

LAPORAN TUGAS KECIL III
IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

13522148 Auralea Alvinia Syaikha

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER II TAHUN 2023/2024

DAFTAR ISI

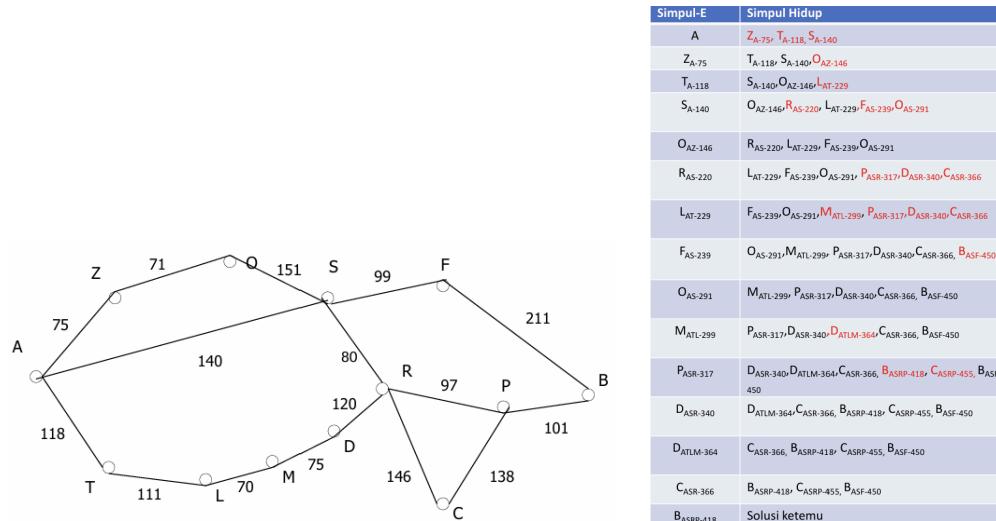
DAFTAR ISI.....	1
BAB I	
LANDASAN TEORI.....	3
1.1. Algoritma Uniform Cost Search.....	3
1.2. Algoritma Greedy Best-first Search.....	4
1.3. Algoritma A*.....	5
BAB II	
ANALISIS DAN IMPLEMENTASI.....	7
2.1 Algoritma Uniform Cost Search.....	7
2.2 Algoritma Greedy Best-first Search.....	8
2.3 Algoritma A*.....	9
BAB III	
SOURCE CODE & ANALISIS IMPLEMENTASI ALGORITMA.....	11
3.1 Dengan Graphical User Interface (GUI).....	11
3.2 Dengan Command Line Interface (CLI).....	23
BAB IV	
HASIL UJI COBA.....	29
4.1 Hasil Uji Coba Algoritma Uniform Cost Search (UCS).....	29
4.2 Hasil Uji Coba Algoritma Greedy Best-first Search.....	32
4.3 Hasil Uji Coba Algoritma A*	35
BAB V	
ANALISIS PERBANDINGAN SOLUSI.....	38
5.1 Berdasarkan Optimalitas.....	38
5.2 Berdasarkan Waktu Eksekusi.....	38
5.3 Berdasarkan Penggunaan Memori.....	39
BAB VI	
PENJELASAN BONUS.....	40
6.1 Graphical User Interface (GUI).....	40
BAB VII	
PENUTUP.....	43
5.4 Kesimpulan.....	43
5.5 Saran.....	43
DAFTAR PUSTAKA.....	44
LAMPIRAN.....	45

GitHub Repository (Latest Release).....	45
Tabel Spesifikasi.....	45

BAB I

LANDASAN TEORI

1.1. Algoritma *Uniform Cost Search*



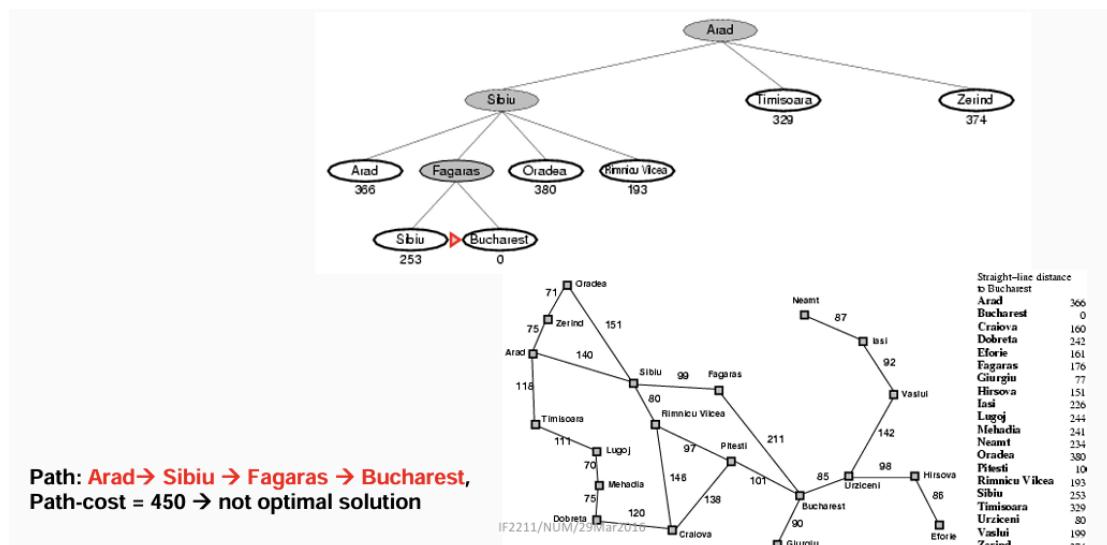
Sumber: [Route-Planning-Bagian1-2021.pdf \(itb.ac.id\)](https://itb.ac.id/)

Algoritma Uniform Cost Search (UCS) merupakan sebuah pendekatan yang efektif dalam mencari jalur terpendek di dalam sebuah graf berbobot. Prinsip kerja dari algoritma ini dimulai dengan inisialisasi pada simpul awal, di mana nilai biaya awal diberikan pada simpul tersebut dan kemudian ditandai sebagai simpul yang telah dieksplorasi. Selanjutnya, algoritma akan melakukan ekspansi dari simpul saat ini dengan mempertimbangkan semua simpul yang dapat dicapai melalui edge yang belum dieksplorasi. Setelah itu, algoritma akan menyaring simpul yang dieksplorasi untuk menentukan apakah memiliki biaya yang lebih rendah dari biaya saat ini. Jika ya, maka biaya dan jalur terpendek menuju simpul tersebut akan diperbarui. Langkah-langkah ekspansi dan penyaringan akan diulangi hingga simpul tujuan ditemukan atau tidak ada simpul yang tersisa untuk dieksplorasi.

Kelebihan dari algoritma UCS termasuk optimalitasnya dalam menemukan jalur terpendek serta keserbagunaannya karena dapat digunakan dalam graf dengan bobot yang beragam tanpa perlu mengetahui estimasi jarak ke simpul tujuan. Namun, algoritma ini memiliki keterbatasan dalam hal

kompleksitas waktu yang tinggi terutama pada graf yang kompleks serta membutuhkan penyimpanan biaya dan jalur untuk setiap simpul yang dieksplorasi yang dapat menghabiskan banyak memori. UCS seringkali diterapkan dalam sistem navigasi, pemetaan jalan, perencanaan rute, serta dalam pemrosesan bahasa alami untuk menemukan jalur terpendek dalam graf representasi struktur bahasa seperti graf dependensi. Dengan pemahaman mengenai prinsip kerja, kelebihan, dan keterbatasannya, algoritma UCS dapat menjadi alat yang sangat berguna dalam penyelesaian berbagai masalah dalam bidang kecerdasan buatan.

1.2. Algoritma *Greedy Best-first Search*

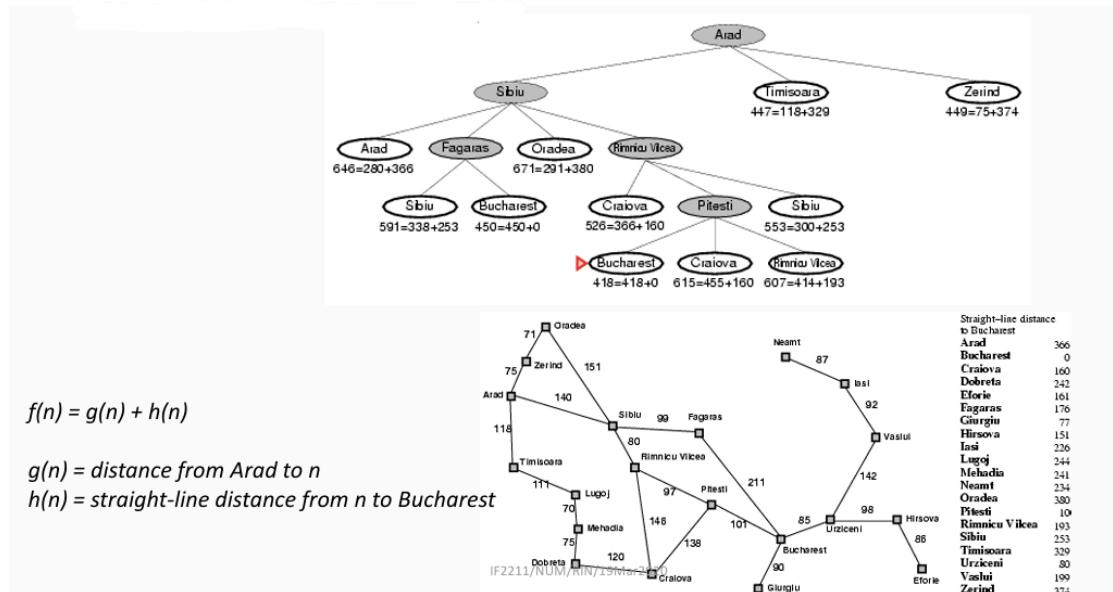


Sumber: [Route-Planning-Bagian1-2021.pdf \(itb.ac.id\)](https://Route-Planning-Bagian1-2021.pdf)

