Implementasi Algoritma IDS Dalam Menyelesaikan Permainan Text Twist dan Optimalisasi Menggunakan Algoritma Backtrack

Muhammad Syarafi Akmal – 13522076 [1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
[1] 13522076@std.stei.itb.ac.id

***Abstract*—Tugas makalah ini memiliki topik terkait implementasi algoritma DFS dalam menyelesaikan permainan *puzzle* Text Twist. Text twist adalah sebuah permainan teka-teki kata berbahasa inggris yang melibatkan penyusunan kata-kata (dengan panjang 3 hingga 6 karakter) dari 6 huruf acak yang disediakan. Dengan menggunakan konsep algoritma IDS, solusi dari teka-teki ini dapat ditelusuri dengan menyusun kata-kata dengan 1 huruf per level. Hal ini membuat algoritma IDS menulusuri semua kemungkinan, bahkan yang bukan merupakan solusi. Oleh karena itu, algoritma *backtrack* digunakan untuk optimalisasi yang nantinya akan dibandingkan berdasarkan waktu eksekusinya.**

Keywords—Text Twist; Depth First Search; Backtrack; Perbandingan

I. PENDAHULUAN

Text Twist adalah sebuah permainan teka-teki kata yang memberikan tantangan kepada pemainnnya untuk menyusun kata-kata sebanyak mungkin atau sebanyak batasan dari permainan tersebut berdasarkan kumpulan huruf-huruf yang diberikan dengan waktu yang terbatas (2 menit). Mekanisme menantang tersebut membuat permainan ini popular bahkan hingga sekarang karena dapat meningkatkan kemampuan *vocabulary* dan *cognitive*.

Permainan online Text Twist biasanya akan memberikan 6 huruf acak dan kata-kata yang perlu disusun dengan panjang kata antara 3 hingga 6 kata yang akan disusun pada tabel disediakan sejumlah dengan kata yang perlu dicari.

Gambar 1.1, Tampilan awal Text Twist.

(Sumber : https://zone.msn.com/en/texttwist/default.htm)

Dari huruf-huruf yang disediakan, pemain perlu menyusun kata dengan menekan secara sekuensial huruf-huruf ke kotak biru yang tersedia di bagian atasnya. Setelah pemain menyusun kata, pemain perlu menekan tombol enter dan kata yang telah disusun akan tercantum pada tabel menandakan kata tersebut sudah disusun.

Gambar 1.2, Tampilan permainan setelah menyusun kata.

(Sumber : https://zone.msn.com/en/texttwist/default.htm)

Gambar 1.3, Tampilan permainan setelah menekan tombol enter.

(Sumber : https://zone.msn.com/en/texttwist/default.htm)

Sebagai *engineer*, hal yang terlintas di pikiran kita saat melihat permainan ini adalah untuk melakukan otomasi penyelesaian (walaupun tujuan permainannya untuk mengasah

otak). Oleh karena itu, makalah ini akan membahas terkait penggunaan pendekatan algoritma pencarian *Iterative Deepening Search* (IDS) dan *Backtrack* untuk melakukan optimalisasi untuk menyelesaikan teka-teki ini. Kedua algoritma ini akan dibandingkan hingga dapat diambil kesimpulan apakah *Backtrack* akan membuat algoritma pencarian IDS menjadi lebih mangkus.

II. Landasan Teori

A. *Iterative Deepening Search (IDS)*

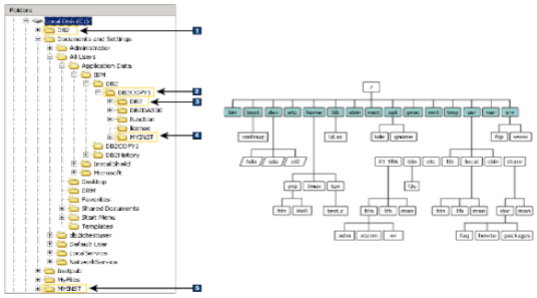
Iterative Deepening Search (IDS) adalah sebuah algoritma pencarian fundamental yang merupakan perpaduan antara *Depth Limited Search* (DLS) yang mirip dengan *Depth First Search* (DFS) dengan batasan level pohon penelusuran. Sehingga *on a certain extent*, IDS merupakan BFS yang menelusuri tiap *branch* pada kedalaman tertentu.

IDS menerapkan DLS secara iterative, dengan artian IDS akan melakukan iterasi DLS terhadap batasan tingkat level pohon penelusurannya.

a. *Depth First Search* (DFS)

DFS adalah algoritma pencarian secara traversal (*searching tree*). Dimulai dari *root node*, DFS akan menelusuri masing-masing *branch* mendalam hingga ditemukan solusi (ada kemungkinan tidak menemukan solusi). Di dunia komputasi, DFS digunakan dalam menyelesaikan teka-teki, *web crawling*, *directory search*, dan sebagainya.

