

LAPORAN TUGAS KECIL I
IF2211 STRATEGI ALGORITMA



Laporan ini dibuat untuk memenuhi tugas
Mata Kuliah IF 2211 Strategi Algoritma

Disusun Oleh :

Vincent Ho (13520093)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER I TAHUN 2021/2022

BAB I

ALGORITMA BRUTE FORCE

Algoritma Brute Force yang saya terapkan dalam menyelesaikan Word Search Puzzle memiliki langkah – langkah sebagai berikut. Algoritma akan melakukan pencarian kata mulai dari kata kunci yang paling pertama. Saat telah ditemukan kata kunci tersebut pada Word Search Puzzle, maka program akan menampilkan solusinya dan melanjutkan pencarian kata kunci kedua dan seterusnya hingga seluruh kata kunci ditemukan. Pencarian kata kunci dimulai dari ujung kiri atas (koordinat matriks[0][0]) dan memiliki alur pergeseran mengecek kolom dari kiri ke kanan dan baris dari atas ke bawah. Apabila kata kunci berhasil ditemukan maka pencarian kata kunci berikutnya akan dimulai lagi dari ujung kiri atas. Hal ini dicapai dengan menggunakan nested loop untuk mengiterasi setiap baris dan kolom dari puzzle yang diinput.

Algoritma brute force yang saya buat memiliki beberapa optimalisasi, yaitu: program akan mengecek terlebih dahulu huruf pertama dari puzzle dan membandingkannya dengan huruf pertama dari kata kunci, apabila berbeda maka algoritma pencocokan kata tidak akan dijalankan, dan program akan melanjutkan untuk mengecek huruf puzzle berikutnya.

Apabila ternyata huruf pertama dari puzzle dan kata kunci sama, maka algoritma pencocokan kata akan dijalankan. Pencocokan kata dilakukan dengan urutan sebagai berikut: arah kanan, arah kiri, arah bawah, arah atas, diagonal ke arah kanan bawah, diagonal ke arah kiri bawah, diagonal ke arah kanan atas, dan diagonal ke arah kiri atas. Apabila dalam proses pencocokan kata ditemukan kata kunci, maka proses tersebut akan berhenti, solusi akan ditampilkan dan kemudian program akan lanjut mencari kata kunci berikutnya. Misalnya saat sedang mencari kata kunci pertama dan pada proses pencocokan kata ke arah kiri kata kunci berhasil ditemukan, maka proses pencocokan kata tidak akan dilanjutkan (tidak mencari lagi ke arah bawah, atas, dst.), algoritma akan lanjut untuk mencari kata kunci berikutnya.

Proses pencocokan kata juga tidak akan dijalankan apabila kata sudah tidak mungkin ditemukan di arah tersebut. Misalnya saat mencari kata kunci dengan panjang 7 karakter, maka apabila jumlah kolom puzzle adalah 10, saat telah mencapai kolom ke 4, proses pencocokan kata ke arah kanan tidak akan dilakukan lagi. Hal ini juga berlaku untuk segala arah.

