

TUGAS KECIL 1 IF2211
Strategi Algoritma



Diampu oleh:

Dr. Ir. Rinaldi Munir, M.T.

Disusun oleh:

Sabilul Huda (13523072)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

APLIKASI BRUTE FORCE DALAM PROGRAM

Dalam pembuatan program ini, digunakan algoritma brute force sebagai algoritma utama. Sebagai ringkasan, berikut adalah persoalan yang dijawab pada program ini. Diberikan integer M, N, P, string mode, dan P buah *piece*. *Pieces* tersebut akan disusun pada suatu papan berukuran MxN. Dalam penyelesaian masalah tersebut, diterapkan algoritma brute force sebagai berikut:

1. Seluruh *pieces* diubah dalam bentuk *list of coords* yang tiap *coord* menyimpan informasi koordinat blok yang terisi pada *piece* tersebut.
2. Setiap *piece* ditransformasikan menggunakan rotasi 0 derajat, rotasi 90 derajat, rotasi 180 derajat, rotasi 270 derajat, dan pencerminan terhadap sumbu x untuk keempat hasil transformasi tersebut.
3. Hasil transformasi tiap *piece* yang diberikan disimpan dalam List of List of *allTransforms*
4. *Piece* pertama diletakkan pada puzzle.
5. *Piece* selanjutnya diletakkan pada puzzle, jika tidak bisa, digunakan hasil transformasi lain dari *piece* tersebut yang sudah tersimpan pada *allTransforms*,
6. Jika semua transformasi sudah dicoba, *piece* tersebut akan dikeluarkan dari board dan diganti dengan *Piece* selanjutnya.
7. Ulangi langkah 5-6 hingga ditemukan solusi atau seluruh kemungkinan telah dicoba.

SOURCE CODE

Berikut adalah source code yang digunakan dalam bahasa java:

```
import java.io.*;

import java.util.*;

class RecordState{

    public List<char[][]> boardRecord;

    public char[] pieceRecord;

    public int[] transformIdxRecord;

    public List<Pair<Integer, Integer>> firstEmptyRecord;

    RecordState(List<char[][]> boardRecord, char[] pieceRecord, int[]
transformIdxRecord, List<Pair<Integer, Integer>> firstEmptyRecord){

        this.boardRecord = boardRecord;
```

```

        this.pieceRecord = pieceRecord;

        this.transformIdxRecord = transformIdxRecord;

        this.firstEmptyRecord = firstEmptyRecord;
    }
}

class Pair<X, Y> {

    public final X x;

    public final Y y;

    public Pair(X x, Y y) {

        this.x = x;

        this.y = y;

    }

};

class Piece {

    public final char c;

    public Pair<Integer,Integer> firstPosition;

    public List<Pair<Integer, Integer>> coords;

    public char[][] pairInMatrix;

    public Piece(char c, List<Pair<Integer, Integer>> coords) {

        this.c = c;

        int X = coords.get(0).x;

        int Y = coords.get(0).y;

        for (Pair<Integer, Integer> coord : coords) {

            if (coord.x < X) {

                X = coord.x;

```

```

    }

    if (coord.y < Y) {

        Y = coord.y;

    }

}

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

    coords.set(i, new Pair<>(coords.get(i).x - X, coords.get(i).y -

Y));

}

this.coords = coords;

this.pairInMatrix = pieceToMatrix(coords, c);

char[][] matrix = this.pairInMatrix;

int xMin = 0;

while (matrix[xMin][0] == ' '){

    xMin ++;

}

this.firstPosition = new Pair<>(0, xMin);

};

public char[][] pieceToMatrix(List<Pair<Integer, Integer>> coords, char c){

    int numRows = 0;

    int numCol = 0;

    for (Pair<Integer, Integer> coord : coords){

        if (coord.x > numRows){

            numRows = coord.x;

        }

    }

