

LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:
Henry Filberto Shenelo 13523108

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

BAB I ALGORITMA BRUTE FORCE.....	3
BAB II SOURCE CODE.....	5
BAB III SCREENSHOT HASIL TEST.....	24
LINK REPOSITORY.....	30
CHECKLIST.....	30

BAB I

ALGORITMA BRUTE FORCE

A. Penjelasan Dasar

Algoritma Brute Force, pendekatan yang lurus atau straightforward untuk memecahkan suatu persoalan. Algoritma ini menggunakan metode pencarian atau pemecahan masalah dengan mencoba semua kemungkinan solusi secara sistematis sampai menemukan jawaban yang benar. Teknik ini biasanya sederhana tetapi bisa sangat tidak efisien, terutama untuk masalah dengan ruang kemungkinan yang besar. Algoritma ini biasanya bergantung pada kekuatan komputasi yang tinggi untuk mendapatkan semua solusi yang tepat daripada menggunakan teknik yang canggih. Beberapa contoh permasalahan yang dapat diselesaikan menggunakan algoritma Brute Force antara lain:

1. Mencari elemen terbesar atau terkecil dalam suatu list dengan membandingkan suatu elemen dengan semua elemen yang ada untuk mendapatkan nilai terbesar/terkecil.
2. Pencarian secara berurutan dengan membandingkan nilai yang dicari dengan setiap elemen dalam daftar.

B. Aplikasi *Brute Force* pada Penyelesaian IQ Puzzle Pro

Pada permainan IQ Puzzle Pro, akan terdapat sebuah papan berukuran $M \times N$. Objektif dari permainan ini adalah menyusun P jumlah blok pada papan sehingga papan tersebut terisi penuh. Terdapat juga versi IQ Puzzle Pro dengan objektif membentuk suatu piramida dengan blok-blok tersebut. IQ Puzzle Pro bisa dibuat simulasinya dengan menggunakan Algoritma Brute Force. Ini artinya program akan mencari semua kemungkinan posisi dari tiap blok (beserta rotasinya) agar dapat mengisi papan hingga penuh.

Berikut adalah penjelasan singkat akan aplikasi algoritma Brute Force terhadap IQ Puzzle Pro.

1. Program menerima file .txt yang berisi ukuran papan serta daftar blok yang tersedia. Data ini kemudian diproses (*parsing*) agar setiap blok tersimpan dalam bentuk list koordinat.
2. Setiap blok akan diperlakukan seperti sebuah matriks dengan elemen berupa huruf (A-Z) dan " " (spasi). Setiap blok kemudian diubah ke dalam bentuk koordinat. Blok dikonversi ke dalam koordinat relatif dengan titik (0,0) yang terletak pada bagian kiri atas dari matriks.
3. Setiap blok mulai dari yang pertama di list akan dicoba dimasukkan ke dalam papan pada setiap posisi (x,y). Jika blok tidak dapat dimasukkan pada posisi manapun, blok akan mencoba semua rotasi (90°, 180°, 270°). Jika blok bisa ditempatkan, program lanjut ke blok berikutnya dengan langkah yang sama.
4. Jika tidak ada posisi dan rotasi yang memungkinkan untuk suatu blok, blok sebelumnya akan dihapus (*backtrack*). Program kemudian mencoba kembali dengan posisi atau rotasi lain untuk blok sebelumnya.

5. Proses ini berlanjut hingga semua blok berhasil ditempatkan, yang berarti papan terisi penuh dan solusi ditemukan. Jika semua kemungkinan sudah dicoba dan tidak ada solusi yang valid, artinya tidak terdapat solusi.

Adapun untuk kasus piramida, terdapat sedikit perbedaan:

1. Setiap blok dikonversi ke dalam koordinat relatif 3D (x, y, z) dengan titik referensi (0,0,0) di sudut kiri bawah piramida.
2. Setiap blok dicoba ditempatkan pada berbagai posisi dalam piramida, dengan mencoba pada berbagai stack/tingkat mulai dari bawah hingga tingkat atas.
3. Blok juga coba dirotasi 45° dalam 3D sehingga ada 8 kemungkinan rotasi.

