Laporan Tugas Kecil IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun Oleh:
Andri Nurdianto (13523145)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

Daftar Isi

Bab 1

1.1 Deskripsi Masalah

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:
1. Board (Papan) – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
2. Blok/Piece – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Tugas anda adalah menemukan cukup satu solusi dari permainan IQ Puzzler Pro dengan menggunakan algoritma Brute Force, atau menampilkan bahwa solusi tidak ditemukan jika tidak ada solusi yang mungkin dari puzzle.

Pemain berusaha untuk mengisi bagian papan yang kosong dengan menggunakan blok yang tersedia

PERHATIAN: Untuk tucil ini permainan diawali dengan papan kosong
Permainan dinyatakan selesai jika pemain mampu mengisi seluruh papan dengan blok (dalam tugas kecil ini ada kemungkinan pemain tidak dapat mengisi seluruh papan)

Bab 2
Implementasi Program

2.1 Algoritma

1. Program akan mengecek terlebih dahulu stack yang berisi shape kosong atau tidak.
2. Jika stack kosong, program akan mengecek papan telah selesai terisi atau tidak.
3. Jika papan terisi dan semua shape dipakai, program menemukan solusi.
4. Jika papan tidak terisi dan semua shape dipakai, program tidak menemukan solusi.
5. Shape diambil dari stack bernama shapeList yang berisi semua kemungkinan potongan puzzle.
6. Shape akan dimasukkan ke dalam papan dengan mencoba semua kemungkinan rotasi dan pencerminan.
7. Program akan mengecek shape bisa ditaruh atau tidak.
8. Jika bisa ditaruh, shape akan ditaruh ke papan dan jika tidak shape akan dirotasi atau dicerminkan.
9. Banyak percobaan akan bertambah satu untuk setiap kemungkinan bentuk dan bagian papan yang bisa ditaruh.
10. Jika kondisi semua isi stack terpakai dan papan terisi dengan benar, proses percobaan akan berhenti dan mengembalikan nilai true.
11. Jika kondisi no. 10 belum terpenuhi, program akan menghapus potongan puzzle dari papan.
12. Program akan mengembalikan shape ke dalam stack agar bisa dicoba di posisi lain atau dengan bentuk yang berbeda.
13. Jika semua percobaan dijalankan dan solusi tidak ditemukan, program akan mengembalikan nilai false.

2.2 Pseudocode

```
FUNCTION processSolve() RETURN boolean:
    IF shapeList is empty THEN
        IF finishBoard() is true THEN
            RETURN true
        ELSE
            RETURN false
    shape ← pop(shapeList)

    FOR each row index i in board:
        FOR each column index j in board:
            FOR each rotate/mirror piece in shape:
                IF canPlacePiece(piece, i, j) is true THEN
                    placePiece(piece, i, j)
                    noOfAttempt ← noOfAttempt + 1

                    IF processSolve() return true THEN
                        RETURN true
```

```
                    removePiece(piece, i, j)

    push(shapeList, shape)
    RETURN false
```

2.3 Source Program

Main.java

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

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        File file = FileInput.userFile(input);

        String[] lines = FileInput.readFile(file);
        HandleInput.checkInput(lines);

        Grid.Board(lines);
        Piece.processPieces(lines);

        Solver.dataTransform();

        Solver.timeElapsed();
        System.out.println();
        boolean result = Solver.processSolve();
        if(!result) {
            System.out.println("Tidak ada solusi");
        } else {
            System.out.println("Solusi ditemukan!");
            System.out.println();
            Grid.printBoard(Grid.board);
        }
        System.out.println();
        System.out.println("Estimasi waktu penyelesaian: " +
Solver.timeEstimated + "ms");
        System.out.println();
        System.out.println("Banyak percobaan: " +
Solver.noOfAttempt);
        System.out.println();

        if (result) {
            FileOutput.savingSolution(lines);
        }

        input.close();
    }
}
```

FileInput.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileInput {
    public static File userFile(Scanner input) {

        System.out.print("Masukkan nama file: ");
        String fileName = input.nextLine();

        File file = new File("input/" + fileName);


        return file;
    }

    public static void printFile(File file) {
        try {
            Scanner fileScanner = new Scanner(file);
            while (fileScanner.hasNextLine()) {
                System.out.println(fileScanner.nextLine());
            }
            fileScanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File tidak ditemukan.");
        }
    }

    public static String[] readFile(File file) {
        try {
            Scanner fileScanner = new Scanner(file);
            String[] lines = new String[10000];
            int count = 0;

            while (fileScanner.hasNextLine() && count <
lines.length) {
                lines[count++] = fileScanner.nextLine();
            }
            fileScanner.close();

            String[] result = new String[count];
            System.arraycopy(lines, 0, result, 0, count);
            return result;

        } catch (FileNotFoundException e) {
            System.out.println("File tidak ditemukan." );
            return new String[0];
        }
```

```
    }
}
```

