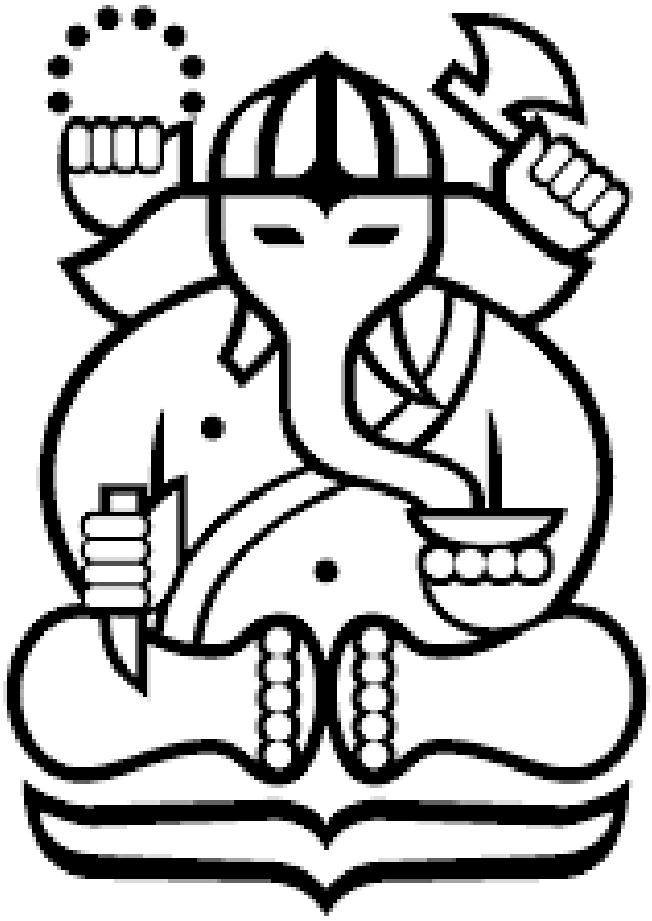
Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

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



Disusun oleh:

Dzaky Satrio Nugroho 13522059

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

2024

Daftar Isi

Bab 1: Analisis dan Implementasi

Word ladder (juga dikenal sebagai *Doublets, word-links, change-the-word puzzles, paragrams, laddergrams*, atau *word golf*) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. *Word ladder* ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai *start word* dan *end word*. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara *start word* dan *end word*. Banyaknya huruf pada *start word* dan *end word* selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata.

Untuk penyelesaian *word ladder* dapat digunakan algoritma penentuan rute seperti *Uniform Cost Search (UCS)*, *Greedy Best-First Search* dan A*. Setiap algoritma memiliki fungsi evaluasi f(n) untuk setiap node atau kata. Penjelasan algoritmanya sebagai berikut :

1. *Uniform Cost Search (UCS)*

Untuk algoritma UCS digunakan fungsi evaluasi f(n) = g(n), dimana g(n) adalah harga untuk mencapai n dari *node* awal. Untuk persoalan ini jarak dari tiap node besarnya sama, yaitu 1. Karena jaraknya sama, maka untuk persoalan ini UCS sama dengan BFS dalam urutan node yang dibangkitkan dan *path* yang dilalui. Langkah-langkah algoritma ini :

1. Masukkan kata awal dengan cost 0 ke dalam queue.
2. Catat kata awal telah dikunjungi
3. Jika kata terdepan queue merupakan kata akhir, maka algoritma selesai.
4. Jika tidak, bangkitkan semua kata yang memiliki perbedaan 1 huruf dengan kata terdepan queue. Untuk setiap kata, jika kata tersebut valid (ada dalam *dictionary* dan belum dikunjungi) masukkan kata tersebut ke belakang queue dengan cost kata terdepan queue ditambah 1 dan catat kata telah dikunjungi.
5. Hapus kata terdepan dari queue.
6. Ulangi langkah 3-6.

2. *Greedy Best-First Search*

Untuk algoritma GBFS digunakan fungsi evaluasi f(n) = h(n), dimana h(n) adalah harga perkiraan dari *node* n ke node tujuan. h(n) yang digunakan adalah banyaknya karakter yang berbeda dari kata n. Kekurangan dari algoritma ini adalah tidak menjamin ditemukannya solusi yang maksimal dikarenakan bisa saja menyangkut pada optimum lokal. Langkah-langkah algoritma ini :

1. Masukkan kata awal dengan cost 0 ke dalam priority queue.
2. Catat kata awal telah dikunjungi
3. Jika kata terdepan priority queue merupakan kata akhir, maka algoritma selesai.
4. Jika tidak, bangkitkan semua kata yang memiliki perbedaan 1 huruf dengan kata terdepan queue. Untuk setiap kata, jika kata tersebut valid (ada dalam *dictionary*

dan belum dikunjungi) masukkan kata tersebut ke priority queue yang terurut cost membesar dengan cost jarak dari kata tujuan dan catat kata telah dikunjungi.
5. Hapus kata yang di cek dari queue.
6. Ulangi langkah 3-6.

3. A (A Star)*

Untuk algoritma GBFS digunakan fungsi evaluasi f(n) = g(n) + h(n), dimana g(n) adalah harga untuk mencapai n dari *node* awal dan h(n) adalah harga perkiraan dari *node* n ke node tujuan. h(n) yang digunakan adalah banyaknya karakter yang berbeda dari kata n. Fungsi heuristik pada algoritma *A Star* bersifat *admissible*, karena fungsi yang digunakan untuk menghitung merupakan bestcase jarak dari kedua buah node. Secara teoritis, algoritma *A Star* lebih efisien dari algoritma UCS dalam penyelesaian *Word Ladder*, karena algoritma *A Star* memperhitungkan perkiraan jarak menuju tujuan sehingga pemilihan jalurnya tentu lebih baik. Langkah-langkah algoritma ini :

1. Masukkan kata awal dengan cost 0 ke dalam priority queue.
2. Catat kata awal telah dikunjungi
3. Jika kata terdepan priority queue merupakan kata akhir, maka algoritma selesai.
4. Jika tidak, bangkitkan semua kata yang memiliki perbedaan 1 huruf dengan kata terdepan queue. Untuk setiap kata, jika kata tersebut valid (ada dalam *dictionary* dan belum dikunjungi) masukkan kata tersebut ke priority queue yang terurut cost membesar dengan cost jarak dari kata tujuan ditambah dan catat kata telah dikunjungi.
5. Hapus kata yang di cek dari queue.
6. Ulangi langkah 3-6.

Bab 2: Source Code dan Implementasi

Program ini terdiri dari 3 file, yaitu Algo.java yang berisi algoritma, Node.java yang menyimpan objek *node* dan GUI.java yang berisi *Graphical User Interface*. Pertama, pada kelas Node berisi g yang menyimpan g(n) h yang menyimpan h(n) kata yang menyimpan kata dan path yang menyimpan path dari node awal ke node ini.

```
public class Node {
    public int g,h;
    public String kata;
    public ArrayList<String> path;

    public Node(int g,int h, String kata, ArrayList<String> path){
        this.g = g;
        this.h = h;
        this.kata = kata;
        this.path = path;
    }
}
```

Kedua, pada kelas Algo terdapat variabel static yang berfungsi sebagai variabel global. Variabel dictionary untuk menyimpan kata kata berbahasa inggris yang valid dan diambil dari sini, visited untuk menyimpan kata yang telah dikunjungi, String start dan end untuk menyimpan kata awal dan tujuan, startTime dan endTime untuk menghitung waktu, dikunjungi untuk menyimpan node yang dikunjungi, len untuk menyimpan panjang kata, path untuk menyimpan path yang dilewati, dan queue untuk menyimpan queue.

```
public class Algo {
    public static Set<String> dictionary, visited = new HashSet<>();
    public static String start, end;
    public static long startTime, endTime, dikunjungi;
    public static int len;
    public static ArrayList<String> path;
    public static ArrayList<Node> queue = new ArrayList<>();
```

