**Laporan Tugas Kecil 3 IF2211 Strategi Algoritma**

**Semester II tahun 2023/2024**

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

Disusun oleh: Jimly Nur Arif (13522123)



**PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG 2023**

# Daftar Isi

[Daftar Isi ii](#_Toc165963613)

[Penjelasan Permainan 1](#_Toc165963614)

[Penjelasan Algoritma 1](#_Toc165963615)

[A\* Algorithm 1](#_Toc165963616)

[Langkah-langkah A\* algorithm 2](#_Toc165963617)

[Greedy Best First Search 2](#_Toc165963618)

[Langkah-langkah GBFS 2](#_Toc165963619)

[Uniform Cost Search 3](#_Toc165963620)

[Langkah-langkah UCS 3](#_Toc165963621)

[Source Code 3](#_Toc165963622)

[Modul WordLadderAStar 4](#_Toc165963623)

[Modul WordLadderGBFS 5](#_Toc165963624)

[Modul WordLadderUCS 6](#_Toc165963625)

[Modul print 8](#_Toc165963626)

[Modul Hamming Distance 9](#_Toc165963627)

[Modul Main 9](#_Toc165963628)

[Test Case 11](#_Toc165963629)

[Uji Coba 14](#_Toc165963630)

[Analisis 16](#_Toc165963631)

[Lampiran 17](#_Toc165963632)

[Tabel Checklist 17](#_Toc165963633)

[Pranala repository GitHub: 17](#_Toc165963634)

# Penjelasan Permainan

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.

Berikut adalah aturan dasar untuk bermain Word Ladder:

Kata Awal dan Akhir: Kata awal dan akhir harus kata nyata.

Perubahan Satu Huruf: Setiap kata dalam tangga harus juga merupakan kata nyata. Anda hanya dapat mengubah satu huruf pada kata sebelumnya untuk membuat kata baru dalam tangga.

Tangga Kata: Anda harus mengubah kata secara bertahap hingga mencapai kata akhir dalam tangga.

Tujuan: Coba selesaikan tangga secepat mungkin dengan mengubah kata-kata dengan benar.

Skor: Skor dihitung berdasarkan waktu dan jumlah tebakan yang salah. Usahakan untuk mendapatkan skor sekecil mungkin.

Misalnya, jika kita ingin mengubah kata “CAT” menjadi “DOG,” kita bisa mengikuti tangga berikut:

CAT

COT

DOT

DOG

# Penjelasan Algoritma

## A\* Algorithm

Algoritma A (A-Star)\* adalah algoritma pencarian yang digunakan dalam pemrograman komputer dan kecerdasan buatan untuk mencari jalur terpendek atau solusi optimal antara dua titik dalam graf atau ruang pencarian. Algoritma ini memadukan teknik pencarian breadth-first (pencarian melebar) dan heuristic (pemilihan berdasarkan perkiraan) untuk mencapai keseimbangan antara kecepatan dan optimalitas

Proses A\* adalah

Inisialisasi: Tentukan simpul awal dan simpul tujuan. Inisialisasi himpunan simpul terbuka (open set) yang berisi simpul yang akan diperiksa dan himpunan simpul tertutup (closed set) yang berisi simpul-simpul yang telah diperiksa.

Hitung biaya awal: Tentukan biaya awal untuk mencapai simpul awal (biasanya 0).

Evaluasi simpul awal: Hitung perkiraan biaya sisa (estimated-cost-to-go) dari simpul awal ke simpul tujuan menggunakan fungsi heuristik. Hitung perkiraan biaya total (estimated-total-cost) dari simpul awal ke simpul tujuan dengan menambahkan biaya sejauh ini dan perkiraan biaya sisa.

### Langkah-langkah A\* algorithm

Inisialisasi dua list tipe Node (misalnya openList dan closeList)

Tambahkan Node src di openList.

Hingga openList tidak kosong, lakukan hal berikut

Temukan node di openList dengan nilai f terkecil (katakanlah x)

Hapus x dari openList

Temukan dan tambahkan semua penerus x dan tambahkan ke openList.

Untuk setiap penerus (katakanlah y) lakukan hal berikut -

Jika y adalah tujuannya maka hentikan pencarian.

Jika tidak, hitung g dan h untuk y. y.g = x.g + jarak antara y dan x (dalam kasus word ladder adalah 1) y.h = jarak antara tujuan

Jika sebuah node dengan posisi yang sama dengan y dan f yang lebih rendah ada di openList, lewati penerus ini.