Berikut adalah pseudocode dari algoritma Brute Force yang digunakan.

```
function solvePuzzle2D(blockIndex):  
  
    if (blockIndex >= totalBlocks) then  
        if (isBoardFull()) then  
            puzzleSolved = true  
            return  
        currentBlock = blocks[blockIndex]  
        currentLetter = blockLetters[blockIndex]  
  
        for rotation in 0 to 3: //rotasi  
            rotatedBlock = rotateBlock2D(currentBlock, rotation)  
  
            for x = 0 to Rows-1:  
                for y = 0 to Cols-1:  
                    iterations++  
  
                    if canPlaceBlock(x, y, rotatedBlock):  
                        placeBlock(x, y, rotatedBlock, currentLetter)  
                        solvePuzzle(blockIndex + 1) //blok selanjutnya  
                        if puzzleSolved:  
                            return  
  
// backtrack  
removeBlock(x, y, rotatedBlock)
```

```

function solvePuzzlePyramid(blockIndex):

    if blockIndex >= totalBlocks:

        if isPyramidFull():

            puzzleSolved = true

            return

    currentBlock = blocks[blockIndex]

    currentLetter = blockLetters[blockIndex]

    for rotation in 0 to 7: //rotasi 3D 45 derajat

        rotatedBlock = rotateBlock3D(currentBlock, rotation)

        for bottomStack = 0 to totalStacks-1:

            for endStack = bottomStack to totalStacks-1:

                for x = 0 to pyramidWidth-1:

                    for y = 0 to pyramidHeight-1:

                        iterations++

                        if canPlaceBlockInPyramid(bottomStack,
endStack, x, y, rotatedBlock):

                            placeBlockInPyramid(bottomStack,
endStack, x, y, rotatedBlock, currentLetter)

```

```

        solvePuzzlePyramid(blockIndex + 1)

        if puzzleSolved:

            return

        // backtrack

        removeBlockFromPyramid(bottomStack,
endStack, x, y, rotatedBlock)

```

BAB II

SOURCE CODE

Tugas kecil ini diimplementasikan dengan menggunakan Bahasa pemrograman Java. Berikut adalah daftar class yang digunakan:

- a. BlockInput.class
- b. Rotate.class
- c. TransformMatrix.class
- d. Pyramid.class
- e. Main.class

Selain itu, daftar functions dan procedures yang digunakan adalah sebagai berikut.

- a. ReadInputFromFile()
- b. BlockToCoordinates(List<String> block)
- c. validateCharacterInput(String line)
- d. getLetter(String line)
- e. TransformMatrix(int rows, int cols)
- f. eraseBlock(int pivotX, int pivotY, List<int[]> block)
- g. BlockFitCheck(int pivotX, int pivotY, List<int[]> block)
- h. addBlock(int pivotX, int pivotY, List<int[]> block, char alphabet)
- i. printMatrix()
- j. saveMatrix(String filename, boolean solutionFound)
- k. saveMatrixAsImage(String filename)
- l. Pyramid(int baseRows, int baseCols)
- m. BlockFitCheckPyramid(int bottomStack, int topStack, int pivotX, int pivotY, List<int[]> block)
- n. addBlockPyramid(int bottomStack, int topStack, int pivotX, int pivotY, List<int[]> block, char alphabet)
- o. eraseBlockPyramid(int stack, int pivotX, int pivotY, List<int[]> block)
- p. printPyramid()
- q. PyramidFullCheck()
- r. savePyramidMatrix(String filename, boolean solutionFound)
- s. savePyramidAsImage(String filename)
- t. printBlock(List<int[]> block)
- u. rotate90(List<int[]> block)
- v. NormalizeCoordinates(List<int[]> block)
- w. RotateBlocks(List<int[]> block, int NumofRotationBy90)
- x. rotate45(List<int[]> block)
- y. NormalizeCoordinates3D(List<int[]> block)
- z. RotateBlocksBy45(List<int[]> block, int rotation)
- aa. main(String[] args)
- bb. BoardFullCheck()
- cc. solvePuzzle(int blockIndex)

dd. solvePuzzlePyramid(int blockIndex)

