Laporan Tugas Kecil 3

Word Ladder Solver

Disusun oleh:

Vanson Kurnialim 13522049 (K01)

Mata Kuliah IF 2211 - Strategi Algoritma

Program Studi S1 Teknik Informatika

Sekolah Teknik Elektro dan Informatika

A. Analisis dan Implementasi

Terdapat 3 algoritma yang digunakan dalam pencarian solusi *Word Ladder* yaitu *Uniformed Cost Search* (UCS), *Greedy Best First Search,* dan A*. UCS termasuk dalam jenis algoritma *Uninformed Search* atau bisa dibilang *blind search* karena kita tidak memiliki informasi tambahan dalam proses pencarian yang membantu kita dalam memilih *node* yang lebih dekat dengan *goal*. Secara teori, UCS berbeda dengan BFS dan IDS yang mencari solusi berdasarkan panjang *path*. UCS memilih pembangkitan *node* berdasarkan *cost* yang diperlukan untuk mencapai *node* dari *start node* atau biasa disebut fungsi g(n). Berikut adalah langkah-langkah yang diimplementasikan dalam program untuk algoritma UCS :

- *Start node* dimasukkan ke dalam *queue* sebagai sebuah *path*.
- Dilakukan pengulangan dengan kondisi berhenti jika *goal* sudah ditemukan atau *queue* kosong.
- Di dalam pengulangan, *dequeue* antrian dan ambil *node* terakhir dari *path* yang di*dequeue.*
- Cari semua kata yang memiliki perbedaan satu huruf dari *node* dan diperiksa apakah *goal node*.
- Jika iya, pengulangan selesai dan solusi didapatkan. Jika tidak, *node* dimasukkan ke dalam *queue* sebagai *path* baru.
- Setiap *node* yang sudah pernah diaktifkan tidak akan diaktifkan kembali.

Perlu diperhatikan, tidak ada aspek *cost* dalam algoritma UCS yang diimplementasikan. Hal ini karena *cost* yang ada pada seluruh *node* yang bertetangga berjumlah 1, atau bisa dibilang tidak ada *cost* sama sekali. Tentu pada kasus UCS lain seperti pencarian rute, *cost* merupakan jarak antar *node*. Namun, pada *Word Ladder* ini, tiap *node* yang bertetangga hanya diperbolehkan memiliki perbedaan 1 huruf, sehingga *cost*-nya dapat diabaikan. Penggunaan *queue* dan alur pemasukkan pada antrian sudah memastikan urutan yang benar, dengan *cost* terkecil yang lebih dahulu dibanding *cost* yang lebih besar. Maka pada program ini, algoritma UCS tidak berbeda dibanding algoritma BFS.

Pada algoritma *Greedy Best First Search* atau disingkat GBFS, hanya satu *path* yang akan dibangkitkan dari awal sampai akhir algoritma. GBFS memilih *node* optimum lokal pada setiap pembangkitannya. GBFS termasuk dalam jenis algoritma *Informed Search* dikarenakan digunakan batasan atau fungsi pembantu h(n) yang bersifat heuristic. h(n) ditentukan berdasarkan jarak atau *cost* dari suatu *node* menuju *goal*. Pada kasus ini, fungsi heuristic menghitung perbedaan huruf dari *node* dibanding *goal.* Berikut adalah langkah-langkah algoritma GBFS dalam program :

- *Start node* dimasukkan dalam sebuah *path*.
- Dilakukan pengulangan dengan kondisi berhenti jika *goal* sudah ditemukan atau sudah tidak ada anakan dari *node* yang dapat dipilih.
- Di dalam pengulangan, diperiksa semua tetangga dari elemen terakhir pada *path*.
- Diambil satu *child node* dengan h(n) terkecil untuk dimasukkan ke dalam *path*.

Pada GBFS ini, hasil yang didapatkan belum tentu hasil optimal dikarenakan setiap *node* yang dibangkitkan merupakan optimum lokal tanpa memikirkan apakah pilihan tersebut merupakan pilihan yang paling optimal secara keseluruhan atau global. Dengan alasan yang sama, *goal* bisa tidak ditemukan dikarenakan sudah tidak ada *child node* selanjutnya atau bisa dibilang terjebak dalam optimum lokal.

Algoritma yang terakhir adalah algoritma A* yang juga termasuk dalam jenis algoritma *Informed Searched*. Algoritma ini bisa dibilang gabungan dari UCS dan GBFS yang menggunakan keunggulan dari kedua algoritma dan menggabungkannya dalam satu algoritma. Pada setiap pemilihan pembangkitan *node*, diperiksa fungsi f(n) yang memeriksa *cost* dari *node* tersebut. Fungsi ini merupakan penjumlahan dari fungsi g(n) dan fungsi heuristic h(n) yang telah didefinisikan di atas. Berikut adalah langkah-langkah dalam penggunaan algoritma A* dalam program ini :

- Dibuat sebuah *priority queue* dengan prioritas fungsi f(n). Elemen dengan f(n) terkecil akan diprioritaskan.
- *Start node* dimasukkan ke dalam *queue* sebagai sebuah *path*.
- Dilakukan pengulangan dengan kondisi berhenti jika *goal* sudah ditemukan atau *queue* kosong.

