STRATEGI ALGORITMA
TUGAS KECIL 1
IQ Puzzler Pro

I. Latar Belakang Tugas Kecil

Tugas yang diberikan adalah membuat algoritma brute force untuk mencari solusi dari sebuah *jigsaw puzzle*. Program menerima input terkait luas papan puzzle, total potongan jigsaw puzzle, dan potongan-potongan jigsaw puzzlenya. Lalu, program harus dapat menentukan apakah terdapat sebuah solusi atau tidak dari puzzle tersebut, berdasarkan informasi yang diberi. Apabila iya, program akan mencetak papan yang sudah terisi penuh dengan semua jigsaw puzzle yang diberi.

II. Algoritma Brute Force yang Digunakan

```
public static boolean IQPuzzlerPro(int puzzlePieceIdx) {
    if (puzzlePieceIdx == puzzlePieces.size()) return modules.Puzzle.isBoardFilled(board);
    char[][] puzzlePiece = puzzlePieces.get(puzzlePieceIdx);
    List<char[][]> pieceVariants = modules.Puzzle.getAllPieceVariations(puzzlePiece);
    for (char[][] uniquePiece : pieceVariants) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (modules.Puzzle.isPlacable(board, i, j, uniquePiece)) {
                    modules.Puzzle.placePiece(board, i, j, uniquePiece);
                    if (IQPuzzlerPro(puzzlePieceIdx + 1)) return true;
                    modules.Puzzle.removePiece(board, uniquePiece, i, j);
                    caseCtr++;
                }
            }
        }
    }
    return false;
}
```

Untuk menyelesaikan persoalan diatas, solusi dicari menggunakan fungsi rekursif bernama “backtracking” yang bersifat *brute force* dan *depth first search* (DSA). DSA merupakan algoritma rekursif yang fokus mencari kedalaman dari pohon te rlebih dahulu (pohon dalam konteks ini adalah pohon biner) sebelum melakukan *backtracking*.

Fungsi IQPuzzlerPro menerima parameter berupa puzzlePieceIdx yang merupakan index untuk list yang menyimpan semua potongan puzzle (semua potongan puzzle didapatkan dari fungsi preProcess pada file Main.java). Fungsi ini mengembalikan nilai boolean untuk menandakan apakah set puzzle (set: semua potongan puzzle bersama board puzzlenya) memiliki sebuah solusi.

Fungsi IQPuzzlerPro merupakan fungsi rekursif yang memanfaatkan backtracking. Fungsi IQPuzzlerPro berguna untuk mendapatkan semua kemungkinan dari peletakkan potongan puzzle pada papan.

Fungsi tersebut terdiri dari dua komponen, yaitu komponen basis dan komponen rekursif. Komponen basis merupakan batasan untuk rekursi agar fungsi dapat berhenti saat semua potongan telah di uji coba pada setiap kasus. Kasus basis dalam fungsi IQPuzzlerPro membatasi rekursi saat puzzlePieceIdx sudah sama nilainya dengan panjang list puzzlePieces (list yang menyimpan potongan-potongan puzzle dalam bentuk matrix). puzzlePieceIdx = puzzlePieces.size() menandakan puzzlePiece terakhir sudah ditempatkan pada board atau sudah di-cek semua kemungkinan peletakkannya.

Komponen rekursif fungsi tersebut dimulai dengan mengambil puzzlePiece ke-i dari list puzzlePieces. Lalu, program membuat sebuah list bernama pieceVariants yang dapat diisi dengan potongan puzzle dalam bentuk matrix. List pieceVariants bermanfaat untuk menyimpan semua variasi dari satu potong puzzle. Setiap potongan puzzle dapat di rotasi 90 derajat, atau direfleksi seperti sebuah cermin. Potongan puzzle yang digunakan akan diduplikasi (setelah didapatkan variannya) dan di-*insert* ke dalam list pieceVariants.

