

LAPORAN TUGAS KECIL
IF2211 Strategi Algoritma
Penyelesaian Permainan Word Ladder
Menggunakan Algoritma UCS, Greedy Best First
Search, dan A*



Disusun oleh:

Immanuel Sebastian Girsang

(13522058)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
DASAR TEORI.....	2
1.1. Permainan WordLadder.....	2
1.2. Algoritma UCS (Uniform Cost Search).....	2
1.3. Algoritma GBFS (Greedy Best First Search).....	3
1.4. Algoritma A* (A Star).....	4
BAB II.....	5
ANALISIS DAN IMPLEMENTASI ALGORITMA.....	5
2.1. Analisis Strategi Umum.....	5
2.1.1. Analisis Simpul.....	5
2.1.2. Analisis Sisi.....	5
2.1.3. Analisis Strategi Implementasi.....	5
2.2. Analisis Cost UCS $g(n)$ dan Hubungannya dengan BFS.....	7
2.3. Penggunaan Heuristik dan admisibilitasnya.....	8
2.4. Analisis Cost GBFS dan jaminannya dengan solusi optimal.....	8
2.5. Analisis Cost A*.....	9
2.6. Analisis efisiensi A* dibanding UCS.....	9
2.7. Pembuatan map untuk mempermudah pencarian anak.....	10
BAB III.....	11
Source Code Program dan Penjelasan Class.....	11
3.1. Penjelasan Abstraksi Class serta Stuktur Kode.....	11
Gambar 1. Struktur Class.....	11
3.2. Penjelasan Class Algorithm.....	12
3.3. Penjelasan Class Node.....	13
3.4. Penjelasan Class MapGenerator.....	14
3.5. Penjelasan Class TextParser.....	14
3.6. Penjelasan Class WordLadderGameCLI.....	15
BAB IV.....	16
Analisis Perbandingan Algoritma.....	16
4.1. Uji Coba Dengan Tangkapan Layar.....	16
4.1.1. Test Case 1 (Tidak Ada Solusi).....	16
4.1.2. Test Case 2 ATOM to UNAU.....	19
4.1.3. Test Case 3 NYLON to ILLER.....	22
4.1.4. Test Case 4 ATLASES to CABARET.....	25
4.1.5. Test Case 5 TABLE to CROWN.....	28
4.1.6. Test Case 6 GIMLETS to TREEING.....	31
4.2. Analisis Optimalitas hasil.....	34
4.3. Analisis Waktu Eksekusi.....	34
4.4. Analisis memori yang Dibutuhkan.....	35

BAB V.....	36
Implementasi Bonus.....	36
5.1. Struktur GUI.....	36
5.2. Penjelasan Frame Masukan.....	36
5.3. Penjelasan Frame Hasil.....	37
BAB VI.....	38
KESIMPULAN.....	38
BAB VII.....	39
Lampiran.....	39
7.1. Source Code Class Algorithm.....	40
7.2. Source Code Class Node.....	42
7.3. Source Code Class Map Generator.....	43
7.4. Source Code Class Text Parser.....	44
7.5. Source Code Class WorldLadderGameCLI.....	45
7.6. Source Code Class WorldLadderGameGUI.....	47

BAB I

DASAR TEORI

1.1. Permainan WordLadder

Word ladder adalah permainan kata yang populer untuk berbagai usia. Diciptakan oleh Lewis Carroll pada tahun 1877, permainan ini juga dikenal dengan beberapa nama, termasuk Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf. Dalam permainan ini, pemain diberi dua kata, yaitu kata awal dan kata akhir, dan harus menemukan rangkaian kata yang menghubungkan keduanya. Setiap kata dalam rangkaian harus memiliki jumlah huruf yang sama, dan dua kata yang berurutan hanya boleh berbeda satu huruf. Tujuan permainan ini adalah menemukan solusi optimal, yakni rangkaian kata yang paling singkat untuk menghubungkan kata awal dan kata akhir.

1.2. Algoritma UCS (Uniform Cost Search)

Uniform Cost Search (UCS) adalah algoritma pencarian yang termasuk dalam kategori pencarian tanpa informasi (uninformed search) atau sering disebut sebagai pencarian buta (blind search). Ini berarti bahwa algoritma UCS tidak menggunakan informasi tambahan selain yang diberikan dalam definisi masalah.

Algoritma UCS dapat dianggap sebagai perpanjangan dari algoritma Breadth-First Search (BFS), tetapi dengan mempertimbangkan nilai atau biaya dari setiap jalur. UCS melakukan ekspansi simpul berdasarkan biaya jalur dari titik awal. Algoritma ini bertujuan untuk menemukan solusi yang optimal dengan memprioritaskan jalur yang memiliki biaya terendah.

1.3. Algoritma GBFS (Greedy Best First Search)

Greedy Best First Search (GBFS) adalah algoritma pencarian yang termasuk dalam kategori pencarian dengan informasi (informed search). Ini berarti bahwa algoritma ini menggunakan informasi tambahan atau pengetahuan khusus tentang masalah untuk membimbing pencarian menuju solusi. Dalam hal ini, informasi tambahan tersebut biasanya berupa "heuristic", yang merupakan perkiraan jarak atau biaya antara simpul saat ini dengan simpul tujuan.

Tidak seperti algoritma Uniform Cost Search (UCS), yang mempertimbangkan biaya aktual dari jalur, GBFS berfokus pada *heuristic cost*.

Heuristic cost adalah nilai yang memberikan gambaran mengenai seberapa dekat atau jauh simpul tertentu dari simpul tujuan. Fungsi heuristic digunakan untuk menghitung nilai perkiraan ini, biasanya berdasarkan informasi tentang topografi atau struktur dari masalah yang dihadapi.

Dalam prosesnya, GBFS memilih simpul berikutnya untuk diekspansi berdasarkan heuristic cost yang paling rendah. Dengan kata lain, GBFS "bernafsu" untuk mencapai tujuan secepat mungkin, memilih simpul yang terlihat paling menjanjikan untuk mendekati tujuan. Misalnya, dalam masalah jalur terpendek (*shortest path*), fungsi heuristic bisa mengukur jarak geometris (seperti jarak Euclidean atau Manhattan) dari simpul saat ini ke simpul tujuan.

Sementara GBFS memiliki keunggulan dalam hal kecepatan karena langsung menuju ke arah yang tampaknya paling menguntungkan, algoritma ini tidak menjamin solusi yang optimal atau paling efisien. Ini karena GBFS tidak memperhitungkan biaya dari jalur yang ditempuh, hanya fokus pada nilai heuristic. Sehingga, dalam beberapa kasus, algoritma ini bisa tersesat atau terjebak dalam jalur yang tampaknya menjanjikan tetapi sebenarnya lebih panjang atau lebih mahal.

1.4. Algoritma A* (A Star)

Algoritma A* (dikenal juga sebagai A Star) adalah salah satu metode pencarian terinformasi yang banyak digunakan dalam ilmu komputer dan kecerdasan buatan untuk menemukan jalur terpendek atau solusi optimal. Ini adalah algoritma yang cerdas karena memanfaatkan informasi tambahan untuk membantu membuat keputusan selama proses pencarian, berbeda dengan metode pencarian buta yang hanya mengikuti aturan sederhana tanpa informasi lebih lanjut tentang tujuan.

Dalam A*, simpul-simpul dalam struktur data diberi nilai atau bobot yang terdiri dari dua komponen utama: biaya aktual dari simpul awal (biasanya akar) ke simpul saat ini, dan biaya heuristik dari simpul saat ini ke simpul tujuan. Biaya heuristik adalah perkiraan tentang seberapa jauh simpul tertentu dari tujuan, dan biasanya dihitung menggunakan berbagai metode seperti jarak Manhattan, Euclidean, atau metode lainnya tergantung pada konteks masalahnya. Biaya heuristik inilah yang membuat A* menjadi algoritma terinformasi.

Algoritma ini berfungsi dengan mengevaluasi simpul-simpul dalam antrian prioritas, dengan simpul yang memiliki biaya gabungan (biaya aktual + biaya heuristik) terendah menjadi prioritas untuk dieksplorasi terlebih dahulu. Dalam setiap langkah, algoritma memilih simpul dengan total biaya terendah, lalu mengembangkan simpul itu dengan menambahkan simpul-simpul tetangga ke antrian prioritas. Proses ini terus berulang sampai simpul tujuan ditemukan atau sampai semua simpul telah dieksplorasi tanpa menemukan solusi. Dengan kata lain, A* merupakan gabungan antara algoritma UCS dengan algoritma GBFS.

BAB II

ANALISIS DAN IMPLEMENTASI ALGORITMA

2.1. Analisis Strategi Umum

Untuk dapat menyelesaikan permasalahan WordLadder, perlu dilakukan *mapping* dari setiap elemen permasalahan ke dalam hal yang lebih terstruktur dan dapat diselesaikan dengan menggunakan algoritma UCS, GBS maupun A*. Setelah itu, perlu digeneralisir bagaimana menyelesaikan komponen-komponen permasalahan tersebut dengan algoritma yang ada. Keunikan dari permainan ini adalah pada kata-kata valid yang ada. Tantangan muncul dari mencari kata yang pas untuk bisa dicapai dengan “sekali loncat” dari kata yang lain, dan itu menuju ke solusi.

2.1.1. Analisis Simpul

Simpul pada permasalahan ini merupakan setiap kata yang “valid” dalam bahasa Inggris menurut *dictionary* yang dijadikan acuan. Simpul-simpul inilah yang nanti akan dihitung *cost*-nya menggunakan algoritma-algoritma yang ada.

2.1.2. Analisis Sisi

Adanya sisi yang menghubungkan dua buah kata menandakan bahwa kedua kata tersebut bisa dicapai dengan hanya melakukan sekali penggantian huruf. Namun, perlu dipastikan juga bahwa kedua kata tersebut sama-sama valid.

2.1.3. Analisis Strategi Implementasi

Dengan simpul dan sisi yang sudah terdefinisi, selanjutnya perlu diperhatikan bagaimana mengimplementasikan ketiga algoritma tersebut. Sebenarnya, UCS, GBS dan A* memiliki langkah-langkah umum yang sama dalam memecahkan suatu masalah. Perbedaan dari ketiganya adalah *cost* yang menjadi dasar perhitungan masing-masing algoritma. Perhitungan *cost* yang digunakan oleh masing-masing algoritma akan dijelaskan di bagian selanjutnya. Pada bagian ini, diasumsikan *cost* sudah diketahui. Kita akan menggunakan sebuah *priority queue* yang melambangkan antrian dari simpul-simpul yang akan di ekspan. Hal yang menjadi prioritas pada antrian ini adalah simpul-simpul yang memiliki *cost* terendah. Kita juga akan memiliki 2 Map yaitu *visitedMap* untuk menyimpan simpul yang sudah pernah di **ekspan** dan juga siapa *parent*-nya, serta ***visitedDouble***

untuk menyimpan simpul yang sudah pernah di masukkan ke *queue* beserta dengan berapa *cost*nya. Langkah-langkah pemecahan masalahnya adalah sebagai berikut:

1. Buat simpul awal yaitu kata awal yang dimasukkan pengguna. Simpul ini akan menjadi awal ekspansi kita. Simpul ini memiliki *parent* berupa string kosong untuk menandakan bahwa dia simpul pertama. Simpan juga variable *counter* untuk mengetahui berapa banyak simpul yang sudah dikunjungi.
2. Masukkan simpul awal ke *priority queue*
3. Keluarkan antrian terdepan dari *priority queue*
4. Tambahkan nilai *counter*
5. Masukkan simpul saat ini ke dalam visitedMap
6. Cek apakah simpul saat ini merupakan tujuan kita. Jika iya, akhiri pencarian, dan iterasi balik visitedMap untuk melihat siapa yang menjadi orangtua dari simpul tujuan hingga ditemukan parent berupa string kosong yang merupakan kata awal, lalu kembalikan hasilnya. Jika tidak, lanjutkan ke langkah selanjutnya.
7. Lakukan ekspansi atau dalam kasus ini artinya adalah mencari semua kata valid dalam bahasa inggris yang **bisa dicapai dengan melakukan perubahan 1 huruf terhadap simpul yang sedang kita evaluasi.**
8. Hitung *cost* dari masing-masing anak dan ubah dia menjadi sebuah simpul dengan menyimpan *parent*, yaitu simpul yang sedang kita ekspansi serta berapa *cost* nya.
9. Untuk melakukan optimasi, maka kita hanya akan memasukkan anak yang **belum pernah di ekspan** dan juga memiliki *cost* yang **lebih murah** dari **anak dengan posisi yang sama yang pernah dimasukkan sebelumnya.** Hal ini dilakukan dengan mengecek apakah simpul saat ini ada di visitedMap yang sudah dibuat sebelumnya. Jika ada, jangan masukkan. Jika tidak ada, lanjutkan cek apakah anak saat ini sudah pernah dimasukkan ke visitedDouble. Jika tidak pernah, maka anak boleh dimasukkan ke *priority queue*, jika pernah maka cek apakah nilai yang disimpan di visitedDouble lebih kecil daripada *cost* dari anak saat ini.