- Di dalam pengulangan, *dequeue* antrian dan ambil *node* terakhir dari *path* yang di*dequeue.*
- Cari semua kata yang memiliki perbedaan satu huruf dari *node* dan diperiksa apakah *goal node*.
- Jika iya, pengulangan selesai dan solusi didapatkan. Jika tidak, *node* dimasukkan ke dalam *queue* sebagai *path* baru.
- Setiap *node* yang sudah pernah diaktifkan tidak akan diaktifkan kembali.

Fungsi heuristic yang diterapkan pada kasus ini tidak pernah *overestimate* karena perbedaan huruf antara *node* dan *goal* bisa dengan tepat didapatkan. Maka dari itu, algoritma A* tidak akan terjebak dalam optimum lokal atau bisa dibilang algoritma ini *admissible*. Selain itu, dikarenakan A* memiliki keuntungan dari kedua algoritma UCS dan GBFS, A* lebih efisien dibanding kedua algoritma tersebut. Selain mengecek keseluruhan kemungkinan seperti UCS, A* juga membangkitkan *node* yang lebih berpotensi 'berhasil' lewat fungsi heuristicnya.

B. Source Code Program

Berikut adalah method-method yang ada di dalam kelas WordLadder :

```
public static boolean isEnglishWord(String word) {
        try {
            File bank = new File("words_alpha.txt") ;
            Scanner reader = new Scanner(bank) ;

            while (reader.hasNextLine()) {
                String temp = reader.nextLine() ;
                if (temp.equals(word)) {
                    return true ;
                }
            }
            reader.close() ;
        }
        catch(FileNotFoundException e) {
            System.out.println("File not found!") ;
        }
        return false ;
```

```
    }
```

Method isEnglishWord berfungsi untuk memeriksa apakah sebuah *String* terdapat di dalam kamus yang digunakan.

```
public static void getAllWords(List<String> bankStrings, int length)
{
        try {
            File bank = new File("words_alpha.txt") ;
            Scanner reader = new Scanner(bank) ;

            while (reader.hasNextLine()) {
                String temp = reader.nextLine() ;
                if (temp.length() == length) {
                    bankStrings.add(temp) ;
                }
            }

            reader.close() ;
        }
        catch(FileNotFoundException e) {
            System.out.println("File not found!") ;
        }
    }
```

Method getAllWords berfungsi untuk mengambil semua kata di kamus dengan *length* sesuai yang tertera di parameter dan memasukkannya ke dalam sebuah *List of String*.

```
public static boolean oneLetterDiff(String word1, String word2) {
        int result = 0 ;
        for (int i = 0 ; i < word1.length() ; i++) {
            if (word1.charAt(i) != word2.charAt(i)) {
                result += 1 ;
            }
        }
        if (result == 1) {
            return true ;
        }
```

```
        else {
            return false ;
        }
    }
```

Method oneLetterDiff berfungsi untuk mengembalikan *true* jika kedua *String* di dalam parameter hanya memiliki satu perbedaan huruf dan *false* jika tidak.

```
public static int countLetterDiff(String word1, String word2) {
        int result = 0 ;
        for (int i = 0 ; i < word1.length() ; i++) {
            if (word1.charAt(i) != word2.charAt(i)) {
                result += 1 ;
            }
        }
        return result ;
    }
```

Method countLetterDiff mengembalikan *int* jumlah perbedaan huruf antara dua *String*.

```
public   static   List<String>   getAllPossibleWords(String   word,
List<String> bank) {
        List<String> result = new ArrayList<>() ;
        for(int i = 0 ; i < bank.size() ; i++) {
            if (oneLetterDiff(word, bank.get(i))) {
                result.add(bank.get(i)) ;
            }
        }
        return result ;
    }
```

Method getAllPossibleWords mengembalikan sebuah *List of String* yang diambil dari elemen parameter kedua dengan perbedaan satu huruf dari parameter pertama.

```
public static void GBFS(String start, String goal) {
        // set start time
        long startTime = System.currentTimeMillis() ;
```

```
            List<String> result = new ArrayList<>() ; // declare
variable hasil akhir
        result.add(start) ;

        // list berisi seluruh String yang panjangnya sama
        List<String> bankStrings = new ArrayList<>();
        getAllWords(bankStrings, start.length()) ;

        // set berisi String yang telah diaktivasi
        Set<String> stringSet = new HashSet<>();
        stringSet.add(start) ;

        int wordCount = 0 ;

        while(true) {
            if (result.getLast().equals(goal)) {
                break ;
            }
                                    List<String>   possibleWords   =
getAllPossibleWords(result.getLast(), bankStrings) ;
            int indexNext = -1;
            int countLetterDiff = 999;
            for (int i = 0 ; i < possibleWords.size() ; i++) {
                if (!result.contains(possibleWords.get(i))) {
                                                int   tempCount   =
countLetterDiff(possibleWords.get(i), goal) ;
                    if (tempCount <= countLetterDiff) {
                        countLetterDiff = tempCount ;
                        indexNext = i ;
                    }
                }
            }
            if (indexNext == -1) {
                break ;
            }
            result.add(possibleWords.get(indexNext)) ;
            stringSet.add(possibleWords.get(indexNext)) ;
            wordCount += 1 ;
        }
```

```
        long endTime = System.currentTimeMillis() ;
        if (result.size() == 0 || !result.getLast().equals(goal)) {
            // result.clear();
            System.out.println("No path found!");
        }
        else {
            System.out.println(result);
        }
        System.out.printf("Node visited : %d \n" , wordCount);
            System.out.printf("Execution time : %d ms", (endTime -
startTime), "\n");
    }
```

