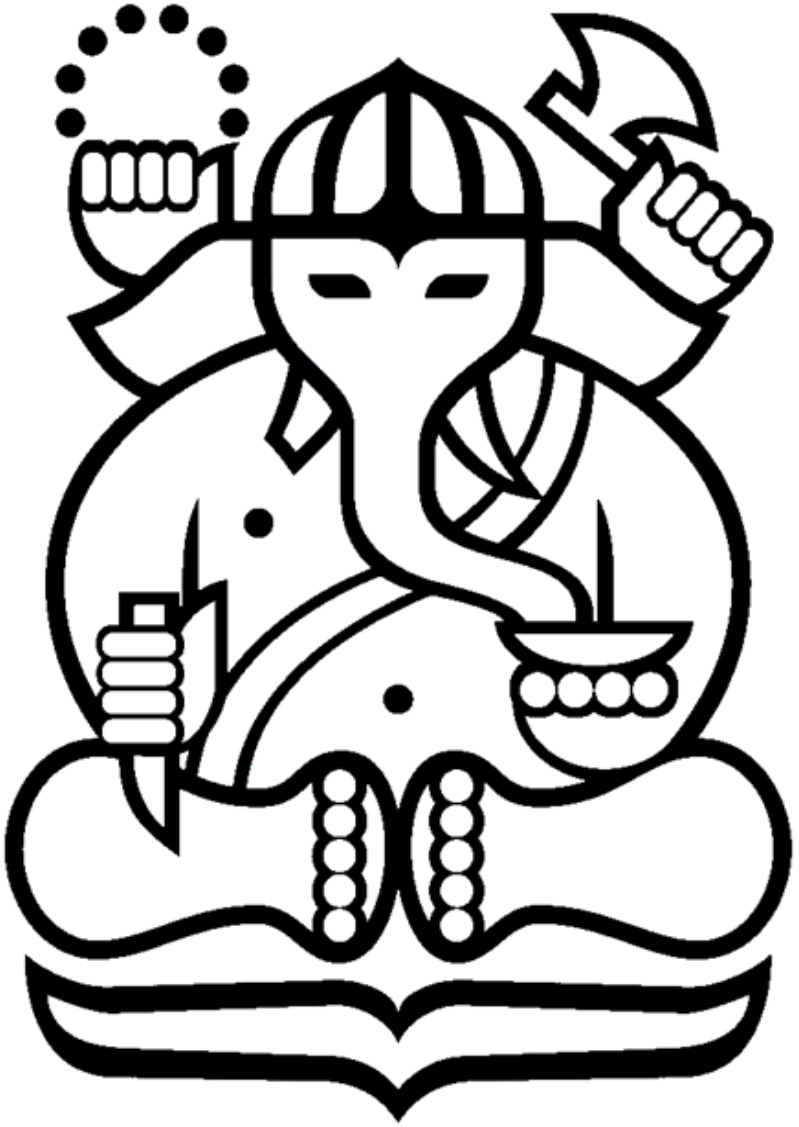
Laporan Tugas Kecil 3 IF2211 Strategi Algoritma

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



Disusun oleh:

Ahmad Farid Mudrika **13522008**

A. Penjelasan algoritma pencarian solusi

a. Menggunakan algoritma UCS

Dalam upaya mencari solusi permainan *Word Ladder* menggunakan algoritma UCS, secara garis besar adalah pencarian BFS karena *cost* untuk pergerakan tiap *node* adalah sama, yaitu satu.

Untuk implementasinya, akan dibuat sebuah queue dengan elemen pertama sebagai start. Berikutnya, queue akan dipop dan semua kata yang berjarak satu dari kata yang dipop akan dimasukkan dalam queue. Proses ini berulang sampai end ditemukan atau queue kosong, yang berarti solusi tidak ada.

b. Menggunakan algoritma Greedy Best First Search(GBFS)

Pada dasarnya pendekatan algoritma GBFS adalah dengan menghitung heuristik cost yang dibutuhkan untuk mencapai end. Dimulai dari kata start, akan dicari semua kata dengan panjang yang sama dan perbedaan 1 karakter. Untuk tiap kata ini akan dimasukkan ke PriorityQueue dengan urutan prioritas seberapa dekat kata tersebut dengan end. Dengan kata lain, berapa perubahan huruf yang dibutuhkan agar kata tersebut mencapai end.

