**TUGAS KECIL 3**

**STRATEGI ALGORITMA**

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


**Nama:**

**Hafizh Hananta Akbari**

**(13522132)**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2023/2024**

# BAB I
# ANALISIS ALGORITMA

## 1.1 Algoritma

### 1.1.1 Algoritma UCS

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian yang menggunakan konsep bobot. Bobot pada algoritma ini merujuk ke biaya-biaya jalur dari suatu simpul ke simpul lainnya. Algoritma ini melakukan pencarian secara iteratif untuk menjelajahi simpul-simpul dan memperbarui nilai bobot pada setiap iterasinya. Setelah itu, algoritma ini mengambil jalur dengan bobot yang terkecil. Jalur dengan bobot terkecil dari simpul akar ke n disebut sebagai g(n).

Algoritma ini memiliki kemiripan dengan BFS (Breadth-First Search) dalam peninjauan nodenya. Keduanya meninjau node-node secara bertahap. Namun, UCS memperhatikan bobot pada setiap iterasinya sehingga mampu menghasilkan hasil yang berbeda dengan BFS.

### 1.1.2 Algoritma Greedy Best First Search

Algoritma Greedy Best First Search adalah algoritma pencarian yang menggunakan nilai heuristik lokal atau estimasi jarak untuk memilih jalur yang diambil. Pengambilan nilai heuristik pada algoritma ini berarti algoritma ini menggunakan estimasi atau perkiraan jarak dibanding dengan menggunakan nilai jarak yang sebenarnya. Hasil yang didapat diambil dengan menggunakan nilai heuristik yang didapat dimana jalur dari n ke simpul tujuan disebut sebagai f(n). Karena itu, hasil yang didapat cenderung cukup cepat didapat meskipun terkadang kurang optimal sehingga algoritma ini bergantung dengan kasus untuk dapat digunakan dengan baik.

### 1.1.3 Algoritma A*

Algoritma A* (dipanggil sebagai A-star) adalah algoritma pencarian yang menggabungkan algoritma Djikstra dengan Greedy Best First Search. Hal ini berarti algoritma ini mencari simpul dengan bobot jalur yang terkecil beserta dengan menilai estimasi atau perkiraan agar dapat mengambil jalur yang kemungkinan besar menghasilkan jalur yang terpendek. Secara teoritis, algoritma A* bisa lebih efisien dibanding algoritma UCS. Hal ini karena algoritma A* meninjau nilai heuristik untuk mempercepat proses. Karena menggunakan nilai bobot yang sebenarnya dan nilai estimasi atau heuristik, algoritma ini bisa dianggap sebagai algoritma yang cenderung dapat memberikan hasil yang optimal. Pada kasus dimana nilai heuristik lebih kecil dibanding nilai yang

sebenarnya, algoritma ini memenuhi sifat admissible yang dimiliki dan akan selalu memberikan solusi yang optimal.

## 1.2 Implementasi Algoritma

### 1.2.1 Algoritma UCS

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian yang mencari bobot terkecil untuk setiap iterasinya. Berikut adalah langkah-langkahnya.

1. Dari node awal, inisialisasi priority queue yang berbasis bobot
2. Queue node awal ke queue yang sudah terinisialisasi
3. Lakukan iterasi yang mengambil node dengan bobot terkecil dari queue, mengambil node tersebut sebagai solusi apabila node tersebut merupakan solusi yang dicari, dan memasukkan node-node yang belum dikunjungi ke dalam queue sesuai dengan bobot yang dimiliki
4. Mengembalikan solusi sesuai dengan algoritma atau jalur kosong apabila tidak ada solusi yang didapat

### 1.2.2 Algoritma Greedy Best First Search

Algoritma Greedy Best First Search adalah algoritma yang menggunakan nilai estimasi atau nilai heuristik untuk menentukan jalur dibanding nilai yang sebenarnya. Berikut adalah langkah-langkahnya.

1. Dari node awal, inisialisasi priority queue yang berbasis nilai heuristik
2. Queue node awal ke queue yang sudah terinisialisasi
3. Lakukan iterasi yang mengambil node dengan nilai heuristik terkecil dari queue, mengambil node tersebut sebagai solusi apabila node tersebut merupakan solusi yang dicari, dan memasukkan node-node yang belum dikunjungi ke dalam queue sesuai dengan bobot yang dimiliki
4. Mengembalikan solusi sesuai dengan algoritma atau jalur kosong apabila tidak ada solusi yang didapat

### 1.2.3 Algoritma A*

Algoritma A* (dipanggil sebagai algoritma A-star) adalah algoritma yang menggunakan nilai estimasi atau nilai heuristik serta bobot terkecil untuk menentukan jalur dibanding nilai yang sebenarnya. Berikut adalah langkah-langkahnya.