Algoritma Greedy Best First Search (GBFS) adalah pendekatan pencarian jalur yang digunakan dalam kecerdasan buatan dan pemrosesan bahasa alami. Berbeda dengan algoritma pencarian jalur tradisional, GBFS tidak mempertimbangkan semua kemungkinan langkah, melainkan hanya memilih langkah yang paling dekat dengan tujuan berdasarkan suatu heuristik. Prinsip kerja GBFS mirip dengan algoritma Greedy, di mana langkah yang diambil ditentukan oleh penilaian heuristik tanpa mempertimbangkan konsekuensi jangka panjang. Dengan demikian, GBFS cenderung untuk "berpikir" secara lokal dan memilih langkah yang terlihat paling menjanjikan pada setiap titik dalam pencarian. Kelebihan utama dari GBFS adalah

kecepatan pencarian yang tinggi karena hanya mempertimbangkan langkah yang paling dekat dengan tujuan. Namun, kelemahannya adalah bahwa algoritma ini tidak menjamin solusi optimal karena dapat terjebak dalam lokal minimum atau maksimum. Penerapan GBFS sangat luas, terutama dalam navigasi, pencarian jalan, dan perencanaan rute. Dalam pemrosesan bahasa alami, GBFS dapat digunakan untuk menemukan jalur terpendek dalam graf dependensi atau dalam proses parsing sintaksis. Dengan pemahaman yang mendalam tentang prinsip kerja dan karakteristiknya, GBFS dapat menjadi alat yang efektif dalam menyelesaikan berbagai permasalahan dalam bidang kecerdasan buatan dan pemrosesan bahasa alami.

1.3. Algoritma A*



Sumber: [Route-Planning-Bagian2-2021.pdf \(itb.ac.id\)](https://Route-Planning-Bagian2-2021.pdf)

Algoritma A* merupakan algoritma pencarian jalur yang efisien dan sering digunakan dalam kecerdasan buatan dan pemrosesan bahasa alami. A* memadukan teknik Greedy Best First Search (GBFS) dengan pendekatan berbasis biaya seperti dalam algoritma Dijkstra. Prinsip kerja A* adalah mencari jalur terpendek dari simpul awal ke simpul tujuan dengan mempertimbangkan biaya aktual dari simpul awal ke simpul saat ini (biaya g)

dan estimasi biaya dari simpul saat ini ke simpul tujuan (biaya h), yang disebut heuristik. Algoritma A* menggunakan fungsi heuristik untuk memprediksi biaya yang tersisa menuju tujuan, yang membantu dalam menghindari pencarian yang tidak produktif. Dengan menggabungkan biaya aktual dan estimasi biaya sisa ($f = g + h$), A* mampu menemukan jalur optimal dengan waktu eksekusi yang relatif cepat. Keunggulan utama A* adalah kemampuannya dalam menemukan solusi optimal dan efisien dalam graf dengan bobot yang beragam. Namun, kelemahannya terletak pada kompleksitas perhitungan fungsi heuristik yang dapat mempengaruhi kinerja algoritma secara keseluruhan. A* sering digunakan dalam berbagai aplikasi termasuk navigasi, perencanaan rute, dan pemetaan jalan. Dalam pemrosesan bahasa alami, A* dapat diterapkan untuk parsing sintaksis atau pencarian jalur terpendek dalam graf dependensi. Dengan pemahaman yang mendalam tentang prinsip kerja dan implementasinya, A* dapat menjadi alat yang sangat berguna dalam menyelesaikan berbagai masalah dalam bidang kecerdasan buatan dan pemrosesan bahasa alami.

BAB II

ANALISIS DAN IMPLEMENTASI

2.1 Algoritma *Uniform Cost Search*

Algoritma Uniform Cost Search (UCS) digunakan untuk mencari jalur terpendek dari suatu kata awal (`beginWord`) ke kata akhir (`endWord`) dalam daftar kata (`wordList`). Program dimulai dengan mengakses sebuah file teks yang berisi daftar kata, dimana setiap kata terdapat pada baris yang berbeda. File tersebut diolah, dan setiap kata dimasukkan ke dalam sebuah List. Kamus kata ini menjadi landasan penting dalam proses penemuan jalur dalam Word Ladder. Setelah kamus disiapkan, pengguna diminta untuk memasukkan kata awal dan kata akhir. Program melakukan validasi terhadap kedua kata ini untuk memastikan bahwa keduanya memiliki panjang yang sama, keduanya merupakan sebuah kata dan tidak mengandung karakter lain selain alfabet, dan kata akhir terdapat dalam kamus kata. Jika salah satu kata tersebut tidak valid, program akan menampilkan pesan kesalahan dan keluar.

Dalam implementasi algoritma UCS, program dimulai dengan memeriksa apakah kata akhir ada dalam daftar kata. Jika tidak, program mengembalikan jalur kosong dan jumlah node yang telah dikunjungi. Selanjutnya, sebuah PriorityQueue digunakan untuk menyimpan node-node yang dikunjungi secara berurutan berdasarkan biaya mereka. Set `visited` digunakan untuk melacak node-node yang telah dikunjungi. Program kemudian memulai iterasi menggunakan loop while hingga PriorityQueue tidak kosong.

Di setiap iterasi, node dengan biaya terendah diambil dari PriorityQueue. Jika node yang diambil merupakan kata akhir, jalur dari kata awal ke kata akhir ditemukan dan program mengembalikan jalur tersebut bersama dengan jumlah node yang telah dikunjungi. Jika tidak, program menambahkan node-node baru ke PriorityQueue untuk memeriksa kemungkinan kata-kata yang dapat dijangkau dari kata saat ini dengan biaya 1 lebih tinggi dari biaya sebelumnya. Proses ini berlanjut hingga jalur dari kata awal ke kata akhir ditemukan atau tidak ada node yang tersisa untuk dieksplorasi. Jika kata akhir tidak dapat dicapai, program mengembalikan jalur kosong dan jumlah node yang telah dikunjungi.

Dalam algoritma UCS, prioritas setiap node ditentukan oleh fungsi $f(n) = g(n)$, di mana $g(n)$ adalah jumlah langkah dari awal hingga node tersebut. Dalam penerapannya, UCS hanya mempertimbangkan biaya berdasarkan jumlah langkah tanpa memperhitungkan tambahan heuristik. Meskipun UCS hanya memperhitungkan biaya berdasarkan langkah, hasilnya tetap optimal untuk konteks ini karena memastikan bahwa node dengan biaya terendah selalu diekstraksi dan diekspansi terlebih dahulu.

2.2 Algoritma *Greedy Best-first Search*

Algoritma Greedy Best-First Search (Greedy BFS) adalah suatu metode pencarian yang mengarahkan eksplorasi berdasarkan estimasi jarak ke tujuan, yang dikenal sebagai heuristik. Dalam konteks Word Ladder, heuristik ini bisa dihitung sebagai jumlah karakter yang berbeda antara kata saat ini dan kata tujuan.

Pada awalnya, kamus kata-kata diubah menjadi sebuah Set untuk pencarian yang lebih efisien. Selanjutnya, sebuah Priority Queue digunakan untuk menyimpan prioritas node-node yang akan diekspansi. Prioritas dalam algoritma ini didasarkan pada heuristik, yang dikenal sebagai $h(n)$, yang mengukur seberapa dekat sebuah kata saat ini dengan kata tujuan. Algoritma ini memanfaatkan fungsi `hammingDistance` untuk menghitung jarak Hamming antara dua kata, yang mengukur perbedaan karakter antara keduanya.

Algoritma kemudian memulai iterasi menggunakan loop while hingga Priority Queue tidak kosong. Di setiap iterasi, node dengan jarak Hamming terkecil diambil dari Priority Queue untuk diekspansi seterusnya, dan sisa elemen yang ada pada Priority Queue dihapus. Jika node yang diambil merupakan kata akhir, jalur dari kata awal ke kata akhir ditemukan dan algoritma mengembalikan jalur tersebut bersama dengan jumlah node yang telah dikunjungi. Jika tidak, algoritma menambahkan node-node baru ke Priority Queue untuk memeriksa kemungkinan kata-kata yang dapat dijangkau dari kata saat ini. Proses ini berlanjut hingga jalur dari kata awal ke kata akhir ditemukan atau tidak ada node yang tersisa untuk eksplorasi.

Berbeda dengan algoritma lain seperti A* atau Uniform-Cost Search (UCS), Greedy BFS hanya mempertimbangkan heuristik untuk menentukan prioritas ekspansi tanpa memperhitungkan biaya langkah yang telah dilalui, dibuktikan dengan nilai $f(n)$ (yang menentukan prioritas) sama dengan $h(n)$. Oleh karena itu, Greedy BFS cenderung mengeksplorasi node yang

tampaknya paling dekat dengan tujuan, meskipun ada risiko bahwa jalur yang dipilih bukanlah jalur terpendek. Meskipun Greedy BFS dapat menghasilkan solusi dengan cepat, ini tidak menjamin bahwa jalur tersebut adalah yang terpendek atau paling efisien. Dengan mengorbankan optimalitas demi kecepatan, Greedy BFS bisa menghasilkan hasil yang kurang optimal dalam beberapa kasus, termasuk dalam persoalan Word Ladder.

2.3 Algoritma A*

Algoritma A* adalah metode pencarian jalur yang menerapkan konsep heuristik untuk menaksir sisa biaya yang diperlukan untuk mencapai tujuan, dan kemudian menggabungkan dengan biaya langkah yang telah dikeluarkan. Algoritma A* Search menggabungkan pendekatan *Greedy Best-First Search* dengan *Uniform-Cost Search* (UCS) dengan mempertimbangkan biaya langkah yang telah ditempuh serta heuristik estimasi jarak menuju tujuan.

