

LAPORAN TUGAS KECIL 3

IF2211 - STRATEGI ALGORITMA

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



Nama : Rici Trisna Putra

NIM : 13522026

Kelas : 02

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

Bab 1

Deskripsi Masalah

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

How To Play

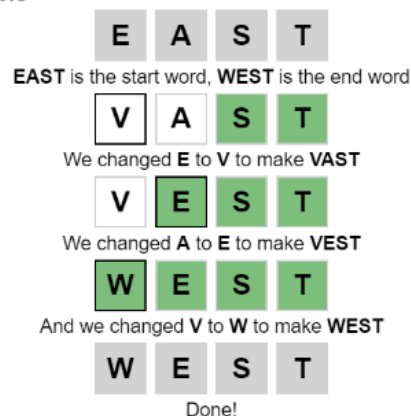
This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

Rules

Weave your way from the start word to the end word.

Each word you enter **can only change 1 letter** from the word above it.

Example



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

Bab 2

Implementasi Algoritma UCS, Greedy Best First Search dan A*

2.1 Uniform Cost Search (UCS)

Uniform Cost Search merupakan suatu algoritma pencarian rute tanpa informasi atau *uniformed search*. UCS menggunakan fungsi untuk menghitung biaya kumulatif terendah dari simpul awal untuk menentukan jalur ($g(n)$). UCS umumnya digunakan pada graf berbobot untuk menemukan jalur dengan biaya kumulatif terendah yang diperluas sesuai biaya traversalnya dari simpul awal. UCS termasuk dalam uninformed search karena tidak mempertimbangkan keadaan simpul atau ruang pencarian. UCS biasanya diimplementasikan menggunakan suatu priority queue untuk menyimpan simpul hidup dengan urutan biaya yang menaik.

Dalam algoritma UCS ini saya membuat sebuah class yang merepresentasikan kata sebagai sebuah simpul pohon, sebuah priority queue untuk menyimpan simpul-simpul hidup yang akan diekspansi, dan sebuah himpunan yang menyimpan semua kata yang sudah pernah diekspansi. Adapun fungsi biaya $g(n)$ yang digunakan adalah jumlah huruf berbeda antara dua kata dari simpul awal.

Dengan kata awal sebagai awalan, langkah-langkah algoritmanya adalah sebagai berikut:

1. Ambil simpul hidup dengan biaya paling rendah pada priority queue dan masukkan kata pada simpul ini ke dalam himpunan.
2. Cek kata sama dengan kata yang dituju (*goal node*), apabila benar maka kembalikan rangkaian kata pada jalur dari simpul awal ke simpul ini.
3. Ekspan kata pada simpul tersebut sehingga menghasilkan semua kata yang hanya beda satu huruf dan terdapat pada kamus.
4. Untuk tiap kata hasil ekspansi buat menjadi simpul dengan simpul induknya adalah simpul yang sedang diekspansi. Biaya dari simpul ini adalah beda huruf dari kata simpul dengan kata pada simpul induknya kemudian ditambah biaya dari simpul induknya.
5. Cek apakah tiap kata hasil ekspansi ini terdapat dalam himpunan (sudah pernah diekspansi sebelumnya), apabila belum maka masukkan ke priority queue terurut menaik sesuai biaya $g(n)$ dari simpul tersebut.
6. Ulangi hingga ditemukan solusi atau priority queue kosong.

Apabila diperhatikan lagi, algoritma UCS pada kasus *word ladder* ini mirip dengan algoritma *Breadth-first-search* (BFS) pada aspek urutan simpul yang dikunjungi karena tiap simpul anak memiliki biaya yang mirip yakni $1 + \text{biaya simpul induk}$ (karena pasti hanya berbeda satu huruf saja dari simpul induknya) sehingga tiap simpul anak akan dikunjungi dengan urutan yang mirip sekali dengan yang ada pada algoritma BFS.

2.2 Greedy Best First Search (GBFS)

Greedy Best First Search merupakan sebuah algoritma pencarian rute dengan informasi atau *informed search* yang evaluasi pemilihan rutanya menggunakan suatu fungsi heuristik $h(n)$. pada suatu graf berbobot, GBFS mengabaikan bobot dari suatu simpul untuk menentukan simpul selanjutnya dan bergantung hanya pada hasil fungsi heuristik yang digunakan. Sesuai namanya pula, Greedy Best First Search tidak dapat melakukan runut-balik (*backtracking*) sehingga mengakibatkan GBFS tidak menjadi solusi optimal. Pada kasus *word ladder*, algoritma ini tidak menjadi solusi optimal atau bahkan bisa tidak mendapatkan solusi apapun karena dapat mengalami jalan buntu (*dead end*).