Berikut adalah *source code* dari program yang telah dibuat

1. File: BlockInput.java

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

public class BlockInput {

    public static int M;
    public static int N;
    public static int P;
    public static String Type;
    public static List<Character> blockLetters = new ArrayList<>();

    private static String[] COLORS = {
        "#FF0000", "#00FF00", "#0000FF", "#FFFF00", "#FF00FF", "#00FFFF", "#800000", "#808000",
        "#008000", "#800080", "#008080", "#000080", "#FFA500", "#A52A2A", "#8A2BE2", "#5F9EA0",
        "#7FFF00", "#D2691E", "#DC143C", "#FF8C00", "#8B008B", "#556B2F", "#9932CC", "#8B0000",
        "#E9967A", "#9400D3"
    };

    public static Map<Character, String> letterColors = new HashMap<>();

    public static List<List<String>> ReadInputFromFile() {
        Scanner inputScanner = new Scanner(System.in);

        // Nama file
        System.out.print(s:"Masukkan nama file: ");
        String filePath = "../test/" + inputScanner.nextLine().trim();

        List<List<String>> blocks = new ArrayList<>();

        //Mapping warna
        int colorIndex = 0;

        try {
            Scanner fileScanner = new Scanner(new File(filePath));

            // Baca dimensi dan jumlah blok

            if (fileScanner.hasNextLine()) {
                String[] numbers = fileScanner.nextLine().split("\\s+");
                if (numbers.length != 3) {
                    System.out.println(x:"Error: Baris pertama harus berisi 3 bilangan bulat.");
                    return null;
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + filePath);
            return null;
        }

        // Baca blok
        while (fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine().trim();
            if (line.isEmpty()) continue;

            List<String> block = new ArrayList<>();
            String[] letters = line.split("\\s+");
            for (String letter : letters) {
                block.add(letter);
            }

            blocks.add(block);
        }

        return blocks;
    }
}
```

```

public static List<List<String>> ReadInputFromFile() {
    try {
        M = Integer.parseInt(numbers[0]);
        N = Integer.parseInt(numbers[1]);
        P = Integer.parseInt(numbers[2]);
        if (M <= 0 || N <= 0) {
            System.out.println(x:"Error: Dimensi matriks harus lebih besar dari 0.");
            return null;
        }
        if (P <= 0 || P > 26) {
            System.out.println(x:"Error: Jumlah blok harus antara 1-26.");
            return null;
        }
    } catch (NumberFormatException e) {
        System.out.println(x:"Error: Baris pertama harus berisi 3 bilangan bulat.");
        return null;
    }
} else {
    System.out.println(x:"Error: File harus memiliki setidaknya dua baris.");
    return null;
}

// Baca tipe
if (fileScanner.hasNextLine()) {
    Type = fileScanner.nextLine().trim().toUpperCase();
    if (!"DEFAULT".equals(Type) && !"PYRAMID".equals(Type)) {
        System.out.println(x:"Error: Tipe harus merupakan 'DEFAULT' atau 'PYRAMID'.");
        return null;
    }
} else {
    System.out.println(x:"Error: File harus memiliki setidaknya dua baris.");
    return null;
}

