

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma
Penyelesaian Permainan Word Ladder Menggunakan Algoritma
UCS, Greedy Best First Search, dan A*



Disusun oleh:

Emery Fathan Zwageri

13522079

A. Penjelasan algoritma pencarian solusi

1. Algoritma UCS digunakan dalam permainan Word Ladder dengan melakukan pencarian BFS karena semua pergerakan memiliki cost yang sama, yaitu satu. Implementasinya menggunakan sebuah queue dengan elemen pertama sebagai start, kemudian queue akan di-pop dan semua kata yang berjarak satu dari kata yang di-pop akan dimasukkan ke dalam queue. Proses ini terus berlanjut sampai end ditemukan atau queue kosong, menandakan bahwa solusi tidak ada.

2. Pada algoritma GBFS, pendekatannya adalah dengan menghitung heuristik cost untuk mencapai end. Dimulai dari kata start, mencari semua kata dengan panjang yang sama dan perbedaan 1 karakter. Setiap kata ini dimasukkan ke dalam PriorityQueue dengan urutan prioritas berdasarkan seberapa dekat kata tersebut dengan end, yaitu berapa perubahan huruf yang dibutuhkan agar kata tersebut mencapai end. PriorityQueue akan di-pop, dan kata tetangga (kata dengan perbedaan karakter 1 yang belum pernah di-pop) dimasukkan ke PriorityQueue dengan prioritas yang paling dekat dengan kata end. Proses ini diulang hingga ditemukan end atau PriorityQueue kosong, dan jika end ditemukan, akan dikembalikan path yang diambil; jika tidak, solusi tidak ada di kamus.

3. Algoritma A* menggabungkan UCS dan GBFS dengan membuat PriorityQueue yang sama dengan GBFS namun dengan urutan prioritas yang ditentukan oleh kedalaman ditambah heuristik terkecil. PriorityQueue dimulai dengan kata start, di-pop, dan kata tetangga dimasukkan ke PriorityQueue dengan prioritas yang paling dekat dengan kata end. Proses ini diulang hingga ditemukan end atau PriorityQueue kosong, dengan hasil kembalian path yang diambil jika end ditemukan, atau tidak ada solusi jika tidak ditemukan di kamus.

B. Source code program

a. Folder src

1.class Astar

```
1 import java.util.Set;
2 import java.util.HashMap;
3 import java.util.PriorityQueue;
4 import java.util.Map;
5 import java.util.ArrayList;
6 import java.util.Comparator;
7 public class Astar {
8     public static int cost_counter(String curr,String end ){
9         int diff = 0;
10        for(int i=0;i<curr.length();i++){
11            if(end.charAt(i)!=curr.charAt(i)){
12                diff++;
13            }
14        }
15        return diff;
16    }
17    public static void solve(String start,String end,Set<String> words){
18        Map<String,Boolean> visited = new HashMap<>();
19
20        PriorityQueue<Pair<String,Integer>> queue = new PriorityQueue<>(Comparator.comparingInt(s->s.getValue()));
21        visited.put(start, true);
22        Map<String,String> parent =new HashMap<>();
23        Map<String,Integer> dist = new HashMap<>();
24
25        queue.add(new Pair<String,Integer>(start, cost_counter(start, end) ));
26        boolean Found = false;
27        int cnt = 1;
28        while(!queue.isEmpty()){
29            Pair<String,Integer> current = queue.poll();
30            if(current.getKey().equals(end)){
31                Found = true;
32                break;
33            }
34            Set<String> neighbour= utils.Neighbour.getNeighbour(current.getKey(), words);
35            for(String s:neighbour){
36                if(visited.get(s)==null){
37                    cnt++;
38                    queue.add(new Pair<String,Integer>(s, cost_counter(current.getKey(), end)+1+current.getValue()));
39                    visited.put(s,true);
40                    parent.put(s,current.getKey());
41                }
42            }
43        }
44        ArrayList<String> path = new ArrayList<>();
45        if(Found){
46            String curr = end ;
47            while(!curr.equals(start)){
48                path.add(0,curr);
49                curr = parent.get(curr);
50            }
51            if(curr.equals(start)){
52                path.add(0, curr);
53            }
54        }
55        for(String word: path){
56            System.out.println(word);
57        }
58        System.out.println("Visited node: "+ cnt);
59        System.out.println("Length Solution : "+ (path.size()-1));
60    }
61 }
62 }
```

class Astar merupakan class yang berisi algoritma Astar method solve untuk mencari solusi menggunakan algoritma astar dengan parameter start string yaitu kata mulai end string yaitu kata akhir dan set of string yaitu set yang berisi isi kamus. Method cost_counter menghitung cost pada astar yaitu berapa perubahan huruf dari kata awal ke kata sekarang ditambah kata sekarang ke kata akhir