Method readDictionary untuk memasukkan list kata dari file txt ke variabel dictionary.

```
// Membaca file txt yang berisi kata inggris yang valid
public static void readDictionary(String filePath) {
    dictionary = new HashSet<String>();
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
        String line;
        while ((line = reader.readLine()) != null) {
            dictionary.add(line.trim().toLowerCase());
        }
    } catch (IOException e) {
        System.err.println("Error reading from file: " + e.getMessage());
    }
}
```

Method UCS yang menerima kata awal dan kata akhir dan mengembalikan waktu dan jalur yang dilewati.

```
public static void UCS(){
    // Membersihkan visited, queue dan dikunjungi untuk GUI
    visited.clear();
    queue.clear();
    dikunjungi = 0;
    len = start.length();
    len = start.length();
    // Memulai perhitungan waktu
    startTime = System.currentTimeMillis();

    // Deklarasi node awal
    Node t = new Node(g:0,h:0, start, new ArrayList<String>());

    // Memasukkan node awal pada queue
    queue.add(t);
    visited.add(start);
```

```
    // Memulai proses UCS
    while(true){
        // Menambah node yang dikunjungi
        dikunjungi++;

        // Kata tujuan ditemukan! keluar dari while loop
        if(queue.get(index:0).kata.equals(end)){
            break;
        }

        // Menghasilkan kata yang berbeda 1 huruf
        for(int i=0;i<len;i++){
            for(char c : "abcdefghijklmnopqrstuvwxyz".toCharArray()){
                if(queue.get(index:0).kata.charAt(i) != c){
                    String tempString = queue.get(index:0).kata.substring(beginIndex:0, i) + c + queue.get(index:0).kata.substring(i+1);
                    // Jika kata valid dan belum dilewati
                    if(!visited.contains(tempString) && dictionary.contains(tempString)){
                        ArrayList<String> tempPath = new ArrayList<>(queue.get(index:0).path);
                        tempPath.add(tempString);
                        Node tempTree = new Node(queue.get(index:0).g+1,h:0, tempString, tempPath);
                        queue.add(tempTree);
                        visited.add(tempString);
                    }
                }
            }
        }

        // Menghapus kata yang sudah dicek dari queue
        queue.remove(index:0);
    }

    // Menghentikan perhitungan waktu
    endTime = System.currentTimeMillis();

    // Mencatat jalur yang telah ditemukan
    path = queue.get(index:0).path;
}
```

Method calculateDistanceToFinish yang menghitung jarak dari kata sekarang ke kata akhir.

```
public static int calculateDistanceToFinish(String s){
    int ret = 0;
    for(int i=0;i<s.length();i++){
        if(s.charAt(i) != end.charAt(i)){
            ret++;
        }
    }
    return ret;
}
```

Method insertGBFS untuk memasukan node pada queue sesuai cost.

```
public static void insertGBFS(Node t){
    if(queue.isEmpty() || queue.get(queue.size()-1).h <= t.h){
        queue.add(t);
    } else {
        for(int i=0;i<queue.size();i++){
            if(queue.get(i).h > t.h){
                queue.add(i, t);
                break;
            }
        }
    }
}
```

Method delete untuk menghapus kata yang di cek pada GBFS dan A*.

```
public static void delete(String s){
    for(int i=0;i<queue.size();i++){
        if(queue.get(i).kata.equals(s)){
            queue.remove(i);
            break;
        }
    }
}
```

Method GBFS yang menerima kata awal dan kata akhir dan mengembalikan waktu dan jalur yang dilewati.

```java
public static void GBFS(){
    // Membersihkan visited, queue dan dikunjungi untuk GUI
    visited.clear();
    queue.clear();
    dikunjungi = 0;
    len = start.length();
    // Memulai perhitungan waktu
    startTime = System.currentTimeMillis();

    // Deklarasi node awal
    Node t = new Node(g:0, calculateDistanceToFinish(start), start, new ArrayList<String>());

    // Memasukkan node awal pada queue
    queue.add(t);
    visited.add(start);
```

```java
// Memulai proses GBFS
while(true){
    // Menambah node yang dikunjungi
    dikunjungi++;
    Node treeSekarang = queue.get(index:0);

    // Kata tujuan ditemukan! keluar dari while loop
    if(treeSekarang.kata.equals(end)){
        break;
    }

    // Menghasilkan kata yang berbeda 1 huruf
    for(int i=0;i<len;i++){
        for(char c : "abcdefghijklmnopqrstuvwxyz".toCharArray()){
            if(treeSekarang.kata.charAt(i) != c){
                String tempString = treeSekarang.kata.substring(beginIndex:0, i) + c + treeSekarang.kata.substring(i+1);
                // Jika kata valid dan belum dilewati
                if(!visited.contains(tempString) && dictionary.contains(tempString)){
                    ArrayList<String> tempPath = new ArrayList<>(treeSekarang.path);
                    tempPath.add(tempString);
                    Node tempTree = new Node(g:0,calculateDistanceToFinish(tempString), tempString, tempPath);
                    insertGBFS(tempTree);
                    visited.add(tempString);
                }
            }
        }
    }

    // Menghapus kata yang sudah dicek dari queue
    delete(treeSekarang.kata);
}

// Menghentikan perhitungan waktu
endTime = System.currentTimeMillis();

// Mencatat jalur yang telah ditemukan
path = queue.get(index:0).path;
```

Method insertAS untuk memasukan node pada queue sesuai cost.

```java
public static void insertAS(Node t){
    if(queue.isEmpty() || queue.get(queue.size()-1).h + queue.get(queue.size()-1).g <= t.h + t.g){
        queue.add(t);
    } else {
        for(int i=0;i<queue.size();i++){
            if(queue.get(i).h + queue.get(i).g > t.h + t.g){
                queue.add(i, t);
                break;
            }
        }
    }
}
```

Method AS yang menerima kata awal dan kata akhir dan mengembalikan waktu dan jalur yang dilewati.