Proses dimulai dengan membangun sebuah kamus kata-kata yang akan digunakan untuk memeriksa validitas kata dan mencari tetangga. Kamus ini diwakili oleh sebuah Set yang terbentuk dari daftar kata yang diberikan sebagai input. Selanjutnya, sebuah priority queue digunakan untuk menentukan urutan ekspansi node. Prioritas ditentukan oleh nilai $f(n) = g(n) + h(n)$, di mana $g(n)$ adalah biaya langkah dari awal hingga node tersebut, dan $h(n)$ adalah heuristik yang mengestimasi jarak dari node tersebut ke titik tujuan, yaitu endWord. Node awal, yang merupakan kata awal, ditambahkan ke dalam priority queue dengan nilai $f(n)$ awal yang dihitung dari heuristik dan biaya langkah yang nol. Selama priority queue belum kosong, algoritma mengekstrak node dengan prioritas terendah dari priority queue. Jika node yang diekstrak merupakan kata akhir, algoritma mengembalikan jalur yang ditemukan beserta jumlah node yang telah dikunjungi.

Jika node yang diekstrak bukan kata akhir, algoritma melakukan ekspansi dengan mempertimbangkan semua kemungkinan kata yang berbeda satu karakter dari kata saat ini. Kata-kata yang memenuhi kriteria, yaitu belum pernah dikunjungi dan terdapat dalam kamus, ditambahkan ke dalam priority queue dengan nilai $f(n)$ yang dihitung berdasarkan heuristik dan biaya langkah yang baru. Jika tidak ada jalur yang ditemukan atau endWord tidak dapat dicapai, algoritma mengembalikan jalur kosong dan jumlah node yang telah dikunjungi.

Heuristik pada algoritma A* bisa dianggap admissible karena tidak mengestimasi biaya yang lebih tinggi dari biaya aktual menuju tujuan dengan

cara memperkirakan langkah minimum untuk mencapai tujuan. Secara teoritis, algoritma A* dapat menjadi lebih efisien daripada UCS karena UCS hanya mempertimbangkan biaya langkah yang telah dilakukan tanpa mempertimbangkan informasi apapun tentang tujuan. A* menggunakan heuristik untuk memperkirakan biaya sisa menuju tujuan, yang memungkinkan pencarian yang lebih efisien. Dalam hal ini, algoritma A* dapat mempercepat pencarian dengan mengarahkan eksplorasi ke arah yang lebih relevan.

BAB III

SOURCE CODE & ANALISIS IMPLEMENTASI ALGORITMA

3.1 Dengan *Graphical User Interface (GUI)*

MainClass.java

```
● ● ●
1 package GUI;
2 /*
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
5  */
6 e*/
7
8 /**
9  *
10 */
11 * @author Auralea A S
12 */
13 public class MainClass {
14     public static void main(String[] args){
15         System.out.println("Hello");
16
17         MainFrame myFrame = new MainFrame();
18         myFrame.init();
19     }
20 }
```

MainFrame.java

```
● ● ●
1 package GUI;
2 /*
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
5  */
6 e*/
7
8 import javax.swing.JFrame;
9 import javax.swing.WindowConstant ;
10 /**
11  *
12 */
13 * @author Auralea A S
14 */
15 public class MainFrame extends JFrame{
16     public void init(){
17         setTitle("Word Ladder Solve ");
18         setSize(450, 300);
19         setDefaultCloseOperation (WindowConstant .EXIT_ON_CLOS );
20         setVisible (true);           s             E
21     }   e
22
23
24     public static void main (String[] args){
25         MainFrame myFrame = new MainFrame();
26         myFrame.init();
27     }
28 }
```

InputFrame.java

```
1 package GUI;
2 /*
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
4  * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
5  */
6
7
8 import java.io.BufferedReader ;
9 import java.io.FileNotFoundException ;
10 import java.io.FileReader ;
11 import java.io.IOException;
12 import java.util.ArrayList;
13 import java.util.List;
14
15 /**
16 *
17 * @author Auralea A S
18 */
19 public class InputFram extends javax.swing.JFrame {
20     /**
21      * Creates new form InputFram
22      */
23     public InputFram () {
24         initComponents ();
25     }
26     /**
27      * This method is called from within the constructor to initialize the form.
28      * WARNING: Do NOT modify this code. The content of this method is always
29      * regenerated by the Form Editor.
30      */
31     // <editor-fold defaultstate="collapsed" desc="Generated Code">
32     private void initComponents () {
33         enterStartWor = new javax.swing.JLabel();
34         dstartWord = new javax.swing.JTextField();
35         enterEndwor = new javax.swing.JLabel();
36         dndWord = new javax.swing.JTextField();
37         jLabel1 = new javax.swing.JLabel();
38         btnUCS = new javax.swing.JButton();
39         btnGreedy = new javax.swing.JButton();
40         btnAstar = new javax.swing.JButton();
41         jScrollPane = new javax.swing.JScrollPane();
42         resultAre = new javax.swing.JTextArea();
43         jLabel2 = new javax.swing.JLabel();
44         btnClear = new javax.swing.JButton();
45         jPanel1 = new javax.swing.JPanel();
46         jLabel3 = new javax.swing.JLabel();
47
48         setDefaultCloseOperation (javax.swing.WindowConstants .EXIT_ON_CLOS );
49
50
51
52     setDefaultCloseOperatio (javax.swing.WindowConstant .EXIT_ON_CLOS );
```



The screenshot shows a Java Swing application window. The code is displayed in a text area with line numbers on the left. The code is written in Java and defines various components like buttons, labels, and scroll panes.

```
1 enterStartWor .setFont(new java.awt.Font("Nirmala UI Semilight", 1, 14)); // NOI18N
2 dnterStartWor .setText("Enter Start Wor t");
3 d
4 enterEndWor .setFont(new java.awt.Font("Nirmala UI Semilight", 1, 14)); // NOI18N
5 dnterEndWor .setText("Enter End Wor ");
6 d
7 jLabel1.setFont(new java.awt.Font("Nirmala UI Semilight", 1, 14)); // NOI18N
8 jLabel1.setText("Choose Algorith m:");
9 m:
10 btnUCS.setText("Uniform Cost Searc ");
11 btnUCS.addActionListener (new java.awt.event.ActionListener () {
12     public void actionPerformed (java.awt.event.ActionEvent evt) {
13         btnUCSActionPerforme (evt);
14         d
15     }
16 });
17 btnGreedy.setText("Greedy Best-first Searc ");
18 btnGreedy.addActionListener (new java.awt.event.ActionListener () {
19     public void actionPerformed (java.awt.event.ActionEvent evt) {
20         btnGreedyActionPerforme (evt);
21         d
22     }
23 });
24 btnAstar.setText("A*");
25 btnAstar.addActionListener (new java.awt.event.ActionListener () {
26     public void actionPerformed (java.awt.event.ActionEvent evt) {
27         btnAstarActionPerforme (evt);
28         d
29     }
30 });
31 resultAre .setColumn (20);
32 resultAre .setFont(new java.awt.Font("Nirmala UI Semilight", 1, 12)); // NOI18N
33 resultAre .setRows(5);
34 jScrollPane .setViewportView (resultAre );
35 2 w a
36 resultLabel.setFont(new java.awt.Font("Nirmala U , 1, 14)); // NOI18N
37 resultLabel.setText("RESULT");
38 I"
39 btnClear.setFont(new java.awt.Font("Nirmala U , 1, 14)); // NOI18N
40 btnClear.setText("CLEAR");
41 btnClear.addActionListener (new java.awt.event.ActionListener () {
42     public void actionPerformed (java.awt.event.ActionEvent evt) {
43         btnClearActionPerforme (evt);
44         d
45     }
46 });
47 jPanell.setBackground (new java.awt.Color(164, 207, 185));
d
```



The screenshot shows a Java code editor with a large block of Java code. The code is a complex Swing application setup, likely for a word ladder solver. It involves multiple panels, labels, and buttons. The code uses various layout managers like GroupLayout, BoxLayout, and GridLayout. It also includes numerous annotations such as `// NOI18N`, `Short.MAX_VALUE`, and `javax.swing.GroupLayout.PREFERRED_SIZE`. The code is heavily annotated with numbers (1 through 53) and letters (E, d, t, p) to indicate specific components or steps in the layout process.

```
1  jLabel1.setBackground (new java.awt.Color(164, 207, 185));
2  jLabel1.setFont(new java.awt.Font("Nirmala U , 1, 20)); // NOI18N
3  jLabel1.setForeground (new java.awt.Color(255, 255, 255));
4  jLabel1.setText("Word Ladder Solve ");
5  jLabel1.setR
6  javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
7  jPanel1.setLayout(jPanel1Layout);
8  jPanel1Layout.setHorizontalGroup(
9      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
10     .addGroup(jPanel1Layout.createSequentialGroup()
11         .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
12         .addGap(10, 10, 10)
13         .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
14     )
15     .addGroup(jPanel1Layout.createSequentialGroup()
16         .addGap(10, 10, 10)
17         .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
18         .addGap(10, 10, 10)
19         .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
20         .addGap(10, 10, 10)
21         .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
22     )
23 )
24 
25 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
26 getContentPane().setLayout(layout);
27 layout.setHorizontalGroup(
28     layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
29     .addGroup(layout.createSequentialGroup()
30         .addGap(10, 10, 10)
31         .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
32         .addGap(10, 10, 10)
33         .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
34         .addGap(10, 10, 10)
35         .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
36         .addGap(10, 10, 10)
37         .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
38         .addGap(10, 10, 10)
39         .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
40         .addGap(10, 10, 10)
41         .addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
42         .addGap(10, 10, 10)
43         .addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
44         .addGap(10, 10, 10)
45         .addComponent(jLabel10, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
46         .addGap(10, 10, 10)
47         .addComponent(jLabel11, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
48         .addGap(10, 10, 10)
49         .addComponent(jLabel12, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
50         .addGap(10, 10, 10)
51         .addComponent(jLabel13, javax.swing.GroupLayout.PREFERRED_SIZE, 232, javax.swing.GroupLayout.PREFERRED_SIZE)
52         .addGap(10, 10, 10)
53     )
54 )
55 
```