1. Dari node awal, inisialisasi priority queue yang berbasis biaya total yang berupa hasil penambahan nilai heuristik dengan bobot terkecil
2. Queue node awal ke queue yang sudah terinisialisasi
3. Lakukan iterasi yang mengambil node dengan biaya total terkecil dari queue, mengambil node tersebut sebagai solusi apabila node tersebut merupakan solusi yang dicari, memperbaiki jalur yang diambil tergantung dengan biaya yang dibutuhkan dari node awal ke node tetangga, dan memasukkan node-node yang belum dikunjungi ke dalam queue sesuai dengan bobot yang dimiliki
4. Mengembalikan solusi sesuai dengan algoritma atau jalur kosong apabila tidak ada solusi yang didapat

# BAB 2
# SOURCE CODE

## 2.1 Algoritma

```
import java.util.*;

public class Algorithms {

    // UCS (Uniform Cost Search)
    public static List<String> findLadderUCS(String start, String end,
Set<String> wordList) {
        Queue<Node> queue = new
PriorityQueue<>(Comparator.comparingInt(Node::getCost));
        Set<String> visited = new HashSet<>();
        Map<String, String> parent = new HashMap<>(); // Initialize parent map

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

        while (!queue.isEmpty()) {
            Node current = queue.poll();
            String currentWord = current.getWord();
            if (currentWord.equals(end)) {
                return constructPath(start, end, parent);
            }

            List<String> neighbors = getNeighbors(currentWord, wordList);
            for (String neighbor : neighbors) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    parent.put(neighbor, currentWord);
                    queue.offer(new Node(neighbor, current.getCost() + 1));
                }
            }
        }
        List<String> path = constructPath(start, end, parent);
        return path;
    }

    // Greedy Best First Search
    public static List<String> findLadderGreedyBFS(String start, String end,
Set<String> wordList) {
        Queue<Node> queue = new
PriorityQueue<>(Comparator.comparingInt(Node::getHeuristic));
        Set<String> visited = new HashSet<>();
```

```
        Map<String, String> parent = new HashMap<>();

        queue.offer(new Node(start, calculateHeuristic(start, end)));
        visited.add(start);

        while (!queue.isEmpty()) {
            Node current = queue.poll();
            String currentWord = current.getWord();
            if (currentWord.equals(end)) {
                return constructPath(start, end, parent);
            }

            List<String> neighbors = getNeighbors(currentWord, wordList);
            for (String neighbor : neighbors) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    parent.put(neighbor, currentWord);
                    queue.offer(new Node(neighbor, calculateHeuristic(neighbor,
end)));
                }
            }
        }
        List<String> path = constructPath(start, end, parent);
        return path;
    }

    // A* (A-star)
    public static List<String> findLadderAStar(String start, String end,
Set<String> wordList) {
        PriorityQueue<Node> openSet = new
PriorityQueue<>(Comparator.comparingInt(Node::getF));
        Map<String, Integer> gScore = new HashMap<>();
        Map<String, String> cameFrom = new HashMap<>();

        openSet.offer(new Node(start, 0, calculateHeuristic(start, end)));
        gScore.put(start, 0);

        while (!openSet.isEmpty()) {
            Node current = openSet.poll();
            String currentWord = current.getWord();
            if (currentWord.equals(end)) {
                return reconstructPath(cameFrom, currentWord);
            }

            List<String> neighbors = getNeighbors(currentWord, wordList);
            for (String neighbor : neighbors) {
                int tentativeGScore = gScore.getOrDefault(currentWord,
Integer.MAX_VALUE) + 1;
```

```
                if (tentativeGScore < gScore.getOrDefault(neighbor,
Integer.MAX_VALUE)) {
                    cameFrom.put(neighbor, currentWord);
                    gScore.put(neighbor, tentativeGScore);
                    openSet.offer(new Node(neighbor, tentativeGScore,
calculateHeuristic(neighbor, end)));
                }
            }
        }
        List<String> path = reconstructPath(cameFrom, end);
    return path;
    }

    private static int calculateHeuristic(String current, String end) {
        int heuristic = 0;
        for (int i = 0; i < current.length(); i++) {
            if (current.charAt(i) != end.charAt(i)) {
                heuristic++;
            }
        }
        return heuristic;
    }

    private static List<String> constructPath(String start, String end,
Map<String, String> parent) {
        List<String> path = new ArrayList<>();
        String current = end;
        while (!current.equals(start)) {
            path.add(0, current);
            current = parent.get(current);
        }
        path.add(0, start);
        return path;
    }

    private static List<String> getNeighbors(String word, Set<String> wordList)
{
        List<String> neighbors = new ArrayList<>();
        char[] chars = word.toCharArray();
        for (int i = 0; i < word.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;
    }

    private static class Node {
        private String word;
        private int cost;
        private int heuristic;

        public Node(String word, int cost) {
            this.word = word;
            this.cost = cost;
        }

        public Node(String word, int cost, int heuristic) {
            this.word = word;
            this.cost = cost;
            this.heuristic = heuristic;
        }

        public String getWord() {
            return word;
        }

        public int getCost() {
            return cost;
        }

        public int getHeuristic() {
            return heuristic;
        }

        public int getF() {
            return cost + heuristic;
        }
    }
    private static List<String> reconstructPath(Map<String, String> cameFrom,
String currentWord) {
        List<String> path = new ArrayList<>();
        while (currentWord != null) {
            path.add(0, currentWord);
            currentWord = cameFrom.get(currentWord);
        }
        return path;
    }
}
```