Method GBFS merupakan salah satu method utama dalam program ini. Sesuai namanya, method ini mengurus proses pencarian sampai dengan menampilkan hasil dengan algoritma *Greedy Best First Search*. Method ini menerima 2 parameter yaitu kata awal dan tujuan.

```
public static void Astar(String start, String goal) {
        // set start time
        long startTime = System.currentTimeMillis() ;

        // Membuat antrian node
             Queue<List<String>> antrian = new PriorityQueue<>(new
wordComparator(goal)) ;
        List<String> awal = new ArrayList<>() ;
        awal.add(start) ;
        antrian.add(awal) ;

             List<String> result = new ArrayList<>() ; // declare
variable hasil akhir

        // list berisi seluruh String yang panjangnya sama
        List<String> bankStrings = new ArrayList<>();
        getAllWords(bankStrings, start.length()) ;

        // set berisi String yang telah diaktivasi
```

```
        Set<String> stringSet = new HashSet<>();
        stringSet.add(start) ;

        int wordCount = 0 ;

        while (!antrian.isEmpty()) {
            List<String> anak = antrian.poll() ;
                                    List<String>   possibleWords   =
getAllPossibleWords(anak.getLast(), bankStrings) ;
            boolean checkResult = false ;

            for(int i = 0 ; i < possibleWords.size() ; i++) {
                if (!stringSet.contains(possibleWords.get(i))) {
                    if (possibleWords.get(i).equals(goal)) {
                        checkResult = true ;
                        result = anak ;
                        result.add(goal) ;
                        break ;
                    }
                    List<String> in = new ArrayList<>(anak) ;
                    in.add(possibleWords.get(i)) ;
                    antrian.add(in) ;
                    wordCount += 1 ;
                    stringSet.add(possibleWords.get(i)) ;
                }
            }

            if (checkResult) {
                break ;
            }
        }
        long endTime = System.currentTimeMillis() ;
        if (result.size() == 0 || !result.getLast().equals(goal)) {
            // result.clear();
            System.out.println("No path found!");
        }
        else {
            System.out.println(result);
        }
        System.out.printf("Node visited : %d \n" , wordCount);
```

```
            System.out.printf("Execution time : %d ms", (endTime -
startTime), "\n");
    }
```

Method Astar sama seperti sebelumnya, namun menggunakan algoritma A* dalam proses pencariannya.

```
public static void UCS(String start, String goal) {
        // set start time
        long startTime = System.currentTimeMillis() ;

        // Membuat antrian node
        Queue<List<String>> antrian = new LinkedList<>() ;
        List<String> awal = new ArrayList<>() ;
        awal.add(start) ;
        antrian.add(awal) ;

             List<String> result = new ArrayList<>() ; // declare
variable hasil akhir

        // list berisi seluruh String yang panjangnya sama
        List<String> bankStrings = new ArrayList<>();
        getAllWords(bankStrings, start.length()) ;

        // set berisi String yang telah diaktivasi
        Set<String> stringSet = new HashSet<>();
        stringSet.add(start) ;

        int wordCount = 0 ;

        while (!antrian.isEmpty()) {
            List<String> anak = antrian.poll() ;
                                        List<String>   possibleWords   =
getAllPossibleWords(anak.getLast(), bankStrings) ;
            boolean checkResult = false ;

            for(int i = 0 ; i < possibleWords.size() ; i++) {
                if (!stringSet.contains(possibleWords.get(i))) {
                    if (possibleWords.get(i).equals(goal)) {
```

```
                        checkResult = true ;
                        result = anak ;
                        result.add(goal) ;
                        break ;
                    }
                    List<String> in = new ArrayList<>(anak) ;
                    in.add(possibleWords.get(i)) ;
                    antrian.add(in) ;
                    stringSet.add(possibleWords.get(i)) ;
                    wordCount += 1 ;
                }
            }

            if (checkResult) {
                break ;
            }
        }
        long endTime = System.currentTimeMillis() ;
        if (result.size() == 0 || !result.getLast().equals(goal)) {
            // result.clear();
            System.out.println("No path found!");
        }
        else {
            System.out.println(result);
        }
        System.out.printf("Node visited : %d \n" , wordCount);
            System.out.printf("Execution time : %d ms", (endTime -
startTime), "\n");
    }
```