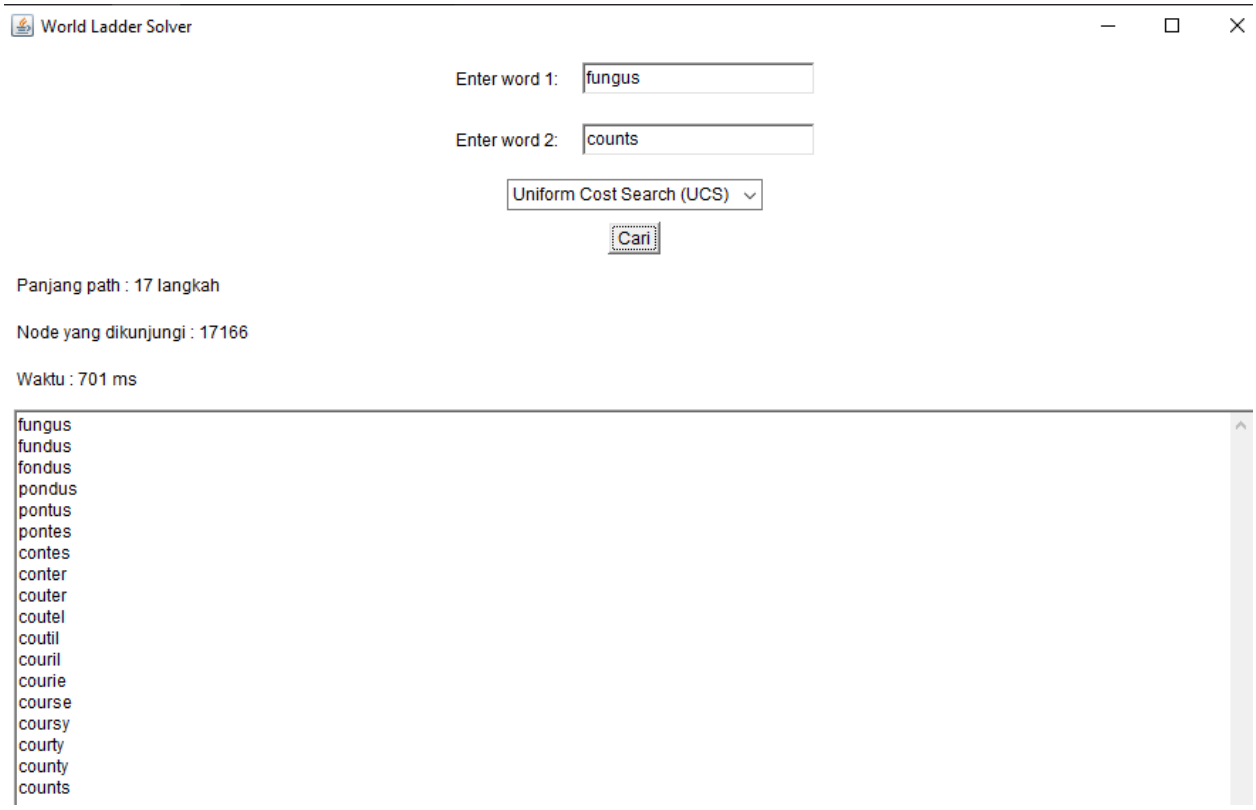
```java
public static void AS(){
    // Membersihkan visited, queue dan dikunjungi untuk GUI
    visited.clear();
    queue.clear();
    dikunjungi = 0;
    len = start.length();
    // Memulai perhitungan waktu
    startTime = System.currentTimeMillis();

    // Deklarasi node awal
    Node t = new Node(g:0, calculateDistanceToFinish(start), start, new ArrayList<String>());

    // Memasukkan node awal pada queue
    queue.add(t);
    visited.add(start);
```

```java
// Memulai proses AS
while(true){
    // Menambah node yang dikunjungi
    dikunjungi++;
    Node treeSekarang = queue.get(index:0);

    // Kata tujuan ditemukan! keluar dari while loop
    if(treeSekarang.kata.equals(end)){
        break;
    }

    // Menghasilkan kata yang berbeda 1 huruf
    for(int i=0;i<len;i++){
        for(char c : "abcdefghijklmnopqrstuvwxyz".toCharArray()){
            if(treeSekarang.kata.charAt(i) != c){
                String tempString = treeSekarang.kata.substring(beginIndex:0, i) + c + treeSekarang.kata.substring(i+1);
                // Jika kata valid dan belum dilewati
                if(!visited.contains(tempString) && dictionary.contains(tempString)){
                    ArrayList<String> tempPath = new ArrayList<>(treeSekarang.path);
                    tempPath.add(tempString);
                    Node tempTree = new Node(treeSekarang.g+1,calculateDistanceToFinish(tempString), tempString, tempPath);
                    insertAS(tempTree);
                    visited.add(tempString);
                }
            }
        }
    }

    // Menghapus kata yang sudah dicek dari queue
    delete(treeSekarang.kata);
}

// Menghentikan perhitungan waktu
endTime = System.currentTimeMillis();

// Mencatat jalur yang telah ditemukan
path = queue.get(index:0).path;
```