BAB II

SOURCE CODE PROGRAM

Program Word Search Puzzle Solver ini dibuat menggunakan bahasa Java.

```
import java.io.*;
import java.util.Scanner;

public class Main {

    static int countSearch = 0;
    static int countFound = 0;

    public static void searchBruteForce(char[][] wordPuzzle, String[]
puzzleKey) {

        int row = wordPuzzle.length;
        int col = wordPuzzle[0].length;
        int numberOfPuzzleKey = puzzleKey.length;

        int countLetters;
        int keyLength;
        String key;

        for (int i = 0; i < numberOfPuzzleKey; i++) {
            key = puzzleKey[i];
            keyLength = puzzleKey[i].length();

            boolean found = false;

            for (int currRow = 0; currRow < row; currRow++) {
                for (int currCol = 0; currCol < col; currCol++) {
                    countSearch++;

                    if (wordPuzzle[currRow][currCol] == key.charAt(0)) {

                        // searchRight
                        if (!found) {
                            countLetters = 0;
                            while ((currCol + keyLength) <= col &&
(countLetters < keyLength)) {
                                countSearch++;
                                if (wordPuzzle[currRow][currCol +
countLetters] == key.charAt(countLetters)) {
                                    countLetters++;
                                } else {
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        }
    }
    if (countLetters == keyLength) {
        found = true;
        System.out.println(key);

        char[][] displayPuzzle =
createEmptyPuzzle(row, col);
        for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
            displayPuzzle[currRow][currCol +
lettersIndex] = key.charAt(lettersIndex);
        }
        printPuzzle(displayPuzzle);
    }
}

// searchLeft
if (!found) {
    countLetters = 0;
    while ((currCol >= keyLength - 1) && (countLetters
< keyLength)) {
        countSearch++;
        if (wordPuzzle[currRow][currCol -
countLetters] == key.charAt(countLetters)) {
            countLetters++;
        } else {
            break;
        }
    }
    if (countLetters == keyLength) {
        found = true;
        System.out.println(key);

        char[][] displayPuzzle =
createEmptyPuzzle(row, col);
        for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
            displayPuzzle[currRow][currCol -
lettersIndex] = key.charAt(lettersIndex);
        }
        printPuzzle(displayPuzzle);
    }
}

// searchDown
if (!found) {
    countLetters = 0;

```

```

        while ((currRow + keyLength) <= row &&
(countLetters < keyLength)) {
            countSearch++;
            if (wordPuzzle[currRow +
countLetters][currCol] == key.charAt(countLetters)) {
                countLetters++;
            } else {
                break;
            }
        }
        if (countLetters == keyLength) {
            found = true;
            System.out.println(key);

            char[][] displayPuzzle =
createEmptyPuzzle(row, col);
            for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
                displayPuzzle[currRow +
lettersIndex][currCol] = key.charAt(lettersIndex);
            }
            printPuzzle(displayPuzzle);
        }

        // searchUp
        if (!found) {
            countLetters = 0;
            while ((currRow >= keyLength - 1) && (countLetters
< keyLength)) {
                countSearch++;
                if (wordPuzzle[currRow -
countLetters][currCol] == key.charAt(countLetters)) {
                    countLetters++;
                } else {
                    break;
                }
            }
            if (countLetters == keyLength) {
                found = true;
                System.out.println(key);

                char[][] displayPuzzle =
createEmptyPuzzle(row, col);
                for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
                    displayPuzzle[currRow -
lettersIndex][currCol] = key.charAt(lettersIndex);

```

```

        }
        printPuzzle(displayPuzzle);
    }
}

// searchDownRight
if (!found) {
    countLetters = 0;
    while ((currRow + keyLength <= row) && (currCol +
keyLength <= col)

        && (countLetters < keyLength)) {
        countSearch++;
        if (wordPuzzle[currRow + countLetters][currCol
+ countLetters] == key

            .charAt(countLetters)) {
                countLetters++;
            } else {
                break;
            }
        }
        if (countLetters == keyLength) {
            found = true;
            System.out.println(key);

            char[][] displayPuzzle =
createEmptyPuzzle(row, col);
            for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
                displayPuzzle[currRow +
lettersIndex][currCol + lettersIndex] = key
                    .charAt(lettersIndex);
            }
            printPuzzle(displayPuzzle);
        }
    }

// searchDownLeft
if (!found) {
    countLetters = 0;
    while ((currRow + keyLength <= row) && (currCol >=
keyLength - 1)

        && (countLetters < keyLength)) {
        countSearch++;
        if (wordPuzzle[currRow + countLetters][currCol
- countLetters] == key

            .charAt(countLetters)) {
                countLetters++;