Dalam algoritma GBFS ini saya membuat suatu larik bertipe string untuk menyimpan seluruh kata yang dipilih oleh fungsi heuristik. Fungsi heuristik yang saya gunakan adalah jumlah beda huruf antara suatu kata dengan kata tujuan. Dengan kata awal sebagai awalan maka langkah-langkah algoritmanya adalah sebagai berikut:

1. Ekspan kata sehingga menghasilkan semua kata yang beda satu huruf dengan kata tersebut.
2. Untuk tiap kata hasil ekspan hitung hasil evaluasi fungsi heuristiknya, yakni perbedaan kata antara tiap kata dengan kata tujuan.
3. Cari kata dengan evaluasi fungsi heuristik terendah dan pilih kata tersebut.
4. Cek apakah kata yang terpilih sudah pernah masuk ke larik, apabila sudah maka terjadi suatu loop sehingga terjadi *dead end*.
5. Apabila belum maka masukkan kata ke dalam larik dan jadikan kata tersebut sebagai kata yang akan diekspansi.
6. Cek apakah kata yang dimasukkan terakhir ke larik merupakan kata tujuan, apabila y kembalikan isi larik.
7. Ulangi hingga ditemukan solusi atau terjadi *dead end*.

2.3 A*

A* merupakan suatu algoritma pencarian rute dengan informasi atau *informed search* yang dapat menemukan rute dengan efisien pada suatu graf berarah yang tidak mengandung bobot negatif. A* biasanya diimplementasikan menggunakan priority queue seperti UCS dengan bobotnya yang dihitung dengan menggabungkan fungsi biaya pada Uniform Cost Search dan fungsi heuristik pada Greedy Best First Search untuk menentukan jalur yakni $f(n) = g(n) + h(n)$. Fungsi $f(n)$ ini mengestimasi biaya terendah dari simpul tersebut ke simpul tujuan. Fungsi $f(n)$ yang digunakan pada A* ini harus *admissible* yakni untuk tiap simpul fungsi heuristiknya $h(n)$ tidak mengoverestimasi biaya sesungguhnya dari simpul tersebut ke simpul tujuan.

Dalam algoritma A* ini saya membuat sebuah class yang merepresentasikan kata sebagai simpul pohon, sebuah priority queue untuk menyimpan simpul-simpul hidup yang akan diekspansi, dan sebuah map untuk menyimpan biaya terkecil untuk mencapai simpul dengan kata tertentu. Adapun fungsi estimasi $f(n)$ yang saya gunakan adalah $g(n)$ yang

digunakan pada UCS yakni jumlah huruf berbeda antara dua kata dari simpul awal dan $h(n)$ yang digunakan pada BGFS yakni jumlah beda huruf antara suatu kata dengan kata tujuan.

Dengan kata awal sebagai awalan maka langkah-langkah algoritmanya adalah sebagai berikut:

1. Ambil simpul hidup dengan biaya paling rendah pada priority queue.
2. Cek apakah kata tersebut sama dengan kata tujuan (goal node), apabila benar maka kembalikan rangkaian kata pada jalur dari simpul awal ke simpul ini.
3. Ekspan kata pada simpul tersebut sehingga menghasilkan semua kata yang hanya beda satu huruf dan terdapat pada kamus.
4. Untuk tiap kata hasil ekspansi buat menjadi simpul dengan simpul induknya adalah simpul yang sedang diekspansi. Biaya dari simpul ini dihitung dari penjumlahan beda huruf dari kata simpul hingga simpul awal $g(n)$ dan dijumlahkan dengan beda huruf dari kata simpul dengan kata simpul akhir $f(n)$.
5. Cek apakah kata ini belum masuk ke map atau total cost nya saat ini lebih kecil dari entry dengan kata yang sama pada map, apabila benar maka masukkan kata dan total cost pada map kemudian masukkan simpul ke dalam priority queue.
6. Ulangi hingga ditemukan solusi atau priority queue kosong.

