**TUGAS KECIL**

**Implementasi Algoritma Brute Force dalam Penyelesaian Word Search Puzzle**

**LAPORAN**

**Diajukan sebagai salah satu tugas mata kuliah IF2211 Strategi Algoritma pada**

**Semester II**

**Tahun Akademik 2021-2022**

**oleh**

**Owen Christian Wijaya 13520124**

**A picture containing text

Description automatically generated**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

DAFTAR ISI

[BAB I. ALGORITMA *BRUTE FORCE* 2](#_Toc93931769)

[BAB II. *SOURCE PROGRAM* DALAM BAHASA JAVA 4](#_Toc93931770)

[2.1 CharObj,java 4](#_Toc93931771)

[2.2 Exec.java 4](#_Toc93931772)

[2.3 MainProgram.java 4](#_Toc93931773)

[2.4 Matcher.java 6](#_Toc93931774)

[2.5 Matrix.java 12](#_Toc93931775)

[2.6 SearchWord.java 12](#_Toc93931776)

[2.7 ReadFile.java 16](#_Toc93931777)

[BAB III. PENGUJIAN (INPUT/OUTPUT) 17](#_Toc93931778)

[3.1 Pengujian terhadap file small1.txt (ukuran 14 x 12) 17](#_Toc93931779)

[3.2 Pengujian terhadap file small2.txt (ukuran 16 x 14) 18](#_Toc93931780)

[3.3 Pengujian terhadap file small3.txt (ukuran 14 x 12) 19](#_Toc93931781)

[3.4 Pengujian terhadap file medium1.txt (ukuran 20 x 18) 20](#_Toc93931782)

[3.5 Pengujian terhadap file medium2.txt (ukuran 22 x 20) 21](#_Toc93931783)

[3.6 Pengujian terhadap file medium3.txt (ukuran 24 x 22) 22](#_Toc93931784)

[3.7 Pengujian terhadap file large1.txt (ukuran 32 x 30) 23](#_Toc93931785)

[3.8 Pengujian terhadap file large2.txt (ukuran 34 x 32) 24](#_Toc93931786)

[3.9 Pengujian terhadap file large3.txt (ukuran 36 x 34) 25](#_Toc93931787)

# BAB I. ALGORITMA *BRUTE FORCE*

Dalam tugas kecil ini, algoritma *brute force* diimplementasikan dalam pencarian kata-kata yang ada dalam permainan *word search puzzle*. *Puzzle* disimpan dalam bentuk matriks karakter, sementara kata kuncidisimpan ke dalam sebuah *list*. Algoritma *brute force* akan mencari keberadaan sebuah kata dalam *puzzle* dengan pencocokan karakter pertama dari kata kunci yang hendak dicari dengan karakter yang ada di dalam *puzzle* secara sekuensial. Apabila ditemukan karakter yang sesuai dengan karakter pertama kata kunci, maka pencarian dilanjutkan dengan mencocokkan karakter-karakter lain dengan karakter di kata kunci.

Pencocokan dilakukan terhadap delapan arah: ke arah kanan, ke arah kiri, ke arah atas, ke arah bawah, ke arah diagonal atas kiri, ke arah diagonal atas kanan, ke arah diagonal bawah kiri, dan ke arah diagonal bawah kanan. Sebelum melakukan pencocokan, program akan melakukan pengecekan apakah panjang kata kunci melebihi dimensi matriks *puzzle*.

Apabila panjang kata kunci yang hendak dicek sesuai arah melebihi dimensi matriks *puzzle* (akan menyebabkan *out of bounds*), maka pengecekan ke arah tersebut tidak akan dilakukan dan program akan melakukan pengecekan ke arah lainnya. Sebaliknya, apabila panjang kata kunci tidak menyebabkan *out of bounds*, maka pengecekan akan dilakukan. Pengecekan akan dihentikan apabila ditemukan satu karakter di arah tersebut yang tidak cocok. Jika pengecekan yang dilakukan berhasil, maka pencarian akan dihentikan dan program akan memberikan keluaran berupa arah yang ditemukan, jumlah perbandingan yang dilakukan, dan waktu yang dibutuhkan untuk melakukan perbandingan. Setelah semua kata kunci dicocokkan, program akan memberikan keluaran berupa matriks awal dengan kata kunci yang di-*highlight* untuk menandai posisi kata kunci di dalam *puzzle*.

