Tucil 3

Nama: Devinzen

NIM: 13522064

Algoritma UCS (Uniform Cost Search)

Algoritma UCS melakukan ekspansi di simpul yang costnya (g(n)) paling kecil. Dalam kasus word ladder, costnya dihitung dari langkah simpul awal menuju simpul tersebut.

Langkah-langkah implementasinya:

1. Melakukan ekspansi pada simpul awal.
2. Lakukan ekspansi pada simpul yang belum pernah dilakukan ekspansi yang costnya paling kecil. Kalau ada kata yang sama tidak akan dimasukkan.
3. Ulangi Langkah 2 sampai menemukan solusi atau sampai tidak ada lagi simpul yang bisa diekspansi (tidak ada solusi).

Pada algoritma UCS, setiap langkah hanya menambah 1 cost, dan karena simpul dibangkitkan dari yang costnya paling kecil, maka algoritma UCS pada kasus word ladder sama dengan algoritma BFS.

Algoritma Greedy Best First Search

Algoritma ini mencari simpul yang estimasi langkah ke tujuannya (f(n)) paling kecil. Dalam kasus word ladder, estimasi dihitung dari berapa banyak huruf yang berbeda dari solusinya.

Langkah-langkah implementasinya:

1. Melakukan ekspansi pada simpul terbaru (elemen terakhir dalam array).
2. Dari semua langkah yang mungkin, cari langkah yang estimasinya paling kecil dan belum ada dalam array. Kalau ada beberapa yang sama pilih kata dengan urutan abjad lebih kecil untuk dimasukkan ke array.
3. Ulangi langkah 2 sampai menemukan solusi atau simpul tidak bisa diekspansi (tidak ada solusi).

Pada implementasi algoritma ini tidak ada backtracking, sehingga mungkin saja tidak ditemukan solusi walaupun ada solusinya. Algoritma ini juga tidak menjamin solusinya optimal.

Algoritma A*

Algoritma A* melakukan ekspansi pada simpul yang h(n) nya paling kecil, dihitung dari f(n) (estimasi langkah) + g(n) (cost). Algoritma ini lebih efisien daripada UCS karena UCS tidak memperhatikan seberapa dekat simpul itu menuju solusi, jadi diexpand semuanya.

Langkah-langkah implementasinya:

1. Melakukan ekspansi pada simpul awal.

2. Lakukan ekspansi pada simpul yang belum pernah dilakukan ekspansi yang h(n) nya paling kecil. Kalau ada kata yang sama tidak akan dimasukkan.
3. Ulangi Langkah 2 sampai menemukan solusi atau sampai tidak ada lagi simpul yang bisa diekspansi (tidak ada solusi).

Heuristik yang digunakan pada algoritma ini admissible karena h(n) tidak overestimate. Karena satu langkah hanya boleh mengubah satu huruf, langkah yang diperlukan untuk menuju suatu kata tidak mungkin kurang dari jumlah huruf yang berbeda.