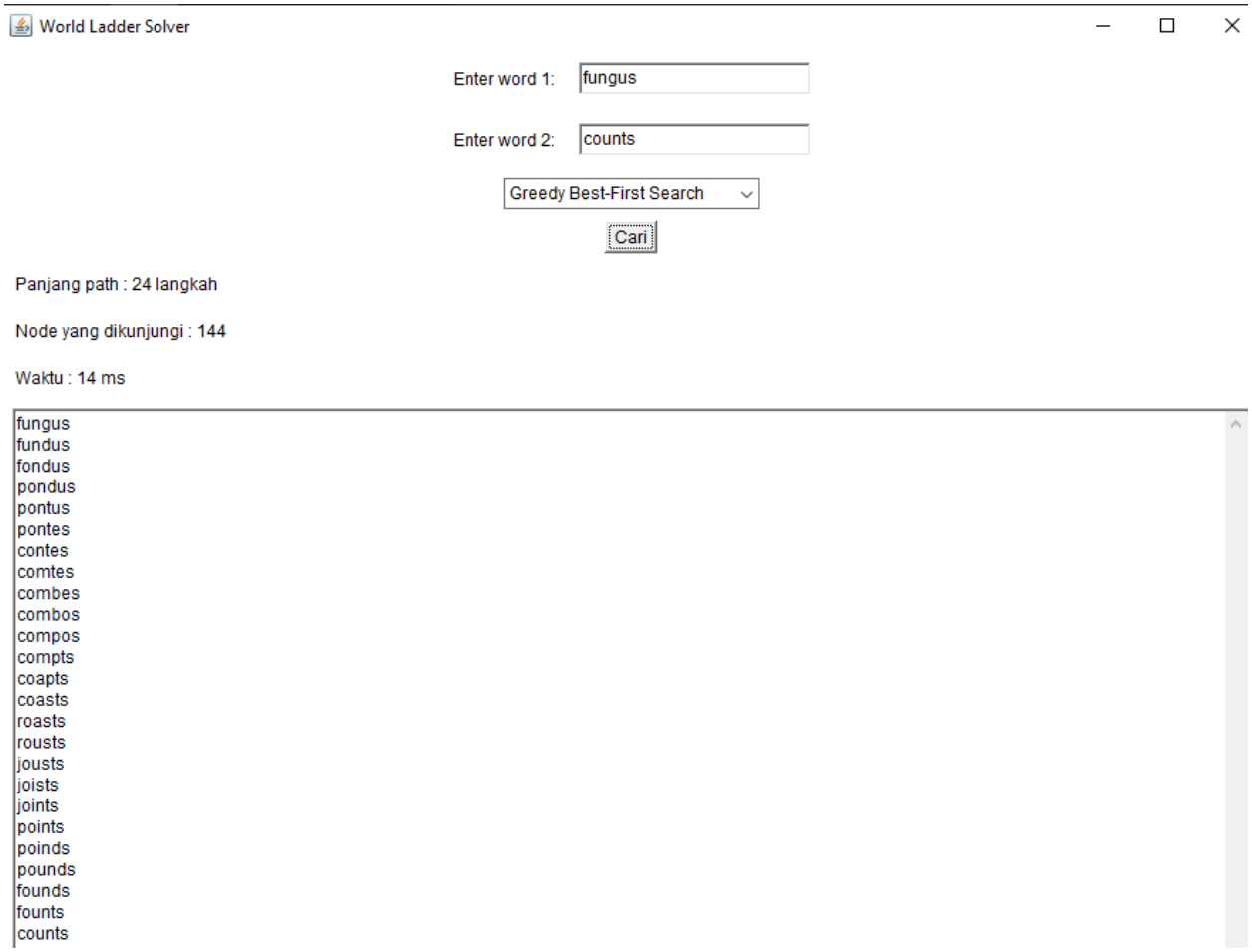
Bab 3: *Input* dan *Output* Program

fungus -> counts

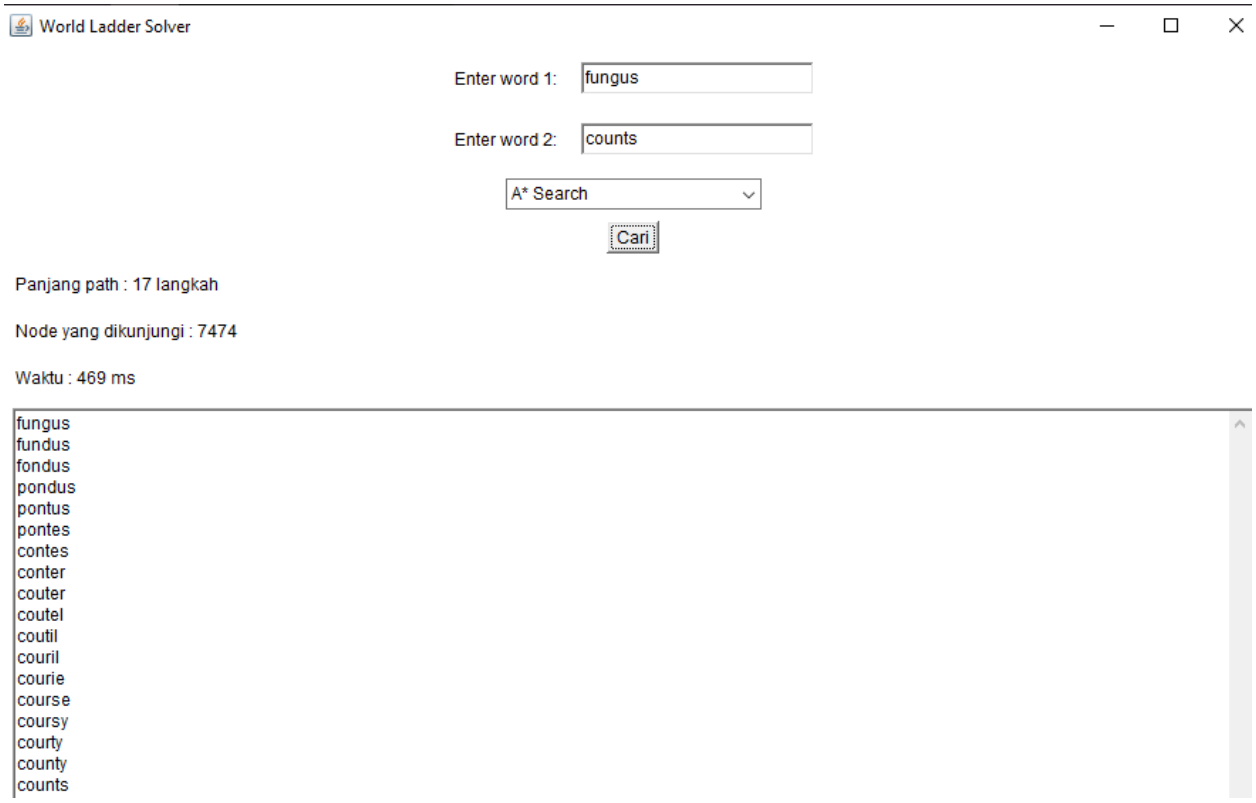
1. UCS



2. GBFS


World Ladder Solver
Enter word 1:
fungus
Enter word 2:
counts
Greedy Best-First Search
Cari
Panjang path : 24 langkah
Node yang dikunjungi : 144
Waktu : 14 ms
fungus
fundus
fondus
pondus
pontus
pontes
contes
comtes
combes
combos
compos
compts
coapts
coasts
roasts
rousts
jousts
joists
joints
points
poinds
pounds
founds
founts
counts

3. A*



greedy -> normal

1. UCS


World Ladder Solver
Enter word 1:
greedy
Enter word 2:
normal
Uniform Cost Search (UCS)
Cari
Panjang path : 16 langkah
Node yang dikunjungi : 12723
Waktu : 498 ms
greedy
breedy
breezy
freezy
freety
fretty
pretty
pratty
pratey
prater
frater
framer
foamer
former
formel
formal
normal

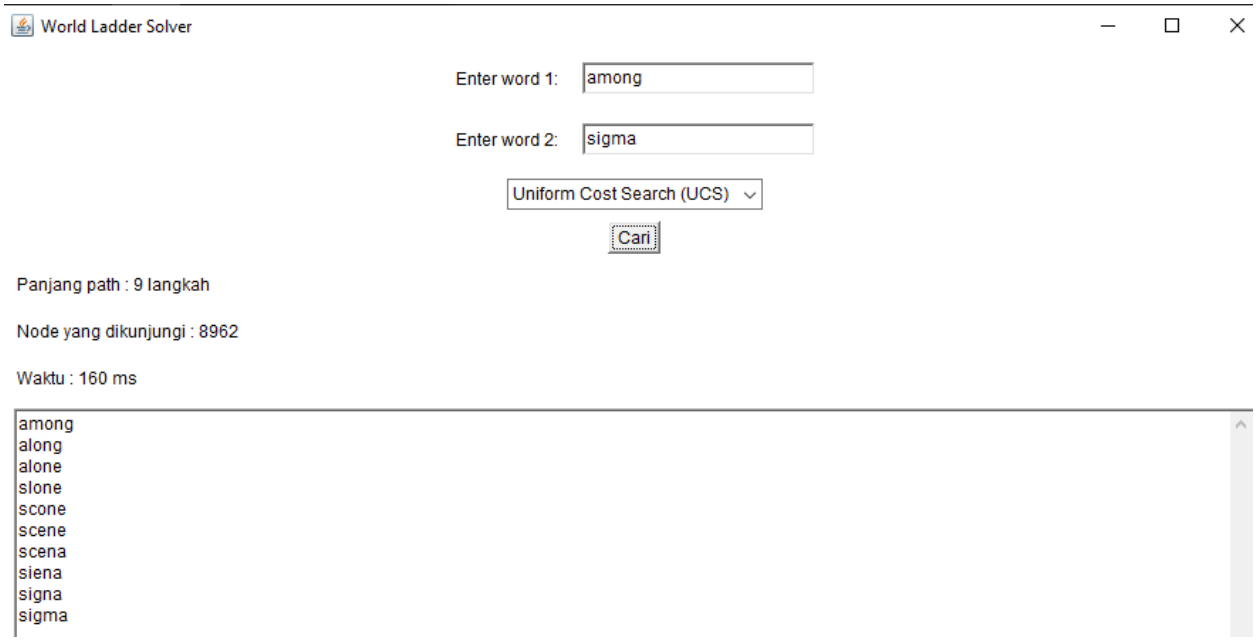
2. GBFS


World Ladder Solver
Enter word 1:
greedy
Enter word 2:
normal
Greedy Best-First Search
Cari
Panjang path : 16 langkah
Node yang dikunjungi : 148
Waktu : 10 ms
greedy
breedy
breeds
bleeds
bleeps
sleeps
steeps
streps
strips
stripe
striae
strial
serial
ferial
fermal
formal
normal

