

TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA
Penyelesaian Permainan Word Ladder Menggunakan
Algoritma UCS, Greedy Best First Search, dan A*



DISUSUN OLEH:
Aland Mulia Pratama **13522124**

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

DAFTAR ISI	1
BAB I	3
DESKRIPSI MASALAH DAN ALGORITMA	3
1.1 Algoritma UCS	3
1.2 Algoritma A*	3
1.3 Algoritma Greedy Best First Search	4
1.4 Permainan Word Ladder	5
BAB II	7
ANALISIS PERMASALAHAN PERMAINAN WORD LADDER DENGAN UCS, A* & BFS	7
2.1 Langkah-Langkah Pemecahan Masalah	7
2.1.1 Algoritma UCS	7
2.1.2 Algoritma A*	7
2.1.3 Algoritma Greedy Best First Search	8
2.2 Analisis Algoritma dalam Word Ladder Solver	8
BAB III	11
IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA & BONUS GUI	11
3.1 Backend Program	11
3.1.1 DictionaryWord.java	11
3.1.2 Pair.java	11
3.1.3 Result.java	12
3.1.4 Astar.java	12
3.1.5 UCS.java	13
3.1.6 GreedyBestFirstSearch.java	13
3.1.7 Response.java	14
3.1.8 Api.java	15
3.2 Frontend Program	15
3.2.1 page.js	15
3.2.2 stickybar.js	16
3.2.3 wordladdergrid.js	16
BAB IV	17
SOURCE CODE	17
PROGRAM	17
4.1 Repository Program	17
4.2 Source Code Program	17
4.2.1 DictionaryWord.java	17
4.2.2 Pair.java	19
4.2.3 Result.java	19
4.2.4 Astar.java	20

4.2.5 UCS.java	23
4.2.6 GreedyBestFirstSearch.java	25
4.2.7 Response.java	27
4.2.8 Api.java	29
4.2.9 page.js	30
4.2.10 stickybar.js	37
4.2.11 wordladdergrid.js	38
BAB V	39
PENGUJIAN DAN ANALISIS ALGORITMA DALAM PERMAINAN WORD LADDER	39
5.1 Validasi Program	39
5.2 Pengujian Algoritma UCS, A*, dan Greedy Best First Search	41
BAB VI	55
PENUTUP	55
6.1 Kesimpulan	55
6.2 Komentar	56
6.3 Lampiran	56
DAFTAR PUSTAKA	57

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma UCS

Uniform-Cost Search adalah sebuah varian dari algoritma Dijkstra. Di sini, alih-alih memasukkan semua simpul ke dalam antrian prioritas, kita hanya memasukkan sumbernya terlebih dahulu, kemudian satu per satu dimasukkan ketika diperlukan. Pada setiap langkah, kita memeriksa apakah item sudah ada dalam antrian prioritas (menggunakan array yang telah dikunjungi). Jika ya, kita melakukan pengurangan kunci, jika tidak, kita memasukkannya.

Varian dari Dijkstra ini berguna untuk grafik yang tak terbatas dan grafik yang terlalu besar untuk direpresentasikan dalam memori. Uniform-Cost Search biasanya digunakan dalam Kecerdasan Buatan. Uniform-Cost Search mirip dengan algoritma Dijkstra. Dalam algoritma ini, dari keadaan awal, kita akan mengunjungi keadaan-keadaan yang berdekatan dan akan memilih keadaan yang paling murah biayanya. Kemudian, kita akan memilih keadaan berikutnya yang paling murah biayanya dari semua keadaan yang belum dikunjungi dan berdekatan dengan keadaan yang telah dikunjungi.

Dengan cara ini, kita akan mencoba mencapai keadaan tujuan. Perhatikan bahwa kita tidak akan melanjutkan jalur melalui keadaan tujuan. Bahkan jika kita mencapai keadaan tujuan, kita akan terus mencari jalur-jalur lain yang mungkin, jika ada beberapa tujuan. Kita akan menyimpan antrian prioritas yang akan memberikan keadaan berikutnya yang paling murah biayanya dari semua keadaan yang berdekatan dengan keadaan yang telah dikunjungi.

Kompleksitas Waktu:

$$O(m^{(1+\lfloor l/e \rfloor)})$$

di mana,

m adalah jumlah maksimum tetangga yang dimiliki oleh sebuah node

l adalah panjang dari jalur terpendek ke keadaan tujuan

e adalah biaya terkecil dari sebuah tepi

1.2 Algoritma A*

Algoritma A* Search adalah salah satu teknik terbaik dan populer yang digunakan dalam pencarian jalur dan penelusuran graf. Secara informal, algoritma A* Search, berbeda dari teknik penelusuran lainnya, memiliki "otak". Artinya, ini adalah algoritma yang benar-benar cerdas yang membedakannya dari algoritma konvensional lainnya. Fakta ini dijelaskan secara detail dalam bagian-bagian berikut. Dan juga perlu disebutkan bahwa banyak permainan dan peta berbasis web menggunakan algoritma ini untuk menemukan jalur terpendek dengan sangat efisien.

Dalam algoritma A*, nilai $f(n)$ dari setiap simpul dihitung dengan menggabungkan bobot jalur terpendek dari titik awal ke simpul tersebut ($g(n)$) dengan nilai heuristik dari simpul tersebut ke titik tujuan ($h(n)$). Metode heuristik seperti fungsi jarak Euclidean atau Manhattan dapat digunakan untuk menghitung nilai heuristik ini. Seperti UCS, A* menggunakan pendekatan pencarian graf terurut. Namun, dalam setiap iterasinya, A* memilih simpul dengan nilai $f(n)$ terendah dari daftar simpul yang belum dikunjungi. Setelah itu, simpul tersebut ditambahkan ke daftar simpul yang telah dikunjungi, dievaluasi, dan diperiksa apakah simpul tersebut adalah simpul tujuan. Jika iya, algoritma mengembalikan jalur dengan biaya minimum dari simpul awal ke simpul tujuan. Jika tidak, algoritma akan memperluas simpul tersebut dan menambahkan anak-anak dari simpul yang sedang diperiksa ke daftar simpul yang belum dikunjungi.

1.3 Algoritma Greedy Best First Search

Greedy Best-First Search adalah algoritma pencarian yang berusaha untuk menemukan jalur yang paling menjanjikan (optimum lokal) dari titik awal yang diberikan ke tujuan. Algoritma ini memberikan prioritas pada jalur-jalur yang tampaknya paling menjanjikan, tanpa memperhatikan apakah jalur tersebut benar-benar merupakan jalur terpendek. Cara kerja algoritma ini adalah dengan mengevaluasi biaya dari setiap jalur yang mungkin, lalu memperluas jalur dengan biaya terendah. Proses ini diulang sampai tujuan tercapai.

Algoritma ini bekerja dengan menggunakan fungsi heuristik untuk menentukan jalur mana yang paling menjanjikan. Fungsi heuristik memperhitungkan biaya dari jalur saat ini dan biaya perkiraan dari jalur-jalur yang tersisa. Jika biaya dari jalur saat ini lebih rendah dari biaya perkiraan dari jalur-jalur yang tersisa, maka jalur saat ini dipilih. Proses ini diulang sampai tujuan tercapai.

Greedy Best-First Search bekerja dengan mengevaluasi biaya dari setiap jalur yang mungkin lalu memperluas jalur dengan biaya terendah. Proses ini diulang sampai tujuan tercapai. Algoritma ini menggunakan fungsi heuristik untuk menentukan jalur mana yang paling menjanjikan. Fungsi heuristik memperhitungkan biaya dari jalur saat ini dan perkiraan biaya dari jalur-jalur yang tersisa. Jika biaya dari jalur saat ini lebih rendah dari perkiraan biaya dari jalur-jalur yang tersisa, maka jalur saat ini dipilih. Proses ini diulang sampai tujuan tercapai.

Greedy Best-First Search memiliki beberapa keuntungan yang membuatnya menjadi pilihan yang baik dalam berbagai aplikasi. Pertama, algoritma ini relatif sederhana dan mudah diimplementasikan, sehingga dapat dengan cepat diterapkan dalam berbagai konteks. Selain itu, Greedy Best-First Search juga sangat cepat dan efisien, sehingga cocok digunakan dalam aplikasi di mana kecepatan pemrosesan data menjadi faktor penting. Algoritma ini juga membutuhkan sedikit memori, menjadikannya cocok untuk digunakan dalam aplikasi dengan keterbatasan memori. Kemampuannya yang fleksibel membuat Greedy Best-First Search dapat disesuaikan dengan berbagai jenis

masalah dan dengan mudah diperluas untuk menangani masalah yang lebih kompleks.

Namun, seperti halnya dengan kebanyakan algoritma, Greedy Best-First Search juga memiliki kelemahan. Salah satunya adalah hasil yang mungkin tidak akurat, karena algoritma ini hanya fokus pada menemukan jalur yang paling menjanjikan tanpa menjamin keoptimalan. Selain itu, Greedy Best-First Search juga rentan terhadap fenomena optima lokal, di mana jalur yang dipilih mungkin tidak selalu merupakan jalur terbaik secara keseluruhan. Selain itu, algoritma ini membutuhkan fungsi heuristik yang memadai untuk dapat bekerja, yang dapat menambah kompleksitas pada implementasinya. Terakhir, Greedy Best-First Search tidak selalu menjamin penemuan solusi, terutama jika algoritma terjebak dalam siklus atau jika ruang pencarian terlalu kompleks. Oleh karena itu, meskipun Greedy Best-First Search memiliki kelebihan dalam hal kecepatan dan efisiensi, perlu diperhatikan juga kelemahannya dalam menjamin keoptimalan dan kelengkapan solusi.

1.4 Permainan *Word Ladder*



Gambar 1. Penemu permainan *Word Ladder*

(Sumber: https://en.wikipedia.org/wiki/Lewis_Carroll#/media/File:LewisCarrollSelfPhoto.jpg)

Word Ladder, yang juga dikenal dengan berbagai nama seperti Doublets, word-links, atau paragrams, adalah permainan kata yang terkenal yang ditemukan oleh Lewis Carroll pada tahun 1877. Dalam permainan ini, pemain diberikan dua kata awal yang disebut sebagai start word dan end word, dan tujuannya adalah untuk menemukan rantai kata yang menghubungkan kedua kata tersebut. Setiap kata dalam rantai harus berbeda hanya dalam satu huruf dari kata sebelumnya. Untuk memenangkan permainan, pemain harus menemukan solusi optimal yang meminimalkan jumlah kata dalam rantai. Meskipun peraturan permainannya cukup sederhana, namun membuat solver untuk menyelesaikan Word Ladder ini adalah tantangan yang menarik untuk menghasilkan solusi paling optimal.

