

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 Fauzan Azhim

13522153

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2024

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1	
DESKRIPSI TUGAS.....	4
BAB 2	
LANDASAN TEORI.....	6
2.1 Dasar Teori.....	6
2.1.1 Algoritma Uniform-Cost Search (UCS).....	6
2.1.2 Algoritma A-Star (A*).....	6
2.1.3 Algoritma Greedy Best First Search (Greedy BFS).....	7
2.2 Bahasa Pemrograman Java.....	7
2.2.1 Java Swing.....	7
BAB 3	
ANALISIS PEMECAHAN MASALAH.....	8
3.1 Langkah - langkah Pemecahan Masalah.....	8
3.1.1 Algoritma Uniform-Cost Search (UCS).....	8
3.1.2 Algoritma A-Star (A*).....	9
3.1.3 Algoritma Greedy Best First Search (Greedy BFS).....	10
3.2 Proses Pemetaan Masalah Menjadi Elemen - Elemen Algoritma.....	11
3.2.1 Algoritma Uniform-Cost Search (UCS).....	11
3.2.1.1 Graf Berbobot.....	11
3.2.1.2 Queue.....	11
3.2.2.3 Set.....	12
3.2.2 Algoritma A-Star (A*).....	12
3.2.2.1 Graf Berbobot.....	12
3.2.2.2 Priority Queue.....	13
3.2.2.3 Set.....	14
3.2.3 Algoritma Greedy Best First Search (Greedy BFS).....	14
3.2.3.1 Graf Berbobot.....	14
3.2.3.2 Set.....	15
3.3 Fitur fitur fungsional tambahan.....	16
3.3.1 Graphical User Interface (GUI).....	16
3.3.2 Random Solvable Word.....	16
3.3.3 Random Word.....	17
3.3.4 Play Game.....	17
3.3.5 Dictionary.....	17
BAB 4	
IMPLEMENTASI DAN PENGUJIAN.....	18

4.1 Spesifikasi Teknis Program.....	18
4.1.1 Algoritma Uniform-Cost Search (UCS).....	18
4.1.1.1 Kelas UCS.....	18
4.1.1.2 Fungsi Solver.....	19
4.1.1.3 Prosedur Proses Simpul.....	19
4.1.2 Algoritma A-Star (A*).....	20
4.1.2.1 Kelas A-Star.....	20
4.1.2.2 Fungsi Solver.....	21
4.1.2.3 Fungsi Proses Node.....	22
4.1.3 Algoritma Greedy Best First Search (Greedy BFS).....	23
4.1.3.1 Kelas GBFS.....	23
4.1.3.2 Fungsi Solver.....	24
4.1.3.3 Fungsi Proses Node.....	24
4.1.4 Algoritma Lain.....	25
4.1.4.1 Hamming Distance.....	25
4.1.4.2 Levenshtein Distance.....	26
4.1.5 Parser.....	26
4.1.5.1 Kelas Parser.....	26
4.1.5.2 isWordExist.....	27
4.1.5.3 getWordList.....	27
4.1.5.4 getRandomWord.....	28
4.2 Tata Cara Penggunaan Program.....	29
4.3 Hasil Pengujian.....	30
4.3.1 Kasus 1.....	30
4.3.2 Kasus 2.....	30
4.3.3 Kasus 3.....	31
4.3.4 Kasus 4.....	32
4.3.5 Kasus 5.....	33
4.3.6 Kasus 6.....	35
4.4 Analisis Hasil.....	36
BAB 5	
KESIMPULAN DAN SARAN.....	37
5.1 Kesimpulan.....	37
5.2 Saran.....	38
5.3 Refleksi.....	38
LAMPIRAN.....	39
DAFTAR PUSTAKA.....	40

BAB 1

DESKRIPSI TUGAS

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, 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 solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.

How To Play

This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

Rules

Weave your way from the start word to the end word.
Each word you enter **can only change 1 letter** from the word above it.

Example

E	A	S	T
---	---	---	---

EAST is the start word, WEST is the end word

V	A	S	T
---	---	---	---

We changed E to V to make VAST

V	E	S	T
---	---	---	---

We changed A to E to make VEST

W	E	S	T
---	---	---	---

And we changed V to W to make WEST

W	E	S	T
---	---	---	---

Done!

Privacy
[Privacy Policy](#)

Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder
(Sumber: <https://wordwormdormdork.com/>)

Permainannya cukup sederhana bukan? Jika belum paham dengan peraturan permainannya, cobalah untuk memainkan permainannya pada link sumber di atas. Jika sudah paham dengan permainannya, sekarang adalah waktunya kalian untuk membuat sebuah solver permainan tersebut dengan harapan kita dapat menemukan solusi paling optimal untuk menyelesaikan permainan Word Ladder ini.

BAB 2

LANDASAN TEORI

2.1 Dasar Teori

2.1.1 Algoritma Uniform-Cost Search (UCS)

Algoritma Uniform Cost Search (UCS) merupakan sebuah algoritma uninformed search yang dapat digunakan untuk menentukan rute. Algoritma menerima data berisi simpul data serta rute antara simpul yang diberikan sebuah skor. Algoritma Uniform Cost Search bekerja dengan cara mempertimbangkan semua jalur yang mungkin dari titik awal ke titik tujuan. Untuk setiap langkah, algoritma ini memilih jalur dengan biaya terendah. Proses ini dilakukan secara berulang hingga mencapai titik tujuan atau tidak ada jalur lagi yang tersedia. Algoritma ini sering digunakan dalam masalah optimisasi dan perutean. Tujuan utama dari Uniform Cost Search adalah mencari jalur dengan biaya terendah dari titik awal ke titik tujuan.

2.1.2 Algoritma A-Star (A*)

Algoritma A* (A-Star) adalah algoritma pencarian yang digunakan dalam pemrograman komputer dan kecerdasan buatan untuk mencari jalur terpendek atau solusi optimal antara dua titik dalam graf atau ruang pencarian. Algoritma ini memadukan teknik pencarian breadth-first (pencarian melebar) dan heuristic (pemilihan berdasarkan perkiraan) untuk mencapai keseimbangan antara kecepatan dan optimalitas.

Algoritma A* bekerja dengan mengeksplorasi simpul-simpul (node) dalam graf secara berurutan, dimulai dari simpul awal dan bergerak ke simpul-simpul tetangga. Pada setiap langkah, algoritma memilih simpul berikutnya yang diharapkan memberikan jalur terpendek menuju tujuan berdasarkan estimasi heuristic.

Algoritma A* memiliki kelebihan dalam efisiensi dan kemampuan untuk menemukan jalur terpendek secara optimal jika heuristic yang digunakan adil dan konsisten. Oleh karena itu, algoritma ini sering digunakan dalam berbagai aplikasi, seperti perencanaan lintasan, permainan video, sistem navigasi, dan bidang lain yang membutuhkan pencarian jalur efisien.

2.1.3 Algoritma Greedy Best First Search (Greedy BFS)

Algoritma Greedy Best First Search adalah salah satu algoritma pencarian yang berbasis pada heuristik. Algoritma ini bekerja dengan mengevaluasi biaya setiap jalur yang mungkin dan kemudian memperluas jalur dengan biaya terendah. Proses ini diulangi sampai tujuan dicapai. Algoritma ini menggunakan fungsi heuristik untuk menentukan jalur mana yang paling menjanjikan.

Fungsi heuristik memperhitungkan biaya jalur saat ini dan perkiraan biaya jalur yang tersisa. Jika biaya jalur saat ini lebih rendah dari perkiraan biaya jalur yang tersisa, maka jalur saat ini yang dipilih. Proses ini diulangi sampai tujuan tercapai.

2.2 Bahasa Pemrograman Java

Java adalah bahasa pemrograman berorientasi objek dan platform perangkat lunak yang banyak digunakan yang bisa berjalan di miliaran perangkat, termasuk komputer notebook, perangkat seluler, konsol game, perangkat medis, dan banyak lainnya. Aturan dan sintaks Java didasarkan pada bahasa C dan C++.

2.2.1 Java Swing

Java Swing adalah toolkit widget untuk antarmuka grafis pengguna (GUI) Java yang mencakup satu set widget yang kaya. Ini adalah bagian dari Java Foundation Class (JFC) dan mencakup beberapa paket untuk mengembangkan aplikasi desktop yang kaya di Java. Swing sangat berguna untuk membuat aplikasi berbasis GUI di Java dan memberikan kontrol yang besar kepada pengembang atas tampilan dan perilaku aplikasi mereka.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah - langkah Pemecahan Masalah

3.1.1 Algoritma Uniform-Cost Search (UCS)

Algoritma pencarian Uniform-Cost Search (UCS) adalah algoritma pencarian lintas yang digunakan untuk mencari jalur terpendek dalam graf dengan bobot, di mana biaya untuk melewati setiap tepi dari simpul ke simpul lain adalah berbeda. UCS mirip dengan algoritma Dijkstra, tetapi UCS menggunakan pendekatan yang lebih hemat memori dengan hanya menyimpan simpul yang belum diproses dengan biaya terendah saat ini.