Algoritma pencocokan dilakukan dengan mencocokkan karakter pertama di kata kunci dengan tiap karakter di matriks secara sekuensial. Apabila ditemukan karakter di matriks yang sama dengan karakter pertama kata kunci yang hendak dicari, fungsi-fungsi tersebut akan dipanggil. Fungsi menerima argumen berupa matriks *puzzle*, kata kunciyang telah dipisah dalam bentuk *array*, dan koordinat dimana karakter pertama ditemukan. Setiap fungsi akan melakukan pengecekan sesuai arah. Dengan menggunakan skema pencarian menggunakan *boolean*, fungsi akan melakukan pencocokan per karakter sampai *out of bounds*. Apabila ada satu karakter yang tidak cocok, pencarian akan berhenti dan fungsi akan mengembalikan Exec dengan flag bernilai false. Sebaliknya, apabil pencocokan berhasil secara keseluruhan, maka program akan melakukan perubahan terhadap elemen matriks yang dicek. Nilai variabel colorChar dalam elemen charObj dalam matriks akan diubah menjadi karakter berwarna sesuai dengan nilai koordinat dan panjang kata yang dicek menggunakan kode ANSI. Pada akhir program, matriks yang akan ditampilkan adalah matriks dengan variabel colorChar yang telah berwarna.

*Compiler Java* bersifat *just-in-time,* dan karena itu, pemanggilan instansi pertama akan mengambil waktu lebih lama. Oleh karena itu, untuk membuat hasil pengukuran yang lebih akurat, program akan memanggil instansi search terlebih dahulu sebelum melakukan iterasi. Pemanggilan dilakukan dengan mengecek kata pertama di daftar kata kunci dengan indeks -1. Indeks -1 menandakan bahwa *runtime* hanya untuk melakukan pemanggilan instansi, sehingga pada saat melakukan iterasi, waktu yang diperoleh lebih akurat dan tidak menghitung waktu pemanggilan instansi.

# BAB II. *SOURCE PROGRAM* DALAM BAHASA JAVA

## 2.1 CharObj,java

public class CharObj{

    String oriChar;

    String colorChar;

    public CharObj(String character){

        this.oriChar = character;

        this.colorChar = character;

    }

}

## 2.2 Exec.java

public class Exec {

    int count;

    boolean flag;

    public Exec(int compCount, boolean flag, String[] keyword, int i, int j){

        this.count = compCount;

        this.flag = flag;

    }

}

## 2.3 MainProgram.java

import java.util.Scanner;

import java.util.ArrayList;

public class MainProgram {

    public static void main(String[] args){

        long duration = 0;

        long count = 0;

        long[] results;

        Matrix m = new Matrix(100, 100);

        ArrayList<String> keywords = new ArrayList<String>();

        Scanner sc = new Scanner(System.in);

        System.out.println(" \_\_\_\_  \_     \_\_\_\_  \_\_\_\_  \_\_\_\_\_ \_\_\_\_  \_\_\_\_  \_\_\_\_  \_    ");

        System.out.println("/  \_\_\\/ \\ /\\/\_   \\/\_   \\/  \_\_//  \_ \\/  \_\_\\/   \_\\/ \\ /|");

        System.out.println("|  \\/|| | || /   / /   /|  \\  | / \\||  \\/||  /  | |\_||");

        System.out.println("|  \_\_/| \\\_/|/   /\_/   /\_|  /\_ | |-|||    /|  \\\_ | | ||");