DFS dan BFS untuk penelusuran direktori (folder)



Gambar A.1, DFS and BFS *directory searching*.

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf)

b. *Depth Limited Search* (DLS)

DLS adalah ekstensi dari algoritma DFS dimana penelusuran pohon dibatasi pada level tertentu. DLS memungkinkan untuk penerapan DFS agar tidak mengalami penelusuran yang tak terbatas pada satu branch. Di dunia komputasi, DLS digunakan pada *Artificial Intelligence* untuk penelusuran permasalahan yang memiliki ruang penelusuran yang besar dan *exhaustive search* tidak memungkinkan untuk dilakukan.

IDS memungkinkan untuk memadukan optimalitas pencarian BFS (karena DFS belum tentu menemukan solusi) dan efisiensi memori yang merupakan kelebihan dari DFS.

B. *Backtrack*

Backtracking adalah algoritma yang merupakan ekstensi dari DFS. Dalam backtracking, penelusuran dilakukan dengan kemungkinan untuk memotong cabang pada pohon penelusuran (pruning) jika ditemukan bahwa cabang tersebut tidak mengarah pada solusi. Hal ini dimungkinkan karena adanya bound function (fungsi pembatas) yang mengevaluasi apakah suatu cabang layak untuk terus dieksplorasi.

Berikut adalah komponen umum dari algoritma *backtrack*:

a. Solusi persoalan X = (x_1, x_2, x_3, …, x_i), x_i merupakan elemen dari himpunan input atau status yang harus di pilih setiap *node*.

b. Fungsi pembangkit T (x_1, x_2, x_3, …, x_k), membangkitkan x_k yang merupakan komponen solusi untuk tiap branch yang masih bisa dibangkitkan.

c. Fungsi pembatas B (x_1, x_2, x_3, …, x_k), mengevaluasi tiap status dari bangkitan sebelum atau setelah melakukan fungsi pembatas. B akan bernilai *true* jika hasil bangkitan mengarah ke solusi atau tidak melanggar *constraint* yang sudah ditentukan. Jika B bernilai *false* maka akan dilakukan *pruning* (dibuang), dan akan dilanjutkan untuk sebaliknya.

Contoh penggunaan *backtrack*:

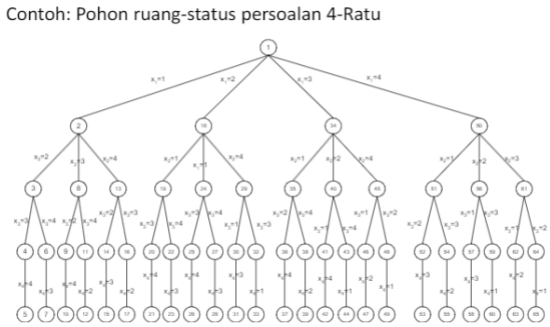
a. N-Queens *problem*, *backtracking* digunakan untuk menempatkan N ratu di papan catur NxN sehingga tidak ada dua ratu yang saling menyerang. Algoritma ini membangkitkan posisi ratu satu per satu dan menggunakan fungsi pembatas untuk memastikan bahwa tidak ada dua ratu yang berada dalam satu baris, kolom, atau diagonal yang sama. Namun, *pendekatan* penyelesaian solusi *backtracking* ini memungkinkan untuk tidak perlu mengkhawatirkan penempatan ratu pada baris sehingga lebih efisien saat melakukan pencarian solusi.

Komponen utama *backtracking* dari 4x4 N-Queens problem:

1. Solusi persoalan X = (x_1, x_2, x_3, x_4) $x_i \in \{1, 2, 3, 4\}$, dengan x_i menyatakan nilai indeks dari kolom dan indeks pada X menyatakan baris pada papan catur.
2. Fungsi pembangkit T (x_1, x_2, x_3, …, x_k), maka untuk tiap x_k, kemungkinan nilai yang akan dibangkitkan merupakan semua elemen dari x_i yang masih memenuhi *bound function*.
3. Fungsi pembatas B (x_1, x_2, x_3, …, x_k), untuk tiap x_k, B(x_k) harus memenuhi beberapa syarat untuk bernilai *true* yaitu:

- Nilai bangkitan yang merupakan elemen dari x_i tidak boleh terkandung di *parent-parent* dari *node* tersebut.
- Nilai bangkitan yang merupakan elemen dari x_i tidak boleh menjadi diagonal dari *parent-parent node* tersebut. Berikut adalah pseudocode untuk pengecekan diagonal:

```
if abs (parent.col – child.col) = abs
(parent.row – child.row) then
  ➔ true // maka terdapat diagonal
endif
  ➔ false
```



Gambar B.1, pohon solusi dari penyelesaian N-Queens menggunakan *backtrack*.