HandleInput.java

```
public class HandleInput {
    public static void checkInput(String[] lines) {
        if (lines == null || lines.length < 3) {
            System.out.println("Format tidak sesuai, file
input kurang dari 3 baris");
            System.exit(1);
        }

        String[] firstLine = lines[0].trim().split("\\s+");
        if (firstLine.length != 3) {
            System.out.println("Format tidak sesuai, baris
pertama harus memiliki 3 input angka");
            System.exit(1);
        }

        if (!lines[1].strip().equals("DEFAULT")) {
            System.out.println("Format bentuk tidak sesuai,
hanya bisa menerima format DEFAULT");
            System.exit(1);
   }

        int N = 0;
        int M = 0;
        int P = 0;
        try {
            N = Integer.parseInt(firstLine[0]);
            M = Integer.parseInt(firstLine[1]);
            P = Integer.parseInt(firstLine[2]);

            if (N <= 0 || M <= 0 || P <= 0) {
                System.out.println("Format tidak sesuai");
                System.exit(1);
            }
        } catch (NumberFormatException e) {
            System.out.println("Format tidak sesuai");
            System.exit(1);
        }

        if (lines[1].trim().isEmpty()) {
            System.out.println("Format tidak sesuai");
            System.exit(1);
        }

        if (lines.length < P + 2) {
```

```
            System.out.println("Format tidak sesuai");
            System.exit(1);
        }
    }
}
```

Grid.java

```
public class Grid {
    public static char[][] board;

    public static void Board(String[] lines) {
        String[] data = lines[0].split(" ");
        int row = Integer.parseInt(data[0]);
        int column = Integer.parseInt(data[1]);

        board = new char[row][column];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++) {
                board[i][j] = '0';
            }
        }
    }

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

    public static String boardColoring(char character) {
        String init = "\u001B[0m";
        String[] color = {
            "\u001B[31m",
            "\u001B[32m",
            "\u001B[33m",
            "\u001B[34m",
            "\u001B[35m",
            "\u001B[36m",
            "\u001B[91m",
            "\u001B[92m",
            "\u001B[93m",
            "\u001B[94m",
            "\u001B[95m",
            "\u001B[96m",
```

```
        };

        if (character >= 'A' && character <= 'Z') {
            int index = (character - 'A') % color.length;
            return color[index] + character + init;
        }

        return "\u001B[37m" + character + init;
    }

}
```

Piece.java

```
import java.util.List;
import java.util.ArrayList;

public class Piece {
    public static List <char[][]> pieceList = new
ArrayList<>();

    public static int getPieceCount(String[] lines) {
        String[] data = lines[0].split(" ");
        return Integer.parseInt(data[2]);
    }

    public static void printPieceCount(int pieceCount) {
        System.out.println(pieceCount);
    }

    public static char[][] createPieceShape(String[]
pieceLines) {
        int row = pieceLines.length;
        if (row == 0) {
            return null;
        }

        int maxLength = 0;
        for (String line : pieceLines) {
            maxLength = Math.max(maxLength, line.length());
        }

        char[][] shape = new char[row][maxLength];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < maxLength; j++) {
                shape[i][j] = ' ';
            }
        }
```

```
        for (int i = 0; i < row; i++) {
            char[] rowChars = pieceLines[i].toCharArray();
            for (int j = 0; j < rowChars.length; j++) {
                shape[i][j] = rowChars[j];
            }
        }

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < maxLength; j++) {
                if (shape[i][j] == ' ') {
                    shape[i][j] = '0';
                }
            }
        }

        return shape;
    }

    public static void printPiece(char[][] shape) {
        for (char[] row : shape) {
            System.out.println(new String(row));
        }
    }

    public static void processPieces(String[] lines) {
        int index = 2;
        while (index < lines.length) {
            String first = lines[index].trim();
            if (first.isEmpty()) {
                index++;
                continue;
            }

            char letter = first.charAt(0);
            int pieceStart = index;

            while (index < lines.length &&
lines[index].trim().charAt(0) == letter) {
                index++;
            }

            int pieceSize = index - pieceStart;
            String[] pieceLines = new String[pieceSize];
            System.arraycopy(lines, pieceStart, pieceLines,
0, pieceSize);

            char[][] shape = createPieceShape(pieceLines);
            pieceList.add(shape);
        }
```

```
    }
}
```