Pada source code untuk algoritma ini, program menggunakan class yaitu Algorithms dan Node. Algorithms digunakan untuk menampung ketiga algoritma yang digunakan pada program ini sementara Node digunakan dalam pemrosesan yang dilakukan oleh ketiga algoritma yang digunakan. Method yang digunakan adalah calculateHeuristic yang digunakan untuk menghitung nilai heuristik, constructPath yang digunakan untuk membuat jalur dari node awal ke node tujuan, getNeighbors yang digunakan untuk mendapat node tetangga pada Word Ladder, dan reconstructPath yang berfungsi seperti constructPath namun digunakan pada algoritma A* atau A-star.

## 2.1 Main

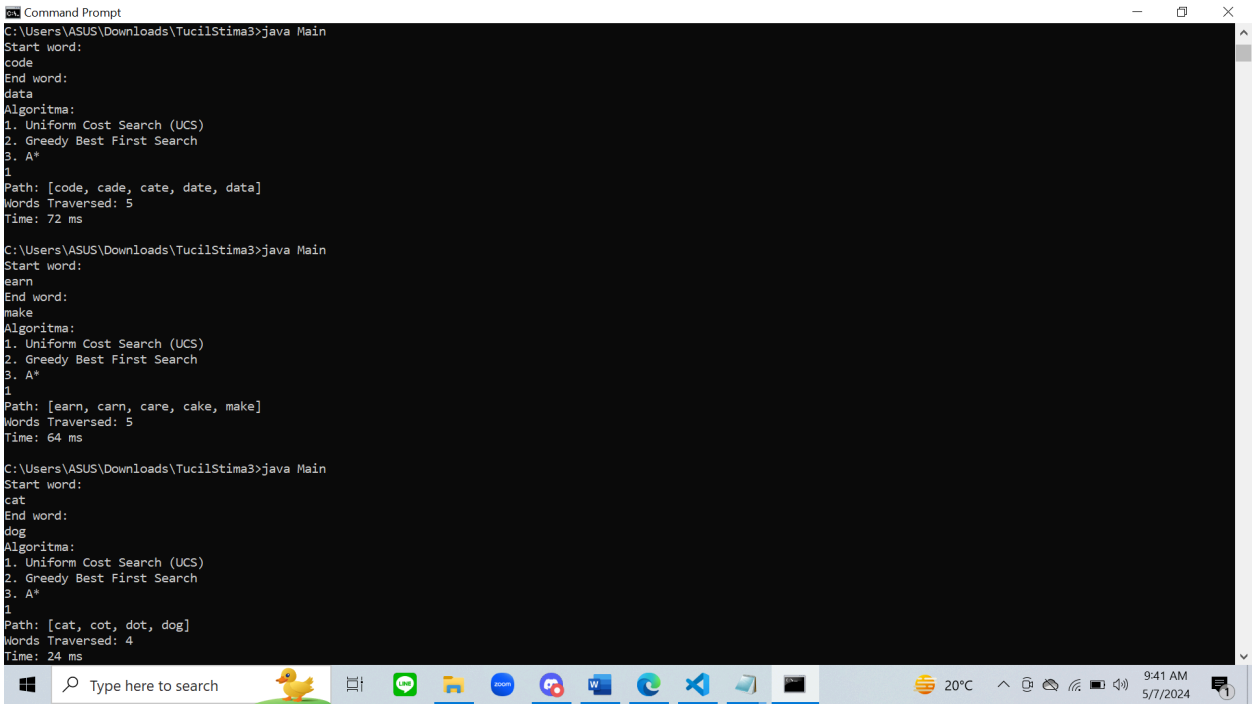
```
public class Main {
    public static void main(String[] args) {
        try {
            Set<String> wordList = loadDictionary("dictionary.txt");
            Scanner scanner = new Scanner(System.in);

            // Input start word dan end word
            System.out.println("Start word:");
            String start = scanner.next().toLowerCase();

            System.out.println("End word:");
            String end = scanner.next().toLowerCase();

            // Pilih algoritma
            System.out.println("Algoritma:");
            System.out.println("1. Uniform Cost Search (UCS)");
            System.out.println("2. Greedy Best First Search");
            System.out.println("3. A*");
            int choice = scanner.nextInt();

            // Mulai hitung waktu
            long startTime = System.currentTimeMillis();

            // Switch case algoritma
            List<String> ladder;
            switch (choice) {
                case 1:
                    ladder = Algorithms.findLadderUCS(start, end, wordList);
                    break;
                case 2:
                    ladder = Algorithms.findLadderGreedyBFS(start, end,
wordList);
                    break;
                case 3:
                    ladder = Algorithms.findLadderAStar(start, end, wordList);
                    break;
```