Langkah pertama, lakukan inisialisasi dengan simpul awal sebagai simpul saat ini. Tentukan bahwa biaya dari simpul awal ke dirinya sendiri adalah 0. Inisialisasi juga mencakup membuat daftar simpul yang belum diproses dan menginisialisasi struktur data yang diperlukan untuk melacak biaya terendah untuk mencapai setiap simpul.

Lakukan pencarian dengan loop sampai simpul tujuan ditemukan atau tidak ada simpul yang tersisa untuk diproses.

1. Ambil simpul yang memiliki biaya terendah dari daftar simpul yang belum diproses.
2. Jika simpul tersebut adalah simpul tujuan, maka pencarian selesai.
3. Jika tidak, ekspansi simpul tersebut dengan menambahkan semua simpul tetangga yang belum dieksplorasi ke daftar simpul yang belum diproses.
4. Untuk setiap tetangga yang ditambahkan, perbarui biaya terpendek yang diketahui untuk mencapainya jika perlu.
5. Tandai simpul saat ini sebagai sudah diproses.
6. Saat pencarian berakhir, jalur terpendek dari simpul awal ke simpul tujuan dapat direkonstruksi dengan mengikuti pointer parent dari simpul tujuan ke simpul awal.

UCS menjamin menemukan jalur terpendek karena secara iteratif mempertimbangkan setiap kemungkinan jalur dengan biaya terendah. Namun, UCS bisa menjadi lambat jika graf memiliki banyak simpul.

3.1.2 Algoritma A-Star (A*)

Algoritma A-Star menggunakan pendekatan heuristik yang menggabungkan biaya aktual dari titik awal ke simpul tertentu (dinyatakan oleh $g(n)$) dengan estimasi biaya yang tersisa dari simpul tersebut ke titik tujuan (dinyatakan oleh $h(n)$). Secara singkat, algoritma mencoba untuk meminimalkan fungsi $f(n) = g(n) + h(n)$, di mana $f(n)$ adalah biaya total yang diestimasi untuk mencapai tujuan melalui simpul tertentu.

Langkah-langkah algoritma A-Star adalah :

1. Mulai dengan memasukkan titik awal ke dalam himpunan terbuka dan mengatur biaya $g(\text{awal})$ menjadi 0.
2. Selama himpunan terbuka tidak kosong:
 - a. Pilih simpul dengan nilai f terkecil dari himpunan terbuka (simpul dengan perkiraan biaya total terkecil).
 - b. Jika simpul tersebut adalah titik tujuan, maka jalur telah ditemukan. Selesai.
 - c. Jika tidak, pindahkan simpul tersebut dari himpunan terbuka ke himpunan tertutup.
 - d. Periksa setiap simpul tetangga yang terhubung dengan simpul saat ini:
 - i. Jika simpul tetangga tidak ada dalam himpunan terbuka, tambahkan ke himpunan terbuka dan tentukan $g(n)$ serta $h(n)$ untuknya.
 - ii. Jika simpul tetangga sudah ada dalam himpunan terbuka, periksa apakah jalur baru ke simpul tersebut lebih baik (dengan membandingkan $g(n)$ yang diestimasi). Jika ya, update nilai $g(n)$ dan $h(n)$ untuk simpul tetangga dan atur simpul tersebut sebagai tetangga yang terhubung dengan jalur saat ini.
3. Jika himpunan terbuka kosong dan titik tujuan belum ditemukan, jalur tidak ada.

3.1.3 Algoritma Greedy Best First Search (Greedy BFS)

Algoritma Greedy Best First Search (Greedy BFS) adalah salah satu algoritma pencarian jalur yang digunakan dalam kecerdasan buatan dan pemecahan masalah optimasi. Seperti namanya, Greedy BFS adalah pendekatan *Greedy* yang selalu memilih simpul yang paling dekat dengan tujuan saat ini tanpa mempertimbangkan biaya total dari simpul tersebut.

Langkah-langkah umum algoritma Greedy Best First Search:

1. Mulai dengan memasukkan titik awal ke dalam himpunan terbuka.
2. Selama simpul yang akan di proses masih ada:
 - a. Cek jika simpul tersebut adalah titik tujuan, maka jalur telah ditemukan. Selesai.
 - b. Jika tidak, pindahkan simpul tersebut dari himpunan terbuka ke himpunan tertutup.
 - c. Periksa setiap simpul tetangga yang terhubung dengan simpul saat ini:
 - i. Jika simpul tetangga belum ada dalam himpunan tertutup, tambahkan ke larik simpul dan tentukan nilai heuristik untuknya.
 - d. Ubah simpul yang akan diproses dengan simpul terbaik pada larik simpul.
3. Jika simpul yang akan diproses tidak ada dan titik tujuan belum ditemukan, jalur tidak ada.

Algoritma GBFS cenderung cepat tetapi tidak selalu menghasilkan solusi yang optimal karena kecenderungannya untuk fokus pada simpul yang dekat dengan tujuan tanpa mempertimbangkan biaya total. Oleh karena itu, algoritma ini sering digunakan dalam situasi di mana solusi cepat diperlukan dan kualitas solusi tidak menjadi prioritas utama.

3.2 Proses Pemetaan Masalah Menjadi Elemen - Elemen Algoritma

3.2.1 Algoritma Uniform-Cost Search (UCS)

3.2.1.1 Graf Berbobot

Graf terdiri dari beberapa simpul dan sisi. Dua Simpul dikatakan ketetanggaan, jika dua simpul ini dihubungkan oleh sisi secara langsung. Kemudian salah satu bentuk dari graf yaitu graf tidak berbobot dan graf berbobot. Graf berbobot merupakan graf yang setiap sisinya memiliki sebuah bobot berupa state/angka.

Pada algoritma Uniform-Cost Search, bobot pada graf adalah nilai heuristik dari UCS yaitu harga dari simpul awal ke simpul saat ini.

$$F(n) = g(n)$$

Untuk word ladder solver cost pada suatu simpul adalah sama saja dengan kedalaman pada Breadth First Search.

3.2.1.2 Queue

Queue merupakan salah satu bentuk struktur data yang diilustrasikan sebagai antrian dengan prioritas tertentu seperti prioritas angka terbesar dan terkecil. Dalam Priority Queue memiliki ciri seperti setiap elemen dalam antrian memiliki prioritas yang dapat berupa integer atau lainnya. Kemudian setiap elemen dengan prioritas tertinggi akan diambil (Dequeue) terlebih dahulu sebelum prioritas lebih rendah. Jika dua elemen atau lebih memiliki prioritas yang sama, pengambilan elemen akan dilakukan sesuai dengan urutan dari antrian. Bentuk dari Priority Queue dapat berupa array atau list atau binary heap.

Queue dalam algoritma Uniform-Cost Search digunakan untuk menentukan urutan simpul yang akan dikunjungi. Pada Uniform-Cost Search seharusnya struktur data Priority Queue digunakan dibandingkan struktur data

Queue, namun karena untuk setiap node yang akan dikunjungi pasti akan memiliki cost yang sama yaitu satu dan Priority Queue jika menemukan cost yang sama maka akan memproses yang terlebih dahulu ditemukan. Ini berarti algoritma Uniform-Cost Search sama saja dengan Breadth First Search pada Word Ladder Solver.

3.2.2.3 Set

Set adalah struktur data dalam pemrograman komputer yang digunakan untuk menyimpan kumpulan elemen unik tanpa adanya urutan tertentu. Artinya, set tidak mengizinkan duplikat, dan elemen-elemennya tidak diatur dalam urutan tertentu seperti yang terjadi dalam array atau list.

Dalam set, setiap elemen dianggap sebagai entitas tunggal, dan set berfungsi untuk menyimpan kumpulan elemen tanpa peduli dengan urutan di mana mereka dimasukkan atau berapa kali elemen tersebut muncul.

Set pada Algoritma Uniform-Cost Search digunakan untuk mencatat simpul mana saja yang sudah pernah dikunjungi sehingga tidak akan dikunjungi lagi. Hal ini berguna untuk menghindari pengolahan ulang node yang sama dan juga membantu dalam pengecekan siklus dalam graf.