Lalu, program akan mengiterasi berdasarkan semua posisi yang terdapat pada papan puzzle untuk mencoba semua kemungkinan peletakkan. Setiap posisi akan diuji terlebih dahulu, apakah dapat ditempatkan oleh potongan puzzle tersebut. Apabila bisa, maka potongan puzzle tersebut akan ditempatkan pada papan berdasarkan posisi tersebut, lalu fungsi akan memanggil dirinya sendiri lagi dengan puzzlePieceIdx+1 untuk lanjutkan uji coba pada potongan puzzle berikutnya.

Apabila rekursif tersebut berhasil, maka fungsi akan mengembalikan nilai true. Apabila tidak, maka fungsi akan menghapus potongan puzzle tersebut dari papan puzzle, dan menambahkan hitungan kasus yang telah dicoba (caseCtr).

III. Source Program

1. Main.java

```java
import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import java.util.Scanner;

/* To do
 * 1. Extract pieces DONE
 * 2. isPlacable DONE
 * 3. Placing piece on board DONE
```

```
 * 4. Deleting piece from board
 * 5. Algorithmmmmmmmm
 */


public class Main {
    private static int caseCtr = 0;
    private static int totalPuzzlePieces;
    private static char[][] board;
    private static List<char[][]> puzzlePieces = new ArrayList<>();
    private static int rows;
    private static int cols;
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        preProcess(args);
        long startTime = System.currentTimeMillis();
        if (!IQPuzzlerPro(0)) {
            System.out.println("Gagal mendapatkan solusi");
            return;
        }
        long endTime = System.currentTimeMillis();
              System.out.println("Waktu pencarian: " + (endTime -
startTime) + " ms");
              System.out.println("Jumlah kasus yang ditinjau: " +
caseCtr);
        printColored();
        saveSolvedBoard();
    }

    public static void preProcess(String[] args) {
        System.out.print("Masukkan nama file: ");
        String filename = scanner.nextLine();
        String filepath = "test/input/" + filename;

        try {
                                          List<String> inputList =
Files.readAllLines(Paths.get(filepath));

            rows = getRow(inputList.get(0));
```

```
            cols = getCol(inputList.get(0));
            totalPuzzlePieces = getTotalPieces(inputList.get(0));

            board = getBoardConfig(inputList.get(1), rows, cols);

                        puzzlePieces  =  getPuzzlePieces(inputList,
totalPuzzlePieces);
        }
        catch (IOException e) {
                    System.out.println("Terjadi  kesalahan:  "  +
e.getMessage());
        }
    }

    // extract nums to help get total puzzle piece and board size
    public static int[] extractNums(String line) {
        String[] parts = line.split(" ");
        int[] nums = new int[parts.length];

        for (int i = 0; i < parts.length; i++) {
            nums[i] = Integer.parseInt(parts[i]);
        }
        return nums;
    }

    // get row length
    public static int getRow(String firstLine) {
        int[] nums = extractNums(firstLine);
        int row = nums[0];
        return row;
    }

    // get col length
    public static int getCol(String firstLine) {
        int[] nums = extractNums(firstLine);
        int col = nums[1];
        return col;
    }

    // get total pieces
```

```
    public static int getTotalPieces(String firstLine) {
        int[] nums = extractNums(firstLine);
        int totalPuzzlePieces = nums[2];
        return totalPuzzlePieces;
    }

    // get config
     public static char[][] getBoardConfig(String config, int row,
int col) {
        if (config.equals("DEFAULT")) {
               char[][] board = modules.Puzzle.createCharMatrix(row,
col);
            return modules.Puzzle.fillBoardWithDot(board);
        } else {
            System.out.println("Error loading config: " + config);
            return new char[row][col];
        }
    }

    // string to arr of char
    public static char[] strToChar(String str) {
        return str.toCharArray();
    }

    // get first nonSpaceCharacter
    public static char getFirstNonSpaceCharacter(char[] line) {
        for (char c : line) {
            if (c != ' ') {
                return c;
            }
        }
        return ' ';
    }

    // extract puzzle pieces
        public  static  List<char[][]>  getPuzzlePieces(List<String>
inputList, int totalPuzzlePieces) {
        List<char[][]> puzzlePieces = new ArrayList<>();
        int strIdx = 2;
```

```
        while (strIdx < inputList.size()) {
            char[] currString = strToChar(inputList.get(strIdx));
            char currChar = getFirstNonSpaceCharacter(currString);
            char currBlock = currChar;
            int maxCol = 0;

            int tempIdx = strIdx;
                            while (tempIdx < inputList.size() &&
getFirstNonSpaceCharacter(strToChar(inputList.get(tempIdx))) ==