2.class Gbfs

```

1  import java.util.ArrayList;
2  import java.util.Comparator;
3  import java.util.HashMap;
4  import java.util.Map;
5  import java.util.PriorityQueue;
6  import java.util.Queue;
7  import java.util.Set;
8
9  public class Gbfs{
10     public static int cost_counter(String start,String curr ){
11         int diff = 0;
12         for(int i=0;i<start.length();i++){
13             if(start.charAt(i)!=curr.charAt(i)){
14                 diff++;
15             }
16         }
17         return diff;
18     }
19
20     public static void solve(String start,String end,Set<String> words){
21         Map<String,Boolean> visited = new HashMap<>();
22         PriorityQueue<String> queue = new PriorityQueue<>(Comparator.comparing(s->cost_counter(s,end)));
23         visited.put(start, true);
24         Map<String,String> parent =new HashMap<>();
25         queue.add(start);
26         boolean Found = false;
27         int cnt = 1;
28         while(!queue.isEmpty()){
29             String current = queue.poll();
30             if(current.equals(end)){
31                 Found = true;
32                 break;
33             }
34             visited.put(current,true);
35             Set<String> neighbour= utils.Neighbour.getNeighbour(current, words);
36
37             for(String s:neighbour){
38                 if(visited.get(s)==null){
39                     cnt++;
40
41                     queue.add(s);
42                     parent.put(s,current );
43                 }
44             }
45         }
46         ArrayList<String> path = new ArrayList<>();
47         if(Found){
48             String curr = end ;
49             while(!curr.equals(start)){
50                 path.add(0,curr);
51                 curr = parent.get(curr);
52             }
53             if(curr.equals(start)){
54                 path.add(0, curr);
55             }
56         }
57         for(String word: path){
58             System.out.println(word);
59         }
60         System.out.println("Visited node: "+ cnt);
61         System.out.println("Length Solution : "+ (path.size()-1));
62     }
63
64 }

```

class Gbfs merupakan class yang berisi algoritma greedy best first search method solve untuk mencari solusi menggunakan algoritma astar dengan parameter start string yaitu kata mulai end string yaitu kata akhir dan set of string yaitu set yang berisi isi kamus. Method cost_counter menghitung cost pada Gbfs yaitu berapa perubahan huruf dari kata sekarang ke kata akhir

3.class Ucs

```

1  import java.util.Set;
2  import java.util.HashMap;
3
4  import java.util.Map;
5  import java.util.Queue;
6  import java.util.LinkedList;
7  import java.util.ArrayList;
8  public class Ucs{
9      public static void solve(String start,String end,Set<String> words){
10         Map<String,Boolean> visited = new HashMap<>();
11         Queue<String> queue = new LinkedList<>();
12         visited.put(start, true);
13         Map<String,String> parent =new HashMap<>();
14         queue.add(start);
15         boolean Found = false;
16         int cnt = 1;
17         while(!queue.isEmpty()){
18             String current = queue.poll();
19             if(current.equals(end)){
20                 Found = true;
21                 break;
22             }
23             Set<String> neighbour= utils.Neighbour.getNeighbour(current, words);
24             for(String s:neighbour){
25                 if(visited.get(s)==null){
26                     cnt++;
27                     visited.put(s,true);
28                     queue.add(s);
29                     parent.put(s,current );
30                 }
31             }
32         }
33         ArrayList<String> path = new ArrayList<>();
34         if(Found){
35             String curr = end ;
36             while(!curr.equals(start)){
37                 path.add(0,curr);
38                 curr = parent.get(curr);
39             }
40             if(curr.equals(start)){
41                 path.add(0, curr);
42             }
43         }
44         for(String word: path){
45             System.out.println(word);
46         }
47         System.out.println("Visited node: "+ cnt);
48         System.out.println("Length Solution : "+ (path.size()-1));
49     }
50 }