```

```

        if (coord.y > numCol){

            numCol = coord.y;

        }

    }

    char[][] matrix = new char[numRow + 1][numCol + 1];

    for (int i = 0; i < matrix.length; i++) {

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

            matrix[i][j] = ' ';

        }

    }

    for (Pair<Integer, Integer> coord : coords) {

        matrix[coord.x][coord.y] = c;

    }

    return matrix;

}

public static Piece Rotate(Piece x) {

    List<Pair<Integer, Integer>> newCoords = new ArrayList<>();

    for (Pair<Integer, Integer> coord : x.coords) {

        newCoords.add(new Pair<>(-coord.y, coord.x));

    }

    return new Piece(x.c, newCoords);

};

public static Piece ReflectX(Piece x) {

    List<Pair<Integer, Integer>> newCoords = new ArrayList<>();

```

```
for (Pair<Integer, Integer> coord : x.coords) {  
    newCoords.add(new Pair<>(coord.x, -coord.y));  
}  
  
return new Piece(x.c, newCoords);  
};
```

```
public static List<Piece> AllTransforms(Piece x) {  
    List<Piece> transforms = new ArrayList<>();  
  
    Piece rotated0 = x;  
  
    Piece rotated90 = Rotate(rotated0);  
  
    Piece rotated180 = Rotate(rotated90);  
  
    Piece rotated270 = Rotate(rotated180);  
  
    Piece reflectedX = ReflectX(x);  
  
    Piece reflectedX90 = Rotate(reflectedX);  
  
    Piece reflectedX180 = Rotate(reflectedX90);  
  
    Piece reflectedX270 = Rotate(reflectedX180);  
  
    if (!isPieceIn(transforms, rotated0)) {  
        transforms.add(rotated0);  
    }  
  
    if (!isPieceIn(transforms, rotated90)) {  
        transforms.add(rotated90);  
    }  
  
    if (!isPieceIn(transforms, rotated180)) {  
        transforms.add(rotated180);  
    }  
  
    if (!isPieceIn(transforms, rotated270)) {  
        transforms.add(rotated270);  
    }  
}
```

```

    }

    if (!isPieceIn(transforms, reflectedX)) {

        transforms.add(reflectedX);

    }

    if (!isPieceIn(transforms, reflectedX90)) {

        transforms.add(reflectedX90);

    }

    if (!isPieceIn(transforms, reflectedX180)) {

        transforms.add(reflectedX180);

    }

    if (!isPieceIn(transforms, reflectedX270)) {

        transforms.add(reflectedX270);

    }

    return transforms;

};

public static void printPieceKoordinat(Piece x) {

    for (Pair<Integer, Integer> coord : x.coords) {

        System.out.print("(" + coord.x + ", " + coord.y + ")" + ", ");

    }

    System.out.println();

};

public static boolean isKoordInPiece(Piece p, Pair<Integer, Integer> koord)
{

    for (Pair<Integer, Integer> coord : p.coords) {

        if (coord.x == koord.x && coord.y == koord.y) {

            return true;

        }

    }

}

```

```

        }

    }

    return false;

};

public static boolean equalsPiece(Piece x, Piece y) {

    if(x.firstPosition.x != y.firstPosition.x || x.pairInMatrix.length !=
y.pairInMatrix.length || x.pairInMatrix[0].length != y.pairInMatrix[0].length){

        return false;

    }

    for (Pair<Integer, Integer> coord : x.coords) {

        if (!isKoordInPiece(y, coord)) {

            return false;

        }

    }

    return true;

};

public static boolean isPieceIn(List<Piece> pieces, Piece x) {

    for (Piece piece : pieces) {

        if (equalsPiece(piece, x)) {

            return true;

        }

    }

    return false;

};

public static void printPiece(Piece x) {