(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf)

Aplikasi *backtrack* dalam dunia komputasi:

a. **Pemecahan Teka-teki:** *Backtrack* sering digunakan dalam memecahkan teka-teki seperti Sudoku, teka-teki silang, dan berbagai permainan papan seperti N-Queens atau masalah Knight's Tour.

b. **Optimasi Kombinatorial:** *Backtrack* digunakan dalam masalah optimasi kombinatorial di mana Anda perlu menemukan solusi terbaik dari sejumlah kemungkinan yang sangat besar. Contohnya termasuk Travelling Salesman Problem dan Subset Sum Problem.

c. **Network routing:** *Backtrack* dapat digunakan dalam algoritma perutean jaringan untuk menemukan jalur optimal antara dua titik dalam jaringan.

d. **Pemrosesan Bahasa Alami:** *Backtrack* digunakan dalam tugas pemrosesan bahasa alami seperti penguraian dan pemeriksaan tata bahasa.

e. **Algoritma Genetika**: *Backtrack* dapat digunakan dalam algoritma genetika untuk menjelajahi ruang pencarian secara efisien guna menemukan solusi optimal.

f. **Constraint Satisfaction Problems:** *Backtrack* biasanya digunakan dalam menyelesaikan *constraint satisfaction problems* di mana Anda perlu mencari nilai variabel yang memenuhi serangkaian batasan.

Backtracking memungkinkan optimalisasi dari algoritma pencarian yang menerapkan exhaustive search seperti BFS, DFS, dan Brute Force. Akan tetapi, backtracking hanya efektif jika karakteristik permasalahan dapat ditentukan dari awal, sehingga fungsi pembatas dapat diaplikasikan dengan tepat untuk memangkas cabang yang tidak produktif dan menghemat waktu serta sumber daya komputasi.

III. IMPLEMENTASI

Penyelesaian Text Twist akan diimplementasikan dalam bahasa pemrograman Java dan akan membutuhkan referensi kamus yang cukup *reliable* yaitu kamus oracle [1]. Karena Text Twist online merupakan permainan yang cukup tua, maka akan ada beberapa referensi dari kamus oracle yang tidak akan bisa di-*submit* ke permainan.

A. Implementasi dalam IDS

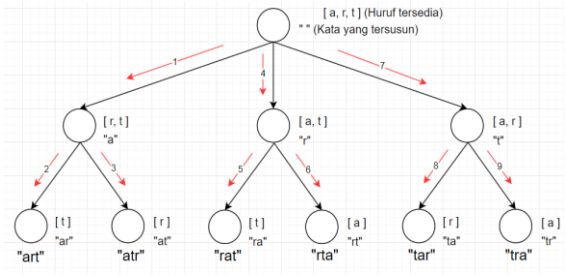
Penyelesaian ini dilakukan dengan pendekatan IDS karena kita perlu melakukan proses DLS sebanyak rentang panjang kata yang perlu dicari (dalam kasus Text Twist online, panjang kata memiliki rentang 3-6 huruf). Penyelesaian secara individual juga diimplementasikan dengan DLS karena sudah memiliki *depth* yang pasti (3-6 huruf). DLS disini digunakan untuk melakukan permutasi terhadap semua kemungkinan susunan dari huruf yang disediakan. Pada program ini, aka nada tig akelas utama yaitu kelas yang menangani algoritma, menangani main program, dan menangani validasi kata.

a. Pemodelan pohon solusi

Pada permasalahan ini, tiap *node* pada pohon solusi akan membawa informasi berupa:

- Huruf yang tersedia (dalam bentuk array), hal ini diperlukan karena kita akan melakukan pencarian exhaustive terhadap setiap huruf yang akan disusun dan untuk meminimalisir proses pencarian, apabila dilakukan dengan menelusuri setiap kemungkinan susunan dari semua alphabet dan harus melakukan pengecekan lagi maka akan terjadi proses yang boros.
- Susunan kata yang sudah dibuat, tentu saja bagian ini diperlukan untuk mendapatkan hasil akhir.
- Kedalaman yang sudah ditelusuri, bagian ini meruapakan komponen agar DLS dapat berjalan dengan sesuai (untuk proses terminasi).

Melihat pemodelan yang sudah dibuat, maka pendekatan penyelesaian akan dilakukan secara rekursif agar tidak ada tumpang tindih data antar proses branch lain. Berikut adalah pohon solusi yang dirancang dengan kasus 3 huruf {a, r, t}.



Gambar A.1, Ilustrasi pohon solusi DFS.