3.2.2 Algoritma A-Star (A*)

3.2.2.1 Graf Berbobot

Graf terdiri dari beberapa simpul dan sisi. Dua Simpul dikatakan ketetanggaan, jika dua simpul ini dihubungkan oleh sisi secara langsung. Kemudian salah satu bentuk dari graf yaitu graf tidak berbobot dan graf berbobot. Graf berbobot merupakan graf yang setiap sisinya memiliki sebuah bobot berupa state/angka.

Pada algoritma A-Star, bobot pada graf adalah nilai heuristik dari A-Star yaitu menggabungkan biaya aktual dari titik awal ke simpul tertentu (dinyatakan

oleh $g(n)$) dengan estimasi biaya yang tersisa dari simpul tersebut ke titik tujuan (dinyatakan oleh $h(n)$).

$$F(n) = g(n) + h(n)$$

Nilai $g(n)$ sama seperti pada algoritma UCS yaitu kedalaman simpul tersebut dari simpul awal. Nilai $h(n)$ dihitung melalui algoritma *hamming distance*. Algoritma *hamming distance* adalah algoritma yang menghitung jumlah perbedaan huruf pada letak yang sama pada dua buah kalimat yang panjangnya sama.

Nilai heuristik perkiraan pada algoritma A-Star yang diterapkan pada word ladder solver ini bersifat *admissible*. Hal ini dikarenakan nilai $F(n)$ akan selalu lebih kecil atau sama dengan nilai / cost aslinya. Definisi cost yang di buat pada A-Star word ladder solver adalah jumlah input yang harus dimasukkan untuk bisa mencapai suatu simpul. Maksimal perubahan huruf untuk bisa pindah ke simpul lain hanyalah satu huruf. Hamming Distance yang diterapkan adalah antara node tersebut ke simpul akhir. Oleh karena itu heuristik A-Star akan selalu kurang dari atau sama dengan nilai aslinya.

3.2.2.2 Priority Queue

Priority Queue adalah struktur data khusus yang memungkinkan kita menyimpan elemen-elemen dengan nilai prioritas tertentu. Konsep dasar Priority Queue mirip dengan antrian biasa, namun perbedaannya adalah setiap elemen dalam Priority Queue memiliki nilai prioritas yang menentukan urutan elemen saat diambil. Elemen dengan nilai prioritas tertinggi akan diambil terlebih dahulu, bukan elemen yang paling lama berada dalam antrian. Jika ada elemen yang memiliki prioritas yang sama maka yang dijalankan terlebih dahulu adalah yang masuk lebih dulu.

Pada Algoritma A-Star, nilai prioritas adalah nilai heuristik pada A-Star, yaitu $F(n)$. Pada priority queue yang digunakan untuk mengatur simpul pertama yang diproses adalah yang memiliki nilai $F(n)$ terkecil. $F(n)$ terkecil ini

menandakan bahwa node ini adalah node yang kira-kira jarak dari awal sampai akhirnya terpendek.

3.2.2.3 Set

Set adalah struktur data dalam pemrograman komputer yang digunakan untuk menyimpan kumpulan elemen unik tanpa adanya urutan tertentu. Artinya, set tidak mengizinkan duplikat, dan elemen-elemennya tidak diatur dalam urutan tertentu seperti yang terjadi dalam array atau list.

Dalam set, setiap elemen dianggap sebagai entitas tunggal, dan set berfungsi untuk menyimpan kumpulan elemen tanpa peduli dengan urutan di mana mereka dimasukkan atau berapa kali elemen tersebut muncul.

Set pada Algoritma A-Star digunakan untuk mencatat simpul mana saja yang sudah pernah dikunjungi sehingga tidak akan dikunjungi lagi. Hal ini berguna untuk menghindari pengolahan ulang node yang sama dan juga membantu dalam pengecekan siklus dalam graf.

3.2.3 Algoritma Greedy Best First Search (Greedy BFS)

3.2.3.1 Graf Berbobot

Graf terdiri dari beberapa simpul dan sisi. Dua Simpul dikatakan ketetanggaan, jika dua simpul ini dihubungkan oleh sisi secara langsung. Kemudian salah satu bentuk dari graf yaitu graf tidak berbobot dan graf berbobot. Graf berbobot merupakan graf yang setiap sisinya memiliki sebuah bobot berupa state/angka.

Pada algoritma Greedy Best First Search, bobot pada graf adalah nilai heuristik dari Greedy BFS yaitu estimasi jarak simpul ke simpul akhir.

$$F(n) = h(n)$$

Untuk word ladder solver cost pada suatu simpul adalah sama saja dengan nilai dari algoritma *hamming distance*, yaitu banyak huruf yang berbeda pada lokasi yang sama di dua kalimat yang panjangnya sama.

3.2.3.2 Set

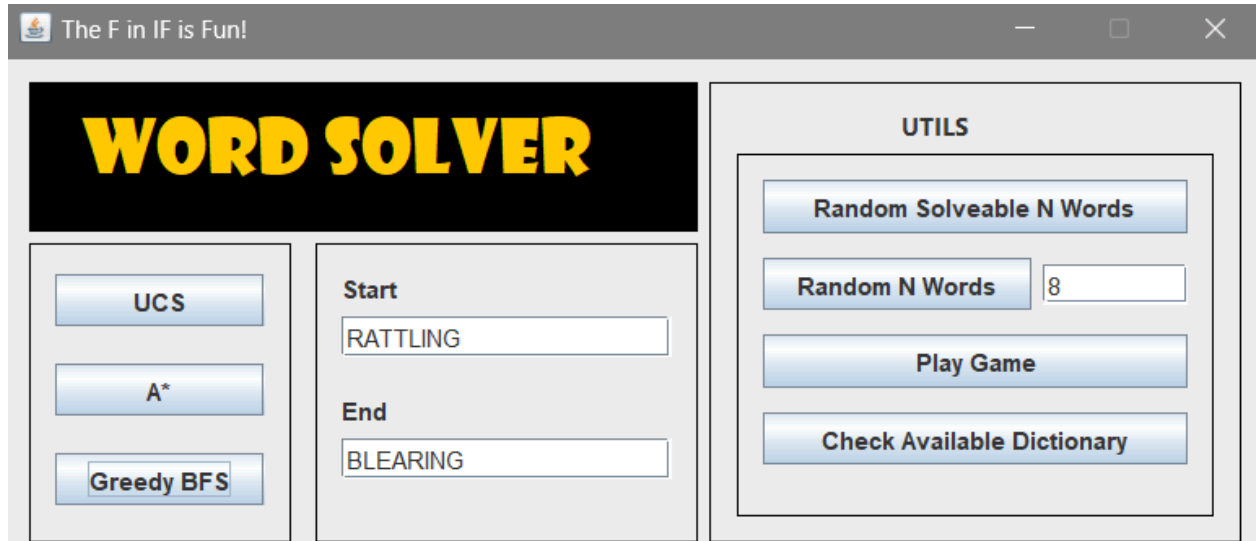
Set adalah struktur data dalam pemrograman komputer yang digunakan untuk menyimpan kumpulan elemen unik tanpa adanya urutan tertentu. Artinya, set tidak mengizinkan duplikat, dan elemen-elemennya tidak diatur dalam urutan tertentu seperti yang terjadi dalam array atau list.

Dalam set, setiap elemen dianggap sebagai entitas tunggal, dan set berfungsi untuk menyimpan kumpulan elemen tanpa peduli dengan urutan di mana mereka dimasukkan atau berapa kali elemen tersebut muncul.

Set pada Algoritma Greedy Best First Search digunakan untuk mencatat simpul mana saja yang sudah pernah dikunjungi sehingga tidak akan dikunjungi lagi. Hal ini berguna untuk menghindari pengolahan ulang node yang sama dan juga membantu dalam pengecekan siklus dalam graf.

3.3 Fitur fitur fungsional tambahan

3.3.1 Graphical User Interface (GUI)



Gambar 3.1 Tampilan Utama GUI

Fitur fungsional pertama adalah adanya Antarmuka Pengguna Grafis (APG). Fitur ini dibuat agar pengguna tidak bosan dan menambah keindahan dari program. Fitur ini juga dapat menggambarkan hasil atau result dengan sangat baik dan membuat fitur-fitur tambahan lainnya terlihat sangat bagus.

3.3.2 Random Solvable Word

Fitur fungsional kedua adalah adanya fungsi untuk melakukan randomisasi kata-kata awal dan akhir yang mempunyai jalur. Fungsi ini dibuat untuk memudahkan testing antara fungsi-fungsi *solver*. Untuk memverifikasi apakah kata awal dan akhir yang dipilih mempunyai jalur, saya menggunakan fungsi solver A-Star. Karena A-Star adalah algoritma yang optimal dan cepat dibandingkan algoritma UCS dan Greedy BFS.