Jika iya, maka jangan masukkan. Jika tidak, maka aman untuk dimasukkan ke *priority queue*.

10. Masukkan setiap anak yang memenuhi kondisi di langkah sebelumnya ke *priority queue*, dan tandai di Map visitedDouble dengan nilai mereka saat ini.
11. Cek apakah *priority queue* sudah kosong, jika iya maka artinya solusi tidak ditemukan. Jika tidak, maka lakukan lagi semua langkah mulai dari langkah nomor 3.

Untuk melakukan optimasi, dibuatlah suatu stuktur data *map* yang sudah menyimpan semua anak dari suatu kata dan disimpan di mesin lokal kita. Hal ini dibuat dengan struktur data Map yang dimana kuncinya adalah sebuah kata, dan nilainya adalah List dari semua kata yang bisa dicapai dengan mengganti 1 huruf kata tersebut. Dengan ini, langkah 6 menjadi lebih mudah sebab yang perlu dilakukan hanyalah melakukan *lookup* dari map yang ada berapa nilainya dengan kompleksitas $O(1)$. Penjelasan pembuatan *map* ini akan dijelaskan di bagian selanjutnya.

2.2. Analisis *Cost* UCS $g(n)$ dan Hubungannya dengan BFS

Pada permainan ini, *cost* dari algoritma UCS adalah kedalaman dari kata yang sedang dicari. Hal ini bisa didapatkan dengan menambahkan kedalaman dari simpul orangtua dengan 1. Rasionalisasi dari hal ini adalah karena pada permasalahan WordLadder, relatif tidak ada *cost* yang membedakan antara transformasi suatu kata ke kata lainnya yang valid. Misal perubahan dari kata “rood” ke “root” akan sama harganya secara aturan permainan dengan “rood” ke “mood”, yaitu 1 operasi penggantian huruf. Hal ini membuat bisa dibilang harga untuk setiap jalur yang valid menjadi anak dari suatu simpul adalah 1, atau secara matematis

$$g(n) = p(n) + 1$$

dengan $g(n) = \text{cost}$ dari simpul saat ini

$p(n) =$ kedalaman dari simpul orangtua n

Dengan pengetahuan ini, dapat disimpulkan bahwa UCS pada kasus WordLadder sebenarnya merupakan pencarian dengan BFS apabila dipandang dari

urutan simpul yang dibangkitkan. Hal ini disebabkan karena simpul yang berada di kedalaman yang sama akan memiliki cost yang sama dan simpul yang berada di kedalaman lebih dalam memiliki cost lebih besar sehingga pasti ditelusuri belakangan. Hal ini sama dengan BFS yang mencari simpul yang ada di kedalaman sama terlebih dahulu sebelum lanjut ke kedalaman selanjutnya. Dengan begitu, pasti dijamin UCS akan memberikan solusi yang optimal (paling pendek).

2.3. Penggunaan Heuristik dan admisibilitasnya

Pada kasus wordLadder, heuristik yang dapat dipilih adalah banyak huruf yang berbeda antara kata saat ini dengan kata tujuan. Misalkan kata “mood” dengan kata “rood” memiliki nilai heuristik 1 karena adanya 1 huruf berbeda yaitu r. Sedangkan kata “castle” dengan “lovers” memiliki nilai heuristik 6 sebab semua hurufnya berbeda.

Suatu heuristik dikatakan *admissible* apabila untuk setiap $h(n)$ maka nilai sebenarnya yang dibutuhkan mencapai *goal* dari n harus lebih besar atau sama dengan $h(n)$ atau secara matematis

$$h(n) \leq h^*(n)$$

Dengan $h(n)$ adalah nilai heuristik simpul n

$h^*(n)$ nilai sesungguhnya simpul n mencapai tujuan

Heuristik ini *admissible* karena untuk kasus apapun, langkah minimal yang dibutuhkan untuk merubah suatu kata menjadi kata tujuan (dengan kasus terbaik adalah semua kata yang menjadi kata perantaranya valid) adalah banyaknya huruf yang berbeda di antara mereka. Pada banyak kasus, cost ini jelas akan menjadi lebih besar sebab adanya *constraint* bahwa kata yang menjadi perantara perubahan mereka haruslah valid.

2.4. Analisis Cost GBFS dan jaminannya dengan solusi optimal

Dengan menggunakan fungsi heuristik yang sudah dijelaskan sebelumnya, maka **GBFS jelas tidak menjamin solusi optimal**. Hal ini disebabkan karena pada kasus wordLadder, *constraint* utama dari permainan ini adalah pada validitas kata yang digunakan. Bisa jadi, langkah terpendek yang dibutuhkan untuk mencapai

kata A menuju kata B, perlu menggunakan pergantian terhadap beberapa kata yang relatif terkesan “tidak penting” sebab tidak mendekatkan kita ke *goal node*, tetapi justru kata-kata inilah yang dapat “mensetup” kita agar kemudian bisa terjembatani oleh kata-kata yang memang secara langsung terhubung dengan kata tujuan.

Pada GBFS, kita relatif tidak mencari solusi terbaik dalam jangka panjang, melainkan hanya mengambil *local optima* dari setiap langkah. Hal ini kemudian menjadi sulit untuk membawa pada solusi optimal sebab nantinya kita akan selalu mencari local optimum yang bisa jadi membawa pada kata-kata yang tidak valid dan bahkan bisa berakhir tidak menemukan solusi sama sekali karena tidak adanya kata yang bisa menjembatani mereka.

2.5. Analisis Cost A*

Dengan sudah diketahuinya $h(n)$ serta $g(n)$ yang digunakan, maka cost untuk algoritma merupakan gabungan dari cost sebenarnya simpul saat ini (didapat dari penjelasan algoritma UCS) serta nilai heuristik yang dijelaskan pada bagian sebelumnya. Secara matematis, dapat dirumuskan sebagai berikut

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

dengan $f(n) = \text{cost A*}$ dari simpul n

$g(n) = \text{cost}$ dari algoritma UCS yang sudah dijelaskan sebelumnya

$h(n) = \text{cost}$ heuristik yang sudah dijelaskan sebelumnya

Karena sudah dijelaskan pula bahwa heuristik yang digunakan optimal, maka dapat dipastikan hasil dari algoritma A* juga optimal.

2.6. Analisis efisiensi A* dibanding UCS

Secara teoritis, karena heuristik yang digunakan oleh A* *admissible*, maka UCS serta A* keduanya dijamin menghasilkan solusi optimal. Maka yang dapat dibandingkan adalah seberapa efisien pengunjungan simpul yang dilakukan oleh kedua algoritma. Perlu diingat bahwa A* merupakan *informed search* sedangkan UCS merupakan *uninformed search* maka pastilah A* lebih efisien secara pengunjungan simpul. Ia tidak hanya secara buta mengunjungi semua simpul pada level yang sama, tetapi dia juga mempertimbangkan seberapa mungkin simpul ini menuju ke tujuan. Hal ini membuat simpul-simpul yang dikunjungi oleh A* relatif

lebih “menjanjikan” dibandingkan hanya dengan secara buta memilih simpul seperti UCS. Maka dari itu, dapat disimpulkan bahwa A* akan lebih efisien dari sisi pengunjungan node.

2.7. Pembuatan map untuk mempermudah pencarian anak

Karena pada konteks ini dianggap kata dalam Bahasa Inggris tidak akan pernah berubah, maka untuk melakukan optimasi terhadap pencarian anak dari setiap kata (kata lain yang bisa dicapai hanya dengan sekali pergantian huruf) akan disimpan dalam suatu txt file yang akan di *load* ketika program pertama kali dijalankan. Pembuatan map ini dilakukan dengan langkah-langkah sebagai berikut.

1. Baca txt file yang menjadi *dictionary* acuan
2. Untuk setiap kata dalam txt file tersebut, masukkan kedalam Set yang nantinya akan berisi semua kata valid dalam Bahasa Inggris.
3. Lakukan iterasi terhadap seluruh kata di set tersebut.
4. Aplikasikan fungsi untuk mendapatkan semua anak yang mungkin didapat dari kata tersebut (akan dijelaskan kemudian)
5. Ketika hasil didapatkan (berupa list of words), masukkan value tersebut kedalam map dengan kuncinya adalah kata yang sedang diperiksa.
6. Ulangi hingga seluruh kata telah diiterasi
7. Simpan map hasil kedalam sebuah txt file.

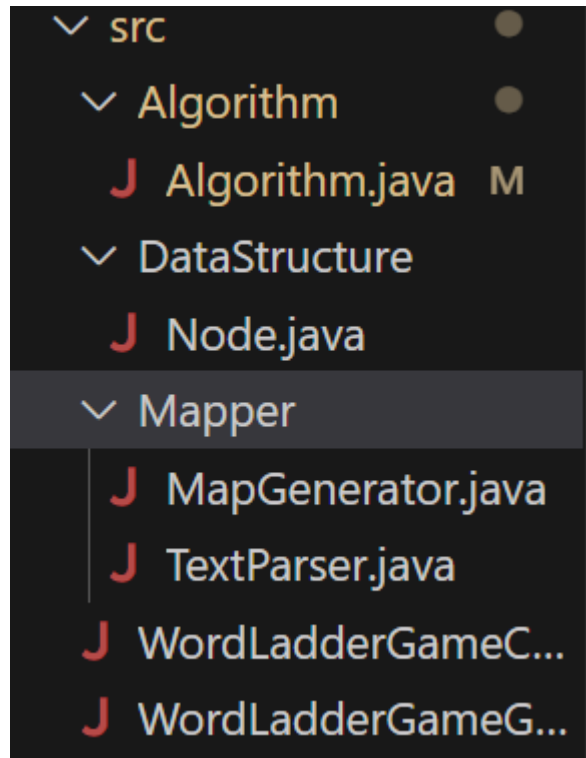
Pengjelasan detail fungsi mendapatkan semua anak dari suatu kata adalah sebagai berikut:

1. Siapkan sebuah ArrayList yang akan menyimpan hasil
2. Lakukan iterasi sebanyak panjang kata. Hal ini untuk melakukan mengubah huruf ke-i pada kata
3. Pada setiap iterasi, lakukan iterasi lagi dari huruf a sampai z untuk mendapatkan “variansi” kata yang sudah diganti satu huruf. Pastikan pula untuk memastikan huruf yang sedang diperiksa bukan huruf dari kata asal.
4. Apabila ternyata kata baru yang terbentuk setelah perubahan merupakan kata yang valid dalam Bahasa Inggris, maka masukkan kedalam list.
5. Teruskan iterasi dan ketika iterasi selesai dan list sudah didapatkan, lakukan sorting berdasarkan huruf agar mirip seperti dictionary asli.

BAB III

Source Code Program dan Penjelasan Class

3.1. Penjelasan Abstraksi Class serta Stuktur Kode



Gambar 1. Struktur Class

Terdapat 6 Kelas utama yang dibuat pada program ini, yaitu Algorithm, Node, MapGenerator, TextParser, WordLadderGameCLI, serta WordLadderGameGUI. Untuk program berbasis CLI, akan dijalankan melalui class WordLadderGameCLI, sedangkan berbasis GUI akan dijalankan melalui WordLadderGameGUI.

Secara umum, flow dari program adalah ketika WordLadderGameCLI/ GUI dipanggil, maka program akan menggunakan static function dari Class TextParser serta MapGenerator untuk melakukan load terhadap *dictionary* serta map yang akan digunakan. Setelah itu, program akan menerima Input dari pengguna dan melakukan berbagai validasi terhadap hal-hal yang dimasukkan. Setelah semua input dipastikan valid, maka Class Algorithm akan dipakai untuk mengevaluasi

hasil input tersebut serta memberikan kembalian berupa hal-hal yang ingin ditampilkan ke pengguna.