Gambar A.1 merupakan ilustrasi dari hasil pohon solusi yang didapatkan melalui proses DLS. Dapat dilihat proses penelusuran akan dilakukan dari branch paling kiri terlebih dahulu dan akan ditelusuri hingga mencapai batasan yaitu apabila sudah mencapai kedalaman sebesar jumlah kata yang tersedia di awal yaitu 3 (iterasi ke-3 disimplifikasi menjadi "art", dll.).

Diperoleh semua susunan yang didapat berupa :

Himpunan calon kata = {art, atr, rat, rta, tar, tra}

b. Pemodelan Algoritma

Penyelesaian akan kita lakukan secara rekursif karena lebih natural. Berikut merupakan komponen dari rekursi:

- Basis, basis dari rekursi ini adalah saat parameter kedalaman penelusuran sudah tercapai. Ketika berada di kasus ini, maka perlu dilakukan validasi terhadap hasil kata yang disusun berdasarkan referensi kamus oracle [1].
- Rekursi, apabila kita mengeneralisasi proses ekspansi pohon solusi maka dapat dilihat bahwa, secara traversal, rekursi akan dipanggil dengan parameter huruf tersedia yang sudah direduksi dan parameter susunan kata yang sudah diperbarui. Traversal dilakukan berdasarkan huruf yang tersedia.

Berikut adalah *pseudocode* untuk algoritma IDS Text Twist:

```
function recursiveTextTwistDFS(letters,
currentWord, depth):

   if depth = 0 and currentWord.validWord and
currentWord not in result then

      result.add(currentWord)

   endif

   letter in letters traversal
     newWord ← currentWord.append(letter)

     remainingLetters ← letters.remove(letter)

     recursiveTextTwistDFS (remainingLetters,
newWord, depth-1)
```

Penjelasan komponen *pseudocode*:

- Letters: parameter yang menampung huruf yang tersedia untuk penyusunan (list of char/string).
- currentWord: parameter yang merupakan kondisi susunan kata di sebuah *node*.
- depth: parameter yang digunakan sebagai terminator proses DLS.
- Penjelasan alur: 1) Fungsi akan memeriksa apabila depth bernilai 0, currentWord tidak ada di result (agar tidak duplikat), dan currentWord adalah kata yang valid sebagai basis/terminasi. 2) Apabila bukan, fungsi akan melakukan traversal terhadap seluruh huruf di letters, currentWord akan di append dengan letter sebagai newWord dan akan menginisiasi *clone* dari letters dengan elemen letter yang sudah dihapus sebagai remainingLetters. Lalu, fungsi akan memanggil rekursi dengan parameter baru di tempat yang berkesesuaian dan depth dikurang 1.

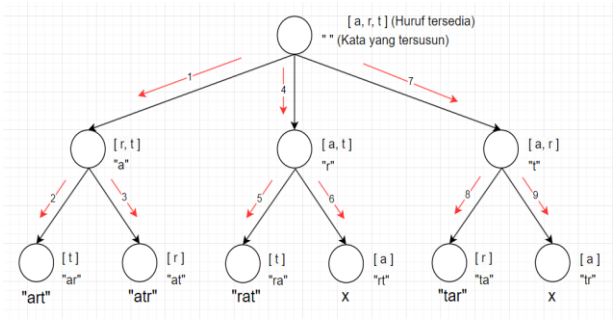
B. Implementasi algoritma Backtrack

Pada algoritma IDS terdapat proses yang boros yaitu ketika menelusuri susunan huruf yang tidak mungkin menjadi kata. Kasus ini dapat dilihat pada gambar A.1 di bagian "rta". Apabila kita lihat di kontruksi sebelumnya "rt", kita dapat mengambil kesimpulan bahwa dari susunan huruf ini tidak mungkin dibentuk sebuah kata. Alhasil, proses IDS menjadi boros karena akan menggunakan memori yang tidak berguna dan akan memperlabat waktu eksekusi. Hal ini akan menjadi penentu komponen utama *bounding function* pada penyusunan algoritma *backtrack*.

a. Penentuan komponen *backtrack*

- Solusi persoalan X = (x_1, x_2, x_3, …, x_i) x_i ∈ {himpunan huruf yang tersedia}. (X merupakan string).
- Fungsi pembangkit T (x_1, x_2, x_3, …, x_k), maka untuk tiap x_k, kemungkinan nilai yang akan dibangkitkan merupakan semua elemen dari x_i yang masih memenuhi *bound function.*
- Fungsi pembatas B (x_1, x_2, x_3, …, x_k), untuk tiap x_k, B(x_k) harus memenuhi beberapa syarat untuk bernilai *true* yaitu:
  1. Nilai x_k harus merupakan prefix dari kata dengan length i.

Berikut adalah pohon solusi yang didapat berdasarkan kasus 3 huruf {a, r, t}.



