LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

Penyelesaian Word Search Puzzle dengan Algoritma Brute Force

NIM	: 13520072
Nama	: Jova Andres Riski Sirait
Kelas	: K03
Tanggal Pembuatan	: 24 Januari 2022

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

A. Algoritma Brute Force

Algoritma brute force adalah algoritma penyelesaian masalah dengan pendekatan yang sederhana, langsung, dan dengan cara yang jelas. Pendekatan algoritma brute force adalah dengan mengecek semua kemungkinan yang ada, sehingga solusi yang diminta hampir selalu dapat ditemukan pada satu atau lebih kombinasi yang dihasilkan. Meskipun cara ini tidak efisien dari sisi waktu, karena kombinasi yang dihasilkan mungkin sangat banyak, algoritma brute force cukup mudah diimplementasikan dan dapat menyelesaikan kebanyakan masalah dalam pemrograman.

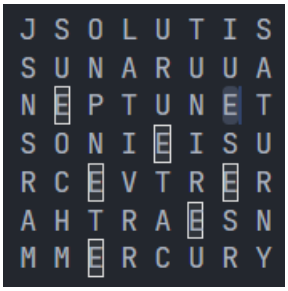
Pada tugas kali ini, yaitu penyelesaian word search puzzle, setiap kata akan dicek keberadaannya di dalam puzzle menggunakan algoritma brute force, dengan langkah-langkah sebagai berikut.

1. Dokumen teks berisi puzzle dan daftar kata yang ada akan dibaca dan disimpan sebagai data, puzzle akan disimpan sebagai matriks, sedangkan daftar kata akan disimpan ke dalam list.

   J S O L U T I S
   S U N A R U U A
   N E P T U N E T
   S O N I E I S U
   R C E V T R E R
   A H T R A E S N
   M M E R C U R Y
   (Puzzle akan disimpan sebagai matriks yang berisi karakter)

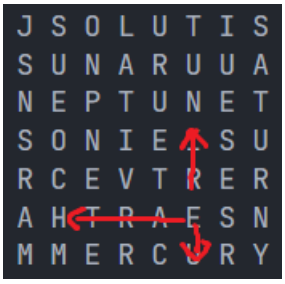
   EARTH
   JUPITER
   MARS
   MERCURY
   NEPTUNE
   SATURN
   URANUS
   VENUS
   (Daftar kata disimpan sebagai list)

2. Program akan mengiterasi satu per satu kata dari daftar kata yang telah disediakan untuk dicari dalam puzzle. Pengecekan pertama adalah perbandingan huruf pertama, jika huruf pertama kata yang dicari ada di dalam matriks, maka proses pengecekan akan dilanjutkan ke tahap selanjutnya. Jika tidak ada, maka pencarian akan gagal.
   Sebagai contoh, pada puzzle di atas, jika kita ingin menemukan kata EARTH dalam puzzle, maka langkah yang pertama yang dilakukan adalah mencari huruf E di dalam puzzle tersebut.



Gambar 1.1 Huruf E yang akan di cek

3. Proses pengecekan akan dilakukan ke setiap arah yang memungkinkan, yaitu ke atas, bawah, kiri, kanan, kanan atas, kanan bawah, kiri atas, dan kiri bawah. Sebelum pengecekan huruf yang sesuai dilakukan, terlebih dahulu dilakukan pengecekan panjang kata terhadap sisa huruf yang mungkin pada arah tertentu. Jika jumlah huruf yang dicari ada n, sedangkan jumlah huruf pada puzzle ke arah tertentu lebih dari n, maka pengecekan ke arah tersebut sudah dipastikan gagal.



Gambar 1.2 Pengecekan beberapa arah

Pada gambar di atas, pengecekan ke atas tidak akan mendapatkan hasil yang sesuai karena huruf kedua berbeda, pengecekan ke bawah akan gagal karena huruf yang tersedia tidak cukup untuk kata EARTH, sedangkan pengecekan ke kiri berhasil karena setiap hurufnya sesuai.

4. Perbandingan akan dilakukan terhadap setiap huruf yang ada dalam kata yang dicari secara berurutan pada satu arah tertentu. Jika hurufnya sesuai sampai akhir, maka kata telah ditemukan. Akan dibuat matriks salinan puzzle yang hanya berisi kata yang dicari, sedangkan kata lain akan digantikan oleh karakter ‘-‘, kemudian hasil akan ditunjukkan ke layar. Jika tidak, maka pengecekan dilanjutkan ke arah yang lain, dan jika tidak ada yang sesuai, maka pencarian gagal.

```
Solution for EARTH
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- H T R A E - - -
- - - - - - - - -
68 letter comparisons
```

Gambar 1.3 Solusi untuk kata EARTH pada puzzle

5. Setelah satu kata selesai dicari, maka akan dilanjutkan ke kata berikutnya.

B. Source Code Program

Program ditulis dalam bahasa Java, dan terdiri dari dua buah file yaitu Main.java sebagai kode utama dan juga PuzzleSolver.java yang berisi algoritma utama (brute force).