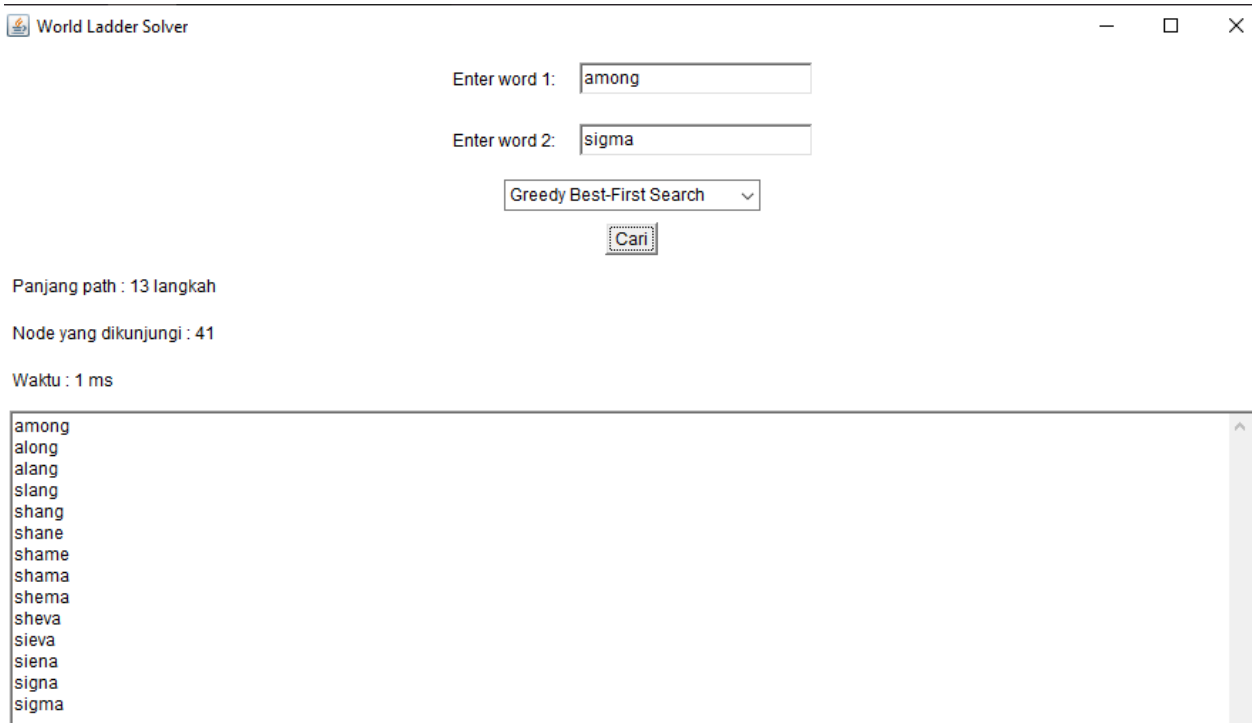
3. A*

among -> sigma

1. UCS



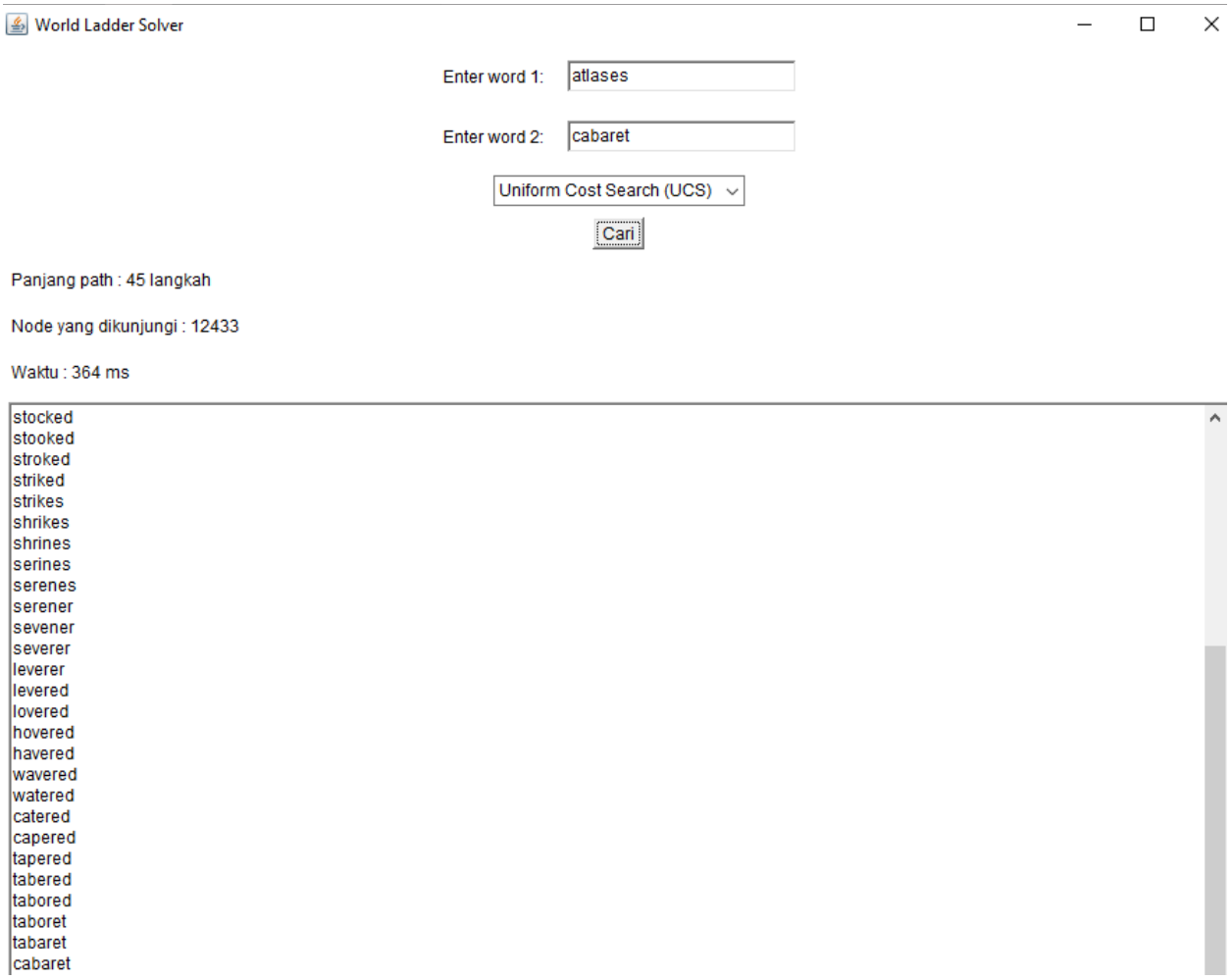
2. GBFS



3. A*

atlases -> cabaret

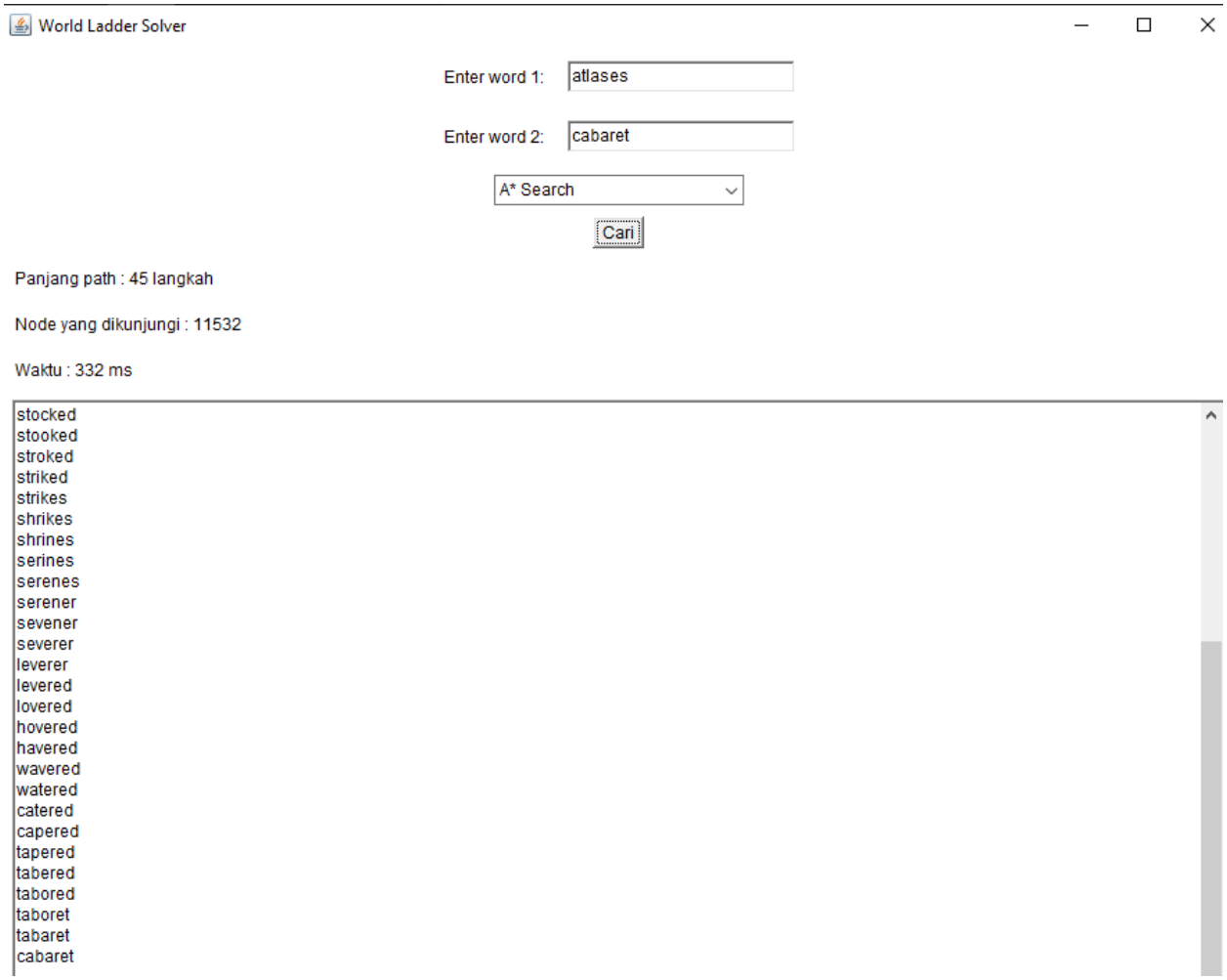
1. UCS


World Ladder Solver
Enter word 1:
atlases
Enter word 2:
cabaret
Uniform Cost Search (UCS)
Cari
Panjang path : 45 langkah
Node yang dikunjungi : 12433
Waktu : 364 ms
stocked
stooked
stroked
striked
strikes
shrikes
shrines
serines
serenes
serener
sevener
severer
leverer
levered
lovered
hovered
havered