```

```

        } else {
            break;
        }
    }
    if (countLetters == keyLength) {
        found = true;
        System.out.println(key);

        char[][] displayPuzzle =
createEmptyPuzzle(row, col);
        for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
            displayPuzzle[currRow +
lettersIndex][currCol - lettersIndex] = key
                .charAt(lettersIndex);
        }
        printPuzzle(displayPuzzle);
    }
}

// searchUpRight
if (!found) {
    countLetters = 0;
    while ((currRow >= keyLength - 1) && (currCol +
keyLength) <= col
        && (countLetters < keyLength)) {
        countSearch++;
        if (wordPuzzle[currRow - countLetters][currCol
+ countLetters] == key
            .charAt(countLetters)) {
            countLetters++;
        } else {
            break;
        }
    }
    if (countLetters == keyLength) {
        found = true;
        System.out.println(key);

        char[][] displayPuzzle =
createEmptyPuzzle(row, col);
        for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
            displayPuzzle[currRow -
lettersIndex][currCol + lettersIndex] = key
                .charAt(lettersIndex);
        }
        printPuzzle(displayPuzzle);
    }
}

```

```

    }
}

// searchUpLeft
if (!found) {
    countLetters = 0;
    while ((currRow >= keyLength - 1) && (currCol >=
keyLength - 1)
        && (countLetters < keyLength)) {
        countSearch++;
        if (wordPuzzle[currRow - countLetters][currCol
- countLetters] == key
            .charAt(countLetters)) {
            countLetters++;
        } else {
            break;
        }
    }
    if (countLetters == keyLength) {
        found = true;
        System.out.println(key);

        char[][] displayPuzzle =
createEmptyPuzzle(row, col);
        for (int lettersIndex = 0; lettersIndex <
keyLength; lettersIndex++) {
            displayPuzzle[currRow -
lettersIndex][currCol - lettersIndex] = key
                .charAt(lettersIndex);
        }
        printPuzzle(displayPuzzle);
    }
}
if (found) {
    break;
}
}
}
if (found) {
    break;
}
}
}

public static int[] getData(String filename) {
    try {

```



```

Scanner s = new Scanner(new File(filename));
String puzzleRow = s.nextLine().replaceAll("\\s", "");

int col = puzzleRow.length();

int row = 0;
String nextline = "notEmpty";
while (s.hasNextLine() && nextline != "") {
    row++;
    nextline = s.nextLine();
}

int numberOfPuzzleKey = 0;
while (s.hasNextLine()) {
    numberOfPuzzleKey++;
    s.nextLine();
}

s.close();

int[] data = new int[3];

data[0] = row;
data[1] = col;
data[2] = numberOfPuzzleKey;
return data;

} catch (Exception e) {
    System.out.println("File not found!");
    return null;
}

}

public static char[][] getPuzzleMatrix(String filename, int[] puzzleData)
{
    try {

        Scanner s = new Scanner(new File(filename));

        int row = puzzleData[0];
        int col = puzzleData[1];

        char[][] wordPuzzle = new char[row][col];
        int puzzleRow = 0;

        while (s.hasNextLine()) {
            int puzzleCol = 0;

```

```

        String str = s.nextLine().replaceAll("\\s", "");
        if (puzzleRow < row) {
            while (puzzleCol < col) {
                wordPuzzle[puzzleRow][puzzleCol] =
str.charAt(puzzleCol);
                puzzleCol++;
            }
            puzzleRow++;
        }
    }

    return wordPuzzle;

} catch (Exception e) {
    System.out.println("File not found!");
    return null;
}

}

public static String[] getPuzzleKey(String filename, int[] puzzleData) {
    try {

        Scanner s = new Scanner(new File(filename));

        String[] puzzleKey = new String[puzzleData[2]];

        int wordIndex = 0;

        String nextline = "notEmpty";
        while (s.hasNextLine() && nextline != "") {
            nextline = s.nextLine();
        }

        while (s.hasNextLine()) {
            nextline = s.nextLine().replaceAll("\\s", "");
            if (nextline != " ") {
                puzzleKey[wordIndex] = nextline;
                wordIndex++;
            }
        }
        return puzzleKey;

    } catch (Exception e) {
        System.out.println("File not found!");
        return null;
    }

}

