

LAPORAN TUGAS KECIL

**Penyelesaian Word Search Puzzle dengan Algoritma
Brute Force**

Ditujukan untuk memenuhi salah satu tugas kecil mata kuliah IF2211 Strategi Algoritma
pada Semester II Tahun Akademik 2021/2022

Disusun oleh:

Sarah Azka Arief (K2)

13520083



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

2022

BAB I

Deskripsi Algoritma Brute Force

Algoritma brute force merupakan teknik penyelesaian masalah yang mana eksistensi solusi dari suatu permasalahan akan diketahui dengan cara mencoba setiap jawaban yang mungkin. Oleh karena itu, metode ini lebih bergantung kepada kemampuan komputasi dan proses mencoba semua kemungkinan ketimbang efisiensi dan optimasi. Pada tugas kecil ini, program dibuat dengan menerapkan algoritma brute force demi mencari penyelesaian dari *word search puzzle* yang diberikan dalam bentuk file txt yang sesuai dengan spesifikasi dari tugas kecil. Program dibuat dengan menggunakan bahasa Java dalam bentuk satu file source code tunggal berjudul WordSearch.java.

Alur dari program dimulai dengan pembacaan input *user*. Input ini berupa nama file txt yang berada pada folder bin yang akan dicari penyelesaiannya. Proses pembacaan terhadap file yang diinput user tersebut terdiri atas penyimpanan isi file yang berupa matriks berisi huruf dalam bentuk suatu 2D *ArrayList of String* yang kemudian diubah menjadi 2D *array of String* bernama 'matrix' serta penyimpanan isi file yang berupa list kata yang harus dicari dalam bentuk suatu *ArrayList* berisi *array of char* bernama 'words'. Setelah proses pembacaan file dan penyimpanan informasi dari file telah selesai, 'matrix' dan 'words' diproses menggunakan 8 fungsi yang dibedakan berdasarkan arah pencariannya untuk mencari penyelesaian dari *word search puzzle* tersebut. Berikut 8 fungsi tersebut:

1. searchUp(matrix, word)
Mencari kata pada matriks dari bawah ke atas
2. searchDown(matrix, word)
Mencari kata pada matriks dari atas ke bawah
3. searchLeft(matrix, word)
Mencari kata pada matriks dari kanan ke kiri
4. searchRight(matrix, word)
Mencari kata pada matriks dari kiri ke kanan
5. searchDiagUpLeft(matrix, word)
Mencari kata pada matriks dari bawah ke atas dan kanan ke kiri
6. searchDiagUpRight(matrix, word)
Mencari kata pada matriks dari bawah ke atas dan kiri ke kanan
7. searchDiagDownLeft(matrix, word)

Mencari kata pada matriks dari atas ke bawah dan kanan ke kiri

8. `searchDiagDownRight(matrix, word)`

Mencari kata pada matriks dari atas ke bawah dan kiri ke kanan

Sebelum fungsi dimulai, timer dinyalakan untuk menghitung waktu yang dibutuhkan bagi seluruh fungsi untuk mencari penyelesaian dari *word search puzzle* yang diberikan.

Setiap fungsi akan dimulai dengan looping terhadap huruf pertama dari setiap kata yang ada pada list 'word'. Kemudian, di dalam loop tersebut terjadi looping pada 'matrix' agar matriks teriterasi sesuai arah dari masing-masing fungsi. Setiap huruf pada matriks yang diiterasi akan dicek kesamaannya dengan huruf pertama pada kata yang sedang diiterasi. Apabila sama, koordinat huruf tersebut akan disimpan dalam suatu ArrayList bernama 'toPrint' yang berfungsi untuk menyimpan secara temporer koordinat dari setiap huruf yang membentuk suatu kata dan dilanjutkan dengan pengecekan kesamaan dari huruf berikutnya dengan huruf berikutnya dari kata yang sedang diiterasi. Apabila setelahnya ditemukan perbedaan huruf antara yang sedang diiterasi pada matriks dan huruf yang sedang dicek pada kata, maka 'toPrint' akan direset atau dibersihkan menggunakan fungsi clear.

Proses perbandingan huruf terhadap suatu kata akan berakhir apabila seluruh huruf pada matriks telah dicek atau kata yang sedang dibandingkan telah ditemukan. Panjang kata yang sedang dibandingkan digunakan sebagai indikator yang menentukan apakah kata tersebut sudah ditemukan atau belum. Apabila kata yang sedang dibandingkan telah ditemukan, kata tersebut akan dihapus dari 'words' dan hasil 'toPrint' yang berisi koordinat dari setiap huruf yang membentuk kata tersebut akan dimasukkan ke dalam suatu variabel global berupa ArrayList yang bernama 'finalPrint' yang berisi kumpulan koordinat huruf dari semua kata yang ditemukan.