Method UCS ini merupakan metode utama terakhir dalam proses pencarian hasil. Method ini menggunakan algoritma *Uniformed Cost Search*.

```
public static void main(String[] args) {
        System.out.println("Welcome to Word Ladder Solver!");

        Scanner input = new Scanner(System.in) ;
        String start ;
        String end ;
```

```
        while (true) {
            System.out.print("Start word : ") ;
            start = input.next() ;
            start = start.toLowerCase() ;
            if (isEnglishWord(start)) {
                break ;
            }

            System.out.print("It is not a word! Try Again!\n") ;
        }
        while (true) {
            System.out.print("End word : ") ;
            end = input.next() ;
            end = end.toLowerCase() ;
            if (isEnglishWord(end)) {
                break ;
            }
            System.out.print("It is not a word! Try Again!\n") ;
        }

        if (start.length() != end.length()) {
            System.out.println("Word's length doesn't match!") ;
            input.close();
            return ;
        }

         System.out.println("Choose which algorithm to use! \n1. UCS
\n2. Greedy Best First Search \n3. A*") ;
        System.out.print("Input : ") ;
        String algo = input.next() ;
        while(true) {
                    if  (!algo.equals("1")  &&  !algo.equals("2")  &&
!algo.equals("3")) {
                System.out.println("Only the number! Try again!");
                System.out.print("Input : ");
                algo = input.next() ;
            }
            else {
                input.close();
```

```
                break ;
            }
        }

        if (start.equals(end)) {
            System.out.println(start);
            System.out.printf("Node visited : 0 \n");
            System.out.printf("Execution time : 0 ms \n");
        }
        else {
            if(algo.equals("1")) {
                UCS(start, end) ;
            }
            else if (algo.equals("2")) {
                GBFS(start, end);
            }
            else {
                Astar(start, end) ;
            }
        }
    }
```

Method main merupakan awal dan sentral dari program. Ketika program dijalankan maka method main akan dipanggil dan berfungsi untuk menerima dan memvalidasi kata awal dan tujuan lalu menerima algoritma yang ingin digunakan dari user. Jika ketiga method sebelumnya merupakan *engine* dari program ini, main merupakan pusat kontrol dari program.

Selain kelas WordLadder dan method-methodnya yang telah dijelaskan di atas, terdapat satu kelas lagi yang digunakan dalam program ini yaitu kelas wordComparator. Kelas ini mengimplements kelas Comparator yang merupakan kelas bawaan dari *library* Java. kelas wordComparator ini me-*override* fungsi compare yang membandingkan dan menentukan elemen mana yang berada duluan dalam sebuah *queue*. Kelas ini digunakan dalam pembentukan *priority queue* pada algoritma A*, method Astar. Berikut adalah kode yang digunakan :

```
class wordComparator implements Comparator<List<String>> {
```

```
    private String goal ;

    public wordComparator(String goal) {
        this.goal = goal ;
    }
    public int compare(List<String> word1, List<String> word2) {
        // perhitungan cost heuristic h(n), node menuju goal
            int count1 = WordLadder.countLetterDiff(word1.getLast(),
goal) ;
            int count2 = WordLadder.countLetterDiff(word2.getLast(),
goal) ;
        // perhitungan cost g(n), cost dari start menuju node
        count1 += word1.size() ;
        count2 += word2.size() ;

        if (count1 < count2) {
            return -1 ;
        }
        else if (count1 > count2) {
            return 1 ;
        }
        else {
            return 0 ;
        }
    }
}
```

Selain tipe data primitif seperti *int, String,* dan lainnya, struktur data yang digunakan dalam program ini meliputi *queue, priority queue,* dan *List*. Semua struktur data tersebut beserta implementasinya berasal dari *library* Java.

C. Pengujian

Pengujian 1 :

start : gray

end : dull

- Algoritma UCS

```
Welcome to Word Ladder Solver!
Start word : gray
End word : dull
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 1
[gray, dray, drat, doat, dolt, doll, dull]
Node visited : 2548
Execution time : 88 ms
```