Karena Source Code Program terlalu panjang untuk dimasukkan, maka hanya akan dijelaskan kegunaan dari masing-masing kelas serta method yang ada didalamnya. Untuk Source Code lengkap dapat dilihat di bagian lampiran.

3.2. Penjelasan Class Algorithm

Class ini digunakan sebagai “otak” dari seluruh aplikasi. Class ini akan melakukan evaluasi terhadap kata awal dan kata akhir untuk mencari simpul yang bisa saling terhubung dengan menggunakan metode yang dipilih pengguna.

Class Algorithm	
ATRIBUT	
Nama Atribut	Fungsi
String methodType	Algoritma yang digunakan
String startWord	Kata awal
String endWord	Kata tujuan
Map<String,String> visitedMap	Menyimpan node-node yang sudah di ekspan beserta siapa <i>parentnya</i>
METHOD	
Nama Method	Fungsi
public Algorithm(String methodType, String startWord, String endWord)	Default constructor
public ArrayList<String> getPaths()	Untuk mencari path apabila solusi ditemukan
public static ArrayList<String> reverseList(ArrayList<String> list)	Untuk membalik list yang didapatkan (karena hasil akan memiliki tujuan di index pertama ketika iterasi balik)
public int evaluateHeuristic(String thisValue, String goalValue)	Untuk mendapatkan nilai heuristic dari suatu kata terhadap suatu goal
public int evaluateAStar(String currentWord, int level, String goalWord)	Untuk mendapatkan nilai A* dari suatu simpul.

<code>public int evaluatePrice(int level, String currentWord)</code>	Untuk mendapatkan hasil evaluasi dari suatu simpul berdasarkan metode yang digunakan
<code>public Pair<ArrayList<String>, Integer> Evaluate(Map<String, List<String>> maps)</code>	Metode utama untuk melakukan pencarian serta mengembalikan hasilnya ke pengguna.

3.3. Penjelasan Class Node

Class ini digunakan sebagai stuktur data utama dalam program. Node merupakan simpul yang akan dievaluasi nilai-nilainya.

Class Node	
ATRIBUT	
Nama Atribut	Fungsi
Node Parent	Node orangtua
String value	Kata yang disimpan di simpul ini
Int cost	Hasil evaluasi (tergantung dari algoritma)
Int depth	Kedalaman node
METHOD	
Nama Method	Fungsi
<code>public Node(Node parent, String value, int cost, int depth)</code>	Default constructor
<code>public Node getParent()</code>	Getter parent
<code>public String getValue()</code>	Getter value
<code>public int getCost()</code>	Getter cost
<code>public int getDepth()</code>	Getter depth
<code>public static Comparator<Node> compareByCost()</code>	Fungsi komparator antar node

3.4. Penjelasan Class MapGenerator

Class ini digunakan sebagai generator dari map untuk pasangan kunci yaitu kata dan value yang berupa semua anaknya. Program menerima suatu input dictionary txt file dan mengubahnya menjadi sebuah txt file yang lebih muda untuk dibaca sebagai key value map.

Class MapGenerator	
ATRIBUT	
METHOD	
Nama Method	Fungsi
public static List<String> generateOneLetterVariants(String word, Set<String> validWords)	Mencoba semua kombinasi satu huruf yang mungkin dari kata yang diberikan.
public static void main(String[] args)	Melakukan perubahan dictionary awal di dictionary/words.txt yang hanya berisi kata” yang valid menjadi mapping pair of key dan value di dictionary/map.txt

3.5. Penjelasan Class TextParser

Class ini digunakan sebagai parser sekaligus loader ketika program pertama kali dijalankan. Kelas ini memarsing txt file yang telah dibuat sebelumnya untuk menjadi datastruktur yang valid dalam bahasa Java.

Class MapGenerator	
ATRIBUT	
METHOD	
Nama Method	Fungsi
public static Map<String, List<String>> loadMap(String filename)	Melakukan load terhadap map yang telah dijelaskan sebelumnya
public static Set<String>	Melakukan load terhadap validWords

loadSet(String filename)	yang telah dijelaskan sebelumnya
--------------------------	----------------------------------

3.6. Penjelasan Class WordLadderGameCLI

Class ini digunakan sebagai *entrypoint* program CLI. Kelas ini akan menerima serta memvalidasi input pengguna dan melakukan pemanggilan metode dari class-class yang telah disebutkan sebelumnya untuk melakukan kalkulasi dan menampilkan hasilnya ke pengguna.

Class MapGenerator	
ATRIBUT	
METHOD	
Nama Method	Fungsi
public static void main(String[] args)	Melakukan load terhadap map dan dictionary yang telah dijelaskan sebelumnya, menerima input pengguna, lalu melakukan kalkulasi dengan bantuan class Algorithm. Setelah itu menampilkan hasilnya ke layar.

Untuk penjelasan GUI akan dijelaskan di Bab implementasi bonus.

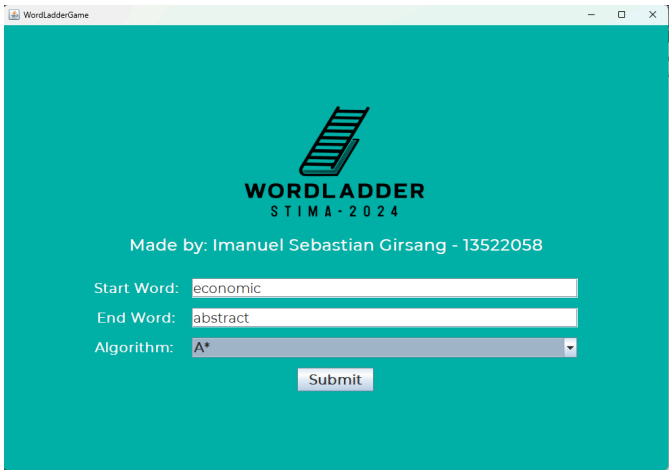
BAB IV

Analisis Perbandingan Algoritma

4.1. Uji Coba Dengan Tangkapan Layar

4.1.1. Test Case 1 (Tidak Ada Solusi)

ECONOMIC to ABSTRACT



WordLadderGame

WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

ECONOMIC to ABSTRACT

Time Taken: 0 ms
Memory used: 1652128 bytes
Visited Nodes: 1
Path Length: 0
Algorithm: A*

Summary Path: []



No solution found :(

WordLadderGame



Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

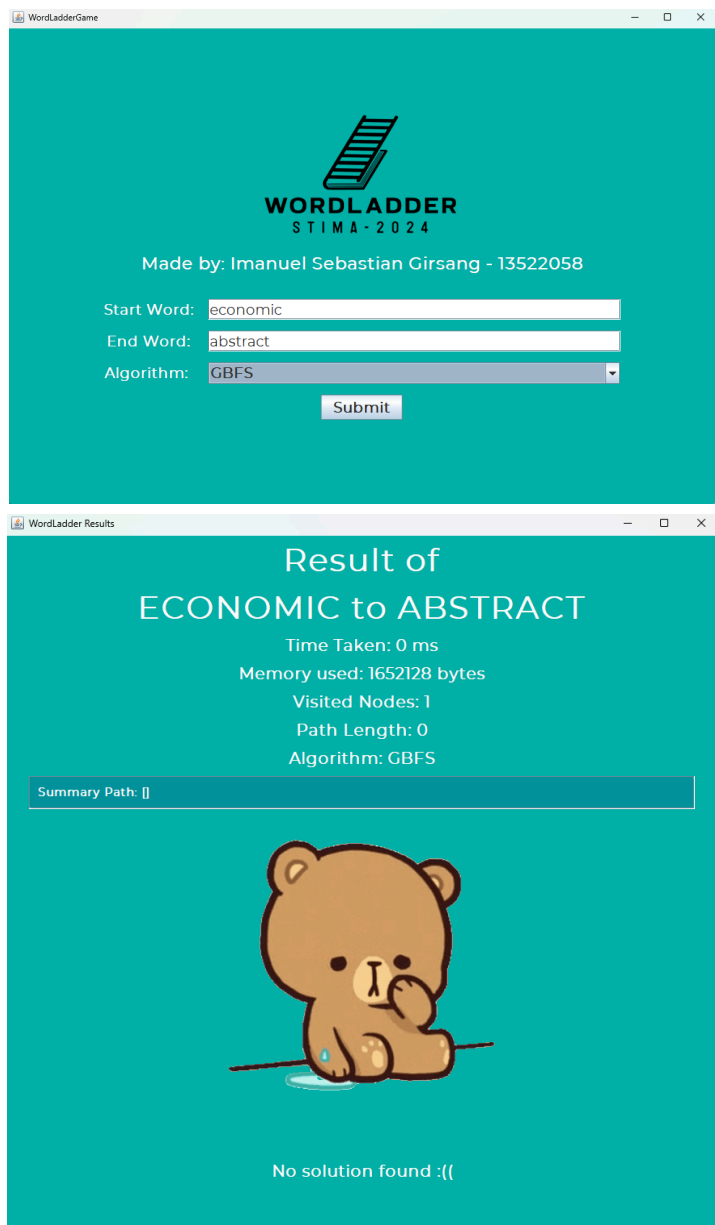
Result of ECONOMIC to ABSTRACT

Time Taken: 4 ms
Memory used: 7713912 bytes
Visited Nodes: 1
Path Length: 0
Algorithm: UCS

Summary Path: []



No solution found :((



MATRIKS	A*	UCS	GBFS
Waktu (ms)	0	4	0
Memory (byte)	1652128	7713912	1652128
Panjang solusi	0	0	0
Node dikunjungi	1	1	1

4.1.2. Test Case 2 ATOM to UNAUI

ATOM to UNAUI

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

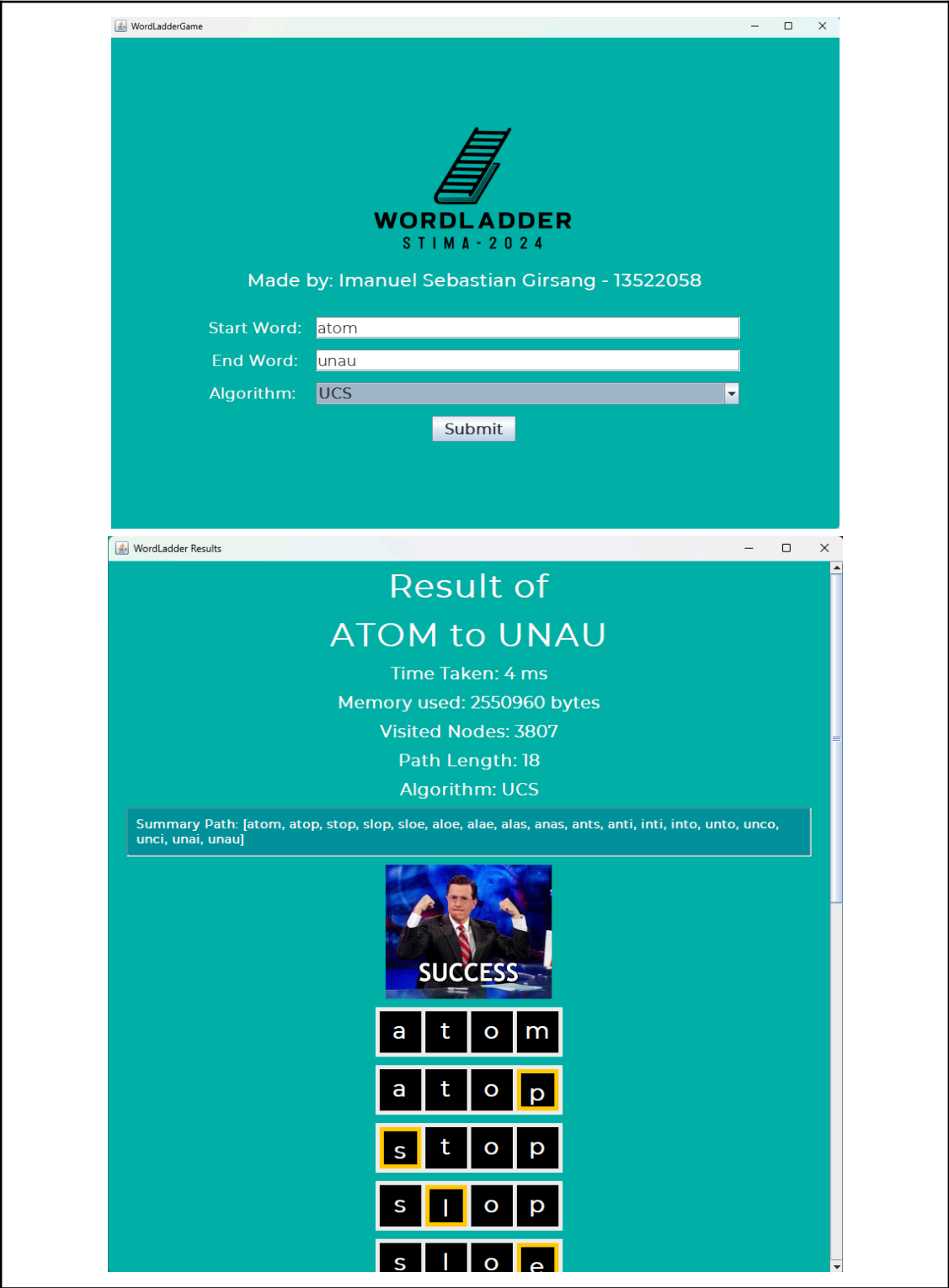
Result of ATOM to UNAUI

Time Taken: 4 ms
Memory used: 1685336 bytes
Visited Nodes: 3804
Path Length: 18
Algorithm: A*