Apabila seluruh fungsi telah selesai dijalankan atau seluruh kata pada 'words' telah ditemukan, timer akan berhenti dan program akan menampilkan ukuran matriks, waktu yang dibutuhkan untuk mencari seluruh kata pada *word search puzzle*, matriks yang berisi hasil kata yang ditemukan secara satu per satu dengan menggunakan fungsi printMatrix untuk mencetak hasil dari 'finalPrint', serta kata yang tersisa alias belum ditemukan dari kumpulan kata pada 'words'. Perlu dicatat bahwa proses menampilkan kata yang tersisa hanya terjadi apabila ada kata pada 'words' yang tidak berhasil diidentifikasi oleh program atau kata tersebut memang tidak ada pada 'matrix' karena adanya *typo* baik pada 'words' yang diberikan ataupun 'matrix' (hingga kini, *test case* yang menunjukkan adanya kata yang tersisa selalu diakibatkan oleh

ketidakadaannya kata pada matriks *word search puzzle*). Jumlah perbandingan pada program ini bertambah setiap kali terjadi proses perbandingan antara huruf pada matriks dan huruf dari kata yang sedang dicari. Perlu dicatat bahwa apabila ditemukan dua huruf berturut-turut pada matriks yang sama dengan huruf pertama kata yang dicari maka akan dilakukan pengecekan dua kali terhadap huruf kedua di matriks tersebut sehingga kedua huruf tersebut akan memiliki total 3 kali perbandingan.

BAB II

Source Program

```
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.util.*;
4
5
6 public class WordSearch{
7     public static String getLetter(ArrayList<char[]> words, int row, int col){
8         return Character.toString(words.get(row)[col]);
9     }
10    public static void printWords(ArrayList<char[]> words){
11        for (int i = 0; i < words.size(); i++){
12            System.out.println(words.get(i));
13        }
14    }
15    public static void updateProgress(ArrayList<String> toPrint, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison,
16        ArrayList<Integer> toDelete, int i, int comparison){
17        ArrayList<String> temp = new ArrayList<>(toPrint);
18        finalPrint.add(temp);
19        finalComparison.add(comparison);
20        toDelete.add(i);
21    }
22    public static void addCoordinate(ArrayList<String> toPrint, int x, int y){
23        String temp = String.valueOf(x) + "," + String.valueOf(y);
24        toPrint.add(temp);
25    }
26    public static void printMatrix(String[][] matrix, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
27        int matrixRow = matrix.length, matrixCol = matrix[0].length;
28        int i = 0, tot = 0;
29        for (ArrayList<String> toPrint : finalPrint){
30            for (int m = 0; m < matrixRow; m++){
31                for (int n = 0; n < matrixCol; n++){
32                    String temp = String.valueOf(m) + "," + String.valueOf(n);
33                    if (toPrint.contains(temp)){
34                        System.out.print(matrix[m][n]);
35                    }
36                    else{
37                        System.out.print("-");
38                    }
39                    System.out.print(" ");
40                }
41                System.out.println();
42            }
43            System.out.println("Total Comparison: " + (finalComparison.get(i)).intValue());
44            tot += (finalComparison.get(i)).intValue();
45            System.out.println();
46            i++;
47        }
48        System.out.println("Total Comparison of All Words: " + tot);
49    }
50    public static void searchUp(String[][] matrix, ArrayList<char[]> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
51        ArrayList<Integer> toDelete = new ArrayList<>();
52        ArrayList<String> toPrint = new ArrayList<>();
53        int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
54    }
```

```
1     for (int i = 0; i < words.size(); i++){
2         int wordCheck = 0;
3         boolean diff = true;
4         for (int j = matrixCol - 1; j >= 0; j--){
5             for (int k = matrixRow - 1; k >= 0; k--){
6                 comparison++;
7                 if (matrix[k][j].equals(getLetter(words, i, wordCheck))){
8                     if (diff == true){
9                         diff = false;
10                    }
11                    addCoordinate(toPrint, k, j);
12                    wordCheck++;
13                }
14                else{
15                    if (diff == false){
16                        diff = true;
17                        wordCheck = 0;
18                        toPrint.clear();
19                        comparison++;
20                        if (matrix[k][j].equals(getLetter(words, i, 0))){
21                            diff = false;
22                            addCoordinate(toPrint, k, j);
23                            wordCheck++;
24                        }
25                    }
26                }
27                if (wordCheck == words.get(i).length){
28                    updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
29                    j = -1;
30                    k = -1;
31                }
32            }
33            wordCheck = 0;
34            diff = true;
35            toPrint.clear();
36        }
37    }
38    for (int i = toDelete.size() - 1; i >= 0; i--){
39        words.remove((toDelete.get(i)).intValue());
40    }
41    return;
42 }
43 public static void searchDown(String[][] matrix, ArrayList<char[]> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
44     ArrayList<Integer> toDelete = new ArrayList<>();
45     ArrayList<String> toPrint = new ArrayList<>();
46     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
47     for (int i = 0; i < words.size(); i++){
48         int wordCheck = 0;
49         boolean diff = true;
50         for (int j = 0; j < matrixCol; j++){
51             for (int k = 0; k < matrixRow; k++){
52                 comparison++;
53                 if (matrix[k][j].equals(getLetter(words, i, wordCheck))){
54                     if (diff == true){
```

```

1         diff = false;
2     }
3     addCoordinate(toPrint, k, j);
4     wordCheck++;
5 }
6 }
7 else{
8     if (diff == false){
9         diff = true;
10        wordCheck = 0;
11        toPrint.clear();
12        comparison++;
13        if (matrix[k][j].equals(getLetter(words, i, 0))){
14            diff = false;
15            addCoordinate(toPrint, k, j);
16            wordCheck++;
17        }
18    }
19    if (wordCheck == words.get(i).length){
20        updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
21        j = matrixCol;
22        k = matrixRow;
23    }
24 }
25 wordCheck = 0;
26 diff = true;
27 toPrint.clear();
28 }
29 }
30 for (int i = toDelete.size() - 1; i >= 0; i--){
31     words.remove((toDelete.get(i)).intValue());
32 }
33 return;
34 }
35 public static void searchLeft(String[][] matrix, ArrayList<char> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
36     ArrayList<Integer> toDelete = new ArrayList<>();
37     ArrayList<String> toPrint = new ArrayList<String>();
38     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
39     for (int i = 0; i < words.size(); i++){
40         int wordCheck = 0;
41         boolean diff = true;
42         for (int j = matrixRow - 1; j >= 0; j--){
43             for (int k = matrixCol - 1; k >= 0; k--){
44                 comparison++;
45                 if (matrix[j][k].equals(getLetter(words, i, wordCheck))){
46                     if (diff == true){
47                         diff = false;
48                     }
49                     addCoordinate(toPrint, j, k);
50                     wordCheck++;
51                 }
52             }
53             else{
54                 if (diff == false){
55                     diff = true;