currBlock) {
                                        maxCol = Math.max(maxCol,
inputList.get(tempIdx).length());
                tempIdx++;
            }

                                    char[][] currPuzzlePiece =
modules.Puzzle.createCharMatrix(1, maxCol);
            int currRow = 0;

                            while (strIdx < inputList.size() &&
getFirstNonSpaceCharacter(strToChar(inputList.get(strIdx))) ==
currBlock) {
                if (currRow > 0) {
                                                currPuzzlePiece =
modules.Puzzle.addRows(currPuzzlePiece, 1);
                }

                currString = strToChar(inputList.get(strIdx));

                if (currString.length > maxCol) {
                                                currPuzzlePiece =
modules.Puzzle.addCols(currPuzzlePiece, currString.length);
                    maxCol = currString.length;
                }

                for (int currCol = 0; currCol < maxCol; currCol++) {
                        char ch = (currCol < currString.length) ?
currString[currCol] : '.';
                    if (ch == ' ') ch = '.';
```

```
                                                    currPuzzlePiece =
modules.Puzzle.insCharToMatrix(currPuzzlePiece, currRow, currCol,
ch);
                }

                currRow++;
                strIdx++;

                if (strIdx < inputList.size()) {
                    currString = strToChar(inputList.get(strIdx));
                                                            currChar =
getFirstNonSpaceCharacter(currString);
                }
            }

            puzzlePieces.add(currPuzzlePiece);

            if (strIdx < inputList.size()) {
                                                        currBlock =
getFirstNonSpaceCharacter(strToChar(inputList.get(strIdx)));
            }
        }
        return puzzlePieces;
    }

    // IQ Puzzle Solver YESYESYES
    public static boolean IQPuzzlerPro(int puzzlePieceIdx) {
                if (puzzlePieceIdx == puzzlePieces.size()) return
modules.Puzzle.isBoardFilled(board);
        char[][] puzzlePiece = puzzlePieces.get(puzzlePieceIdx);
                                    List<char[][]> pieceVariants =
modules.Puzzle.getAllPieceVariations(puzzlePiece);
        for (char[][] uniquePiece : pieceVariants) {
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                            if (modules.Puzzle.isPlacable(board, i, j,
uniquePiece)) {
                                modules.Puzzle.placePiece(board, i, j,
uniquePiece);
```

```
                        if (IQPuzzlerPro(puzzlePieceIdx + 1)) return
true;
                                    modules.Puzzle.removePiece(board,
uniquePiece, i, j);
                        caseCtr++;
                    }
                }
            }
        }
        return false;
    }

    // print colored board
    public static void printColored() {
        final String RESET = "\u001B[0m";
        final String[] COLORS = {
             "\u001B[31m", "\u001B[32m", "\u001B[33m", "\u001B[34m",
"\u001B[35m", "\u001B[36m",
             "\u001B[91m", "\u001B[92m", "\u001B[93m", "\u001B[94m",
"\u001B[95m", "\u001B[96m",
             "\u001B[97m", "\u001B[90m", "\u001B[41m", "\u001B[42m",
"\u001B[43m", "\u001B[44m",
                        "\u001B[45m",  "\u001B[46m",  "\u001B[101m",
"\u001B[102m", "\u001B[103m", "\u001B[104m",
            "\u001B[105m", "\u001B[106m"
        };

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char piece = board[i][j];
                String color = RESET;

                if (piece >= 'A' && piece <= 'Z') {
                    color = COLORS[piece - 'A'];
                }

                System.out.print(color + piece + " " + RESET);
            }
            System.out.println();
        }
```

```
    }

    // Save solution in a txt file in the output folder
    public static void saveSolvedBoard() {
             System.out.print("Apakah anda ingin menyimpan solusi?
(ya/tidak): ");
        String response = scanner.nextLine().trim().toLowerCase();

        if (!response.equals("ya")) {
            System.out.println("Solusi tidak disimpan.");
            return;
        }

        String outputPath = "test/output/solusi.txt";
        try {
            Files.createDirectories(Paths.get("output"));
                    BufferedWriter writer = new BufferedWriter(new
FileWriter(outputPath));

            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    writer.write(board[i][j] + " ");
                }
                writer.newLine();
            }

            writer.close();
               System.out.println("Solusi berhasil disimpan di: " +
outputPath);
        } catch (IOException e) {
                   System.out.println("Gagal menyimpan solusi: " +
e.getMessage());
        }
    }
}
```