Source Code Java

```java
import java.util.Scanner;
import StringCheck.*;
import SearchAlgorithm.*;

public class Main {
    public static void main (String[] Args){
        // input
        Scanner scanner = new Scanner(System.in);
        String path = "../dictionary/dictionary.txt";
        System.out.println("Masukkan kata awal:");
        String start = scanner.nextLine();
        start = start.toLowerCase();
        while (!(StringCheck.checkValid(start) && StringCheck.checkDictionary(path, start, 1))){
            System.out.println("Masukkan kata awal:");
            start = scanner.nextLine();
            start = start.toLowerCase();
        }
        System.out.println("Masukkan kata yang ingin dicapai:");
        String dest = scanner.nextLine();
        dest = dest.toLowerCase();
        while (!(StringCheck.checkValid(dest) && StringCheck.checkSameLength(start, dest) && StringCheck.checkDictionary(path, dest, 1))){
            System.out.println("Masukkan kata yang ingin dicapai:");
            dest = scanner.nextLine();
            dest = dest.toLowerCase();
        }
        System.out.println("Pilih algoritma, masukkan angka diantara 1-3:\n1. Uniform Cost Search\n2. Greedy Best First Search\n3. A*");
        int choice = scanner.nextInt();
        while ((choice < 1) || (choice > 3)){
            System.out.println("Pilih algoritma, masukkan angka diantara 1-3:\n1. Uniform Cost Search\n2. Greedy Best First Search\n3. A*");
            choice = scanner.nextInt();
        }

        long startTime = System.nanoTime();

        // proses
        SearchAlgorithm.AddNode(new StringNode(start, 0, StringCheck.countEstimate(start, dest), -1));
        SearchAlgorithm.AddTask(0);
        int result;
        if (start.equals(dest)){result = 0;}
        else if (choice == 1){result = SearchAlgorithm.UCS(dest, path);}
        else if (choice == 2){result = SearchAlgorithm.Greedy(dest, path);}
        else {result = SearchAlgorithm.AStar(dest, path);}

        long endTime = System.nanoTime();
        long exeTime = (endTime - startTime) / 1000000;

        // output
        if (result == -1){System.out.println("Tidak ada solusi yang ditemukan");} else {
            System.out.println("Solusi ditemukan:");
            SearchAlgorithm.printPath(result);
        }
        System.out.println("Banyak node yang dikunjungi: " + SearchAlgorithm.getVisitedNodes());
        System.out.println("Waktu eksekusi: " + (exeTime) + " ms");
    }
}
```