Summary Path: [atom, atop, stop, slop, sloe, aloe, alae, alas, anas, ants, anti, inti, into, unto, unco, unci, unai, unau]



a	t	o	m
a	t	o	p
s	t	o	p
s	l	o	p
s	l	o	e



WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of ATOM to UNAU

Time Taken: 1 ms
Memory used: 2550928 bytes
Visited Nodes: 220
Path Length: 21
Algorithm: GBFS

Summary Path: [atom, atop, stop, swop, swap, snap, snaw, snow, know, knew, anew, anes, ants, anti, inti, into, unto, unco, unci, unai, unau]



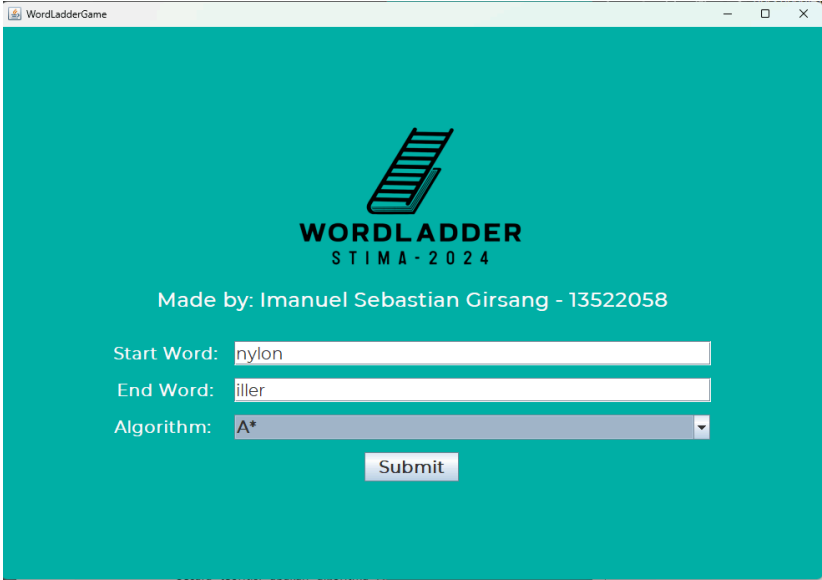
SUCCESS

a t o m
a t o p
s t o p
s w o p
s w a p

MATRIKS	A*	UCS	GBFS
Waktu (ms)	4	4	1
Memory (byte)	1685336	2550960	2550928
Panjang solusi	18	18	21
Node dikunjungi	3804	3807	220

4.1.3. Test Case 3 NYLON to ILLER

NYLON to ILLER



WordLadderGame

WORDLADDER
STIMA - 2024

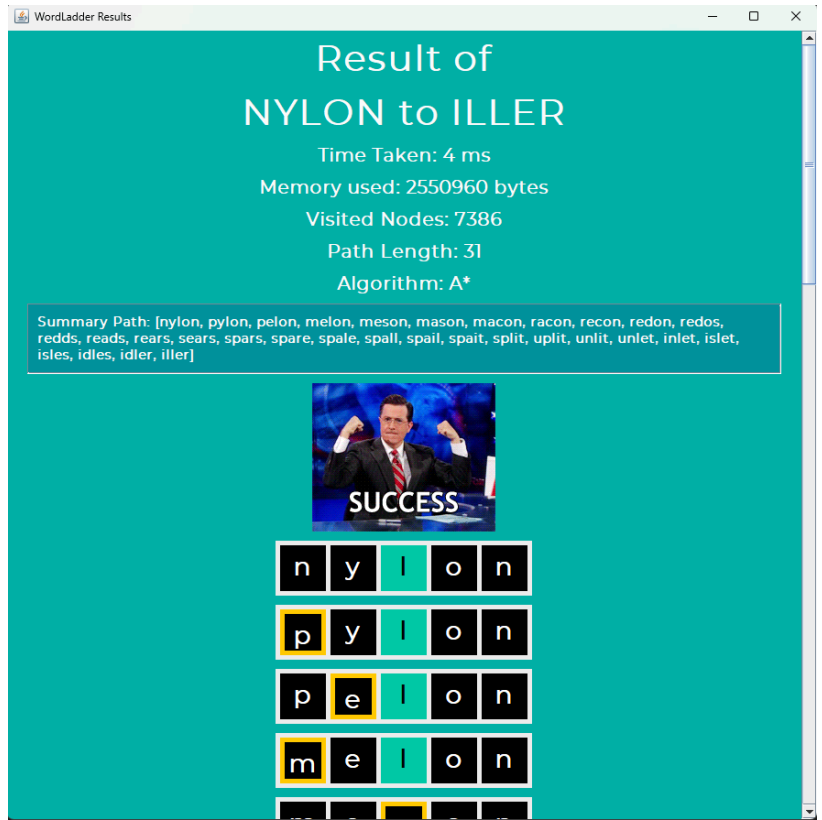
Made by: Imanuel Sebastian Girsang - 13522058

Start Word: nylon

End Word: iller

Algorithm: A*

Submit



WordLadder Results

Result of NYLON to ILLER

Time Taken: 4 ms
Memory used: 2550960 bytes
Visited Nodes: 7386
Path Length: 31
Algorithm: A*

Summary Path: [nylon, pylon, pelon, melon, meson, mason, macon, racon, recon, redon, redos, reds, reads, rears, sears, spars, spare, spale, spall, spail, spait, split, uplit, unlit, unlet, inlet, islet, isles, idles, idler, iller]

SUCCESS

n y l o n
p y l o n
p e l o n
m e l o n

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of NYLON to ILLER

Time Taken: 4 ms
Memory used: 2550960 bytes
Visited Nodes: 7396
Path Length: 31
Algorithm: UCS

Summary Path: [nylon, pylon, pelon, melon, meson, mason, macon, racon, recon, redon, redos, redes, rides, sides, sires, siree, spree, sprue, sprug, sprig, sprit, split, uplit, unlit, unlet, inlet, islet, isled, idled, idler, iller]



SUCCESS

n	y	l	o	n
p	y	l	o	n
p	e	l	o	n
m	e	l	o	n

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of NYLON to ILLER

Time Taken: 4 ms

Memory used: 2550960 bytes

Visited Nodes: 5427

Path Length: 74

Algorithm: GBFS

Summary Path: [nylon, pylon, pelon, melon, meson, mason, macon, racon, recon, rewon, rewan, sewan, sewar, sewer, fewer, feyer, flyer, fleer, flees, floes, sloes, slots, blots, blocs, block, clock, cluck, pluck, plunk, slunk, slung, slang, stang, stane, stone, stove, shove, shote, smote, smoke, spoke, spode, spade, spale, spall, stall, stull, scull, sculp, scalp, scaup, scaur, scour, stour, stout, strut, strum, scrum, scrub, shrub, shrug, sprug, sprig, sprit, split, uplit, unlit, unlet, inlet, islet, isled, idled, idler, iller]



n y l o n

p y l o n

p e l o n

MATRIKS	A*	UCS	GBFS
Waktu (ms)	4	4	4
Memory (byte)	2550960	2550960	2550960
Panjang solusi	31	31	74
Node dikunjungi	7386	7396	5427

4.1.4. Test Case 4 ATLASES to CABARET

ATLASES to CABARET

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of
ATLASES to CABARET

Time Taken: 5 ms

Memory used: 3735184 bytes

Visited Nodes: 7590

Path Length: 53

Algorithm: A*

Summary Path: [atlases, anlases, anlases, unlases, unlades, unladed, unfaded, unfaked, uncaked, uncased, uncases, uneases, ureases, creases, cresses, crosses, crosser, crasser, crasher, brasher, brasier, brakier, beakier, peckier, pickier, dickier, dickies, hickies, hackies, hackles, heckles, deckles, deciles, defiles, defiled, deviled, develed, reveled, raveled, ravened, havened, havered, wavered, watered, catered, capered, tapered, tabered, tabored, taboret, tabaret, cabaret]



SUCCESS

a	t	l	a	s	e	s
a	n	l	a	s	e	s
a	n	l	a	c	e	s
u	n	l	a	c	e	s

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of ATLASES to CABARET

Time Taken: 5 ms

Memory used: 3735184 bytes

Visited Nodes: 7648

Path Length: 53

Algorithm: UCS

Summary Path: [atlases, anlases, anlaces, unlaces, unlades, unladed, unfaded, unfaked, uncaked, uncased, uncases, uneases, ureases, creases, cresses, crosses, crosser, crasser, crasher, brasher, brasier, brakier, beakier, peakier, peckier, pickier, dickier, dickies, hickies, hackies, hackles, heckles, deckles, deciles, defiles, defiled, deviled, reviled, reveled, raveled, ravened, havened, havered, wavered, watered, catered, capered, tapered, tabered, tabored, taboret, tabaret, cabaret]



SUCCESS

a	t	l	a	s	e	s
a	n	l	a	s	e	s
a	n	l	a	c	e	s
u	n	l	a	c	e	s

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of ATLASES to CABARET

Time Taken: 2 ms
Memory used: 3735216 bytes
Visited Nodes: 3553
Path Length: 80
Algorithm: GBFS

Summary Path: [atlases, anlases, anlases, unlases, unlades, unladed, unfaded, unfaked, unbaked, unbased, uncased, uncases, uneases, ureases, creases, cresses, chesses, chasses, classes, clashes, clasher, slasher, slather, slatier, platier, peatier, pestier, pastier, pasties, panties, pandies, candies, caddies, baddies, baldies, ballies, rallies, rallied, dallied, dallier, tallier, talkier, tackier, tackler, cackler, cackled, hackled, heckled, keckled, keckles, deckles, deciles, defiles, refiles, refires, retires, retimes, regimes, regives, relives, relived, relined, repined, rapines, rapines, ravines, ravined, ravened, havened, havered, wavered, watered, catered, capered, tapered, tabered, tabored, taboret, tabaret, cabaret]



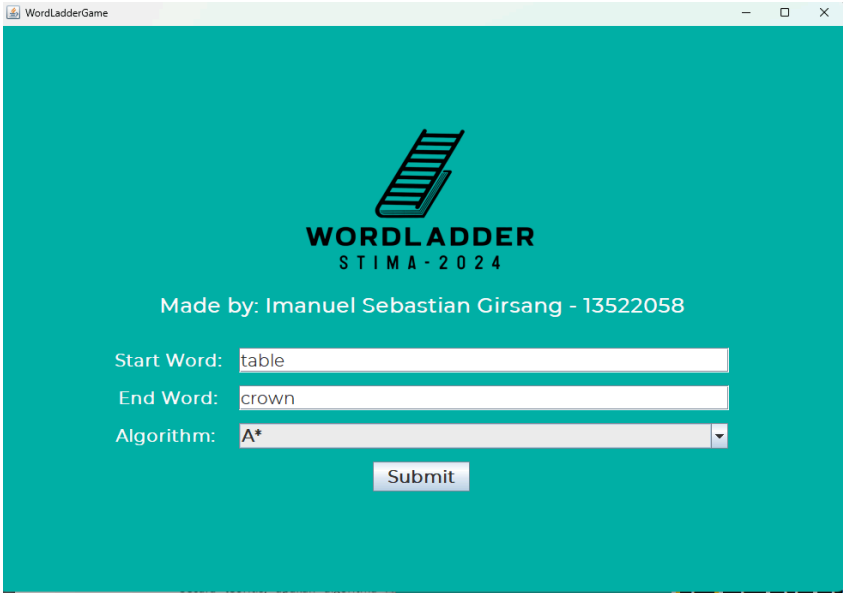
SUCCESS

a	t	l	a	s	e	s
a	n	l	a	s	e	s
a	n	l	a	c	e	s

MATRIKS	A*	UCS	GBFS
Waktu (ms)	5	5	2
Memory (byte)	3735280	3735184	3735216
Panjang solusi	53	53	80
Node dikunjungi	7590	7648	3553