Lain Jika sebuah node dengan posisi yang sama dengan y ada di CloseList dengan f lebih kecil dari y.f lewati penerus ini.

Jika tidak, tambahkan node (y) ke openList.

Masukkan x ke closedList.

## Greedy Best First Search

Algoritma Greedy Best First Search adalah algoritma pencarian yang menggunakan pendekatan greedy untuk mencari solusi. Tujuannya adalah untuk mencari solusi optimal dengan memilih simpul yang paling menjanjikan pada setiap langkah pencarian. Algoritma ini mirip dengan algoritma Best First Search, tetapi memperkuat prinsip greedy dengan hanya mempertimbangkan nilai heuristik dari simpul untuk memilih simpul berikutnya.

### Langkah-langkah GBFS

Tentukan dua list kosong (openList dan closeList).

Masukkan src di openList.

Ulangi saat openList tidak kosong, dan lakukan hal berikut -

Temukan node dengan nilai h terkecil yang ada di openList (node).

Hapus node dari openList, dan masukkan ke closeList.

Periksa semua node yang berdekatan dari node tersebut, apakah ada di antara mereka yang sama dengan Destination Word. Jika ya, maka hentikan proses pencarian karena kita sudah sampai di tujuan.

Jika tidak, temukan semua node yang berdekatan yang tidak ada di openList atau closeList dan masukkan ke dalam openList.

Jika openList sudah kosong, berarti tidak ada kemungkinan jalur antara src dan dest.

## Uniform Cost Search

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian graf yang digunakan untuk mencari jalur yang memiliki biaya terendah dari simpul awal ke simpul tujuan dalam sebuah graf dengan bobot pada setiap lintasan.

Time complexity nya adalah   
A math symbols and symbols

Description automatically generated with medium confidence

Space Complexity nya adalah  
A math symbols with numbers

Description automatically generated with medium confidence

### Langkah-langkah UCS

Mulai dari simpul awal dengan **cost** 0.

Ambil simpul dengan biaya terendah dari **prioqueue**.

Periksa apakah itu simpul tujuan. Jika ya, selesai.

Perluas simpul dengan menambahkan tetangga-tetangganya ke antrian prioritas.

Jika simpul tetangga sudah dieksplorasi, perbarui **costnya** jika yang baru lebih rendah.

Ulangi langkah-langkah di atas sampai simpul tujuan ditemukan atau antrian prioritas kosong.

Jika simpul tujuan ditemukan, kembalikan jalur terpendek.

Jika tidak, tidak ada jalur yang memungkinkan dari simpul awal ke tujuan.

# Source Code

Tugas ini dilaksanakan dengan menggunakan Bahasa pemrograman *Java* dengan library yang diperlukan untuk utility dan library untuk mengakses file word.txt

## Modul WordLadderAStar

package src;

import java.util.\*;

public class WordLadderAStar {

    static class Node {

        String word;

        int f;

        int g;

        Node parent;

        Node(String word, int g, int h, Node parent) {

            this.word = word;

            this.g = g;

            this.f = g + h;

            this.parent = parent;

        }

    }

    static List<String> findPath(String start, String end, Set<String> wordList) {

        PriorityQueue<Node> openList = new PriorityQueue<>(Comparator.comparingInt(node -> node.f));

        Set<String> closedList = new HashSet<>();

        Map<String, Node> parentMap = new HashMap<>();

        Node startNode = new Node(start, 0, hamming.hammingDistance(start, end), null);

        openList.add(startNode);

        while (!openList.isEmpty()) {

            Node current = openList.poll();

            closedList.add(current.word);

            if (current.word.equals(end)) {

                List<String> path = new ArrayList<>();

                while (current != null) {

                    path.add(0, current.word);

                    current = current.parent;

                }

                return path;

            }

            for (String neighbor : wordList) {

                if (!closedList.contains(neighbor) && hamming.hammingDistance(current.word, neighbor) == 1) {

                    int g = current.g + 1;

                    int h = hamming.hammingDistance(neighbor, end);

                    Node newNode = new Node(neighbor, g, h, current);

                    if (!openList.contains(newNode) || g < newNode.g) {

                        openList.add(newNode);

                        parentMap.put(neighbor, current);

                    }

                }

            }

        }

        return null;

    }

}

Modul ini digunakan untuk logika Word Ladder dengan Algoritma A\* Search

## Modul WordLadderGBFS

package src;

import java.util.\*;

public class WordLadderGBFS {

    static List<String> greedyBestFirstSearch(String start, String end, Set<String> wordList) {

        List<String> ladder = new ArrayList<>();