Gambar B.1, ilustrasi pohon solusi *backtrack*.

b. Pemodelan algoritma

Backtracking adalah teknik pemrograman yang digunakan untuk menyelesaikan masalah dengan mencoba membangun solusi secara bertahap, satu bagian pada satu waktu, dan menghilangkan solusi yang gagal untuk memenuhi batasan masalah di setiap langkah. Teknik ini sangat mirip dengan Depth-First Search (DFS), tetapi dengan tambahan mekanisme untuk memeriksa dan menghentikan ekspansi cabang tertentu lebih awal jika diketahui bahwa cabang tersebut tidak dapat menghasilkan solusi yang valid (fungsi bound).

Berikut adalah *pseudocode* untuk algoritma *backtrack* Text Twist:

```
function recursiveTextTwistDLS(letters,
currentWord, depth):

   if depth = 0 and currentWord.validWord and
currentWord not in    result then

      result.add(currentWord)

   endif

   if (boundFunc(currentWord) then

      letter in letters traversal
        newWord ← currentWord.append(letter)

        remainingLetters ←
letters.remove(letter)

        recursiveTextTwistDFS (remainingLetters,
newWord, depth-1)

   endif

function boundFunc(word)

   → word.isPrefixValid
```

Pada *pseudocode* di atas, hal yang diubah hanya di bagian traversal, fungsi hanya akan melakukan traversal apabila fungsi bound bernilai true yaitu currentWord adalah sebuah prefix dari kata dengan panjang yang sesuai dengan depth dan penambahan fungsi *bound*.

C. Pemodelan kelas di Java

Program ini akan memiliki 3 kelas utama yaitu kelas main, kelas TextTwist, dan kelas WordValidator.

a. WordValidator

Kelas ini akan digunakan untuk menampung kamus dan metode-metode yang relevan. Kelas ini dibuat dengan tujuan agar instansiasinya spesifik untuk panjang kata tertentu. Hal ini membuat kata-kata yang ditampung akan lebih sedikit dan membuat program lebih efektif saat melakukan pencarian. Berikut adalah gambar kelas WordValidator:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class WordValidator {
    public String dictionary_filename = "oracle_dictionary.txt";
    public List<String> dictionary_map = new ArrayList<String>();

    public WordValidator(int length) {···

    public Boolean isPrefixValid(String word) {···

    public Boolean isWordValid(String word) {···
}
```

Gambar C.1, desain kelas WordValidator.

Penjelasan komponen kelas WordValidator:

- Dictionary_filename, atribut ini berfungsi sebagai *placeholder* path menuju referensi kamus. Tujuan eksistensinya adalah apabila dari user ingin mengubah referensi kamus di tengah main program.
- Dictionary_map, atribut ini berfungsi untuk menampung semua kata yang ada di kamus sesuai dengan panjang dari katanya yang akan diinisiasi di konstruktor.
- isPrefixValid, metode ini berfungsi untuk memeriksa apakah sebuah prefix valid berdasarkan dictionary_map.
- isWordValid, metode ini berfungsi untuk memeriksa apakah sebuah kata valid berdasarkan dictionary_map.

b. TextTwist (Algoritma)

Kelas ini digunakan untuk menampung algoritma-algoritma yang akan digunakan untuk menyelesaikan permainan Text Twist.

```java
import java.util.ArrayList;
import java.util.List;

public class TextTwist {
    private WordValidator w;
    public ArrayList<String> result;

    public TextTwist(int length, ArrayList<String> letters) {
        this.w = new WordValidator(length);
        this.result = new ArrayList<>();
    }

    public Boolean boundFunction(String word) {...

    public void recursiveTextTwistBT(ArrayList<String> let, String word, int depth) {...

    public void recursiveTextTwistDLS(ArrayList<String> let, String word, int depth) {...