4.1.5. Test Case 5 TABLE to CROWN

TABLE to CROWN



WordLadderGame

WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

Result of TABLE to CROWN

Time Taken: 1 ms
Memory used: 3735280 bytes
Visited Nodes: 121
Path Length: 10
Algorithm: A*

Summary Path: [table, cable, carle, carls, carps, corps, coops, crops, crows, crown]



SUCCESS

t	a	b	l	e
c	a	b	l	e
c	a	r	l	e
c	a	r	l	s
c	a	r	p	s

WordLadderGame



WORDLADDER

STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of TABLE to CROWN

Time Taken: 14 ms
Memory used: 3735280 bytes
Visited Nodes: 5128
Path Length: 10
Algorithm: UCS

Summary Path: [table, cable, carle, carls, carps, corps, coops, crops, crows, crown]



t	a	b	l	e
c	a	b	l	e
c	a	r	l	e
c	a	r	l	s
c	a	r	p	s

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of TABLE to CROWN

Time Taken: 1 ms
Memory used: 2403592 bytes
Visited Nodes: 45
Path Length: 15
Algorithm: GBFS

Summary Path: [table, cable, carle, carte, caste, casts, cants, cents, celts, colts, coots, coops, crops, crows, crown]



SUCCESS

t	a	b	l	e
c	a	b	l	e
c	a	r	l	e
c	a	r	t	e
c	a	s	t	e

MATRIKS	A*	UCS	GBFS
Waktu (ms)	1	14	1
Memory (byte)	3735280	3735280	2403592
Panjang solusi	10	10	15
Node dikunjungi	121	5128	45

4.1.6. Test Case 6 GIMLETS to TREEING

GIMLETS to TREEING



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

Submit

WordLadder Results

Result of GIMLETS to TREEING

Time Taken: 7 ms
Memory used: 3740736 bytes
Visited Nodes: 4047
Path Length: 25
Algorithm: A*

Summary Path: [gimlets, giglets, wiglets, willets, willers, billers, ballers, bailers, bailees, bailies, dailies, doilies, dollies, collies, collins, codlins, codling, codding, godding, goading, grading, graying, greying, greening, treeing]



SUCCESS

g	i	m	l	e	t	s
g	i	g	l	e	t	s
w	i	g	l	e	t	s
w	i	l	l	e	t	s

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of GIMLETS to TREEING

Time Taken: 15 ms
Memory used: 10399144 bytes
Visited Nodes: 6285
Path Length: 25
Algorithm: UCS

Summary Path: [gimlets, giglets, wiglets, willets, willers, billers, ballers, bailers, bailees, bailies, ballies, bullies, cullies, collies, collins, codlins, codling, godling, godding, goading, grading, graying, greying, greeing, treeing]



g	i	m	l	e	t	s
g	i	g	l	e	t	s
w	i	g	l	e	t	s
w	i	l	l	e	t	s
w	i	l	l	e	t	s

WordLadderGame



WORDLADDER
STIMA - 2024

Made by: Imanuel Sebastian Girsang - 13522058

Start Word:

End Word:

Algorithm:

WordLadder Results

Result of GIMLETS to TREEING

Time Taken: 1 ms

Memory used: 2515656 bytes

Visited Nodes: 663

Path Length: 77

Algorithm: GBFS

Summary Path: [gimlets, giglets, wiglets, willets, willers, tillers, tollers, toolers, tooters, mooters, molters, molders, polders, powders, powters, porters, porkers, workers, wormers, warmers, warpers, harpers, hampers, tampers, tappers, toppers, hoppers, hoopers, hookers, hookeys, honkeys, donkeys, dinkeys, pinkeys, punkeys, punkers, punters, putters, puttees, putties, tutties, tatties, tattier, nattier, nittier, wittier, withier, withies, within, withing, tithing, titling, tilling, tilting, tinting, tinning, tanning, tanking, tasking, tasting, hasting, halting, halving, helving, helping, heaping, leaping, leaning, loaning, loading, goading, grading, trading, tracing, trucing, trueing, treeing]



g i m l e t s

g i g l e t s

w i g l e t s

MATRIKS	A*	UCS	GBFS
Waktu (ms)	7	15	1
Memory (byte)	3740736	10399144	2515656
Panjang solusi	25	25	77
Node dikunjungi	4047	6285	663

4.2. Analisis Optimalitas hasil

Saat dilakukan pengujian, algoritma A* dan Uniform Cost Search (UCS) konsisten memberikan solusi terpendek untuk setiap masalah yang dihadapi, sesuai dengan teori yang mendasari mereka. Keduanya mampu menemukan jalur optimal karena menggunakan pendekatan yang mempertimbangkan seluruh informasi yang relevan, baik jarak maupun biaya, untuk mencapai hasil yang paling pendek.

Sementara itu, Greedy Best-First Search (GBFS) menunjukkan hasil yang tidak selalu optimal dalam hal panjang atau jarak solusi. Ini karena GBFS hanya fokus pada langkah selanjutnya yang dianggap paling menjanjikan, tanpa mempertimbangkan bagaimana keputusan tersebut akan memengaruhi solusi secara keseluruhan. Sifat algoritma yang hanya mencari solusi terbaik di tingkat lokal inilah yang membuat GBFS terkadang menghasilkan solusi yang kurang optimal dibandingkan A* atau UCS.

4.3. Analisis Waktu Eksekusi

Dalam pengujian, Greedy Best-First Search (GBFS) terbukti menjadi algoritma dengan waktu eksekusi paling cepat. Algoritma A* dan Uniform Cost Search (UCS) menunjukkan waktu eksekusi yang bervariasi, tergantung pada kompleksitas kasus yang diujikan. Secara umum, GBFS unggul dalam kecepatan karena sifatnya yang langsung memilih jalur yang tampak paling menjanjikan berdasarkan heuristik, sehingga membutuhkan lebih sedikit waktu untuk menemukan solusi.

Namun, pada kasus yang lebih kompleks atau "tricky", seperti test case nomor 5 dan 6, UCS cenderung mengunjungi lebih banyak simpul dibandingkan A* dan GBFS. Hal ini menyebabkan waktu eksekusi UCS menjadi lebih lambat dalam skenario ini. Sebaliknya, pada kasus yang lebih sederhana, perbedaan waktu eksekusi antara A* dan UCS tidak terlalu signifikan. Hal ini menunjukkan bahwa kecepatan eksekusi bisa sangat bergantung pada jenis kasus yang dihadapi, dengan GBFS menjadi pilihan paling cepat dalam banyak skenario.

4.4. Analisis memori yang Dibutuhkan

Penggunaan memori oleh algoritma pencarian dapat bervariasi tergantung pada kondisi perangkat saat itu. Hal ini disebabkan oleh keterbatasan dalam mengukur penggunaan memori secara spesifik untuk setiap proses. Namun, berdasarkan pengujian, Greedy Best-First Search (GBFS) menggunakan memori paling sedikit, diikuti oleh A*, dan kemudian Uniform Cost Search (UCS). Ini bisa dimengerti karena GBFS cenderung mengunjungi lebih sedikit simpul dibandingkan UCS dan A*, sehingga jumlah data yang disimpan lebih rendah dan penggunaan memori untuk proses-proses yang ada menjadi lebih hemat.

Sementara itu, baik A* maupun UCS, penggunaan memori bisa bervariasi tergantung pada kasus yang diuji. UCS, yang mengunjungi lebih banyak simpul karena mengevaluasi setiap kemungkinan, bisa membutuhkan lebih banyak memori jika kasusnya kompleks. Sebaliknya, A* biasanya menggunakan memori lebih sedikit karena memanfaatkan heuristik untuk memilih jalur yang lebih efisien, sehingga mengurangi jumlah simpul yang harus diunjungi dan data yang harus disimpan. Secara keseluruhan, GBFS tetap menjadi algoritma dengan penggunaan memori paling hemat, sementara A* dan UCS memiliki penggunaan memori yang dapat berbeda tergantung pada kasus yang dihadapi.

BAB V

Implementasi Bonus

5.1. Struktur GUI

Untuk mengimplementasikan GUI, digunakan kaskas Swing yang disediakan langsung oleh Java. Kemudian, Dibuat sebuah Kelas baru yaitu WorldLadderGameGUI yang digunakan untuk menampung tampilan-tampilan yang akan dibuat. Untuk kode lengkap dari GUI bisa dilihat di github ataupun di bagian lampiran karena akan menjadi terlalu panjang untuk ditampilkan di bagian ini. Pada dasarnya, GUI diimplementasikan dengan membuat 2 frame yaitu frame untuk menerima input pengguna, serta frame untuk menampilkan hasil dari algoritma. Didalam setiap framenya, akan ada beberapa panel yang

5.2. Penjelasan Frame Masukan

Frame input ditampilkan saat program dijalankan pertama kali dengan memanggil metode `createAndShowGUI`. Frame ini memiliki satu panel utama bernama `mainPanel`, yang berisi berbagai komponen seperti logo, nama penulis, dua label dengan `textfield`, `combobox` untuk memilih algoritma, dan tombol "Submit". Ketika pengguna mengklik tombol "Submit", nilai dari setiap field akan diambil untuk diproses.

Validasi dilakukan untuk memastikan bahwa input, baik "start word" maupun "end word", tidak kosong, dan memastikan bahwa kata-kata tersebut valid sesuai dengan kamus yang digunakan oleh aplikasi. Selain itu, validasi juga mencakup pengecekan apakah ada kata yang terkait di dalam kamus dengan input pengguna.

Ada juga `key listener` yang memungkinkan pengguna menggunakan tombol panah atas dan bawah untuk memindahkan kursor antara input "start word" dan "end word". Tekanan tombol Enter akan memiliki efek yang sama dengan mengklik tombol "Submit".

Ketika tombol "Submit" ditekan, algoritma yang dipilih akan digunakan untuk menemukan jalur yang sesuai dengan input pengguna. Setelah hasil diperoleh, frame baru akan dibuka untuk menampilkan jalur yang ditemukan dan hasil lainnya terkait input pengguna.

5.3. Penjelasan Frame Hasil

Frame hasil adalah jendela yang dibuka saat tombol "Submit" ditekan pada frame utama. Ini membantu memisahkan bagian antara input pengguna dan tampilan hasil dari pemrosesan data. Dalam frame ini, berbagai informasi ditampilkan kepada pengguna untuk memberikan konteks tentang hasil yang diperoleh. Informasi yang ditampilkan meliputi:

1. Algoritma yang digunakan untuk mencari solusi (misalnya UCS, GBFS, atau A*).
2. Kata awal ("start word") dan kata akhir ("end word") yang dimasukkan oleh pengguna.
3. Waktu eksekusi yang diperlukan untuk menjalankan algoritma.
4. Jumlah node yang dikunjungi selama pencarian jalur.
5. Penggunaan memori oleh proses pencarian.

Selain itu, frame hasil menampilkan jalur yang mengilustrasikan perubahan kata di setiap langkahnya, memberikan pengguna pemahaman visual tentang bagaimana pencarian solusi terjadi. Untuk melakukan ini, dibuat panel baru di mana setiap langkah dalam jalur pencarian disusun dalam kotak-kotak yang mewakili perubahan kata di setiap tahap. Setiap langkah menunjukkan kata yang terbentuk dari perubahan huruf dari kata sebelumnya, memungkinkan pengguna melihat transisi antar kata secara bertahap.

Untuk membantu pengguna memahami jalur dengan lebih baik, huruf yang sudah sesuai dengan hasil akhir diberi warna berbeda. Hal ini memberikan indikasi visual yang jelas tentang bagaimana setiap langkah dalam jalur mendekati solusi akhir.