PriorityQueue yang dibuat akan di-*pop*, dan tiap kata tetangga, yaitu kata dengan perbedaan karakter 1 yang belum pernah di-*pop* akan dimasukkan ke PriorityQueue dengan prioritas yang paling dekat dengan kata *end*. Proses ini akan diulang hingga ditemukan *end*, atau hingga queue kosong. Jika *end* ditemukan, akan dikembalikan *path* yang diambil, atau jika tidak ditemukan, berarti solusi tidak ada di kamus.

c. Menggunakan algoritma A*

Algoritma A* adalah penggabungan antara UCS dan GBFS. Algoritma ini akan membuat PriorityQueue yang sama dengan GBFS, dengan urutan prioritas adalah kedalaman ditambah heuristik terkecil. PriorityQueue akan dimulai dengan kata *start.*

PriorityQueue yang dibuat akan di-pop, dan tiap kata tetangga, yaitu kata dengan perbedaan karakter 1 yang belum pernah di-pop akan dimasukkan ke PriorityQueue dengan prioritas yang paling dekat dengan kata end. Proses ini akan diulang hingga ditemukan end, atau hingga queue kosong. Jika end ditemukan, akan dikembalikan path yang diambil, atau jika tidak ditemukan, berarti solusi tidak ada di kamus.

B. Source code program

a. Pembacaan Kamus

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class WordReader {
         public  static  Set<String>  readWordsFromFile(String
filePath) throws IOException {
        Set<String> words = new HashSet<>();

             try  (BufferedReader  br  =  new  BufferedReader(new
FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                words.add(line.trim().toLowerCase());
            }
        }

        return words;
    }
}
```

b. Algoritma UCS

```
      public void UCS(String start, String end, Set<String>
wordSet){
        Instant start_time = Instant.now();
        Map<String, String> edgeMap = new HashMap<>();
        Map<String, Boolean> visited = new HashMap<>();
        for (String str : wordSet) {
            visited.put(str, false);
        }
        Queue<String> queue = new LinkedList<>();
        queue.add(start);
        visited.put(start, true);
        boolean found = false;
         // try (BufferedWriter writer = new BufferedWriter(new
FileWriter("output.txt"))) {
        while (!queue.isEmpty()) {
            String current = queue.poll();

            this.visitednode++;
            if (current.equals(end)) {
                found = true;
                break;
            }
                                        Set<String>   paths   =
WordFinder.findWordsWithOneCharDifference(current, wordSet);
            // writer.write(current + ": " + paths + "\n");
            for (String word : paths) {
                if (!visited.get(word)) {
                    visited.put(word, true);
                    queue.add(word);
                    edgeMap.put(word, current);
                }
            }
        }
        if (found) {
            String current = end;
            while (!current.equals(start)) {
                this.solution.add(0, current);
                current = edgeMap.get(current);
            }
            this.solution.add(0, start);
        }
        // } catch (IOException e) {
        //     e.printStackTrace();
```

```
        // }
        Instant end_time = Instant.now();
        duration = Duration.between(start_time, end_time);
    }
```

c. Algoritma Greedy Best First Search

```
public void GreedyBestFirstSearch(String start, String end,
Set<String> wordSet){
        Instant start_time = Instant.now();
        Map<String, String> edgeMap = new HashMap<>();
        Map<String, Boolean> visited = new HashMap<>();
        for (String str : wordSet) {
            visited.put(str, false);
        }
                        PriorityQueue<String>  queue  =  new
PriorityQueue<>(Comparator.comparing(s                    ->
WordFinder.getDistance(s, end)));
        queue.add(start);
        visited.put(start, true);
        boolean found = false;
         // try (BufferedWriter writer = new BufferedWriter(new
FileWriter("output.txt"))) {
        while (!queue.isEmpty()) {
            String current = queue.poll();
            visited.put(current, true);
            this.visitednode++;
            if (current.equals(end)) {
                found = true;
                break;
            }
                                        Set<String>  paths  =
WordFinder.findWordsWithOneCharDifference(current, wordSet);
             // writer.write(current + ": " + paths + "\n"); //
Write to file instead of printing
            for (String word : paths) {
                if (!visited.get(word)) {

                    queue.add(word);
                    edgeMap.put(word, current);
                }
            }
        }
        if (found) {
            String current = end;
```