```

```

1         wordCheck = 0;
2         toPrint.clear();
3         comparison++;
4         if (matrix[j][k].equals(getLetter(words, i, 0))){
5             diff = false;
6             addCoordinate(toPrint, j, k);
7             wordCheck++;
8         }
9     }
10 }
11 if (wordCheck == words.get(i).length){
12     updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
13     j = -1;
14     k = -1;
15 }
16 }
17 wordCheck = 0;
18 diff = true;
19 toPrint.clear();
20 }
21 }
22 for (int i = toDelete.size() - 1; i >= 0; i--){
23     words.remove((toDelete.get(i)).intValue());
24 }
25 return;
26 }
27 public static void searchRight(String[][] matrix, ArrayList<char> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
28     ArrayList<Integer> toDelete = new ArrayList<>();
29     ArrayList<String> toPrint = new ArrayList<String>();
30     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
31     for (int i = 0; i < words.size(); i++){
32         int wordCheck = 0;
33         boolean diff = true;
34         for (int j = 0; j < matrixRow; j++){
35             for (int k = 0; k < matrixCol; k++){
36                 comparison++;
37                 if (matrix[j][k].equals(getLetter(words, i, wordCheck))){
38                     if (diff == true){
39                         diff = false;
40                     }
41                     addCoordinate(toPrint, j, k);
42                     wordCheck++;
43                 }
44             }
45             else{
46                 if (diff == false){
47                     diff = true;
48                     wordCheck = 0;
49                     toPrint.clear();
50                     comparison++;
51                     if (matrix[j][k].equals(getLetter(words, i, 0))){
52                         diff = false;
53                         addCoordinate(toPrint, j, k);
54                         wordCheck++;

```

```

1      }
2      }
3      if (wordCheck == words.get(i).length){
4          updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
5          j = matrixRow;
6          k = matrixCol;
7      }
8      }
9      wordCheck = 0;
10     diff = true;
11     toPrint.clear();
12 }
13 }
14 for (int i = toDelete.size() - 1; i >= 0; i--){
15     words.remove((toDelete.get(i)).intValue());
16 }
17 return;
18 }
19
20 public static void searchDiagUpLeft(String[][] matrix, ArrayList<char[]> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
21     ArrayList<Integer> toDelete = new ArrayList<>();
22     ArrayList<String> toPrint = new ArrayList<String>();
23     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
24     for (int i = 0; i < words.size(); i++){
25         int wordCheck = 0;
26         boolean diff = true;
27         for (int j = 0; j < matrixCol; j++){
28             int row = matrixRow - 1;
29             int col = j;
30             while (col >= 0 && row >= 0){
31                 comparison++;
32                 if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
33                     if (diff == true){
34                         diff = false;
35                     }
36                     addCoordinate(toPrint, row, col);
37                     wordCheck++;
38                 }
39                 else{
40                     if (diff == false){
41                         diff = true;
42                         wordCheck = 0;
43                         toPrint.clear();
44                         comparison++;
45                         if (matrix[row][col].equals(getLetter(words, i, 0))){
46                             diff = false;
47                             addCoordinate(toPrint, row, col);
48                             wordCheck++;
49                         }
50                     }
51                 }
52             }
53             if (wordCheck == words.get(i).length){
54                 updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
55                 j = matrixCol;

```

```

1         break;
2     }
3     row--;
4     col--;
5 }
6 wordCheck = 0;
7 diff = true;
8 toPrint.clear();
9 }
10 for (int j = 0; j < matrixRow - 1; j++){
11     int col = matrixCol - 1;
12     int row = j;
13     while (col >= 0 && row >= 0){
14         comparison++;
15         if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
16             if (diff == true){
17                 diff = false;
18             }
19             addCoordinate(toPrint, row, col);
20             wordCheck++;
21         }
22         else{
23             if (diff == false){
24                 diff = true;
25                 wordCheck = 0;
26                 toPrint.clear();
27                 comparison++;
28                 if (matrix[row][col].equals(getLetter(words, i, 0))){
29                     diff = false;
30                     addCoordinate(toPrint, row, col);
31                     wordCheck++;
32                 }
33             }
34         }
35     }
36     if (wordCheck == words.get(i).length){
37         updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
38         j = matrixCol;
39         break;
40     }
41     row--;
42     col--;
43 }
44 wordCheck = 0;
45 diff = true;
46 toPrint.clear();
47 }
48 }
49 for (int i = toDelete.size() - 1; i >= 0; i--){
50     words.remove((toDelete.get(i)).intValue());
51 }
52 return;
53 }
54 public static void searchDiagUpRight(String[][] matrix, ArrayList<char[]> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
55     ArrayList<Integer> toDelete = new ArrayList<>();