}
```

Gambar C.2, desain kelas TextTwist.

Penjelasan komponen kelas TextTwist:

- WordValidator w, atribut ini berfungsi sebagai patokan kamus untuk tiap instansiasi yang akan spesifik dengan panjang kata.
- Result, berfungsi untuk menampung semua hasil dari pencarian.
- boundFunction, metode ini adalah metode yang berisi implementasi untuk pengecekan bound (sudah dijelaskan di *pseudocode*).
- recursiveTextTwistBT, algoritma penyelesaian Text Twist dengan *backtrack* (sudah dijelaskan di *pseudocode*).
- recursiveTextTwistDLS, algoritma penyelesaian Text Twist dengan DLS (sudah dijelaskan di *pseudocode*).

c. Main

Kelas ini berisi handling DLS secara iteratif (IDS) dan juga menyediakan tampilan CLI agar lebih rapih.

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        ArrayList<String> letters = new ArrayList<>();

        // Input
        Scanner input = new Scanner(System.in);
        System.out.println(x:"=====Text Twist Solver=====");
        System.out.println(x:"");
        System.out.print(s:"Masukkan huruf-huruf: ");
        String letters_input = input.nextLine();
        String[] inputs = letters_input.split(regex:"[ ,]+");
        for (int i = 0; i < inputs.length; i++) {
            letters.add(inputs[i]);
        }
        System.out.println(x:"Pilih Algoritma : ");
        System.out.println(x:"1. DFS");
        System.out.println(x:"2. Backtrack (Optimalisasi DFS)");
        System.out.print(s:"Masukkan pilihan algoritma: ");
        Scanner algo_input = new Scanner(System.in);
        String algo = algo_input.nextLine();
        System.out.println(x:"");
        System.out.println("===Menggunakan algoritma " + algo + "===");

        // Main handling
        for (int i = 3; i <= inputs.length; i++) {
            System.out.println("Kata-kata dengan length " + i + ":");
            TextTwist t = new TextTwist(i, letters);
            final long startTime = System.currentTimeMillis();
            if (algo.equals(anObject:"1")) {
                t.recursiveTextTwistDLS(letters, word:"", i);
            } else {
                t.recursiveTextTwistBT(letters, word:"", i);
            }
            final long endTime = System.currentTimeMillis();
            System.out.println(t.result);
            System.out.println("took : " + (endTime - startTime) + " ms");
            System.out.println(x:"");
        }
    }
}
```

Gambar C.3, desain kelas main.

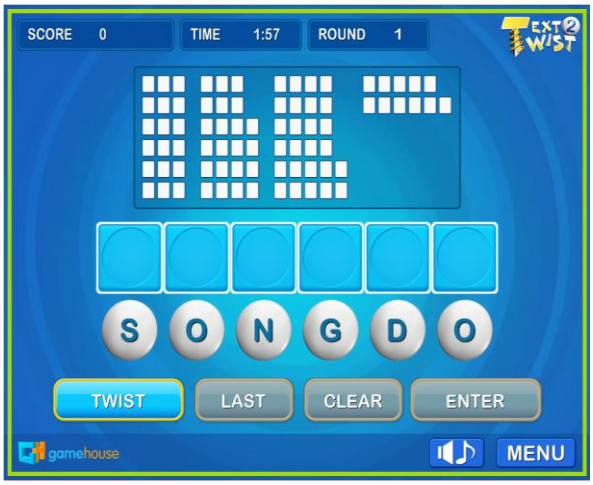
Penanganan IDS terdapat pada bagian yang dikomentari dengan "// Main handling".

IV. Percobaan dan Analisis

A. *Percobaan*

Percobaan akan dilakukan sebanyak 3 round.

a. Percobaan 1 (round 1)



Gambar A.1, percobaan 1.

```
PS C:\Users\ASUS\Documents\Sem 4\Stima\Implementation\makalah\TextTwistSolver> java Main
=====Text Twist Solver=====

Masukkan huruf-huruf: s, o, n, g, d, o
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 1.

===Menggunakan algoritma 1.===
Kata-kata dengan length 3:
[son, sod, ons, ods, nos, nog, nod, noo, god, goo, dos, don, dog]
took : 7 ms

Kata-kata dengan length 4:
[song, soon, snog, nogs, nods, gods, goos, goon, good, dons, dong, dogs]
took : 9 ms

Kata-kata dengan length 5:
[snood, goons, goods, dongs]
took : 10 ms

Kata-kata dengan length 6:
[godson]
took : 16 ms
```

Gambar A.2, hasil 1 IDS.

```
PS C:\Users\ASUS\Documents\Sem 4\Stima\Implementation\makalah\TextTwistSolver> java Main
=====Text Twist Solver=====

Masukkan huruf-huruf: s, o, n, g, d, o
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 2

===Menggunakan algoritma 2===
Kata-kata dengan length 3:
[son, sod, ons, ods, nos, nog, nod, noo, god, goo, dos, don, dog]
took : 8 ms

Kata-kata dengan length 4:
[song, soon, snog, nogs, nods, gods, goos, goon, good, dons, dong, dogs]
took : 9 ms

Kata-kata dengan length 5:
[snood, goons, goods, dongs]
took : 12 ms

Kata-kata dengan length 6:
[godson]
took : 17 ms
```

Gambar A.3, hasil 1 *backtrack*.