How To Play

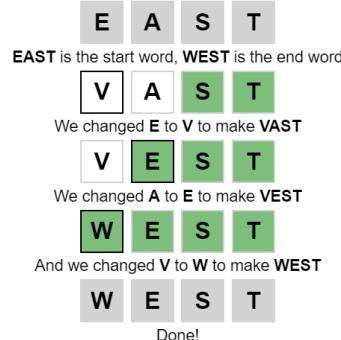
This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

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



Gambar 2. Ilustrasi dan Peraturan Permainan *Word Ladder*

(Sumber: <https://wordwormdormdork.com/>)

BAB II

ANALISIS PERMASALAHAN PERMAINAN WORD LADDER DENGAN UCS, A* & BFS

2.1 Langkah-Langkah Pemecahan Masalah

2.1.1 Algoritma UCS

Uniform Cost Search (UCS), yang merupakan versi dari algoritma Dijkstra yang dioptimalkan untuk mencari jalur terpendek dalam hal biaya antara dua node. Algoritma ini mencari jalur terpendek dari kata awal (start) ke kata tujuan (end) dalam ruang pencarian yang ditentukan oleh dictionary. Berikut adalah langkah-langkah pemecahan masalah yang diambil oleh algoritma UCS ini:

1. Algoritma dimulai dengan menambahkan node awal ke openList dengan biaya 0.
2. Selama openList tidak kosong, keluarkan node dengan biaya terendah dari openList.
3. Jika node ini adalah node tujuan, jalur dari awal ke tujuan telah ditemukan. Gunakan parents untuk merekonstruksi dan kembalikan jalur ini.
4. Jika node belum dikunjungi, tandai sebagai dikunjungi dan perbarui visitedCount.
5. Untuk setiap node yang diproses, identifikasi semua tetangganya (dalam kasus ini, kata-kata yang dapat dibentuk dengan mengubah satu huruf pada satu waktu).
6. Untuk setiap tetangga, hitung biaya baru untuk mencapai tetangga ini, yang merupakan biaya untuk mencapai node saat ini ditambah satu (setiap langkah dianggap memiliki biaya 1).
7. Jika tetangga belum tercatat dalam cost atau memiliki biaya yang lebih tinggi dari biaya baru. Perbarui cost untuk tetangga dan tandai node saat ini sebagai orang tua dari tetangga ini dalam parents.

2.1.2 Algoritma A*

Algoritma A* diterapkan pada program word ladder solver untuk menemukan jalur terpendek dari satu kata (start) ke kata lain (end) menggunakan kamus atau set kata yang diberikan. Berikut adalah langkah-langkah pemecahan masalah Algoritma A*:

1. Dimulai dengan menambahkan kata awal (start) ke openList dengan heuristic awal dari kata tersebut ke kata tujuan (end).
2. Selama openList tidak kosong, maka kata saat ini (currentWord) diperoleh dengan mengeluarkan elemen dari openList. Jika currentWord sama dengan end, maka jalur dari start ke end telah ditemukan. Jalur ini direkonstruksi dari end ke start menggunakan parents dan dikembalikan sebagai hasil.
3. Jika kata saat ini belum dikunjungi, proses menandainya sebagai dikunjungi dan melakukan iterasi melalui setiap karakter kata tersebut.
4. Untuk setiap karakter dalam currentWord, kode mencoba menggantinya dengan setiap huruf dalam alfabet dari 'a' sampai 'z'.

5. Jika neighbor ada dalam dictionary dan belum dikunjungi, algoritma memperhitungkan biaya untuk mencapai neighbor (biaya saat ini + 1).
6. Jika neighbor belum ada dalam costSoFar atau biaya baru lebih rendah dari biaya sebelumnya, maka perbarui costSoFar untuk neighbor. Setelah itu hitung nilai fungsi heuristik total untuk neighbor dan tambahkan ke openList. Tandai currentWord sebagai orangtua dari neighbor dalam parents.

2.1.3 Algoritma Greedy Best First Search

Algoritma Greedy Best-First Search (GBFS) dalam kasus ini dirancang untuk menemukan jalur secepat mungkin dari kata awal (start) ke kata tujuan (end) menggunakan sebuah kamus kata berbahasa Inggris dalam file .txt. Kode ini menggunakan heuristik untuk memandu pencarian ke tujuan dengan memprioritaskan kata yang paling mendekati tujuan secara heuristik. Berikut adalah langkah-langkah pemecahan masalah yang diikuti oleh algoritma GBFS ini:

1. Memulai dengan menambahkan kata start ke dalam queue dengan nilai heuristiknya terhadap end.
2. Menandai start sebagai kata yang sudah dikunjungi dan menginkrement visitedCount yaitu jumlah node yang dikunjungi selama berlangsungnya program.
3. Selama queue tidak kosong, algoritma melanjutkan akan mengeluarkan kata dari queue berdasarkan prioritas heuristik terendah. Jika kata ini adalah end, maka jalur dari start ke end telah ditemukan. Jalur ini kemudian direkonstruksi dari end ke start menggunakan parent dan dikembalikan sebagai hasil. Jika bukan, iterasi melalui setiap "tetangga" dari kata ini, yaitu kata-kata yang bisa dibentuk dengan mengubah satu huruf dari kata saat ini.
4. Untuk setiap tetangga yang dihasilkan. Jika tetangga belum dikunjungi, proses dengan menandai sebagai dikunjungi. Menyimpan kata saat ini sebagai "orang tua" dari tetangga ini. Menambahkan tetangga ke queue dengan nilai heuristiknya dan menginkrement visitedCount.
5. Jika loop selesai tanpa menemukan end, kembalikan hasil dengan jalur kosong dan jumlah node yang dikunjungi, menunjukkan bahwa tidak ada jalur yang ditemukan. Jika jalur ditemukan, kembalikan jalur tersebut dan jumlah node yang dikunjungi.

GBFS memprioritaskan eksplorasi berdasarkan seberapa dekat kata-kata ke tujuan berdasarkan heuristik yang bisa sangat efisien jika heuristiknya baik. Namun, GBFS tidak selalu menjamin solusi optimal karena fokus pada jarak heuristik bisa mengabaikan faktor biaya atau jarak yang sebenarnya yang lebih pendek.

2.2 Analisis Algoritma dalam Word Ladder Solver

Dalam membuat program Word Ladder Solver pada Tugas Kecil 3 IF2211 Strategi Algoritma, saya menggunakan kamus dengan format file .txt yang diberikan melalui kolom QnA. Kamus bahasa inggris yang saya gunakan dapat diakses pada

[pranala](#) berikut. Dalam kamus tersebut tersedia 80.368 kosakata dalam bahasa inggris. Setiap solusi word ladder dihasilkan berdasarkan kosakata yang terdapat pada kamus tersebut. Input juga divalidasi terlebih dahulu apakah terdaftar dalam kamus yang saya gunakan.

Algoritma A* yang kami rancang menggunakan fungsi evaluasi berupa $f(n) = g(n) + h(n)$. Definisi dari $f(n)$ adalah nilai estimasi total biaya terkecil dari node awal ke node tujuan melewati node n. Ini dihitung sebagai $f(n) = g(n) + h(n)$, di mana $h(n)$ adalah heuristic yang mengestimasi biaya paling murah dari n ke tujuan. $g(n)$ adalah biaya dari node awal (start) ke node n. Dalam kasus ini, biaya diukur berdasarkan jumlah langkah atau perubahan yang diperlukan untuk berubah dari satu kata ke kata berikutnya dalam dictionary. $h(n)$ adalah heuristik yang digunakan untuk mengestimasi biaya terendah dari node n ke tujuan end. Implementasi dari definisi $f(n)$, $g(n)$, dan $h(n)$ dapat dilihat pada gambar 3, gambar 4, dan gambar 5 secara berturut-turut.

```
int f = newCost + heuristic(neighbor, end);
openList.offer(new Pair<>(f, neighbor));
```

Gambar 3. Implementasi definisi $f(n)$

```
private static int heuristic(String word, String endWord) {
    int count = 0;
    for (int i = 0; i < word.length(); i++) {
        if (i < endWord.length() && word.charAt(i) != endWord.charAt(i)) {
            count++;
        }
    }
    return count;
};
```

Gambar 4. Implementasi definisi $h(n)$

```
int newCost = currentCost + 1;
if (!costSoFar.containsKey(neighbor) || newCost < costSoFar.get(neighbor)) {
    costSoFar.put(neighbor, newCost);
```

Gambar 5. Implementasi definisi $g(n)$

Heuristik yang digunakan dalam algoritma A* untuk menyelesaikan masalah word ladder adalah admissible. Heuristik admissible adalah heuristik yang tidak pernah *overestimates* biaya sebenarnya untuk mencapai tujuan dari node saat ini. Dalam hal word ladder, mengganti setiap karakter yang tidak cocok dengan karakter yang sesuai dalam kata tujuan adalah langkah yang sah dan diperlukan. Heuristik ini menilai biaya minimum yang mungkin diperlukan untuk transformasi dari startWord ke endWord dengan mengasumsikan bahwa setiap perubahan satu huruf menghasilkan kata yang valid dan membawa kata tersebut lebih dekat ke tujuan. Dalam konteks algoritma A* untuk word ladder solver, heuristik berbasis jumlah karakter yang tidak cocok adalah admissible karena menyediakan perkiraan biaya yang konservatif atau tidak pernah lebih untuk mencapai tujuan.

UCS adalah algoritma yang memprioritaskan node berdasarkan biaya jalur terendah dari node awal. UCS menggunakan priority queue untuk menyimpan node, di mana node dengan biaya total terendah selalu diekstrak terlebih dahulu. Dalam kasus word ladder solver, semua langkah memiliki biaya yang sama dalam UCS. Karena setiap langkah memiliki biaya yang sama maka urutan eksplorasi node dalam UCS akan sama

dengan BFS. Keduanya akan mengeksplorasi node berdasarkan kedalaman mereka dari node awal. Dalam hal ini, kedua algoritma cenderung menghasilkan jalur yang sama dari node awal ke tujuan karena mereka menggunakan kriteria yang sama untuk memilih antara node yang dapat diekspansi selanjutnya ketika terdapat beberapa pilihan.

Secara teoritis, algoritma Greedy Best First Search (GBFS) tidak menjamin solusi optimal (jumlah kata yang digunakan dalam menyelesaikan word ladder) untuk persoalan word ladder. GBFS berfokus pada mengurangi perkiraan jarak ke tujuan tanpa mempertimbangkan biaya total dari jalur yang diambil. Karena itu, algoritma ini mungkin mengabaikan jalur yang lebih pendek atau lebih efisien karena terpaku pada tujuan jangka pendek untuk mendekati tujuan dengan cepat. Oleh karena itu, GBFS merupakan algoritma yang optimal secara waktu eksekusi tetapi belum tentu optimal terhadap biaya dari startWord menuju endWord.

BAB III

IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA & BONUS GUI

Dalam pembuatan program Word Ladder Solver, penulis menggunakan bahasa pemrograman java sesuai dengan spesifikasi tugas untuk melakukan komputasi algoritma UCS, A*, dan Greedy Best First Search untuk menemukan solusi Word Ladder. Implementasi Algoritma dan API program diletakkan ke dalam folder backend. Penulis juga melakukan implementasi GUI (Graphic User Interface) menggunakan website dengan framework Next.js. Implementasi GUI dari program diletakkan pada folder frontend.