```



```

        for (int i = 0; i < x.pairInMatrix.length; i++) {
            for (int j = 0; j < x.pairInMatrix[i].length; j++) {
                System.out.print(x.pairInMatrix[i][j] + " ");
            }
            System.out.println();
        }
    };
}

class Board {
    public char[][] board;
    public Pair<Integer, Integer> firstEmpty;
    public boolean isNormalBoard;

    public Board(char[][] board, Pair<Integer, Integer> firstEmpty, boolean
isNormal){
        this.board = board;
        this.firstEmpty = firstEmpty;
        this.isNormalBoard = isNormal;
    }
}

public class IQPuzzleSolver {
    private static int M, N, P;
    private static String mode = "Default";
    private static Board board;
    private static List<Piece> pieces;
    private static List<List<Piece>> allTransforms;
    private static int[] pieceOrder;

```

```
private static List<Integer> unusedPiece = new ArrayList<>();

private static int tryCounter;

private static boolean solved;

public static final String RESET = "\u001B[0m";

public static final String[] COLORS = {

    "\u001B[31m", // A - Merah

    "\u001B[32m", // B - Hijau

    "\u001B[33m", // C - Kuning

    "\u001B[34m", // D - Biru

    "\u001B[35m", // E - Ungu

    "\u001B[36m", // F - Cyan

    "\u001B[91m", // G - Merah Terang

    "\u001B[92m", // H - Hijau Terang

    "\u001B[93m", // I - Kuning Terang

    "\u001B[94m", // J - Biru Terang

    "\u001B[95m", // K - Ungu Terang

    "\u001B[96m", // L - Cyan Terang

    "\u001B[97m", // M - Putih Terang

    "\u001B[90m", // N - Abu-abu Gelap

    "\u001B[41m", // O - Latar Merah

    "\u001B[42m", // P - Latar Hijau

    "\u001B[43m", // Q - Latar Kuning

    "\u001B[44m", // R - Latar Biru

    "\u001B[45m", // S - Latar Ungu

    "\u001B[46m", // T - Latar Cyan

    "\u001B[100m", // U - Latar Abu-abu Gelap
```

```

        "\u001B[101m", // V - Latar Merah Terang

        "\u001B[102m", // W - Latar Hijau Terang

        "\u001B[103m", // X - Latar Kuning Terang

        "\u001B[104m", // Y - Latar Biru Terang

        "\u001B[105m" // Z - Latar Ungu Terang

    };

    public static String getColor(char c) {

        if (c >= 'A' && c <= 'Z') {

            return COLORS[c - 'A'];

        }

        return RESET;

    }

    private static Board initiateBoard(int M, int N){

        char[][] board = new char[M][N];

        for (int i = 0; i < M; i++) {

            for (int j = 0; j < N; j++) {

                board[i][j] = ' ';

            }

        }

        return new Board(board, new Pair<>(0, 0), true);

    };

    private static Board putPiece(Board board, Piece piece){

```

```

char[][] newBoard = board.board;

char[][] copyBoard = new char[newBoard.length][newBoard[0].length];

for (int i = 0; i < newBoard.length; i++) {
    for (int j = 0; j < newBoard[i].length; j++) {
        copyBoard[i][j] = newBoard[i][j];
    }
}

Pair<Integer, Integer> firstEmpty = board.firstEmpty;
Pair<Integer, Integer> firstPosisiton = piece.firstPosition;

boolean isNormal = true;

int diffRow = firstEmpty.x - firstPosisiton.x;
int diffCol = firstEmpty.y - firstPosisiton.y;

for (Pair<Integer, Integer> coord : piece.coords) {
    if (coord.x + diffRow < 0 || coord.x + diffRow >= newBoard.length
    || coord.y + diffCol < 0 || coord.y + diffCol >= newBoard[0].length){
        isNormal = false;
        break;
    } else {
        if (newBoard[coord.x + diffRow][coord.y + diffCol] != ' '){
            isNormal = false;
            break;
        } else {
            newBoard[coord.x + diffRow][coord.y + diffCol] = piece.c;
        }
    }
}

if (isNormal){