2. Puzzle.Java

```
package modules;
```

```
import java.util.Arrays;
import java.util.LinkedHashSet;
import java.util.Random;
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;

public class Puzzle {
    // Driver
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        char[][] puzzle = null;

        while (true) {
            System.out.println("\n==== PUZZLE MATRIX MENU ====");
            System.out.println("1. Create Puzzle");
            System.out.println("2. Rotate 90 Degrees");
            System.out.println("3. Mirror Matrix");
            System.out.println("4. Add Rows");
            System.out.println("5. Add Columns");
            System.out.println("6. Insert Character");
            System.out.println("7. Fill with Random Chars");
            System.out.println("8. Copy Matrix");
            System.out.println("9. Print Matrix");
            System.out.println("0. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    puzzle = createMatrix(scanner);
                    break;
                case 2:
                                    if (puzzle != null) puzzle =
Puzzle.rotate90(puzzle);
                            else System.out.println("Create a matrix
first!");
                    break;
                case 3:
```

```
                                    if (puzzle != null) puzzle =
Puzzle.mirror(puzzle);
                                else System.out.println("Create a matrix
first!");
                    break;
                case 4:
                    if (puzzle != null) {
                            System.out.print("Enter number of rows to
add: ");
                        int rows = scanner.nextInt();
                        puzzle = Puzzle.addRows(puzzle, rows);
                          } else System.out.println("Create a matrix
first!");
                    break;
                case 5:
                    if (puzzle != null) {
                         System.out.print("Enter number of columns to
add: ");
                        int cols = scanner.nextInt();
                        puzzle = Puzzle.addCols(puzzle, cols);
                          } else System.out.println("Create a matrix
first!");
                    break;
                case 6:
                    if (puzzle != null) {
                        System.out.print("Enter row index: ");
                        int row = scanner.nextInt();
                        System.out.print("Enter column index: ");
                        int col = scanner.nextInt();
                        System.out.print("Enter character: ");
                        char ch = scanner.next().charAt(0);
                         puzzle = Puzzle.insCharToMatrix(puzzle, row,
col, ch);
                          } else System.out.println("Create a matrix
first!");
                    break;
                case 7:
                                            if (puzzle != null)
Puzzle.fillBoardWithRandomChars(puzzle);
```

```
                        else System.out.println("Create a matrix
first!");
                    break;
                case 8:
                    if (puzzle != null) {
                                        char[][] copiedMatrix = new
char[puzzle.length][puzzle[0].length];
                        Puzzle.copyMatrix(puzzle, copiedMatrix);
                        System.out.println("Copied Matrix:");
                        Puzzle.printPuzzle(copiedMatrix);
                          } else System.out.println("Create a matrix
first!");
                    break;
                case 9:
                    if (puzzle != null) {
                        System.out.println("Current Matrix:");
                        Puzzle.printPuzzle(puzzle);
                            } else System.out.println("No matrix to
display!");
                    break;
                case 0:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                             System.out.println("Invalid choice, try
again.");
            }