Fungsi estimasi $f(n)$ yang digunakan pada kasus ini merupakan fungsi yang *admissible* karena fungsi heuristik $h(n)$ yakni jumlah beda huruf antara kata dengan kata akhir merupakan fungsi yang bersifat optimis untuk mengestimasi jumlah kata yang dibutuhkan untuk sampai kata tujuan. Hal ini dikarenakan jumlah perubahan huruf paling minimum yang dibutuhkan untuk mencapai kata akhir adalah persis sama dengan beda jumlah kata saat ini dengan kata akhir. Sehingga jumlah step yang dibutuhkan adalah setidaknya sebanyak $h(n)$ atau lebih. Sehingga untuk tiap node n , $h(n) \leq h^*(n)$ (*admissible*).

Bab 3

Source Code Program

Pada implementasi ini saya membuat 6 buah file java berbeda yakni:

1. Dictionary.java
2. Helper.java
3. UCS.java
4. GBFS.java
5. AStar.java
6. Main.java

3.1 Dictionary.java

Merupakan sebuah kelas yang memiliki:

1. Himpunan String bernama dictionary yang bersifat statik dan digunakan untuk menyimpan hasil bacaan dictionary sehingga bisa diakses oleh kelas lain.
2. Method statik readDictionary() yang berfungsi untuk membaca suatu dictionary.txt. Hasil pembacaan ini disimpan pada statik member dictionary yang dijelaskan pada poin 1.

3.2 Helper.java

Merupakan sebuah kelas yang memiliki:

1. Method statik findAdjacentWords(String seed) yang bertugas mencari semua String beda satu huruf dari suatu String seed yang terdapat dalam kamus.
2. Method statik findDifference(String s1, String s2) yang bertugas menghitung jumlah beda huruf antara s1 dan s2

3.3 UCS.java

Merupakan sebuah kelas yang memiliki:

1. Definisi Kelas UCSNodeComparator yang digunakan untuk mengurutkan UCSNode pada priority queue dengan urutan menaik
2. Definisi Kelas UCSNode yang digunakan untuk merepresentasikan kata menjadi sebuah simpul dimana tiap simpul memiliki:
 - a. Integer cost yang menyimpan biaya simpul dari simpul awal.
 - b. String word yang menyimpan kata simpul.
 - c. UCSNode parent yang menyimpan referensi simpul induk.
 - d. Dua buah Konstruktor UCSNode
 - e. Method statik findPath(UCSNode node) yang mengembalikan urutan kata dari node ke simpul awal.

3. Definisi kelas UCS yang memiliki:
 - a. Statik Integer `node_count` yang berfungsi untuk menyimpan jumlah node dikunjungi untuk tiap eksekusi `UCSFind()`
 - b. Method statik `UCSFind(String start, String end)` yang berfungsi melakukan algoritma UCS untuk mencari penyelesaian Word Ladder.

3.4 GBFS.java

Merupakan sebuah kelas yang memiliki:

1. Statik Integer `node_count` yang berfungsi untuk menyimpan jumlah node dikunjungi untuk tiap eksekusi `GBFSFind()`
2. Method statik `GBFSFind(String start, String end)` yang berfungsi melakukan algoritma GBFS untuk mencari penyelesaian Word Ladder.

3.5 AStar.java

Merupakan sebuah kelas yang memiliki:

1. Definisi Kelas `AStarNodeComparator` yang digunakan untuk mengurutkan `AStarNode` pada priority queue dengan urutan menaik
2. Definisi Kelas `AStarNode` yang digunakan untuk merepresentasikan kata menjadi sebuah simpul dimana tiap simpul memiliki:
 - a. Integer `cost_from_root` yang menyimpan biaya simpul dari simpul awal $g(n)$.
 - b. Integer `total_cost` yang menyimpan biaya total dari fungsi estimasi $f(n) = g(n) + h(n)$.
 - c. String `word` yang menyimpan kata simpul.
 - d. `AStarNode` `parent` yang menyimpan referensi simpul induk.
 - e. Dua buah Konstruktors `AStarNode`
 - f. Method statik `findPath(AStarNode node)` yang mengembalikan urutan kata dari node ke simpul awal.
3. Definisi kelas UCS yang memiliki:
 - a. Statik Integer `node_count` yang berfungsi untuk menyimpan jumlah node dikunjungi untuk tiap eksekusi `AStarFind()`
 - b. Method statik `AStarFind(String start, String end)` yang berfungsi melakukan algoritma A* untuk mencari penyelesaian Word Ladder.