        System.out.println("\\\_/   \\\_\_\_\_/\\\_\_\_\_/\\\_\_\_\_/\\\_\_\_\_\\\\\_/ \\|\\\_/\\\_\\\\\_\_\_\_/\\\_/ \\|");

        System.out.println();

        System.out.println("Welcome to the Puzzearch solver!");

        ReadFile.readText(m, keywords, sc);

        System.out.println("Read puzzle from the file: (Size: " + m.rows + " x " + m.cols + ")");

        m.printOriMatrix();

        System.out.println("\nThere are " + keywords.size() +" read keywords from the file: ");

        for(int i = 0; i < keywords.size(); i++){

            System.out.println("- " + keywords.get(i));

        }

        System.out.println("Press any key to start searching!");

        sc.nextLine();

        SearchWord.search(m, keywords.get(0), -1);

        for(int i = 0; i < keywords.size(); i++){

            results = SearchWord.search(m, keywords.get(i), i);

            duration += results[0];

            count += results[1];

        }

        m.printColorMatrix();

        System.out.println();

        System.out.println("Comparison time in total (parsing excluded): " );

        System.out.println(duration + " ns (" + String.format("%.3f", (double)(duration / 10e5)) + " ms)");

        System.out.println("Comparison count in total: " + count + " time(s)");

        System.out.println("Press any key to quit...");

        sc.nextLine();

    }

}

## 2.4 Matcher.java

public class Matcher {

    public static Exec checkHL(Matrix m, String[] keyword, int i, int j){ //horizontal left

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + 1 - keyword.length < 0){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i][j - count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i][j - a].colorChar =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i][j - a].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkHR(Matrix m, String[] keyword, int i, int j){ //horizontal right

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + keyword.length > m.cols){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;                if (!m.buffer[i][j + count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i][j + a].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i][j + a].colorChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkVU(Matrix m, String[] keyword, int i, int j){ //vertical upper

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (i - keyword.length + 1 < 0){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i - count][j].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i - a][j].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16\* i)  + (16\*j) + 19) % 185) + 20) + "m" + m.buffer[i - a][j].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkVL(Matrix m, String[] keyword, int i, int j){

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (i + keyword.length > m.rows){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i + count][j].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i + a][j].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i + a][j].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkDLU(Matrix m, String[] keyword, int i, int j){ //diagonal left upper

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + 1 - keyword.length < 0 || i - keyword.length + 1 < 0){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i - count][j - count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i - a][j - a].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i - a][j - a].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkDLL(Matrix m, String[] keyword, int i, int j){ //diagonal left lower

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + 1 - keyword.length < 0 || i + keyword.length > m.rows){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i + count][j - count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i + a][j - a].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i + a][j - a].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkDRU(Matrix m, String[] keyword, int i, int j){ //diagonal right upper

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + keyword.length > m.cols || i - keyword.length + 1 < 0){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i - count][j + count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i - a][j + a].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i - a][j + a].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

    public static Exec checkDRL(Matrix m, String[] keyword, int i, int j){ //diagonal right lower

        boolean flag = true;

        int compCount = 0;

        Exec tempExec;

        if (j + keyword.length > m.cols || i + keyword.length > m.rows){

            flag = false;

        } else {

            int count = 1;

            while (count < keyword.length && flag){

                compCount++;

                if (!m.buffer[i + count][j + count].oriChar.equals(keyword[count])){

                    flag = false;

                } else {

                    count++;

                }

            }

        }

        if(flag){

            int a = 0;

            while (a < keyword.length){

                m.buffer[i + a][j + a].colorChar  =  "\u001B[1m\u001B[38;5;" + (((keyword.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m" + m.buffer[i + a][j + a].oriChar + "\u001B[0m";

                a++;

            }

        }

        tempExec = new Exec(compCount, flag);

        return tempExec;