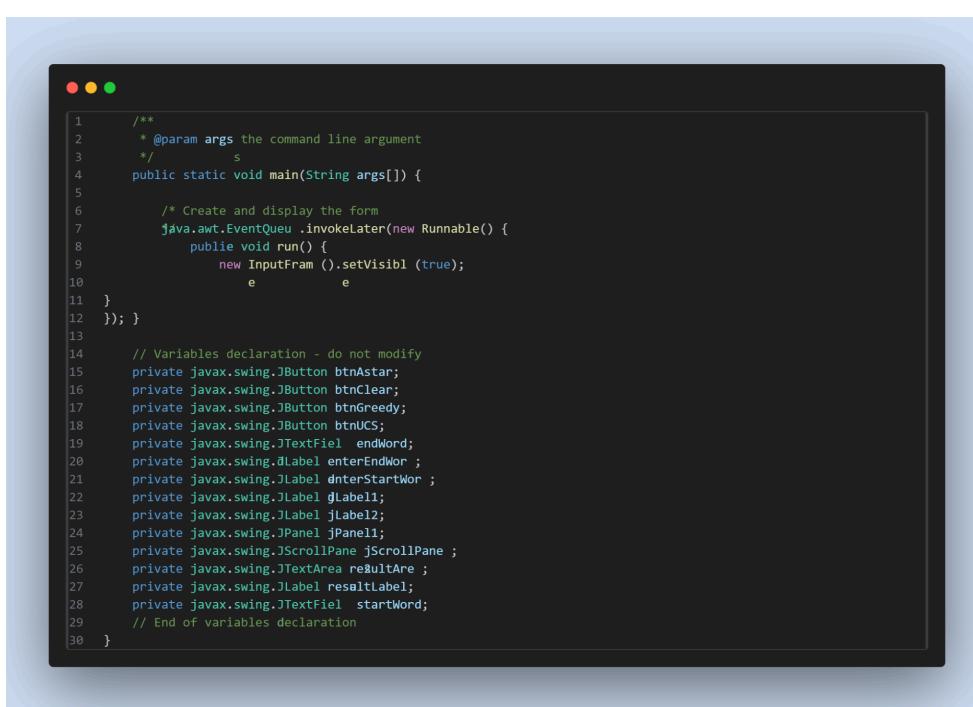


The screenshot shows a Java code editor with a scrollable code area. The code is written in Java and defines a window setup and validation logic for start and end words.

```
1 layout.setVerticalGrou ( 
2     layout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)
3     addGropp(layout.createSequentialGroup ())
4     . addComponen (jPanel1, javax.swing.GroupLayout.PREFERRED_SIZ , javax.swing.GroupLayout.
5     DEFAULT_SIZ , javax.swing.GroupLayout.PREFERRED_SIZ )           E
6     . addComponen (enterStartWor , javax.swing.GroupLayout.PREFERRED_SIZ , 20, javax.swing.
7     GroupLayout.PREFERRED_SIZ )      d                         E
8     E     addPreferredGa (javax.swing.LayoutStyle.ComponentPlacemen .RELATED)
9     . addComponen (startWord, javax.swing.GroupLayout.PREFERRED_SIZ , javax.swing.GroupLayout.
10    DEFAULT_SIZ , javax.swing.GroupLayout.PREFERRED_SIZ )           E
11   E     addGap(18, 18, 18)   E
12   . addComponen (enterEndWor )
13   . addPreferredGd (javax.swing.LayoutStyle.ComponentPlacemen .UNRELATED)
14   . addComponen (endWord, javax.swing.GroupLayout.PREFERRED_SIZ , javax.swing.GroupLayout.
15     DEFAULT_SIZ , javax.swing.GroupLayout.PREFERRED_SIZ )           E
16   E     addGap(26, 26, 26)   E
17   . addComponen (jLabel1)
18   . addGap(18, 18, 18)
19   . addGroup(layout.createParallelGroup (javax.swing.GroupLayout.Alignment.BASELINE)
20     . addComponen (btnUcs)
21     . addComponen (btnGreedy)
22     . addComponen (btnAstar))
23   . addPreferredGa (javax.swing.LayoutStyle.ComponentPlacemen .RELATED, 30, Short.MAX_VALUE)
24   . addComponen (resultLabel)          t
25   . addPreferredGd (javax.swing.LayoutStyle.ComponentPlacemen .UNRELATED)
26   . addComponen (jScrollPane , javax.swing.GroupLayout.PREFERRED_SIZ , 99, javax.swing.GroupLayout.
27     PREFERRERED_SIZ )      t      2                         E
28   E     addGap(18, 18, 18)
29   . addComponen (btnClear)
30   . addGap(23, 23, 23))
31   .
32   .
33
34 private void btnUcsActionPerforme (java.awt.event.ActionEvent evt) {
35     String inStartWord = startWord.getText().trim();
36     String inEndWord = endWord.getText().trim();
37
38     if (inEndWord.split(" ").length > 1 || inStartWord.split(" ").length > 1) {
39         resultAre .setText("Error! Start word and end word must contain only one wor   ");
40         return;      d."
41     }
42
43     // Validate start word and end wor
44     if (!inStartWord.matches("[a-zA-Z]+") || !inEndWord.matches("[a-zA-Z]+")) {
45         resultAre .setText(
46             "Error! Startword and end word must contain only alphabetical characters and cannot be empt   ");
47         return;
48     }
49
50     // Validate start word and end word length
51     if (inStartWord.length() != inEndWord.length()) {
52         resultAre .setText("Error! Start word and end word should have the same character leng   ");
53         return;      h"
54     }
}
```

```
1  List<String> wordList = new ArrayList<>();
2  // Populate wordList from a file or any other source
3  try (BufferedRead br = new BufferedRead (new FileRead ("dict.tx " )) {
4      String line;           r          r          t"
5      while ((line = br.readLine()) != null) {
6          wordList.add(line.trim());
7
8      } resultAre .setText(wordList.get(5));
9  } catch (FileNotFoundException e) {
10     e.printStackTrace ();
11     System.err.println("File not found:  + e.getMessage ());
12 } catch (IOException e){
13     e.printStackTrace ();
14     System.err.println("Error reading file:  + e.getMessage ());
15     e
16 }
17 if (!wordList.contains(inEndWord)) {
18     resultAre .setText("Error! End word is not in the word lis  ");
19     return;
20     t."
21 }
22 // Call the UCS algorithm
23 long startTime = System.nanoTime();
24 Pair<List<String>, Integer> result = UniformCostSearc .uniformCostSearc (inStartWord, inEndWord,
wordList);
25 long endTime = System.nanoTime();
26 long duration = (endTime - startTime) / 1000000;
27
28 // Display the result
29 if (!result.getKey().isEmpty()) {
30     resultAre .setText("Path from  + inStartWord + " to " + inEndWord + ": " + result.getKey() + "\n"
31     a      + "Number of nodes visited:  + result.getValue() + "\n"
32     + "Execution time:  + duration + " millisecond );
33 } else { "
34     resultAre .setText("No path found from  + inStartWord + " to " + inEndWord + "\n"
35     a      + "Number of nodes visited:  + result.getValue());
36
37 } }
38
39 private void btnAstarActionPerfome (java.awt.event.ActionEvent evt
) {
40     d
41     String inStartWord = startWord.getText().trim();
42     String inEndWord = endWord.getText().trim();
43
44     List<String> wordList = new ArrayList<>();
45     // Populate wordList from a file or any other source
46     try (BufferedRead br = new BufferedRead (new FileRead ("dict.tx " )) {
47         String line;           r          r          t"
48         while ((line = br.readLine()) != null) {
49             wordList.add(line.trim());
50
51         } resultAre .setText(wordList.get(5));
52     } catch (FileNotFoundException e) {
53         e.printStackTrace ();
54         System.err.println("File not found:  + e.getMessage ());
55     } catch (IOException e){
56         e.printStackTrace ();
57         System.err.println("Error reading file:  + e.getMessage ());
58     }
}
```

```
1 // Call the UCS algorithm
2 long startTime = System.nanoTime();
3 Pair<List<String>, Integer> result = AStarSearch.aStarSearch(inStartWord, inEndWord, wordList);
4 long endTime = System.nanoTime();
5 long duration = (endTime - startTime) / 1000000;
6
7 // Display the result
8 if (!result.getKey().isEmpty()) {
9     resultAre .setText("Path from " + inStartWord + " to " + inEndWord + ": " + result.getKey() + "\n"
10    + "Number of nodes visited: " + result.getValue() + "\n"
11    + "Execution time: " + duration + " millisecond ");
12 } else {
13     resultAre .setText("No path found from " + inStartWord + " to " + inEndWord + "\n"
14    + "Number of nodes visited: " + result.getValue());
15 }
16 }
17
18 private void btnGreedyActionPerfome (java.awt.event.ActionEvent evt
19 ) {
20     String inStartWord = startWord.getText().trim();
21     String inEndWord = endWord.getText().trim();
22
23     List<String> wordList = new ArrayList<>();
24     // Populate wordlist from a file or any other source
25     try (BufferedReader br = new BufferedReader (new FileReader ("dict.txt")) {
26         String line;           r           r           t"
27         while ((line = br.readLine()) != null) {
28             wordlist.add(line.trim());
29         }
30         resultAre .setText(wordList.get(5));
31     } catch (FileNotFoundException e) {
32         e.printStackTrace ();
33     } catch (IOException e){
34         e.printStackTrace ();
35         System.err.println("Error reading file: " + e.getMessage ());
36     }
37 }
38
39 // Call the UCS algorithm
40 long startTime = System.nanoTime();
41 Pair<List<String>, Integer> result = GreedyBestFirstSearc .greedyBestFirstSearc (inStartWord, inEndWord
42 , wordList);
43 long endTime = System.nanoTime();
44 long duration = (endTime - startTime) / 1000000;
45
46 // Display the result
47 if (!result.getKey().isEmpty()) {
48     resultAre .setText("Path from " + inStartWord + " to " + inEndWord + ": " + result.getKey() + "\n"
49    + "Number of nodes visited: " + result.getValue() + "\n"
50    + "Execution time: " + duration + " millisecond ");
51 } else {
52     resultAre .setText("No path found from " + inStartWord + " to " + inEndWord + "\n"
53    + "Number of nodes visited: " + result.getValue());
54 }
55
56 private void btnClearActionPerfome (java.awt.event.ActionEvent evt
57 ) {
58     startWord.setText("");
59     endWord.setText("");
60     resultAre .setText("");
61 }
```



```
1  /**
2   * @param args the command line argument
3   */
4  public static void main(String args[]) {
5
6      /* Create and display the form
7       * @param args
8       */
9      new InputFrame().setVisible(true);
10     e = e;
11 };
12 });
13
14 // Variables declaration - do not modify
15 private javax.swing.JButton btnAstar;
16 private javax.swing.JButton btnClear;
17 private javax.swing.JButton btnGreedy;
18 private javax.swing.JButton btnUCS;
19 private javax.swing.JTextField endWord;
20 private javax.swing.JLabel enterEndWord;
21 private javax.swing.JLabel enterStartWord;
22 private javax.swing.JLabel jLabel1;
23 private javax.swing.JLabel jLabel2;
24 private javax.swing.JPanel jPanel1;
25 private javax.swing.JScrollPane jScrollPane1;
26 private javax.swing.JTextArea resultArea;
27 private javax.swing.JLabel resultLabel;
28 private javax.swing.JTextField startWord;
29 // End of variables declaration
30 }
```