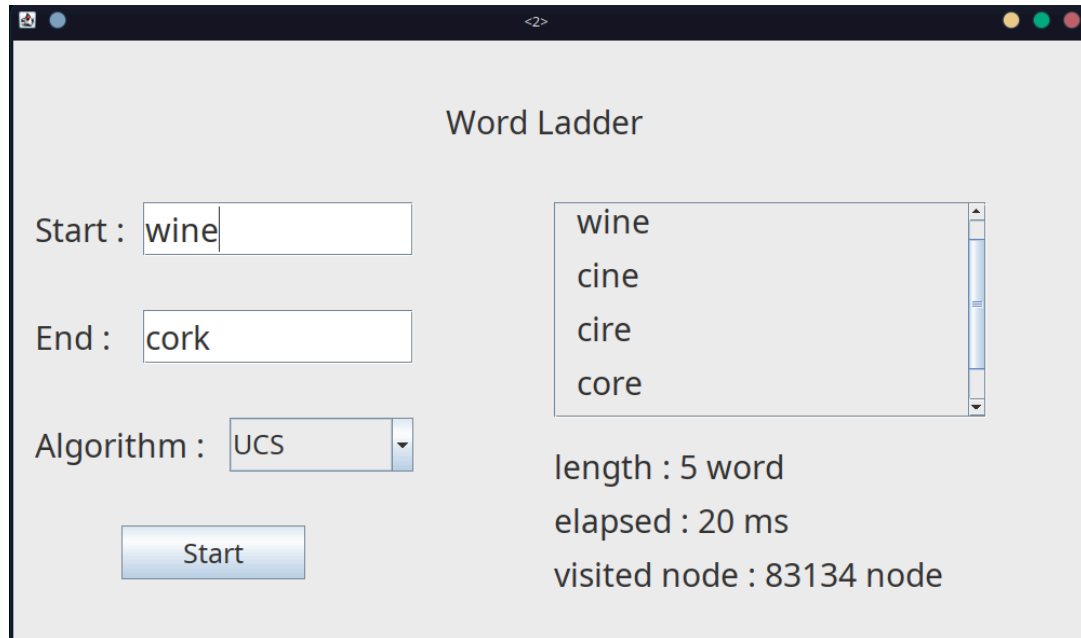
3.6 Main.java

Merupakan sebuah kelas utama yang mengandung fungsi main program dan implementasi GUI Bonus dimana Java Swing digunakan. Kelas ini tidak perlu terlalu dijelaskan karena lumayan banyak teknis Java Swing yang sekiranya tidak berhubungan dengan fokus pengerjaan tugas kecil ini. Intinya di dalam kelas ini terdapat fungsi main program, fungsi `draw` untuk melakukan paint pada GUI dan beberapa action listener yang digunakan untuk mengawasi klik pada button dan melakukan respons.

Bab 4

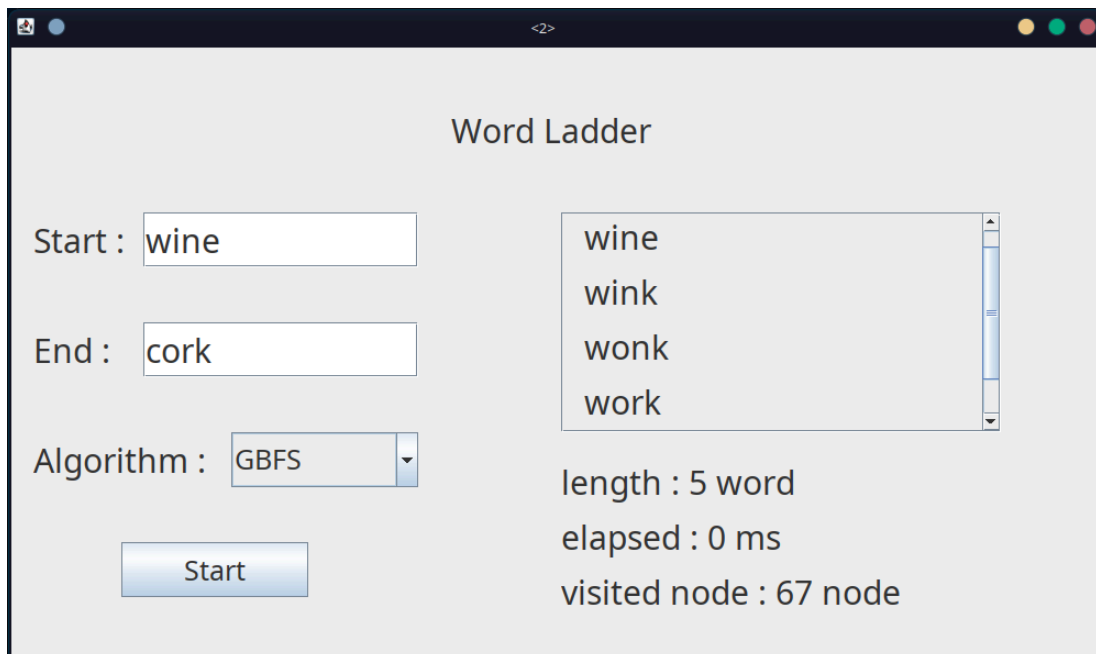
Tangkapan Layar Input dan Output Program