Gambar A.4, gambar hasil permainan Text Twist round 1.

b. Percobaan 2 (round 2)

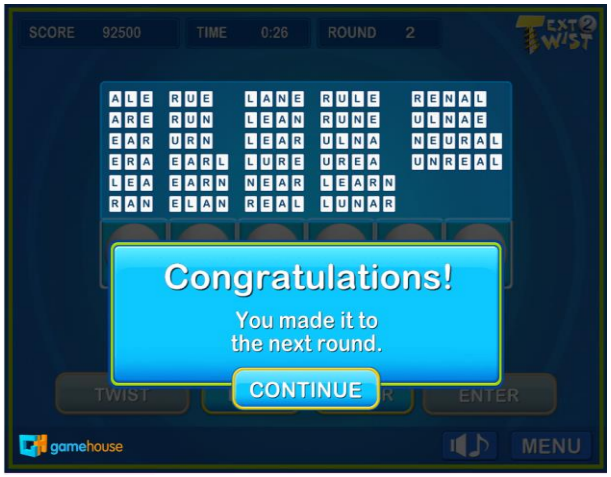
Gambar A.5, gambar percobaan 2.

```
Masukkan huruf-huruf: u, l, r, a, n, e
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 1

===Menggunakan algoritma 1===
Kata-kata dengan length 3:
[urn, lar, leu, lea, run, rue, ran, ale, are, ane, nae, era, ern, eau, ear]
took : 3 ms

Kata-kata dengan length 4:
[ulan, ulna, urea, lure, luna, lune, lane, lear, lean, rule, rune, rale, real, nurl, near, elan, earl, earn]
took : 8 ms

Kata-kata dengan length 5:
[ulnar, ulnae, ureal, lunar, learn, renal]
took : 24 ms

Kata-kata dengan length 6:
[unreal, neural]
took : 41 ms
```

Gambar A.6, hasil 2 IDS.

```
PS C:\Users\ASUS\Documents\Sem 4\Stima\Implementation\makalah\TextTwistSolver> java Main
=====Text Twist Solver=====

Masukkan huruf-huruf: u, l, r, a, n, e
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 2

===Menggunakan algoritma 2===
Kata-kata dengan length 3:
[urn, lar, leu, lea, run, rue, ran, ale, are, ane, nae, era, ern, eau, ear]
took : 8 ms

Kata-kata dengan length 4:
[ulan, ulna, urea, lure, luna, lune, lane, lear, lean, rule, rune, rale, real, nurl, near, elan, earl, earn]
took : 11 ms

Kata-kata dengan length 5:
[ulnar, ulnae, ureal, lunar, learn, renal]
took : 14 ms

Kata-kata dengan length 6:
[unreal, neural]
took : 20 ms
```

Gambar A.7, hasil 2 *backtrack*.



Gambar A.8, gambar hasil permainan Text Twist round 2.

c. Percobaan 3 (round 3)

Gambar A.9, gambar percobaan 3.

```
PS C:\Users\ASUS\Documents\Sem 4\Stima\Implementation\makalah\TextTwistSolver> java Main
=====Text Twist Solver=====

Masukkan huruf-huruf: n, k, e, d, i, f
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 1

===Menggunakan algoritma 1===
Kata-kata dengan length 3:
[ken, kef, kin, kid, kif, end, den, din, die, ink, fen, fed, fin, fie, fid]
took : 3 ms

Kata-kata dengan length 4:
[neif, nide, kine, kind, kief, defi, dink, dine, dike, fend, fink, fine, find]
took : 10 ms

Kata-kata dengan length 5:
[knife, inked, fined, fiend]
took : 26 ms

Kata-kata dengan length 6:
[knifed, finked]
took : 40 ms
```

Gambar A.10, hasil 3 IDS.

```
Masukkan huruf-huruf: n, k, e, d, i, f
Pilih Algoritma :
1. DFS
2. Backtrack (Optimalisasi DFS)
Masukkan pilihan algoritma: 2

===Menggunakan algoritma 2===
Kata-kata dengan length 3:
[ken, kef, kin, kid, kif, end, den, din, die, ink, fen, fed, fin, fie, fid]
took : 8 ms

Kata-kata dengan length 4:
[neif, nide, kine, kind, kief, defi, dink, dine, dike, fend, fink, fine, find]
took : 8 ms

Kata-kata dengan length 5:
[knife, inked, fined, fiend]
took : 6 ms

Kata-kata dengan length 6:
[knifed, finked]
took : 12 ms
```

Gambar A.11, hasil 3 *backtrack*.

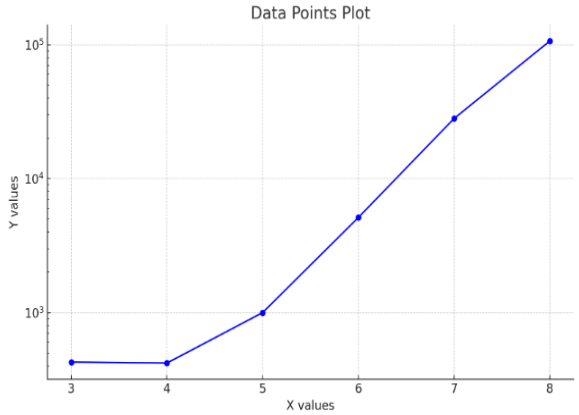
Gambar A.12, hasil permainan Text Twist round 3.