AStarSearch.java

```
 1 package GUI;
2 import java.util.ArrayList;
3 import java.util.Arrays;
4 import java.util.Comparato ;
5 import java.util.HashSet;
6 import java.util.List;
7 import java.util.PriorityQueue ;
8 import java.util.Set;
9
10 class AStarSearch {
11     // Calculates the Hamming distance between two word
12     public static int hammingDistance (String word1, String word2) {
13         int distance =0;
14         for (int i = 0; i < word1.length(); i++) {
15             if (word1.charAt(i) != word2.charAt(i)) {
16                 distance++;
17             }
18         }
19         return distance;
20     }
21
22     // Performs A* search algorithm to find the shortest path from beginWord to endWord
23     public static Pair<List<String>, Integer> aStarSearch(String beginWord, String endWord, List<String> wordList) {
24         Set<String> myset = new HashSet<>(wordList);
25         PriorityQueue <Node> pq = new PriorityQueue<>((Comparato .comparingIn (a -> (a.cost + hammingDistance (a.word, endWord)))));
26         pq.add(new Node(beginWord, 0,eArrays.asList(beginWord)));
27         Set<String> visited = new HashSet<>(); // track visited node
28         List<String> resultPath = new ArrayList<>();
29         int nodesVisited = 0;
30         d
31         while (!pq.isEmpty()) {
32             Node curr = pq.poll();
33             nodesVisited++;
34             d
35             if (curr.word.equals(endWord)) {
36                 resultPath = curr.path;
37                 return new Pair<>(resultPath , nodesVisited );
38             }
39         }
40         visited.add(curr.word);
41
42         // Check for all possible i-depth word
43         for (int i = 0; i < curr.word.length(); ++i) {
44             StringBuilder temp = new StringBuilder (curr.word);
45             for (char c = 'a'; c <= 'z'; ++c) {
46                 temp.setCharAt(i, c);
47                 String tempWord = temp.toString();
48                 if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
49                     continue; // Skip the same word, visited words, and words not in myse
50                 else
51                     List<String> newPath = new ArrayList<>(curr.path);
52                     newPath.add(tempWord);
53                     pq.add(new Node(tempWord, curr.cost + 1, newPath));
54             }
55         }
56     }
57
58     // If end word is not reachable
59     return new Pair<>(new ArrayList<>(), nodesVisited );
60 }
61 }
```

UniformCostSearch.java

```
1 package GUI;
2
3 /*
4  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
5  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
6  */
7
8
9 import java.util.ArrayList;
10 import java.util.Arrays;
11 import java.util.Comparato ;
12 import java.util.HashSet;
13 import java.util.List;
14 import java.util.PriorityQueu ;
15 import java.util.Set;
16
17 /**
18 *
19 * @author Auralea A S
20 */
21 class UniformCostSearc {
22     h public static Pair<List<String>, Integer> uniformCostSearc (String beginWord, String endWord, List<String> wordlist) {
23         Set<String> myset = new HashSet<>();
24         boolean isPresent = false;
25
26         // Insert all words from Dict in myse
27         for (String word : wordlist) {
28             if (endWord.equals(word))
29                 isPresent = true;
30             myset.add(word); // Insert word in Dic
31             t
32         } if (!isPresent) // endWord is not present in Dic
33             return new Pair<>(new ArrayList<>(), 0);
34
35         PriorityQueu <Node> pq = new PriorityQueu <>(Comparato .comparingIn (a -> a.cost));
36         pq.add(new Node(beginWord, 0,eArrays.asList(beginWord))); // Start nod
37         Set<String> visited = new HashSet<>(); // Track visited node
38         List<String> resultPat = new ArrayList<>();
39         int nodesVisite = 0;
40         d
41         while (!pq.isEmpty()) {
42             Node curr = pq.poll();
43             nodesVisite++;
44             d
45             if (curr.word.equals(endWord)) {
46                 resultPat = curr.path;
47                 return new Pair<>(resultPat , nodesVisite );
48                 h           d
49         }
50         visited.add(curr.word);
51
52         // Check for all possible 1-depth word
53         for (int i = 0; i < curr.word.length(); ++i) {
54             StringBuilde temp = new StringBuilde (curr.word);
55             for (char c = 'a'; c <= 'e'; ++c) {
56                 temp.setCharAt(i, c);
57                 String tempWord = temp.toString();
58                 if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
59                     continue; // Skip the same word, visited words, and words not in myse
60                 t
61                 List<String> newPath = new ArrayList<>(curr.path);
62                 newPath.add(tempWord);
63                 pq.add(new Node(tempWord, curr.cost + 1, newPath));
64
65         }
66     }
67 }
68     // If end word is not reachabl
69     return new Pair<>(new ArrayList<>(), nodesVisite );
70     d
71 }
```

GreedyBestFirstSearch.java

```
1 package GUI;
2
3
4 import java.util.ArrayList;
5 import java.util.Arrays;
6 import java.util.Comparato ;
7 import java.util.HashSet;
8 import java.util.List;
9 import java.util.PriorityQueu ;
10 import java.util.Set;
11
12 /*
13  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
14  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
15  */
16
17 /**
18 *
19 * @author Auralea A S
20 */
21 class GreedyBestFirstSearc {
22     //Function to calculate the Hamming distance between two word
23     public static int hammingDistanc (String word1, String word2) {
24         int distance =0;
25         for (int i = 0; i < word1.length(); i++) {
26             if (word1.charAt(i) != word2.charAt(i)) {
27                 distance++;
28
29             }
30         }
31         return distance;
32     }
33     public static Pair<List<String>, Integer> greedyBestFirstSearc (String beginWord, String endWord, List<String
> wordlist) {
34         Set<String> myset = new HashSet<>(wordlist);
35         PriorityQueu <Node> pq = new PriorityQueu <>((Comparato .comparingIn (a -> hammingDistanc (a.word, endWord
)))); e           e           r           t           e
36         pq.add(new Node(beginWord, 0, Arrays.asList(beginWord))); // Start nod
37         Set<String> visited = new HashSet<>(); // Track visited node
38         List<String> resultPat  = new ArrayList<>();
39         int nodesVisite = 0;
40         d
41         while (!pq.isEmpty()) {
42             Node curr = pq.poll();
43             pq.clear();
44             nodesVisite++;
45             d
46             if (curr.word.equals(endWord)) {
47                 resultPat = curr.path;
48                 return new Pair<>(resultPat , nodesVisite );
49             }
50         }
51         visited.add(curr.word);
52
53         // Check for all possible 1-depth word
54         for (int i = 0; i < curr.word.length(); ++i) {
55             StringBuilde  temp = new StringBuilde (curr.word);
56             for (char c = 'a'; c <= 'e'; ++c) {
57                 temp.setCharAt(i, c);
58                 String tempWord = temp.toString();
59                 if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
60                     continue; // Skip the same word, visited words, and words not in myse
61
62                 List<String> newPath = new ArrayList<>(curr.path);
63                 newPath.add(tempWord);
64                 pq.add(new Node(tempWord, curr.cost + 1, newPath));
65
66             }
67         }
68
69         // If end word is not reachabl
70         return new Pair<>(new ArrayList<>(), nodesVisite );
71     }
72 }
```

Pair.java

```
 1 package GUI;
 2 /*
 3  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 5  */
 6
 7
 8 /**
 9  *
10  * @author Auralea A S
11  */
12 class Pair<K, V> {
13     private K key;
14     private V value;
15
16     public Pair(K key, V value) {
17         this.key = key;
18         this.value = value;
19     }
20
21     public K getKey() {
22         return key;
23     }
24
25     public V getValue() {
26         return value;
27     }
28 }
29 }
```

Node.java

```
 1 package GUI;
 2 /*
 3  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 4  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 5  */
 6
 7
 8 /**
 9  *
10  * @author Auralea A S
11  */
12 import java.util.ArrayList;
13 import java.util.List;
14
15 class Node {
16     String word;
17     int cost;
18     List<String> path;
19
20     public Node(String word, int cost, List<String> path) {
21         this.word = word;
22         this.cost = cost;
23         this.path = new ArrayList<>(path);
24     }
25 }
26 }
```