3.3.3 Random Word

Fitur fungsional ketiga adalah adanya fungsi untuk melakukan randomisasi kata-kata awal dan akhir. Mencari kata-kata random yang ada sesuai input panjang kata yang diinginkan.

3.3.4 Play Game

Fitur fungsional ke-empat adalah adanya permainan yang dapat dimainkan. Karena game yang ada pada website / online tidak memiliki kamus yang sama dengan program yang saya punya, maka saya memutuskan untuk membuat game fungsional sendiri untuk bisa melakukan testing lanjutan dan mencoba menikmati gamenya.

3.3.5 Dictionary

Fitur fungsional ke-lima adalah adanya fitur mencari kata dalam kamus yang dimiliki. Fitur ini dibuat karena adanya beberapa kata yang tidak dimengerti dan ingin mengetahui apakah memang ada artinya. Selain itu fitur ini memudahkan bagi pemain yang tidak memiliki wawasan yang luas terhadap Bahasa Inggris (Amerika).

BAB 4

IMPLEMENTASI DAN PENGUJIAN

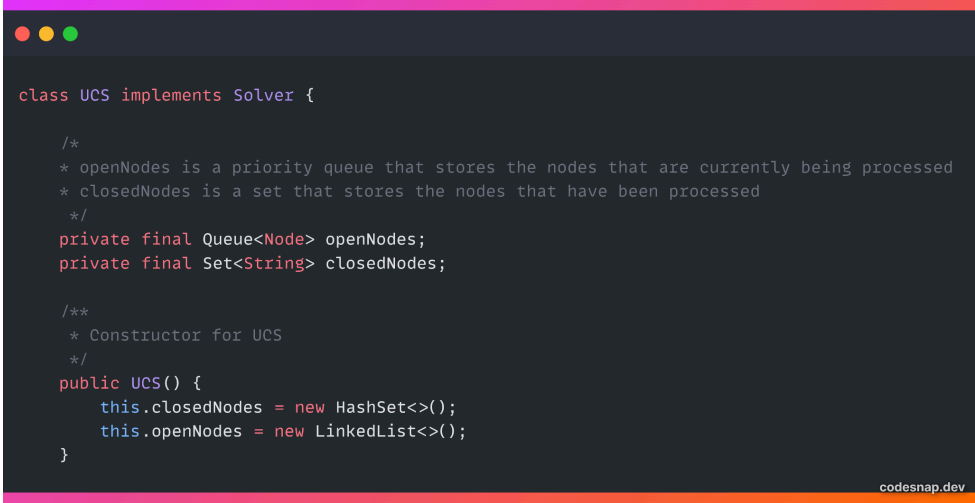
4.1 Spesifikasi Teknis Program

4.1.1 Algoritma Uniform-Cost Search (UCS)

4.1.1.1 Kelas UCS

Kelas Uniform-Cost Search mengimplementasikan interface Solver. Interface Solver sendiri mewajibkan membuat fungsi Solver yang menerima dua buah parameter string dan mengeluarkan kelas Result. Kelas UCS mempunyai dua variabel private:

1. openNodes <Queue> : Penyimpanan nodes yang akan diproses
2. closedNodes <Set> : Penyimpanan nama nodes yang sudah diproses



```
class UCS implements Solver {  
  
    /*  
    * openNodes is a priority queue that stores the nodes that are currently being processed  
    * closedNodes is a set that stores the nodes that have been processed  
    */  
    private final Queue<Node> openNodes;  
    private final Set<String> closedNodes;  
  
    /**  
    * Constructor for UCS  
    */  
    public UCS() {  
        this.closedNodes = new HashSet<>();  
        this.openNodes = new LinkedList<>();  
    }  
}
```

Gambar 4.1 Codesnap kelas UCS

4.1.1.2 Fungsi Solver

Fungsi utama solver menerima parameter dua buah string yaitu start dan end. Fungsi ini akan menjadi fungsi utama menjalankan algoritma mencari hasil penyelesaian.



```
public Result solve(String start, String end) {
    System.gc();
    long timeStart = System.nanoTime();
    int memoryStart = (int) ((Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / 1024);

    openNodes.add(new Node(start));

    while (!openNodes.isEmpty()) {
        Node current = openNodes.poll();

        if (current.getWord().equals(end)){
            int memory = Result.getMemoryUsed(memoryStart);
            return new Result(Result.getExecutionTime(timeStart), memory, Node.getPath(current) , closedNodes.size());
        }

        else {
            closedNodes.add(current.getWord());
            processNode(current);
        }
    }

    System.gc();
    return null;
}
```

Gambar 4.2 Codesnap kelas UCS

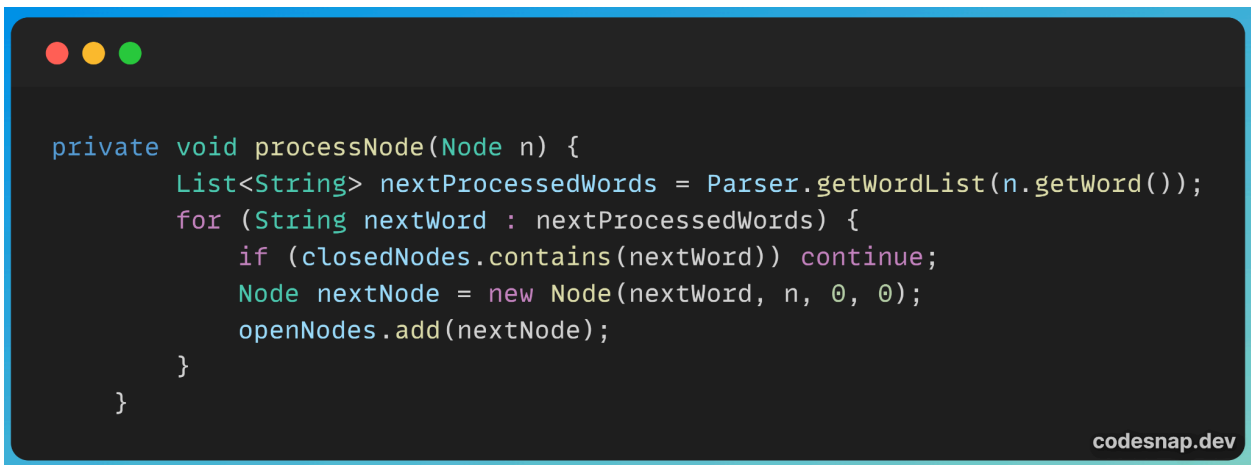
Pada 3 baris pertama kita akan melakukan inisialisasi untuk membuat hasil memory dan waktu yang digunakan selama proses pencarian hasil. Lalu tambahkan node awal ke dalam queue.

Lalu proses node pertama yang ada pada queue, jika node yang diproses sama dengan node akhir maka hentikan loop. Jika tidak maka jalankan fungsi ProsesNode dan tambahkan node saat ini ke node yang sudah diproses.

Jika queue sampai kosong maka jawaban tidak ditemukan dan fungsi akan mengembalikan NULL.

4.1.1.3 Prosedur Proses Simpul

Prosedur ini menerima node yang akan diproses. Node yang akan diproses akan diambil tetangganya pada fungsi “Parser.getWordList”. Untuk setiap tetangga yang ditemukan akan dicek apakah node ini sudah dikunjungi atau belum. Kalau belum maka akan dimasukkan pada queue openNodes.



```
private void processNode(Node n) {  
    List<String> nextProcessedWords = Parser.getWordList(n.getWord());  
    for (String nextWord : nextProcessedWords) {  
        if (closedNodes.contains(nextWord)) continue;  
        Node nextNode = new Node(nextWord, n, 0, 0);  
        openNodes.add(nextNode);  
    }  
}
```

Gambar 4.3 Snapcode Fungsi Proses Simpul UCS

4.1.2 Algoritma A-Star (A*)

4.1.2.1 Kelas A-Star

Kelas Uniform-Cost Search mengimplementasikan interface Solver. Interface Solver sendiri mewajibkan membuat fungsi Solver yang menerima dua buah parameter string dan mengeluarkan kelas Result. Kelas UCS mempunyai dua variabel private:

1. penNodes <Queue> : Penyimpanan nodes yang akan diproses
2. closedNodes <Set> : Penyimpanan nama nodes yang sudah diproses

```

public class A_star implements Solver {
    /*
     * openList adalah priority queue yang menyimpan node-node yang akan diakses
     * closedList adalah set yang menyimpan node-node yang sudah diakses
     * end adalah kata akhir yang ingin dicapai
     */
    private final PriorityQueue<Node> openList;
    private final Set<String> closedList;
    private String end;

    /**
     * Constructor untuk A*
     */
    public A_star() {
        this.openList = new PriorityQueue<>(Comparator.comparingDouble(Node::getF));
        this.closedList = new HashSet<>();
    }
}

```

Gambar 4.4 Kelas A_star

PriorityQueue pada openList telah diinisialisasi untuk membandingkan nilai F pada kelas Node. Sehingga node yang memiliki nilai F terendahlah yang akan diproses terlebih dahulu.