```

```

public static char[][] createEmptyPuzzle(int row, int col) {

    countFound++;

    char[][] emptyPuzzle = new char[row][col];
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            emptyPuzzle[i][j] = '-';
        }
    }
    return emptyPuzzle;
}

public static void printPuzzle(char[][] wordPuzzle) {
    for (int i = 0; i < wordPuzzle.length; i++) {
        for (int j = 0; j < wordPuzzle[0].length; j++) {
            System.out.print(wordPuzzle[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println();
}

public static void puzzleSolution(char[][] wordPuzzle, String[] puzzleKey,
int row, int col) {

    long startTime = System.currentTimeMillis();

    searchBruteForce(wordPuzzle, puzzleKey);

    long endTime = System.currentTimeMillis();

    System.out.println("Execution time: " + (endTime - startTime) + "
ms");
}

public static void main(String[] args) {
    Scanner keyboardInput = new Scanner(System.in);

    System.out.print("Insert File Name: ");
    String filename = keyboardInput.next();
    keyboardInput.close();

    File fileName = new File(filename);

    boolean exists = fileName.exists();

```

```

    if (exists) {

        int[] puzzleData = getData(filename);
        char[][] wordPuzzle = getPuzzleMatrix(filename, puzzleData);
        String[] puzzleKey = getPuzzleKey(filename, puzzleData);

        int row = puzzleData[0];
        int col = puzzleData[1];
        int nPuzzleKey = puzzleData[2];

        System.out.println();
        System.out.println("Puzzle size: " + row + "x" + col);
        System.out.println("Puzzle keywords: " + nPuzzleKey + " words");
        System.out.println();
        System.out.println("=====");
        System.out.println("=          Solution          =");
        System.out.println("=====");
        System.out.println();

        puzzleSolution(wordPuzzle, puzzleKey, row, col);
        System.out.println("Number of comparisons: " + countSearch);
        System.out.println("Words found: " + countFound);

    } else {
        System.out.println(filename + " does not exist!");
    }

}
}

```


small2.txt

```
Insert File Name: small2.txt
```

Puzzle size: 16x14

Puzzle keywords: 10 words

```
=====
=           Solution           =
=====
```

AMBON

```

- - - - - A - - - - -
- - - - - M - - - - -
- - - - - B - - - - -
- - - - - O - - - - -
- - - - - N - - - - -

```

BALI

BALI

FLORES

GAYO

MANDAR

OSING

GNISO

SASAK

A 20x20 grid of dots. The letters are placed at the following intersections (row, column):

- S at (15, 15)
- A at (15, 17)
- S at (15, 19)
- A at (15, 21)
- K at (15, 23)

SUMBAWA

A
W
A
B
M
U
S

TENGGER

TORAJA

```
Execution time: 87 ms
Number of comparisons: 1608
Words found: 10
```

small3.txt

Insert File Name: small3.txt

Puzzle size: 18x16

Puzzle keywords: 11 words

=====

=	Solution	=
---	----------	---

=====

APEL

A
P
E
L

APRIKOT

A
P
R
I
K
O
T

ASEROLA

A S E R O L A

AVOKAD

D A K O V A

BLACKBERRY

B
L
A
C
K
B
E
R
R
Y

BLEWAH

B
L
E
W
A
H

BLUBERI

B L U B E R I

LENGKENG

G
N
E
K
G
N
E
L

PIR

R
I
P

PISANG

G
N
A
S
I
P

RAMBUTAN

N
A
T
U
B
M
A
R

Execution time: 112 ms
Number of comparisons: 2931
Words found: 11

medium1.txt

Insert File Name: medium1.txt

Puzzle size: 20x18

Puzzle keywords: 15 words

```
=====
=           Solution           =
=====
```

AMPUTATION

NO ITATATAUTUPPMAA

ATTORNEY

Y
E
N
R
O
T
T
A

BICYCLING

G
 N
 I
 L
 C
 Y
 C
 I
 B

CARTER

RETRAC

CHERISH

CULTIVATED

CUSP

p
 S
 U
 C

GASOMETER

RETEMOSAG

MAKE

E K A M

MUZZLED

D
E
L
Z
Z
U
M

```
Execution time: 188 ms
Number of comparisons: 3967
Words found: 15
```