        }
    }

    // Helper function to create a matrix
    public static char[][] createMatrix(Scanner scanner) {
        System.out.print("Enter number of rows: ");
        int rows = scanner.nextInt();
        System.out.print("Enter number of columns: ");
        int cols = scanner.nextInt();

        char[][] matrix = new char[rows][cols];
        System.out.println("Enter matrix elements row by row:");
```

```
        for (int i = 0; i < rows; i++) {
            String line = scanner.next();
             for (int j = 0; j < Math.min(line.length(), cols); j++)
{
                matrix[i][j] = line.charAt(j);
            }
        }
        return matrix;
    }

    // KERJAAN GW //

    // 1. Rotate puzzle
    public static char[][] rotate90(char[][] puzzle) {
        int row = puzzle.length;
        int col = puzzle[0].length;
        char[][] rotated = new char[col][row];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                rotated[j][row - 1 - i] = puzzle[i][j];
            }
        }
        return rotated;
    }

    // 2. Mirror puzzle
    public static char[][] mirror(char[][] puzzle) {
        int row = puzzle.length;
        int col = puzzle[0].length;
        char[][] mirrored = new char[row][col];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                mirrored[i][j] = puzzle[i][col - 1 - j];
            }
        }
        return mirrored;
    }
```

```
    // 3. Print puzzle
    public static void printPuzzle(char[][] puzzle) {
        for (char[] row : puzzle) {
            System.out.println(Arrays.toString(row));
        }
        System.out.println();
    }

    // 4. Create matrix
    public static char[][] createCharMatrix(int N, int M) {
        char[][] matrix = new char[N][M];
        return matrix;
    }

    public static void fillBoardWithRandomChars(char[][] board) {
        Random rand = new Random();
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                  board[i][j] = (char) ('A' + rand.nextInt(26)); //
Huruf A-Z
            }
        }
    }

    // 5. Print board
    public static void printBoard(char[][] board) {
        for (char[] row : board) {
            for (char cell : row) {
                System.out.print(cell + " ");
            }
            System.out.println();
        }
    }

    // 6. Insert element to matrix
     public static char[][] insCharToMatrix(char[][] matrix, int row,
int col, char character) {
          if (row < 0 || row >= matrix.length || col < 0 || col >=
matrix[0].length) {
```

```
            System.out.println("Error: Cannot insert, index out of
range");
            return matrix;
        }
        matrix[row][col] = character;
        return matrix;
    }

    // 7. Add Row to Matrix
    public static char[][] addRows(char[][] matrix, int addedRows) {
        if (matrix.length == 0) { // handle case matrix.length = 0
            return new char[addedRows][0];
        }
                    char[][]  newMatrix  =  new  char[addedRows  +
matrix.length][matrix[0].length];
         copyMatrix(matrix, newMatrix); // save the value of the old
matrix
        return newMatrix; // new matrix with added rows
    }

    // 8. Add Col to Matrix
    public static char[][] addCols(char[][] matrix, int addedCols) {
        if (matrix.length == 0) {
            return new char[0][addedCols];
        }

                                    char[][]    newMatrix    =    new
char[matrix.length][matrix[0].length + addedCols];
        copyMatrix(matrix, newMatrix);
        return newMatrix;
    }

    // 9. Copy Matrix
       public  static  void  copyMatrix  (char[][]  matrix,  char[][]
copyMatrix) {
         if (matrix.length > copyMatrix.length || matrix[0].length >
copyMatrix[0].length) {
            System.out.println("Error: Cannot copy, index out of
range");
            return;
```

```
        }
        for (int i = 0 ; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                copyMatrix[i][j] = matrix[i][j];