Solver.java

```
import java.util.Deque;
import java.util.ArrayDeque;

public class Solver {
    public static Deque <char[][][]> shapeList = new
ArrayDeque<>();
    public static double timeEstimated = 0;
    public static long noOfAttempt = 0;

    public static char[][] rotate(char[][] shape) {
        int row = shape.length;
        int column = shape[0].length;
        char[][] newShape = new char[column][row];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++) {
                newShape[j][row - 1 - i] = shape[i][j];
            }
        }

        return newShape;
    }

    public static char[][] mirror(char[][] shape) {
        int row = shape.length;
        int column = shape[0].length;
        char[][] newShape = new char[row][column];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++) {
                newShape[i][j] = shape[i][column - 1 - j];
            }
        }

        return newShape;
    }

    public static char[][][] shapeTransform(char[][] shape) {
        char[][][] newShape = new char[8][][];
        //rotate
        newShape[0] = shape;
        newShape[1] = rotate(newShape[0]);
        newShape[2] = rotate(newShape[1]);
        newShape[3] = rotate(newShape[2]);
```

```
        // mirror
        newShape[4] = mirror(newShape[0]);
        newShape[5] = rotate(newShape[4]);
        newShape[6] = rotate(newShape[5]);
        newShape[7] = rotate(newShape[6]);

        return newShape;
    }

    public static void dataTransform() {
        for (char[][] piece: Piece.pieceList) {
            shapeList.offer(shapeTransform(piece));
        }
    }

    public static boolean finishBoard() {
        for (char[] row : Grid.board) {
            for (char column : row) {
                if (column == '0'){
                    return false;
                }
            }
        }
        return true;
    }

    public static boolean processSolve() {
        if(shapeList.isEmpty()) {
            if(finishBoard()) {
                return true;
            } else {
                return false;
            }
        }

        char[][][] shape = shapeList.pop();

        for (int i = 0; i < Grid.board.length; i++) {
            for (int j = 0; j < Grid.board[0].length; j++) {
                for (char[][] piece : shape) {
                    if (canPlacePiece(piece, i, j)) {
                        placePiece(piece, i, j);
                        noOfAttempt++;
                        if(processSolve()) {
                            return true;
                        }
                        removePiece(piece, i, j);
                    }
                }
            }
```

```
        }

        shapeList.push(shape);
        return false;
    }

    public static boolean canPlacePiece(char[][] piece, int
x, int y) {
        int pieceRow = piece.length;
        int pieceColumn = piece[0].length;
        int boardRow = Grid.board.length;
        int boardColumn = Grid.board[0].length;

        for (int i = 0; i < pieceRow; i++) {
            for (int j = 0; j < pieceColumn; j++) {
                if (piece[i][j] != '0') {
                    int boardX = x + i;
                    int boardY = y + j;

                    if (boardX >= boardRow || boardY >=
boardColumn) {
                        return false;
                    }

                    if (Grid.board[boardX][boardY] != '0') {
                        return false;
                    }
                }
            }
        }
        return true;
    }

    public static void placePiece(char[][] piece, int x, int
y) {
        int pieceRow = piece.length;
        int pieceColumn = piece[0].length;

        for (int i = 0; i < pieceRow; i++) {
            for (int j = 0; j < pieceColumn; j++) {
                if (piece[i][j] != '0') {
                    int boardX = x + i;
                    int boardY = y + j;
                    Grid.board[boardX][boardY] = piece[i][j];
                }
            }
        }
    }

    public static void removePiece(char[][] piece, int x, int
```

```
y) {
        int pieceRow = piece.length;
        int pieceColumn = piece[0].length;

        for (int i = 0; i < pieceRow; i++) {
            for (int j = 0; j < pieceColumn; j++) {
                if (piece[i][j] != '0') {
                    int boardX = x + i;
                    int boardY = y + j;
                    Grid.board[boardX][boardY] = '0';
                }
            }
        }
    }

    public static void timeElapsed() {
        long start = System.nanoTime();
        processSolve();
        long finish = System.nanoTime();
        timeEstimated = (double)(finish - start) / 1e+6;
    }
}
```

FileOutput.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;

public class FileOutput {
    public static final Scanner inputFile = new
Scanner(System.in);
    public static char[][] board;

    public static Path fileName(String prompt) {
        System.out.print(prompt);
        String fileName = inputFile.nextLine().trim();

        Path path = Paths.get("test", fileName);

        return path;
    }

    public static void saveSolutionFile(String[] lines) {
        Path filePath = fileName("Masukkan nama file (contoh:
solusi.txt): ");
```