3.2 Dengan *Command Line Interface (CLI)*

Main.java

```
1 import java.io.BufferedReader ;
2 import java.io.FileReader ;
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.Arrays;
6 import java.util.Comparator;
7 import java.util.HashSet;
8 import java.util.List;
9 import java.util.PriorityQueue ;
10 import java.util.Scanner;
11 import java.util.Set;
12
13 public class Main {
14
15     // ----- UCS -----
16
17     public static Pair<List<String>, Integer> uniformCostSearch (String beginWord, String endWord, List<String> wordList) {
18         Set<String> myset = new HashSet<>();
19         boolean isPresent = false;
20
21         // Insert all words from Dict in myse
22         for (String word : wordList) {
23             if (endWord.equals(word))
24                 isPresent = true;
25             myset.add(word); // Insert word in Dic
26         }
27         if (!isPresent) // endWord is not present in Dic
28             return new Pair<>(new ArrayList<>(), 0);
29
30         PriorityQueue <Node> pq = new PriorityQueue <>(Comparador .comparingIn (a -> a.cost));
31         pq.add(new Node(beginWord, 0, earrays.asList(beginWord))); // Start nod
32         Set<String> visited = new HashSet<>(); // Track visited node
33         List<String> resultPat = new ArrayList<>();
34         int nodesVisite = 0;
35         d
36         while (!pq.isEmpty()) {
37             Node curr = pq.poll();
38             nodesVisite++;
39             d
40             if (curr.word.equals(endWord)) {
41                 resultPat = curr.path;
42                 return new Pair<>(resultPat , nodesVisite );
43             }
44         }
45         visited.add(curr.word);
46
47         // Check for all possible 1-depth word
48         for (int i = 0; i < curr.word.length(); ++i) {
49             StringBuilder temp = new StringBuilder (curr.word);
50             for (char c = 'a'; c <= 'z'; ++c) {
51                 temp.setCharAt(i, c);
52                 String tempWord = temp.toString();
53                 if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
54                     continue; // Skip the same word, visited words, and words not in myse
55                 t
56                 List<String> newPath = new ArrayList<>(curr.path);
57                 newPath.add(tempWord);
58                 pq.add(new Node(tempWord, curr.cost + 1, newPath));
59
60             }
61         }
62
63         // If end word is not reachabl
64         return new Pair<>(new ArrayList<>(), nodesVisite );
65     }
}
```



```
1 // ----- GREEDY BFS -----
2 -
3 // Function to calculate the Hamming distance between two word
4 public static int hammingDistance (String word1, String word2) {
5     int distance =0;
6     for (int i = 0; i < word1.length(); i++) {
7         if (word1.charAt(i) != word2.charAt(i)) {
8             distance++;
9         }
10    }
11    return distance;
12 }
13
14 public static Pair<List<String>, Integer> greedyBestFirstSearch (String beginWord, String endWord, List<String> > wordList) {
15     Set<String> myset = new HashSet<>(wordList);
16     PriorityQueu <Node> pq = new PriorityQueu <> (Comparato .comparingIn (a -> hammingDistanc (a.word, endWord
17 ))); e e r t e
18     pq.add(new Node(beginWord, 0, Arrays.asList(beginWord))); // Start nod
19     Set<String> visited = new HashSet<>(); // Track visited node
20     List<String> resultPat = new ArrayList<>();
21     int nodesVisite = 0;
22     d
23     while (!pq.isEmpty()) {
24         Node curr = pq.poll();
25         nodesVisite++;
26         d
27         if (curr.word.equals(endWord)) {
28             resultPat = curr.path;
29             return new Pair<>(resultPat , nodesVisite );
30         }
31         visited.add(curr.word);
32
33         // Check for all possible 1-depth word
34         for (int i = 0; i < curr.word.length(); ++i) {
35             StringBuilde temp = new StringBuilde (curr.word);
36             for (char c = 'a'; c <= 'z'; ++c) {
37                 temp.setCharAt(i, c);
38                 String tempWord = temp.toString();
39                 if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
40                     continue; // Skip the same word, visited words, and words not in myse
41                 t
42                 List<String> newPath = new ArrayList<>(curr.path);
43                 newPath.add(tempWord);
44                 pq.add(new Node(tempWord, curr.cost + 1, newPath));
45
46     }
47 } }
48
49 // If end word is not reachabl
50 return new Pair<>(new ArrayList<>(), nodesVisite );
51 }
52 }
```

```
1 // ----- A STAR -----
2 -
3 public static Pair<List<String>, Integer> aStarSearch(String beginWord, String endWord, List<String> wordList
4 ) {
5     Set<String> myset = new HashSet<>(wordList);
6     PriorityQueu <Node> pq = new PriorityQueu <>(Comparato .comparingIn (a -> (a.cost + hammingDistanc (a.word
7 , endWord))));                                e          r          t          e
8     pq.add(new Node(beginWord, 0, Arrays.asList(beginWord))); // Start nod
9     Set<String> visited = new HashSet<>(); // Track visited node
10    List<String> resultPat  = new ArrayList<>();
11    int nodesVisite  = 0;
12    d
13    while (!pq.isEmpty()) {
14        Node curr = pq.poll();
15        nodesVisite++;
16        if (curr.word.equals(endWord)) {
17            resultPat  = curr.path;
18            return new Pair<>(resultPat , nodesVisite );
19        }
20        visited.add(curr.word);
21
22        // Check for all possible 1-depth word
23        for (int i = 0; i < curr.word.length(); ++i) {
24            StringBuilder temp = new StringBuilder (curr.word);
25            for (char c = 'a'; c <= 'e'; ++c) {
26                temp.setCharAt(i, c);
27                String tempWord = temp.toString();
28                if (tempWord.equals(curr.word) || visited.contains(tempWord) || !myset.contains(tempWord))
29                    continue; // Skip the same word, visited words, and words not in myse
30                t
31                List<String> newPath = new ArrayList<>(curr.path);
32                newPath.add(tempWord);
33                pq.add(new Node(tempWord, curr.cost + 1, newPath));
34
35    }
36 } }
37
38 // If end word is not reachabl
39 return new Pair<>(new ArrayList<>(), nodesVisite );
40 }
```

```
1 // -----
2 // ----- MAIN -----
3 public static void main(String[] args) {
4     Scanner scanner = new Scanner(System.in);
5
6     // Parse dict.tx
7     List<String> wordList = new ArrayList<>();
8     try (BufferedReader br = new BufferedReader(new FileReader("dict.tx"))) {
9         String line;           r           r           t"
10        while ((line = br.readLine()) != null) {
11            wordList.add(line.trim());
12        }
13    } catch (IOException e) {
14        e.printStackTrace();
15    } finally {
16        e
17
18        // Enter begin word
19        System.out.print("Enter the begin word: ");
20        String beginWord = scanner.nextLine();
21
22        if (beginWord.split(" ").length > 1) {
23            System.out.println("Error: The begin word must contain only one word");
24            scanner.close(); d."
25            return;
26        }
27
28        if (!beginWord.chars().allMatch(Character::isLetter)) {
29            System.out.println("Error: The begin word must contain only alphabetical character");
30            scanner.close(); s."
31            return;
32        }
33
34        // Enter end word
35        System.out.print("Enter the end word: ");
36        String endWord = scanner.nextLine();
37
38        if (endWord.split(" ").length > 1) {
39            System.out.println("Error: The end word must contain only one word");
40            scanner.close(); d."
41            return;
42        }
43
44        if (!endWord.chars().allMatch(Character::isLetter)) {
45            System.out.println("Error: The end word must contain only alphabetical character");
46            scanner.close(); s."
47            return;
48        }
49
50        if (!wordList.contains(endWord)) {
51            System.out.println("Error: The end word is not present in the word list");
52            scanner.close(); t."
53            return;
54        }
55
56        // Check begin word and end word length
57        if (beginWord.length() != endWord.length()) {
58            System.out.println("Error: Begin word and end word must have the same length");
59            scanner.close(); h."
60            return;
61        }
}
```

```
1 System.out.println("\n\n");
2 System.out.println(" _____ CHOOSE ALGORITHM _____ ");
3 System.out.println("| 1. Create using Uniform Cost Search   ");
4 System.out.println("| 2. Create using Greedy Best-first Search   ");
5 System.out.println("| 3. Create using A*   ");
6 System.out.println("|");
7 System.out.println("|");
8 System.out.println("-----");
9 System.out.print("> Enter your option (1 / 2 / 3): ");
10 String userInput = scanner.nextLine();
11
12
13 if(userInput.equals("1")){
14
15     long startTime = System.nanoTime();
16     Pair<List<String>, Integer> result = uniformCostSearch (beginWord, endWord, wordList);
17     long endTime = System.nanoTime();      h
18     long duration = (endTime - startTime) / 1000000; // Convert to microsecond
19                                         s
20
21     if (!result.getKey().isEmpty()) {
22         System.out.print("Path from " + beginWord + " to " + endWord + ": ");
23         for (String word : result.getKey())
24             System.out.print(word + " ");
25         System.out.println();
26         System.out.println("Number of nodes visited: " + result.getValue());
27         System.out.println("Execution time: " + duration + " milisecond ");
28     } else {
29         System.out.println("No path found from " + beginWord + " to " + endWord);
30         System.out.println("Number of nodes visited: " + result.getValue());
31         System.out.println("Execution time: " + duration + " milisecond ");
32                                         s
33     }
34     scanner.close();
35
36 }else if(userInput.equals("2")){
37
38     long startTime = System.nanoTime();
39     Pair<List<String>, Integer> result = uniformCostSearch (beginWord, endWord, wordList);
40     long endTime = System.nanoTime();      h
41     long duration = (endTime - startTime) / 1000000; // Convert to microsecond
42                                         s
43     if (!result.getKey().isEmpty()) {
44         System.out.print("Path from " + beginWord + " to " + endWord + ": ");
45         for (String word : result.getKey())
46             System.out.print(word + " ");
47         System.out.println();
48         System.out.println("Number of nodes visited: " + result.getValue());
49         System.out.println("Execution time: " + duration + " milisecond ");
50     } else {
51         System.out.println("No path found from " + beginWord + " to " + endWord);
52         System.out.println("Number of nodes visited: " + result.getValue());
53         System.out.println("Execution time: " + duration + " milisecond ");
54                                         s
55     }
56     scanner.close();
57 }
```

```
1     }else{
2
3         long startTime = System.nanoTime();
4         Pair<List<String>, Integer> result = uniformCostSearch (beginWord, endWord, wordList);
5         long endTime = System.nanoTime();    h
6         long duration = (endTime - startTime) / 1000000; // Convert to microsecond
7                                         s
8
9         if (!result.getKey().isEmpty()) {
10             System.out.print("Path from " + beginWord + " to " + endWord + ": ");
11             for (String word : result.getValue())
12                 System.out.print(word + " ");
13             System.out.println();
14             System.out.println("Number of nodes visited: " + result.getValue());
15             System.out.println("Execution time: " + duration + " millisecond ");
16         } else {
17             System.out.println("No path found from " + beginWord + " to " + endWord);
18             System.out.println("Number of nodes visited: " + result.getValue());
19             System.out.println("Execution time: " + duration + " millisecond ");
20                                         s
21         }
22         scanner.close();
23     }
24 }
25 }
26 }
```