Pendekatan ini tidak hanya memberikan informasi kepada pengguna tentang proses pencarian, tetapi juga menyediakan tampilan visual yang memudahkan pemahaman jalur yang dilalui dari kata awal ke kata akhir. Dengan memisahkan antara input dan hasil dalam dua frame yang berbeda, aplikasi memberikan antarmuka pengguna yang lebih terstruktur dan jelas, memudahkan navigasi dan interaksi.

BAB VI

KESIMPULAN

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, dapat disimpulkan bahwa algoritma A* adalah pilihan terbaik untuk menyelesaikan permasalahan ini karena memberikan performa yang optimal dan paling efisien. Keunggulan A* terletak pada penggunaan fungsi heuristik yang bersifat admissible, yang memastikan bahwa algoritma ini dapat menemukan solusi terpendek dengan konsistensi dan akurasi yang tinggi.

Dengan kombinasi antara pencarian berbasis biaya dan heuristik, A* mampu menyeimbangkan eksplorasi dan eksploitasi sehingga dapat mengurangi jumlah simpul yang harus dievaluasi.

Namun, jika data yang digunakan meningkat secara signifikan atau jika fokus utama bukan pada menemukan solusi terpendek melainkan kecepatan eksekusi, Greedy Best-First Search (GBFS) bisa menjadi alternatif yang layak. GBFS cenderung lebih cepat karena hanya mempertimbangkan heuristik tanpa memperhatikan biaya total, sehingga dapat menemukan solusi lebih cepat daripada A* dalam beberapa kasus. Meskipun hasilnya mungkin tidak selalu optimal, pendekatan ini efektif untuk aplikasi di mana waktu perhitungan lebih penting daripada solusi terpendek.

Dengan demikian, A* tetap menjadi pilihan yang paling cocok untuk kasus-kasus yang memerlukan solusi optimal dan efisiensi yang tinggi. Sementara GBFS dapat menjadi alternatif yang bermanfaat ketika kecepatan menjadi prioritas dan solusi optimal tidak terlalu diperlukan. Keputusan untuk menggunakan algoritma mana bergantung pada kebutuhan spesifik dan karakteristik permasalahan yang sedang dihadapi.

BAB VII

Lampiran

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 dapat melakukan visualisasi proses pembuatan kurva.	✓	

Link Repository GitHub : https://github.com/ImanuelSG/Tucil3_13522058.git

7.1. Source Code Class Algorithm

```
1 package Algorithm;
2
3 import DataStructure.Node;
4 import java.util.PriorityQueue;
5 import java.util.ArrayList;
6 import java.util.HashMap;
7 import java.util.List;
8 import java.util.Map;
9 import javafx.util.Pair;
10
11 public class Algorithm {
12
13     private String methodType; // Untuk nyimpen metode yang dipake
14     private String startWord; // Untuk nyimpen kata awal
15     private String endWord; // Untuk nyimpen kata akhir
16     private Map<String, String> visitedMap; // Biar tidak terjadi cycle, disimpan semua visitedNode dan siapa parentnya,
17                                           // Parent dari kata pertama adalah ""
18
19     // Constructor, yang membedakan UCS, GBF, dan A* adalah cara menghitung cost
20     public Algorithm(String methodType, String startWord, String endWord) {
21         this.methodType = methodType;
22         this.startWord = startWord;
23         this.endWord = endWord;
24         this.visitedMap = new HashMap<>();
25     }
26
27     // Untuk mendapatkan path dari startWord ke endWord (dengan visitedMap)
28     // basically reverse engineering :)
29     public ArrayList<String> getPaths() {
30         ArrayList<String> path = new ArrayList<>();
31         String current = endWord;
32
33         // Dicari dari kata terakhir ke kata pertama (ditandai dengan parent = "")
34         while (current != "") {
35             path.add(current);
36             current = visitedMap.get(current);
37         }
38         return Algorithm.reverseList(path);
39     }
40
41     // Method untuk membalikkan list
42     public static ArrayList<String> reverseList(ArrayList<String> list) {
43         ArrayList<String> reversedList = new ArrayList<>();
44         for (int i = list.size() - 1; i >= 0; i--) {
45             reversedList.add(list.get(i));
46         }
47         return reversedList;
48     }
49
50     // Method to evaluate heuristic using Hamming distance (counting character
51     // differences), semakin kecil semakin baik.
52     public int evaluateHeuristic(String thisValue, String goalValue) {
53         int counter = 0;
54         // Ensure the lengths are the same
55         if (thisValue.length() != goalValue.length()) {
56             throw new IllegalArgumentException("The words must be of the same length.");
57         }
58         // Count character differences
59         for (int i = 0; i < thisValue.length(); i++) {
60             if (thisValue.charAt(i) != goalValue.charAt(i)) {
61                 counter++; // Increment counter when characters differ
62             }
63         }
64         return counter;
65     }
66
67     // Method to evaluate the A* cost: g(n) + h(n), level adalah depth dari node
68     // yang dievaluasi.
69     public int evaluateAStar(String currentWord, int level, String goalWord) {
70
71         return level + evaluateHeuristic(currentWord, goalWord); // A* evaluation
72     }
73
74     // Method untuk mengevaluasi cost dari node, tergantung dari metode yang dipakai
75     // Kalau UCS akan mengembalik depthnya,
76     // Kalau GBFS akan mengembalik heuristicnya,
77     // Kalau A* akan mengembalik g(n) + h(n)
78     public int evaluatePrice(int level, String currentWord) {
79         if (this.methodType.equals("UCS")) {
80             return level;
81         } else if (this.methodType.equals("GBFS")) {
82             return evaluateHeuristic(currentWord, this.endWord);
83         } else if (this.methodType.equals("A*")) {
84             return evaluateAStar(currentWord, level, this.endWord);
85         } else {
86             throw new IllegalArgumentException("Invalid method type.");
87         }
88     }
89 }
```

```

1 // Method untuk mengevaluasi path dari startWord ke endWord
2 // Parameter maps merupakan mapping antara kata dengan tetangganya (kata valid
3 // yang bisa dicapai dengan mengganti salah satu huruf)
4 public Pair<ArrayList<String>, Integer> Evaluate(Map<String, List<String>> maps) {
5
6     // Priority queue untuk menyimpan node yang akan diekspan, compareByCost
7     // membandingkan cost dari node
8     PriorityQueue<Node> queue = new PriorityQueue<>(Node.compareByCost());
9     Node root = new Node(null, startWord, evaluatePrice(0, startWord), 0);
10
11     // visitedDouble untuk menyimpan node yang sudah pernah diekspan dan harganya.
12     // Ini untuk memprune node yang sudah terlalu mahal untuk tidak dimasukkan ke
13     // queue
14     Map<String, Integer> visitedDouble = new HashMap<>();
15
16     // Ini untuk tahu berapa node visited
17     int counter = 0;
18
19     queue.add(root);
20
21     // Selama queue belum penuh maka lanjut terus
22     while (!queue.isEmpty()) {
23
24         Node current = queue.poll();
25
26         // Kalau sudah pernah di ekspan, maka skip
27         if (visitedMap.get(current.getValue()) != null) {
28             continue;
29         }
30
31         counter++;
32
33         visitedMap.put(current.getValue(),
34             (current.getParent() != null) ? current.getParent().getValue() : "");
35
36         // Jika sudah sampai ke endWord, maka return path dan jumlah node yang
37         // dikunjungi
38         if (current.getValue().equals(endWord)) {
39
40             return new Pair<>(getPaths(), counter);
41         }
42
43         // masukkan ke visitedMap
44
45         // Ambil semua tetangga dari node yang sedang diekspan
46         List<String> neighbors = maps.get(current.getValue());
47         // Jika tidak ada tetangga, maka skip
48         if (neighbors == null) {
49             continue;
50         }
51
52         // Iterasi setiap neighbor
53         for (String neighbor : neighbors) {
54
55             // Hanya consider yang belum pernah di ekspan
56             if (visitedMap.get(neighbor) == null) {
57
58                 // Kalau sudah pernah dimasukkan ke queue namun cost ini lebih mahal, maka
59                 // continue (menghindari double expanding)
60                 if (visitedDouble.get(neighbor) != null
61                     && visitedDouble.get(neighbor) <= evaluatePrice(current.getDepth() + 1, neighbor)) {
62                     continue;
63                 }
64
65                 Node newNode = new Node(current, neighbor, evaluatePrice(current.getDepth() + 1, neighbor),
66                     current.getDepth() + 1);
67                 queue.add(newNode);
68                 // visitedDouble untuk keep track node yang udah pernah masuk ke queue dengan
69                 // cost yang ada
70                 // (ini digunakan untuk memprune potensial node yang lebih mahal)
71                 visitedDouble.put(neighbor, evaluatePrice(current.getDepth() + 1, neighbor));
72             }
73         }
74     }
75
76     // Jika tidak ada path yang ditemukan, maka return empty path dan jumlah node
77     // yang dikunjungi
78     System.out.println("No path found");
79     return new Pair<>(new ArrayList<>(), counter);
80 }
81
82 }
83
84

```

7.2. Source Code Class Node

```
1 package DataStructure;
2
3 import java.util.Comparator;
4
5 public class Node {
6     private Node parent; // Parent node dari simpul sekarang, untuk root parent = null
7     private String value; // Kata yang disimpan di simpul
8     private int cost; // Harga untuk mencapai simpul
9     private int depth; // Kedalaman simpul
10
11     // Constructor initializes parent and value, for root it will be null
12
13     public Node(Node parent, String value, int cost, int depth) {
14         this.parent = parent;
15         this.value = value;
16         this.cost = cost;
17         this.depth = depth;
18     }
19
20     // Getter for parent
21     public Node getParent() {
22         return this.parent;
23     }
24
25     // Getter for value
26     public String getValue() {
27         return this.value;
28     }
29
30     // Getter for cost
31     public int getCost() {
32         return this.cost;
33     }
34
35     // Getter for depth
36     public int getDepth() {
37         return this.depth;
38     }
39
40     // Method to compare two nodes by their cost
41     public static Comparator<Node> compareByCost() {
42         return (node1, node2) -> Integer.compare(node1.getCost(), node2.getCost());
43     }
44 }
45
46
```

7.3. Source Code Class Map Generator

```
1 package Mapper;
2
3 import java.io.*;
4 import java.util.*;
5 import java.util.stream.Collectors;
6
7 public class MapGenerator {
8
9     // Generate all possible one-letter variants of a word that are valid words
10    // (Basically nyoba semua kombinasi satu huruf yang mungkin dari kata yang diberikan)
11    public static List<String> generateOneLetterVariants(String word, Set<String> validWords) {
12        List<String> variants = new ArrayList<>();
13        char[] wordChars = word.toCharArray();
14
15        for (int i = 0; i < wordChars.length; i++) {
16            char originalChar = wordChars[i];
17            for (char c = 'a'; c <= 'z'; c++) {
18                if (c != originalChar) {
19                    wordChars[i] = c;
20                    String newWord = new String(wordChars);
21
22                    // kalau kata yang baru valid
23                    if (validWords.contains(newWord)) {
24                        variants.add(newWord);
25                    }
26                }
27            }
28            wordChars[i] = originalChar;
29        }
30
31        Collections.sort(variants); // Sort variants in lexicographical order
32        return variants;
33    }
34
35    public static void main(String[] args) {
36        // Open the file containing the words
37        File inputFile = new File("dictionary/words.txt");
38        Set<String> validWords = new HashSet<>();
39
40        try (BufferedReader br = new BufferedReader(new FileReader(inputFile))) {
41            String line;
42            while ((line = br.readLine()) != null) {
43                String word = line.trim();
44                if (!word.isEmpty()) {
45                    validWords.add(word);
46                }
47            }
48        } catch (IOException e) {
49            System.err.println("Error reading input file: " + e.getMessage());
50            return;
51        }
52
53        // Create the key-value mapping
54        Map<String, List<String>> wordMap = new HashMap<>();
55        for (String word : validWords) {
56            List<String> variants = generateOneLetterVariants(word, validWords);
57            if (!variants.isEmpty()) {
58                wordMap.put(word, variants);
59            }
60        }
61
62        // Sort the keys of the map
63        List<String> sortedKeys = new ArrayList<>(wordMap.keySet());
64        Collections.sort(sortedKeys);
65
66        // Save the key-value pairs to a text file
67        File outputFile = new File("dictionary/map.txt");
68        try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
69            for (String key : sortedKeys) {
70                List<String> variants = wordMap.get(key);
71                String formattedVariants = variants.stream().collect(Collectors.joining(", "));
72                String line = String.format("%s: %s\n", key, formattedVariants);
73
74                writer.write(line);
75            }
76
77            writer.flush();
78        } catch (IOException e) {
79            System.err.println("Error writing to output file: " + e.getMessage());
80        }
81
82        System.out.println("Key-value mapping saved to 'map.txt'");
83    }
84 }
85
```