```

```

1 ArrayList<String> toPrint = new ArrayList<String>();
2 int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
3 for (int i = 0; i < words.size(); i++){
4     int wordCheck = 0;
5     boolean diff = true;
6     for (int j = matrixCol - 1; j >= 0; j--){
7         int row = matrixRow - 1;
8         int col = j;
9         while (col < matrixCol && row >= 0){
10             comparison++;
11             if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
12                 if (diff == true){
13                     diff = false;
14                 }
15                 addCoordinate(toPrint, row, col);
16                 wordCheck++;
17             }
18             else{
19                 if (diff == false){
20                     diff = true;
21                     wordCheck = 0;
22                     toPrint.clear();
23                     comparison++;
24                     if (matrix[row][col].equals(getLetter(words, i, 0))){
25                         diff = false;
26                         addCoordinate(toPrint, row, col);
27                         wordCheck++;
28                     }
29                 }
30             }
31             if (wordCheck == words.get(i).length){
32                 updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
33                 j = -1;
34                 break;
35             }
36             row--;
37             col++;
38         }
39         wordCheck = 0;
40         diff = true;
41         toPrint.clear();
42     }
43     for (int j = matrixRow - 2; j >= 0; j--){
44         int col = 0;
45         int row = j;
46         while (col < matrixCol && row >= 0){
47             comparison++;
48             if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
49                 if (diff == true){
50                     diff = false;
51                 }
52                 addCoordinate(toPrint, row, col);
53                 wordCheck++;
54             }

```

```

1         else{
2             if (diff == false){
3                 diff = true;
4                 wordCheck = 0;
5                 toPrint.clear();
6                 comparison++;
7                 if (matrix[row][col].equals(getLetter(words, i, 0))){
8                     diff = false;
9                     addCoordinate(toPrint, row, col);
10                    wordCheck++;
11                }
12            }
13        }
14        if (wordCheck == words.get(i).length){
15            updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
16            j = -1;
17            break;
18        }
19        row--;
20        col++;
21    }
22    wordCheck = 0;
23    diff = true;
24    toPrint.clear();
25 }
26
27 for (int i = toDelete.size() - 1; i >= 0; i--){
28     words.remove(toDelete.get(i).intValue());
29 }
30 return;
31 }
32
33 public static void searchUpdownLeft(String[][] matrix, ArrayList<char[]> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
34     ArrayList<Integer> toDelete = new ArrayList<Integer>();
35     ArrayList<String> toPrint = new ArrayList<String>();
36     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
37     for (int i = 0; i < words.size(); i++){
38         int wordCheck = 0;
39         boolean diff = true;
40         for (int j = matrixCol - 1; j >= 0; j--){
41             int col = matrixCol - 1;
42             int row = j;
43             while (col >= 0 && row < matrixRow){
44                 comparison++;
45                 if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
46                     if (diff == true){
47                         diff = false;
48                     }
49                     addCoordinate(toPrint, row, col);
50                     wordCheck++;
51                 }
52                 else{
53                     if (diff == false){
54                         diff = true;
55                         wordCheck = 0;

```



```

1      toPrint.clear();
2      comparison++;
3      if (matrix[row][col].equals(getLetter(words, i, 0))){
4          diff = false;
5          addCoordinate(toPrint, row, col);
6          wordCheck++;
7      }
8  }
9  }
10     if (wordCheck == words.get(i).length){
11         updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
12         j = -1;
13         break;
14     }
15     row++;
16     col--;
17 }
18 wordCheck = 0;
19 diff = true;
20 toPrint.clear();
21 }
22 for (int j = matrixCol - 2; j >= 0; j--){
23     int row = 0;
24     int col = j;
25     while (col >= 0 && row < matrixRow){
26         comparison++;
27         if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
28             if (diff == true){
29                 diff = false;
30             }
31             addCoordinate(toPrint, row, col);
32             wordCheck++;
33         }
34         else{
35             if (diff == false){
36                 diff = true;
37                 wordCheck = 0;
38                 toPrint.clear();
39                 comparison++;
40                 if (matrix[row][col].equals(getLetter(words, i, 0))){
41                     diff = false;
42                     addCoordinate(toPrint, row, col);
43                     wordCheck++;
44                 }
45             }
46         }
47         if (wordCheck == words.get(i).length){
48             updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
49             j = -1;
50             break;
51         }
52         row++;
53         col--;
54     }

```

```

1      wordCheck = 0;
2      diff = true;
3      toPrint.clear();
4  }
5  }
6  for (int i = toDelete.size() - 1; i >= 0; i--){
7      words.remove(toDelete.get(i).intValue());
8  }
9  return;
10 }
11 public static void searchDiagDownRight(String[][] matrix, ArrayList<char> words, ArrayList<ArrayList<String>> finalPrint, ArrayList<Integer> finalComparison){
12     ArrayList<Integer> toDelete = new ArrayList<>();
13     ArrayList<String> toPrint = new ArrayList<>();
14     int matrixRow = matrix.length, matrixCol = matrix[0].length, comparison = 0;
15     for (int i = 0; i < words.size(); i++){
16         int wordCheck = 0;
17         boolean diff = true;
18         for (int j = matrixCol - 1; j >= 0; j--){
19             int row = 0;
20             int col = j;
21             while (col < matrixCol && row < matrixRow){
22                 comparison++;
23                 if (matrix[row][col].equals(getLetter(words, i, wordCheck))){
24                     if (diff == true){
25                         diff = false;
26                     }
27                     addCoordinate(toPrint, row, col);
28                     wordCheck++;
29                 }
30                 else{
31                     if (diff == false){
32                         diff = true;
33                         wordCheck = 0;
34                         toPrint.clear();
35                         comparison++;
36                     }
37                     if (matrix[row][col].equals(getLetter(words, i, 0))){
38                         diff = false;
39                         addCoordinate(toPrint, row, col);
40                         wordCheck++;
41                     }
42                 }
43             }
44             if (wordCheck == words.get(i).length){
45                 updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
46                 j = 1;
47                 break;
48             }
49             row++;
50             col++;
51         }
52         wordCheck = 0;
53         diff = true;
54         toPrint.clear();
55     }