```

class Ucs merupakan class yang berisi algoritma Uniform Cost Search . Method solve untuk mencari solusi menggunakan algoritma UCS dengan parameter start string yaitu kata mulai end string yaitu kata akhir dan set of string yaitu set yang berisi isi kamus. Cost pada UCS adalah seragam yaitu 1 antar neighbour karena tidak ada weight Antar neighbour. Sehingga untuk kasus game worldLadder UCS menjadi BFS biasa

4.class Main


```

1
2 import java.util.Scanner;
3 import utils.Dict;
4 import java.util.Set;
5
6 public class Main {
7     public static void main(String[] args){
8         Set<String> dict = Dict.readFromFile("dictionary/dictionary.txt");
9         while(true){
10             System.out.println("\n\n" + //
11
12                 WELCOME TO //
13
14                 //
15                 //
16                 //
17                 //
18                 \n\n" + //
19
20                 WORDLADDER //
21
22                 //
23                 //
24                 \n\n" + //
25
26                 NEGLARRENVAL //
27
28                 //
29                 //
30                 //
31                 \n\n" + //
32
33                 WARNA APA BOS? //
34
35                 //
36                 //
37                 //
38                 //
39
40             Scanner inp = new Scanner(System.in);
41             String start;
42             String end;
43             System.out.print("Enter start word: ");
44             start = inp.nextLine();
45             while(!start.matches("[a-zA-Z]+") || !dict.contains(start)){
46                 if(!start.matches("[a-zA-Z]+")){
47                     System.out.println("Input have prohibited character, Enter word again: ");
48                     start = inp.nextLine();
49                 }
50                 if(!dict.contains(start)){
51                     System.out.println("Input doesnt exist in dictionary: ");
52                     start = inp.nextLine();
53                 }
54             }
55             System.out.print("Enter end word: ");
56             end = inp.nextLine();
57             while(!end.matches("[a-zA-Z]+") || !dict.contains(end)){
58                 if(!end.matches("[a-zA-Z]+")){
59                     System.out.println("Input have prohibited character, Enter word again: ");
60                     end = inp.nextLine();
61                 }
62                 if(!dict.contains(end)){
63                     System.out.println("Input doesnt exist in dictionary: ");
64                     end = inp.nextLine();
65                 }
66             }
67
68             while(!utils.Validation.val_length(start, end)){
69                 System.out.println("Panjang word harus sama: ");
70                 System.out.print("Enter start word: ");
71                 start = inp.nextLine();
72                 while(!start.matches("[a-zA-Z]+") || !dict.contains(start)){
73                     if(!start.matches("[a-zA-Z]+")){
74                         System.out.println("Input have prohibited character, Enter word again: ");
75                         start = inp.nextLine();
76                     }
77                     if(!dict.contains(start)){
78                         System.out.println("Input doesnt exist in dictionary: ");
79                         start = inp.nextLine();
80                     }
81                 }
82                 System.out.print("Enter end word: ");
83                 end = inp.nextLine();
84                 while(!end.matches("[a-zA-Z]+") || !dict.contains(end)){
85                     if(!end.matches("[a-zA-Z]+")){
86                         System.out.println("Input have prohibited character, Enter word again: ");
87                         end = inp.nextLine();
88                     }
89                     continue;
90                 }
91                 if(!dict.contains(end)){
92                     System.out.println("Input doesnt exist in dictionary: ");
93                     end = inp.nextLine();
94                 }
95             }
96
97             System.out.println("Enter algo:\n1.UCS\n2.GBFS\n3.AStar\nYour choice: ");
98             String algo = inp.nextLine();
99             while(!algo.equals("1") || !algo.equals("2") || !algo.equals("3")){
100                 System.out.println("Enter choice again: ");
101                 algo = inp.nextLine();
102             }
103             if(algo.equals("3")){
104                 Long startTime = System.currentTimeMillis();
105                 Astar.solve(start, end, dict);
106                 Long endTime = System.currentTimeMillis();
107                 System.out.println("Elapsed time: " + (endTime - startTime) + "ms");
108             }
109             else if(algo.equals("2")){
110                 Long startTime = System.currentTimeMillis();
111                 GBFS.solve(start, end, dict);
112                 Long endTime = System.currentTimeMillis();
113                 System.out.println("Elapsed time: " + (endTime - startTime) + "ms");
114             }
115             else if(algo.equals("1")){
116                 Long startTime = System.currentTimeMillis();
117                 UCS.solve(start, end, dict);
118                 Long endTime = System.currentTimeMillis();
119                 System.out.println("Elapsed time: " + (endTime - startTime) + "ms");
120             }
121             System.out.println("Quit(Y/N): ");
122             String isQuit = inp.nextLine();
123             if(isQuit.toLowerCase().equals("y")){
124                 inp.close();
125                 break;
126             }
127         }
128     }
129 }
130
131 }
132