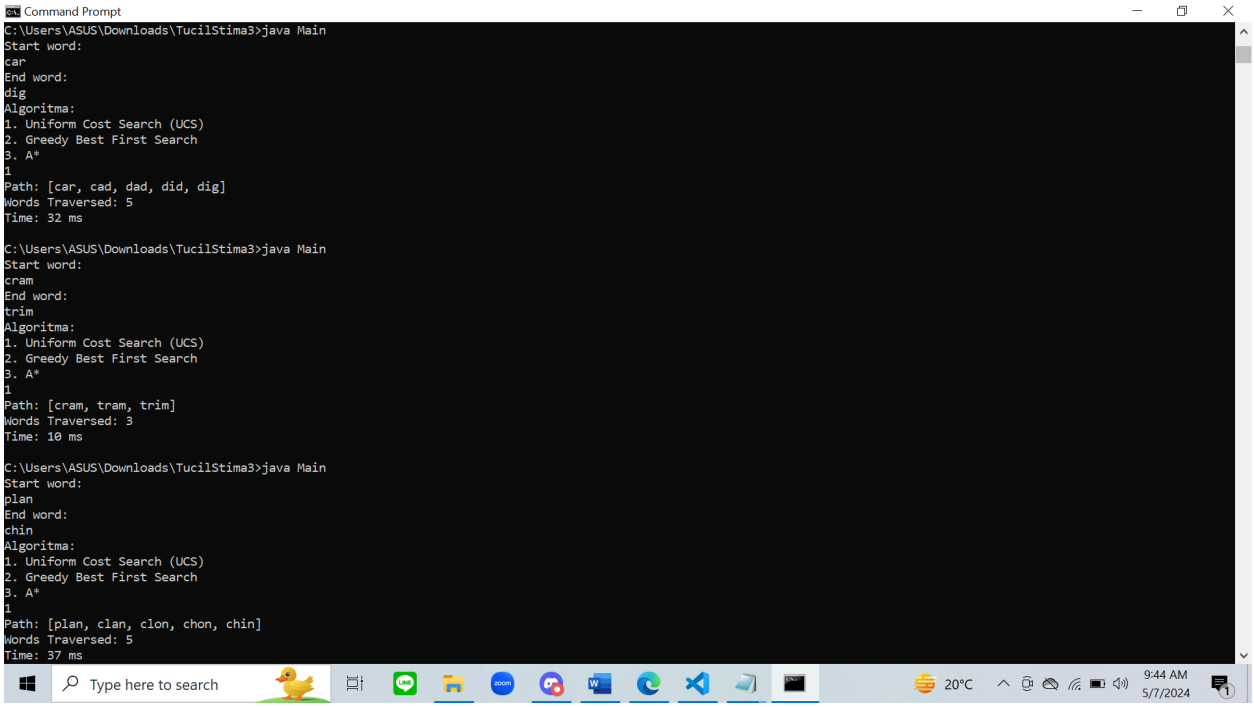
```
                default:
                    System.out.println("Tidak valid");
                    return;
            }

            // Selesai hitung waktu
            long endTime = System.currentTimeMillis();

            // Hitung total waktu
            long elapsedTime = endTime - startTime;

            // Tampilkan output
            if (!ladder.isEmpty()) {
                System.out.println("Path: " + ladder);
                System.out.println("Words Traversed: " + ladder.size()); //
Print the length of the ladder
            }
            else {
                System.out.println("Path: []");
            }
            System.out.println("Time: " + elapsedTime + " ms");

            scanner.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("Dictionary file not found.");
            e.printStackTrace();
        }
    }

    private static Set<String> loadDictionary(String filename) throws
FileNotFoundException {
        Set<String> dictionary = new HashSet<>();
        try (Scanner scanner = new Scanner(new File(filename))) {
            while (scanner.hasNextLine()) {
                dictionary.add(scanner.nextLine().trim().toLowerCase());
            }
        }
        return dictionary;
    }
}
```

Pada source code ini yang digunakan untuk program utama, terdapat class yaitu Main sebagai kelas utama pada program ini. Untuk method, terdapat main yang digunakan untuk menjalankan program dan loadDictionary yang digunakan untuk mengambil data dari dictionary.txt sesuai dengan dictionary yang digunakan di QnA yang tersedia dalam tugas ini.