7.4. Source Code Class Text Parser

```
1 package Mapper;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.util.*;
7
8 public class TextParser {
9     public static Map<String, List<String>> loadMap(String filename) {
10         Map<String, List<String>> data = new HashMap<>();
11         try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
12             String line;
13             while ((line = br.readLine()) != null) {
14                 // Split by colon to separate key and values
15                 String[] keyValue = line.split(":");
16
17                 if (keyValue.length == 2) {
18                     String key = keyValue[0].trim(); // Get the key
19                     // Get the values as a list, removing whitespace and splitting by comma
20                     List<String> values = Arrays.asList(keyValue[1].trim().split(",\\s*"));
21
22                     data.put(key, values);
23                 }
24             }
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28         return data;
29     }
30
31     public static Set<String> loadSet(String filename) {
32         Set<String> data = new HashSet<>();
33
34         try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
35             String line;
36             while ((line = br.readLine()) != null) {
37                 data.add(line.trim());
38             }
39         } catch (IOException e) {
40             e.printStackTrace();
41         }
42
43         return data;
44     }
45
46     public static boolean isValidWord(String word, Set<String> validWords) {
47         return validWords.contains(word);
48     }
49 }
50
```

7.5. Source Code Class WorldLadderGameCLI

```
1 import java.util.Scanner;
2 import java.util.Set;
3
4 import Algorithm.Algorithm;
5 import Mapper.TextParser;
6 import javafx.util.Pair;
7 import java.util.Map;
8 import java.util.List;
9 import java.sql.Time;
10 import java.util.ArrayList;
11 import java.util.InputMismatchException;
12
13 public class WorldLadderGameCLI {
14
15     public static void main(String[] args) {
16         Scanner scanner = new Scanner(System.in);
17
18         try {
19             // Load the data for the word ladder
20             Map<String, List<String>> data = TextParser.loadMap("dictionary/map.txt");
21             Set<String> words = TextParser.loadSet("dictionary/words.txt");
22
23             // Main menu loop
24             boolean running = true;
25             while (running) {
26                 // Display menu options
27                 System.out.println();
28                 System.out.println("Choose an option:");
29                 System.out.println("1. Input words and solver method");
30                 System.out.println("2. Exit");
31
32                 System.out.print("Enter your choice: ");
33                 int choice = scanner.nextInt(); // Read the user's choice
34                 scanner.nextLine(); // Consume the newline character
35
36                 switch (choice) {
37                     case 1:
38                         // Get the words and solver method from the user
39                         String word1, word2;
40                         do {
41                             System.out.print("Enter the first word: ");
42                             word1 = scanner.nextLine().trim().toLowerCase();
43
44                             if (!TextParser.isValidWord(word1, words)) {
45                                 System.out.println("Error: The first word is not valid.");
46                             }
47                         } while (!TextParser.isValidWord(word1, words)); // Continue until the word is valid
48
49                         // Get a valid second word, ensuring it's the same length as the first word
50                         do {
51                             System.out.print("Enter the second word: ");
52                             word2 = scanner.nextLine().trim().toLowerCase();
53
54                             if (!TextParser.isValidWord(word2, words)) {
55                                 System.out.println();
56                                 System.out.println("Error: The second word is not valid.");
57                                 System.out.println();
58                             } else if (word1.length() != word2.length()) {
59                                 System.out.println();
60                                 System.out.println("Error: The second word must be the same length as the first word.");
61                                 System.out.println();
62                             }
63                         } while (!TextParser.isValidWord(word2, words) || word1.length() != word2.length());
64
65                         System.out.println("Select the solver method:");
66                         System.out.println("1. UCS");
67                         System.out.println("2. GBFS");
68                         System.out.println("3. A*");
69
70                         // Validate the solver method choice
71                         int solverChoice = 0; // Initialize with an invalid value
72                         boolean validInput = false;
73
74                         while (!validInput) {
75                             try {
76                                 System.out.print("Enter a solver method (1-3): "); // Request input
77                                 solverChoice = scanner.nextInt(); // Read integer input
78                                 scanner.nextLine(); // Clear any leftover newline
79
80                                 if (solverChoice < 1 || solverChoice > 3) { // Validate range
81                                     System.out.println("Error: Available methods are 1, 2, or 3. Please try again.");
82                                 } else {
83                                     validInput = true; // Valid input received
84                                 }
85                             } catch (InputMismatchException e) { // Handle non-integer input
86                                 System.out.println("Error: Please enter a valid number between 1 and 3.");
87                                 scanner.nextLine(); // Clear invalid input from the buffer
88                             }
89                         }
90                     }
91                 }
92             }
93         }
94     }
95 }
```

```

1      // Map the choice to the solver method name
2      String solverMethod;
3      switch (solverChoice) {
4          case 1:
5              solverMethod = "UCS";
6              break;
7          case 2:
8              solverMethod = "GBFS";
9              break;
10         case 3:
11             solverMethod = "A*";
12             break;
13         default:
14             throw new IllegalStateException("Unexpected value: " + solverChoice);
15     }
16
17     // Create the algorithm instance and evaluate
18
19     Algorithm algorithm = new Algorithm(solverMethod, word1.toLowerCase(), word2.toLowerCase());
20     System.out.println("Solver method: " + solverMethod + "\n" + "Start word: " + word1 + "\n"
21         + "End word: " + word2 + "\n");
22
23     Runtime runtime = Runtime.getRuntime();
24
25     // Run the algorithm and measure the time taken
26     runtime.gc();
27
28     long memorySebelum = runtime.totalMemory() - runtime.freeMemory();
29
30     Time startTime = new Time(System.currentTimeMillis());
31     Pair<ArrayList<String>, Integer> result = algorithm.Evaluate(data);
32
33     long memorySesudah = runtime.totalMemory() - runtime.freeMemory();
34
35     long memory = memorySesudah - memorySebelum;
36     Time endTime = new Time(System.currentTimeMillis());
37
38     System.out
39         .println("Time taken: " + (endTime.getTime() - startTime.getTime()) + " milliseconds");
40     System.out.println("Memory Usage: " + memory + " bytes");
41     // Display the results
42     if (result.getKey().size() != 0) {
43         System.out.println("Path: " + result.getKey());
44         System.out.println("Visited Nodes: " + result.getValue());
45         System.out.println("Path Length: " + (result.getKey().size()));
46     }
47
48     break;
49
50     case 2:
51         // Exit the program
52         running = false;
53         System.out.println("Exiting...");
54         break;
55
56     default:
57         System.out.println("Invalid choice. Please choose 1 or 2.");
58         break;
59 }
60
61 } catch (Exception e) {
62     System.err.println("An error occurred: " + e.getMessage());
63 } finally {
64     scanner.close(); // Ensure the scanner is closed
65 }
66
67 }
68 }
69

```

7.6. Source Code Class WorldLadderGameGUI

```
1 import javax.swing.*;
2 import javax.swing.border.EmptyBorder;
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.sql.Time;
6 import java.util.Set;
7 import java.util.HashSet;
8 import java.util.Map;
9 import java.util.List;
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.Collections;
13 import java.util.Arrays;
14
15 public class WorldLadderGameGUI {
16     private static Set<String> words;
17     private static Map<String, List<String>> data;
18     private static String[] colorPalette = { "#00C9A7", "#00AFA6", "#00949C", "#017A8B", "#266074", "#FCF0FC" };
19
20     public static void main(String[] args) {
21         SwingUtilities.invokeLater(() -> createAndShowGUI());
22     }
23
24     private static void createAndShowGUI() {
25         int countRow = 0;
26         words = new HashSet<>(TextParser.loadSet("dictionary/words.txt"));
27         data = TextParser.loadMap("dictionary/map.txt");
28
29         JFrame frame = new JFrame("WorldLadderGame");
30         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31         frame.setSize(1000, 700);
32
33         // Define fonts
34         Font authorFont = new Font("Montserrat", Font.BOLD, 24); // Author line font
35         Font inputFont = new Font("Montserrat", Font.BOLD, 20); // Input fields and labels font
36         Font textInputFont = new Font("Montserrat", Font.PLAIN, 20); // Text input font
37
38         // Main panel with GridBagLayout for custom arrangement
39         JPanel mainPanel = new JPanel(new GridBagLayout());
40         mainPanel.setBackground(Color.decode(colorPalette[1])); // Set consistent background color
41         GridBagConstraints gbc = new GridBagConstraints();
42         gbc.insets = new Insets(5, 10, 10, 10); // Padding
43         gbc.anchor = GridBagConstraints.CENTER; // Center components horizontally
44
45         // Add an image above the title
46         gbc.gridx = 0; // First column
47         gbc.gridy = countRow; // First row
48         gbc.gridwidth = 3; // Span three columns
49
50         ImageIcon imageIcon = new ImageIcon("img/logo.png"); // Modify with the path to your image file
51
52         // Optionally, you can resize the image
53         Image scaledImage = imageIcon.getImage().getScaledInstance(275, 184, Image.SCALE_SMOOTH);
54         imageIcon = new ImageIcon(scaledImage);
55
56         // Add the image to a JLabel
57         JLabel imageLabel = new JLabel(imageIcon);
58         mainPanel.add(imageLabel, gbc);
59
60         countRow++;
61
62         // Title label spanning three columns
63         gbc.gridx = 0; // Start at the first column
64         gbc.gridy = countRow; // First row
65         gbc.gridwidth = 3; // Span across three columns
66         gbc.gridheight = 1;
67         countRow++; // Increment row count
68
69         // Author label spanning three columns
70         gbc.gridx = 0; // First column
71         gbc.gridy = countRow; // Second row
72         gbc.gridwidth = 3;
73         JLabel authorLabel = new JLabel("Made by: Imanuel Sebastian Girsang - 13522058", JLabel.CENTER);
74         authorLabel.setFont(authorFont); // Set font for the author line
75         authorLabel.setForeground(Color.decode(colorPalette[5])); // Set author line color
76         authorLabel.setBorder(BorderFactory.createEmptyBorder(0, 0, 20, 0));
77         mainPanel.add(authorLabel, gbc);
78
79         countRow++; // Increment row count
80         gbc.gridwidth = 1;
81
82         // First word input field with label
83         gbc.gridx = 0; // First column
84         gbc.gridy = countRow; // Second row
85         JLabel startWordLabel = new JLabel("Start Word:");
86         startWordLabel.setFont(inputFont); // Set font for the label
87         startWordLabel.setForeground(Color.decode(colorPalette[5])); // Set label color
88         mainPanel.add(startWordLabel, gbc);
89
90         gbc.gridx = 1; // Move to the second column
91         gbc.gridwidth = 2; // Span two columns
92         JTextField startWordField = new JTextField(27); // Width of the text field
93         startWordField.setFont(textInputFont); // Set font for the input field
94         mainPanel.add(startWordField, gbc);
95     }
96 }
```

```

1  countRow++; // Increment row count
2
3      // Second word input field with label
4      gbc.gridx = 0; // First column
5      gbc.gridy = countRow; // Third row
6      JLabel endWordLabel = new JLabel("End Word:");
7      endWordLabel.setFont(inputFont); // Set font for the label
8      endWordLabel.setForeground(Color.decode(colorPalette[5])); // Set label color
9      mainPanel.add(endWordLabel, gbc);
10
11     gbc.gridx = 1; // Move to the second column
12     gbc.gridwidth = 2; // Span two columns
13     JTextField endWordField = new JTextField(27); // Width of the text field
14     endWordField.setFont(textInputFont); // Set font for the input field
15     mainPanel.add(endWordField, gbc);
16
17     countRow++; // Increment row count
18
19     // Solver method dropdown with label
20     gbc.gridx = 0; // First column
21     gbc.gridy = countRow; // Fourth row
22     JLabel solverMethodLabel = new JLabel("Algorithm: ");
23     solverMethodLabel.setFont(inputFont); // Set font for the label
24     solverMethodLabel.setForeground(Color.decode(colorPalette[5])); // Set label color
25     mainPanel.add(solverMethodLabel, gbc);
26
27     gbc.gridx = 1; // Move to the second column
28     gbc.gridwidth = 2; // Span two columns
29     JComboBox<String> solverMethodComboBox = new JComboBox<>(new String[] { "UCS", "GBFS", "A*" });
30     solverMethodComboBox.setPreferredSize(new Dimension(570, 30)); // Set preferred size
31     solverMethodComboBox.setFont(inputFont); // Set font for the combo box
32     mainPanel.add(solverMethodComboBox, gbc);
33
34     countRow++; // Increment row count
35
36     // Submit button spanning three columns and centered
37     gbc.gridx = 0; // First column
38     gbc.gridy = countRow; // Fifth row
39     gbc.gridwidth = 3; // Span three columns
40     gbc.anchor = GridBagConstraints.CENTER; // Center the button
41     JButton submitButton = new JButton("Submit");
42     submitButton.setFont(inputFont); // Set font for the submit button
43     mainPanel.add(submitButton, gbc);
44
45     // Add arrow key navigation and enter key submission
46     KeyListener keyListener = new KeyAdapter() {
47         @Override
48         public void keyPressed(KeyEvent e) {
49             int keyCode = e.getKeyCode();
50
51             if (keyCode == KeyEvent.VK_ENTER) {
52                 // Trigger submit button when Enter is pressed
53                 submitButton.doClick();
54             } else if (keyCode == KeyEvent.VK_DOWN) {
55                 // Move focus to the next field when DOWN is pressed
56                 if (e.getSource() == startWordField) {
57                     endWordField.requestFocus();
58                 }
59             } else if (keyCode == KeyEvent.VK_UP) {
60                 // Move focus to the previous field when UP is pressed
61                 if (e.getSource() == endWordField) {
62                     startWordField.requestFocus();
63                 }
64             }
65         }
66     };
67
68     startWordField.addKeyListener(keyListener); // Attach to text fields
69     endWordField.addKeyListener(keyListener);

