kata divisit	Waktu eksekusi (ms)
1. monster -> thunder			
UCS	8	661	635
Greedy	8	39	22
A*	8	222	389
2. cell -> atom			
UCS	8	13123	977
Greedy	17	198	16
A*	8	5837	395
3. ace -> air			
UCS	4	312	36
Greedy	4	40	25
A*	4	205	21
4. coulier -> heavens			
UCS	15	16472	23037
Greedy	42	1229	1955
A*	15	6679	12795
5. cat -> run			
UCS	4	963	17
Greedy	4	71	5
A*	4	472	10
6. house -> water			
UCS	8	9813	2124
Greedy	14	151	31

A*	8	1852	382
----	---	------	-----

4.6 GUI

Pada program kali ini saya menggunakan GUI. Library GUI yang digunakan adalah swing dari Java. Berikut adalah tampilan dari GUI yang telah saya buat.



The screenshot shows a Java Swing window titled "WordLadder GUI". The window has a light gray background and contains the following elements:

- Start Word:** A text label followed by an empty text input field.
- End Word:** A text label followed by an empty text input field.
- Select Algorithm:** A text label followed by three radio button options:
 - ☐ UCS
 - ☐ Greedy-Best First Search
 - ☐ AStar
- Start:** A blue rectangular button with the text "Start" in white.
- Result:** A text label followed by a large, empty rectangular text area for displaying the output.

WordLadder GUI

Start Word:
monster

End Word:
thunder

Select Algorithm

☐ UCS

☐ Greedy-Best First Search

☒ AStar

Start

Result:

Lama waktu: 389 ms
Panjang node: 8
Banyak Node divisit: 222

LIST NODE:
monster
conster
coaster
chaster
chanter

BAB IV

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari ketiga algoritma yang telah diuji UCS, Greedy Best First Search, dan A* didapati bahwa UCS dan A* selalu memiliki panjang node yang sama karena UCS dan A* memberikan hasil yang optimal. Perbedaan dari mereka berdua adalah jumlah simpul yang dikunjungi dan waktu pemrosesan, tetapi hasil yang didapatkan atau panjang simpul masihlah sama. *Greedy Best First Search* memiliki eksekusi yang lebih cepat, tetapi tidak menjamin karena panjang simpul yang dihasilkan terkadang lebih panjang daripada algoritma UCS dan A*. Selain itu, heuristik yang digunakan oleh A* admissible. Admissible dalam konteks algoritma pencarian seperti A* merujuk pada sifat heuristik yang digunakan. Sebuah heuristik dikatakan admissible jika tidak pernah melebihi estimasi biaya dari solusi sebenarnya untuk mencapai tujuan (goal). Dengan kata lain, heuristik yang admissible tidak akan memberikan nilai yang lebih besar dari biaya sebenarnya untuk mencapai solusi terbaik

5.2 Saran

Bagi pembaca yang ingin melakukan apa yang telah penulis lakukan, saran penulis bisa mencoba di bahasa pemrograman lainnya untuk mencari perbedaan dari apa yang sudah penulis lakukan pada bahasa Java.

LAMPIRAN

- Tautan *repository* Github:
https://github.com/Indraswara/Tucil3_13522119
- Penyelesaian

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

DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>