wavered
watered
catered
capered
tapered
tabered
tabored
taboret
tabaret
cabaret

2. GBFS

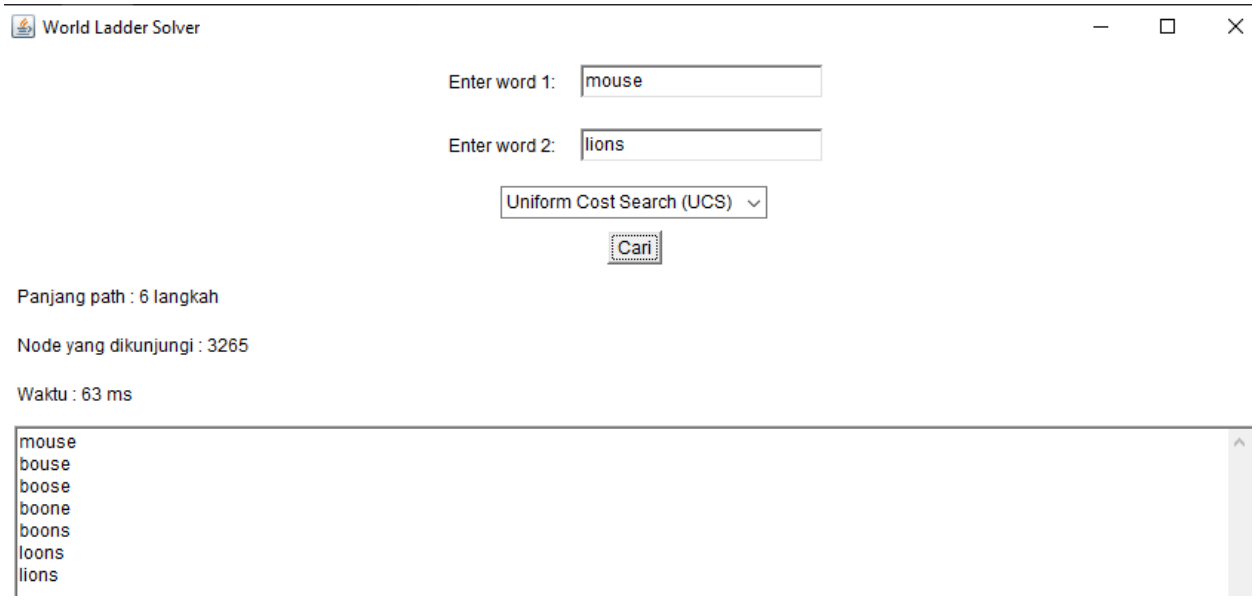

World Ladder Solver
Enter word 1:
atlases
Enter word 2:
cabaret
Greedy Best-First Search
Cari
Panjang path : 66 langkah
Node yang dikunjungi : 1345
Waktu : 84 ms
cantlet
mantlet
martlet
wartlet
warblet
warbled
wabbled
cabbled
cabbler
gabbler
gabeler
gaveler
raveler
ravener
havener
haverer
waverer
waterer
caterer
caperer
capered
tapered
tabered
tabored
taboret
tabaret
cabaret