            }
        }
    }

    // 10. Check if puzzle piece is placable on the board
     public static boolean isPlacable(char[][] board, int currRow,
int currCol, char[][] puzzlePiece) {
        if ((currRow + puzzlePiece.length) > board.length) {
            return false;
        }

        for (int i = 0; i < puzzlePiece.length; i++) {
             if ((currCol + puzzlePiece[i].length > board[0].length))
return false;

            for (int j = 0; j < puzzlePiece[i].length; j++) {
                     if (board[currRow + i][currCol + j] != '.' &&
puzzlePiece[i][j] != '.') {
                    return false;
                }
            }
        }
        return true;
    }

    // 11. Place puzzle piece on board
     public static char[][] placePiece(char[][] board, int currRow,
int currCol, char[][] puzzlePiece) { // make sure udh di check
placable dulu
        for (int i = 0; i < puzzlePiece.length; i++) {
            for (int j = 0; j < puzzlePiece[0].length; j++) {
                if (puzzlePiece[i][j] != '.') {
                                board[currRow + i][currCol + j] =
puzzlePiece[i][j];
                }
            }
```

```
        }
        return board;
    }

    // 12. Get all puzzle variations
       public  static  List<char[][]>  getAllPieceVariations(char[][]
puzzlePiece) {
        List<char[][]> pieceVariations = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            pieceVariations.add(puzzlePiece);
            puzzlePiece = rotate90(puzzlePiece);
        }
        puzzlePiece = mirror(puzzlePiece);
        for (int i = 0; i < 4; i++) {
            pieceVariations.add(puzzlePiece);
            puzzlePiece = rotate90(puzzlePiece);
        }
        pieceVariations = removeDuplicateMatrices(pieceVariations);
        return pieceVariations;
    }

    // 13. fill puzzle board with '.'
    public static char[][] fillBoardWithDot(char[][] emptyBoard) {
        int row = emptyBoard.length;
        int col = emptyBoard[0].length;
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                emptyBoard[i][j] = '.';
            }
        }
        return emptyBoard;
    }

    // 14. Check is board filled
    public static boolean isBoardFilled(char[][] board) {
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (board[i][j] == '.') return false;
            }
        }
```

```
        return true;
    }

    // 15. isPuzzlePieceSame
        public  static  boolean  isSamePuzzlePiece(char[][]  piece1,
char[][] piece2) {
        if ((piece1.length != piece2.length) || (piece1[0].length !=
piece2[0].length)) return false; // check dimensions
        for (int i = 0; i < piece1.length; i++) {
            for (int j = 0; j < piece1[0].length; j++) {
                if (piece1[i][j] != piece2[i][j]) return false;
            }
        }
        return true;
    }

    // 16. Remove Duplicates from matrix list
                            public       static       List<char[][]>
removeDuplicateMatrices(List<char[][]> matrixList) {
        List<char[][]> uniqueMatrices = new ArrayList<>();

        for (char[][] matrix : matrixList) {
            boolean isDuplicate = false;

            for (char[][] uniqueMatrix : uniqueMatrices) {
                if (isSamePuzzlePiece(matrix, uniqueMatrix)) {
                    isDuplicate = true;
                    break;
                }
            }

            if (!isDuplicate) {
                uniqueMatrices.add(matrix);
            }
        }
        return uniqueMatrices;
    }

    // 17. Remove piece from board
```

```
        public static void removePiece(char[][] board, char[][]
puzzlePiece, int row, int col) {
        for (int i = 0; i < puzzlePiece.length; i++) {
            for (int j = 0 ; j < puzzlePiece[0].length; j++) {
                if (puzzlePiece[i][j] != '.') {
                    board[i + row][j + col] = '.';
                }
            }
        }
    }
}
```