File Main.java

```java
import lib.PuzzleSolver;

import java.util.Scanner;

public class Main {

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

        PuzzleSolver puzzleSolver = new PuzzleSolver();
        System.out.print("Puzzle text file: ");
        filename = scanner.nextLine();
        puzzleSolver.readMatrix(filename);
        if (puzzleSolver.fileExist) {
            puzzleSolver.resetSolution();
            puzzleSolver.solve();
            puzzleSolver.displayStat();
        }
    }
}
```

Atribut Utama Kelas Puzzle Solver

```java
package lib;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Objects;

public class PuzzleSolver {
    public boolean fileExist = true;
    final int maxRow = 50;
    final int maxCol = 50;
    final int maxWord = 50;
    public int comparison = 0;
    public int totalComparison = 0;
    public int nRow, nCol, nWord;
    public double time = 0;
    public boolean found = false;
    public String[][] Matrix = new String[maxRow][maxCol];
    public String[][] Solution = new String[maxRow][maxCol];
    public String[] Words = new String[maxWord];
```

Input dan Output

```java
    public void readMatrix(String filename) {
        try {
            // Read puzzle
            String line;
            BufferedReader reader = new BufferedReader(new FileReader("../test/" +
filename));
            int i, j, k;
            i = 0;
            while (!(line = reader.readLine()).isEmpty()) {
                j = 0;
                for (String value : line.split(" ")) {
                    this.Matrix[i][j] = value;
                    j += 1;
                }
                i += 1;
                this.nCol = j;
            }
            this.nRow = i;

            // Read solutions
            k = 0;
            while ((line = reader.readLine()) != null) {
```

```
                this.Words[k] = line;
                k += 1;
            }
            this.nWord = k;

            reader.close();
        } catch (IOException e) {
            this.fileExist = false;
            System.out.println("Error occurred, " + e.getMessage());
        }
    }

    public void resetSolution()
    {
        int i, j;
        for (i = 0; i < this.nRow; i++) {
            for (j = 0; j < this.nCol; j++) {
                this.Solution[i][j] = "-";
            }
        }
    }

    public void displayPuzzle()
    {
        System.out.println();
        System.out.println("PUZZLE");
        int i, j;
        for (i = 0; i < this.nRow; i++) {
            for (j = 0; j < this.nCol; j++) {
                System.out.print(this.Matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public void displaySolutions()
    {
        int i, j;
        for (i = 0; i < this.nRow; i++) {
            for (j = 0; j < this.nCol; j++) {
                System.out.print(this.Solution[i][j] + " ");
            }
            System.out.println();
        }
        System.out.print(this.comparison + " letter comparisons\n");
```

```
    }

    public void displayStat() {
        System.out.println("\nStatistics");
        System.out.println("Total letter comparisons: " + this.totalComparison);
        System.out.println("Brute force time: " + this.time / 1000000 + " ms\n");
    }
```

Algoritma Brute Force

```
    public void checkUp(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (row + 1 >= currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row - i][col],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row - i][col] = this.Matrix[row - i][col];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkDown(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
```

```
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (this.nRow - row + 1 >= currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[i + row][col],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[i + row][col] = this.Matrix[i + row][col];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkRight(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (this.nCol - col + 1 >= currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row][col + i],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row][col + i] = this.Matrix[row][col + i];
                        i += 1;
                    }
```

```
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkLeft(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (col + 1 >= currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row][col - i],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row][col - i] = this.Matrix[row][col - i];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkUpRight(int num, int row, int col) {
```

```
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (row + 1 >= currentWord.length() && this.nCol - col + 1 >=
currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row - i][col + i],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row - i][col + i] = this.Matrix[row - i][col +
i];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkUpLeft(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (row + 1 >= currentWord.length() && col + 1 >= currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row - i][col - i],
currentWord.substring(i, i + 1))) {
```

```
                        match = false;
                    } else {
                        this.Solution[row - i][col - i] = this.Matrix[row - i][col -
i];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkDownRight(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (this.nRow - row + 1 >= currentWord.length() && this.nCol - col + 1 >=
currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row + i][col + i],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row + i][col + i] = this.Matrix[row + i][col +
i];
                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
```

```
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void checkDownLeft(int num, int row, int col) {
        if (!this.found) {
            String currentWord = this.Words[num];
            boolean match = true;
            int i = 1;
            long start, stop;
            start = System.nanoTime();
            if (this.nRow - row + 1 >= currentWord.length() && col + 1 >=
currentWord.length()) {
                this.Solution[row][col] = this.Matrix[row][col];
                while (match && i < currentWord.length()) {
                    this.comparison++;
                    if (!Objects.equals(this.Matrix[row + i][col - i],
currentWord.substring(i, i + 1))) {
                        match = false;
                    } else {
                        this.Solution[row + i][col - i] = this.Matrix[row + i][col -
i];

                        i += 1;
                    }
                }
                stop = System.nanoTime();

                this.time += stop - start;
                if (match) {
                    this.found = true;
                } else {
                    resetSolution();
                }
            }
        }
    }

    public void solve() {
        int N = this.nWord;
        int i, j, nRow, nCol;
        for (i = 0; i < N; i++) {
            j = 0;
```