3.1 Backend Program

Struktur dari backend terbagi menjadi 6 file yaitu DictionaryWord.java, Pair.java, Result.java, Astar.java, UCS.java, GreedyBestFirstSearch.java, Response.java, dan juga Api.java. Dalam pembuatan algoritma untuk program Word Ladder Solver kami memerlukan beberapa utilitas selain daripada algoritma utama dari program. Hal ini yang menyebabkan struktur backend dibagi menjadi beberapa file dengan tujuan untuk meningkatkan modularitas program.

3.1.1 DictionaryWord.java

File ini memiliki tujuan untuk menemukan serangkaian perubahan kata dari kata awal ke kata akhir, di mana setiap langkah perubahan melibatkan hanya satu karakter berbeda pada kata sebelumnya serta memuat kamus bahasa inggris yang telah disediakan.

Methods	Deskripsi
loadDictionary(String filename)	Method ini bertugas untuk memuat kamus kata dari sebuah file teks. Parameter filename adalah nama file yang berisi kamus kata.
getNeighbors(String word, Set<String> dictionary)	Method ini mengembalikan daftar kata-kata bertetangga dari sebuah kata yang diberikan, berdasarkan kamus kata yang diberikan juga.

3.1.2 Pair.java

Pair.java mendefinisikan sebuah kelas generik bernama Pair, yang merepresentasikan pasangan nilai kunci. Setiap objek Pair terdiri dari dua elemen: kunci (key) dan nilai (value).

Methods	Deskripsi
Pair(K key, V value)	Konstruktor kelas Pair yang menginisialisasi objek Pair dengan kunci dan nilai yang diberikan.
getKey()	Method ini mengembalikan kunci (key) dari objek Pair.
getValue()	Method ini mengembalikan nilai (value) dari objek Pair.
compareTo(Pair<K, V> o)	Method ini mengimplementasikan interface Comparable dan memungkinkan objek Pair untuk dibandingkan berdasarkan kuncinya (key). Method ini membandingkan kunci objek saat

	ini dengan kunci objek o yang diberikan sebagai parameter, dan mengembalikan hasil perbandingan.
--	--

3.1.3 Result.java

file ini mendefinisikan kelas Result, yang digunakan untuk menyimpan hasil dari pencarian kata pada algoritma Word Ladder. Kelas ini memiliki dua atribut utama: path dan visitedCount. Kelas Result digunakan untuk menjadi tipe objek yang dikembalikan dari ketiga fungsi algoritma pencarian tersebut.

Methods	Deskripsi
Result(List<String> path, int visitedCount)	Konstruktor kelas Result yang menginisialisasi objek Result dengan jalur (path) dan jumlah kata yang telah dikunjungi (visitedCount) yang diberikan.
getPath()	Method ini mengembalikan jalur (path) yang disimpan dalam objek Result.
getVisitedCount()	Method ini mengembalikan jumlah kata yang telah dikunjungi (visitedCount) yang disimpan dalam objek Result.
setPath(List<String> path)	Mengatur jalur (path) dalam objek Result dengan jalur yang diberikan sebagai parameter.
setVisitedCount(int visitedCount)	Mengatur jumlah kata yang telah dikunjungi (visitedCount) dalam objek Result dengan nilai yang diberikan sebagai parameter.

3.1.4 Astar.java

File ini mengimplementasikan algoritma A* untuk menemukan Word Ladder Solution dari sebuah kata awal ke kata akhir dalam kamus kata yang diberikan. Algoritma A* merupakan modifikasi dari algoritma pencarian graf yang mencari jalur terpendek dari titik awal ke titik akhir dengan mempertimbangkan biaya dan estimasi biaya tersisa.

Attributes / Methods	Deskripsi
heuristic(String word, String endWord)	Menghitung nilai heuristik antara dua kata. Dalam implementasi ini, nilai heuristik adalah jumlah karakter yang berbeda antara dua kata.
wordLadder(String start, String end, Set<String> dictionary)	Implementasi utama algoritma A* untuk mencari Word Ladder dari start ke end dalam kamus kata yang diberikan. Method ini mengembalikan objek Result yang berisi jalur dari start ke end serta jumlah kata yang telah dikunjungi.
openList	PriorityQueue yang menyimpan pasangan nilai f (biaya total) dan kata, diurutkan berdasarkan nilai f terkecil.
visited	Set yang menyimpan kata-kata yang telah dikunjungi.
parents	Map yang menyimpan relasi antara setiap kata dan kata sebelumnya dalam jalur.
costSoFar	Map yang menyimpan biaya terkecil yang ditemukan untuk

	mencapai setiap kata.
startWord	Kata awal untuk pencarian Word Ladder.
endWord	Kata akhir atau tujuan dari pencarian Word Ladder.
dictionary	Tipe data set yang menyimpan kamus data yang telah dibaca.

3.1.5 UCS.java

File ini mengimplementasikan algoritma UCS (Uniform Cost Search) untuk mencari Word Ladder Solution dari sebuah kata awal ke kata akhir dalam kamus kata yang diberikan. Algoritma UCS adalah algoritma pencarian graf yang mencari jalur terpendek dari titik awal ke titik akhir dengan mempertimbangkan biaya yang diperlukan untuk mencapai setiap simpul.

Attributes / Methods	Deskripsi
wordLadder(String start, String end, Set<String> dictionary)	mencari Word Ladder Solution dari start ke end dalam kamus kata yang diberikan. Method ini mengembalikan objek Result yang berisi jalur dari start ke end serta jumlah kata yang telah dikunjungi.
reconstructPath(Map<String, String> parents, String start, String end)	merekonstruksi jalur dari kata awal ke kata akhir berdasarkan map parents yang menyimpan hubungan antara setiap kata dan kata sebelumnya dalam jalur.
openList	PriorityQueue yang menyimpan pasangan nilai f (biaya total) dan kata, diurutkan berdasarkan nilai f terkecil.
visited	Set yang menyimpan kata-kata yang telah dikunjungi.
parents	Map yang menyimpan relasi antara setiap kata dan kata sebelumnya dalam jalur.
cost	Map yang menyimpan biaya terkecil yang ditemukan untuk mencapai setiap kata.
startWord	Kata awal untuk pencarian Word Ladder.
endWord	Kata akhir atau tujuan dari pencarian Word Ladder.
dictionary	Tipe data set yang menyimpan kamus data yang telah dibaca.

3.1.6 GreedyBestFirstSearch.java

File ini mengimplementasikan algoritma Greedy Best-First Search untuk mencari Word Ladder (tangga kata) dari sebuah kata awal ke kata akhir dalam kamus kata yang diberikan. Algoritma Greedy Best-First Search adalah varian dari algoritma Best-First Search yang memprioritaskan simpul yang paling dekat dengan tujuan berdasarkan heuristik.

Attributes / Methods	Deskripsi
findWordLadder(String start, String end, Set<String> dictionary)	mencari Word Ladder dari start ke end dalam kamus kata yang diberikan. Method ini mengembalikan objek Result yang berisi jalur dari start ke end serta jumlah kata yang telah dikunjungi.

reconstructPath(Map<String, String> parents, String start, String end)	Method ini digunakan untuk merekonstruksi jalur dari kata awal ke kata akhir berdasarkan map parent yang menyimpan hubungan antara setiap kata dan kata sebelumnya dalam jalur.
heuristic(String word, String end)	Menghitung nilai heuristic diantara 2 kata.
queue	PriorityQueue yang menyimpan pasangan nilai f (biaya total) dan kata, diurutkan berdasarkan nilai f terkecil.
visited	Set yang menyimpan kata-kata yang telah dikunjungi.
parents	Map yang menyimpan relasi antara setiap kata dan kata sebelumnya dalam jalur.
visitedCount	Variabel untuk menyimpan jumlah kata yang telah dikunjungi.
startWord	Kata awal untuk pencarian Word Ladder.
endWord	Kata akhir atau tujuan dari pencarian Word Ladder.
dictionary	Tipe data set yang menyimpan kamus data yang telah dibaca.

3.1.7 Response.java

File ini berisi sebuah kelas Response yang digunakan untuk membungkus data respons HTTP dalam bentuk JSON. Kelas ini memiliki atribut untuk menyimpan pesan respons, data, waktu eksekusi, jumlah node yang dikunjungi, dan memori yang digunakan selama eksekusi.

Attributes / Methods	Deskripsi
Response(String message, Object data, long executionTime, int visitedNode, long executionMemory)	Konstruktor kelas Response yang menginisialisasi objek Response dengan pesan respons, data, waktu eksekusi, jumlah node yang dikunjungi, dan memori yang digunakan.
getMessage()	mengembalikan pesan respons.
setMessage(String message)	mengatur pesan respons.
getData()	mengembalikan data respons.
setData(Object data)	mengatur data respons.
getExecutionTime()	mengembalikan waktu eksekusi.
setExecutionTime(long executionTime)	digunakan untuk mengatur waktu eksekusi.
getVisitedNode()	mengembalikan jumlah node yang dikunjungi.
setVisitedNode(int visitedNode)	mengatur jumlah node yang dikunjungi.
getExecutionMemory()	Mengembalikan memory yang dibutuhkan selama eksekusi.
setExecutionMemory(int executionMemory)	Mengatur jumlah memori yang digunakan selama eksekusi.
json()	mengonversi objek Response menjadi JSON ResponseEntity. Method ini menggunakan Jackson ObjectMapper untuk mengubah objek menjadi string JSON. Jika konversi berhasil, method ini mengembalikan ResponseEntity dengan status OK

	dan body berisi JSON yang dihasilkan. Jika terjadi kesalahan dalam pemrosesan JSON, method ini mengembalikan ResponseEntity dengan status INTERNAL_SERVER_ERROR dan body berisi pesan kesalahan.
--	--

3.1.8 Api.java

Kelas Api adalah sebuah REST controller yang menangani permintaan HTTP untuk mencari Word Ladder menggunakan algoritma UCS, A*, atau Greedy Best-First Search. Di bawah ini adalah penjelasan mengenai fungsionalitas dan komponen utama dari kelas Api.

Annotation / Methods	Deskripsi
@CrossOrigin	Digunakan untuk memberikan izin kepada sumber daya lintas domain dari http://localhost:3000, sehingga permintaan dari aplikasi frontend yang berjalan di alamat tersebut dapat diakses.
@RestController	Mendefinisikan bahwa kelas Api adalah REST controller yang menangani permintaan HTTP dan menghasilkan data dalam format JSON.
ResponseEntity<Object> api	Merupakan endpoint utama yang menerima permintaan HTTP GET pada path /api dengan parameter start, target, dan algorithm. Method ini melakukan pencarian Word Ladder menggunakan algoritma yang sesuai dengan nilai parameter algorithm, yaitu UCS, A*, atau Greedy Best-First Search.