```

Class Main berisi main program. Disini interface program dibuat. Program membaca dari kamus pada dictionary.txt lalu menerima input kata awal dan kata akhir dan melakukan handle input. Lalu menampilkan sesuai yang diminta spek. Input hanya bisa untuk alphabet(segala karakter selain alpabhet dilarang).

- b. Folder utils
 - 1. class Dict



```
1 package utils;
2 import java.io.BufferedReader;
3 import java.io.FileReader;
4 import java.io.IOException;
5 import java.util.HashSet;
6 import java.util.Set;
7
8 public class Dict {
9     // method untuk mendapatkan set of word dari file txt
10    public static Set<String> readFromFile(String fileName){
11        Set<String> wordSet = new HashSet<>();
12        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
13            String line;
14            while ((line = br.readLine()) != null) {
15                // Convert to lowercase and add to set
16                wordSet.add(line.toLowerCase());
17            }
18        } catch (IOException e) {
19            System.err.println("Error reading the file: " + e.getMessage());
20        }
21        return wordSet;
22    }
23
24
25 }
26
```

Class dict hanya berisi satu method untuk membaca file dan meletakkan isi file kedalam set of String.

- 2. class Neighbour

```
1 package utils;
2 import java.util.HashSet;
3 import java.util.Set;
4 public class Neighbour {
5     public static Set<String> getNeighbour(String s1, Set<String> sS){
6         Set<String> temp = new HashSet<String>();
7         for(String s:sS){
8             if(Validation.is_different1(s,s1)){
9                 temp.add(s);
10            }
11        }
12        return temp;
13    }
14
15
16 }
17
```

Class neighbour berisi method untuk mencari tetangga pada suatu kata. Yaitu seluruh kata dalam kamus yang hanya berbeda 1 huruf.

3. class Validation

```
1  package utils;
2
3
4
5  public class Validation {
6      // static method
7
8      // mengecek apakah panjang String s1 dan s2 sama
9      public static boolean val_length(String s1,String s2){
10         return s1.length()==s2.length();
11     }
12     // mengecek apakah perbedaan huruf pada string berjumlah 1
13     public static boolean is_different1(String s1,String s2){
14         if(!val_length(s1, s2)){
15             return false;
16         }
17         int diff = 0;
18         int length = s1.length();
19         for(int i=0;i<length;i++){
20             if(s1.charAt(i)!=s2.charAt(i)){
21                 diff++;
22             }
23         }
24         return diff==1;
25     }
26
27
28
29
30 }
31
```

Class validation berisi method `val_length` untuk menghitung apakah panjang 2 string sama dan berisi method `is_different1` yaitu menghitung apakah 2 string hanya berbeda 1 huruf.

C. Hasil dalam tangkapan layar

- a. Start,break
- 1.UCS

```

WARNNAAPA.BOS?
Enter start word: start
Enter end word: break
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
1
start
slart
blart
blert
bleat
bleak
break
Visited node: 10449
Length Solution : 6
Elapsed time: 118530ms
Quit(Y/N):
y

```

2.Gbfs

```

Enter start word: start
Enter end word: break
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
2
start
stert
sterk
steak
speak
apeak
aleak
bleak
break
Visited node: 73
Length Solution : 8
Elapsed time: 264ms
Quit(Y/N):

```

3.A*

```

Enter start word: start
Enter end word: break
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
3
start
slart
blart
blert
bleat
bleak
break
Visited node: 3651
Length Solution : 6
Elapsed time: 34677ms
Quit(Y/N):

```

b. baby, crib

1.UCS

```

Enter start word: baby
Enter end word: crib
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
1
baby
babs
cabs
cais
cris
crib
Visited node: 10077
Length Solution : 5
Elapsed time: 163780ms
Quit(Y/N):

```

2.Gbfs

```
Enter end word: crib
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
2
baby
babb
nabb
naib
nair
cair
cais
cris
crib
Visited node: 148
Length Solution : 8
Elapsed time: 238ms
Quit(Y/N):
```