Class Main untuk program utama

```java
package SearchAlgorithm;
import StringCheck.*;
import java.util.List;
import java.util.ArrayList;

public class SearchAlgorithm {
    // atribut
    private static List<StringNode> stringList = new ArrayList<StringNode>(); // list nodes
    private static List<Integer> nodesToExpand = new ArrayList<Integer>(); // list index node mana yang bisa di expand, untuk a*
    private static int VisitedNodes = 0;

    // getter setter
    public static void AddNode(StringNode n){stringList.add(n);}
    public static StringNode getNode(int index){return (stringList.get(index));}
    public static void AddTask(int n){nodesToExpand.add(n);}
    public static void RemoveTask(int idx){nodesToExpand.remove(idx);}
    public static void incrementVisitedNodes(){VisitedNodes++;}
    public static int getVisitedNodes(){return VisitedNodes;}

    // cek apakah string ada di list
    public static int getIndex(String s){
        for (int i = 0; i < stringList.size(); i++){
            if (s.equals(getNode(i).getWord())){return i;}
        }
        return -1;
    }

    // print start path to end path
    public static int printPath(int index){
        if(index == -1){return 0;} else {
            int x = 1;
            StringNode currentNode = getNode(index);
            x += printPath(currentNode.getParentIdx());
            System.out.println(x + ". " + currentNode.getWord());
            return x;
        }
    }

    // algoritma
    public static int UCS(String dest, String path){
        for (int i = 0; i < stringList.size(); i++){
            String currentWord = getNode(i).getWord();

            // expand
            incrementVisitedNodes();
            for (int pos = 0; pos < currentWord.length(); pos++){ // posisi huruf yang ditukar
                StringBuilder next = new StringBuilder(currentWord);
                for (char c = 'a'; c <= 'z'; c++){ // huruf yang ditukar
                    if (c != currentWord.charAt(pos)){
                        next.setCharAt(pos, c);
                        String maybeNext = next.toString();
                        if (StringCheck.checkDictionary(path, maybeNext, 2)){ // cek di kamus
                            if (getIndex(maybeNext) == -1){ // add kalau belum ada
                                StringNode newNode = new StringNode(maybeNext, (getNode(i).getCost() + 1), 0, i);
                                AddNode(newNode);
                                if (maybeNext.equals(dest)){return (stringList.size() - 1);}
                            }
                        }
                    }
                }
            }
        }
        return -1; // loop habis = no solution
    }

}
```

```
    public static int Greedy(String dest, String path){
        for (int i = 0; i < stringList.size(); i++){
            String currentWord = getNode(i).getWord();
            incrementVisitedNodes();
            StringNode nextWord = null;
            for (int pos = 0; pos < currentWord.length(); pos++){ // posisi huruf yang ditukar
                StringBuilder next = new StringBuilder(currentWord);
                for (char c = 'a'; c <= 'z'; c++){ // huruf yang ditukar
                    if (c != currentWord.charAt(pos)){
                        next.setCharAt(pos, c);
                        String maybeNext = next.toString();
                        if (StringCheck.checkDictionary(path, maybeNext, 2)){ // cek di kamus
                            // kalau sama langsung return
                            if (maybeNext.equals(dest)){
                                StringNode newNode = new StringNode(maybeNext, 0, StringCheck.countEstimate(maybeNext, dest), i);
                                AddNode(newNode);
                                return (stringList.size() - 1);
                            }

                            // memilih apakah dimasukkan ke array
                            if (getIndex(maybeNext) == -1){
                                if (nextWord == null){nextWord = new StringNode(maybeNext, 0, StringCheck.countEstimate(maybeNext, dest), i);}
                                else {
                                    StringNode maybeReplaced = new StringNode(maybeNext, 0, StringCheck.countEstimate(maybeNext, dest), i);
                                    if ((maybeReplaced.getEstimate() < nextWord.getEstimate()) || ((maybeReplaced.getEstimate() == nextWord.getEstimate()) && (maybeReplaced.getWord().c
                                        nextWord = maybeReplaced;
                                    }
                                }
                            }
                        }
                    }
                }
            }
            if (nextWord != null){
                AddNode(nextWord);
            }
        }
        return -1; // loop habis = no solution
    }
```

```
    public static int AStar(String dest, String path){
        while (nodesToExpand.size() > 0){
            // cari index
            int idx = 0;
            int minhn = getNode(nodesToExpand.get(0)).hn();
            for (int i = 1; i < nodesToExpand.size(); i++){
                int HN = getNode(nodesToExpand.get(i)).hn();
                if (HN < minhn){
                    idx = i;
                    minhn = HN;
                }
            }
            int listidx = nodesToExpand.get(idx);
            String currentWord = getNode(listidx).getWord();
            // expand
            incrementVisitedNodes();
            RemoveTask(idx);
            for (int pos = 0; pos < currentWord.length(); pos++){ // posisi huruf yang ditukar
                StringBuilder next = new StringBuilder(currentWord);
                for (char c = 'a'; c <= 'z'; c++){ // huruf yang ditukar
                    if (c != currentWord.charAt(pos)){
                        next.setCharAt(pos, c);
                        String maybeNext = next.toString();
                        if (StringCheck.checkDictionary(path, maybeNext, 2)){ // cek di kamus
                            if (getIndex(maybeNext) == -1){ // add kalau belum ada
                                StringNode newNode = new StringNode(maybeNext, (getNode(listidx).getCost() + 1), StringCheck.countEstimate(maybeNext, dest), listidx);
                                AddNode(newNode);
                                AddTask(stringList.size() - 1);
                                if (maybeNext.equals(dest)){return (stringList.size() - 1);}
                            }
                        }
                    }
                }
            }
        }
        return -1; // loop habis = no solution
    }
}
```

Class SearchAlgorithm digunakan untuk algoritma utamanya.

List stringList untuk menyimpan simpul.

Method getIndex mengembalikan index string pada list (mengembalikan -1 kalau tidak ditemukan).

Method printPath mengeluarkan path dari simpul awal menuju simpul akhir.

Method UCS, Greedy, dan AStar adalah algoritma pencarian.

```java
package StringCheck;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

public class StringCheck {
    // validasi string
    public static boolean checkSameLength(String src, String dest){
        if (src.length() != dest.length()){
            System.out.println("Kedua kata panjangnya harus sama!!!");
            return false;
        }
        return true;
    }

    public static boolean checkValid (String s){
        if ((s.length() < 2) || (s.length() > 8)){
            System.out.println("Katanya harus punya panjang 2 sampai 8 huruf!!!");
            return false;
        }
        if (s.matches(".*[^a-z].*")){
            System.out.println("Kata tidak boleh mengandung angka atau karakter lain selain huruf!!!");
            return false;
        }
        return true;
    }

    // baca line dari file txt
    public static String getWordAtLine(String path, long line) throws IOException {
        try (BufferedReader br = new BufferedReader(new FileReader(path))) {
            return br.lines().skip(line).findFirst().orElse(null);
        }
    }