```

```

1 submitButton.addActionListener(new ActionListener() {
2     @Override
3     public void actionPerformed(ActionEvent e) {
4         String startWord = startWordField.getText().trim().toLowerCase();
5         String endWord = endWordField.getText().trim().toLowerCase();
6         String solverMethod = (String) solverMethodComboBox.getSelectedItem();
7
8         // Validate inputs
9
10        if (startWord.isEmpty()) {
11            JOptionPane.showMessageDialog(frame, "Error: Please enter your start word.",
12                "Invalid Input",
13                JOptionPane.ERROR_MESSAGE);
14            return;
15        }
16
17        if (endWord.isEmpty()) {
18            JOptionPane.showMessageDialog(frame, "Error: Please enter your end word.",
19                "Invalid Input",
20                JOptionPane.ERROR_MESSAGE);
21            return;
22        }
23        if (!TextParser.isValidWord(startWord, words)) {
24            JOptionPane.showMessageDialog(frame, "Error: The start word is not valid.", "Invalid Input",
25                JOptionPane.ERROR_MESSAGE);
26            return;
27        }
28
29        if (!TextParser.isValidWord(endWord, words)) {
30            JOptionPane.showMessageDialog(frame, "Error: The end word is not valid.", "Invalid Input",
31                JOptionPane.ERROR_MESSAGE);
32            return;
33        }
34
35        if (startWord.length() != endWord.length()) {
36            JOptionPane.showMessageDialog(frame,
37                "Error: The start word and end word must have the same length.",
38                "Invalid Input", JOptionPane.ERROR_MESSAGE);
39            return;
40        }
41
42        // If valid, proceed with processing the inputs and showing results
43        Algorithm algorithm = new Algorithm(solverMethod, startWord.toLowerCase(), endWord.toLowerCase());
44
45        // Start memory and time counter
46
47        Runtime runtime = Runtime.getRuntime();
48
49        runtime.gc(); // Garbage collection to ensure clean memory
50
51        long memorySebelum = runtime.totalMemory() - runtime.freeMemory();
52
53        Time startTime = new Time(System.currentTimeMillis());
54        Pair<ArrayList<String>, Integer> result = algorithm.Evaluate(data);
55
56        long memorySesudah = runtime.totalMemory() - runtime.freeMemory();
57
58        // Calculate memory used
59
60        long memoryUsed = memorySesudah - memorySebelum;
61        Time endTime = new Time(System.currentTimeMillis());
62
63        long timeTaken = endTime.getTime() - startTime.getTime();
64
65        // Show the results in a new window
66        showResults(startWord, endWord, timeTaken, result.getValue(), result.getKey().size(),
67            result.getKey(), memoryUsed, solverMethod);
68    }
69 });
70
71 frame.add(mainPanel);
72 frame.setVisible(true);
73 }

```

```

1 public static void showResults(String startWord, String endWord, long timeTaken, int visitedNodes, int pathLength,
2     ArrayList<String> path, long memoryUsed, String method) {
3     JFrame resultFrame = new JFrame("WordLadder Results");
4     resultFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
5     resultFrame.setSize(900, 900);
6
7     // Center the frame horizontally on the screen
8     resultFrame.setLocationRelativeTo(null);
9
10    // Set up fonts
11    Font summaryFont = new Font("Montserrat", Font.BOLD, 20);
12
13    Font resultFont = new Font("Montserrat", Font.BOLD, 40);
14    Font pathFont = new Font("Montserrat", Font.BOLD, 28);
15
16    JPanel mainPanel = new JPanel(new BorderLayout());
17
18    // Create a summary panel with GridBagLayout
19    JPanel summaryPanel = new JPanel(new GridBagLayout());
20    GridBagConstraints gbc = new GridBagConstraints();
21    gbc.insets = new Insets(5, 10, 5, 10); // Padding
22    gbc.anchor = GridBagConstraints.CENTER;
23    // Set consistent background color
24
25    int currentRow = 0;
26
27    // Add the result label
28    gbc.gridx = 0; // Leftmost position
29    gbc.gridy = currentRow; // First row
30    gbc.gridwidth = 1; // Reset grid width
31    JLabel resultLabel = new JLabel("Result of", SwingConstants.CENTER);
32    resultLabel.setFont(resultFont);
33    resultLabel.setForeground(Color.decode(colorPalette[5])); // Set label color
34    resultLabel.setBackground(Color.decode(colorPalette[5]));
35    summaryPanel.add(resultLabel, gbc);
36
37    currentRow++; // Increment row count
38
39    gbc.gridy = currentRow; // Second row
40
41    String text = String.format("<b>%s</b> to <b>%s</b>", startWord.toUpperCase(), endWord.toUpperCase());
42
43    JLabel pathLabel = new JLabel("<html>" + text + "</html>", SwingConstants.CENTER); // Use HTML formatting
44
45    pathLabel.setFont(resultFont);
46    pathLabel.setForeground(Color.decode(colorPalette[5])); // Set label color
47    pathLabel.setBackground(Color.decode(colorPalette[5]));
48    summaryPanel.add(pathLabel, gbc);
49
50    // Add time taken
51    currentRow++;
52    gbc.gridy = currentRow;
53    summaryPanel.add(createStyledLabel("Time Taken: " + timeTaken + " ms", summaryFont), gbc);
54
55    currentRow++;
56    gbc.gridy = currentRow;
57    summaryPanel.add(createStyledLabel("Memory used: " + memoryUsed + " bytes", summaryFont), gbc);
58
59    // Add visited nodes
60    currentRow++;
61    gbc.gridy = currentRow;
62    summaryPanel.add(createStyledLabel("Visited Nodes: " + visitedNodes, summaryFont), gbc);
63
64    // Add path length
65    currentRow++;
66    gbc.gridy = currentRow;
67    summaryPanel.add(createStyledLabel("Path Length: " + pathLength, summaryFont), gbc);
68
69    currentRow++;
70    gbc.gridy = currentRow;
71    summaryPanel.add(createStyledLabel("Algorithm: " + method, summaryFont), gbc);
72
73    Font pathSummaryFont = new Font("Montserrat", Font.BOLD, 14);
74
75    currentRow++;
76    gbc.gridy = currentRow;
77    JTextArea textArea = new JTextArea("Summary Path: " + path, (path.size() + 1) / 10, 50);
78    textArea.setFont(pathSummaryFont);
79    textArea.setForeground(Color.decode(colorPalette[5]));
80    textArea.setLineWrap(true);
81    textArea.setWrapStyleWord(true); // Wraps at word boundaries
82    textArea.setEditable(false); // Makes it non-editable
83    textArea.setBackground(Color.decode(colorPalette[2]));
84    textArea.setBorder(new EmptyBorder(10, 10, 10, 10)); // Padding
85
86    summaryPanel.add(new JScrollPane(textArea), gbc);
87
88    currentRow++;
89    gbc.gridy = currentRow;

```

```

1 Icon imageIcon;
2     if (path != null && !path.isEmpty()) {
3         imageIcon = new ImageIcon("img/success.gif");
4     } else {
5         imageIcon = new ImageIcon("img/crybaby.gif");
6     }
7
8     // Add the image to a JLabel
9     JLabel imageLabel = new JLabel(imageIcon);
10    summaryPanel.add(imageLabel, gbc);
11    summaryPanel.setBackground(Color.decode(colorPalette[1])); // Set consistent background color
12    mainPanel.add(summaryPanel, BorderLayout.NORTH);
13
14    // Paths Panel
15    JPanel pathsPanel = new JPanel(new GridBagLayout()); // Layout depends on whether there's a solution
16    GridBagConstraints pathGbc = new GridBagConstraints();
17    pathGbc.gridx = 0; // Posisi kolom pasti sama,
18    pathGbc.gridy = 0; // Posisi row dimulai dari 0
19    pathGbc.insets = new Insets(5, 5, 5, 5);
20    pathsPanel.setBackground(Color.decode(colorPalette[1])); // Set consistent background color
21
22    if (path != null && !path.isEmpty()) {
23
24        for (int i = 0; i < path.size(); i++) {
25            String word = path.get(i);
26            JPanel wordPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
27
28            for (int j = 0; j < word.length(); j++) {
29                JPanel boxPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
30                boxPanel.setPreferredSize(new Dimension(50, 50)); // Uniform size for the box
31                boxPanel.setBackground(Color.BLACK); // Default background color
32                boxPanel.setOpaque(true); // Ensure opacity
33
34                JLabel letterLabel = new JLabel(String.valueOf(word.charAt(j)), JLabel.CENTER);
35                letterLabel.setFont(pathFont); // Set font for the letter
36                letterLabel.setForeground(Color.WHITE); // Default text color
37
38                if (word.charAt(j) == endWord.charAt(j)) { // Matching letter
39                    boxPanel.setBackground(Color.decode(colorPalette[0])); // Green for matched letters
40                    letterLabel.setForeground(Color.BLACK); // Change text color to black
41                }
42
43                if (i > 0 && word.charAt(j) != path.get(i - 1).charAt(j)) {
44                    // Apply a border to emphasize the box around the letter
45                    boxPanel.setBorder(BorderFactory.createLineBorder(Color.ORANGE, 5)); // Box border
46                }
47
48                boxPanel.add(letterLabel); // Add the letter to the box panel
49                wordPanel.add(boxPanel); // Add box panel to word panel
50            }
51            pathGbc.gridy = i; // Set the row for the current word
52
53            pathsPanel.add(wordPanel, pathGbc); // Add word panel to paths panel
54        }
55    } else {
56        // If no solution, display a message
57        pathsPanel.setLayout(new GridLayout(1, 1));
58        JLabel noSolutionLabel = createStyledLabel("No solution found :((", summaryFont);
59        pathsPanel.add(noSolutionLabel);
60    }
61
62    // Set consistent scroll pane background
63    mainPanel.add(pathsPanel); // Add scroll pane to the main panel
64
65    JScrollPane scrollPane = new JScrollPane(mainPanel, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
66        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
67    scrollPane.setBorder(BorderFactory.createEmptyBorder());
68
69    resultFrame.add(scrollPane);
70    resultFrame.setVisible(true); // Show the results frame
71 }
72
73 // Helper method to create styled labels with specific font
74 private static JLabel createStyledLabel(String text, Font font) {
75     JLabel label = new JLabel(text, JLabel.CENTER);
76     label.setFont(font);
77     label.setForeground(Color.decode(colorPalette[5])); // Set label color
78
79     return label;
80 }
81
82 }
83

```