// Membaca blok
List<String> currentBlock = null;
char currentLetter = '\0';
int blockCount = 0;

while (fileScanner.hasNextLine()) {
    String line = fileScanner.nextLine();
    if (!line.isEmpty()) {
        if (!validateCharacterInput(line)) {
            System.out.println(x:"Error: Terdapat karakter yang tidak valid pada file. Hanya A-Z dan spasi yang diperbolehkan.");
            return null;
        }
        char firstChar = getLetter(line);

        if (firstChar != currentLetter) {
            if (currentBlock != null) {
                blocks.add(currentBlock);
            }
            currentBlock = new ArrayList<>();
            currentLetter = firstChar;
            blockLetters.add(firstChar);
            blockCount++;
        }
    }
}

```

```

        if (!letterColors.containsKey(currentLetter)) {
            letterColors.put(currentLetter, COLORS[colorIndex]);
            colorIndex++;
        }

        currentBlock.add(line);
    }

    if (currentBlock != null) {
        blocks.add(currentBlock);
    }

    fileScanner.close();
}

```



```

public static List<List<String>> ReadInputFromFile() {
    // Validasi jumlah blok
    if (blockCount != P) {
        System.out.println("Error: Jumlah blok tidak sesuai dengan jumlah blok (" + P + ").");
        return null;
    }
    return blocks;
} catch (FileNotFoundException e) {
    System.out.println(x:"File tidak ditemukan. Pastikan nama file ditambah dengan .txt");
}
return null;
}

private static boolean validateCharacterInput(String line) {
    for (int i = 0; i < line.length(); i++) {
        char ch = line.charAt(i);
        if (ch != ' ' && (ch < 'A' || ch > 'Z')) {
            return false;
        }
    }
    return true;
}

private static char getLetter(String line) {
    for (int i = 0; i < line.length(); i++) {
        if (line.charAt(i) != ' ') {
            return line.charAt(i);
        }
    }
    return '\0';
}

public static List<int[]> BlockToCoordinates(List<String> block) {
    List<int[]> coordinates = new ArrayList<>();
    int baseRow = 0, baseCol = 0;

    for (int row = 0; row < block.size(); row++) {
        String line = block.get(row);
        for (int col = 0; col < line.length(); col++) {
            if (line.charAt(col) != ' ') {
                coordinates.add(new int[]{row - baseRow, col - baseCol});
            }
        }
    }
    return coordinates;
}

```

2. File: Rotate.java

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

public class Rotate {

    // Print blok dan koordinat
    public static void printBlock(List<int[]> block) {
        for (int[] coord : block) {
            System.out.println("(" + coord[0] + ", " + coord[1] + ")");
        }
        System.out.println();
    }

    // Rotate blok 90 derajat counterclockwise
    private static List<int[]> rotate90(List<int[]> block) {
        List<int[]> rotated = new ArrayList<>();

        // rotasi setiap koordinat pada blok
        for (int[] coord : block) {
            rotated.add(new int[]{-coord[1], coord[0]});
        }

        return NormalizeCoordinates(rotated);
    }

    // Normalisasi koordinat agar mulai dari (0,0)
    private static List<int[]> NormalizeCoordinates(List<int[]> block) {
        int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE;

        for (int[] coord : block) {
            minX = Math.min(minX, coord[0]);
            minY = Math.min(minY, coord[1]);
        }

        List<int[]> normalized = new ArrayList<>();
        for (int[] coord : block) {
            normalized.add(new int[]{coord[0] - minX, coord[1] - minY});
        }

        return normalized;
    }

    // Rotasi blok (0,90,180,270) dengan n kali rotasi 90 counterclockwise
    public static List<int[]> RotateBlocks(List<int[]> block, int NumofRotationBy90) {
        List<int[]> rotated = block;
        for (int i = 0; i < NumofRotationBy90; i++) {
            rotated = rotate90(rotated);
        }
        return rotated;
    }
}
```

```

private static List<int[]> NormalizeCoordinates3D(List<int[]> block) {
    int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE, minZ = Integer.MAX_VALUE;

    for (int[] coord : block) {
        minX = Math.min(minX, coord[0]);
        minY = Math.min(minY, coord[1]);
        minZ = Math.min(minZ, coord[2]);
    }

    List<int[]> normalized = new ArrayList<>();
    for (int[] coord : block) {
        normalized.add(new int[]{coord[0] - minX, coord[1] - minY, coord[2] - minZ});
    }

    return normalized;
}

public static List<int[]> RotateBlocksBy45(List<int[]> block, int rotation) {
    List<int[]> rotated = new ArrayList<>();

    double angle = Math.toRadians(rotation * 45);

    // Apply the rotation around X, Y, Z axes
    for (int[] coord : block) {
        int x = coord[0];
        int y = coord[1];
        int z = 0;

        int newX = x;
        int newY = (int)(y * Math.cos(angle) - z * Math.sin(angle));
        int newZ = (int)(y * Math.sin(angle) + z * Math.cos(angle));

        int rotatedX = (int)(newX * Math.cos(angle) + newZ * Math.sin(angle));
        int rotatedY = newY;
        int rotatedZ = (int)(-newX * Math.sin(angle) + newZ * Math.cos(angle));

        int finalX = (int)(rotatedX * Math.cos(angle) - rotatedY * Math.sin(angle));
        int finalY = (int)(rotatedX * Math.sin(angle) + rotatedY * Math.cos(angle));

        rotated.add(new int[]{finalX, finalY, rotatedZ});
    }

    return NormalizeCoordinates3D(rotated);
}
}