4.1.2.2 Fungsi Solver

Fungsi utama solver menerima parameter dua buah string yaitu start dan end. Fungsi ini akan menjadi fungsi utama menjalankan algoritma mencari hasil penyelesaian.

```

@Override
public Result solve(String start, String end){
    // Setup
    this.end = end;
    System.gc();
    int memoryStart = (int) ((Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / 1024);
    long startTime = System.nanoTime();

    // Start
    openList.add(new Node(start));

    while (!openList.isEmpty()) {
        Node current = openList.poll();
        closedList.add(current.getWord());

        if (current.getWord().equals(end)) {
            int memoryUsed = Result.getMemoryUsed(memoryStart);
            Result result = new Result(Result.getExecutionTime(startTime), memoryUsed, Node.getPath(current), closedList.size());
            System.gc();
            return result;
        } else {
            ProcessNode(current);
        }
    }
    return null;
}

```

Gambar 4.5 Fungsi Solve A_star

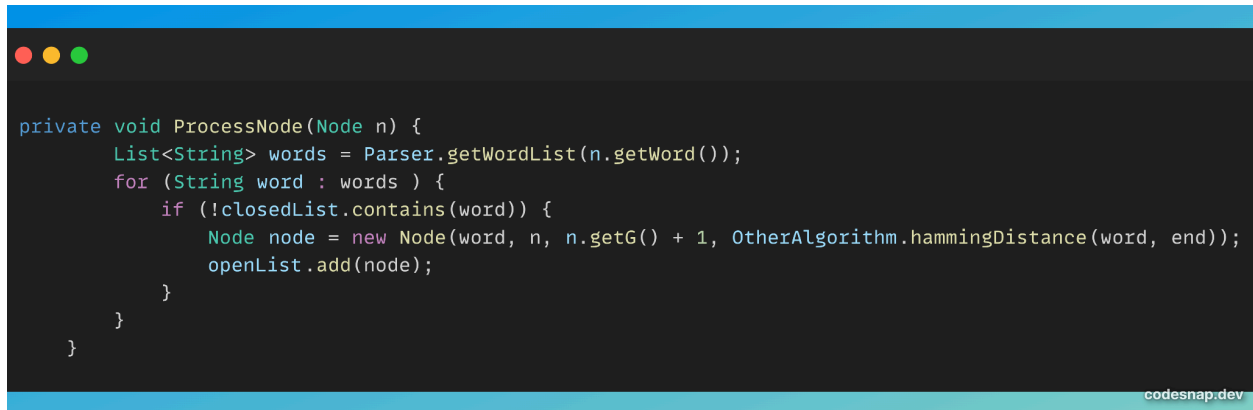
Pada 4 Baris kode pertama kita akan melakukan setup untuk memperhitungkan berapa jumlah memory yang digunakan selama algoritma berjalan dan berapa lama algoritma berjalan hingga selesai.

Pertama, tambahkan node awal ke dalam queue. Selanjutnya akan diambil node pada PriorityQueue paling depan untuk di proses, jika node yang akan diproses bukan node akhir maka jalankan fungsi ProsesNode. Ulangi sampai queue kosong.

Jika queue sampai kosong maka jawaban tidak ditemukan dan fungsi akan mengembalikan NULL.

4.1.2.3 Fungsi Proses Node

Prosedur ini menerima node yang akan diproses. Node yang akan diproses akan diambil tetangganya pada fungsi “Parser.getWordList”. Untuk setiap tetangga yang ditemukan akan dicek apakah node ini sudah dikunjungi atau belum. Kalau belum maka akan dimasukkan pada queue openNodes.



```

private void ProcessNode(Node n) {
    List<String> words = Parser.getWordList(n.getWord());
    for (String word : words) {
        if (!closedList.contains(word)) {
            Node node = new Node(word, n, n.getG() + 1, OtherAlgorithm.hammingDistance(word, end));
            openList.add(node);
        }
    }
}

```

Gambar 4.6 Prosedur Proses Node A_Star

4.1.3 Algoritma Greedy Best First Search (Greedy BFS)

4.1.3.1 Kelas GBFS

Kelas Uniform-Cost Search mengimplementasikan interface Solver. Interface Solver sendiri mewajibkan membuat fungsi Solver yang menerima dua buah parameter string dan mengeluarkan kelas Result. Kelas UCS mempunyai dua variabel private:

1. closedNodes <Set> : Penyimpanan nama nodes yang sudah diproses



```

public class GreedyBFS implements Solver {

    /*
     * closedList adalah set yang menyimpan node-node yang sudah diakses
     * end adalah kata akhir yang ingin dicapai
     */
    private final Set<String> closedList;
    private String end;

    /**
     * Constructor untuk GreedyBFS
     */
    public GreedyBFS() {
        this.closedList = new HashSet<>();
    }
}

```

Gambar 4.7 Kelas GreedyBFS

PriorityQueue pada openList diinisialisasi agar membandingkan nilai $g(n)$ dari node yang masuk.

4.1.3.2 Fungsi Solver

Fungsi utama solver menerima parameter dua buah string yaitu start dan end. Fungsi ini akan menjadi fungsi utama menjalankan algoritma mencari hasil penyelesaian.



```
public Result solve(String start, String end) {
    System.gc();
    this.end = end;
    int memoryStart = (int) ((Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()) / 1024);
    long startTime = System.nanoTime();

    Node current = new Node(start);

    while (true) {

        if (current == null) {
            int memoryUsed = Result.getMemoryUsed(memoryStart);
            return new Result(Result.getExecutionTime(startTime), memoryUsed, null, closedList.size());
        }

        if (current.getWord().equals(end)) {
            int memoryUsed = Result.getMemoryUsed(memoryStart);
            return new Result(Result.getExecutionTime(startTime), memoryUsed, Node.getPath(current), closedList.size());
        } else {
            closedList.add(current.getWord());
            current = ProcessNode(current);
        }
    }
}
```

Gambar 4.7 Fungsi Solver Greedy BFS

Pada 4 Baris kode pertama kita akan melakukan setup untuk memperhitungkan berapa jumlah memory yang digunakan selama algoritma berjalan dan berapa lama algoritma berjalan hingga selesai.

Pertama, jadikan node awal sebagai node yang akan diproses. Jika node yang dimasukkan sama dengan akhir maka keluarkan hasil jika tidak proses node.

Jika hasil node yang diproses adalah NULL maka path tidak ditemukan dan fungsi akan mengembalikan hasil.

4.1.3.3 Fungsi Proses Node

Prosedur ini menerima node yang akan diproses. Node yang akan diproses akan diambil tetangganya pada fungsi “Parser.getWordList”. Untuk setiap tetangga yang ditemukan akan dicek apakah node ini sudah dikunjungi atau

belum. Kalau belum maka akan dimasukkan pada priority queue openList. Pada openlist ini sudah diset bahwa yang paling depan adalah yang nilai $f(n)$ nya paling rendah. Penggunaan PriorityQueue ini dilakukan untuk mempermudah mendapatkan node dengan nilai $f(n)$ terendah. Jika ada node pada openList maka jadikan node pertama sebagai keluaran jika tidak maka keluarkan null.

```
private Node ProcessNode(Node current) {
    PriorityQueue<Node> openList = new PriorityQueue<>(Comparator.comparingDouble(Node::getF));
    for (String nextWord : Parser.wordList.get(current.getWord())) {
        if (!closedList.contains(nextWord)) {
            Node next = new Node(nextWord, current, OtherAlgorithm.hammingDistance(nextWord, end), 0);
            openList.add(next);
        }
    }

    if (!openList.isEmpty()) {
        return openList.poll();
    }

    return null;
}
```

Gambar 4.9 Proses Node Greedy BFS

4.1.4 Algoritma Lain

4.1.4.1 Hamming Distance

Fungsi Hamming Distance adalah sebuah fungsi untuk mengukur perbedaan antara dua deretan kalimat yang memiliki panjang yang sama. Metrik ini mengukur jumlah posisi di mana simbol-simbol yang sesuai di kedua kalimat. Fungsi Hamming Distance ini digunakan untuk menghitung nilai $g(n)$ yaitu estimasi jarak dari simpul tertentu untuk mencapai simpul akhir.

```
public static int hammingDistance(String word1, String word2) {
    int count = 0;
    for (int i = 0; i < word1.length(); i++) {
        if (word1.charAt(i) != word2.charAt(i)) {
            count++;
        }
    }
    return count;
}
```