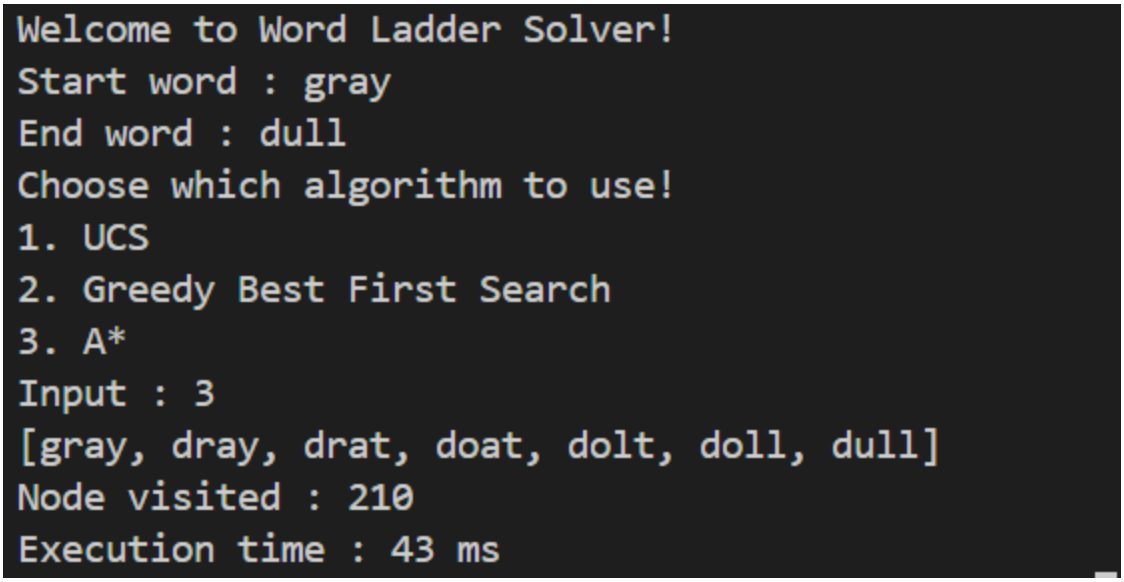
Gambar 1. Pengujian 1

- Algoritma GBFS

```
Welcome to Word Ladder Solver!
Start word : gray
End word : dull
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 2
[gray, dray, draw, drew, drek, dreg, drug, drum, drub, drib, drip, drop, trop, troy, trow, vrow, prow, pros, prop, prom, prog,
prof, prod, trod, trot, tret, trey, trek, tref, tree, dree, pree, prez, prey, prex, prep, peep, peel, weel, well, dell, dull]
Node visited : 41
Execution time : 43 ms
```

Gambar 2. Pengujian 2

- Algoritma A*



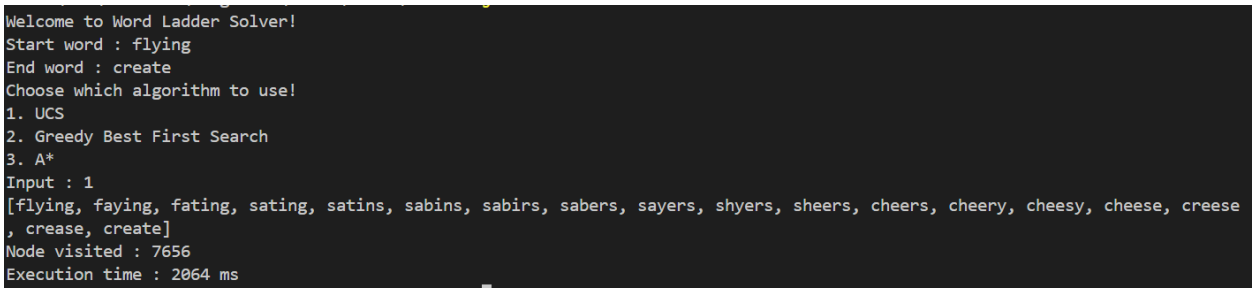
Gambar 3. Pengujian 3

Pengujian 2 :