B. Analisis

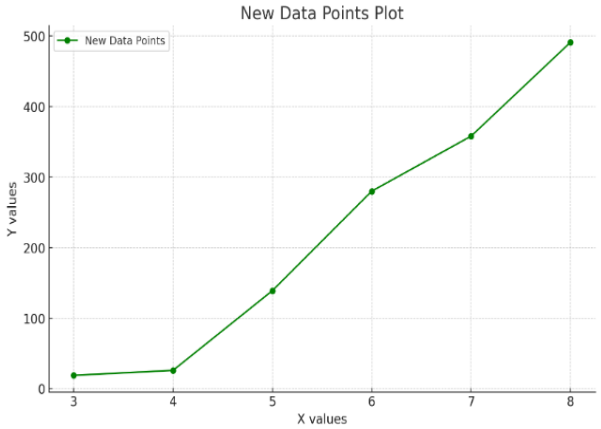
Pada konteks permainan ini, perbedaan waktu eksekusi tidak begitu signifikan dan berpengaruh karena huruf yang diinput relative kecil (hanya 6 huruf), maka untuk kepentingan analisis akan dilakukan percobaan dengan menginput 10 huruf. Berikut adalah data waktu dari percobaan kasus 10 huruf = {n, k, e, d, i, f, a, b, c, e}:

Hasil program (tidak cukup disini): https://drive.google.com/file/d/1LaRdzHVGVCofMN0XGinlIt8rqWDW7uB0/view?usp=sharing (IDS), https://drive.google.com/file/d/1tYgTrg3iQG_uJzYUbATEKf4owKhxzmdH/view?usp=sharing (*Backtrack*)

Panjang huruf	IDS (ms)	*Backtracking* (ms)
3	427	19
4	420	26
5	1000	139
6	5127	280
7	28109	358
8	106452	491
9	357 (tidak ada kata sepanjang 9 di kamus)	0
10	356 (tidak ada kata sepanjang 10 di kamus)	0



Gambar A.12, grafik IDS (x = panjang huruf, y = waktu eksekusi).



Gambar A.13, grafik *Backtrack* (x = panjang huruf, y = waktu eksekusi).

Jika kita lihat pada tabel hasil, *backtracking* jauh lebih unggul dibandingkan dengan IDS. Pada grafik yang disediakan, IDS memiliki pertumbuhan yang memiliki faktor pertumbuhan sangat besar. Hal ini dikarenakan IDS menelusuri semua kemungkinan susunan tanpa melakukan *restriction* apapun, sehingga pertumbuhan ruangnya sangat besar. Dengan mengimplementasikan *backtracking*, factor pertumbuhan kompleksitas yang besar tersebut dipotong dan menghasilkan pertumbuhan kompleksitas algoritma yang jauh lebih kecil dibandingkan IDS.

V. KESIMPULAN

Pada makalah ini, permasalahan pencarian solusi teka-teki kata Text Twist diimplementasikan dengan pendekatan algoritma IDS dan *Backtracking*. Pada konteks permainan ini, perbedaan IDS dan *Backtracking* tidak begitu berpengaruh karena permasalahan hanya terbatas pada menyelesaikan penyusunan kata dari 6 huruf saja. Akan tetapi, apabila kita melakukan peningkatan terhadap input jumlah huruf (10 huruf seperti pada analisis) maka perbedaan IDS dan *Backtracking* sangat signifikan. *Backtracking* memiliki kompleksitas yang jauh lebih kecil dan lebih mangkus dibandingkan IDS. Hal ini disebabkan karena IDS akan menelusuri semua kemungkinan susunan kata tanpa melakukan *restriksi* seperti *backtracking*. Oleh karena itu, pertumbuhan waktu eksekusi dan penggunaan ruang IDS sangat besar. Maka dapat ditarik kesimpulan bahwa pada kasus penyelesaian permainan teka-teki kata Text Twist, algoritma *backtracking* sangat ampuh dalam mengoptimalisasi algoritma IDS hanya dengan perubahan beberapa *line* saja pada kode (penambahan fungsi *bound*).

DAFTAR PUSTAKA

[1] Kamus Oracle, https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt

[2] Text Twist Online, https://www.mindgames.com/game/TextTwist+2

[3] Munir, R. 2022, Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1) , https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf

[4] Munir, R. 2022, Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2) , https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

[5] Munir, R. 2022, Algoritma runut-balik (backtracking) (Bagian 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf

[6] Munir, R. 2022, Algoritma runut-balik (backtracking) (Bagian 2), https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf

LAMPIRAN

Github untuk *source code*:
https://github.com/Akmal2205/TextTwistSolver

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Juni 2024

Muhammad Syarafi Akmal 13522076