3.2 Frontend Program

3.2.1 page.js

File ini adalah komponen React utama yang bertanggung jawab terhadap halaman utama dari website Word Ladder Solver. Berikut adalah state/function yang digunakan dalam page.js ini.

State / Function	Deskripsi
State Hooks	React Hooks untuk menyimpan state dari komponen, seperti length, startWord, endWord, activeAlgorithm, results, errorMessage, submitted, dan loading.
Fungsi Delay	Mendefinisikan fungsi delay yang mengembalikan janji dengan penundaan waktu tertentu. Ini digunakan untuk membuat jeda pada aplikasi.
Handle Algorithm Click	Fungsi yang dipanggil ketika salah satu tombol algoritma ditekan. Ini mengatur state activeAlgorithm sesuai dengan algoritma yang dipilih.
Handle Submit	Fungsi yang dipanggil ketika formulir dikirim. Ini

	mengirimkan permintaan HTTP ke backend dengan parameter yang sesuai, kemudian menanggapi hasilnya dan mengatur state sesuai.
--	--

3.2.2 stickybar.js

Komponen React StickyBar adalah bagian pendukung dari tampilan website yang menampilkan sebuah bar yang menempel di bagian atas layar. Dalam komponen ini terdapat informasi tentang judul aplikasi dan pembuatnya secara jelas dan konsisten.

3.2.3 wordladdergrid.js

Komponen WordLadderGrid adalah komponen React yang bertanggung jawab untuk menampilkan solusi dalam bentuk grid untuk sebuah Word Ladder. Komponen ini menampilkan representasi visual dari Word Ladder, dengan memberikan warna hijau pada karakter yang sudah sesuai pada karakter kata tujuan dengan urutan yang sama. Komponen ini mempermudah pengguna untuk melihat langkah-langkah perubahan kata yang terjadi dalam pencarian Word Ladder.

BAB IV

SOURCE CODE

PROGRAM

4.1 Repository Program

Repository Program dapat diakses melalui tautan GitHub berikut:

https://github.com/alandmprtma/Tucil3_13522124

4.2 Source Code Program

4.2.1 DictionaryWord.java

```
package wordladder.tucil3_13522124;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.Set;

public class DictionaryWord {
    /**
     * Memuat kamus kata-kata dari file teks.
     *
     * @param filename nama file kamus
     * @return himpunan kata-kata dalam kamus
     * @throws IOException jika terjadi kesalahan saat membaca file
     */
    public static Set<String> loadDictionary(String filename) throws
IOException {
        return new HashSet<>(Files.readAllLines(Paths.get(filename)));
    }

    /**
     * Mengembalikan daftar kata-kata yang merupakan tetangga-tetangga
     * dari suatu kata
     * dalam kamus, yaitu kata-kata yang hanya berbeda satu karakter.
     *
     * @param word kata yang akan dicari tetangganya
     */
}
```

```

    * @param dictionary kamus kata-kata yang digunakan sebagai referensi
    * @return daftar kata-kata tetangga dari kata yang diberikan
    */
    public static List<String> getNeighbors(String word, Set<String>
dictionary) {
        List<String> neighbors = new ArrayList<>();
        char[] wordArray = word.toCharArray();

        for (int i = 0; i < word.length(); i++) {
            char originalChar = wordArray[i];
            for (char ch = 'a'; ch <= 'z'; ch++) {
                if (ch != originalChar) {
                    wordArray[i] = ch;
                    String newWord = new String(wordArray);
                    if (dictionary.contains(newWord)) {
                        neighbors.add(newWord);
                    }
                }
            }
            wordArray[i] = originalChar;
        }

        return neighbors;
    }

    /**
     * Merekonstruksi jalur terpendek dari awal ke akhir menggunakan peta
orang tua.
     *
     * @param parents peta yang berisi hubungan orang tua-anak dari
kata-kata
     * @param start kata awal
     * @param end kata target
     * @return daftar kata yang mewakili jalur terpendek dari awal ke
akhir
    */
    public static List<String> reconstructPath(Map<String, String>
parents, String start, String end) {
        LinkedList<String> path = new LinkedList<>();
        for (String at = end; at != null; at = parents.get(at)) {

```

```

        path.addFirst(at);
    }
    return path;
}
}

```

4.2.2 Pair.java

```

package wordladder.tucil3_13522124;

public class Pair<K extends Comparable<K>, V> implements Comparable<Pair<K, V>> {
    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public V getValue() {
        return value;
    }

    @Override
    public int compareTo(Pair<K, V> o) {
        return this.key.compareTo(o.key);
    }
}

```

4.2.3 Result.java

```

package wordladder.tucil3_13522124;

import java.util.List;

public class Result {

```

```

private List<String> path;
private int visitedCount;

public Result(List<String> path, int visitedCount) {
    this.path = path;
    this.visitedCount = visitedCount;
}

public List<String> getPath() {
    return path;
}

public int getVisitedCount() {
    return visitedCount;
}

public void setPath(List<String> path) {
    this.path = path;
}

public void setVisitedCount(int visitedCount) {
    this.visitedCount = visitedCount;
}
}

```

4.2.4 Astar.java

```

package wordladder.tucil3_13522124;

import java.util.*;
import java.io.IOException;

public class Astar {
    /**
     * Fungsi heuristik: menghitung jumlah karakter yang berbeda antara
     * dua kata.
     *
     * @param word kata pertama
     * @param endWord kata kedua
     * @return jumlah karakter yang berbeda antara dua kata
     */
}

```

```

private static int heuristic(String word, String endWord) {
    int count = 0;
    for (int i = 0; i < word.length(); i++) {
        if (i < endWord.length() && word.charAt(i) != endWord.charAt(i)) {
            count++;
        }
    }
    return count;
};

/***
 * Mencari jalur terpendek antara dua kata menggunakan algoritma A*.
 *
 * @param start kata awal
 * @param end kata target
 * @param dictionary himpunan kata-kata valid
 * @return objek Result yang berisi jalur terpendek dan jumlah node yang dikunjungi
 */
public static Result wordLadder(String start, String end, Set<String> dictionary) {
    PriorityQueue<Pair<Integer, String>> openList = new PriorityQueue<>(Comparator.comparingInt(Pair::getKey));
    Set<String> visited = new HashSet<>();
    Map<String, String> parents = new HashMap<>();
    Map<String, Integer> costSoFar = new HashMap<>();
    int visitedCount = 0;

    openList.offer(new Pair<>(heuristic(start, end), start));
    costSoFar.put(start, 0);

    while (!openList.isEmpty()) {
        String currentWord = openList.poll().getValue();
        if (currentWord.equals(end)) {
            List<String> path = new ArrayList<>();
            while (currentWord != null) {
                path.add(currentWord);
                currentWord = parents.get(currentWord);
            }
        }
    }
}

```

```

        Collections.reverse(path);
        return new Result(path, visitedCount);
    }

    if (visited.add(currentWord)) {
        visitedCount++;
        int currentCost = costSoFar.get(currentWord);

        for (int i = 0; i < currentWord.length(); i++) {
            StringBuilder sb = new StringBuilder(currentWord);
            for (char c = 'a'; c <= 'z'; c++) {
                sb.setCharAt(i, c);
                String neighbor = sb.toString();
                if (dictionary.contains(neighbor) &&
!visited.contains(neighbor)) {
                    int newCost = currentCost + 1;
                    if (!costSoFar.containsKey(neighbor) ||
newCost < costSoFar.get(neighbor)) {
                        costSoFar.put(neighbor, newCost);
                        int f = newCost + heuristic(neighbor,
end);
                        openList.offer(new Pair<>(f, neighbor));
                        parents.put(neighbor, currentWord);
                    }
                }
            }
        }
    }

    return new Result(Collections.emptyList(), visitedCount);
}

public static void main(String[] args) {
    try{
        String startWord = "base";
        String endWord = "root";
        Set<String> dictionary =
DictionaryWord.loadDictionary("./Dictionary/words_alpha.txt");
        long startTime = System.currentTimeMillis();
    }
}

```

```

        Result path = wordLadder(startWord, endWord, dictionary);
        long endTime = System.currentTimeMillis();
        System.out.println("Path: " + path);
        long duration = endTime - startTime;
        System.out.println("Waktu eksekusi: " + duration + " ms");
    }catch(IOException e){
        System.out.println("Error reading dictionary file: " +
e.getMessage());
    }
}
}

```

4.2.5 UCS.java

```

package wordladder.tucil3_13522124;

import java.io.IOException;
import java.util.*;

public class UCS {
    /**
     * Mencari jalur terpendek antara dua kata menggunakan algoritma
     Uniform Cost Search.
     *
     * @param start kata awal
     * @param end kata target
     * @param dictionary himpunan kata-kata valid
     * @return objek Result yang berisi jalur terpendek dan jumlah node
     yang dikunjungi
     */
    public static Result wordLadder(String start, String end, Set<String>
dictionary) {
        PriorityQueue<Pair<Integer, String>> openList = new
PriorityQueue<>(Comparator.comparingInt(Pair::getKey));
        Map<String, Integer> cost = new HashMap<>();
        Map<String, String> parents = new HashMap<>();
        Set<String> visited = new HashSet<>();
        int visitedCount = 0;

        openList.offer(new Pair<>(0, start));

```

```

cost.put(start, 0);

while (!openList.isEmpty()) {
    Pair<Integer, String> current = openList.poll();
    String currentWord = current.getValue();

    if (currentWord.equals(end)) {
        return new Result(DictionaryWord.reconstructPath(parents,
start, end), visitedCount);
    }

    if (!visited.contains(currentWord)) {
        visited.add(currentWord);
        visitedCount++;
        List<String> neighbors =
DictionaryWord.getNeighbors(currentWord, dictionary);
        for (String neighbor : neighbors) {
            int newCost = cost.get(currentWord) + 1;
            if (!cost.containsKey(neighbor) || newCost <
cost.get(neighbor)) {
                parents.put(neighbor, currentWord);
                cost.put(neighbor, newCost);
                openList.offer(new Pair<>(newCost, neighbor));
            }
        }
    }
}

return new Result(Collections.emptyList(), visitedCount);
}

public static void main(String[] args) {
    try {
        Set<String> dictionary =
DictionaryWord.loadDictionary("./Dictionary/words_alpha.txt");
        String startWord = "volunteer";
        String endWord = "reception";
        long startTime = System.currentTimeMillis();
        Result path = wordLadder(startWord, endWord, dictionary);
        long endTime = System.currentTimeMillis();
    }
}

```

```

        long duration = endTime - startTime;
        System.out.println("Path: " + path);
        System.out.println("Waktu eksekusi: " + duration + " ms");
    } catch (IOException e) {
        System.out.println("Error reading dictionary file: " +
e.getMessage());
    }
}
}
}