```

```

Pair<Integer, Integer> newFirstEmpty = new Pair<>(0, 0);

boolean found = false;

for (int i = 0; i < newBoard.length; i++) {
    for (int j = 0; j < newBoard[i].length; j++) {
        if (newBoard[i][j] == ' '){
            newFirstEmpty = new Pair<>(i, j);
            found = true;
            break;
        }
    }
    if (found){
        break;
    }
}

return new Board(newBoard, newFirstEmpty, isNormal);
} else {
    Board failBoard = new Board(copyBoard, firstEmpty, isNormal);
    return failBoard;
}
}

private static void printBoard(Board newBoard){
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < N; k++) {
            System.out.print(newBoard.board[j][k] + " ");
        }
        System.out.println();
    }
}

```

```

    }
}

private static void readInput(String fileName) throws IOException{

    BufferedReader br = new BufferedReader(new FileReader(fileName));

    String[] dimensions = br.readLine().split(" ");

    M = Integer.parseInt(dimensions[0]);

    N = Integer.parseInt(dimensions[1]);

    P = Integer.parseInt(dimensions[2]);

    mode = br.readLine();

    pieces = new ArrayList<Piece>();

    char presentChar = ' ';

    String piece = "";

    String tempPiece;

    int i = 0;

    boolean stop = false;

    while (!stop) {

        tempPiece = br.readLine();

        if (i == 0){

            int k = 0;

            while(tempPiece.charAt(k) == ' '){

                k ++;

            }

            presentChar = tempPiece.charAt(k);

```

```

        piece += tempPiece;

        i ++;

    } else if (tempPiece == null){

        pieces.add(convertToPiece(convertToMatrix(piece),
presentChar));

        stop = true;

    } else {

        int k = 0;

        while(tempPiece.charAt(k) == ' '){

            k ++;

        }

        if (tempPiece.charAt(k) == presentChar){

            piece += "\n" ;

            piece += tempPiece;

        } else {

            pieces.add(convertToPiece(convertToMatrix(piece),
presentChar));

            piece = tempPiece;

            presentChar = tempPiece.charAt(k);

            i++;

        }

    }

    br.close();

};

private static char[][] convertToMatrix(String shape) {

    String[] rows = shape.split("\n");

```

```

    int rowsLength = rows.length;

    int[] colsLength = new int[rowsLength];

    for (int i = 0; i < rowsLength; i++) {

        colsLength[i] = rows[i].split("").length;

    }

    int maxColsLength = Arrays.stream(colsLength).max().getAsInt();

    char[][] matrix = new char[rowsLength][maxColsLength];

    for (int i = 0; i < rowsLength; i++) {

        for (int j = 0; j < maxColsLength; j++) {

            matrix[i][j] = ' ';

        }

    }

    for (int i = 0; i < rowsLength; i++) {

        String[] cols = rows[i].split("");

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

            matrix[i][j] = cols[j].charAt(0);

        }

    }

    return matrix;

};

private static List<Pair<Integer, Integer>> convertToPair(char[][] shape) {

    List<Pair<Integer, Integer>> pairs = new ArrayList<>();

    for (int i = 0; i < shape.length; i++) {

        for (int j = 0; j < shape[i].length; j++) {

            if (shape[i][j] != ' '){

```



```

        pairs.add(new Pair<>(i, j));
    }

}

return pairs;
};

private static Piece convertToPiece(char[][] shape, char c) {
    List<Pair<Integer, Integer>> pairs = convertToPair(shape);
    return new Piece(c, pairs);
};

private static char[][] pieceToMatrix(Piece x){
    int numRows = x.pairInMatrix.length;
    int numCol = x.pairInMatrix[0].length;

    char[][] matrix = new char[numRow][numCol];

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

    for (Pair<Integer, Integer> coord : x.coords) {
        matrix[coord.x][coord.y] = x.c;
    }

    return matrix;
}