Gambar 4.10 Fungsi HammingDistance

4.1.4.2 Levenshtein Distance

Levenshtein distance adalah sebuah metrik yang digunakan untuk mengukur seberapa berbeda dua rangkaian teks. Jarak ini diukur dengan jumlah operasi minimum yang diperlukan untuk mengubah satu rangkaian menjadi yang lainnya. Operasi yang diperbolehkan biasanya terdiri dari penyisipan (insertion), penghapusan (deletion), atau penggantian (substitution) satu karakter.

```
static int getLevenshteinDistance(String x, String y) {
    int[][] dp = new int[x.length() + 1][y.length() + 1];

    // To check how many characters need to be added to x to make it y
    for (int i = 0; i ≤ x.length(); i++) {
        dp[i][0] = i;
    }
    // To check how many characters need to be added to y to make it x
    for (int j = 0; j ≤ y.length(); j++) {
        dp[0][j] = j;
    }

    // Fill the dp table
    for (int i = 1; i ≤ x.length(); i++) {
        for (int j = 1; j ≤ y.length(); j++) {
            int substitutionCost = x.charAt(i - 1) == y.charAt(j - 1) ? 0 : 1;
            dp[i][j] = Math.min(Math.min(dp[i - 1][j - 1] + substitutionCost, dp[i - 1][j] + 1), dp[i][j - 1] + 1);
        }
    }
    return dp[x.length()][y.length()];
}
```

4.1.5 Parser

4.1.5.1 Kelas Parser

```
public class Parser {

    /*
     * wordList adalah map yang menyimpan kata-kata yang ada pada file
     * dictionary adalah list yang menyimpan kata-kata dan definisinya
     * LOGGER adalah logger yang digunakan untuk mencatat log
     */
    private static final Logger LOGGER = Logger.getLogger(Parser.class.getName());
    public static Map<String, List<String>> wordList = new HashMap<>();
    public static List<List<SimpleEntry<String, String>>> dictionary;
```

Kelas parser berguna untuk memasukkan data-data yang diperlukan untuk melakukan pencarian pada algoritma word ladder solver. Kelas parser sendiri memuat Map yang digunakan untuk mengakses tetangga apa saja yang ada pada suatu node. Kelas ini juga mempunyai dictionary yang menyimpan seluruh kalimat dan definisinya jika ada.

4.1.5.2 isWordExist

```
/**
 * Fungsi untuk mengecek apakah kata ada pada kamus
 * @param word kata yang ingin dicek
 * @return true jika kata tidak ada pada kamus, false jika kata ada pada kamus
 */
public static boolean isWordNotExist(String word) {
    return !wordList.containsKey(word);
}
```

4.1.5.3 getWordList

```
/**
 * Fungsi untuk mendapatkan list kata yang berjarak satu karakter dari kata yang diberikan
 * @param word kata yang ingin dicari
 * @return list kata yang berjarak satu karakter dari kata yang diberikan
 */
public static List<String> getWordList(String word) {
    return wordList.get(word);
}
```

4.1.5.4 getRandomWord

```
/**
 * Fungsi untuk mendapatkan kata acak dari kamus
 * @param length panjang kata yang ingin dicari
 * @return kata acak dari kamus
 */
public static String getRandomWord(int length) {
    List<SimpleEntry<String, String>> list = dictionary.get(length);
    return list.get((int) (Math.random() * list.size())).getKey();
}
```

4.2 Tata Cara Penggunaan Program

Untuk menjalankan program yang telah dibuat, perlu dilakukan langkah - langkah sebagai berikut.

1. Clone Repository GitHub

```
Git clone https://github.com/fauzanazz/Tucil3\_13522153
```

2. Ubah working directory

```
Cd Tucil3_13522153
```

3. Jalankan maven

```
Maven package
```

4. Buka executable jar di folder bin

```
Cd bin  
Java -jar .\WordLadder-1.0_Finish.jar
```

4.3 Hasil Pengujian

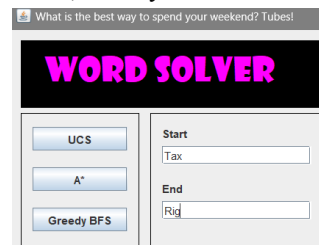
4.3.1 Kasus 1

Results			Results			Results		
Node Visit Count : 372 Memory used : 2897 KB Time elapsed : 4 ms			Node Visit Count : 5 Memory used : 3072 KB Time elapsed : 1 ms			Node Visit Count : 3 Memory used : 382 KB Time elapsed : 1 ms		
T	A	X	T	A	X	T	A	X
R	A	X	R	A	X	R	A	X
R	A	G	R	A	G	R	A	G
R	I	G	R	I	G	R	I	G

Gambar 4.11 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan

Query

- Awal : Tax
- Akhir : Rig



4.3.2 Kasus 2

Results				Results				Results			
Node Visit Count : 76 Memory used : 1349 KB Time elapsed : 0 ms				Node Visit Count : 4 Memory used : 335 KB Time elapsed : 0 ms				Node Visit Count : 3 Memory used : 335 KB Time elapsed : 0 ms			
K	E	R	B	K	E	R	B	K	E	R	B
K	E	R	N	K	E	R	N	K	E	R	N
K	A	R	N	K	A	R	N	K	A	R	N
C	A	R	N	C	A	R	N	C	A	R	N

Gambar 4.12 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan

Query

- Awal : Kerb
- Akhir : Carn

4.3.3 Kasus 3

Results

Node Visit Count : 5177
Memory used : 10842 KB
Time elapsed : 44 ms

D	E	C	A	L
F	E	C	A	L
F	E	T	A	L
F	E	T	A	S
F	E	T	E	S
F	E	T	E	D
F	E	U	E	D
F	L	U	E	D
F	L	U	E	S
G	L	U	E	S

Results

Node Visit Count : 258
Memory used : 2383 KB
Time elapsed : 3 ms

D	E	C	A	L
F	E	C	A	L
F	E	T	A	L
F	E	T	A	S
F	E	T	E	S
F	E	T	E	D
F	E	U	E	D
F	L	U	E	D
F	L	U	E	S
G	L	U	E	S

Results

Node Visit Count : 122
Memory used : 2048 KB
Time elapsed : 0 ms

D	E	C	A	L
C	E	C	A	L
F	E	C	A	L
F	E	R	A	L
F	E	T	A	L
F	E	T	A	S
F	E	T	E	S
C	E	T	E	S
C	A	T	E	S
G	A	T	E	S
G	A	G	E	S
G	A	L	E	S
G	A	M	E	S
G	A	P	E	S
G	A	S	E	S
G	A	Z	E	S
D	A	Z	E	S
D	A	C	E	S
D	A	L	E	S
B	A	L	E	S
B	A	B	E	S
B	A	K	E	S
B	A	N	E	S
B	A	R	E	S
B	A	S	E	S
B	A	T	E	S
B	I	T	E	S
B	I	C	E	S
B	I	D	E	S
A	I	D	E	S
A	E	D	E	S

A	F	T	E	R
A	P	T	E	R
A	S	T	E	R
A	S	K	E	R
A	S	K	E	D
A	S	H	E	D
A	S	H	E	S
A	C	H	E	S
A	C	M	E	S
A	L	M	E	S
A	L	O	E	S
F	L	O	E	S
F	L	U	E	S
G	L	U	E	S

I am single

WORD SOLVER

UCS

A*

Greedy BFS

Start

decal

End

glues

Gambar 4.13 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan.

Keterangan : Greedy BFS ditampilkan menggunakan dua gambar karena hasil yang didapatkan terlalu banyak.