```

```

1      for (int j = matrixRow - 1; j > 0; j--){
2          int col = 0;
3          int row = j;
4          while (col < matrixCol && row < matrixRow){
5              comparison++;
6              if (matrix[row][col].equals(getLetter(words, i, wordCheck)){
7                  if (diff == true){
8                      diff = false;
9                  }
10                 addCoordinate(toPrint, row, col);
11                 wordCheck++;
12             }
13             else{
14                 if (diff == false){
15                     diff = true;
16                     wordCheck = 0;
17                     toPrint.clear();
18                     comparison++;
19                     if (matrix[row][col].equals(getLetter(words, i, 0)){
20                         diff = false;
21                         addCoordinate(toPrint, row, col);
22                         wordCheck++;
23                     }
24                 }
25             }
26             if (wordCheck == words.get(i).length){
27                 updateProgress(toPrint, finalPrint, finalComparison, toDelete, i, comparison);
28                 j = 0;
29                 break;
30             }
31             row++;
32             col++;
33         }
34         wordCheck = 0;
35         diff = true;
36         toPrint.clear();
37     }
38 }
39 for (int i = toDelete.size() - 1; i >= 0; i--){
40     words.remove((toDelete.get(i)).intValue());
41 }
42 return;
43 }
44 public static void main(String[] args){
45     System.out.println("\n-----\nWORD SEARCH PUZZLE SOLVER\n-----");
46     System.out.print("Enter Filename\\ncfilename.txt: ");
47     Scanner sc = new Scanner(System.in);
48     String filename = "../test/" + sc.nextLine();
49     sc.close();
50     try{
51         ArrayList<ArrayList<String>> tempMatrix = new ArrayList<ArrayList<String>>();
52         ArrayList<char[]> words = new ArrayList<char[]>();
53         boolean isWord = false;
54     }

```

```

1      BufferedReader br = new BufferedReader(new FileReader(filename));
2      String rl = br.readLine();
3      while (rl != null){
4          if (rl.length() == 0){
5              isWord = true;
6          }
7          else{
8              String[] rls = rl.split(" ");
9              if (isWord == false){
10                 ArrayList<String> temp = new ArrayList<String>();
11                 for (String x : rls){
12                     temp.add(x);
13                 }
14                 tempMatrix.add(temp);
15             }
16             else{
17                 String all = "";
18                 for (String x : rls){
19                     all += x;
20                 }
21                 char[] ch = all.toCharArray();
22                 words.add(ch);
23             }
24             rl = br.readLine();
25         }
26     }
27     br.close();
28     String[][] matrix = tempMatrix.stream().map(u -> u.toArray(new String[0])).toArray(String[][]::new);
29     ArrayList<ArrayList<String>> finalPrint = new ArrayList<ArrayList<String>>();
30     ArrayList<Integer> finalComparison = new ArrayList<Integer>();
31     int searchWords = words.size();
32     long startTime = System.nanoTime();
33     searchUp(matrix, words, finalPrint, finalComparison);
34     searchDown(matrix, words, finalPrint, finalComparison);
35     searchLeft(matrix, words, finalPrint, finalComparison);
36     searchRight(matrix, words, finalPrint, finalComparison);
37     searchDiagUpLeft(matrix, words, finalPrint, finalComparison);
38     searchDiagUpRight(matrix, words, finalPrint, finalComparison);
39     searchDiagDownLeft(matrix, words, finalPrint, finalComparison);
40     searchDiagDownRight(matrix, words, finalPrint, finalComparison);
41     long endTime = System.nanoTime();
42     long totalTime = endTime - startTime;
43     double elapsedTimeInSeconds = (double) totalTime / 1_000_000_000;
44     printMatrix(matrix, finalPrint, finalComparison);
45     System.out.println("Matrix size: " + matrix.length + " x " + matrix[0].length);
46     System.out.println("Elapsed Time: " + elapsedTimeInSeconds + " second(s)");
47     if (words.size() == 0){
48         System.out.println("\nWords left: 0/" + searchWords);
49         System.out.println("All words were found during the process");
50     }
51     else{
52         System.out.println("\nWords left: " + words.size() + "/" + searchWords);
53         printWords(words);
54     }
55 }
56 catch (Exception ie){
57     System.out.println("gasabi");
58 }
59 }
60 }