```

```

        private static boolean validation(int M, int N, int P, String mode,
List<Piece> pieces) {

        // validasi nilai M, N

        if (M < 1){

            System.out.println("M harus lebih besar dari 0");

            return false;

        }

        if (N < 1){

            System.out.println("N harus lebih besar dari 0");

            return false;

        }

        // validasi mode

        if (!mode.equals("Default")){

            System.out.println("mode = " + mode);

            System.out.println("Mode tidak valid");

            return false;

        }

        // validasi nilai P

        if (P < 1 || P > 26){

            System.out.println("P harus lebih besar dari 0 dan lebih kecil dari
27");

            return false;

        }

        // validasi jumlah piece yang diinputkan sama dengan P

```

```

    if (pieces.size() != P){

        System.out.println("Jumlah piece yang diinputkan tidak sama dengan
P");

        return false;

    }

    List<char[][]> piecesMat = new ArrayList<>();

    for (int i = 0; i < P; i++){

        piecesMat.add(pieceToMatrix(pieces.get(i)));

    }

    // validasi piece ke i harus berisi karakter huruf ke-i pada alfabet
atau spasi

    for (int i = 0; i < P; i++) {

        char presentChar = (char) (i + 65);

        char[][] piece = piecesMat.get(i);

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

            for (int k = 0; k < piece[j].length; k++) {

                if (piece[j][k] != presentChar && piece[j][k] != ' '){

                    System.out.println("Piece " + (i + 1) + " tidak valid
karena tidak berisi karakter huruf ke-" + (i + 1) + " pada alfabet atau spasi");

                    return false;

                }

            }

        }

    }

    // validasi dimensi piece tidak melebihi board dengan ukuran M x N

    for (int i = 0; i < P; i++) {

```

```

        char[][] piece = piecesMat.get(i);

        if (piece.length > M || piece[0].length > N){

            System.out.println("Piece " + (i + 1) + " tidak valid karena
dimensinya melebihi board dengan ukuran M x N");

            return false;

        }

    }

    // validasi jumlah kotak pada seluruh piece tidak melebihi jumlah kotak
pada board

    int totalPiece = 0;

    for (int i = 0; i < P; i++) {

        char[][] piece = piecesMat.get(i);

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

            for (int k = 0; k < piece[j].length; k++) {

                if (piece[j][k] != ' '){

                    totalPiece++;

                }

            }

        }

    }

    if (totalPiece > M * N){

        System.out.println("Jumlah kotak pada seluruh piece tidak boleh
melebihi jumlah kotak pada board");

        return false;

    }

    // validasi jumlah kotak pada seluruh piece tidak kurang dari jumlah
kotak pada board

```

```

        int totalBoard = M * N;

        if (totalPiece < totalBoard){

            System.out.println("Jumlah kotak pada seluruh piece tidak boleh
kurang dari jumlah kotak pada board");

            return false;

        }

        return true;

    };

    private static void flowControl(int idx){

        if (solved) return;

        if (idx == P) {

            solved = true;

            return;

        }

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

            int num = unusedPiece.get(i);

            int transformIdx = 0;

            while (transformIdx < allTransforms.get(num).size()){

                Piece x = allTransforms.get(num).get(transformIdx);

                char[][] copyBoard = new
char[board.board.length][board.board[0].length];

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

                    for (int k = 0; k < board.board[j].length; k++) {

                        copyBoard[j][k] = board.board[j][k];

                    }

                }

            }

        }

    }