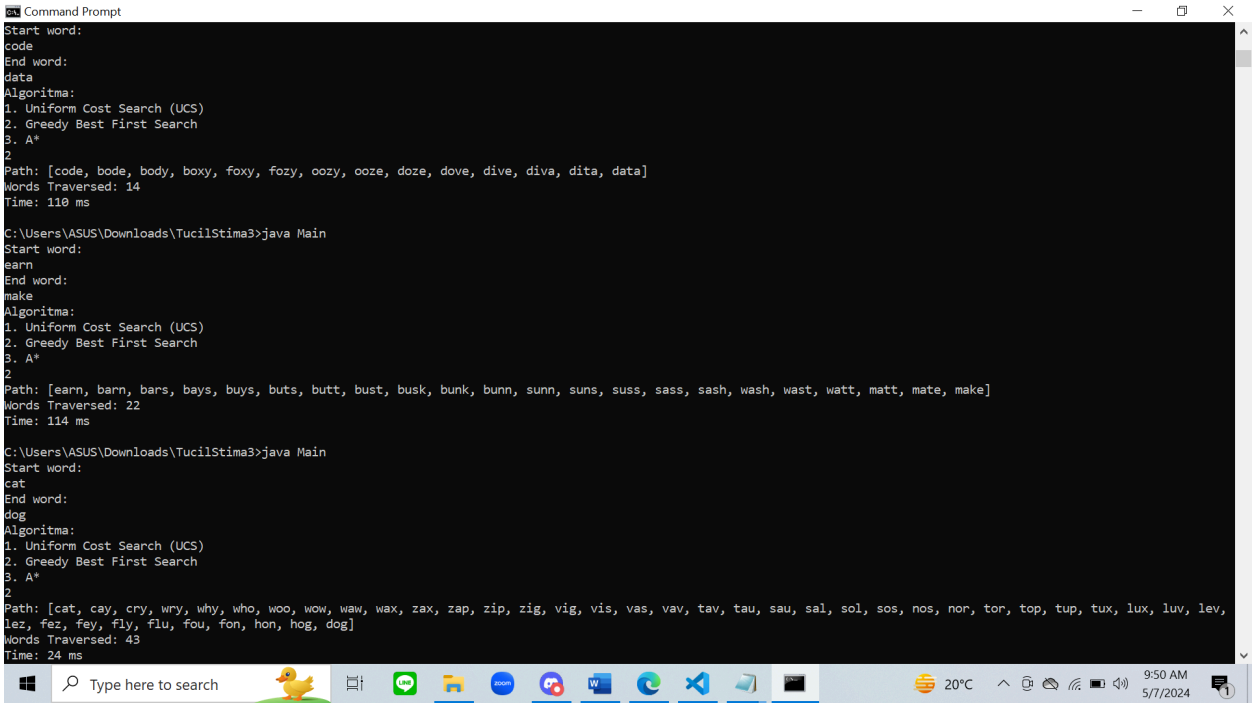
# BAB 3
# TEST CASE DAN ANALISIS

## 3.1 Test Case Algoritma UCS

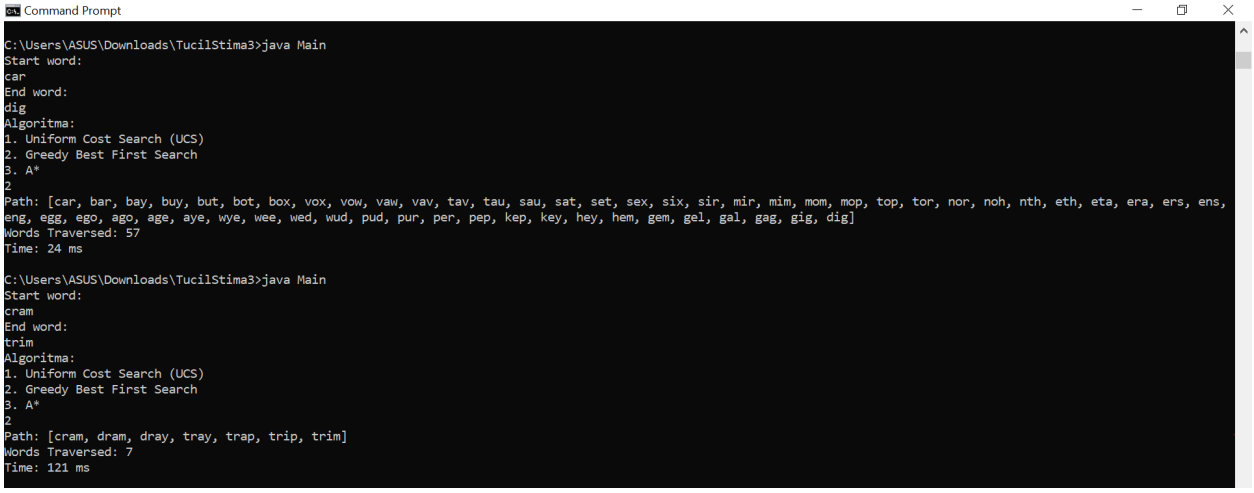


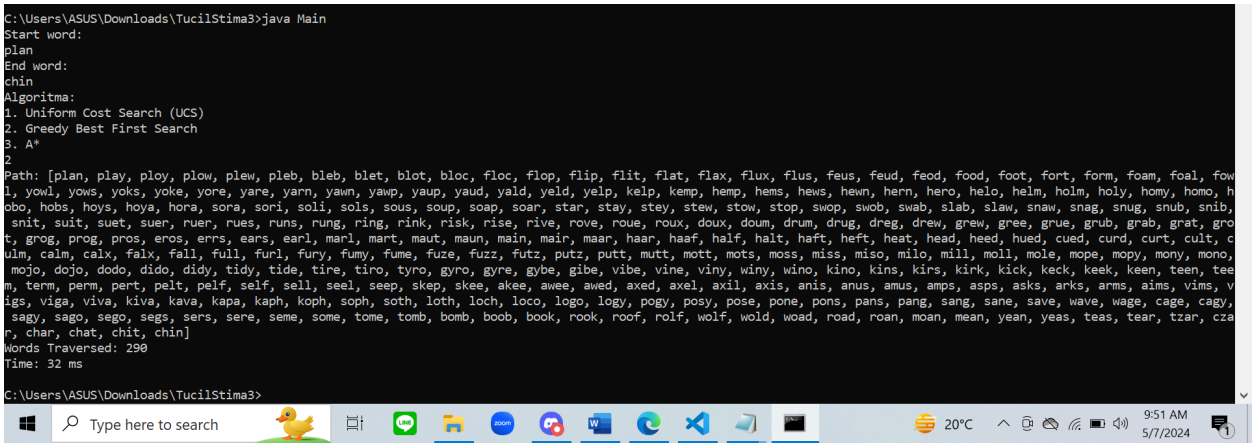
## 3.2 Test Case Algoritma Greedy Best First Search



```
Command Prompt
Start word:
code
End word:
data
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [code, bode, body, boxy, foxy, fozy, oozy, ooze, doze, dove, dive, diva, dita, data]
Words Traversed: 14
Time: 110 ms

C:\Users\ASUS\Downloads\TucilStima3>java Main
Start word:
earn
End word:
make
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [earn, barn, bars, bays, buys, buts, butt, bust, busk, bunk, bunn, sunn, suns, suss, sass, sash, wash, wast, watt, matt, mate, make]
Words Traversed: 22
Time: 114 ms

C:\Users\ASUS\Downloads\TucilStima3>java Main
Start word:
cat
End word:
dog
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [cat, cay, cry, wry, why, who, woo, wow, waw, wax, zax, zap, zip, zig, vig, vis, vas, vav, tav, tau, sau, sal, sol, sos, nos, nor, tor, top, tup, tux, lux, luv, lev,
lez, fez, fey, fly, flu, fou, fon, hon, hog, dog]
Words Traversed: 43
Time: 24 ms
```