```

BAB III

Screenshot Input/Output

No.	Screenshot Terminal	Keterangan
1.	<pre>PS C:\Users\Sarah Azka A\Desktop\if\sem-4\tucil1_13528883\bin> java WordSearch ----- WORD SEARCH PUZZLE SOLVER ----- Enter Filename <filename>.txt: small1.txt ----- ----- ----- ----- ----- ----- N ----- ----- I ----- ----- R ----- ----- D ----- ----- L ----- ----- A ----- ----- Total Comparison: 1799 ----- N ----- ----- I ----- ----- B ----- ----- O ----- ----- R ----- ----- ----- R ----- ----- E ----- ----- H ----- ----- C ----- ----- A ----- ----- E ----- ----- T ----- Total Comparison: 629 ----- ----- ----- ----- ----- N ----- ----- E ----- ----- S ----- ----- K ----- ----- I ----- ----- R ----- ----- E ----- ----- Total Comparison: 1385 ----- ----- ----- N ----- O ----- S ----- N ----- I ----- T ----- S ----- ----- ----- Y L I L ----- Total Comparison: 886</pre>	<p>Size: Small</p> <p>Matrix Size: 14 x 12</p> <p>Words to Search: 14</p> <p>Elapsed Time:</p> <p>0.0149997 second(s)</p> <p>Total Comparison of All Words: 10928</p>


```

PS C:\Users\Sarah Azka A\Desktop\if\sem-4\Tuc11_13520083\bin> java WordSearch
-----
WORD SEARCH PUZZLE SOLVER
-----
Enter Filename
<filename>.txt: small2.txt
-----
D
E
Z
I
R
O
M
E
M
-----
Total Comparison: 2158
-----
A
D
O
P
T
S
-----
Total Comparison: 176
-----
E
C
O
L
O
G
I
S
T
-----
Total Comparison: 1006
-----
E
L
O
N
G
A
T
E
-----
Total Comparison: 1066
-----
-----
K
I
N
D
L
E
-----
Total Comparison: 1453
-----
L
E
A
D
I
N
G
-----
Total Comparison: 1652
-----
P
O
R
E
S
-----
Total Comparison: 2096
-----
R
E
S
C
U
E
-----
Total Comparison: 2349
-----

```

Words: 18375

3.

WORD SEARCH PUZZLE SOLVER

Enter Filename

```
<filename>.txt: small13.txt
```

Total Comparison: 567

Total Comparison: 1621

Total Comparison: 3354

Total Comparison: 1564

Total Comparison: 2232

Total Comparison: 2507

Size: Small

Matrix Size: 18 x 16

Words to Search: 19

Elapsed Time:

0.0184588 second(s)

Total Comparison of All

Words: 28366

	<p> T R E M E N D O U S </p> <p>Total Comparison: 3832</p> <p> O O T A K C O C </p> <p>Total Comparison: 1120</p> <p> E V I T I N I F N I </p> <p>Total Comparison: 1780</p>	<p> L L U C S </p> <p>Total Comparison: 2455</p> <p> D E W E P S </p> <p>Total Comparison: 2565</p> <p> Y L L A U S U </p> <p>Total Comparison: 2806</p>	
--	---	--	--

[illegible]

Size: Medium
Matrix Size: 20 x 22
Words to Search: 24
Elapsed Time:
0.0265403 second(s)
Total Comparison of All
Words: 61995

5.

```
PS C:\Users\Sarah Azka A\Desktop\if\sem-4\Tuci
11_13520883\bin> java WordSearch
```

```
WORD SEARCH PUZZLE SOLVER
```

```
Enter Filename
<filename>.txt: medium2.txt
```

```
-----
- E -
- N -
- I -
- L -
- E -
- F -
-----
```

```
Total Comparison: 2260
```

```
----- R -----
- E -
- T -
- N -
- U -
- H -
-----
```

```
Total Comparison: 2588
```

```
-----
- A -
- F -
- R -
- I -
- C -
- A -
- N -
-----
```

```
Total Comparison: 426
```

```
Total Comparison: 426
```

```
----- C -----
- L -
- A -
- W -
- S -
-----
```

```
Total Comparison: 1689
```

```
C
A
R
N
I
V
O
R
E
-----
```

```
Total Comparison: 5737
```

```
----- S -----
- U -
- M -
- A -
- T -
- R -
- A -
- N -
-----
```

```
Total Comparison: 9837
```

Size: Medium

Matrix Size: 20 x 22

Words to Search: 24

Elapsed Time:

0.0236522 second(s)

Total Comparison of All

Words: 59688

6.

```
PS C:\Users\Sarah Azka A\Desktop\if\sem-4\Tucil1_13520083\b
in> java WordSearch
```

