**Laporan Tugas Kecil 3 IF2211 Strategi Algoritma**

**Semester IV tahun 2023/2024**

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

****

**Disusun oleh:**

**Eduardus Alvito Kristiadi (13522004)**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2024**

**BAB I**

**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. Ilustrasi serta aturan permainan adalah tercantum pada link: <https://wordwormdormdork.com/> Spesifikasi program adalah sebagai berikut:

* Program dibuat dalam bahasa Java berbasis CLI (Command Line Interface) – bonus jika menggunakan GUI – yang dapat menemukan solusi permainan word ladder menggunakan algoritma UCS, Greedy Best First Search, dan A\*.
* Kata-kata yang dapat dimasukkan harus berbahasa Inggris. Cara kalian melakukan validasi sebuah kata dibebaskan, selama kata-kata tersebut benar terdapat pada dictionary dan proses validasi tersebut tidak memakan waktu yang terlalu lama.
* Tugas wajib dikerjakan secara individu.
* Input :

Format masukan dibebaskan, dengan catatan dijelaskan pada README dan laporan. Komponen yang perlu menjadi masukan yaitu.

1. Start word dan end word. Program harus bisa menangani berbagai panjang kata (tidak hanya kata dengan 4 huruf saja seperti Gambar 1).
2. Pilihan algoritma yang digunakan (UCS, Greedy Best First Search, atau A\*)

* Output :

Berikut adalah luaran dari program yang diekspektasikan.

1. Path yang dihasilkan dari start word ke end word (cukup 1 path saja)
2. Banyaknya node yang dikunjungi
3. Waktu eksekusi program

**BAB II**

**ALGORITMA UCS, DAN A\* SECARA UMUM**

**2.1 Algoritma Uniform Cost Search (UCS)**

Algoritma Uniform Cost Search (UCS) adalah algoritma pencarian lintasan terpendek di antara dua simpul pada sebuah graf atau peta dengan menghitung biaya(*cost*) terkecil untuk mencapai simpul tujuan dari simpul awal. Langkah-langkah algoritma UCS adalah sebagai berikut:

1. Tentukan simpul awal dan simpul tujuan.

2. Buat sebuah *priority queue* yang akan digunakan untuk menyimpan simpul-simpul yang akan dikunjungi beserta dengan lintasan yang digunakan untuk mencapai simpul tersebut dan biayanya.

3. Selama *priority queue* tidak kosong, kunjungi simpul yang memiliki lintasan dengan biaya

terendah.

4. Periksa apakah simpul yang sedang dikunjungi adalah simpul tujuan atau bukan. Jika ya, kembalikan lintasan tersebut.

5. Jika simpul yang sedang dikunjungi (*v*) bukan simpul tujuan dan belum pernah dikunjungi

sebelumnya, tambahkan tetangga-tetangganya ke dalam *queue* yang telah dibuat. Lintasan dari masing-masing tetangga adalah lintasan yang ditempuh untuk mencapai simpul *v* ditambahkan dengan dirinya sendiri. *Cost* dihitung dari *cost* untuk mencapai simpul *v* ditambah dengan *cost* dari *v* ke tetangga tersebut.

6. Ulangi langkah 3-5 hingga *priority queue* kosong atau ditemukan solusi. Jika *priority queue* kosong, artinya tidak ada lintasan dari simpul awal ke simpul tujuan.

**2.2 Algoritma A\***

Algoritma A\* adalah algoritma pencarian jalur atau *pathfinding* yang digunakan untuk

menemukan jalur terpendek antara dua simpul atau titik pada sebuah graf atau peta. Dalam masalah

yang telah dijelaskan pada Bab I, langkah-langkah algoritma A\* adalah sebagai berikut:

1. Tentukan simpul asal dan simpul tujuan.
2. Cari terlebih dahulu nilai heuristik tiap simpul untuk algoritma A\*, yang dalam persoalan ini berupa jarak tegak lurus dari simpul ke simpul tujuan, simpan jarak tersebut sebagai atribut simpul.
3. Buat sebuah *priority queue* yang akan digunakan untuk menyimpan simpul-simpul yan akan dikunjungi beserta lintasan yang dilalui untuk mencapai simpul tersebut dan biayanya (dalam hal ini biaya adalah jarak), serta prediksi biaya untuk mencapai simpul tujuan dari simpul saat ini. *Priority queue* akan terurut menaik berdasarkan prediksi biaya yang diperlukan untuk mencapai simpul tujuan dari simpul saat ini, prediksi biaya diperoleh dari biaya sebenarnya yang telah terakumulasi hingga mencapai simpul saat ini ditambah dengan nilai heuristik simpul tersebut.
4. Tambahkan simpul asal ke dalam *priority queue*.
5. Selama *priority queue* tidak kosong, kunjungi simpul pertama pada *priority queue* untuk diperiksa. Periksa apakah simpul yang sedang dikunjungi adalah simpul tujuan atau bukan. Jika, ya kembalikan biaya tempuh sebenarnya dari simpul tersebut sebagai jawaban.
6. Jika simpul yang sedang dikunjungi (*v*) bukan simpul tujuan, periksa apakah tetangganya sudah pernah dikunjungi atau dilewati dalam lintasan untuk mencapai *v*, jika belum maka tambahkan tetangganya ke dalam *priority queue* yang telah dibuat. Lintasan darimasing-masing tetangga adalah lintasan yang ditempuh untuk mencapai simpul *v* ditambahkan dengan dirinya sendiri (tetangga itu sendiri). *Cost* tetangga adalah *cost* untuk mencapai simpul *v* ditambah dengan *cost* dari *v* ke tetangga tersebut. Prediksi biaya dihitung dari *cost* dirinya sendiri ditambah dengan nilai heuristik dirinya sendiri.
7. Ulangi langkah 5-7 hingga *priority queue* kosong atau ditemukan solusi. Jika *priority queue* kosong, artinya tidak ada lintasan dari simpul awal ke simpul tujuan.
   1. **Algoritma GBFS**

Algoritma Greedy Best-First Search (GBFS) adalah metode pencarian yang menggunakan prinsip kegirangan (greediness) untuk mencapai solusi. Algoritma ini merupakan salah satu bentuk algoritma pencarian berinformasi yang mengandalkan heuristik atau pengetahuan tambahan mengenai masalah untuk memperkirakan jarak dari keadaan saat ini ke tujuan akhir. Tujuannya adalah untuk menemukan jalur terpendek atau solusi tercepat dari suatu titik awal ke titik tujuan dengan mengikuti jalur yang tampak paling menjanjikan pada setiap langkahnya. Terdapat fungsi heuristic dalam algoritma ini. Fungsi heuristik adalah kunci dari algoritma pencarian berinformasi seperti GBFS. Fungsi ini mengestimasi biaya (atau jarak) terkecil dari sebuah node ke tujuan akhir. Penentuan fungsi heuristik yang baik sangat penting karena kualitas dan efektivitas pencarian sangat bergantung pada akurasi perkiraan ini.

* 1. **f(n) dan g(n)**
  2. **Apakah heuristik yang digunakan pada algoritma A\* *admissible*?**
  3. **Pada kasus *word ladder*, apakah algoritma UCS sama dengan BFS?**

Dalam kasus ini, penggunaan UCS mirip dengan BFS karena tidak ada pembobotan khusus dalam setiap langkah yang diambil atau dalam kasus ini yaitu tiap node yang dilewati. Pada dasarnya BFS merupakan kasus khusus dari UCS di mana BFS memiliki bobot yang seragam yaitu 1. Dalam kasus ini pun begitu, tiap node yang dilewati mempunyai bobot 1.

* 1. **Secara teoritis, apakah algoritma A\* lebih efisien dibandingkan dengan algoritma UCS pada kasus *word ladder*?**

Iya, A\* lebih efisien karena dia membandingkan bobot node dari jarak menuju kata tujuan dan jarak dari node awal sehingga perhitungannya lebih efisien saat program dijalankan. A\* tidak perlu melewati node sebanyak UCS.

* 1. **Secara teoritis, apakah algoritma *Greedy Best First Search* menjamin solusi optimal untuk persoalan *word ladder*?**

Tidak, karena bisa saja GBFS terjebak dalam Solusi optimum local yang dalam hal ini juga bisa terjadi saat tidak ada kata ditengah-tengah antara dua node. Jadi tidak menjamin optimum golobal.