IV. Test Case (1-7)

1. tc1.txt:

   5 5 8
   DEFAULT
   A
   AA
   B
   BB
   C
   CC
   D
   DD
   EE
   EE
   E
   FF
   FF
   F
   GGG

   output:

```
Masukkan nama file: tc1.txt
Waktu pencarian: 63 ms
Jumlah kasus yang ditinjau: 7386
A G G G D
A A B D D
C C B B E
C F F E E
F F F E E
Apakah anda ingin menyimpan solusi? (ya/tidak):
```

2. tc2.txt:
   4 4 4
   DEFAULT
   AA
   AA
   BB
   BB
   CC
   CC
   DD
   DD

   output:

```
PS C:\Users\Mahesa\OneDrive\ITB\Coding\College\Acade
e\Academic\IF\Smt-4\Strategi Algoritma\Tucil\Tucil 1
nExceptionMessages' '-cp' 'C:\Users\Mahesa\AppData\R
' 'Main'
Masukkan nama file: tc2.txt
Waktu pencarian: 0 ms
Jumlah kasus yang ditinjau: 0
A A B B
A A B B
C C D D
C C D D
Apakah anda ingin menyimpan solusi? (ya/tidak):
```

3. tc3.txt:
   5 5 7
   DEFAULT

AA
 A
G
G
G
 D
DD
CC
C
FF
FFF
EEE
EE
BB
B

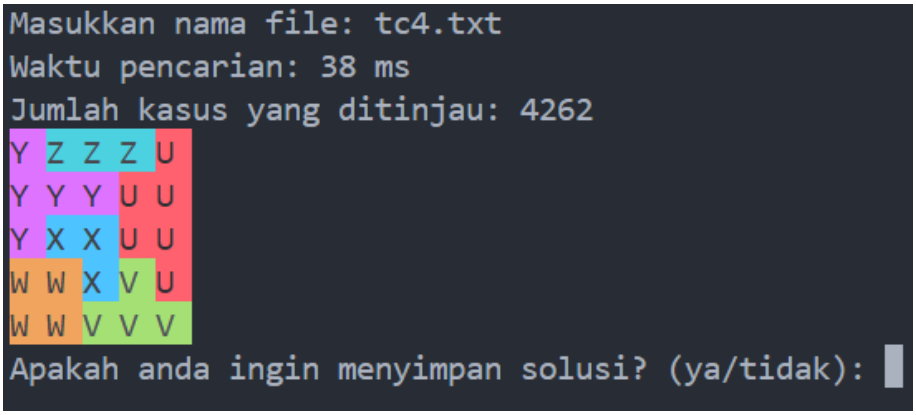
output:

```
Masukkan nama file: tc3.txt
Waktu pencarian: 13 ms
Jumlah kasus yang ditinjau: 382
A A G C C
B A G C D
B B G D D
F F E E E
F F F E E
Apakah anda ingin menyimpan solusi? (ya/tidak):
```

4. tc4.txt:
   5 5 5
   DEFAULT
   ZZZ
   XX
    X
    Y
    Y
   YYY
   WW
   WW
   VVV
    V
   U
   UU
   UU

U

output:



5. tc5.txt:
   4 7 8
   DEFAULT
   AA
   A
   BB
   B
   CCC
    C
   D
   DD
    D
    EE
   EE
   FFF
    F
   GGGG
   H
   H

   output:

```
Masukkan nama file: tc5.txt
Waktu pencarian: 1 ms
Jumlah kasus yang ditinjau: 38
A A B B C C C
A F B D E E C
F F F D D E E
G G G G D H H
```

6. tc6.txt:
   6 4 8
   DEFAULT
   AA
   A
   BB
   B
   CCC
    D
   DD
   FF
    H
   HH
    H
    G
   GG
   E
   EE

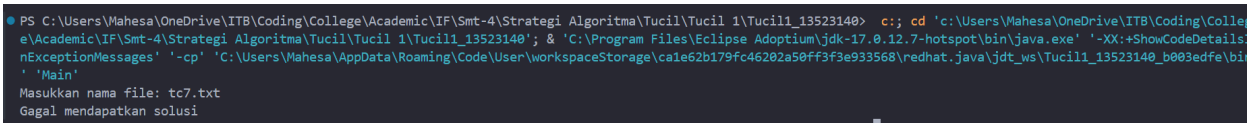
   output:

```
Masukkan nama file: tc6.txt
Waktu pencarian: 18 ms
Jumlah kasus yang ditinjau: 1670
A A B B
A D B H
D D H H
C C C H
E F F G
E E G G
Apakah anda ingin menyimpan solusi? (ya/tidak):
```

7. tc7.txt:

10 10 5
DEFAULT
AAAAAAAAA
BBBBBBBBB
BBBBBBBBB
CCC
CCCCCCCCC
CCCCCCCCC
DDDDDDDDD
DD
EE

output:



V. Pranala repository

https://github.com/mahesa005/Tucil1_13523140