WORD SEARCH PUZZLE SOLVER

Enter Filename

```
<filename>.txt: medium3.txt
```

Total Comparison: 10949

Total Comparison: 860

Total Comparison: 3642

Total Comparison: 5788

Total Comparison: 6732

Size: Medium

Matrix Size: 20 x 22

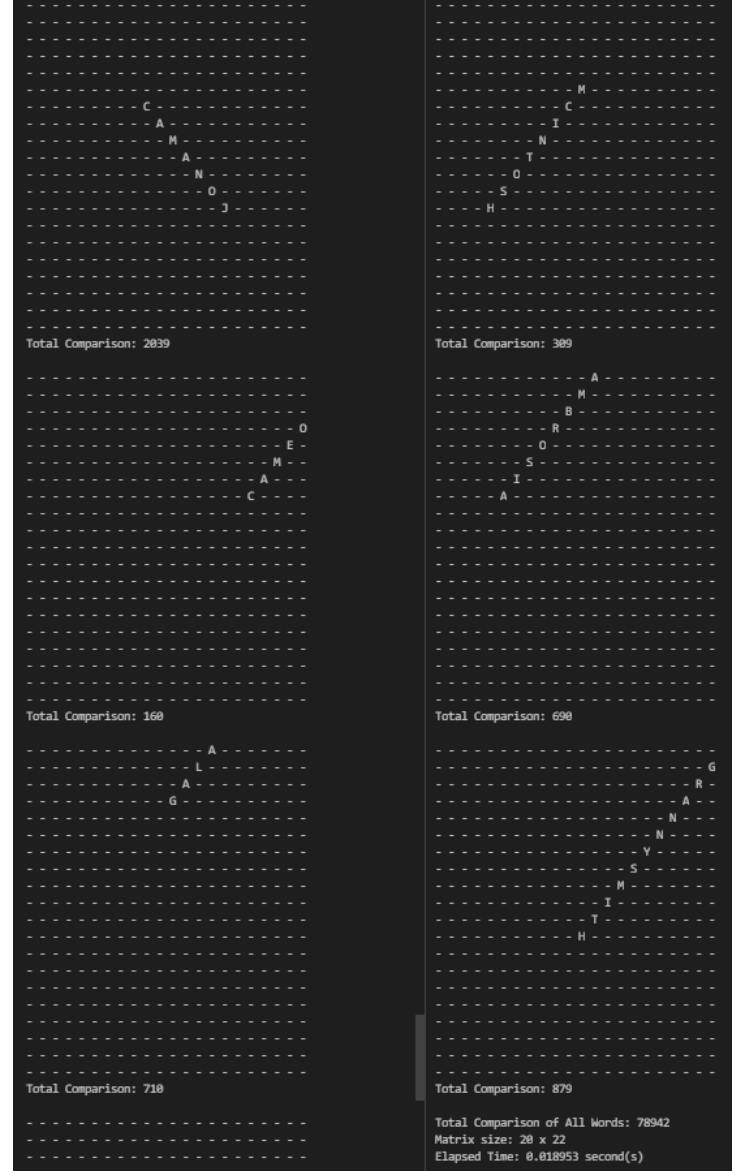
Words to Search: 24

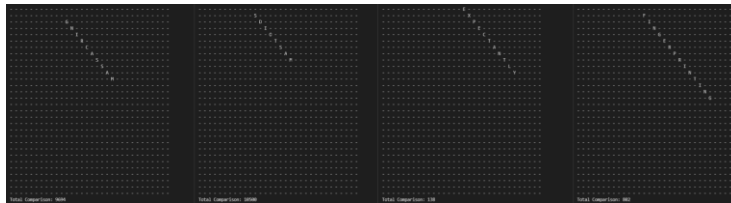
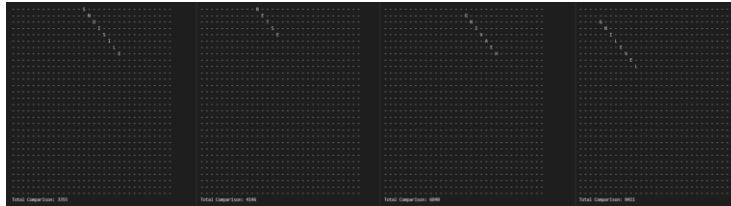
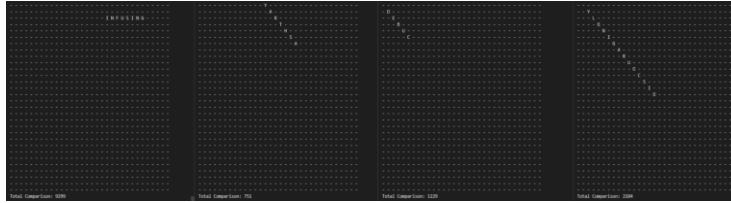
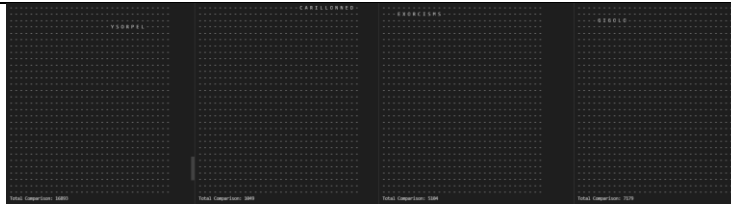
Elapsed Time:

0.018953 second(s)

Total Comparison of All

Words: 78942

<p>7.</p>	 <p>Size: Large Matrix Size: 30 x 32 Words to Search: 25 Elapsed Time: 0.0219198 second(s) Total Comparison of All Words: 187200</p>	<p>Size: Large Matrix Size: 30 x 32 Words to Search: 25 Elapsed Time: 0.0219198 second(s) Total Comparison of All Words: 187200</p>
-----------	---	---