Node.java

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 class Node {
5     String word;
6     int cost;
7     List<String> path;
8
9     public Node(String word, int cost, List<String> path) {
10        this.word = word;
11        this.cost = cost;
12        this.path = new ArrayList<>(path);
13    }
14 }
```

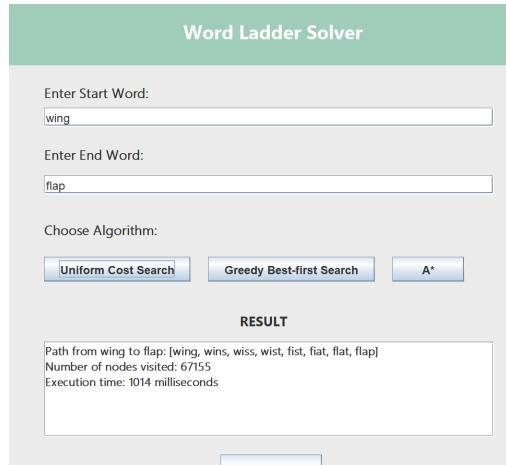
Pair.java

```
1 class Pair<K, V> {
2     private K key;
3     private V value;
4
5     public Pair(K key, V value) {
6         this.key = key;
7         this.value = value;
8     }
9
10    public K getKey() {
11        return key;
12    }
13
14    public V getValue() {
15        return value;
16    }
17 }
```

BAB IV

HASIL UJI COBA

4.1 Hasil Uji Coba Algoritma *Uniform Cost Search* (UCS)

No	Input dan Output
1	 <p>The screenshot shows the 'Word Ladder Solver' application interface. It has input fields for 'Enter Start Word' (hit) and 'Enter End Word' (cog). Below these are three algorithm selection buttons: 'Uniform Cost Search' (selected), 'Greedy Best-first Search', and 'A*'. A 'RESULT' section displays the path from hit to cog: [hit, hot, hog, cog], the number of nodes visited (1660), and the execution time (118 milliseconds). A 'CLEAR' button is at the bottom.</p>
2	 <p>The screenshot shows the 'Word Ladder Solver' application interface. It has input fields for 'Enter Start Word' (wing) and 'Enter End Word' (flap). Below these are three algorithm selection buttons: 'Uniform Cost Search' (selected), 'Greedy Best-first Search', and 'A*'. A 'RESULT' section displays the path from wing to flap: [wing, wins, wiss, wist, fist, fiat, flat, flap], the number of nodes visited (67155), and the execution time (1014 milliseconds). A 'CLEAR' button is at the bottom.</p>

3	<div style="background-color: #c6e2ff; padding: 10px; border-radius: 5px;"><h3 style="margin: 0;">Word Ladder Solver</h3><p>Enter Start Word: <input type="text" value="duty"/></p><p>Enter End Word: <input type="text" value="free"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9;"><p>Path from duty to free: [duty, doty, dole, done, tone, tyne, tyee, tree, free] Number of nodes visited: 52831 Execution time: 693 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>
4	<div style="background-color: #c6e2ff; padding: 10px; border-radius: 5px;"><h3 style="margin: 0;">Word Ladder Solver</h3><p>Enter Start Word: <input type="text" value="rabbit"/></p><p>Enter End Word: <input type="text" value="bitter"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9;"><p>NO PATH FOUND from rabbit to bitter !! Number of nodes visited: 13 Execution time: 34 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>

5	<div style="background-color: #e0f2e0; padding: 10px; width: 100%;"><h3 style="text-align: center;">Word Ladder Solver</h3><p>Enter Start Word: <input type="text" value="cool"/></p><p>Enter End Word: <input type="text" value="chic"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9; margin-top: 5px;"><p>Path from cool to chic: [cool, coop, chop, chip, chic] Number of nodes visited: 1779 Execution time: 44 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>
6	<div style="background-color: #e0f2e0; padding: 10px; width: 100%;"><h3 style="text-align: center;">Word Ladder Solver</h3><p>Enter Start Word: <input type="text" value="tired"/></p><p>Enter End Word: <input type="text" value="tires"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9; margin-top: 5px;"><p>Path from tired to tires: [tired, tires] Number of nodes visited: 3 Execution time: 16 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>

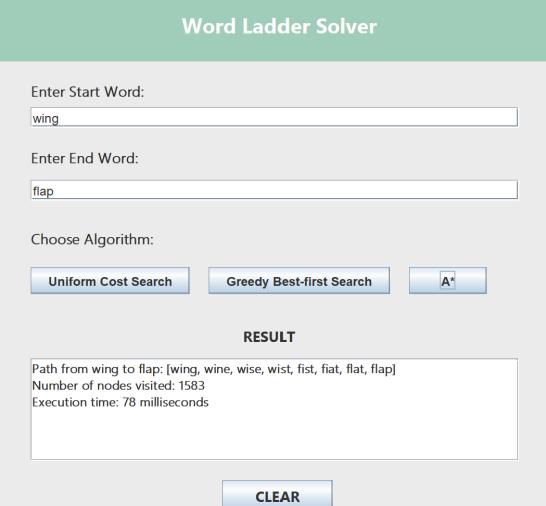
4.2 Hasil Uji Coba Algoritma *Greedy Best-first Search*

No	Input dan Output
1	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="hit"/></p> <p>Enter End Word: <input type="text" value="cog"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px;"> Path from hit to cog: [hit, hot, cot, cog] Number of nodes visited: 4 Execution time: 9 milliseconds </div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>
2	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="wing"/></p> <p>Enter End Word: <input type="text" value="flap"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px;"> Path from wing to flap: [wing, ding, king, ling, ping, ring, sing, ting, zing, zinc, zins, fin] Number of nodes visited: 46 Execution time: 24 milliseconds </div> <p style="text-align: center;">◀ ▶</p> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>

3	<div style="background-color: #c6e2ff; padding: 5px; text-align: center;"><h3>Word Ladder Solver</h3></div> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9; margin-bottom: 10px;"><p>Enter Start Word: <input type="text" value="duty"/></p><p>Enter End Word: <input type="text" value="free"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid #ccc; padding: 5px; background-color: #fff; margin-bottom: 5px;"><p>NO PATH FOUND from duty to free !! Number of nodes visited: 146 Execution time: 33 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>
4	<div style="background-color: #c6e2ff; padding: 5px; text-align: center;"><h3>Word Ladder Solver</h3></div> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9; margin-bottom: 10px;"><p>Enter Start Word: <input type="text" value="rabbit"/></p><p>Enter End Word: <input type="text" value="bitter"/></p><p>Choose Algorithm:</p><p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p><p style="text-align: center;">RESULT</p><div style="border: 1px solid #ccc; padding: 5px; background-color: #fff; margin-bottom: 5px;"><p>NO PATH FOUND from rabbit to bitter !! Number of nodes visited: 2 Execution time: 15 milliseconds</p></div><p style="text-align: center;"><input type="button" value="CLEAR"/></p></div>

5	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="cool"/></p> <p>Enter End Word: <input type="text" value="chic"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input checked="" type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px; height: 60px; overflow: auto;"><p>NO PATH FOUND from cool to chic !! Number of nodes visited: 32 Execution time: 13 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>
6	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="tired"/></p> <p>Enter End Word: <input type="text" value="tires"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input checked="" type="button" value="Greedy Best-first Search"/> <input type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px; height: 60px; overflow: auto;"><p>Path from tired to tires: [tired, tires] Number of nodes visited: 2 Execution time: 8 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>

4.3 Hasil Uji Coba Algoritma A*

No	Input dan Output
1	 <p>Word Ladder Solver</p> <p>Enter Start Word: hit</p> <p>Enter End Word: cog</p> <p>Choose Algorithm:</p> <p>Uniform Cost Search Greedy Best-first Search A*</p> <p>RESULT</p> <p>Path from hit to cog: [hit, hot, cot, cog] Number of nodes visited: 5 Execution time: 12 milliseconds</p> <p>CLEAR</p>
2	 <p>Word Ladder Solver</p> <p>Enter Start Word: wing</p> <p>Enter End Word: flap</p> <p>Choose Algorithm:</p> <p>Uniform Cost Search Greedy Best-first Search A*</p> <p>RESULT</p> <p>Path from wing to flap: [wing, wine, wise, wist, fist, fiat, flat, flap] Number of nodes visited: 1583 Execution time: 78 milliseconds</p> <p>CLEAR</p>

3	<div style="background-color: #c6e2ff; padding: 5px; text-align: center;"><h3>Word Ladder Solver</h3></div> <p>Enter Start Word: <input type="text" value="duty"/></p> <p>Enter End Word: <input type="text" value="free"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A<sup>*</sup>"/></p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><p style="text-align: center;">RESULT</p><p>Path from duty to free: [duty, doty, dote, dore, tore, tyre, tyee, tree, free] Number of nodes visited: 1923 Execution time: 62 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>
4	<div style="background-color: #c6e2ff; padding: 5px; text-align: center;"><h3>Word Ladder Solver</h3></div> <p>Enter Start Word: <input type="text" value="rabbit"/></p> <p>Enter End Word: <input type="text" value="bitter"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input type="button" value="A<sup>*</sup>"/></p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><p style="text-align: center;">RESULT</p><p>NO PATH FOUND from rabbit to bitter !! Number of nodes visited: 13 Execution time: 24 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>