4.1 wine - cork (UCS)



The screenshot shows a window titled "Word Ladder". On the left, there are input fields for "Start : wine", "End : cork", and "Algorithm : UCS" (selected from a dropdown). Below these is a "Start" button. On the right, a list box contains the words "wine", "cine", "cire", and "core". Below the list box, the following statistics are displayed: "length : 5 word", "elapsed : 20 ms", and "visited node : 83134 node".

4.2 wine - cork (GBFS)



The screenshot shows a window titled "Word Ladder". On the left, there are input fields for "Start : wine", "End : cork", and "Algorithm : GBFS" (selected from a dropdown). Below these is a "Start" button. On the right, a list box contains the words "wine", "wink", "wonk", and "work". Below the list box, the following statistics are displayed: "length : 5 word", "elapsed : 0 ms", and "visited node : 67 node".

4.3 wine - cork (A*)

Word Ladder

Start :

End :

Algorithm :

wine
wire
wore
work

length : 5 word
elapsed : 0 ms
visited node : 154 node

4.4 rhyme - chirp (UCS)

Word Ladder

Start :

End :

Algorithm :

rhyme
chyme
chime
chimp

length : 5 word
elapsed : 1 ms
visited node : 132 node

4.5 rhyme - chirp (GBFS)

Word Ladder

Start :

End :

Algorithm :

rhyme
chyme
chime
chirr

length : 5 word
elapsed : 0 ms
visited node : 22 node

4.6 rhyme - chirp (A*)

Word Ladder

Start :

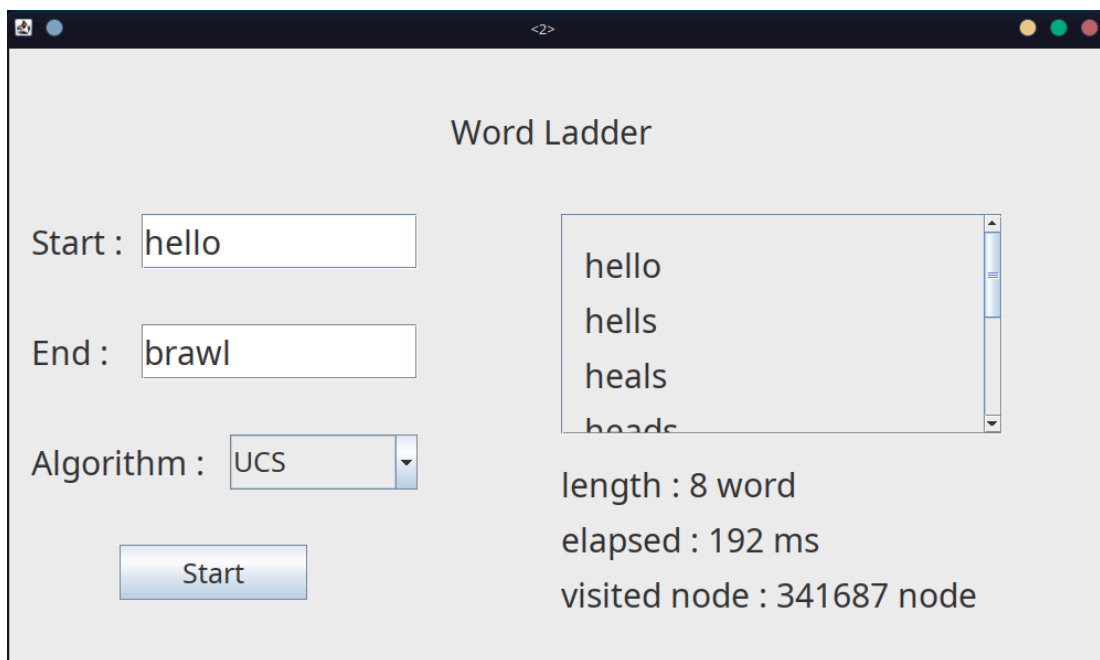
End :