**BAB III**

**IMPLEMENTASI PROGRAM**

**DENGAN BAHASA PEMROGRAMAN JAVA**

Algoritma pada Bab II diimplementasikan dengan menggunakan bahasa Java. *Source*

*code* dari program dapat diakses pada <https://github.com/Edvardesu/Tucil3_13522004>

**FORMAT MASUKAN:**

1. Masukkan startword terlebih dahulu dengan huruf kecil semua (case sensitive)

2. Tekan enter

3. Masukkan targetword dengan huruf kecil semua (case sensitive)

4. Tekan enter

5. Masukkan algoritma yang diinginkan (UCS atau GBFS atau A\*) - case sensitive

6. Tekan enter

7. Akan keluar hasil

*Source code* diimplementasikan secara modular dengan beberapa file yang merepresentasikan algoritma tertentu juga main.py yang berperan sebagai program utama.

1. UCS.java

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

1. GBFS.java



A computer screen with many colorful text

Description automatically generated with medium confidence

A screen shot of a computer code

Description automatically generated

1. Astar.java

A computer screen shot of code

Description automatically generated

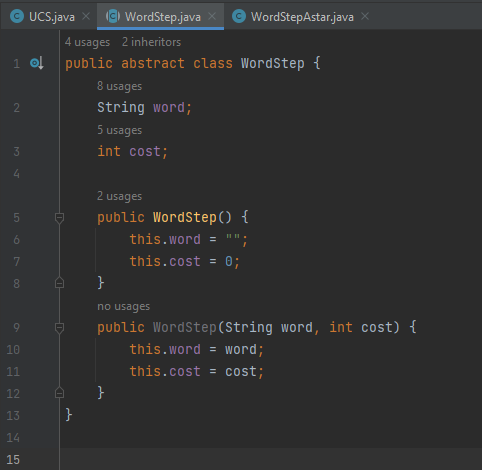
A screenshot of a computer program

Description automatically generated

A computer screen shot of code

Description automatically generated

1. WordStep.java



1. WordStepAstar.java

A screenshot of a computer program

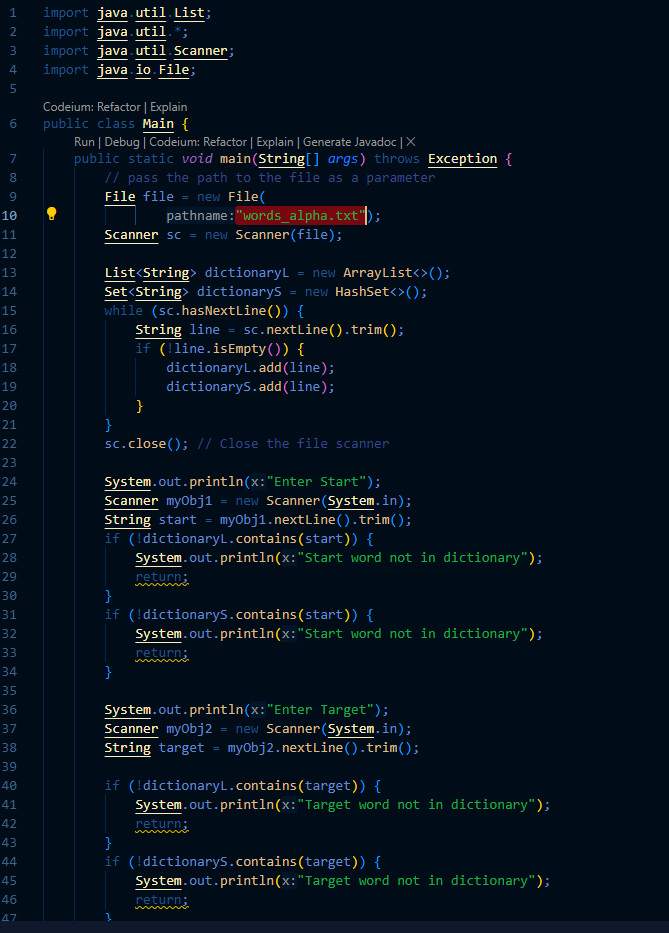
Description automatically generated

1. WordStepUCS.java

A screen shot of a computer program

Description automatically generated

1. Main.java



A screenshot of a computer program

Description automatically generated

**BAB IV**

**EKSPERIMEN**

1. Pengujian 1

Start: Earn

Target: Make

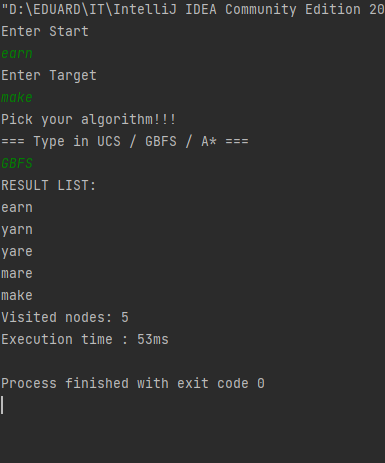
Hasil Pengujian  
UCS:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