Query

- Awal : Decal
- Akhir : Glues

4.3.4 Kasus 4

Results

Node Visit Count : 8188
Memory used : 20480 KB
Time elapsed : 48 ms

S	P	I	T	E	S
S	K	I	T	E	S
S	K	A	T	E	S
S	K	A	T	E	R
S	E	A	T	E	R
S	E	T	T	E	R
T	E	T	T	E	R
T	A	T	T	E	R
T	A	R	T	E	R
T	A	R	T	A	R
T	A	R	T	A	N
P	A	R	T	A	N
P	A	R	T	O	N
P	A	R	S	O	N
P	E	R	S	O	N
P	E	R	E	O	N
H	E	R	E	O	N
H	E	R	E	O	F

Results

Node Visit Count : 6497
Memory used : 5199 KB
Time elapsed : 35 ms

S	P	I	T	E	S
S	P	I	T	E	D
S	P	I	L	E	D
S	A	I	L	E	D
B	A	I	L	E	D
B	A	I	T	E	D
B	A	I	T	E	R
B	A	R	T	E	R
T	A	R	T	E	R
T	A	R	T	A	R
T	A	R	T	A	N
P	A	R	T	A	N
P	A	R	T	O	N
P	A	R	S	O	N
P	E	R	S	O	N
P	E	R	E	O	N
H	E	R	E	O	N
H	E	R	E	O	F

Results

Node Visit Count : 21
Memory used : 1951 KB
Time elapsed : 4 ms

No path found

Gambar 4.14 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan

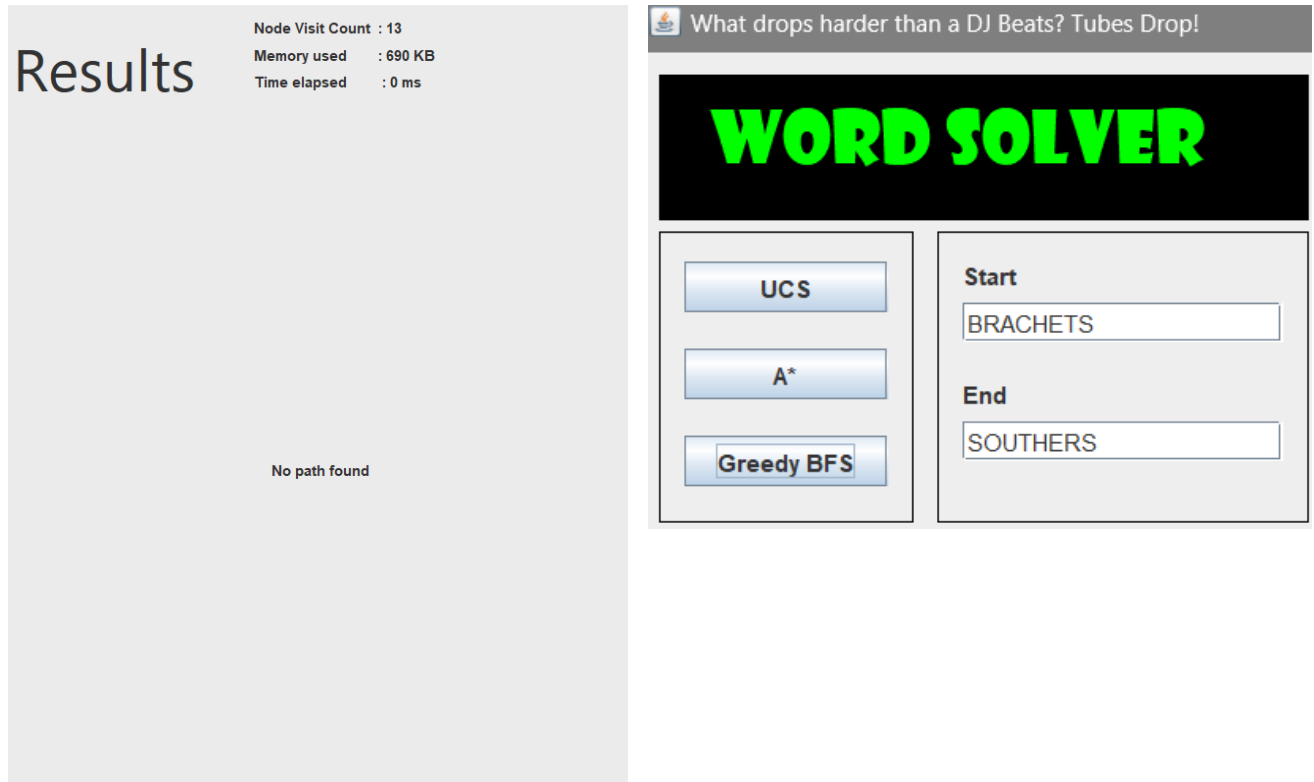
Keterangan : Greedy BFS ditampilkan menggunakan dua gambar karena hasil yang didapatkan terlalu banyak.

Query

- Awal : Spites
- Akhir : Hereof

4.3.5 Kasus 5

Results								Results							
Node Visit Count : 452								Node Visit Count : 354							
Memory used : 2048 KB								Memory used : 2417 KB							
Time elapsed : 2 ms								Time elapsed : 2 ms							
B	R	A	C	H	E	T	S	B	R	A	C	H	E	T	S
B	R	A	C	K	E	T	S	B	R	A	C	K	E	T	S
B	R	A	C	K	E	N	S	B	R	A	C	K	E	N	S
B	L	A	C	K	E	N	S	B	L	A	C	K	E	N	S
S	L	A	C	K	E	N	S	S	L	A	C	K	E	N	S
S	L	A	C	K	E	R	S	S	L	A	C	K	E	R	S
C	L	A	C	K	E	R	S	C	L	A	C	K	E	R	S
C	L	I	C	K	E	R	S	C	L	I	C	K	E	R	S
C	L	I	N	K	E	R	S	C	L	I	N	K	E	R	S
C	L	I	N	G	E	R	S	C	L	I	N	G	E	R	S
C	L	A	N	G	E	R	S	C	L	A	N	G	E	R	S
C	H	A	N	G	E	R	S	C	H	A	N	G	E	R	S
C	H	A	N	T	E	R	S	C	H	A	N	T	E	R	S
C	H	A	T	T	E	R	S	C	H	A	T	T	E	R	S
S	H	A	T	T	E	R	S	S	H	A	T	T	E	R	S
S	W	A	T	T	E	R	S	S	W	A	T	T	E	R	S
S	W	A	T	H	E	R	S	S	W	A	T	H	E	R	S
S	W	I	T	H	E	R	S	S	W	I	T	H	E	R	S
S	M	I	T	H	E	R	S	S	M	I	T	H	E	R	S
S	M	O	T	H	E	R	S	S	M	O	T	H	E	R	S
S	O	O	T	H	E	R	S	S	O	O	T	H	E	R	S
S	O	U	T	H	E	R	S	S	O	U	T	H	E	R	S