```

```

    }

    Pair<Integer,Integer> copyFirstEmpty = new
Pair<>(board.firstEmpty.x, board.firstEmpty.y);

    boolean copyIsNormalBoard = board.isNormalBoard;

    Board recordBoard = new Board(copyBoard, copyFirstEmpty,
copyIsNormalBoard);

    int[] recordPieceOrder = pieceOrder.clone();

    Board newBoard = putPiece(board,
allTransforms.get(num).get(transformIdx));

    board = newBoard;

    if (newBoard.isNormalBoard){

        pieceOrder[idx] = num;

        unusedPiece.remove((Integer) num);

        flowControl(idx + 1);

        if (solved){

            break;

        }

        unusedPiece.add(i, num);

    } else {

        board.isNormalBoard = true;

        tryCounter ++;

    }

    transformIdx ++;

    board = recordBoard;

    pieceOrder = recordPieceOrder;

}

}

};

```

```

        private static void saveResult(String filePath, long executionTime, int
tryCount, String imageFile) {

            try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath))) {

                writer.write("Hasil Pencarian Puzzle\n");

                writer.write("=====\n");

                writer.write("Waktu Eksekusi: " + executionTime + " ms\n");

                writer.write("Jumlah Kasus yang Ditinjau: " + tryCount + "\n");

                writer.write("Tautan Gambar Solusi: " + imageFile + "\n");

                writer.write("\nSolusi Papan:\n");

                for (int i = 0; i < board.board.length; i++) {

                    for (int j = 0; j < board.board[i].length; j++) {

                        writer.write(board.board[i][j] + " ");

                    }

                    writer.write("\n");

                }

                System.out.println("Hasil disimpan dalam: " + filePath);

            } catch (IOException e) {

                System.out.println("Gagal menyimpan hasil: " + e.getMessage());

            }

        }

    }

    public static void main(String[] args) throws IOException {

        Scanner sc = new Scanner(System.in);

        System.out.print("Masukkan nama file: ");

        String fileName = sc.nextLine();

```

```

String fileName1 = "test/" + fileName + ".txt";

readInput(fileName1);

Boolean isValid = validation(M, N, P, mode, pieces);

if (isValid){

    board = initiateBoard(M, N);

    allTransforms = new ArrayList<>();

    pieceOrder = new int[P];

    solved = false;

    tryCounter = 0;

    for (int i = 0; i < P; i++){

        allTransforms.add(Piece.AllTransforms(pieces.get(i)));

        unusedPiece.add(i);

    }

    long startTime = System.currentTimeMillis();

    flowControl(0);

    long endTime = System.currentTimeMillis();

    long executionTime = endTime - startTime;

    if (solved){

        for (int i = 0; i < board.board.length; i++) {

            for (int j = 0; j < board.board[i].length; j++) {

                char cell = board.board[i][j];

                if (cell == ' ') {

                    System.out.print(" ");

                } else {

                    String color = getColor(cell);

```



```

        System.out.print(color + cell + RESET + " ");

    }

}

System.out.println();

}

System.out.println("Solusi ditemukan");

System.out.println("Waktu pencarian = " + executionTime +
"ms");

System.out.println("Banyak kasus yang ditinjau: " +
tryCounter);

System.out.print("Apakah hasil ingin disimpan? (y/n): ");

String save = sc.nextLine();

if (save.equals("y")){

    String filePath = "output/" + fileName + ".txt";

    saveResult(filePath, executionTime, tryCounter,
"image/solution" + fileName + ".png");

    System.out.println("Result tersimpan di " + filePath);

}

} else {

    System.out.println("Solusi tidak ditemukan");

    System.out.println("Jumlah percobaan: " + tryCounter);

}

}

sc.close();

};

}

```

TESTING (INPUT OUTPUT)

1. Testcase 1: Input normal

Input:

5 5 7

Default

A

AA

B

BB

C

CC

D

DD

EE

EE

E

FFF

GG

GG

G

Output:

```
Masukkan nama file: 1
A A B C C
D A B B C
D D E E E
G G G E E
G G F F F
Solusi ditemukan
Waktu pencarian = 2ms
Banyak kasus yang ditinjau: 66
Apakah hasil ingin disimpan? (y/n): y
Hasil disimpan dalam: output/1.txt
Result tersimpan di output/1.txt
```