```
            while (!current.equals(start)) {
                this.solution.add(0, current);
                current = edgeMap.get(current);
            }
            this.solution.add(0, start);
        }
        // } catch (IOException e) {
        //     e.printStackTrace();
        // }
        Instant end_time = Instant.now();
        duration = Duration.between(start_time, end_time);
    }
    public static int getDistance(String word1, String word2){
        int distance = 0;
        for (int i = 0; i < word1.length(); i++) {
            if (word1.charAt(i) != word2.charAt(i)) {
                distance++;
            }
        }
        return distance;
    }
```

d. Algoritma A*

```
    public void GreedyBestFirstSearch(String start, String end,
Set<String> wordSet){
        Instant start_time = Instant.now();
        Map<String, String> edgeMap = new HashMap<>();
        Map<String, Boolean> visited = new HashMap<>();
        for (String str : wordSet) {
            visited.put(str, false);
        }
                        PriorityQueue<String>   queue   =   new
PriorityQueue<>(Comparator.comparing(s                          ->
WordFinder.getDistance(s, end)));
        queue.add(start);
        visited.put(start, true);
        boolean found = false;
         // try (BufferedWriter writer = new BufferedWriter(new
FileWriter("output.txt"))) {
        while (!queue.isEmpty()) {
            String current = queue.poll();
            visited.put(current, true);
            this.visitednode++;
            if (current.equals(end)) {
```

```
                found = true;
                break;
            }
                                        Set<String>   paths   =
WordFinder.findWordsWithOneCharDifference(current, wordSet);
             // writer.write(current + ": " + paths + "\n"); //
Write to file instead of printing
            for (String word : paths) {
                if (!visited.get(word)) {
                    queue.add(word);
                    edgeMap.put(word, current);
                }
            }
        }
        if (found) {
            String current = end;
            while (!current.equals(start)) {
                this.solution.add(0, current);
                current = edgeMap.get(current);
            }
            this.solution.add(0, start);
        }
        // } catch (IOException e) {
        //     e.printStackTrace();
        // }
        Instant end_time = Instant.now();
        duration = Duration.between(start_time, end_time);
    }
      public void Astar(String start, String end, Set<String>
wordSet){
        Instant start_time = Instant.now();
        Map<String, String> edgeMap = new HashMap<>();
        Map<String, Boolean> visited = new HashMap<>();
        for (String str : wordSet) {
            visited.put(str, false);
        }
                            PriorityQueue<Node>   queue   =   new
PriorityQueue<>(Comparator.comparing(s                          ->
WordFinder.getAstarCost(s, end) + s.cost));
         queue.add(new Node(start, 0));//Astar menggunakan node
untuk menyimpan cost saat ini.
        boolean found = false;
```

```
         // try (BufferedWriter writer = new BufferedWriter(new
FileWriter("output.txt"))) {
        while (!queue.isEmpty()) {
            Node current = queue.poll();
            visited.put(current.word, true);
            this.visitednode++;
            if (current.word.equals(end)) {
                found = true;
                break;
            }
                                        Set<String>   paths   =
WordFinder.findWordsWithOneCharDifference(current.word,
wordSet);

            for (String word : paths) {
                if (!visited.get(word)) {
                        queue.add(new Node(word, current.cost +
1));
                    if (!edgeMap.containsKey(word)){
                        edgeMap.put(word, current.word);
                    }

                }
            }
        }
        if (found) {
            String current = end;
            while (!current.equals(start)) {
                this.solution.add(0, current);
                current = edgeMap.get(current);
            }
            this.solution.add(0, start);
        }