    }

}

## 2.5 Matrix.java

public class Matrix {

    CharObj buffer[][];

    int rows;

    int cols;

    public Matrix(int row, int col){

        this.rows = row;

        this.cols = col;

        this.buffer = new CharObj[row][col];

        for(int i = 0; i < this.rows; i++){

            for(int j = 0; j < this.cols; j++){

                this.buffer[i][j] = new CharObj(" ");

            }

        }

    }

    public void printOriMatrix(){

        for(int i = 0; i < this.rows; i++){

            for(int j = 0; j < this.cols; j++){

                System.out.print(this.buffer[i][j].oriChar + " ");

            }

            System.out.println();

        }

    }

    public void printColorMatrix(){

        for(int i = 0; i < this.rows; i++){

            for(int j = 0; j < this.cols; j++){

                System.out.print(this.buffer[i][j].colorChar + " ");

            }

            System.out.println();

        }

    }

}

## 2.6 SearchWord.java

public class SearchWord {

    public static long[] search(Matrix m, String keyword, int index){

        String[] keyArr = keyword.split("");

        String first = keyArr[0];

        boolean found = false;

        int i = 0;

        int j = 0;

        int count = 0;

        long time = 0;

        long tempStart = System.nanoTime();

        while (i < m.rows && !found){

            while (j < m.cols && !found){

                count++;

                if (first.equals(m.buffer[i][j].oriChar)){

                    Exec temp = Matcher.checkHR(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found horizontally right!");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkDRL(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                            System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found diagonally! (right lower)");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkVL(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found vertically lower!");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkDLL(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found diagonally! (left lower)");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkHL(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found horizontally left!");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkDLU(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found diagonally! (left upper)");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkVU(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1 ){

                            time = System.nanoTime() - tempStart;

                              System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found vertically upper!");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                    temp = Matcher.checkDRU(m, keyArr, i, j);

                    if (!found){

                        count += temp.count;

                        if (temp.flag && index != -1 && !found){

                            time = System.nanoTime() - tempStart;

                               System.out.println("Keyword \'" + "\u001B[1m\u001B[38;5;" + (((keyArr.length + (16 \*  i)  + (16 \* j) + 19) % 185) + 20) + "m"  + keyword + "\u001B[0m\' found diagonally! (right upper)");

                            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

                            found = true;

                        }

                    }

                }

                j = j + 1;

            }

            j = 0;

            i = i + 1;

        }

        if (!found && index != -1){

            time = System.nanoTime() - tempStart;

            System.out.println("Keyword \'" + keyword + "\' not found...");

            System.out.println("Comparison: " + count + " time(s) | " + time + " ns (" + String.format("%.3f", (double)(time / 10e5)) + " ms)");

        }

        System.out.println();

        long[] arr = {time, Long.valueOf(count)};

        return arr;

    }

}

## 2.7 ReadFile.java

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.util.ArrayList;

public class ReadFile{

    public static void readText(Matrix m, ArrayList<String> keywords, Scanner sc){

        String filename = "";

        System.out.print("Input your filename (without .txt): ");

        filename = sc.nextLine();

        while (filename == "" || !(new File("../test/" + filename + ".txt").exists())){

            System.out.println("Sorry, file name doesn't exist or it has not been put in the /test folder!");

            System.out.print("Input your filename (without .txt): ");

            filename = sc.nextLine();

        }

        try{

            File text = new File("../test/" + filename + ".txt");

            Scanner sizeReader = new Scanner(text);

            int rowSize = 1;

            while (sizeReader.hasNextLine() && !sizeReader.nextLine().equals("")){

                rowSize++;

            }

            sizeReader.close();

            m.rows = rowSize - 1;

            Scanner lineReader = new Scanner(text);

            m.cols = 0;

            try{

                for(int i = 0; i < m.rows; i++){

                    String line = lineReader.nextLine();

                    String rows[] = line.split(" ");

                    if (m.cols < rows.length){

                        m.cols = rows.length;