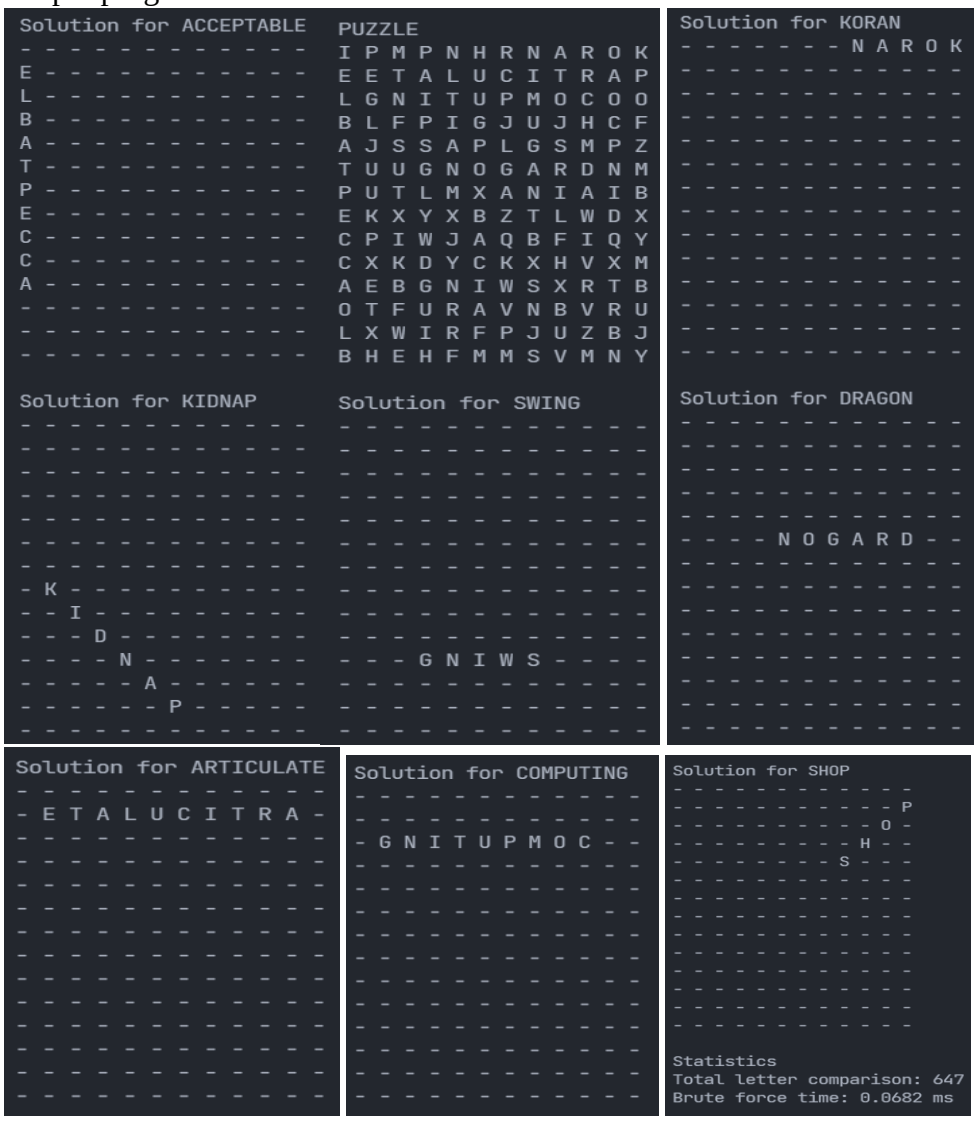
```
            String currentWord = this.Words[i];
            char currentChar = currentWord.charAt(j);
            this.found = false;
            resetSolution();

            nRow = 0;
            nCol = 0;
            while (nRow < this.nRow && !this.found) {
                while (nCol < this.nCol && !this.found) {
                    this.comparison++;
                    if (Objects.equals(this.Matrix[nRow][nCol],
String.valueOf(currentChar))) {
                        checkUp(i, nRow, nCol);
                        checkDown(i, nRow, nCol);
                        checkRight(i, nRow, nCol);
                        checkLeft(i, nRow, nCol);
                        checkUpRight(i, nRow, nCol);
                        checkUpLeft(i, nRow, nCol);
                        checkDownRight(i, nRow, nCol);
                        checkDownLeft(i, nRow, nCol);
                    }
                    nCol++;
                }
                nCol = 0;
                nRow++;
            }

            if (!this.found) {
                System.out.println();
                System.out.println(currentWord + " not found!");
            } else {
                System.out.println();
                System.out.println("Solution for " + currentWord);
                displaySolutions();
            }

            this.totalComparison += this.comparison;
            this.comparison = 0;
        }
    }
}
```

C. Screenshot Input dan Output

1. small1.txt

Ukuran Puzzle	14x12
Banyak kata	8
Banyak perbandingan kata	647

Output program:



Gambar 3.1 Hasil eksekusi percobaan pertama

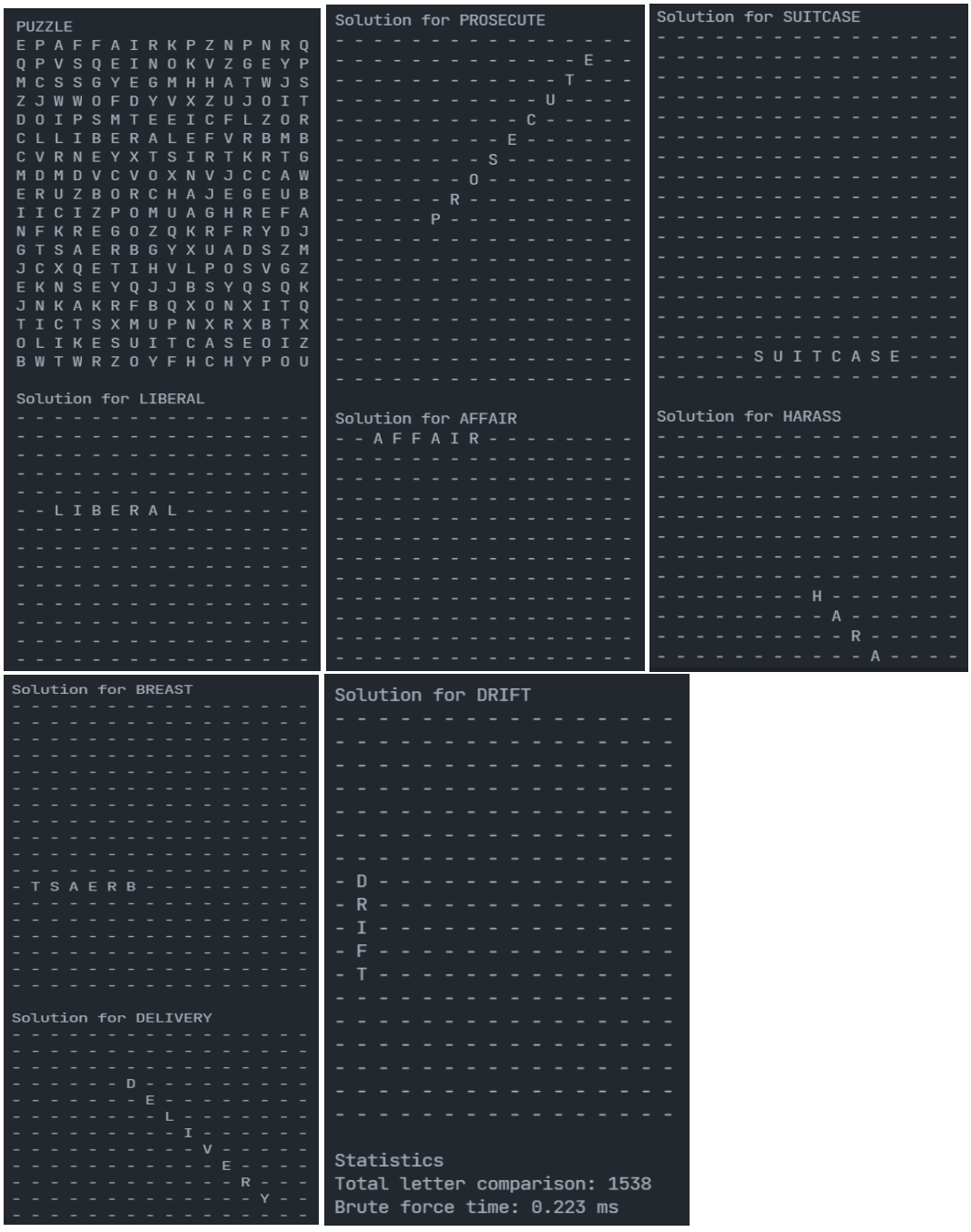
2. small2.txt

Ukuran Puzzle	15x15
Banyak kata	8
Banyak perbandingan kata	1380

Gambar 3.2 Hasil eksekusi percobaan kedua

3. small3.txt

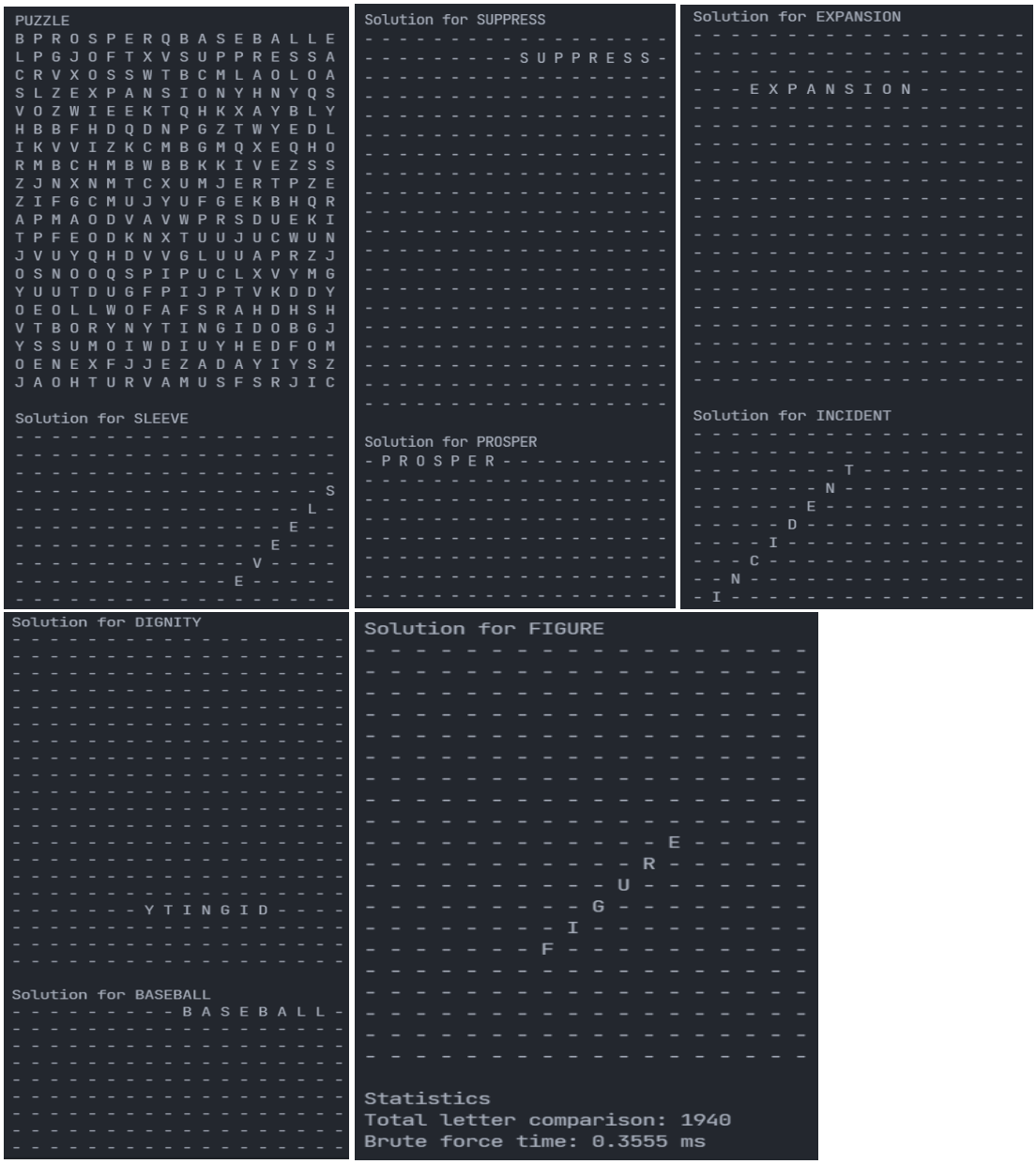
Ukuran Puzzle	16x14
Banyak kata	9
Banyak perbandingan kata	1538