start : flying

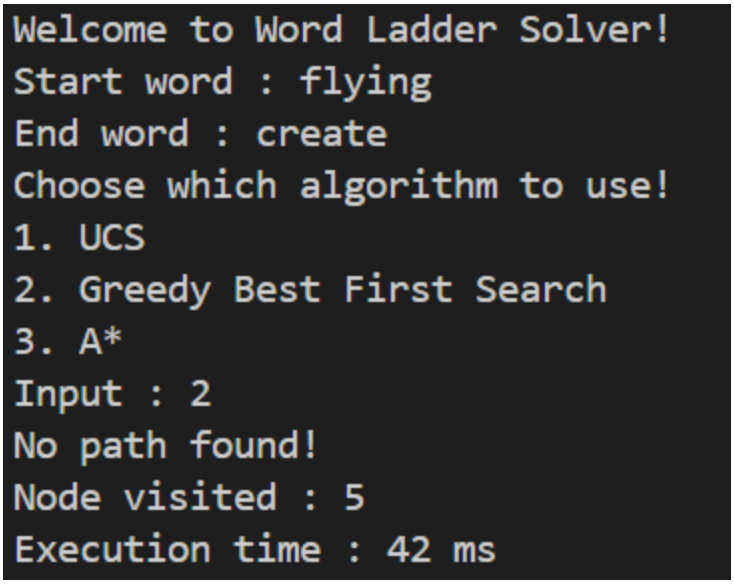
end : create

- Algoritma UCS



Gambar 4. Pengujian 4

- Algoritma GBFS



Gambar 5. Pengujian 5

- Algoritma A*

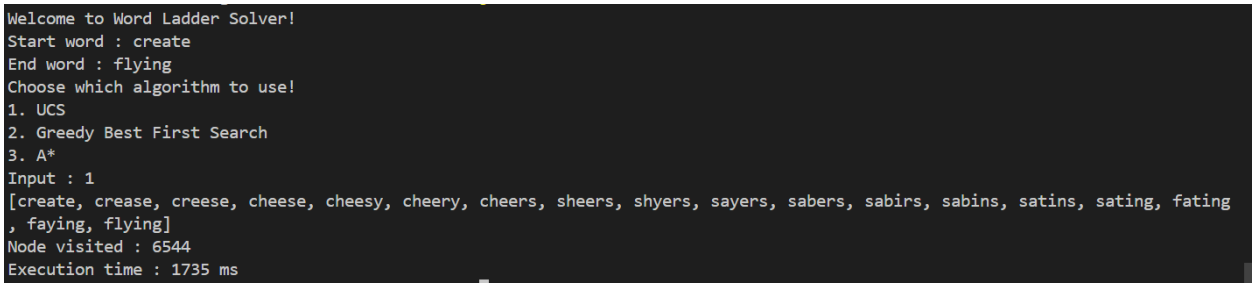
Gambar 6. Pengujian 6

Pengujian 3 :

start : create

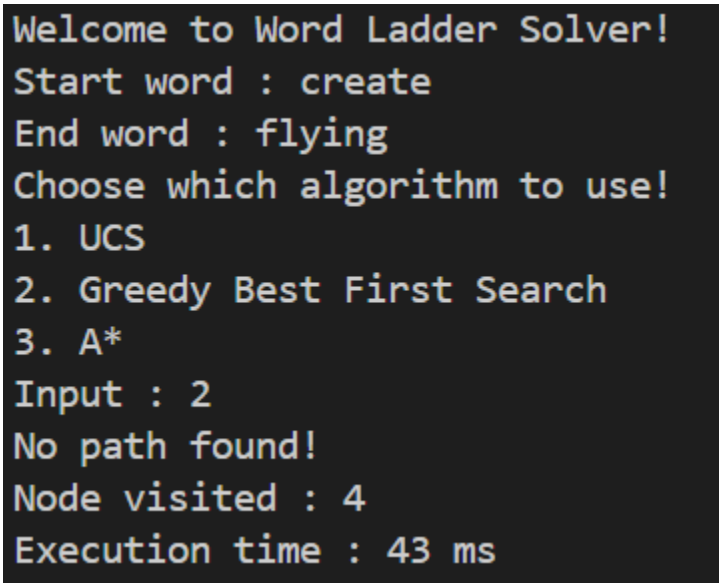
end : flying

- Algoritma UCS



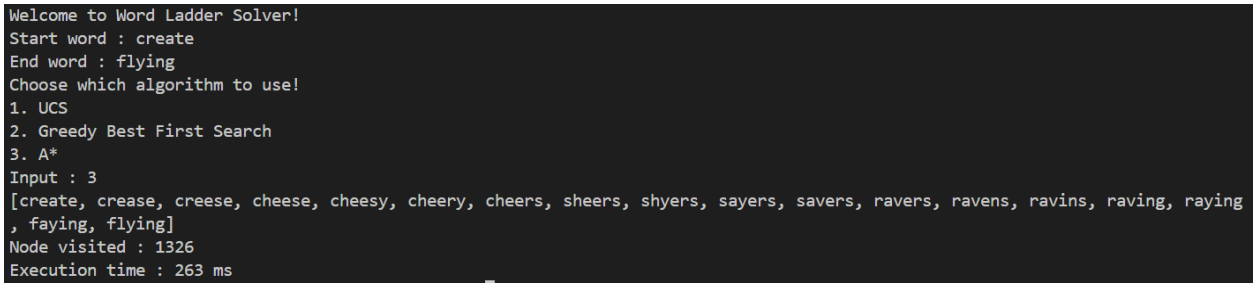
Gambar 7. Pengujian 7

- Algoritma GBFS



Gambar 8. Pengujian 8

- Algoritma A*



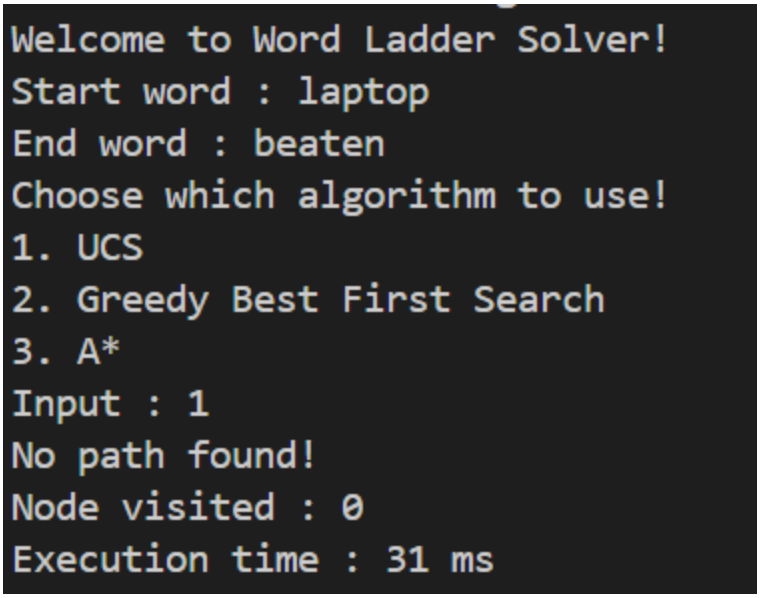
Gambar 9. Pengujian 9

Pengujian 4 :