                    }

                    for(int j = 0; j < rows.length; j++){

                        m.buffer[i][j] = new CharObj(rows[j]);

                    }

                    if (rows.length < m.cols){

                        for(int j = rows.length + 1; j  < m.cols; j++){

                            m.buffer[i][j] = new CharObj(" ");

                        }

                    }

                }

                lineReader.nextLine();

                while(lineReader.hasNextLine()){

                    String kw = lineReader.nextLine();

                    keywords.add(kw);

                }

            } finally {

                lineReader.close();

            }

        } catch (FileNotFoundException e){

            System.out.println("File not found!");

            e.printStackTrace();

        }

    }

}

## BAB III. PENGUJIAN (INPUT/OUTPUT)

## 3.1 Pengujian terhadap file small1.txt (ukuran 14 x 12)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.1.1 Isian berkas small1.txt | Gambar 3.1.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.1.3 Informasi hasil eksekusi per kata | Gambar 3.1.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.2 Pengujian terhadap file small2.txt (ukuran 16 x 14)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.2.1 Isian berkas small2.txt | Gambar 3.2.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.2.3 Informasi hasil eksekusi per kata | Gambar 3.2.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.3 Pengujian terhadap file small3.txt (ukuran 14 x 12)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.3.1 Isian berkas small3.txt | Gambar 3.3.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.3.3 Informasi hasil eksekusi per kata | Gambar 3.3.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.4 Pengujian terhadap file medium1.txt (ukuran 20 x 18)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.4.1 Isian berkas medium1.txt | Gambar 3.4.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.4.3 Informasi hasil eksekusi per kata | Gambar 3.4.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.5 Pengujian terhadap file medium2.txt (ukuran 22 x 20)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.5.1 Isian berkas medium1.txt | Gambar 3.5.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.5.3 Informasi hasil eksekusi per kata | Gambar 3.5.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.6 Pengujian terhadap file medium3.txt (ukuran 24 x 22)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.6.1 Isian berkas medium3.txt | Gambar 3.6.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.6.3 Informasi hasil eksekusi per kata | Gambar 3.6.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.7 Pengujian terhadap file large1.txt (ukuran 32 x 30)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.7.1 Isian berkas large1.txt | Gambar 3.7.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.7.3 Informasi hasil eksekusi per kata | Gambar 3.7.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.8 Pengujian terhadap file large2.txt (ukuran 34 x 32)

|  |  |
| --- | --- |
| Background pattern  Description automatically generated |  |
| Gambar 3.8.1 Isian berkas large2.txt | Gambar 3.8.2 Hasil pembacaan file masukan |
|  |  |
| Gambar 3.8.3 Informasi hasil eksekusi per kata | Gambar 3.8.4 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

## 3.9 Pengujian terhadap file large3.txt (ukuran 36 x 34)

|  |  |
| --- | --- |
|  |  |
| Gambar 3.9.1 Isian berkas large3.txt (Puzzle) | Gambar 3.9.2 Isian berkas large3.txt (Kata kunci) |
|  |  |
| Gambar 3.9.3 Hasil pembacaan file masukan (Puzzle) | Gambar 3.9.4 Hasil pembacaan file masukan (Kata kunci) |
| S |  |
| Gambar 3.9.5 Informasi hasil eksekusi per kata (1) | Gambar 3.9.6 Informasi hasil eksekusi per kata (2) |
|  |  |
| Gambar 3.9.7 Informasi hasil eksekusi per kata (3) | Gambar 3.9.8 Matriks akhir dan informasi waktu eksekusi & banyak perbandingan |

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi tanpa kesalahan (no syntax error) | √ |  |
| 2. Program berhasil running | √ |  |
| 3. Program dapat membaca file masukan dan menuliskan luaran | √ |  |
| 4. Program berhasil menemukan semua kata di dalam puzzle | √ |  |