medium2.txt

```
Insert File Name: medium2.txt
```

Puzzle size: 22x20

Puzzle keywords: 15 words

```
=====
=           Solution           =
=====
```

BALK

-	-	B	-	-	-	-	-	-	-	-	-
-	-	A	-	-	-	-	-	-	-	-	-
-	-	L	-	-	-	-	-	-	-	-	-
-	-	K	-	-	-	-	-	-	-	-	-

CHAMOIS

A 10x10 grid of dots. The letters C, H, A, M, O, I, S are arranged in a diagonal pattern from bottom-left to top-right. The letter 'C' is at the bottom-left, and 'S' is at the top-right. The letters are placed at the intersections of the grid lines.

CODDLING

A 10x10 grid of dots. The word "CODDLE" is formed by removing dots at the following positions (row, column): (1,5) for 'C', (2,6) for 'O', (3,7) for 'D', (4,8) for 'D', (5,9) for 'L', (6,10) for 'I', (7,10) for 'N', and (8,10) for 'G'.

CRINKLY

- C R I N K L Y -

DIVERSE

E S R E V I D

DRAKE

D R A K E

GROMMET

T

E

M

M

O

R

G

HICKORY

Y

R

O

K

C

I

H

IMMORTAL

LATROMMI

KING SIZED

MISERABLE

ELBARESSIM

MOVABLE

- - - - - E
 - - - - - L
 - - - B - -
 - - A - - -
 - V - - - -
 O - - - - -
 M - - - - -

MUSHROOMED

DEMOORHSUM

PEDIATRIC

P
E
D
I
A
T
R
I
C

QUESTIONS

QUESTIONS

Execution time: 217 ms
Number of comparisons: 4469
Words found: 15

[illegible]