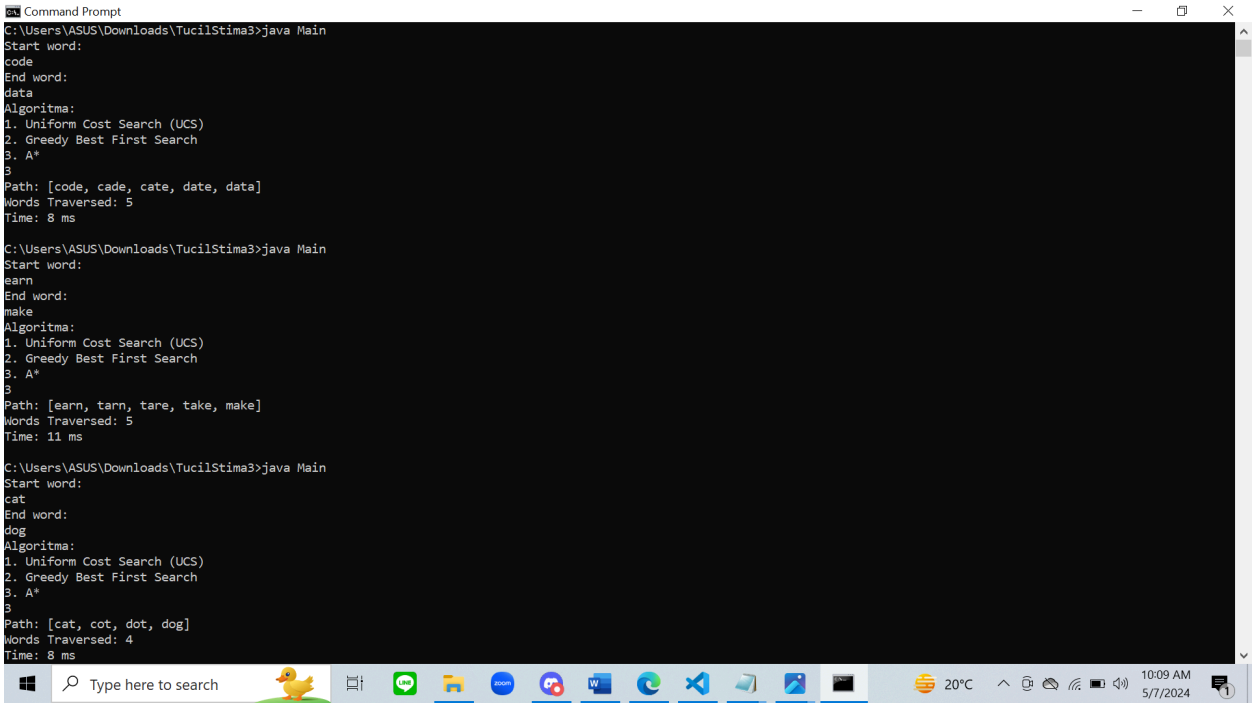


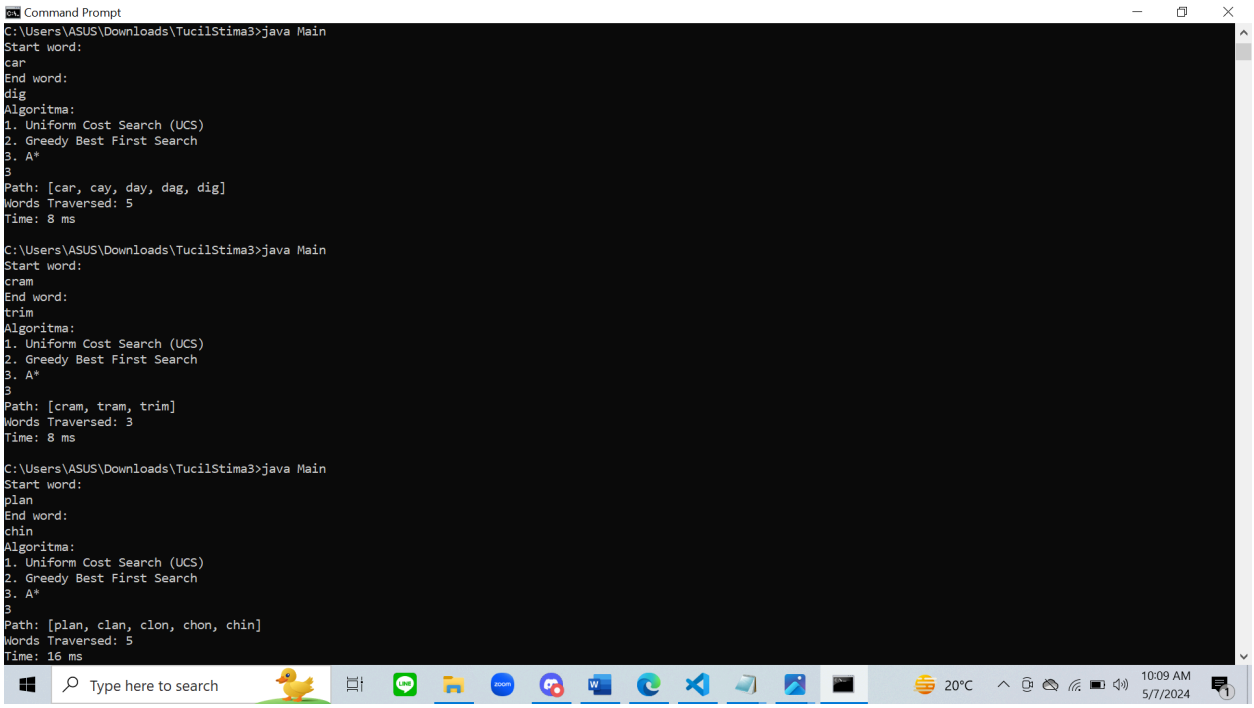
```
Command Prompt

C:\Users\ASUS\Downloads\TucilStima3>java Main
Start word:
car
End word:
dig
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [car, bar, bay, buy, but, bot, box, vox, vow, vaw, vav, tav, tau, sau, sat, set, sex, six, sir, mir, mim, mom, mop, top, tor, nor, noh, nth, eth, eta, era, ers, ens,
eng, egg, ego, ago, age, aye, wye, wee, wed, wud, pud, pur, per, pep, kep, key, hey, hem, gem, gel, gal, gag, gig, dig]
Words Traversed: 57
Time: 24 ms

C:\Users\ASUS\Downloads\TucilStima3>java Main
Start word:
cram
End word:
trim
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [cram, dram, dray, tray, trap, trip, trim]
Words Traversed: 7
Time: 121 ms
```



```
C:\Users\ASUS\Downloads\TucilStima3>java Main
Start word:
plan
End word:
chin
Algoritma:
1. Uniform Cost Search (UCS)
2. Greedy Best First Search
3. A*
2
Path: [plan, play, ploy, plow, plew, pleb, bleb, blet, blot, bloc, floc, flop, flip, flit, flat, flax, flux, flus, feus, feud, feod, food, foot, fort, form, foam, foal, fow
l, yowl, yows, yoks, yoke, yore, yare, yarn, yawn, yawp, yaup, yaud, yald, yeld, yelp, kelp, kemp, hemp, hems, hews, hewn, hern, hero, helo, helm, holm, holy, homy, homo, h
obo, hobs, hoys, hoya, hora, sora, sori, soli, sols, sous, soup, soap, soar, star, stay, stey, stew, stow, stop, swop, swob, swab, slab, slaw, snaw, snag, snug, snub, snib,
 snit, suit, suet, suer, ruer, rues, runs, rung, ring, rink, risk, rise, rive, rove, roue, roux, doux, doum, drum, drug, dreg, drew, grew, gree, grue, grub, grab, grat, gro
t, grog, prog, pros, eros, errs, ears, earl, marl, mart, maut, maun, main, mair, maar, haar, haaf, half, halt, haft, heft, heat, head, heed, hued, cued, curd, curt, cult, c
ulm, calm, calx, falx, fall, full, furl, fury, fumy, fume, fuze, fuzz, futz, putz, putt, mutt, mott, mots, moss, miss, miso, milo, mill, moll, mole, mope, mopy, mony, mono,
 mojo, dojo, dodo, dido, didy, tidy, tide, tire, tiro, tyro, gyro, gyre, gybe, gibe, vibe, vine, viny, winy, wino, kino, kins, kirs, kirk, kick, keck, keek, keen, teen, tee