Gambar 1. Output Testcase 1

2. Testcase 2: Input Normal

Input:

6 6 7

Default

AA

AA

AA

BB
 BB
 CCC
 CCC
 DD
 DDD
 EE
 EEE
 FFF
 F
 GGG
 GGG

Output:

```
Masukkan nama file: 2
E E A A F F
E E G A A F
E D G A A F
D D G G C C
D B B G C C
D B B G C C
Solusi ditemukan
Waktu pencarian = 131ms
Banyak kasus yang ditinjau: 56710
Apakah hasil ingin disimpan? (y/n): y
Hasil disimpan dalam: output/2.txt
Result tersimpan di output/2.txt
```

Gambar 2. Output Testcase 2

3. Testcase 3: jumlah block pada pieces melebihi jumlah block pada board

Input:

5 8 8
 Default
 AA
 AAA
 BBB
 BB
 CC
 CCC
 C
 DD
 DD
 DD
 EEE
 EEE

```

FF
FF
FF
GGGG
GG
HHH
HHH

```

Output:

```

Masukkan nama file: 3
Jumlah kotak pada seluruh piece tidak boleh melebihi jumlah kotak pada board

```

Gambar 3. Output Testcase 3

4. Testcase 4: jumlah block pada pieces kurang dari jumlah block pada board

Input:

```

7 7 9
Default
AA
AAA
BB
BBBB
CC
CCC
CC
DDDD
EEE
EEE
FFF
FF
GG
GGG
HHHH
HH
I
III

```

Output:

```

PS C:\Users\sabil\Downloads\SEMESTER 4\STIMA\tucil_1\Tucil1_13523072> java -cp bin IQPuzzleSolver
Masukkan nama file: 4
Jumlah kotak pada seluruh piece tidak boleh kurang dari jumlah kotak pada board

```

Gambar 4. Output Testcase 4

5. Testcase 5: jumlah block pada pieces melebihi jumlah block pada board

Input:

```

6 8 10
Default

```

```

AAAAA
BB
BBBB
CCC
CC
DDDDD
D
EEEE
E
FFF
FFF
GGG
G
HHHHH
III
III
JJJ
JJJ

```

Output:

```

PS C:\Users\sabil\Downloads\SEMESTER 4\STIMA\tucil_1\tucil1_13523072> java -cp bin IQPuzzleSolver
Masukkan nama file: 5
Jumlah kotak pada seluruh piece tidak boleh melebihi jumlah kotak pada board

```

Gambar 5. Output Testcase 5

6. Testcase 6: jumlah block pada pieces kurang dari jumlah block pada board

Input:

```

8 8 8
Default
AA
AA
AA
BBB
B
CCC
CCC
DDD
DD
EEE
EEE
FFFF
F
GGG
GGG
HH
HHH

```

Output:

```
PS C:\Users\sabil\Downloads\SEMESTER 4\STIMA\tucil_1\tucil1_13523072> java -cp bin IQPuzzleSolver
Masukkan nama file: 6
Jumlah kotak pada seluruh piece tidak boleh kurang dari jumlah kotak pada board
```

Gambar 6. Output Testcase 6

7. Testcase 7: jumlah block pada pieces kurang dari jumlah block pada board

Input:

```
6 9 9
Default
AAAA
AA
BBB
BB
CCC
CCC
DDDD
EEEE
E
FFFF
F
GG
GGG
HHHH
HH
III
I
```

Output:

```
PS C:\Users\sabil\Downloads\SEMESTER 4\STIMA\tucil_1\tucil1_13523072> java -cp bin IQPuzzleSolver
Masukkan nama file: 7
Jumlah kotak pada seluruh piece tidak boleh kurang dari jumlah kotak pada board
```

Gambar 7. Output Testcase 7

PRANALA KE REPOSITORY KODE PROGRAM

Berikut adalah pranala ke repository github yang berisi kode program: [Pranala Kode Program](#)