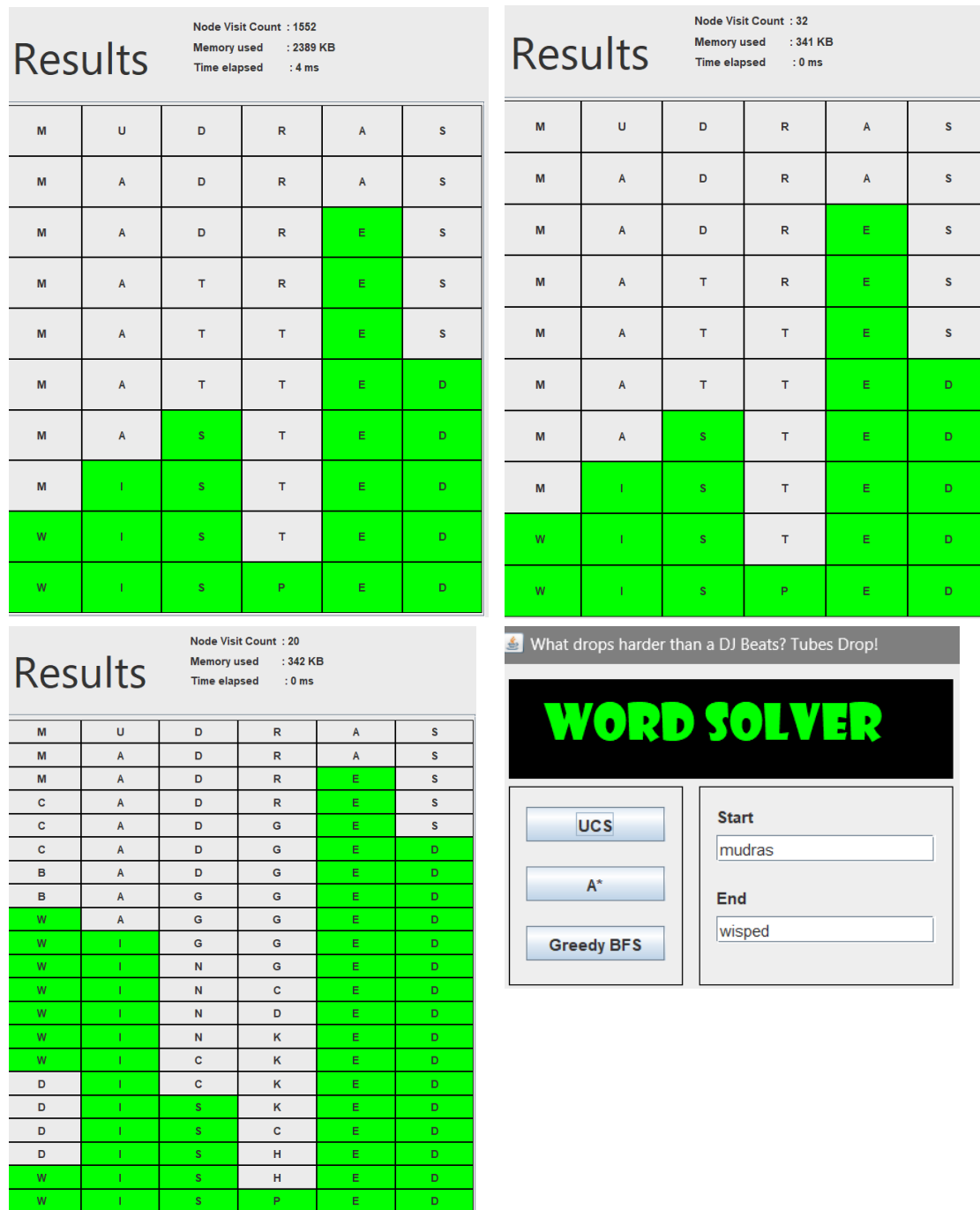


Gambar 4.15 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan

Query

- Awal : BRACHETS
- Akhir : SOUTHERS

4.3.6 Kasus 6



Gambar 4.16 Hasil Pencarian UCS, A-Star, Greedy BFS dari kiri ke kanan

Query

- Awal : mudras
- Akhir : wisped

4.4 Analisis Hasil

Pada pengujian dengan kasus yang mudah, sangat sulit untuk menentukan algoritma mana yang lebih efisien. Namun saat masukkan sudah semakin panjang mulai terlihat perbedaan pada masing-masing algoritma.

Algoritma Uniform-Cost Search selalu menjadi algoritma terlambat dibandingkan algoritma lainnya di semua uji coba. Terkadang perbedaan waktunya sangat besar dan terkadang tidak, tetapi selalu menjadi yang terlambat. Algoritma Uniform-Cost Search juga selalu menjadi algoritma dengan Node Visit Count tertinggi. Tentunya inilah alasan kenapa algoritma Uniform-Cost Search selalu menjadi yang terlambat. Hasil dari algoritma Uniform-Cost Search selalu menjadi hasil yang optimal sebanding dengan algoritma A-Star.

Algoritma A-Star menduduki posisi kedua pada kecepatan dan hanya dikalahkan oleh algoritma Greedy Best First Search. Untuk hasil yang tidak kompleks A-Star berhasil lebih cepat dibandingkan Greedy Best First Search dengan Node Visit Count yang sangat rendah, mendekati jumlah steps yang harus dituju dari start to end. Namun untuk hasil yang kompleks A-Star masih kalah dibandingkan Greedy Best First Search. Hasil dari algoritma A-Star selalu menjadi hasil yang optimal sebanding dengan algoritma Uniform-Cost Search.

Algoritma Greedy Best First Search berada pada posisi pertama pada kecepatan. Kecepatan algoritma Greedy Best First Search selalu berada pada angka 0-2 ms. Walau dengan node count yang lebih tinggi, Greedy Best First Search tetap seringkali memenangkan kecepatan diantara algoritma lainnya. Namun hasil dari algoritma Greedy Best First Search jarang sekali berada-pada posisi optimal dan bahkan bisa terkenal infinity loop dan tidak akan mendapatkan hasil. Hal ini sesuai dengan algoritma Greedy Best First Search yang tidak memperhitungkan jarak dari awal sampai akhir, algoritma Greedy Best First Search memfokuskan ke mendapatkan hasil secepat mungkin.

Penggunaan memory dari ketiga algoritma berbanding lurus dengan jumlah node yang dikunjungi. Hal ini masuk akal karena pada kode yang saya buat, ketiga algoritma mempunyai variabel private yang sama dan method yang mirip kecuali Greedy BFS. Perbandingan pemakaian memori terbesar di juarai algoritma $UCS > A^* > Greedy\ BFS$.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil pengujian dan analisis hasil pengujian, didapatkan kesimpulan yaitu sebagai berikut.

1. Algoritma A-Star sangat bekerja dengan sangat baik dan optimal jika nilai heuristik yang diberikan *admissible*. Algoritma A-Star juga termasuk algoritma yang sangat cepat dalam menemukan jalur yang sesuai dan tidak memakan banyak memori. Secara teori algoritma A-Star lebih efisien dibandingkan algoritma UCS karena tidak menelusuri semua node, tapi hanya menelusuri node yang berkemungkinan lebih dekat ke node jawaban. Telah dibuktikan juga pada uji coba bahwa hasil yang diberikan optimal, sama dengan UCS.
2. Algoritma UCS menghasilkan hasil yang optimal namun kecepatannya sangat lambat. Semakin kompleks jalur yang ada dan semakin banyak simpul yang harus dikunjungi maka UCS akan semakin melambat. UCS yang diimplementasikan pada word ladder solver sangat mirip dengan algoritma BFS dalam hal urutan simpul yang akan diproses dan path yang dibuat dikarenakan cost nya yang hanya bernilai satu. Karena UCS sangat mirip dengan BFS, hal ini membuat UCS menjamin hasil yang optimal untuk persoalan word ladder.
3. Algoritma Greedy BFS mempunyai kecepatan yang sangat konstan melebihi algoritma lainnya. Perbandingan kecepatannya bisa sangat tinggi hingga 25x lebih cepat dibandingkan A-Star. Namun hasil dari Greedy BFS sangat tidak optimal, penggunaan Greedy BFS hanya bagus dilakukan jika hanya ingin mengecek apakah dari start ke end memiliki sebuah jalur.

5.2 Saran

Berdasarkan proses pembuatan program dan hasil yang didapatkan, ada beberapa saran yang bisa diterapkan untuk mengembangkan algoritma lebih lanjut sehingga bisa berjalan secara lebih efisien dan efektif.

1. Menyederhanakan kelas Node. Saat ini kelas Node memiliki banyak atribut yang dapat disederhanakan menjadi satu atribut. Yaitu atribut bertipe double f,g,h. Ketiga atribut ini bisa saja dijadikan satu atribut double f.
2. Saat ini kamus yang digunakan untuk bahasa pada word ladder hanyalah kamus yang digunakan pada game Scrabble versi Bahasa Inggris (USA). Kamus dapat ditambahkan sesuai dengan bahasa lain-lainnya.

5.3 Refleksi

Pada Tugas Besar kali ini, banyak sekali pengalaman yang berhasil saya dapatkan tentang membuat aplikasi berbasis Java. Eksplorasi dalam menerapkan algoritma-algoritma yang telah ditentukan juga merupakan proses yang panjang. Namun satu kesalahan saya selama pembuatan program ini adalah kurang nya paham akan algoritma Greedy Best First Search. Pada awal pembuatan saya menganggap ada kemungkinan Greedy BFS tidak mempunyai jawaban. Ternyata kurangnya eksplorasi akan materi diluar PPT perkuliahan membuat saya harus membuat ulang Greedy BFS.

LAMPIRAN

Tautan repository GitHub: [fauzanazz/Tucil3_13522153: Tugas kecil 3 Stima \(github.com\)](https://github.com/fauzanazz/Tucil3_13522153)

Poin	Ya	Tidak
1. Program dapat 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	✓	

“Kak, kok gampang banget tucilnya?”

“Biar kalian bisa nyantai di antara 2 tubes :D”

“Terima kasih kak! 🙏”

DAFTAR PUSTAKA

- Kurniawan, William Manuel. "penerapan Algoritma UCS Untuk Menentukan Rute Terbaik Menuju ITB." *informatika.stei.itb.ac.id*, [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K2%20\(8\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K2%20(8).pdf). Accessed 4 May 2024.
- "Uniform Cost Search: Cara Kerja dan Kelebihannya." *KantiniT*, 4 June 2023, <https://kantinit.com/algoritma/uniform-cost-search-cara-kerja-dan-kelebihannya/>. Accessed 4 May 2024.
- Rahmawati, Atin. "Algoritma A* (A-Star): Fungsi, Cara Menghitung dan Contohnya." *DosenIT.com*, 17 May 2023, <https://dosenit.com/ilmu-komputer/algoritma-a-star>. Accessed 4 May 2024.
- Jain, Sandeep. "Greedy Best first search algorithm." *GeeksforGeeks*, 18 January 2024, <https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>. Accessed 4 May 2024.
- Munir, Rinaldi. "Penentuan Rute (Route/Path Planning)." *informatika.stei.itb.ac.id*, <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>. Accessed 5 May 2024.
- Munir, Rinaldi. "Penentuan Rute (Route/Path Planning)." *informatika.stei.itb.ac.id*, <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>. Accessed 5 May 2024.