Algorithm :

rhyme
chyme
chime
chirr

length : 5 word
elapsed : 0 ms
visited node : 22 node

4.7 hello - brawl (UCS)



Word Ladder

Start :

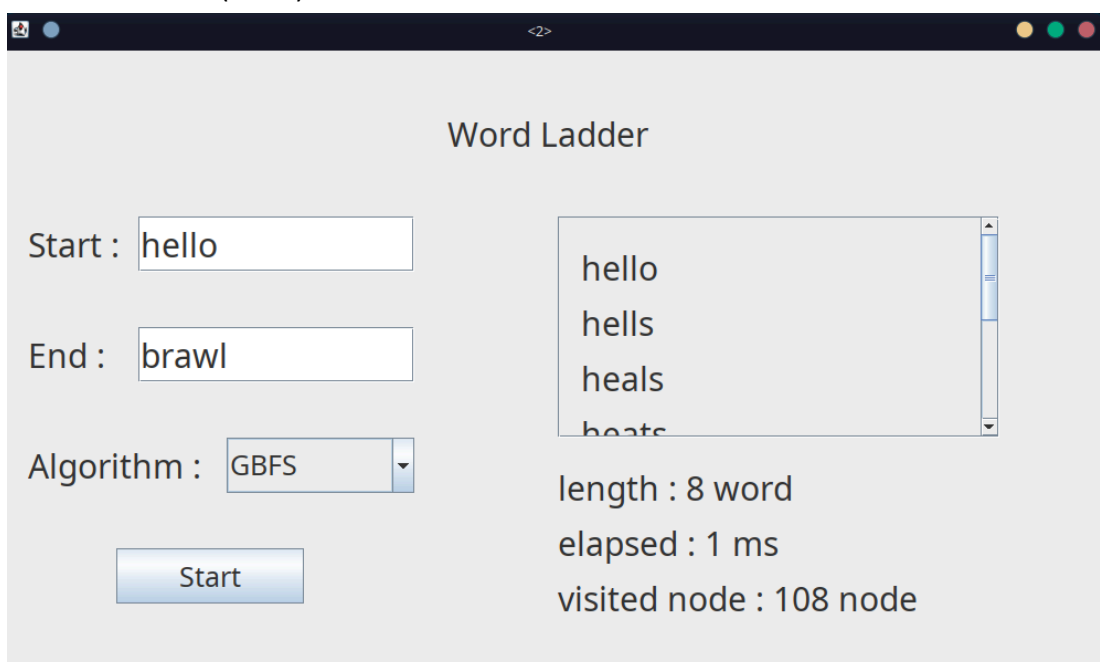
End :

Algorithm :

hello
hells
heals
beats

length : 8 word
elapsed : 192 ms
visited node : 341687 node

4.8 hello - brawl (GBFS)



Word Ladder

Start :