```

```java
    // cek binary search
    public static boolean checkDictionary (String path, String s, int mode){
        try (BufferedReader br = new BufferedReader(new FileReader(path))){
            long bottom = 0;
            long top = br.lines().count() - 1;
            long mid;
            String word;
            while (bottom <= top) {
                mid = (bottom + top) / 2;
                word = getWordAtLine(path, mid);
                if (word == null){return false;}
                int cumpare = s.compareTo(word);
                if (cumpare == 0) {
                    return true;
                } else if (cumpare < 0) {
                    top = mid - 1;
                } else {
                    bottom = mid + 1;
                }
            }
        } catch (IOException e) {
            System.out.println("File tidak ditemukan, tidak bisa validasi kata...");
            return false;
        }
        if (mode == 1) {System.out.println("Kata tidak ditemukan di dictionary!!!");}
        return false;
    }

    // hitung estimasi f(n)
    // kedua string lengthnya sama
    public static int countEstimate(String word, String dest){
        int fn = 0;
        for (int i = 0; i < word.length(); i++){
            if (word.charAt(i) != dest.charAt(i)){fn++;}
        }
        return fn;
    }
}
```

Class StringCheck berisi fungsi-fungsi untuk string.

Method checkSameLength dan checkValid untuk validasi string input.

Method getWordAtLine mengembalikan kata di kamus untuk pencarian.

Method checkDictionary mencari apakah suatu kata merupakan kata Bahasa Inggris yang valid di kamus.

Method countEstimate menghitung jumlah huruf yang berbeda untuk estimasi f(n).

```java
package SearchAlgorithm;

public class StringNode {
    // atribut
    private String word;
    private int cost; // g(n)
    private int estimate; // f(n)
    private int parent; // pointer list

    // konstruktor
    public StringNode(String s, int gn, int fn, int p){
        word = s;
        cost = gn;
        estimate = fn;
        parent = p;
    }

    // getter setter
    public String getWord(){return word;}
    public int getCost(){return cost;}
    public int getEstimate(){return estimate;}
    public int hn(){return (cost + estimate);} // h(n) untuk algoritma A*
    public int getParentIdx(){return parent;}
}
```

Class StringNode berfungsi sebagai container untuk simpul, dengan method hn menghitung estimasi h(n).