```

4.2.6 GreedyBestFirstSearch.java

```

package wordladder.tucil3_13522124;

import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Set;

public class GreedyBestFirstSearch {
    /**
     * Mencari jalur terpendek antara dua kata menggunakan algoritma
     Greedy Best-First Search.
     *
     * @param start kata awal
     * @param end kata target
     * @param dictionary himpunan kata-kata valid
     * @return objek Result yang berisi jalur terpendek dan jumlah node
     yang dikunjungi
     */
    public static Result findWordLadder(String start, String end,
Set<String> dictionary) {
        PriorityQueue<Pair<String, Integer>> queue = new
PriorityQueue<>(Comparator.comparingInt(Pair::getValue));
        Set<String> visited = new HashSet<>();
        Map<String, String> parent = new HashMap<>();
        int visitedCount = 0; // Inisialisasi counter untuk node yang
dikunjungi
    }
}

```

```

queue.offer(new Pair<>(start, heuristic(start, end)));
visited.add(start);
visitedCount++;

while (!queue.isEmpty()) {
    Pair<String, Integer> node = queue.poll();
    String currentWord = node.getKey();

    if (currentWord.equals(end)) {
        return new Result(DictionaryWord.reconstructPath(parent,
start, end), visitedCount);
    }

    for (String neighbor :
DictionaryWord.getNeighbors(currentWord, dictionary)) {
        if (!visited.contains(neighbor)) {
            visited.add(neighbor);
            parent.put(neighbor, currentWord);
            queue.offer(new Pair<>(neighbor, heuristic(neighbor,
end)));
            visitedCount++; // Menginkrement jumlah node yang
dikunjungi
        }
    }
}

return new Result(Collections.emptyList(), visitedCount); // Jika
tidak ditemukan jalur
}

/**
 * Menghitung nilai heuristik antara dua kata.
 *
 * @param word kata pertama
 * @param end kata kedua
 * @return jumlah karakter yang berbeda antara dua kata
 */
private static int heuristic(String word, String end) {
    int mismatch = 0;
    for (int i = 0; i < word.length(); i++) {

```

```

        if (word.charAt(i) != end.charAt(i)) {
            mismatch++;
        }
    }
    return mismatch;
}

}

```

4.2.7 Response.java

```

package wordladder.tucil3_13522124;

import org.springframework.http.ResponseEntity;
import com.fasterxml.jackson.core.JsonProcessingException;
import org.springframework.http.HttpStatus;
import com.fasterxml.jackson.databind.ObjectMapper;

public class Response {
    private String message;
    private Object data;
    private double executionTime;
    private int visitedNode;
    private long executionMemory;

    // Constructor
    public Response(String message, Object data, double executionTime, int
visitedNode, long executionMemory) {
        this.message = message;
        this.data = data;
        this.executionTime = executionTime;
        this.visitedNode = visitedNode;
        this.executionMemory = executionMemory;
    }

    // Getters and Setters
    public String getMessage() {
        return message;
    }
}

```

```
public void setMessage(String message) {
    this.message = message;
}

public Object getData() {
    return data;
}

public void setData(Object data) {
    this.data = data;
}

public double getExecutionTime() {
    return executionTime;
}

public void setExecutionTime(double executionTime) {
    this.executionTime = executionTime;
}

public int getVisitedNode() {
    return visitedNode;
}

public void setVisitedNode(int visitedNode) {
    this.visitedNode = visitedNode;
}

public long getExecutionMemory() {
    return executionMemory;
}

public void setExecutionMemory(int executionMemory) {
    this.executionMemory = executionMemory;
}

/**
 * Mengonversi objek Response menjadi representasi JSON
ResponseEntity.
```

```

/*
 * @return ResponseEntity yang berisi representasi JSON dari objek
Response
 */
public ResponseEntity<String> json() {
    ObjectMapper mapper = new ObjectMapper();
    try {
        String json = mapper.writeValueAsString(this);
        return ResponseEntity.status(HttpStatus.OK).body(json);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("{\"message\":
:\"Error processing JSON\"}");
    }
}
}

```

4.2.8 Api.java

```

package wordladder.tucil3_13522124;

import java.io.IOException;
import java.util.Set;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
public class Api {
    @GetMapping(value = "/api", produces = "application/json")
    public ResponseEntity<Object> api(
        @RequestParam(value = "start") String startWord,
        @RequestParam(value = "target") String endWord,
        @RequestParam(value = "algorithm") String method) {
        startWord = startWord.toLowerCase();
        endWord = endWord.toLowerCase();
    }
}

```

```

        Set<String> dictionary = null;
        try {
            dictionary =
DictionaryWord.loadDictionary("src/main/java/wordladder/tucil3_13522124/Di
ctionary/words_alpha.txt");
        } catch (IOException e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
reading dictionary file: " + e.getMessage());
        }
        if(!dictionary.contains(startWord) ||
!dictionary.contains(endWord)){
            return ResponseEntity.ok(new Response("Start Word or End Word
is not a valid word.",null,0,0,0));
        }
        Result path = null;
        Runtime runtime = Runtime.getRuntime();
        runtime.gc();
        long startTime = System.nanoTime();
        long memoryBefore = runtime.freeMemory();
        if(method.equals("UCS")){
            path = UCS.wordLadder(startWord, endWord, dictionary);
        }
        else if(method.equals("ASTAR")){
            path = Astar.wordLadder(startWord, endWord, dictionary);
        } else {
            path = GreedyBestFirstSearch.findWordLadder(startWord,
endWord, dictionary);
        }
        long memoryAfter = runtime.freeMemory();
        long endTime = System.nanoTime();
        double executionTime = (endTime-startTime)/1000000.0;
        long executionMemory = (memoryBefore - memoryAfter)/1000;
        return ResponseEntity.ok(new Response(null, path.getPath(),
executionTime, path.getVisitedCount(),executionMemory));
    }
}

```

4.2.9 page.js

```
"use client"
```

```

import React, { useState } from 'react';
import StickyBar from '../components/stickybar';
import WordLadderGrid from '../components/wordladdergrid';
import { BeatLoader } from "react-spinners";

export default function Home() {
    //Fungsi untuk membuat delay pada website
    function delay(ms) {
        return new Promise(resolve => setTimeout(resolve, ms));
    }

    // State untuk menyimpan panjang kata, kata awal, dan kata akhir
    const [length, setLength] = useState('');
    const [startWord, setStartWord] = useState('');
    const [endWord, setEndWord] = useState('');
    const [activeAlgorithm, setActiveAlgorithm] = useState('');
    const [results, setResults] = useState(null);
    const [errorMessage, setErrorMessage] = useState(null);
    const [submitted, setSubmitted] = useState(false);
    const [loading, setLoading] = useState(false);

    const baseStyle = "mx-4 border-black rounded px-7 pb-[8px] pt-[10px]
font-black text-sm uppercase leading-normal transition duration-150
ease-in-out focus:outline-none focus:ring-0";

    const dynamicStyle = (isActive) =>
        `${baseStyle} ${isActive ? 'text-black border-neutral-100
bg-neutral-500 bg-opacity-50' : 'text-black border-black border-2
hover:border-neutral-100 hover:bg-neutral-500 hover:bg-opacity-10
hover:text-neutral-40'}`;

    const handleAlgorithmClick = (algorithm) => {
        console.log("Algorithm", algorithm);
        setActiveAlgorithm(algorithm);
    };

    const handleSubmit = async (event) => {
        setLoading(true);
        setSubmitted(false);
        event.preventDefault();
    };
}

```

```

if (startWord === '' || endWord === '') {
    setErrorMessage("Please complete the start and the end word.");
    await delay(1500);
    setLoading(false)
    setErrorMessage(null);
    return;
}
else if (activeAlgorithm === ''){
    setErrorMessage("Please choose the algorithm.");
    await delay(1500);
    setLoading(false)
    setErrorMessage(null);
    return;
}
else if (startWord.length < length || endWord.length < length){
    setErrorMessage("Start Word or End Word not suitable with the length input.");
    await delay(1500);
    setLoading(false)
    setErrorMessage(null);
    return;
}
else if (startWord === endWord){
    setErrorMessage("Start Word and End Word must a different value.");
    await delay(1500);
    setLoading(false)
    setErrorMessage(null);
    return;
}
try {
    const response = await
fetch(`http://localhost:8080/api?start=${startWord}&target=${endWord}&algorithm=${activeAlgorithm}`, {
        method: 'GET', // Metode HTTP yang digunakan
        headers: {
            'Content-Type': 'application/json'
        },
    });
    if (!response.ok) {
        throw new Error(`HTTP error! Status: ${response.status}`);
    }
}

```

```

        }

        const data = await response.json(); // Mengonversi response menjadi
JSON

        if(data.message != null){
            setErrorMessage(data.message);
            await delay(1500);
            setLoading(false);
            setErrorMessage(null);
        }
        else{
            console.log(data);
            setResults(data);
        }
    } catch (error) {
        console.error("Error:", error);
        setErrorMessage(error.message || "An unexpected error occurred");
    } finally {
        setLoading(false);
        setSubmitted(true);
    }
}

return (
<html>
<body>
<div className="flex flex-col w-full items-center bg-white">
<StickyBar />
<div className="mt-5 ">
    <label htmlFor="lengthInput" className="flex text-sm font-medium
text-black justify-center">
        Panjang Kata:
    </label>
    <input
        type="number"
        id="lengthInput"
        value={length}
        onChange={(e) => {
            setLength(e.target.value);
            setStartWord(' ');
        }}
    />
</div>
</div>

```

```

        setEndWord('');
        setActiveAlgorithm('');
    }
}



IF2211 Strategi Algoritma - Tugas Kecil 3 | 34


```

```

        value={endWord}
        onChange={(e) => {
          const { value } = e.target;
          setEndWord(value); // Mengatur nilai endWord dengan nilai
dari input
          setSubmitted(false);
        }}
        maxLength={length}
        className="mt-1 block w-full px-3 py-2 border text-gray-700
border-gray-300 rounded-md shadow-sm focus:outline-none
focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
      />
    <h4 className="mb-2 text-xl font-semibold">Algorithm Type</h4>
    <div>
      <div className='flex justify-center'>
        <button
          type="button"
          className={dynamicStyle(activeAlgorithm === 'UCS')}
          onClick={() => handleAlgorithmClick('UCS')}
          data-twe-ripple-init
          data-twe-ripple-color="light"
        >
          UCS
        </button>
        <button
          type="button"
          className={dynamicStyle(activeAlgorithm === 'ASTAR')}
          onClick={() => handleAlgorithmClick('ASTAR')}
          data-twe-ripple-init
          data-twe-ripple-color="light"
        >
          A*
        </button>
        <button
          type="button"
          className={dynamicStyle(activeAlgorithm === 'GBFS')}
          onClick={() => handleAlgorithmClick('GBFS')}
          data-twe-ripple-init
          data-twe-ripple-color="light"
        >