        PriorityQueue<Node> queue = new PriorityQueue<>(Comparator.comparingInt(a -> a.distance));

        Set<String> visited = new HashSet<>();

        queue.offer(new Node(start, 0));

        while (!queue.isEmpty()) {

            Node current = queue.poll();

            String currentWord = current.word;

            if (currentWord.equals(end)) {

                // ladder.add(end);

                while (current != null) {

                    ladder.add(0, current.word);

                    current = current.parent;

                }

                return ladder;

            }

            visited.add(currentWord);

            for (String word : wordList) {

                if (!visited.contains(word) && hamming.hammingDistance(word, currentWord) == 1) {

                    queue.offer(new Node(word, hamming.hammingDistance(word, end), current));

                }

            }

        }

        return ladder;

    }

    static class Node {

        String word;

        int distance;

        Node parent;

        Node(String word, int distance) {

            this.word = word;

            this.distance = distance;

        }

        Node(String word, int distance, Node parent) {

            this.word = word;

            this.distance = distance;

            this.parent = parent;

        }

    }

}

Modul ini digunakan untuk logika Word Ladder dengan Algoritma Greedy Best First Search

## Modul WordLadderUCS

package src;

import java.util.\*;

public class WordLadderUCS {

    static class Node {

        String word;

        int cost;

        Node parent;

        Node(String word, int cost, Node parent) {

            this.word = word;

            this.cost = cost;

            this.parent = parent;

        }

    }

    static List<String> findWordLadder(String startWord, String endWord, Set<String> wordList) {

        Queue<Node> queue = new PriorityQueue<>(Comparator.comparingInt(node -> node.cost));

        Set<String> visited = new HashSet<>();

        queue.offer(new Node(startWord, 0, null));

        visited.add(startWord);

        while (!queue.isEmpty()) {

            Node current = queue.poll();

            if (current.word.equals(endWord)) {

                return constructPath(current);

            }

            List<String> neighbors = getNeighbors(current.word, wordList);

            for (String neighbor : neighbors) {

                if (!visited.contains(neighbor)) {

                    int cost = current.cost + 1; // Assuming uniform cost

                    queue.offer(new Node(neighbor, cost, current));

                    visited.add(neighbor);

                }

            }

        }

        return null; // No ladder found

    }

    static List<String> getNeighbors(String word, Set<String> wordList) {

        List<String> neighbors = new ArrayList<>();

        char[] chars = word.toCharArray();

        for (int i = 0; i < chars.length; i++) {

            char originalChar = chars[i];

            for (char c = 'a'; c <= 'z'; c++) {

                if (c != originalChar) {

                    chars[i] = c;

                    String newWord = new String(chars);

                    if (wordList.contains(newWord)) {

                        neighbors.add(newWord);

                    }

                }

            }

            chars[i] = originalChar;

        }

        return neighbors;

    }

    static List<String> constructPath(Node endNode) {

        List<String> ladder = new ArrayList<>();

        Node current = endNode;

        while (current != null) {

            ladder.add(0, current.word);

            current = current.parent;

        }

        return ladder;

    }

}

Modul ini digunakan untuk logika Word Ladder dengan Algoritma Uniform Cost Search

## Modul print

package src;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashSet;

import java.util.Set;

public class print {

    static Set<String> readWordListFromFile(String filePath, String start) {

        Set<String> wordList = new HashSet<>();

        int startLength = start.length();

        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

            String line;

            while ((line = reader.readLine()) != null) {

                line = line.trim();

                if (line.length() == startLength) {

                    wordList.add(line);

                }

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

        return wordList;

    }

}

Modul ini digunakan untuk mengambil kata-kata yang panjangnya sesuai dengan masukan Source Word dari dictionary yakni word.txt

## Modul Hamming Distance

package src;

public class hamming {

    static int hammingDistance(String word1, String word2) {

        int count = 0;

        for (int i = 0; i < word1.length(); i++) {

            if (word1.charAt(i) != word2.charAt(i)) {

                count++;

            }

        }

        return count;

    }

}

Modul Hamming Distance digunakan untuk penentu nilai pada pririty queue

## Modul Main

package src;

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        System.out.println("Welcome to Word Ladder!");

        System.out.println("Please Select Algorithm: ");

        System.out.println("1. A\* Algorithm");

        System.out.println("2. Greedy Best First Search Algorithm");

        System.out.println("3. Uniform Cost Search Algorithm");

        Scanner scanner = new Scanner(System.in);

        int choice = scanner.nextInt();

        System.out.println("Masukkan Source Word: ");

        String start = scanner.next();