GBFS:



A\*:

A screenshot of a computer program

Description automatically generated

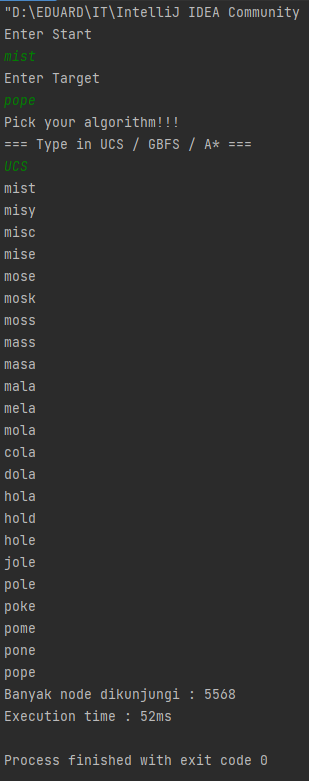
1. Pengujian 2

Start: mist

Target: pope

Hasil Pengujian

UCS:



GBFS:

A screenshot of a computer program

Description automatically generated

A\*:

A screenshot of a computer program

Description automatically generated

1. Pengujian 3

Start: blue  
Target: bird

Hasil Pengujian

UCS:

A screenshot of a computer

Description automatically generated

GBFS:

A screenshot of a computer program

Description automatically generated

A\*:

A screenshot of a computer program

Description automatically generated

1. Pengujian 4

Start: bride  
Target: thumb

Hasil Pengujian

UCS:

A screenshot of a computer

Description automatically generated

GBFS:  
A screenshot of a computer program

Description automatically generated

A\*:

A screenshot of a computer program

Description automatically generated

1. Pengujian 5

Start: plant  
Target: brake

Hasil Pengujian

UCS:

A screenshot of a computer

Description automatically generated

GBFS:

A screenshot of a computer program

Description automatically generated

A\*:

A screenshot of a computer program

Description automatically generated

1. Pengujian 6

Start: bridge

Target: church

Hasil Pengujian

UCS:

A screenshot of a computer program

Description automatically generated

GBFS:

A screenshot of a computer

Description automatically generated

A\*:

A screenshot of a computer program

Description automatically generated

**OPTIMALITAS**

Untuk masalah optimalitas, didapat bahwa A\* lah yang paling optimal karena durasinya paling cepat dan memilik probabilitas lebih tinggi untuk mendapatkan hasil akhir (path found). A star juga memiliki jumlah kata dalam path lebih sedikit dan lebih akurat sehingga meningkatkan optimalitas.

**WAKTU EKSEKUSI**Untuk waktu eksekusi, A\* lah yang seringkali paing cepat. Hal ini dikarenakan A\* menempuh jumlah kata dalam path yang lebih sedikit. Bisa dilihat pula dari hasil pengujian, bahwa mayoritas A\* lah yang paling cepat dalam execution time.

**MEMORI**

Untuk penggunaan memori, yang paling unggul ialah Greedy sebab Greedy memiliki jumlah node dikunjungi lebih sedikit disbanding yang lain. Hal ini pula yang menyebabkan Greedy terkadang tidak mendapatkan hasil akhir (no path found).

**BAB V**

**KESIMPULAN DAN KOMENTAR**

**5.1 Kesimpulan**

Algoritma UCS, GBFS, serta A\* sangat berguna dalam metode pencarian jalur (pathfinding). Ketiga algoritma tersebut dapat dirasakan kegunaanya dalam tugas besar ini yaitu Word Ladder. Dalam Word Ladder terdapat perbedaan antara ketiga algoritma tersebut dalam hal g(x) dan h(x). Masing-masing memiliki kelebihan dan kekurangan.

**5.2 Komentar**

Pengerjaan tugas besar ini terlalu terburu-buru dan sangat mendekati deadline. Dapat disarankan untuk ke depannya lebih memanfaatkan waktu dengan baik agar tugas besar menjadi lebih sempurna hasilnya.

**BAB VI**

**LAMPIRAN**

Link Repository: <https://github.com/Edvardesu/Tucil3_13522004>

Tabel Tugas:

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