```

```

        Greedy Best First Search
    </button>
    </div>
    {errorMessage && (
        <div className='text-black text-center mt-4
mb-4'>{errorMessage}</div>
    ) }
    {loading && (
        <div className="flex justify-center items-center mt-[25px]
mb-[50px]">
            <BeatLoader color="#000000" loading={loading} size={15}>
/>
            <p className="ml-2 text-black">Loading...</p>
        </div>
    ) }
    {!loading && (<div className='flex justify-center'><button
type="submit"
            className='border-black text-black mt-4 mx-4 mb-[15px] rounded
border-2 px-7 pb-[8px] pt-[10px] text-sm font-bold uppercase
leading-normal transition duration-150 ease-in-out
hover:border-neutral-100 hover:bg-neutral-500 hover:bg-opacity-10
hover:text-black focus:border-neutral-100 focus:text-neutral-100
focus:outline-none focus:ring-0 active:border-neutral-200
active:text-neutral-200 dark:hover:bg-neutral-100
dark:hover:bg-opacity-10'
            data-twe-ripple-init
            data-twe-ripple-color="light">
                Submit!
            </button></div>
    ) }
    {results && results.data.length !== 0 && submitted && (
        <div className="p-8 flex flex-col items-center">
            <WordLadderGrid words={results.data} />
            <p className="text-center mt-4 text-xl text-black">Found
Word Ladder Solution from <strong>{startWord}</strong> to
<strong>{endWord}</strong> in <strong>{results.executionTime}
milliseconds</strong>!</p>
            <p className="text-black text-center mt-4
text-xl">Solution Length: <strong>{results.data.length}</strong></p>
    )
)
}

```

```

        <p className="text-black text-center mt-4 text-xl">Nodes
Traversed: <strong>{results.visitedNode}</strong></p>
        <p className="text-black text-center mt-4
text-xl">Memory Used: <strong>{results.executionMemory} KB</strong></p>
    </div>
)
{
    results && results.data.length == 0 && submitted && (
        <div className="p-8 flex flex-col items-center">
        <WordLadderGrid words={results.data} />
        <p className="text-center mt-4 text-xl text-black">No Word
Ladder Solution found from <strong>{startWord}</strong> to
<strong>{endWord}</strong> in <strong>{results.executionTime}
milliseconds</strong>!</p>
        <p className="text-black text-center mt-4 text-xl">Nodes
Traversed: <strong>{results.visitedNode}</strong></p>
        <p className="text-black text-center mt-4 text-xl">Memory
Used: <strong>{results.executionMemory} KB</strong></p>
    </div>
)
</div>
</div>
</form>
</div>
)
)
</div>
</body>
</html>
) ;
}

```

4.2.10 stickybar.js

```

const StickyBar = () => {
    return (
        <div className="h-[75px] sticky top-0 bg-white bg-opacity-50 py-4
z-20 mx-12 flex text-black items-center justify-around w-full
border-gray-300 border-[2px]">
        <p className="text-black font-bold text-3xl"> Word Ladder
Solver</p>
        <p>Made by&ampnbsp;
    
```

```

        <code className="font-mono font-bold">Aland Mulia Pratama -
13522124</code></p>
    </div>
) ;
}

export default StickyBar;

```

4.2.11 wordladdergrid.js

```

function WordLadderGrid({ words }) {
    return (
        <div className={`${`grid grid-rows-${words.length} gap-2`} `}>
            {words.map((word, wordIndex) => (
                <div key={wordIndex} className="flex w-full">
                    {word.toUpperCase().split('').map((char, charIndex) => {
                        let bgColor = 'bg-gray-200';
                        if (wordIndex > 0) {
                            // Membandingkan dengan kata sebelumnya
                            const endChar = words[words.length - 1][charIndex] || '';
                            if (char === endChar.toUpperCase()) {
                                bgColor = 'bg-green-500';
                            }
                        }
                    })
                    return (
                        <div key={charIndex} className={`${`w-16 h-16 flex
justify-center items-center border border-gray-300 ${bgColor} text-xl
text-black font-bold mx-2 my-2`} `}>
                            {char}
                        </div>
                    );
                ))}
            </div>
        ) );
}

export default WordLadderGrid;

```

BAB V

PENGUJIAN DAN ANALISIS ALGORITMA DALAM PERMAINAN WORD LADDER

5.1 Validasi Program

Word Ladder Solver

Made by Aland Mulia Pratama - 13522124

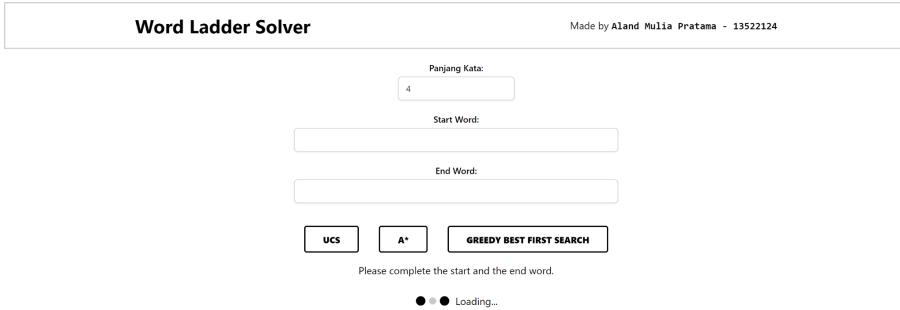
Panjang Kata:

Start Word:

End Word:

Please complete the start and the end word.

● ● ● Loading...



Gambar 6. Validasi input kosong

Word Ladder Solver

Made by Aland Mulia Pratama - 13522124

Panjang Kata:

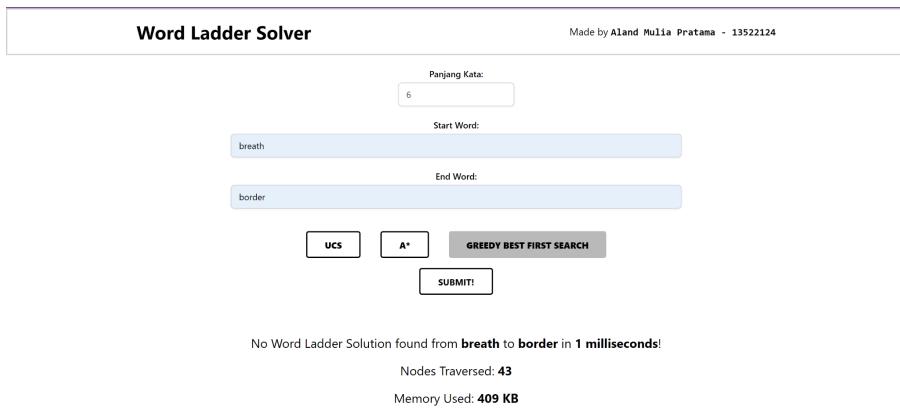
Start Word:

End Word:

No Word Ladder Solution found from **breath** to **border** in **1 milliseconds!**

Nodes Traversed: **43**

Memory Used: **409 KB**



Gambar 7. Validasi tidak ada solusi yang memenuhi

Word Ladder Solver

Made by Aland Mulia Pratama - 13522124

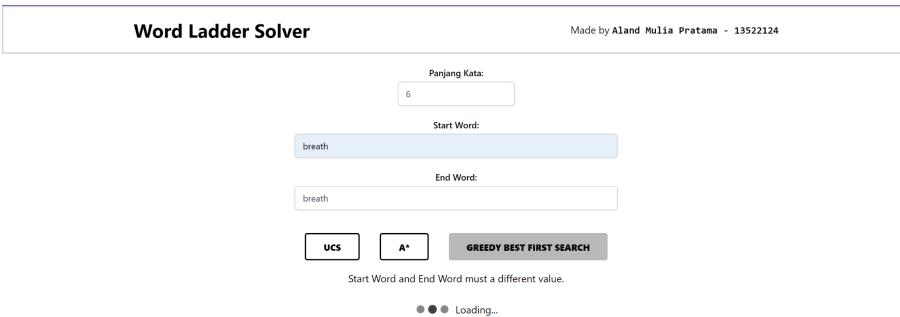
Panjang Kata:

Start Word:

End Word:

Start Word and End Word must a different value.

● ● ● Loading...



Gambar 8. Validasi input startWord dan endWord sama

The screenshot shows the 'Word Ladder Solver' interface. At the top, it says 'Panjang Kata: 5'. Below that, 'Start Word:' contains 'makan' and 'End Word:' contains 'minum'. At the bottom, there are three algorithm selection buttons: 'UCS' (selected), 'A*', and 'GREEDY BEST FIRST SEARCH'. A message below the buttons states 'Start Word or End Word is not a valid word.' and '● ● ● Loading...'. The code at the bottom right reads 'Made by Aland Mulia Pratama - 13522124'.

Gambar 9. Validasi startWord dan endWord bukan kata yang valid pada kamus yang digunakan

The screenshot shows the 'Word Ladder Solver' interface. At the top, it says 'Panjang Kata: 4'. Below that, 'Start Word:' contains 'sand' and 'End Word:' contains 'grit'. At the bottom, there are three algorithm selection buttons: 'UCS' (selected), 'A*', and 'GREEDY BEST FIRST SEARCH'. A message below the buttons states 'Please choose the algorithm.' and '● ● ● Loading...'. The code at the bottom right reads 'Made by Aland Mulia Pratama - 13522124'.

Gambar 10. Validasi algoritma belum dipilih

The screenshot shows the 'Word Ladder Solver' interface. At the top, it says 'Panjang Kata: 5'. Below that, 'Start Word:' contains 'sand' and 'End Word:' contains 'grit'. At the bottom, there are three algorithm selection buttons: 'UCS' (selected), 'A*' (disabled), and 'GREEDY BEST FIRST SEARCH' (disabled). A message below the buttons states 'Start Word or End Word not suitable with the length input.' and '● ● ● Loading...'. The code at the bottom right reads 'Made by Aland Mulia Pratama - 13522124'.