start : laptop

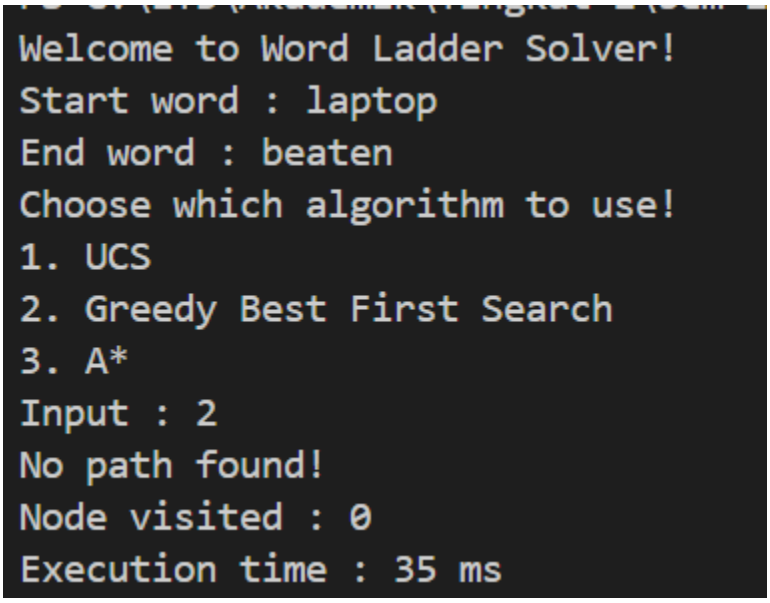
end : beaten

- Algoritma UCS



Gambar 10. Pengujian 10

- Algoritma GBFS



Gambar 11. Pengujian 11

- Algoritma A*

```
Welcome to Word Ladder Solver!
Start word : laptop
End word : beaten
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 3
No path found!
Node visited : 0
Execution time : 32 ms
```

Gambar 12. Pengujian 12

Pengujian 5 :

start : beef

end : hoof

- Algoritma UCS

```
Welcome to Word Ladder Solver!
Start word : beef
End word : hoof
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 1
[beef, been, peen, peon, poon, poof, hoof]
Node visited : 2868
Execution time : 120 ms
```

Gambar 13. Pengujian 13

- Algoritma GBFS

```
Welcome to Word Ladder Solver!
Start word : beef
End word : hoof
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 2
[beef, reef, reif, seif, serf, surf, turf, tuff, toff, doff, coff, coof, hoof]
Node visited : 12
Execution time : 35 ms
```

Gambar 14. Pengujian 14

- Algoritma A*

```
Welcome to Word Ladder Solver!
Start word : beef
End word : hoof
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 3
[beef, bees, gees, goes, goos, goof, hoof]
Node visited : 410
Execution time : 44 ms
```

Gambar 15. Pengujian 15

Pengujian 6 :

start : mouse

end : blobs

- Algoritma UCS

```
Welcome to Word Ladder Solver!
Start word : mous
It is not a word! Try Again!
Start word : mouse
End word : blobs
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 1
[mouse, rouse, route, routs, bouts, boots, blots, blobs]
Node visited : 3173
Execution time : 185 ms
```