Gambar 3.3 Hasil eksekusi percobaan ketiga

4. medium1.txt

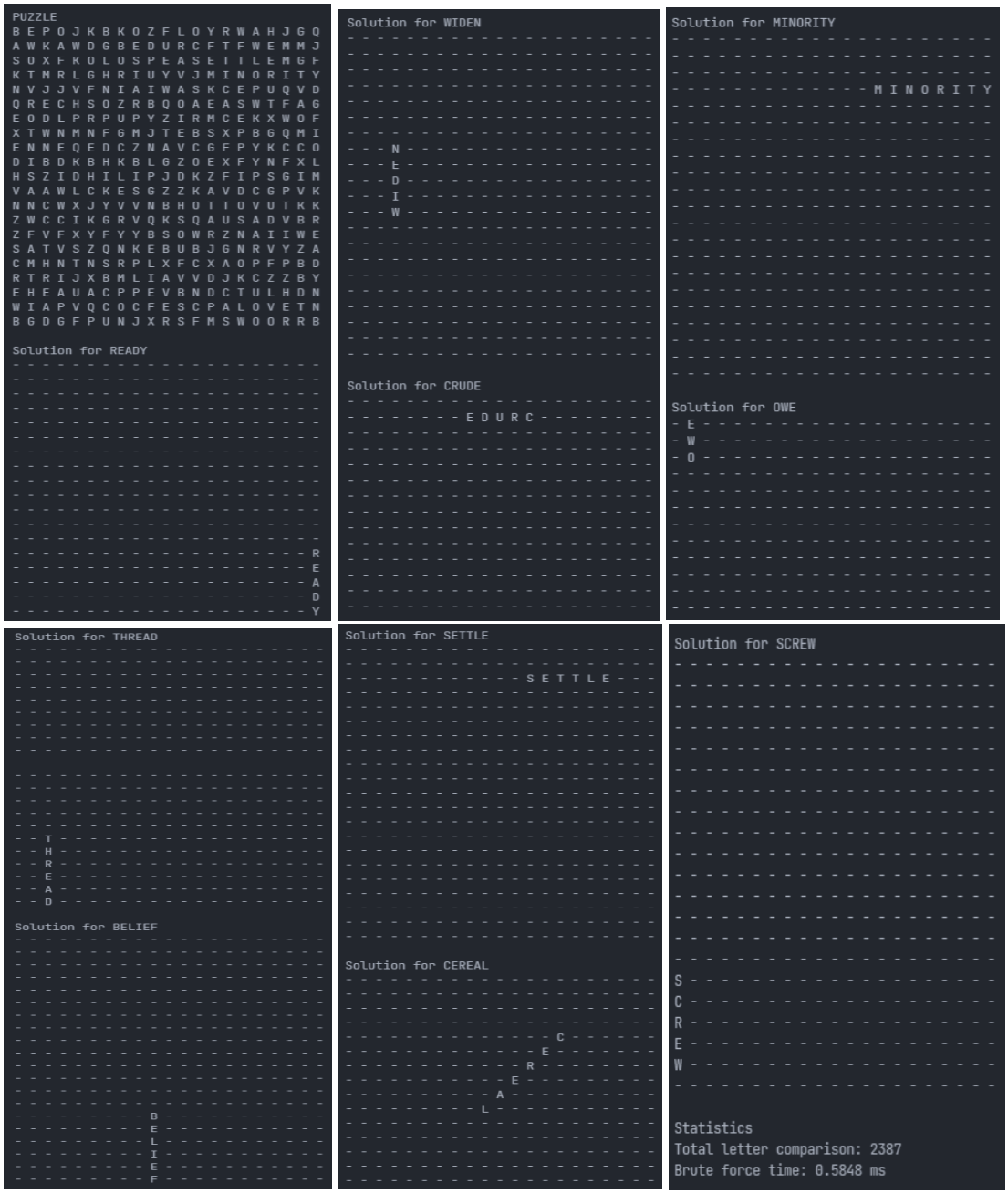
Ukuran Puzzle	20x18
Banyak kata	10
Banyak perbandingan kata	1940