BANKNOTE

B
A
N
K
N
O
T
E

CONDUCTIVE

E
L
E
C
T
R
O
N
I
C

DISORDERLY

D
I
S
O
R
D
E
R
L
Y

DONUT

D
O
N
U
T

GENERATION

G E N E R A T I O N

GEOGRAPHY

G E O G R A P H Y

GUTTERING

G
N
I
R
E
T
T
U
G

KIDS

S
D
I
K

PARADISE

P
A
R
A
D
I
S
E

POSSESSION

N O I S S E S S O P

PROSCRIBE

- E -
- B -
- I -
- R -
- C -
- S -
- O -
- R -
- P -

PROVE

- E V O R P -

RESIDENT

- R E S I D E N T -

Execution time: 256 ms
Number of comparisons: 4451
Words found: 15

large1.txt

```
Insert File Name: large1.txt
```

Puzzle size: 32x30

Puzzle keywords: 16 words

```
=====
=           Solution           =
=====
```

ALLOOF

F
O
O
L
A

AUDIENCE

A U D I E N C E

BLOODLINE

E
 N
 I
 L
 D
 O
 O
 L
 B

CAMP

C
A
M
P

CARER

R E R A C

DECORATING

D
E
C
O
R
A
T
I
N
G

DAMSEL

L
E
S
M
A
D

DEFLECT

T
C
E
L
F
E
D

DESIRED

D
E
S
I
R
E
D

ENORMOUS

S
U
O
M
R
O
N
E

DOOMING

D
O
O
M
I
N
G

EQUESTRIAN

N
A
I
R
T
S
E
U
Q
E

FEEDBACK

- K C A B D E E F -

FUELLED

- F -
- U -
- E -
- L -
- L -
- E -
- D -

FORMERLY

- Y L R E M R O F -

GIGGLING

- G -
- N -
- I -
- L -
- G -
- G -
- I -
- G -

Execution time: 449 ms
Number of comparisons: 12442
Words found: 16

large2.txt

```
Insert File Name: large2.txt
```

Puzzle size: 34x32

Puzzle keywords: 17 words

```
=====
=           Solution           =
=====
```

BANGLADESH

HSE DALGNAB

- ANIHC

BRAZIL

Z

CHINA

C

CONGO

O
G
N
O
C

ETHIOPIA

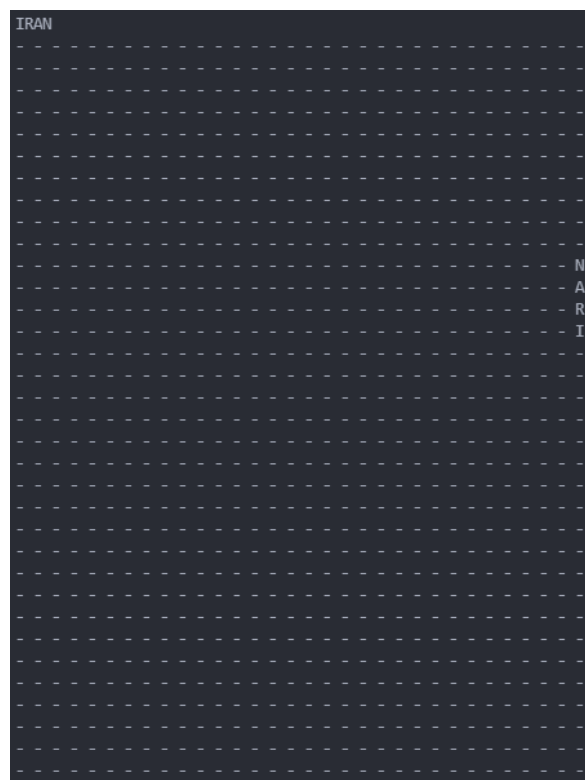
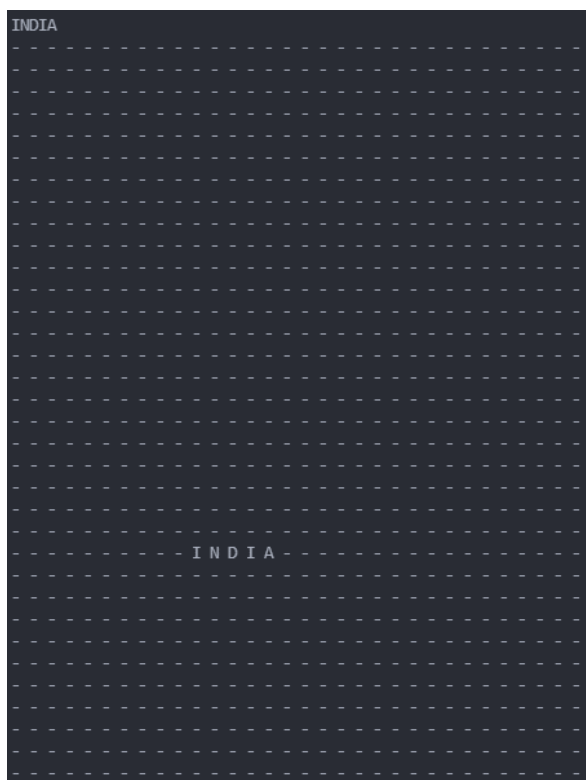
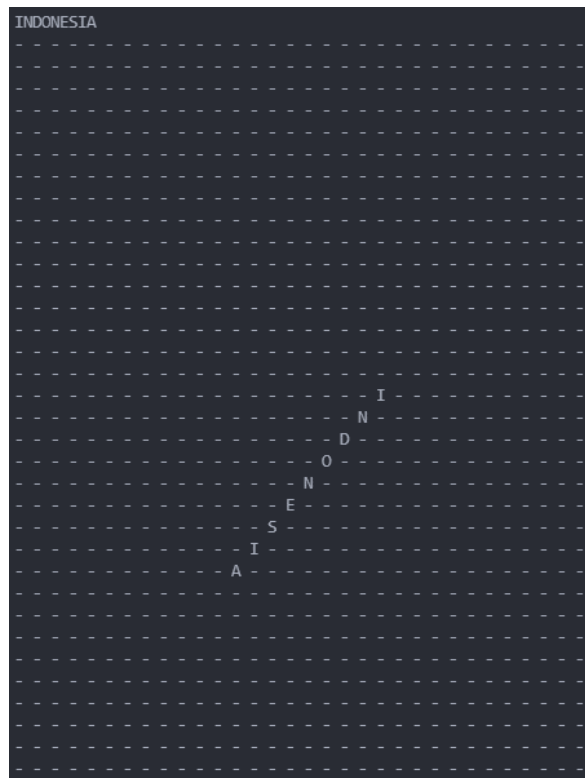
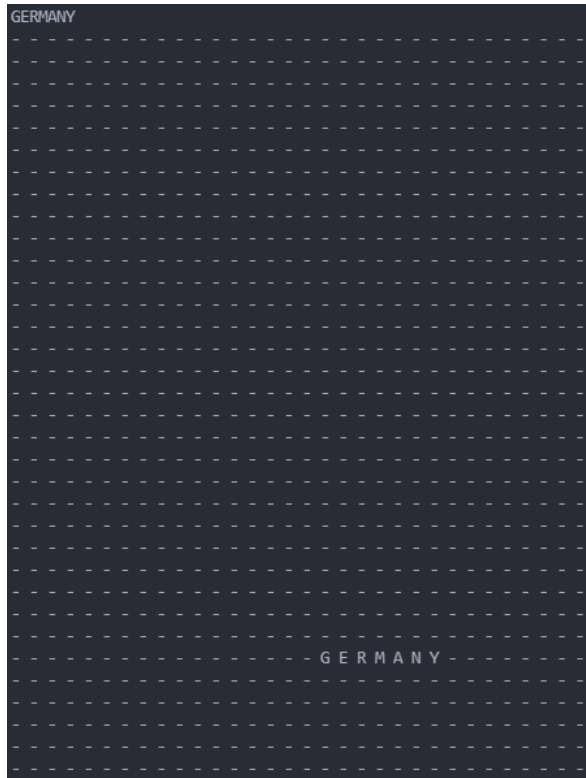
A I P O I H T E