Tangkapan Layar Test

```
Masukkan kata awal:
egg
Masukkan kata yang ingin dicapai:
top
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. egg
2. erg
3. ern
4. eon
5. ton
6. top
Banyak node yang dikunjungi: 127
Waktu eksekusi: 261397 ms
```

```
Masukkan kata awal:
egg
Masukkan kata yang ingin dicapai:
top
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. egg
2. ego
3. ago
4. abo
5. aba
6. abs
7. aas
8. tas
9. tap
10. top
Banyak node yang dikunjungi: 9
Waktu eksekusi: 13266 ms
```

```
Masukkan kata awal:
egg
Masukkan kata yang ingin dicapai:
top
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. egg
2. erg
3. ern
4. eon
5. ton
6. top
Banyak node yang dikunjungi: 14
Waktu eksekusi: 25822 ms
```

```
Masukkan kata awal:
java
Masukkan kata yang ingin dicapai:
code
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. java
2. fava
3. fave
4. cave
5. cove
6. code
Banyak node yang dikunjungi: 78
Waktu eksekusi: 214958 ms
```

```
Masukkan kata awal:
Java
Masukkan kata yang ingin dicapai:
cOd3
Kata tidak boleh mengandung angka atau karakter lain selain huruf!!!
Masukkan kata yang ingin dicapai:
cOdE
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. java
2. fava
3. fave
4. cave
5. cade
6. code
Banyak node yang dikunjungi: 5
Waktu eksekusi: 10438 ms
```

```
Masukkan kata awal:
java
Masukkan kata yang ingin dicapai:
code
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. java
2. fava
3. fave
4. cave
5. cove
6. code
Banyak node yang dikunjungi: 11
Waktu eksekusi: 28256 ms
```

```
Masukkan kata awal:
COLDER
Masukkan kata yang ingin dicapai:
SOLVER
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. colder
2. solder
3. solver
Banyak node yang dikunjungi: 8
Waktu eksekusi: 73703 ms
```

```
Masukkan kata awal:
colder
Masukkan kata yang ingin dicapai:
solver
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. colder
2. solder
3. solver
Banyak node yang dikunjungi: 2
Waktu eksekusi: 14046 ms
```

```
Masukkan kata awal:
colder
Masukkan kata yang ingin dicapai:
solver
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. colder
2. solder
3. solver
Banyak node yang dikunjungi: 2
Waktu eksekusi: 17095 ms
```

```
Masukkan kata awal:
casts
Masukkan kata yang ingin dicapai:
roses
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. casts
2. costs
3. coses
4. roses
Banyak node yang dikunjungi: 82
Waktu eksekusi: 737339 ms
```

```
Masukkan kata awal:
casts
Masukkan kata yang ingin dicapai:
roses
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. casts
2. cases
3. coses
4. roses
Banyak node yang dikunjungi: 3
Waktu eksekusi: 11978 ms
```

```
Masukkan kata awal:
CASTS
Masukkan kata yang ingin dicapai:
ROSES
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. casts
2. costs
3. coses
4. roses
Banyak node yang dikunjungi: 4
Waktu eksekusi: 16370 ms
```

```
Masukkan kata awal:
c
Katanya harus punya panjang 2 sampai 8 huruf!!!
Masukkan kata awal:
ccccccccc
Katanya harus punya panjang 2 sampai 8 huruf!!!
Masukkan kata awal:
cc
Kata tidak ditemukan di dictionary!!!
Masukkan kata awal:
cosigned
Masukkan kata yang ingin dicapai:
cosigner
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. cosigned
2. cosigner
Banyak node yang dikunjungi: 1
Waktu eksekusi: 9092 ms
```

```
Masukkan kata awal:
cosigned
Masukkan kata yang ingin dicapai:
extreme
Kedua kata panjangnya harus sama!!!
Masukkan kata yang ingin dicapai:
cOsIgnEr
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. cosigned
2. cosigner
Banyak node yang dikunjungi: 1
Waktu eksekusi: 10063 ms
```

```
PS C:\Users\devin\Tucil3_13522064\bin> java Main
Masukkan kata awal:
cosigned
Masukkan kata yang ingin dicapai:
cosigner
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. cosigned
2. cosigner
Banyak node yang dikunjungi: 1
Waktu eksekusi: 9799 ms
```

```
Masukkan kata awal:
fuci
Masukkan kata yang ingin dicapai:
BUCK
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
1
Solusi ditemukan:
1. fuci
2. fuck
3. buck
Banyak node yang dikunjungi: 5
Waktu eksekusi: 24523 ms
```

```
PS C:\Users\devin\Tucil3_13522064\bin> java Main
Masukkan kata awal:
FUCI
Masukkan kata yang ingin dicapai:
buck
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
2
Solusi ditemukan:
1. fuci
2. fuck
3. buck
Banyak node yang dikunjungi: 2
Waktu eksekusi: 4817 ms
```

```
Masukkan kata awal:
FUCI
Masukkan kata yang ingin dicapai:
BUCK
Pilih algoritma, masukkan angka diantara 1-3:
1. Uniform Cost Search
2. Greedy Best First Search
3. A*
3
Solusi ditemukan:
1. fuci
2. fuck
3. buck
Banyak node yang dikunjungi: 2
Waktu eksekusi: 3083 ms
```

Perbandingan

Solusi yang diberikan UCS dan A* optimal karena memperhitungkan cost yang sudah dilalui, sedangkan Solusi dari Greedy best first search belum tentu optimal.

Waktu eksekusi UCS paling lama karena membangkitkan simpul tanpa memperhitungkan estimasi ke tujuan, sedangkan Greedy best first search dan A* lebih cepat.

Memori yang dibutuhkan algoritma Greedy lebih kecil daripada UCS dan A* karena pada implementasi di sini tidak ada backtracking.

Link Repository Github

https://github.com/Devinzenzhang/Tucil3_13522064