3.A*

```
Enter start word: baby
Enter end word: crib
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
3
baby
babs
cabs
cais
cris
crib
Visited node: 5334
Length Solution : 5
Elapsed time: 34923ms
Quit(Y/N):

```

c. Start four,five

1.UCS

Enter start word: four

Enter end word: five

Enter algo:

1.UCS

2.GBFS

3.AStar

Your choice:

1

four

foud

ford

fore

fire

five

Visited node: 9338

Length Solution : 5

Elapsed time: 104310ms

Quit(Y/N):

2. GBFS

```
Enter start word: four
Enter end word: five
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
2
four
foud
ford
fore
fire
five
Visited node: 165
Length Solution : 5
Elapsed time: 251ms
Quit(Y/N):
```

3. A*

```
Enter start word: four
Enter end word: five
Enter algo:
1.UCS
2.GBFS
3.AStar
Your choice:
3
four
foud
fond
find
fine
five
Visited node: 6001
Length Solution : 5
Elapsed time: 37402ms
Quit(Y/N):
```

D. Analisis dan Pembahasan

Dari hasil percobaan yang dilakukan UCS memiliki waktu yang paling lambat karena tidak menggunakan heuristic. Tetapi solusi ucs dijamin optimal. A* lebih cepat dari UCS karena menggabungkan heuristic GBFS dan cost pada UCS sehingga lebih cepat dari UCS dan optimal. GBFS tidak optimal karena menggunakan greedy yang mana hanya mementingkan optimum lokal saja. Namun karena seperti itu maka GBFS menjadi yang tercepat.

Dalam implementasi algoritma, pencarian cost dilakukan menggunakan rumus $f(n)$, di mana $f(n)$ adalah perkiraan total cost dari start ke end. $g(n)$ adalah cost yang sudah ditempuh untuk mencapai node n , dan $h(n)$ adalah perkiraan cost dari n ke end. $f(n)$ juga digunakan untuk menentukan urutan prioritas di PriorityQueue. Dalam UCS, $f(n)$ sama dengan $g(n)$. Pada GBFS, $f(n)$ adalah $h(n)$, sementara pada A*, $f(n)$ adalah penjumlahan dari $g(n)$ dan $h(n)$.

Heuristik yang digunakan dalam algoritma A* adalah admissible, yang artinya heuristik ini tidak akan mengestimasi jarak yang lebih besar dari node saat ini ke goal. Sebagai contoh, jika heuristik memperkirakan jarak dari m ke n adalah 2, maka jarak sebenarnya tidak mungkin kurang dari 2. Jika kata-kata di antara m dan n ada dalam kamus, maka cost sebenarnya pasti 2. Namun, jika kata-kata di antara tidak ada dalam kamus, diperlukan lebih dari 2 perubahan untuk mencapai goal.

Dalam kasus Word Ladder, algoritma UCS pada dasarnya sama dengan BFS karena UCS akan selalu mencari path dengan kedalaman yang lebih kecil terlebih dahulu.

Secara teoritis, algoritma A* dianggap lebih efisien daripada UCS karena menggunakan heuristic untuk memilih rute terdekat. Namun, dalam praktiknya, terkadang waktu yang dibutuhkan oleh algoritma A* bisa lebih sedikit dari UCS dengan hasil yang tidak optimal. Hal ini mungkin disebabkan oleh penggunaan PriorityQueue atau penambahan elemen ke PriorityQueue yang tidak optimal.

Algoritma GBFS secara teoritis tidak menjamin optimalitas karena hanya mengandalkan heuristic untuk menentukan rute terbaik. Namun, dalam praktiknya, GBFS bisa jauh lebih cepat dari algoritma lainnya meskipun hasilnya tidak optimal. Meskipun demikian, tidak ada jaminan bahwa GBFS akan selalu menghasilkan solusi yang tidak optimal, seperti yang terlihat dalam beberapa kasus di lapangan..

Link Repository Github:

https://github.com/mrsuiii/Tucil3_13522079

<https://github.com/dwyl/english-words>(dictionary) yang words.txt

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

Disclaimer: 1. Program lambat karena saya tidak membuat adjacency list terlebih dahulu. Jadi dalam setiap loop baru mengecek neighbour dari suatu kata.

2. Dictionary yang digunakan adalah words.txt pada
<https://github.com/dwyl/english-words>