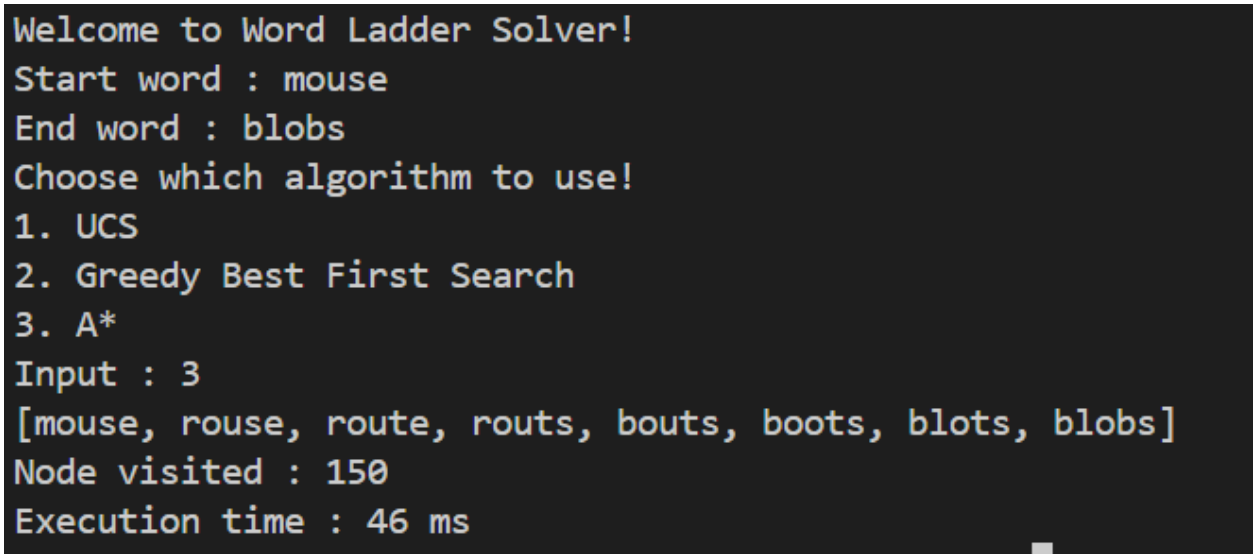
Gambar 16. Pengujian 16

- Algoritma GBFS

```
Welcome to Word Ladder Solver!
Start word : mouse
End word : blobs
Choose which algorithm to use!
1. UCS
2. Greedy Best First Search
3. A*
Input : 2
[mouse, moose, roose, roost, boost, boast, blast, blest, bless, blebs, blobs]
Node visited : 10
Execution time : 34 ms
```

Gambar 17. Pengujian 17

- Algoritma A*



Gambar 18. Pengujian 18

D. Hasil Analisis Solusi

Melihat pada pengujian pertama, hasil dari algoritma UCS dan A* sama persis sedangkan hasil dari algoritma GBFS lebih panjang dan tentunya bukan merupakan jalur terpendek. Hal ini disebabkan oleh sifat dari algoritma GBFS yang mencari optimum lokal. Terlebih lagi, optimum lokal yang ditemukan pada kasus *Word Ladder* cukup banyak pada satu waktu sehingga proses harus memilih dari beberapa optimum lokal sebelum mendapatkan jalur yang tepat. Hasil dari A* lebih cepat dan juga mengunjungi *node* yang lebih sedikit dari UCS.

Pada pengujian kedua, lagi-lagi UCS dan A* memiliki hasil jalur yang identik namun memiliki perbedaan waktu yang sangat berbeda. A* mendapatkan hasilnya dengan waktu 0,7 ms dan mengunjungi ±3000 *nodes* sedangkan UCS memiliki waktu 2 s dan mengunjungi ±7000 *nodes*. Terlihat dengan jelas, untuk mencapai hasil yang sama, A* jauh lebih efektif. Pada pengujian ini, GBFS tidak menemukan solusi. Hal ini dikarenakan proses GBFS tidak menemukan *child* dari *node* atau bisa dibilang proses terjebak dalam optimum lokal. Setelah diteliti lebih jauh, alur pencarian GBFS adalah sebagai berikut [flying, frying, crying, wrying, trying, truing]. Kata ‘truing’ hanya memiliki satu tetangga yaitu ‘trying’, namun ‘trying’ sudah pernah dikunjungi oleh *path*. Kondisi inilah yang menyebabkan terjadinya *stuck* padahal jelas terbukti oleh 2 algoritma

lainnya bahwa terdapat *path* menuju *goal*. Hal yang sama terjadi pula pada pengujian 3 namun dengan kata yang berbeda.

Untuk pengujian keempat, ketiga algoritma tidak mendapatkan hasil. Melihat *node* yang dikunjungi dan setelah diperiksa lebih lanjut, didapatkan bahwa “laptop” memang tidak memliki kata apapun dalam kamus dengan perbedaan satu huruf.

Pada pengujian kelima dan keenam tidak ada pengamatan yang berbeda untuk algoritma UCS dan A*. Namun dalam kedua pengujian tersebut, GBFS mendapatkan hasil lebih cepat dibanding kedua algoritma lainnya. Walaupun begitu, hasil yang didapatkan bukanlah hasil yang optimum atau bukan solusi terpendek.

Secara garis besar, *nodes* yang dikunjungi merepresentasikan memori yang digunakan oleh perangkat dalam menjalankan program. Algoritma GBFS memiliki *nodes* visited yang paling sedikit karena algoritma tersebut menjadikan semua *node* yang dibangkitkan dan tidak memeriksa jalur lain. Maka penggunaan memori pada algoritma ini adalah yang paling efisien. Algoritma A* memiliki penggunaan memori yang lebih baik dibanding algoritma UCS, melihat *node* yang dikunjungi selalu lebih sedikit dibanding UCS, apalagi pada pengujian keenam perbedaannya cukup ekstrim.

Berdasarkan pengujian dan analisis yang dilakukan, dapat disimpulkan bahwa algoritma A* merupakan algoritma paling optimal dalam menyelesaikan *Word Ladder* ini dengan waktu eksekusi paling cepat serta solusi terpendek. Algoritma UCS memiliki waktu eksekusi yang cukup standar tapi dipastikan mendapatkan hasil yang tepat. Namun, karena dalam kasus ini A* tidak mungkin *overestimate* dalam fungsi heuristicnya, maka algoritma A* juga dipastikan mendapatkan hasil yang tepat. Algoritma GBFS disimpulkan pilihan yang paling tidak dapat diandalkan dengan berbagai permasalahan yang telah dipaparkan di atas.

E. Repository

Tautan *repository github* : https://github.com/VansonK/Tucil3_13522049.git

Jika ada masalah hubungi ID Line : vansonk

Tabel Spesifikasi Program

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 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**