Gambar 3.4 Hasil eksekusi percobaan keempat

5. medium2.txt

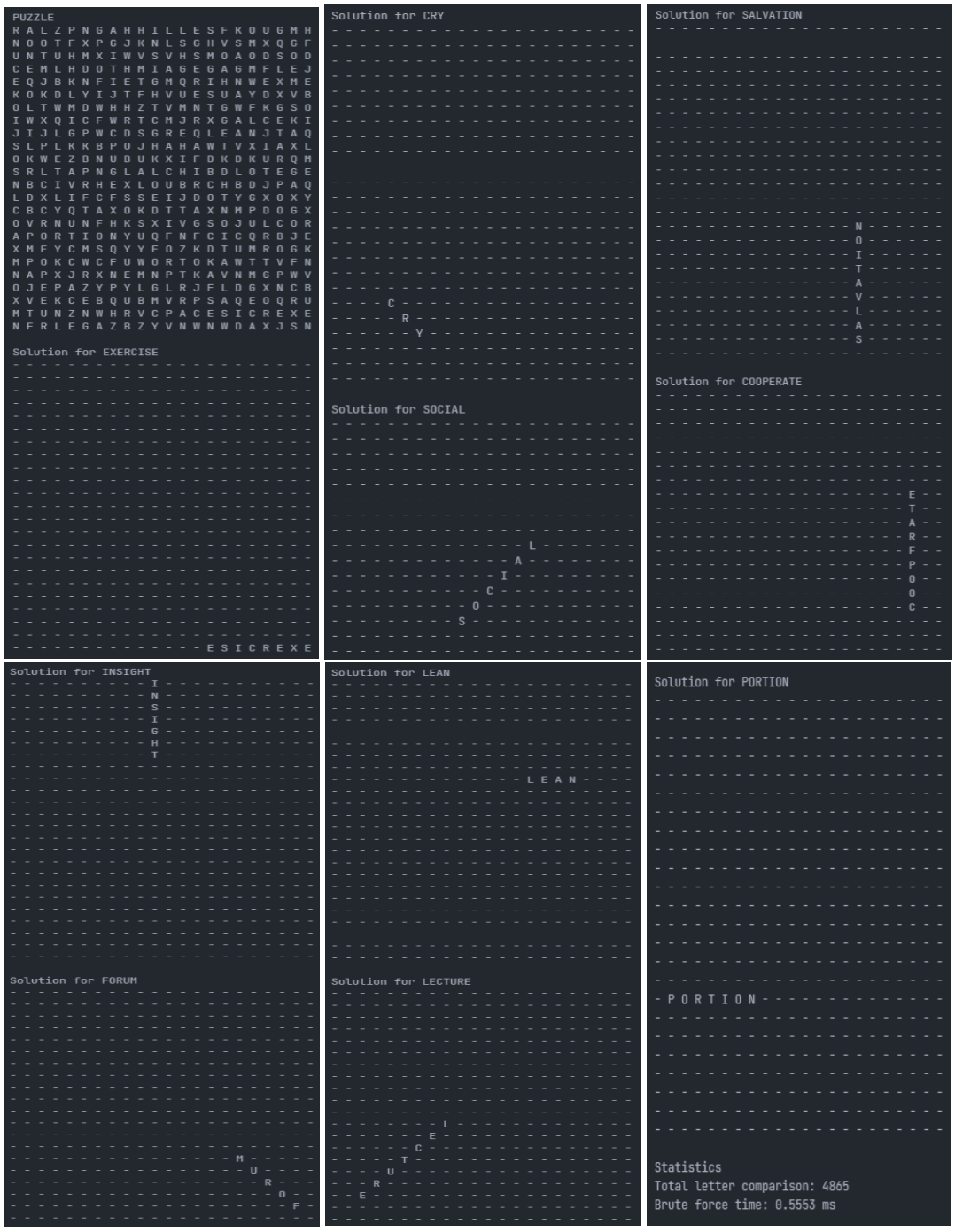
Ukuran Puzzle	21x21
Banyak kata	11
Banyak perbandingan kata	2387