        String filePath = "word.txt";

        Set<String> wordList = print.readWordListFromFile(filePath, start);

        while (!wordList.contains(start)) {

            System.out.println("Start word not found in word.txt file");

            System.out.println("Masukkan Start Word: ");

            start = scanner.next();

        }

        System.out.println("Masukkan Destination Word: ");

        String end = scanner.next();

        while (!wordList.contains(end)) {

            System.out.println("Destination word not found in word.txt file");

            System.out.println("Masukkan Destination Word: ");

            end = scanner.next();

        }

        switch (choice) {

            case 1:

                runAStar(start, end, wordList);

                break;

            case 2:

                runGBFS(start, end, wordList);

                break;

            case 3:

                runUCS(start, end, wordList);

                break;

            default:

                System.out.println("Invalid choice.");

        }

        scanner.close();

    }

    public static void runAStar(String start, String end, Set<String> wordList) {

        List<String> ladder = WordLadderAStar.findPath(start, end, wordList);

        if (ladder != null) {

            System.out.println("Shortest path from " + start + " to " + end + ": " + ladder);

        } else {

            System.out.println("No path found from " + start + " to " + end);

        }

    }

    public static void runGBFS(String start, String end, Set<String> wordList) {

        List<String> ladder = WordLadderGBFS.greedyBestFirstSearch(start, end, wordList);

        if (ladder != null) {

            System.out.println("Shortest path from " + start + " to " + end + ": " + ladder);

        } else {

            System.out.println("No path found from " + start + " to " + end);

        }

    }

    public static void runUCS(String start, String end, Set<String> wordList) {

        List<String> ladder = WordLadderUCS.findWordLadder(start, end, wordList);

        if (ladder != null) {

            System.out.println("Shortest path from " + start + " to " + end + ": " + ladder);

        } else {

            System.out.println("No path found from " + start + " to " + end);

        }

    }

}

Modul ini merupakan modul utama yang menjalankan seluruh file

# Test Case

A screenshot of a computer

Description automatically generated

Fig. 1 Test case earn-make

A screenshot of a computer program

Description automatically generated

Fig. 2 Test Case belt-loop

A screenshot of a computer program

Description automatically generated

Fig. 3 Test Case frown-smile

A computer screen shot of a black screen

Description automatically generated

Fig. 4 Test Case door-jamb

A screenshot of a computer program

Description automatically generated

Fig. 4 Test Case team-crew

A screenshot of a computer program

Description automatically generated

Fig. 5 Test Case take-away

A screenshot of a computer program

Description automatically generated

Fig. 6 Test Case grass-roots

# Uji Coba

A computer screen shot of a black and white screen

Description automatically generated

Fig. 2a Screenshoot CLI u/ input normal

A computer screen shot of a black and white screen

Description automatically generated

Fig. 2b masukan tidak ada di kamus

A black background with white text

Description automatically generated

Fig. 2c Panjang Destination word tidak sama dengan Source Word

# Analisis

Berdasarkan data pengujian Test Case dapat disimpulkan:

1. GBFS cenderung lebih cepat dan mengreturn path yang lebih panjang (tidak optimal)
2. A\* cenderung lebih efisien dalam menjelajahi node dan mereturn path yang optimal
3. UCS optimal karena menggunakan cost actual (non heuristic), namun cenderung lebih lama
4. F(n) adalah nilai total dari cost actual ditambah nilai heuristic dengan kata lain f(n) = g(n) + h(n).
5. G(n) adalah biaya actual untuk menuju node n dari simpul awal.
6. Heuristik yang saya gunakan Admissible. Heuristik admissible adalah heuristik yang tidak melebih biaya sebenarnya (biaya sejati) dari simpul tertentu ke tujuan. Jika heuristik tidak admissible, tidak ada jaminan bahwa A\* akan menemukan solusi optimal. meskipun A\* mungkin menemukan solusi, solusi tersebut tidak dijamin menjadi yang terbaik atau optimal.
7. Karena cost antar simpul bertetangga sama, maka dalam kode word ladder saya, UCS dan BFS sama saja.
8. Dalam hal return path, A\* dan UCS sama saja efisiennya
9. Dalam hal memori dan waktu, A\* selalu sama atau lebih optimal dari UCS.
10. Secara teoritis, greedy best first search tidak menjamin Solusi optimal.
11. Memori yang dibutuhkan dapat diinterpretasikan dari banyak node yang dikunjungi pada CLI

# Lampiran

## Tabel Checklist

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

## Pranala repository GitHub:

github.com/jimlynurarif/Tucil3\_13522123