m, term, perm, pert, pelt, pelf, self, sell, seel, seep, skep, skee, akee, awee, awed, axed, axel, axil, axis, anis, anus, amus, amps, asps, asks, arks, arms, aims, vims, v
igs, viga, viva, kiva, kava, kapa, kaph, koph, soph, soth, loth, loch, loco, logo, logy, pogy, posy, pose, pone, pons, pans, pang, sang, sane, save, wave, wage, cage, cagy,
 sagy, sago, sego, segs, sers, sere, seme, some, tome, tomb, bomb, boob, book, rook, roof, rolf, wolf, wold, woad, road, roan, moan, mean, yean, yeas, teas, tear, tzar, cza
r, char, chat, chit, chin]
Words Traversed: 290
Time: 32 ms

C:\Users\ASUS\Downloads\TucilStima3>
```

## 3.3 Test Case Algoritma A*




## 3.4 Analisis Hasil

Hasil yang didapat menunjukkan bahwa algoritma A* memiliki waktu pemrosesan yang cepat serta mengembalikan rute yang singkat dibanding dengan yang lain. Untuk algoritma UCS, waktu pemrosesannya cenderung lebih lama meskipun sudah bisa mengembalikan rute yang

singkat. Untuk algoritma Greedy Best First Search, waktu pemrosesan serta rute yang diambil cenderung lebih lama dan banyak.

Selain waktu pemrosesan, terdapat aspek optimalitas dimana dalam program ini, algoritma A* merupakan algoritma yang paling optimal dengan waktu yang cepat dan hasil yang baik. Algoritma UCS memiliki hasil yang cukup baik meskipun mengambil waktu lebih lama. Untuk algoritma Greedy Best First Search, hasil yang diberikan kurang optimal dengan waktu yang paling lama juga.

Lalu, secara penggunaan memori, algoritma UCS dan algoritma Greedy Best First Search tidak memakan banyak memori. Hal ini karena algoritma UCS dan Greedy Best First Search tidak menyimpan banyak hal dimana UCS hanya menyimpan bobot dan Greedy Best First Search hanya menyimpan sedikit perihal node yang dilalui. Algoritma A* memakan memori yang lebih besar dibanding kedua algoritma lainnya karena algoritma ini mengambil beberapa informasi berupa biaya total dan nilai heuristik.

### 3.5 Repository

https://github.com/Hapish/Tucil3_13522132