Gambar 3.5 Hasil eksekusi percobaan kelima

6. medium3.txt

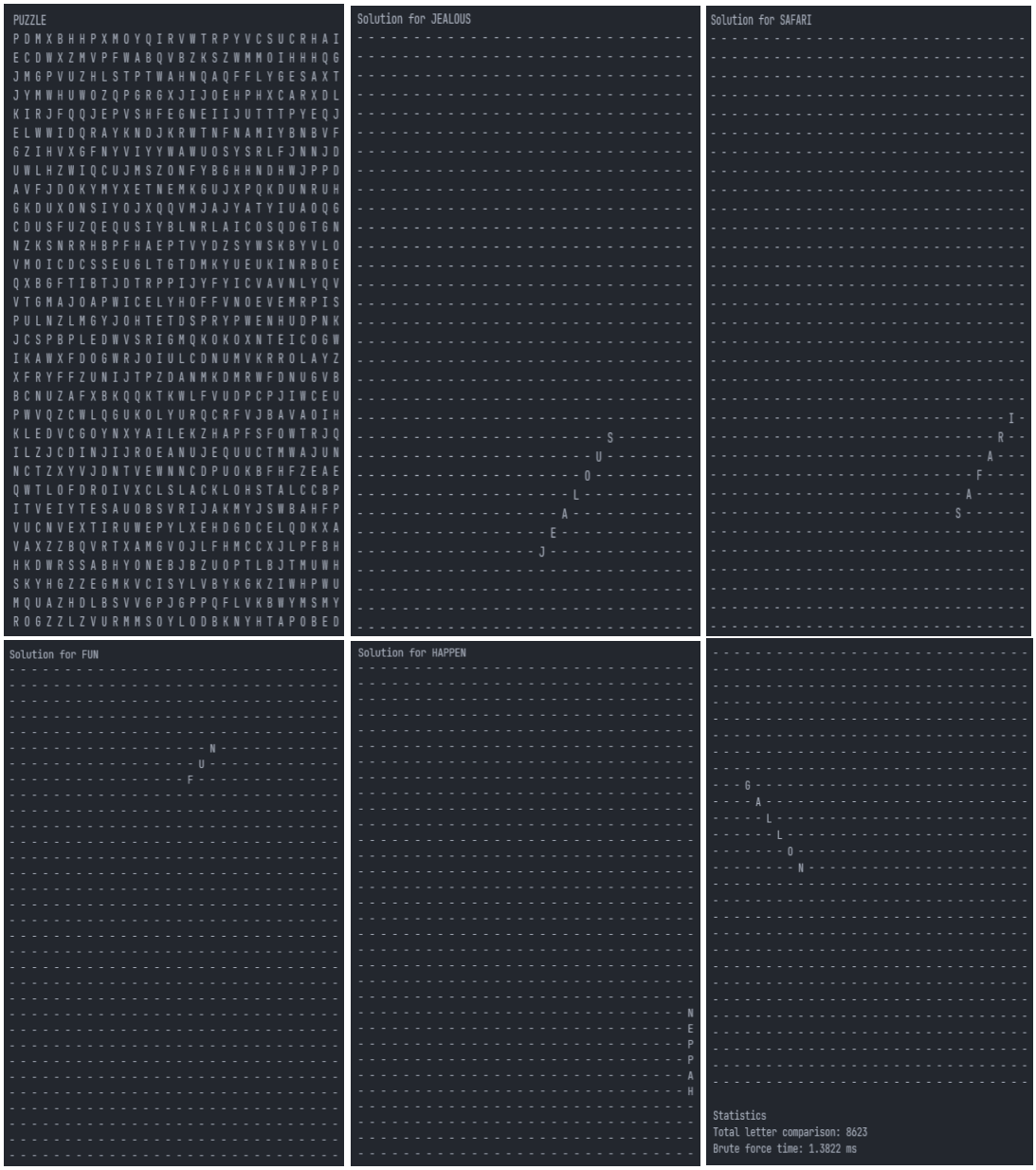
Ukuran Puzzle	24x22
Banyak kata	12
Banyak perbandingan kata	4865



Gambar 3.6 Hasil eksekusi percobaan keenam

7. large1.txt

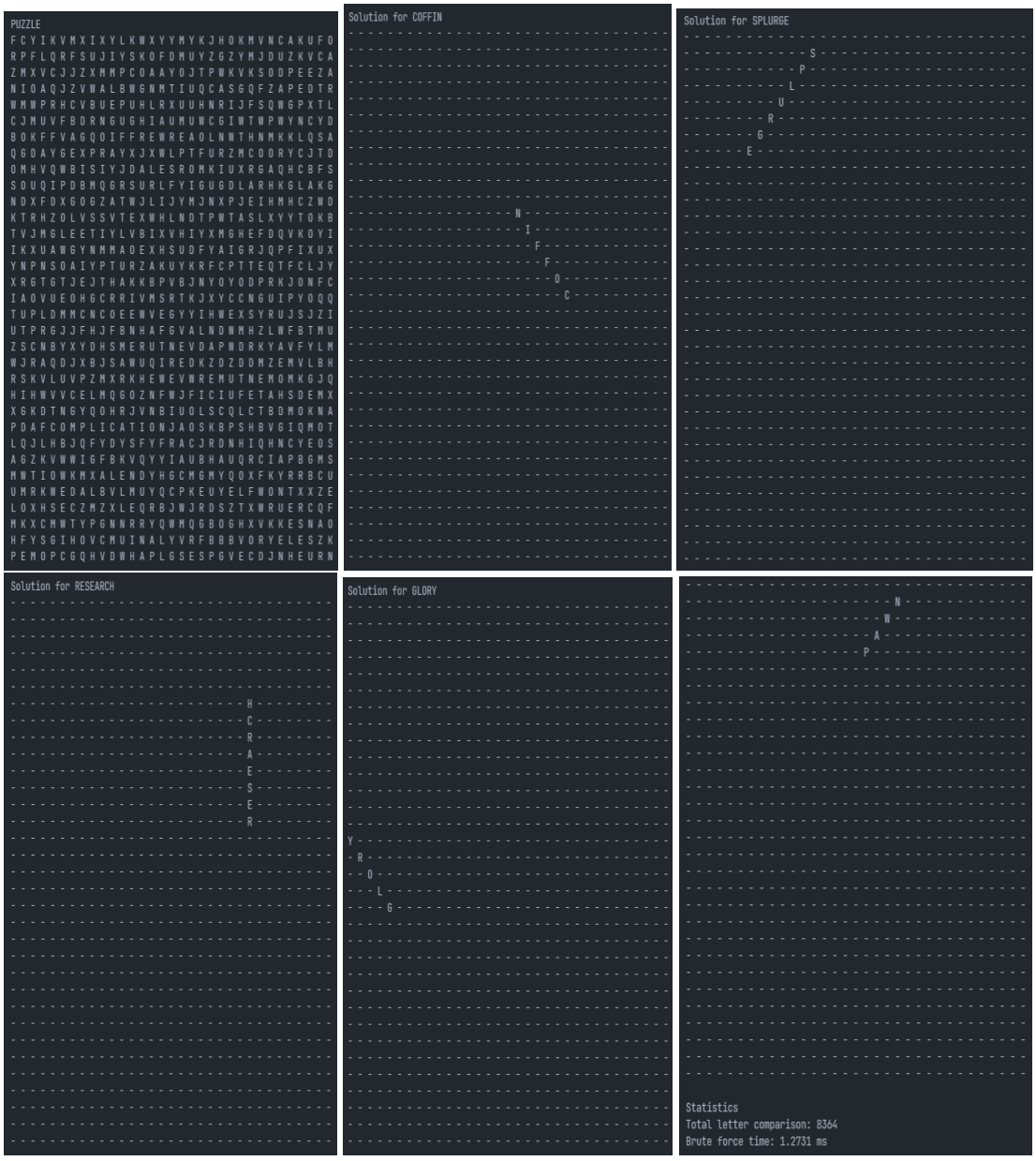
Ukuran Puzzle	32x30
Banyak kata	13
Banyak perbandingan kata	8623