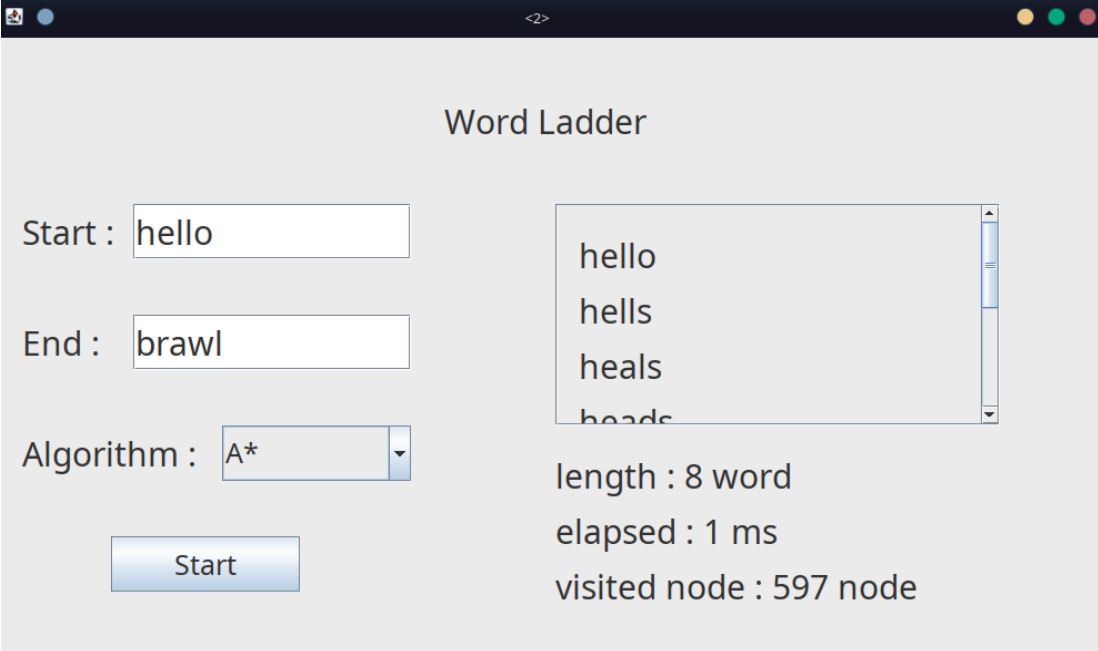
End :

Algorithm :

hello
hells
heals
beats

length : 8 word
elapsed : 1 ms
visited node : 108 node

4.9 hello - brawl (A*)



Word Ladder

Start :

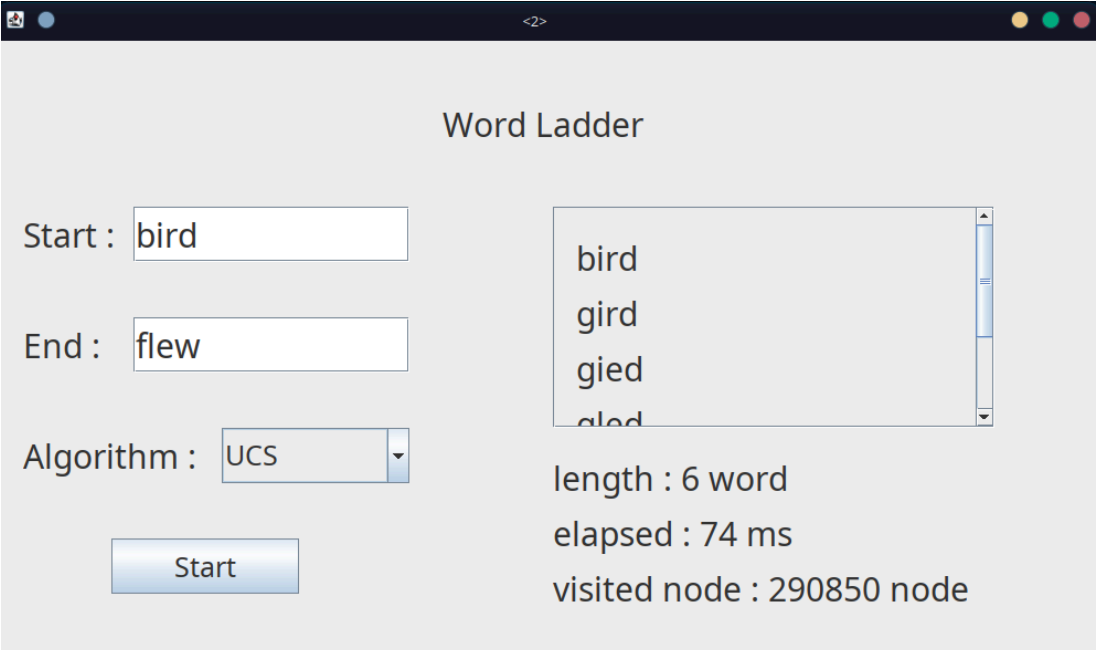
End :

Algorithm :

hello
hells
heals
heads

length : 8 word
elapsed : 1 ms
visited node : 597 node

4.10.bird - flew (UCS)



Word Ladder

Start :

End :

Algorithm :

bird
gird
gied
aled

length : 6 word
elapsed : 74 ms
visited node : 290850 node

4.11.bird - flew (GBFS)

Word Ladder

Start :

End :