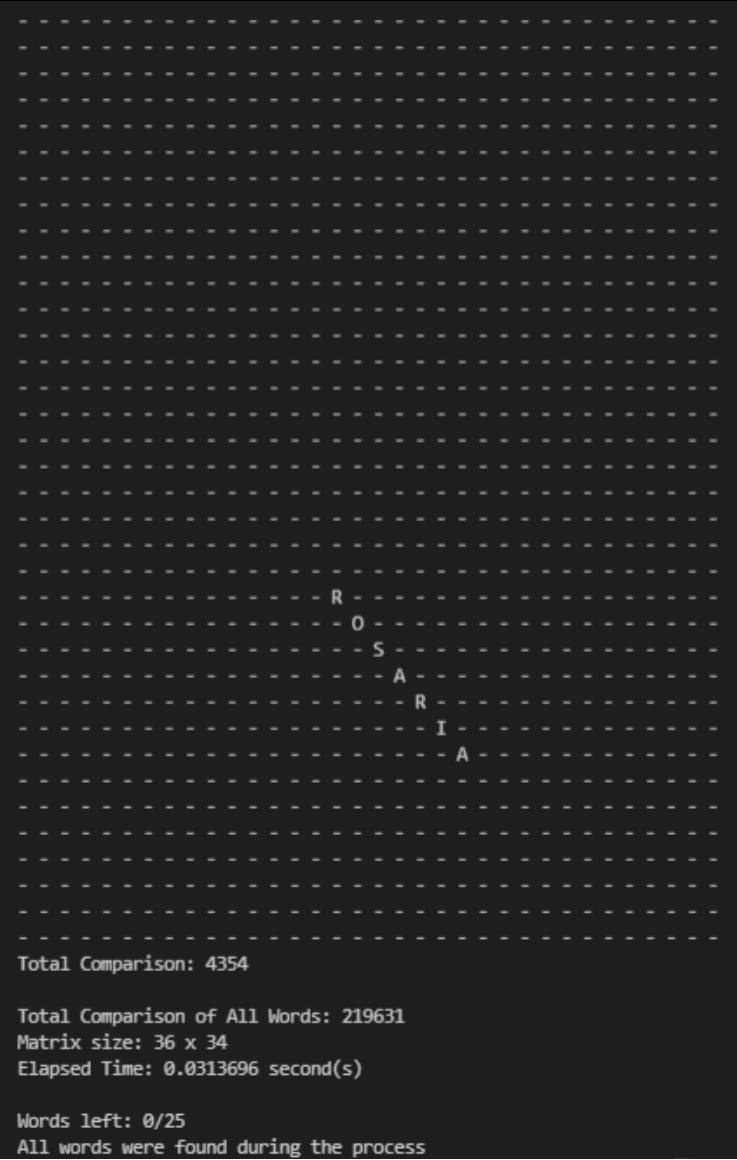
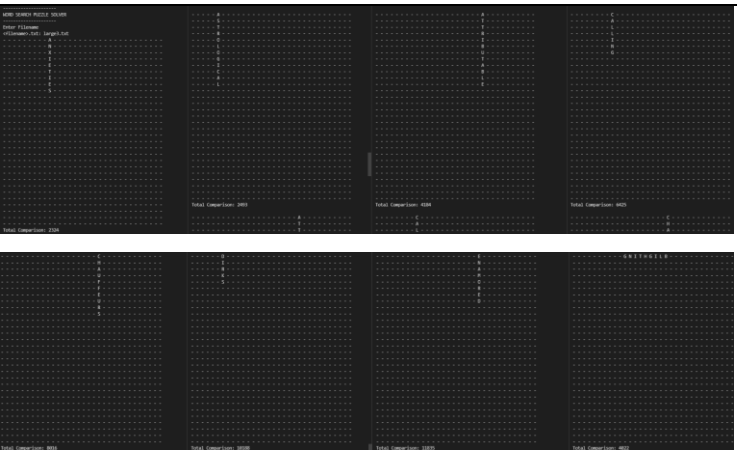
```
- - - - -
- I -
- N -
- D -
- U -
- S -
- T -
- R -
- I -
- A -
- L -
- I -
- S -
- T -
- S -
- - - - -

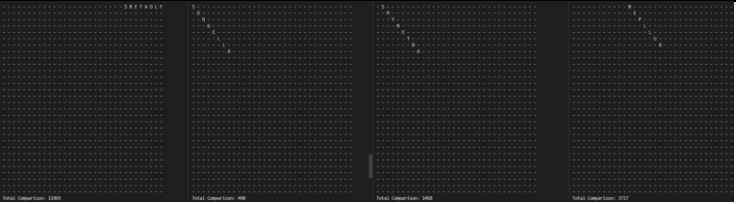
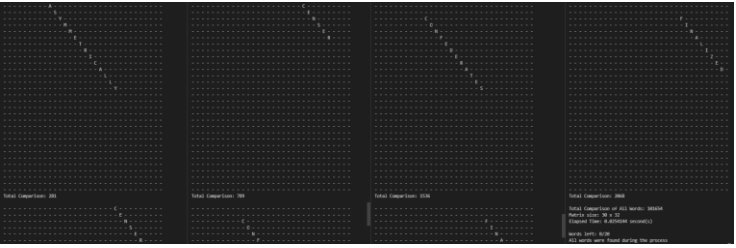
Total Comparison: 2262

Total Comparison of All Words: 187200
Matrix size: 30 x 32
Elapsed Time: 0.0219198 second(s)

Words left: 0/25
All words were found during the process
```

8.		<p>Size: Large</p> <p>Matrix Size: 36 x 34</p> <p>Words to Search: 25</p> <p>Elapsed Time:</p> <p>0.0313696 second(s)</p> <p>Total Comparison of All Words: 219631</p>

	 <p>Total Comparison: 4354</p> <p>Total Comparison of All Words: 219631</p> <p>Matrix size: 36 x 34</p> <p>Elapsed Time: 0.0313696 second(s)</p> <p>Words left: 0/25</p> <p>All words were found during the process</p>	
9.		<p>Size: Large</p> <p>Matrix Size: 30 x 32</p> <p>Words to Search: 20</p> <p>Elapsed Time:</p> <p>0.0254144 second(s)</p> <p>Total Comparison of All Words: 101654</p>

LAMPIRAN

Github Repository: [azkazkazka/word-search-puzzle-solver \(github.com\)](https://github.com/azkazkazka/word-search-puzzle-solver)

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.	✓	