Gambar 3.7 Hasil eksekusi percobaan ketujuh

8. large2.txt

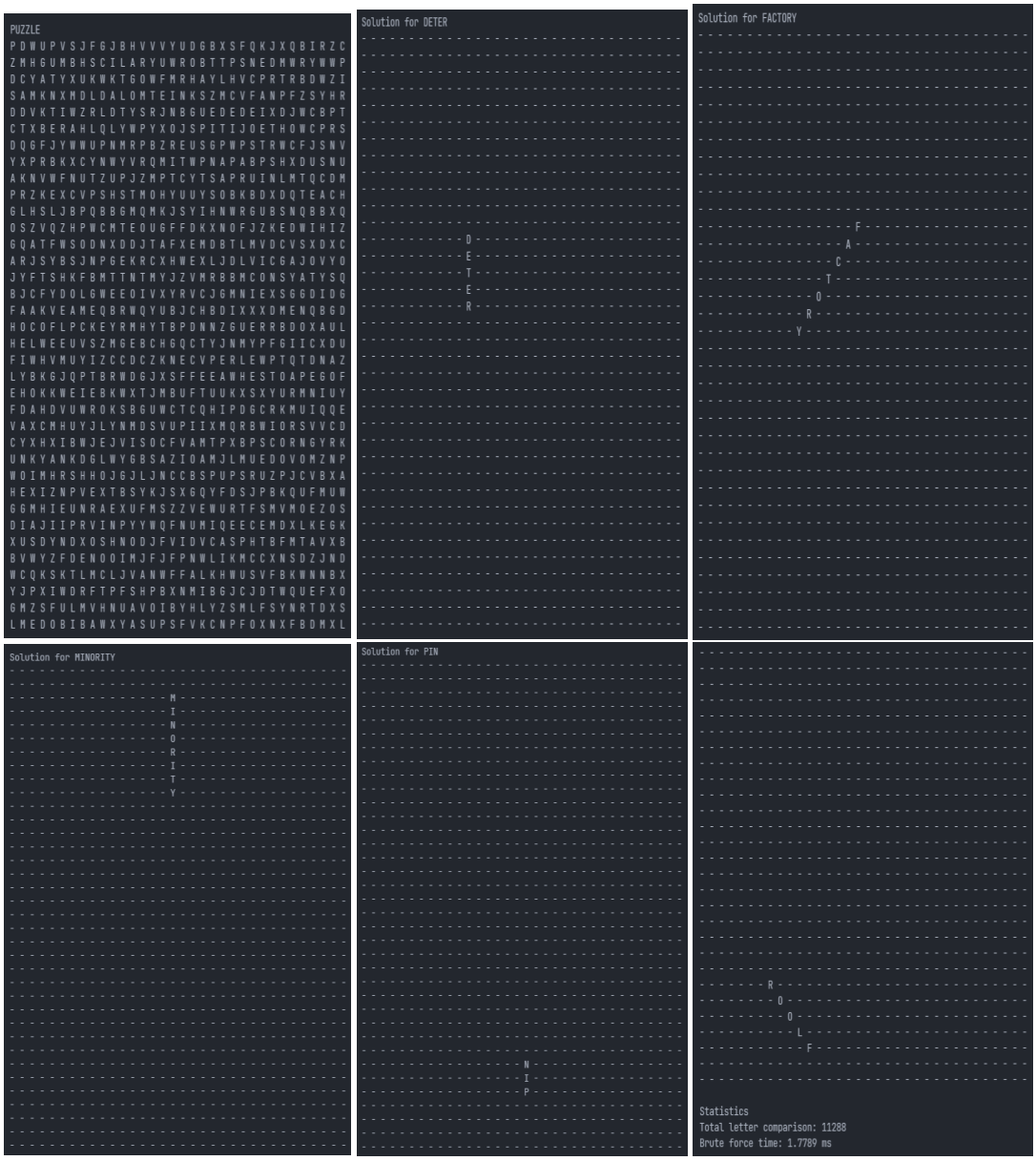
Ukuran Puzzle	33x33
Banyak kata	14
Banyak perbandingan kata	8364



Gambar 3.8 Hasil eksekusi percobaan kedelapan

9. large3.txt

Ukuran Puzzle	36x34
Banyak kata	15
Banyak perbandingan kata	11288



Gambar 3.9 Hasil eksekusi percobaan kesembilan

Keterangan:

- Tidak semua output program ditampilkan dikarenakan keterbatasan ruang pada terminal.
- Waktu eksekusi tidak disajikan karena bisa berbeda tergantung environment pengujian.

D. Alamat Drive

https://github.com/jovaandres/Tucil1_13520072

E. Tabel Checklist

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.	✓	