Gambar 11. Validasi panjang kata awal dan tujuan berbeda

5.2 Pengujian Algoritma UCS, A*, dan Greedy Best First Search

TEST 1		
Word Ladder Solver		Made by Aland Mulia Pratama - 13522124
ALGORITMA	Memory per Nodes (KB/Nodes)	HASIL
UCS	5.343	<p>Panjang Kata: 4</p> <p>Start Word: sand</p> <p>End Word: grit</p> <p>UCS A* GREEDY BEST FIRST SEARCH</p> <p>SUBMIT!</p> <pre> graph TD sand[sand] --> san1[san] san1 --> sat1[sat] sat1 --> sat2[sat] sat2 --> grit[grit] </pre> <p>Found Word Ladder Solution from sand to grit in 33.1205 milliseconds!</p> <p>Solution Length: 6</p> <p>Nodes Traversed: 2262</p> <p>Memory Used: 12087 KB</p>

A*	10.3	<p>Found Word Ladder Solution from sand to grit in 0.7641 milliseconds!</p> <p>Solution Length: 6</p> <p>Nodes Traversed: 21</p> <p>Memory Used: 217 KB</p>
GBFS	1.39	<p>Found Word Ladder Solution from sand to grit in 0.3797 milliseconds!</p> <p>Solution Length: 7</p> <p>Nodes Traversed: 46</p> <p>Memory Used: 64 KB</p>

TEST 2		
Word Ladder Solver		Made by Aland Mulia Pratama - 13522124
ALGORITMA	Memory per Nodes	HASIL
UCS	4.284	<p>P O P</p> <p>F O P</p> <p>F O N</p> <p>F U N</p> <p>Found Word Ladder Solution from pop to fun in 9.8199 milliseconds!</p> <p>Solution Length: 4</p> <p>Nodes Traversed: 538</p> <p>Memory Used: 2305 KB</p>

A*	7.4	<pre> P O P F O P F O N F U N </pre> <p>Found Word Ladder Solution from pop to fun in 0.2088 milliseconds!</p> <p>Solution Length: 4</p> <p>Nodes Traversed: 5</p> <p>Memory Used: 37 KB</p>
GBFS	2.521	<pre> P O P F O P F O N F U N </pre> <p>Found Word Ladder Solution from pop to fun in 0.1648 milliseconds!</p> <p>Solution Length: 4</p> <p>Nodes Traversed: 46</p> <p>Memory Used: 116 KB</p>

TEST 3

Word Ladder Solver

Made by Aland Mulia Pratama - 13522124

Panjang Kata:	<input type="text" value="5"/>
Start Word:	<input type="text" value="loose"/>
End Word:	<input type="text" value="swell"/>
<input type="button" value="UCS"/> <input type="button" value="A*"/> <input type="button" value="GREEDY BEST FIRST SEARCH"/>	
<input type="button" value="SUBMIT!"/>	

ALGORITMA	Memory per Nodes	HASIL
UCS	2.142	 <p>Found Word Ladder Solution from loose to swell in 99.4802 milliseconds!</p> <p>Solution Length: 12</p> <p>Nodes Traversed: 6689</p> <p>Memory Used: 14333 KB</p>

A*	7.09	<table border="1"> <tbody> <tr><td>L</td><td>O</td><td>O</td><td>S</td><td>E</td></tr> <tr><td>N</td><td>O</td><td>O</td><td>S</td><td>E</td></tr> <tr><td>N</td><td>O</td><td>I</td><td>S</td><td>E</td></tr> <tr><td>P</td><td>O</td><td>I</td><td>S</td><td>E</td></tr> <tr><td>P</td><td>E</td><td>I</td><td>S</td><td>E</td></tr> <tr><td>S</td><td>E</td><td>I</td><td>S</td><td>E</td></tr> <tr><td>S</td><td>E</td><td>I</td><td>N</td><td>E</td></tr> <tr><td>S</td><td>P</td><td>I</td><td>N</td><td>E</td></tr> <tr><td>S</td><td>P</td><td>I</td><td>L</td><td>E</td></tr> <tr><td>S</td><td>P</td><td>I</td><td>L</td><td>L</td></tr> <tr><td>S</td><td>P</td><td>E</td><td>L</td><td>L</td></tr> <tr><td>S</td><td>W</td><td>E</td><td>L</td><td>L</td></tr> </tbody> </table> <p>Found Word Ladder Solution from loose to swell in 13.2127 milliseconds! Solution Length: 12 Nodes Traversed: 764 Memory Used: 5417 KB</p>	L	O	O	S	E	N	O	O	S	E	N	O	I	S	E	P	O	I	S	E	P	E	I	S	E	S	E	I	S	E	S	E	I	N	E	S	P	I	N	E	S	P	I	L	E	S	P	I	L	L	S	P	E	L	L	S	W	E	L	L
L	O	O	S	E																																																										
N	O	O	S	E																																																										
N	O	I	S	E																																																										
P	O	I	S	E																																																										
P	E	I	S	E																																																										
S	E	I	S	E																																																										
S	E	I	N	E																																																										
S	P	I	N	E																																																										
S	P	I	L	E																																																										
S	P	I	L	L																																																										
S	P	E	L	L																																																										
S	W	E	L	L																																																										

GBFS	1.349	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>L</td><td>O</td><td>O</td><td>S</td><td>E</td></tr> <tr><td>L</td><td>O</td><td>O</td><td>I</td><td>E</td></tr> <tr><td>L</td><td>O</td><td>U</td><td>I</td><td>E</td></tr> <tr><td>L</td><td>O</td><td>U</td><td>I</td><td>S</td></tr> <tr><td>L</td><td>O</td><td>U</td><td>T</td><td>S</td></tr> <tr><td>L</td><td>O</td><td>O</td><td>T</td><td>S</td></tr> <tr><td>S</td><td>O</td><td>O</td><td>T</td><td>S</td></tr> <tr><td>S</td><td>W</td><td>O</td><td>T</td><td>S</td></tr> <tr><td>S</td><td>W</td><td>A</td><td>T</td><td>S</td></tr> <tr><td>S</td><td>W</td><td>A</td><td>G</td><td>S</td></tr> <tr><td>S</td><td>W</td><td>A</td><td>G</td><td>E</td></tr> <tr><td>S</td><td>W</td><td>A</td><td>L</td><td>E</td></tr> <tr><td>S</td><td>C</td><td>A</td><td>L</td><td>E</td></tr> <tr><td>S</td><td>C</td><td>A</td><td>L</td><td>L</td></tr> <tr><td>S</td><td>H</td><td>A</td><td>L</td><td>L</td></tr> <tr><td>S</td><td>H</td><td>E</td><td>L</td><td>L</td></tr> <tr><td>S</td><td>W</td><td>E</td><td>L</td><td>L</td></tr> </table> <p>Found Word Ladder Solution from loose to swell in 2.2874 milliseconds! Solution Length: 17 Nodes Traversed: 189 Memory Used: 255 KB</p>	L	O	O	S	E	L	O	O	I	E	L	O	U	I	E	L	O	U	I	S	L	O	U	T	S	L	O	O	T	S	S	O	O	T	S	S	W	O	T	S	S	W	A	T	S	S	W	A	G	S	S	W	A	G	E	S	W	A	L	E	S	C	A	L	E	S	C	A	L	L	S	H	A	L	L	S	H	E	L	L	S	W	E	L	L
L	O	O	S	E																																																																																			
L	O	O	I	E																																																																																			
L	O	U	I	E																																																																																			
L	O	U	I	S																																																																																			
L	O	U	T	S																																																																																			
L	O	O	T	S																																																																																			
S	O	O	T	S																																																																																			
S	W	O	T	S																																																																																			
S	W	A	T	S																																																																																			
S	W	A	G	S																																																																																			
S	W	A	G	E																																																																																			
S	W	A	L	E																																																																																			
S	C	A	L	E																																																																																			
S	C	A	L	L																																																																																			
S	H	A	L	L																																																																																			
S	H	E	L	L																																																																																			
S	W	E	L	L																																																																																			

TEST 4

Word Ladder Solver Made by Aland Mulia Pratama - 13522124

Panjang Kata:

Start Word:

End Word:

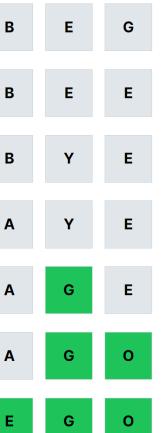
ALGORITMA	Memory per Nodes	HASIL

UCS	3.384	<table border="1" data-bbox="975 206 1183 1056"> <tr><td>D</td><td>E</td><td>P</td><td>U</td><td>T</td><td>Y</td></tr> <tr><td>D</td><td>E</td><td>P</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>P</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>F</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>F</td><td>U</td><td>S</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>U</td><td>S</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>U</td><td>N</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>I</td><td>N</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>I</td><td>N</td><td>T</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>I</td><td>N</td><td>T</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>I</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>E</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>E</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>E</td><td>E</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>S</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>L</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>L</td><td>L</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>E</td><td>T</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>O</td><td>T</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>O</td><td>N</td></tr> </table> <p>Found Word Ladder Solution from deputy to balloon in 72.7601 milliseconds!</p> <p>Solution Length: 22 Nodes Traversed: 6794 Memory Used: 22996 KB</p>	D	E	P	U	T	Y	D	E	P	U	T	E	R	E	P	U	T	E	R	E	F	U	T	E	R	E	F	U	S	E	R	E	T	U	S	E	R	E	T	U	N	E	R	E	T	I	N	E	R	E	T	I	N	T	R	E	M	I	N	T	R	E	M	I	N	D	R	E	M	E	N	D	R	E	S	E	N	D	R	E	S	E	E	D	R	E	S	T	E	D	B	E	S	T	E	D	B	E	L	T	E	D	B	E	L	L	E	D	B	A	L	L	E	D	B	A	L	L	E	T	B	A	L	L	O	T	B	A	L	L	O	N
D	E	P	U	T	Y																																																																																																																																	
D	E	P	U	T	E																																																																																																																																	
R	E	P	U	T	E																																																																																																																																	
R	E	F	U	T	E																																																																																																																																	
R	E	F	U	S	E																																																																																																																																	
R	E	T	U	S	E																																																																																																																																	
R	E	T	U	N	E																																																																																																																																	
R	E	T	I	N	E																																																																																																																																	
R	E	T	I	N	T																																																																																																																																	
R	E	M	I	N	T																																																																																																																																	
R	E	M	I	N	D																																																																																																																																	
R	E	M	E	N	D																																																																																																																																	
R	E	S	E	N	D																																																																																																																																	
R	E	S	E	E	D																																																																																																																																	
R	E	S	T	E	D																																																																																																																																	
B	E	S	T	E	D																																																																																																																																	
B	E	L	T	E	D																																																																																																																																	
B	E	L	L	E	D																																																																																																																																	
B	A	L	L	E	D																																																																																																																																	
B	A	L	L	E	T																																																																																																																																	
B	A	L	L	O	T																																																																																																																																	
B	A	L	L	O	N																																																																																																																																	