5	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="cool"/></p> <p>Enter End Word: <input type="text" value="chic"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input checked="" type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px; width: fit-content;"><p>Path from cool to chic: [cool, coon, chon, chin, chic] Number of nodes visited: 18 Execution time: 12 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>
6	<p style="text-align: center;">Word Ladder Solver</p> <p>Enter Start Word: <input type="text" value="tired"/></p> <p>Enter End Word: <input type="text" value="tires"/></p> <p>Choose Algorithm:</p> <p style="text-align: center;"><input type="button" value="Uniform Cost Search"/> <input type="button" value="Greedy Best-first Search"/> <input checked="" type="button" value="A*"/></p> <p style="text-align: center;">RESULT</p> <div style="border: 1px solid black; padding: 5px; width: fit-content;"><p>Path from tired to tires: [tired, tires] Number of nodes visited: 2 Execution time: 7 milliseconds</p></div> <p style="text-align: center;"><input type="button" value="CLEAR"/></p>

BAB V

ANALISIS PERBANDINGAN SOLUSI

5.1 Berdasarkan Optimalitas

1) UCS (Uniform-Cost Search)

UCS menjamin menemukan solusi optimal dalam konteks ruang pencarian dengan biaya langkah yang seragam. Dalam Word Ladder, UCS akan menemukan jalur terpendek antara dua kata, tetapi tidak mempertimbangkan heuristik.

2) Greedy BFS (Greedy Best-First Search)

Greedy BFS cenderung menemukan solusi yang cepat tetapi tidak selalu optimal. Algoritma ini hanya mempertimbangkan heuristik untuk menentukan prioritas ekspansi, yang bisa mengakibatkan melewatkannya jalur yang lebih efisien.

3) A* Search

A* Search dirancang untuk mencari solusi optimal dengan mempertimbangkan biaya langkah dan estimasi heuristik. Dengan menggunakan heuristik yang admissible, A* dapat menemukan jalur terpendek dengan meminimalkan jumlah langkah.

5.2 Berdasarkan Waktu Eksekusi

1) UCS (Uniform-Cost Search)

Waktu eksekusi UCS tergantung pada kompleksitas ruang pencarian. Dalam beberapa kasus, ini bisa sangat lambat karena UCS mengeksplorasi semua kemungkinan langkah tanpa mempertimbangkan heuristik.

2) Greedy BFS (Greedy Best-First Search)

Greedy BFS cenderung lebih cepat daripada UCS karena fokus pada heuristik yang mengarahkan eksplorasi ke arah tujuan. Namun, kecepatannya bisa berkurang karena kemungkinan melewatkannya solusi optimal.

3) A* Search

Waktu eksekusi A* tergantung pada kualitas heuristik dan kompleksitas ruang pencarian. Dalam banyak kasus, A* dapat

menemukan solusi optimal dengan waktu yang lebih cepat dibandingkan UCS atau Greedy BFS karena memadukan informasi tentang biaya langkah dan estimasi heuristik.

5.3 Berdasarkan Penggunaan Memori

4) UCS (Uniform-Cost Search)

UCS cenderung menggunakan banyak memori karena harus menyimpan semua node yang dieksplorasi dalam priority queue.

5) Greedy BFS (Greedy Best-First Search)

Greedy BFS juga dapat menggunakan banyak memori karena perlu menyimpan semua node yang dieksplorasi untuk menentukan prioritas ekspansi.

6) A* Search

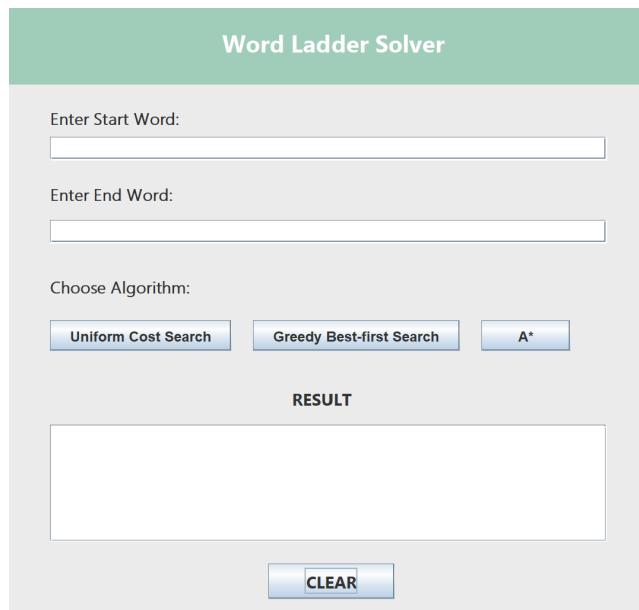
A* menggunakan memori dalam cara yang serupa dengan Greedy BFS tetapi dengan tambahan estimasi heuristik. Ini bisa membuatnya membutuhkan lebih sedikit memori dibandingkan UCS karena bisa melewati beberapa node yang kurang menjanjikan.

Dalam banyak kasus, A* adalah pilihan yang lebih baik karena menawarkan keseimbangan antara optimalitas, waktu eksekusi, dan penggunaan memori. Namun, keefektifan relatif dari setiap algoritma akan tergantung pada karakteristik spesifik dari masalah yang dihadapi.

BAB VI

PENJELASAN BONUS

6.1 *Graphical User Interface (GUI)*



Kode detail untuk GUI diatas dapat dilihat pada Bab 3 Subbab 3.1. Pada GUI diatas terdapat beberapa komponen penting yaitu:

1. InputFrame

Kelas ini merupakan frame utama dari aplikasi GUI. Di dalamnya terdapat elemen-elemen antarmuka pengguna seperti label, text field, tombol, dan area teks. (Untuk kode terdapat pada Bab 3 Subbab 3.1)

2. Elemen Antarmuka Pengguna

- `enterStartWord`: Label untuk meminta pengguna memasukkan kata awal.
- `startWord`: Text field tempat pengguna memasukkan kata awal.
- `enterEndWord`: Label untuk meminta pengguna memasukkan kata akhir.
- `endWord`: Text field tempat pengguna memasukkan kata akhir.
- `jLabel1`: Label yang menunjukkan pilihan algoritma.
- `btnUCS`: Tombol untuk memilih algoritma Uniform Cost Search.
- `btnGreedy`: Tombol untuk memilih algoritma Greedy Best-first Search.
- `btnAstar`: Tombol untuk memilih algoritma A*.
- `resultArea`: Area teks untuk menampilkan hasil pencarian jalur.
- `resultLabel`: Label yang menunjukkan bagian hasil.
- `btnClear`: Tombol untuk menghapus masukan dan hasil yang ditampilkan.
- `jPanel1`: Panel untuk menampilkan judul aplikasi.

3. Metode Aksi

Terdapat metode aksi untuk tombol-tombol yang ada di GUI. Ketika tombol ditekan, metode aksi akan dipanggil untuk memproses masukan pengguna dan menjalankan algoritma pencarian yang sesuai.

- btnUCSActionPerformed : Metode untuk menjalankan algoritma UCS setelah memvalidasi input pengguna.
- btnAstarActionPerformed : Metode untuk menjalankan algoritma A* setelah memeriksa masukan pengguna.
- btnGreedyActionPerformed : Metode untuk menjalankan algoritma Greedy Best-first Search setelah memastikan masukan pengguna valid.
- btnClearActionPerformed : Metode untuk menghapus masukan pengguna dan hasil pencarian yang ditampilkan.

4. Metode Utama

Metode `main` merupakan metode utama yang akan membuat dan menampilkan frame InputFrame ketika aplikasi dijalankan.

Dengan menggunakan GUI ini, pengguna dapat memasukkan kata awal dan kata akhir, memilih algoritma yang ingin digunakan, dan melihat hasil pencarian jalur antara kedua kata tersebut.

BAB VII

PENUTUP

5.4 Kesimpulan

Dalam penelitian ini, penulis menerapkan tiga algoritma yang berbeda, yaitu UCS, Greedy Best First Search, dan A*, untuk menyelesaikan permainan Word Ladder. Penulis menemukan bahwa setiap algoritma memiliki kelebihan dan kelemahan masing-masing dalam hal optimalitas, waktu eksekusi, dan penggunaan memori.

UCS, meskipun memastikan solusi optimal, mungkin memerlukan waktu yang lebih lama untuk menemukan jalur terpendek karena mempertimbangkan semua kemungkinan langkah secara berurutan tanpa memperhitungkan informasi tentang tujuan. Di sisi lain, Greedy Best First Search dan A* menggunakan heuristik untuk memperkirakan biaya sisa menuju tujuan. Namun, Greedy Best First Search cenderung mengorbankan optimalitas demi kecepatan karena hanya mempertimbangkan heuristik dalam menentukan prioritas ekspansi. Sementara itu, A* memadukan biaya langkah dengan heuristik untuk menentukan prioritas ekspansi, yang memungkinkan pencarian jalur yang lebih efisien.

5.5 Saran

Berdasarkan hasil penelitian ini, ada beberapa saran untuk penelitian selanjutnya:

1. Memperluas kajian dengan menguji algoritma-algoritma lain yang mungkin lebih efektif dalam menyelesaikan permainan Word Ladder.
2. Meneliti penggunaan heuristik yang lebih canggih untuk meningkatkan kinerja algoritma A*.
3. Mengoptimalkan implementasi algoritma untuk mengurangi waktu eksekusi dan penggunaan memori.
4. Membandingkan kinerja algoritma dengan ukuran masukan yang berbeda untuk mendapatkan pemahaman yang lebih mendalam tentang kekuatan dan kelemahan masing-masing pendekatan.

DAFTAR PUSTAKA

Munir, R. (2024). *Strategi Algoritma*. Retrieved from Homepage Rinaldi

Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/>

Baeldung. (2024). *Obtaining the Path in the Uniform Cost Search Algorithm*: [Obtaining the Path in the Uniform Cost Search Algorithm | Baeldung on Computer Science](#)

GeeksforGeeks. (2024). *A* search algorithm*: [A* Search Algorithm - GeeksforGeeks](#)

GeeksforGeeks. (2024). *Greedy best first search algorithm*: [Greedy Best first search algorithm - GeeksforGeeks](#)

GeeksforGeeks. (2024). *Concepts of hamming distance*: [Concepts of hamming distance - GeeksforGeeks](#)

GeeksforGeeks. (2024). *Introduction to Java Swing*: [Introduction to Java Swing - GeeksforGeeks](#)

LAMPIRAN

GitHub Repository (Latest Release)

[auraleaas/Tucil3_13522148: Tugas Kecil 3 IF2211 Strategi Algoritma Semester II tahun 2023/2024 \(github.com\)](https://github.com/auraleaas/Tucil3_13522148)

Tabel Spesifikasi

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