3. A*

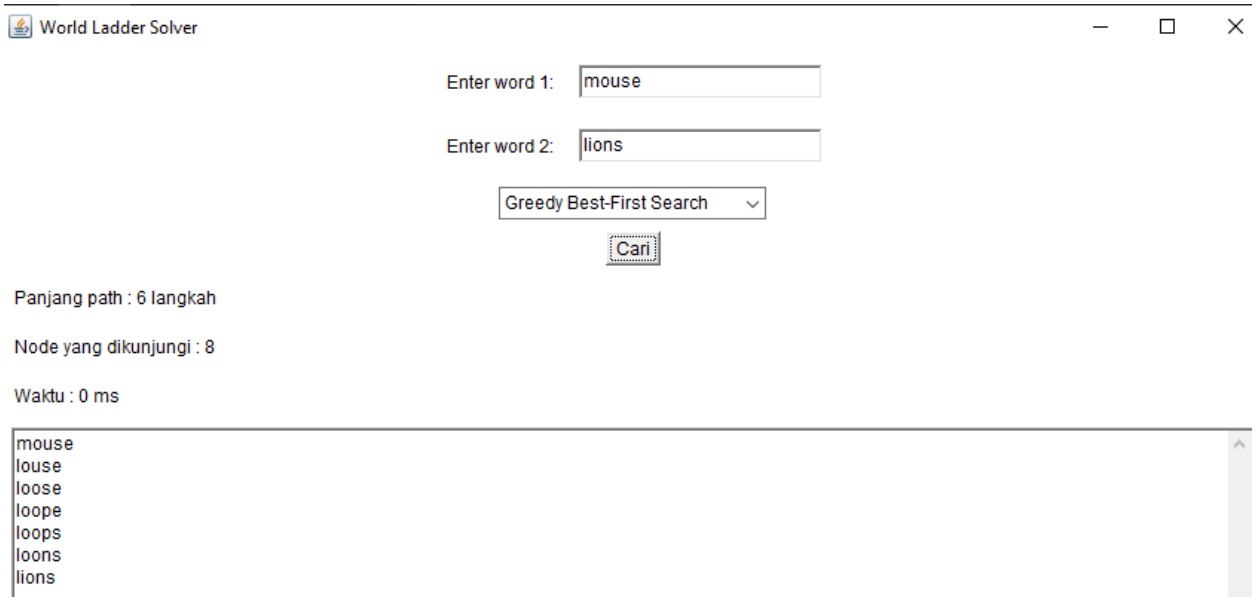


mouse -> lions

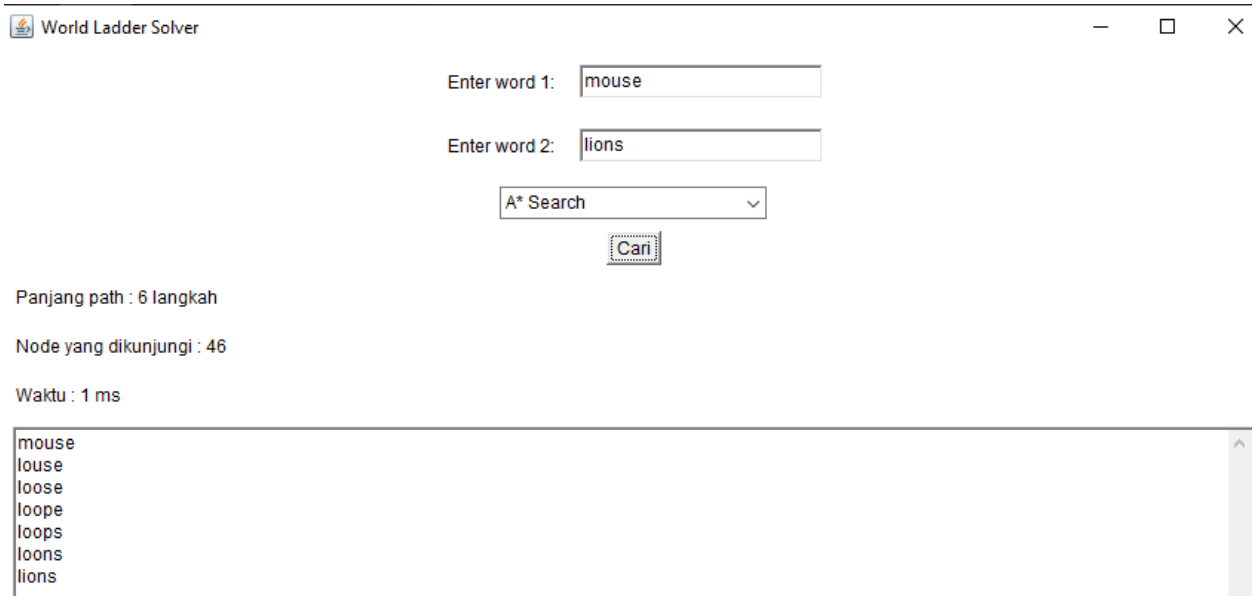
1. UCS


World Ladder Solver
Enter word 1:
mouse
Enter word 2:
lions
Uniform Cost Search (UCS)
Cari
Panjang path : 6 langkah
Node yang dikunjungi : 3265
Waktu : 63 ms
mouse
bouse
boose
boone
boons
loons
lions

2. GBFS

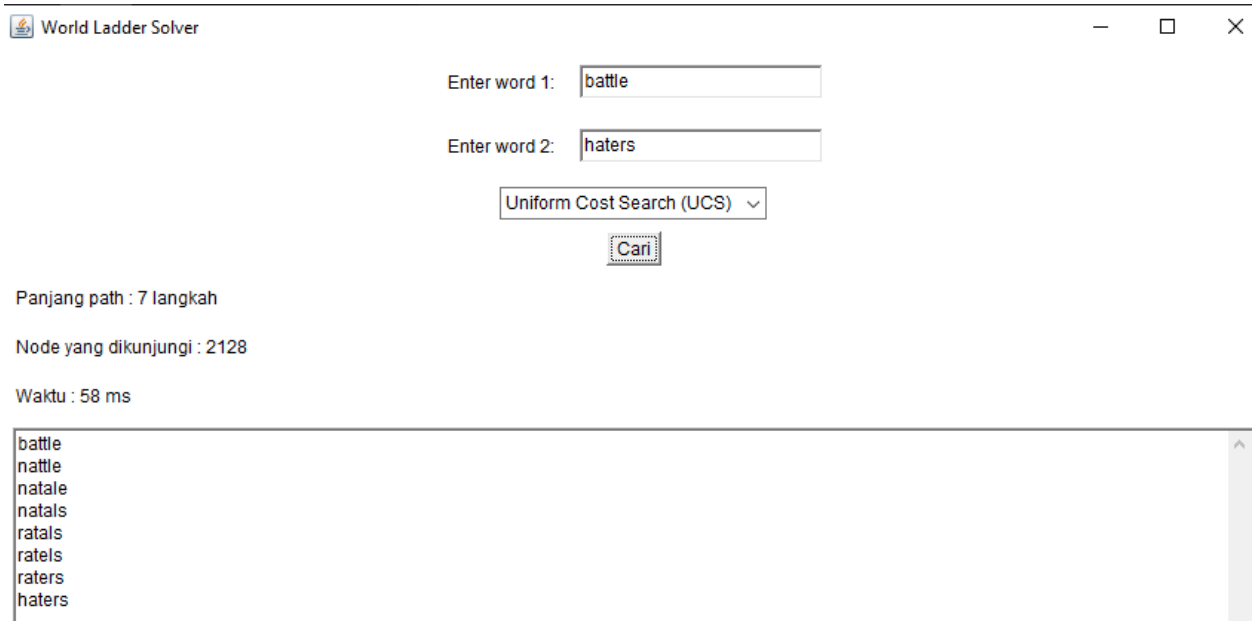

World Ladder Solver
Enter word 1:
mouse
Enter word 2:
lions
Greedy Best-First Search
Cari
Panjang path : 6 langkah
Node yang dikunjungi : 8
Waktu : 0 ms
mouse
louse
loose
loope
loops
loons
lions

3. A*

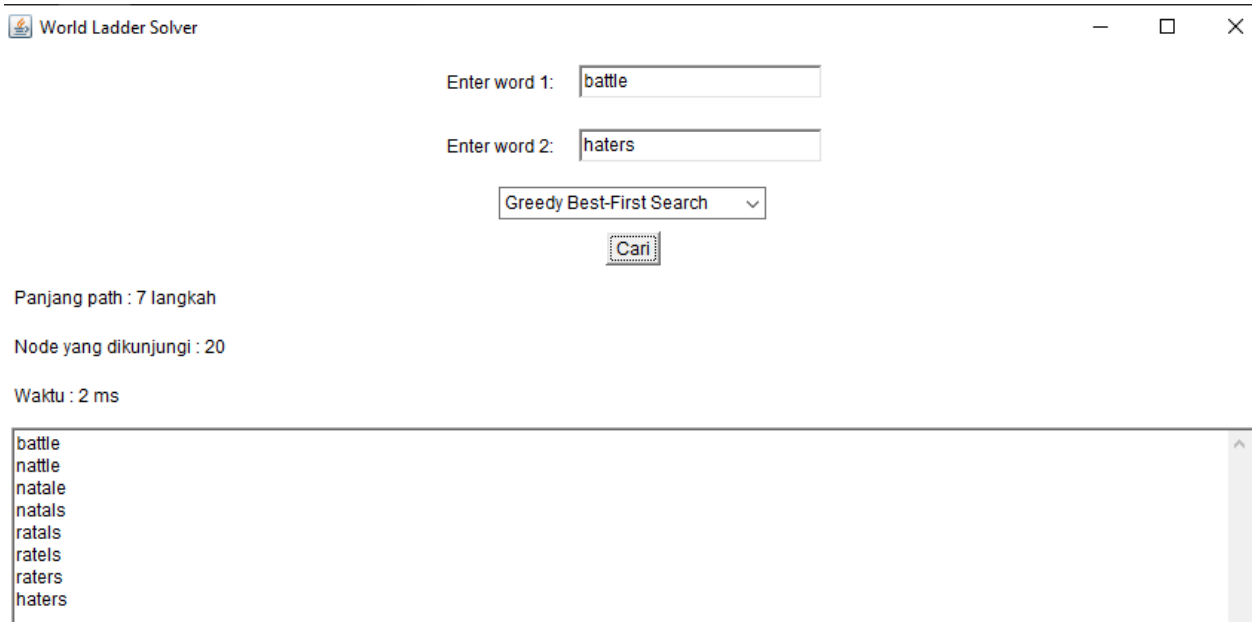


battle -> haters

1. UCS



2. GBFS


World Ladder Solver
Enter word 1:
battle
Enter word 2:
haters
Greedy Best-First Search
Cari
Panjang path : 7 langkah
Node yang dikunjungi : 20
Waktu : 2 ms
battle
nattle
natale
natals
ratals
ratels
raters
haters