A*	8.544	<table border="1" data-bbox="962 200 1191 1140"> <tr><td>D</td><td>E</td><td>P</td><td>U</td><td>T</td><td>Y</td></tr> <tr><td>D</td><td>E</td><td>P</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>P</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>F</td><td>U</td><td>T</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>F</td><td>U</td><td>S</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>U</td><td>S</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>U</td><td>N</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>I</td><td>N</td><td>E</td></tr> <tr><td>R</td><td>E</td><td>T</td><td>I</td><td>N</td><td>T</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>I</td><td>N</td><td>T</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>I</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>M</td><td>E</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>E</td><td>N</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>E</td><td>E</td><td>D</td></tr> <tr><td>R</td><td>E</td><td>S</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>S</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>L</td><td>T</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>E</td><td>L</td><td>L</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>E</td><td>D</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>O</td><td>T</td></tr> <tr><td>B</td><td>A</td><td>L</td><td>L</td><td>O</td><td>N</td></tr> </table> <p>Found Word Ladder Solution from deputy to balloon in 18.8275 milliseconds! Solution Length: 22 Nodes Traversed: 1575 Memory Used: 13458 KB</p>	D	E	P	U	T	Y	D	E	P	U	T	E	R	E	P	U	T	E	R	E	F	U	T	E	R	E	F	U	S	E	R	E	T	U	S	E	R	E	T	U	N	E	R	E	T	I	N	E	R	E	T	I	N	T	R	E	M	I	N	T	R	E	M	I	N	D	R	E	M	E	N	D	R	E	S	E	N	D	R	E	S	E	E	D	R	E	S	T	E	D	B	E	S	T	E	D	B	E	L	T	E	D	B	E	L	L	E	D	B	A	L	L	E	D	B	A	L	L	O	T	B	A	L	L	O	N
D	E	P	U	T	Y																																																																																																																											
D	E	P	U	T	E																																																																																																																											
R	E	P	U	T	E																																																																																																																											
R	E	F	U	T	E																																																																																																																											
R	E	F	U	S	E																																																																																																																											
R	E	T	U	S	E																																																																																																																											
R	E	T	U	N	E																																																																																																																											
R	E	T	I	N	E																																																																																																																											
R	E	T	I	N	T																																																																																																																											
R	E	M	I	N	T																																																																																																																											
R	E	M	I	N	D																																																																																																																											
R	E	M	E	N	D																																																																																																																											
R	E	S	E	N	D																																																																																																																											
R	E	S	E	E	D																																																																																																																											
R	E	S	T	E	D																																																																																																																											
B	E	S	T	E	D																																																																																																																											
B	E	L	T	E	D																																																																																																																											
B	E	L	L	E	D																																																																																																																											
B	A	L	L	E	D																																																																																																																											
B	A	L	L	O	T																																																																																																																											
B	A	L	L	O	N																																																																																																																											

		<p>The diagram illustrates a word ladder puzzle where each word is represented as a 4x4 grid of letters. The path starts at 'DEPUTY' (top row) and ends at 'BALLOON' (bottom row). The words in the sequence are: DEPUTY, DEPUTE, REPUTE, REFUTE, REFUZE, RETUZE, RETUNE, RETINE, RATTINE, PATTINE, PATTINS, PATTIOS, RATIOS, RATION, RATTION, RATTEN, BATTEEN, BATTEN, BAITER, BAILER, BALLER, BALLET, BALLOT, and BALLOON. Green squares highlight the letters that change between adjacent words in the ladder.</p> <p>Found Word Ladder Solution from deputy to balloon in 1.705 milliseconds! Solution Length: 24 Nodes Traversed: 268 Memory Used: 893 KB</p>
GBFS	3.332	

TEST 5		
Word Ladder Solver		Made by Aland Mulia Pratama - 13522124
Panjang Kata: <input type="text" value="3"/>		
Start Word: <input type="text" value="beg"/>		
End Word: <input type="text" value="ego"/>		
<input type="button" value="UCS"/> <input type="button" value="A*"/> <input type="button" value="GREEDY BEST FIRST SEARCH"/>		
<input type="button" value="SUBMIT!"/>		
ALGORITMA	Memory per Nodes	HASIL

UCS	4.191	 <p>Found Word Ladder Solution from beg to ego in 7.0058 milliseconds! Solution Length: 7 Nodes Traversed: 945 Memory Used: 3961 KB</p>
A*	4.521	 <p>Found Word Ladder Solution from beg to ego in 2.2606 milliseconds! Solution Length: 7 Nodes Traversed: 324 Memory Used: 1465 KB</p>

GBFS	1.968	<p>A word ladder diagram showing the path from 'beg' to 'ego'. The words are arranged in a grid:</p> <table border="1"> <tr><td>B</td><td>E</td><td>G</td></tr> <tr><td>K</td><td>E</td><td>G</td></tr> <tr><td>K</td><td>E</td><td>Y</td></tr> <tr><td>K</td><td>A</td><td>Y</td></tr> <tr><td>K</td><td>A</td><td>T</td></tr> <tr><td>E</td><td>A</td><td>T</td></tr> <tr><td>E</td><td>A</td><td>R</td></tr> <tr><td>E</td><td>R</td><td>R</td></tr> <tr><td>E</td><td>R</td><td>G</td></tr> <tr><td>E</td><td>G</td><td>G</td></tr> <tr><td>E</td><td>G</td><td>O</td></tr> </table> <p>Found Word Ladder Solution from beg to ego in 0.4073 milliseconds! Solution Length: 11 Nodes Traversed: 160 Memory Used: 315 KB</p>	B	E	G	K	E	G	K	E	Y	K	A	Y	K	A	T	E	A	T	E	A	R	E	R	R	E	R	G	E	G	G	E	G	O
B	E	G																																	
K	E	G																																	
K	E	Y																																	
K	A	Y																																	
K	A	T																																	
E	A	T																																	
E	A	R																																	
E	R	R																																	
E	R	G																																	
E	G	G																																	
E	G	O																																	

TEST 6

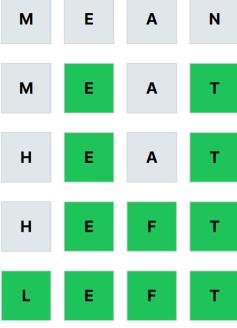
Word Ladder Solver
Made by Aland Mulia Pratama - 13522124

Panjang Kata:

Start Word:

End Word:

ALGORITMA	Memory per Nodes	HASIL

UCS	5.4	 <p>Found Word Ladder Solution from mean to left in 7.5709 milliseconds! Solution Length: 5 Nodes Traversed: 1007 Memory Used: 5438 KB</p>
A*	11.038	 <p>Found Word Ladder Solution from mean to left in 0.5248 milliseconds! Solution Length: 5 Nodes Traversed: 26 Memory Used: 287 KB</p>

GBFS	2.73	 <p>Found Word Ladder Solution from mean to left in 0.3489 milliseconds!</p> <p>Solution Length: 6</p> <p>Nodes Traversed: 107</p> <p>Memory Used: 293 KB</p>

BAB VI

PENUTUP

6.1 Kesimpulan

Dalam Tugas Kecil 3 IF2211 Strategi Algoritma untuk mencari solusi permainan Word Ladder menggunakan algoritma UCS, A*, dan Greedy Best First Search (GBFS) memiliki tujuan yang sama yaitu mencari sebuah solusi dari kata awal menuju kata akhir. Namun, terdapat beberapa kelebihan, kekurangan dan perbedaan cara kerja dari ketiga algoritma tersebut untuk mencari sebuah solusi Word Ladder.

Dalam kasus word ladder, Uniform Cost Search (UCS) menggunakan strategi greedy yang diperluas berdasarkan biaya terkecil. Karena biaya dari satu node ke node lainnya sama, maka prinsip Greedy dari algoritma UCS tidak dapat diterapkan sehingga urutan node yang dibangkitkan dan hasil path yang dihasilkan akan sama dengan algoritma Breadth First Search (BFS). Namun, dalam pencarian yang kompleks penggunaan memory dalam mengunjungi satu nodes lebih unggul algoritma UCS daripada algoritma A*. Dapat dipastikan juga bahwa solusi yang dihasilkan algoritma UCS dijamin optimal dari panjang kata yang dibutuhkan untuk menyelesaikan solusi.

Algoritma A* merupakan algoritma pencarian graf yang menggunakan suatu fungsi heuristik. Fungsi heuristik disini digunakan untuk memperhitungkan biaya terkecil dari simpul awal (start word) menuju simpul tujuan (end word). Fungsi heuristik dalam kasus pencarian solusi Word Ladder adalah *admissible* sehingga solusi yang dihasilkan menggunakan algoritma A* pasti optimal secara biaya. Perbedaan antara algoritma A* dan UCS adalah waktu dan memori eksekusi yang diperlukan lebih sedikit dibandingkan algoritma UCS. Jumlah node yang dikunjungi dalam Algoritma A* juga lebih sedikit dibandingkan pada algoritma UCS.

Sementara itu, algoritma Greedy Best First Search (GBFS) menggunakan fungsi heuristik yang mirip seperti A*. Hanya saja fungsi heuristik pada GBFS berfokus pada keputusan cepat dalam implementasinya. Tentu saja fungsi heuristik menyebabkan algoritma ini membutuhkan waktu dan memori eksekusi yang lebih sedikit dibandingkan algoritma UCS dan A*. Hanya saja, algoritma GBFS ini tidak selalu menghasilkan solusi yang optimum global secara biaya atau jumlah kata yang diperlukan untuk menyelesaikan solusi. Dalam menyelesaikan permasalahan yang kompleks seringkali algoritma ini terjebak dalam solusi optimum lokal. Berdasarkan subbab 5.2, dari 6 pengujian, algoritma GBFS menghasilkan solusi yang sama optimalnya ketika mendapat input kata awal yaitu pop dan kata akhir yaitu fun.

Dalam kesimpulannya, ketiga algoritma ini memiliki kelebihan dan kekurangan masing-masing. Pemilihan algoritma A*, UCS, atau GBFS dapat disesuaikan berdasarkan kebutuhan penggunaan dan kasus permasalahan. Algoritma UCS baik digunakan untuk menyelesaikan permasalahan yang memiliki tingkat kompleksitas yang tinggi sedangkan algoritma A* akan optimal pada penyelesaian masalah yang sederhana. Pada kasus permasalahan yang kompleks, Algoritma UCS dapat unggul dari segi penggunaan

memory untuk tiap nodes yang dikunjungi dan waktu eksekusi yang dibutuhkan. Algoritma GBFS dapat digunakan untuk menyelesaikan masalah yang mengutamakan kecepatan waktu dibandingkan dengan ketepatan dari solusi permasalahan atau biaya untuk menyelesaikan solusi permasalahan.

6.2 Komentar

Dengan mengerjakan Tugas Kecil 3 ini, saya mendapat pengetahuan baru terhadap penggunaan framework spring-boot yang disediakan untuk bahasa pemrograman java. Dengan framework spring-boot, file java yang akan dirun tidak perlu dicompile seolah-olah bahasa ini dijalankan secara interpreter oleh framework tersebut. Selain itu, kami juga mengetahui kelebihan dan kekurangan dari algoritma A*, UCS, dan Greedy Best First Search sehingga kami dapat menerapkan algoritma yang sesuai untuk kasus permasalahan yang akan saya hadapi di masa yang akan datang.

6.3 Lampiran



Gambar 12. Icon GUI Word Ladder Solver

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

DAFTAR PUSTAKA

*A** search algorithm. GeeksforGeeks. (2024a, March 7).
<https://www.geeksforgeeks.org/a-search-algorithm/>

“Documentation - Tailwind CSS,” tailwindcss.com. <https://tailwindcss.com/docs>

GeeksforGeeks. (2023, April 20). *Uniform-cost search (dijkstra for large graphs)*.
<https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/>

GeeksforGeeks. (2024b, January 18). *Greedy best first search algorithm*.
<https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>

“Getting Started – React,” reactjs.org. <https://reactjs.org/docs>

Spring Boot. Spring Boot :: Spring Boot. (n.d.). <https://docs.spring.io/spring-boot/>