        Instant end_time = Instant.now();
        duration = Duration.between(start_time, end_time);
    }
```

C. Hasil dalam tangkapan layar

a. Start frown, End smile

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: frown
End word: smile
Enter the algorithm (UCS/Astar/GBFS): ucs
Solution found (8) :
frown
flown
flowe
flote
slote
smote
smite
smile
Visited nodes: 3075
Time taken: 39018ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: frown
End word: smile
Enter the algorithm (UCS/Astar/GBFS): gbfs
Solution found (8) :
frown
flown
flowe
flote
slote
smote
smite
smile
Visited nodes: 8
Time taken: 194ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: frown
End word: smile
Enter the algorithm (UCS/Astar/GBFS): astar
Solution found (8) :
frown
flown
flowe
flote
slote
smote
smite
smile
Visited nodes: 692
Time taken: 8349ms
```

b. Start baby, End crib

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: baby
End word: crib
Enter the algorithm (UCS/Astar/GBFS): ucs
Solution found (7) :
baby
babs
cabs
caus
crus
crub
crib
Visited nodes: 5935
Time taken: 68449ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: baby
End word: crib
Enter the algorithm (UCS/Astar/GBFS): gbfs
Solution found (9) :
baby
baba
caba
caca
cana
canc
caic
cric
crib
Visited nodes: 11
Time taken: 231ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: baby
End word: crib
Enter the algorithm (UCS/Astar/GBFS): astar
Solution found (7) :
baby
babs
cabs
caus
crus
crub
crib
Visited nodes: 136
Time taken: 1747ms
```

c. Start start, End break

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: start
End word: break
Enter the algorithm (UCS/Astar/GBFS): ucs
Solution found (8) :
start
stert
stent
slent
blent
bleat
bleak
break
Visited nodes: 6113
Time taken: 71240ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: start
End word: break
Enter the algorithm (UCS/Astar/GBFS): gbfs
Solution found (9) :
start
stert
sterk
steak
speak
apeak
aleak
bleak
break
Visited nodes: 11
Time taken: 225ms
```

```
PS C:\Users\User\OneDrive - Institut Teknologi Bandung\Documents\Coding\Tucil3_13522008> java -cp bin Main
Start word: start
End word: break
Enter the algorithm (UCS/Astar/GBFS): astar
Solution found (8) :
start
stert
stent
slent
blent
blenk
bleak
break
Visited nodes: 236
Time taken: 2985ms
```

D. Analisis dan Pembahasan

Dalam implementasinya, pencarian cost menggunakan rumus f(n). f(n) adalah perkiraan total cost dari *start* ke *end.* g(n) adalah cost sejauh ini untuk mencapai n, dan h(n) adalah perkiraan cost dari n ke *end.* f(n) juga merupakan fungsi untuk menentukan pengurutan PriorityQueue. Dalam UCS, f(n) = g(n). Dalam GBFS, f(n) = h(n), dan di A*, f(n) = g(n) + h(n).

Heuristik yang digunakan pada algoritma A* *admissible*, karena heuristik ini tidak pernah meng-*overestimate* jarak dari *node* saat ini ke *goal*. Misalkan kata saat ini adalah *m,* dan *goal* adalah n. Jika heuristik memperkirakan jarak *m* ke *n* adalah 2, tidak mungkin untuk jarak sebenarnya dari *m* ke *n* untuk kurang dari 2. Jika kata antara saat mengubah salah satu karakter di *m* terdapat dalam kamus, maka *cost* sejati pasti adalah 2. Sedangkan jika kata antara tidak ada di kamus, diperlukan perubahan yang lebih dari 2 untuk mencapai *goal*.

Dalam kasus *word ladder*, algoritma UCS sama dengan BFS karena UCS akan selalu menelusuri *path* dengan kedalaman lebih kecil terlebih dahulu.

Secara teoritis, algoritma A* lebih efisien dari UCS karena penggunaan heuristik untuk memilih rute terdekat. Secara praktis, sebagaimana dapat dilihat di bab C, waktu yang dibutuhkan algoritma A* beberapa kali lebih sedikit dari waktu yang dibutuhkan algoritma UCS dengan hasil yang optimal, karena heuristiknya *admissible.*

Secara teoritis, algoritma GBFS tidak dijamin optimal, karena hanya menggunakan pendekatan heuristik untuk menentukan rute terbaik. Secara praktis, sebagaimana dapat dilihat di bab C bagian b, waktu yang dibutuhkan algoritma GBFS beberapa kali lebih cepat dari algoritma lainnya, meskipun hasilnya tidak optimal. Akan tetapi, ini tidak menjamin bahwa algoritma GBFS akan tidak optimal, sebagaimana diperlihatkan di bab C bagian a.

Link Repository Github:
https://github.com/frdmmm/Tucil3_13522008

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 algoirtma 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		✔