3. A*


World Ladder Solver
Enter word 1:
battle
Enter word 2:
haters
A* Search
Cari
Panjang path : 7 langkah
Node yang dikunjungi : 57
Waktu : 2 ms
battle
nattle
natale
natals
ratals
ratels
raters
haters

Bab 4: Analisis Perbandingan Solusi

Berdasarkan Bab 3, didapatkan hasil sebagai berikut :

Test case	UCS	GBFS	A*
fungus -> counts	Panjang jalur : 17 Node yang dikunjungi : 17166 Waktu : 701ms	Panjang jalur : 24 Node yang dikunjungi : 144 Waktu : 14ms	Panjang jalur : 17 Node yang dikunjungi : 7474 Waktu : 469ms
greedy -> normal	Panjang jalur : 16 Node yang dikunjungi : 12723 Waktu : 498ms	Panjang jalur : 16 Node yang dikunjungi : 148 Waktu : 10ms	Panjang jalur : 16 Node yang dikunjungi : 2098 Waktu : 144ms
among -> sigma	Panjang jalur : 9 Node yang dikunjungi : 8962 Waktu : 160ms	Panjang jalur : 13 Node yang dikunjungi : 41 Waktu : 1ms	Panjang jalur : 9 Node yang dikunjungi : 331 Waktu : 6ms
atlases -> cabaret	Panjang jalur : 45 Node yang dikunjungi : 12433 Waktu : 364ms	Panjang jalur : 66 Node yang dikunjungi : 1345 Waktu : 84ms	Panjang jalur : 45 Node yang dikunjungi : 11532 Waktu : 332ms
mouse -> lions	Panjang jalur : 6 Node yang dikunjungi : 3265 Waktu : 63ms	Panjang jalur : 6 Node yang dikunjungi : 8 Waktu : 0ms	Panjang jalur : 6 Node yang dikunjungi : 46 Waktu : 1ms
battle -> haters	Panjang jalur : 7 Node yang dikunjungi : 2128 Waktu : 58ms	Panjang jalur : 7 Node yang dikunjungi : 20 Waktu : 2ms	Panjang jalur : 7 Node yang dikunjungi : 57 Waktu : 2ms

Dari data yang didapatkan dapat ditarik berbagai hal, seperti :

1. Algoritma UCS dan A* selalu menghasilkan solusi yang optimal (jumlah langkah minimum), sedangkan algoritma GBFS tidak selalu menghasilkan solusi yang optimal
2. Dari waktu eksekusi dapat dilihat bahwa algoritma GBFS memiliki waktu yang paling cepat, disusul oleh A* dan terakhir UCS.
3. Untuk memori yang dibutuhkan dapat dilihat dari jumlah *node* yang dikunjungi, sama seperti waktu, algoritma GBFS memiliki penggunaan memori yang paling sedikit, diikuti oleh A* dan terakhir UCS.

Bab 5: Implementasi Bonus

Digunakan GUI dengan API java AWT (Abstract Window Toolkit). Ada kelas GUI yang bertugas untuk menghasilkan GUI nya. Fungsi main yang akan membaca dictionary dan membuat instansiasi GUI.

```java
public static void main(String[] args) {
    // Deklarasi set untuk menyimpan kata inggris yang valid
    Algo.readDictionary(filePath:"../src/dict.txt");
    new GUI();
}
```

Set layout dan inisiasi constraint.

```java
public GUI() {
    // Set Layout
    setLayout(new GridBagLayout());
    GridBagConstraints constraints = new GridBagConstraints(), constraints2 = new GridBagConstraints();

    // Constraints agar flow kebawah dan center
    constraints.fill = GridBagConstraints.NONE;
    constraints.gridx = 0;  // 1 kolom
    constraints.gridy = GridBagConstraints.RELATIVE;  // flow kebawah
    constraints.weightx = 1.0;  // width : 100%
    constraints.insets = new Insets(top:5, left:10, bottom:5, right:10);  // Margin

    // Constraints untuk output
    constraints2.fill = GridBagConstraints.HORIZONTAL;
    constraints2.gridx = 0;  // 1 kolom
    constraints2.gridy = GridBagConstraints.RELATIVE;  // flow kebawah
    constraints2.weightx = 1.0;  // width : 100%
    constraints2.insets = new Insets(top:5, left:10, bottom:5, right:10);  // Margin
```

Input 2 kata dan pilihan algoritma

```
// Input start
Panel firstInputPanel = new Panel();
firstInputPanel.add(new Label(text:"Enter word 1:"));
startField = new TextField(columns:20);
firstInputPanel.add(startField);
add(firstInputPanel, constraints);

// Input end
Panel secondInputPanel = new Panel();
secondInputPanel.add(new Label(text:"Enter word 2:"));
endField = new TextField(columns:20);
secondInputPanel.add(endField);
add(secondInputPanel, constraints);

// Pilihan algoritma
choice = new Choice();
choice.add(item:"Uniform Cost Search (UCS)");
choice.add(item:"Greedy Best-First Search");
choice.add(item:"A* Search");
choice.setPreferredSize(new Dimension(width:2000, height:20));  // Set the preferred size
add(choice, constraints);
```

Button untuk memulai pencarian *path*, dan penempatan label untuk output.

```
// Button cari
findButton = new Button(label:"Cari");
findButton.addActionListener(this);
add(findButton, constraints);

// Label output
pathSize = new Label(text:"");
add(pathSize, constraints2);
nodeSize = new Label(text:"");
add(nodeSize, constraints2);
time = new Label(text:"");
add(time, constraints2);

// Label path
path = new TextArea(rows:5, columns:0);
path.setPreferredSize(new Dimension(width:100,height:20));
constraints.weighty = 1;  // Give extra vertical space to the text area
constraints.fill = GridBagConstraints.BOTH;
add(path, constraints);
```

Setting windows

```
setTitle(title:"World Ladder Solver");
pack();
setVisible(b:true);
setExtendedState(Frame.MAXIMIZED_BOTH);

// Handle tutup window
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
        dispose();
    }
});
```

method actionPerformed untuk memulai pencarian.

```
// Fungsi jika button ditekan
public void actionPerformed(ActionEvent e) {
    // Mereset label output
    pathSize.setText(text:"");
    nodeSize.setText(text:"");
    time.setText(text:"");
    if(e.getSource() == findButton){
        Algo.start = startField.getText();
        Algo.end = endField.getText();
        String selectedChoice = choice.getSelectedItem();

        // validasi input
        if(Algo.start.length() != Algo.end.length()){
            nodeSize.setText(text:"2 Kata panjangnya berbeda");
        } else if(!Algo.dictionary.contains(Algo.start)){
            nodeSize.setText(Algo.start +" bukan kata yang valid");
        } else if(!Algo.dictionary.contains(Algo.end)){
            nodeSize.setText(Algo.end +" bukan kata yang valid");
        } else {
```

```
try {
    // pemanggilan fungsi algoritma
    if(selectedChoice.equals(anObject:"Uniform Cost Search (UCS)")){
        Algo.UCS();
    } else if(selectedChoice.equals(anObject:"Greedy Best-First Search")){
        Algo.GBFS();
    } else {
        Algo.AS();
    }

    // Output ke layar
    pathSize.setText("Panjang path : " + Algo.path.size() + " langkah");
    nodeSize.setText("Node yang dikunjungi : " + Algo.dikunjungi);
    time.setText("Waktu : " + (Algo.endTime - Algo.startTime) + " ms");

    path.setText(Algo.start + "\n");  // Clear previous results
    for (String item : Algo.path) {
        path.append(item + "\n");
    }
} catch (IndexOutOfBoundsException err) {
    // Jika path tidak ada
    nodeSize.setText("Tidak ditemukan path dari " + Algo.start + " menuju " + Algo.end + ".");
}
```

Daftar Pustaka

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

Link Repository

https://github.com/Kizaaaa/Tucil3_13522059