EGYPT

T
P
Y
G
E

FRANCE

F
R
A
N
C
E



ITALY

Y
L
A
T
I

KENYA

A Y N E K

JAPAN

N
A
P
A
J

MEXICO

O
C
I
X
E
M

MYANMAR

```

- - - - - R - - - - -
- - - - - A - - - - -
- - - - - M - - - - -
- - - - - N - - - - -
- - - - - A - - - - -
- - - - - Y - - - - -
- - - - - M - - - - -

```

```
Execution time: 588 ms
Number of comparisons: 15169
Words found: 17
```

large3.txt

Insert File Name: large3.txt

Puzzle size: 36x34

Puzzle keywords: 15 words

=====
= Solution =
=====

APE

E
P
A

BAT

B
A
T

BEE

B
E
E

BUFFALO

B
U
F
F
A
L
O

BUTTERFLY

B
U
T
T
E
R
F
L
Y

CAT

T A C

CAMEL

C
A
M
E
L

CATERPILLAR

R
A
L
L
I
P
R
E
T
A
C

CLAM

- M -
- A -
- L -
- C -

CRAB

- C -
- R -
- A -
- B -

COCKROACH

- H C A O R K C O C -

CRICKET

- T -
- E -
- K -
- C -
- I -
- R -
- C -

DOLPHIN

D
O
L
P
H
I
N

FROG

F
R
O
G

Execution time: 531 ms
Number of comparisons: 14292
Words found: 15

DOVE

D
O
V
E

BAB IV
PENUTUP

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

Repo Github: <https://github.com/vincen-tho/Word-Search-Puzzle-Solver.git>