```
        try (PrintWriter out = new PrintWriter(new
FileWriter(filePath.toFile()))) {
            out.println("Solusi ditemukan!");
            out.println();
            char[][] board = Grid.board;
            for (char[] row : board) {
                out.println(new String(row));
            }
            out.println();
            out.println("Estimasi waktu penyelesaian: " +
Solver.timeEstimated + "ms");
            out.println();
            out.println("Banyak percobaan: " +
Solver.noOfAttempt);
            System.out.println("Solusi berhasil disimpan di "
+ filePath);
        } catch (IOException e) {
            System.err.printf("Error saat menyimpan file:
%s%n", e.getMessage());
        }
    }

    public static void savingSolution(String[] lines) {
        System.out.print("Apakah ingin menyimpan solusi
(ya/tidak)? ");
        if (inputFile.hasNextLine()) {
            String ans = inputFile.nextLine().trim();
            if (ans.equalsIgnoreCase("ya")) {
                saveSolutionFile(lines);
            } else {
                System.out.println();
                System.out.println("Solusi tidak akan
disimpan.");
            }
        } else {
            System.out.println("Tidak ada input yang
tersedia, solusi tidak disimpan.");
        }
    }

}
```

Bab 3

3.1 Eksperimen

Input file1.txt

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Output file1.txt

```
$ java -cp bin Main
Masukkan nama file: file1.txt

Solusi ditemukan!

ABBCC
AABCD
EEEDD
EEFFF
GGGFF

Estimasi waktu penyelesaian: 5.1962ms
```

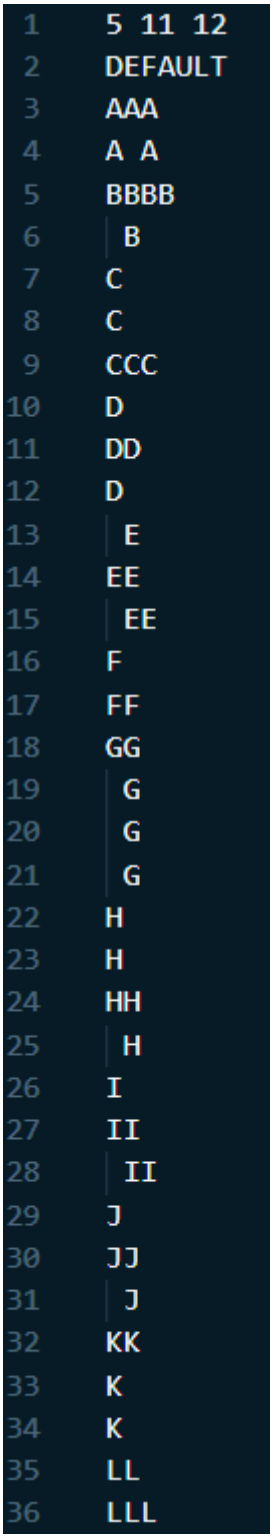
Input file2.txt

```
3 3 3
DEFAULT
AAA
BB
C
CCC
```

Output file2.txt

```
$ java -cp bin Main
Masukkan nama file: file2.txt

Solusi ditemukan!

AAA
BBC
CCC

Estimasi waktu penyelesaian: 0.0142ms

Banyak percobaan: 3
```

Input file3.txt



Output file3.txt

```
$ java -cp bin Main
Masukkan nama file: file3.txt

Solusi ditemukan!

AAABCFDDDEE
AHABCFFDEEG
KHBBCCCIIEG
KHHBLLIIJJG
KKHLLLIJJGG

Estimasi waktu penyelesaian: 15349.6228ms

Banyak percobaan: 8841807
```

Input file4.txt

```
2 2 2
DEFAULT
AA
AA
B
```

Output file4.txt

```
$ java -cp bin Main
Masukkan nama file: file4.txt

Tidak ada solusi

Estimasi waktu penyelesaian: 0.0433ms

Banyak percobaan: 16
```

Input file5.txt

```
2 5 4
DEFAULT
A
BB
CCC
E
```

Output file5.txt

```
$ java -cp bin Main
Masukkan nama file: file5.txt

Tidak ada solusi

Estimasi waktu penyelesaian: 0.0149ms

Banyak percobaan: 32
```

Input file6.txt

```
3 10 8
DEFAULT
A
A
AA
BB
BB
 B
 B
 B
 B
C
DD
 D
E
E
E
FF
GGG
G G
GGG
H
```

Output file6.txt

```
$ java -cp bin Main
Masukkan nama file: file6.txt

Solusi ditemukan!

ABBBBBBGGG
ABBCDDFGHG
AAEEEDFGGG

Estimasi waktu penyelesaian: 4213.2753ms

Banyak percobaan: 5138578
```

Input file7.txt

```
3 3 4
DEFAULT
AAA
 B
BB
C
D
```

Output file7.txt

```
$ java -cp bin Main
Masukkan nama file: file7.txt

Tidak ada solusi

Estimasi waktu penyelesaian: 25.6941ms

Banyak percobaan: 209456
```

Lampiran

Pranala *repository* kode program: https://github.com/drianto/Tucil1_13523145

Tabel Ketercapaian:

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)		✓
6	Program dapat menyimpan solusi dalam bentuk file gambar		✓
7	Program dapat menyelesaikan kasus konfigurasi custom		✓
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	