Algorithm :

bird
birr
bier
beer

length : 8 word
elapsed : 0 ms
visited node : 68 node

4.12.bird - flew (A*)

Word Ladder

Start :

End :

Algorithm :

bird
gird
gied
aled

length : 6 word
elapsed : 1 ms
visited node : 167 node

4.13.tooth - fairy (UCS)

Word Ladder

Start :

End :

Algorithm :

tooth
toots
toits
toile

length : 8 word
elapsed : 130 ms
visited node : 279586 node

4.14.tooth - fairy (GBFS)

Word Ladder

Start :

End :

Algorithm :

tooth
toots
toits
toile

length : 8 word
elapsed : 0 ms
visited node : 78 node

4.15.tooth - fairy (A*)

Word Ladder

Start :

End :

Algorithm :

tooth
toots
toits
toile

length : 8 word
elapsed : 1 ms
visited node : 410 node

4.16.black - mamba (UCS)

Word Ladder

Start :

End :

Algorithm :

black
block
brock
crook

length : 13 word
elapsed : 728 ms
visited node : 1327598 node

4.17.black - mamba (GBFS)

Word Ladder

Start :

End :

Algorithm :

No Solution

length : 1 word
elapsed : 0 ms
visited node : 23 node

4.18.black - mamba (A*)

Word Ladder

Start :

End :

Algorithm :

black
blank
plank
plane

length : 13 word
elapsed : 14 ms
visited node : 15113 node

Bab 5

Analisis Perbandingan solusi UCS, Greedy Best First Search, dan A*

Dari hasil tangkapan layar di bab 4 kita bisa menyimpulkan bahwa algoritma dengan optimalitas paling tinggi adalah Greedy Best First Search karena di semua test case, GBFS membutuhkan waktu dan memori paling sedikit (waktu dari elapsed time dan memori dari visited node). Namun pada test case nomor 11 bisa kita lihat bahwa GBFS tidak berhasil menemukan solusi optimal, tidak seperti UCS dan A* yang pada test case 10 dan 12 berhasil menemukan solusi paling optimal. Bahkan pada test case 17 algoritma GBFS tidak berhasil menemukan solusi apapun. Sehingga meskipun paling optimal, algoritma ini tidak dapat diandalkan.

Sebaliknya algoritma A* memiliki optimalitas yang sedikit lebih bagus dari GBFS karena membutuhkan waktu yang mirip dengan GBFS dan membutuhkan memori sedikit lebih banyak dari GBFS. Akan tetapi algoritma A* adalah algoritma yang komplit sehingga pasti dapat menemukan solusi optimal apabila terdapat solusi optimal dan jumlah simpul tidak tak hingga.

Adapun algoritma dengan optimalitas yang paling rendah adalah algoritma UCS yang membutuhkan waktu dan memori paling besar daripada kedua algoritma sebelumnya. Bahkan waktu yang dibutuhkan bisa 200 kali lipat lebih lama seperti pada test case 7 apabila dibandingkan dengan test case 6 dan 8. Selain itu UCS juga membutuhkan memori yang lumayan signifikan daripada dua algoritma sebelumnya. Namun seperti A* algoritma ini bersifat komplit sehingga pasti menemukan solusi optimal.

Pranala Repository Github

https://github.com/RiciTrisnaP/Tucil3_13522026

Referensi

<https://wordwormdormdork.com>

<https://stackoverflow.com/questions/23722011/efficient-data-structure-that-checks-for-existence-of-string>

<https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html>

<https://stackoverflow.com/questions/1097366/java-swing-revalidate-vs-repaint>

<https://docs.oracle.com/javase/8/docs/api/java/util/Map.html>

<https://www.geeksforgeeks.org/map-interface-java-examples/>

<https://stackoverflow.com/questions/692569/how-can-i-count-the-time-it-takes-a-function-to-complete-in-java>

<https://stackoverflow.com/questions/30632444/make-a-jpanel-change-after-actionlistener>

<https://www.geeksforgeeks.org/hashmap-get-method-in-java/>

<https://www.youtube.com/watch?v=ObVnyA8ar6Q>

<https://www.youtube.com/watch?v=QU9t-TqTnsg&list=PLLQOQa4PqY-v6SUSWSjWA4HmtjQ4HJ3QU>

<https://www.trivusi.web.id/2022/10/apa-itu-algoritma-uniform-c>

<https://stackoverflow.com/questions/5258159/how-to-make-an-executable-jar-file>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>