```

3. File: TransformMatrix.java

```
import java.util.List;
import java.io.FileWriter;
import java.io.IOException;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.File;

public class TransformMatrix {
    public char[][] matrix;
    public int rows, cols;

    // Inisialisasi
    public TransformMatrix(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.matrix = new char[rows][cols];

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

    // Hapus blok
    public void eraseBlock(int pivotX, int pivotY, List<int[]> block) {
        for (int[] coord : block) {
            int x = pivotX + coord[0];
            int y = pivotY + coord[1];

            matrix[x][y] = ' ';
        }
    }

    // cek apakah posisi blok melebihi dimensi matriks
    private boolean isEffective(int x, int y) {
        return x >= 0 && x < rows && y >= 0 && y < cols;
    }
}
```

```

// cek apakah blok bisa ditambahkan
public boolean BlockFitCheck(int pivotX, int pivotY, List<int[]> block) {
    for (int[] coord : block) {
        int x = pivotX + coord[0];
        int y = pivotY + coord[1];

        if (!isEffective(x, y) || matrix[x][y] != ' ') {
            return false;
        }
    }
    return true;
}

public boolean addBlock(int pivotX, int pivotY, List<int[]> block, char alphabet) {
    if (BlockFitCheck(pivotX, pivotY, block)) {
        for (int[] coord : block) {
            int x = pivotX + coord[0];
            int y = pivotY + coord[1];

            matrix[x][y] = alphabet;
        }
        return true;
    }
    return false;
}

// mendapatkan warna teks
private static String getColoredText(String text, String colorCode) {
    return "\u001B[38;2;" + getRGBFromHex(colorCode) + "m" + text + "\u001B[0m";
}

public static String getRGBFromHex(String hex) {
    int r = Integer.valueOf(hex.substring(beginIndex:1, endIndex:3), radix:16);
    int g = Integer.valueOf(hex.substring(beginIndex:3, endIndex:5), radix:16);
    int b = Integer.valueOf(hex.substring(beginIndex:5, endIndex:7), radix:16);
    return r + ";" + g + ";" + b;
}

```

```

// Print matrix
public void printMatrix() {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            char cell = matrix[i][j];

            if (cell == ' ') {
                System.out.print(s: " ");
            } else {
                String hexColor = BlockInput.letterColors.getOrDefault(cell, defaultValue:"#FFFFFF");
                String rgbColor = getRGBFromHex(hexColor);
                System.out.print("\u001B[38;2;" + rgbColor + "m" + cell + "\u001B[0m ");
            }
        }
        System.out.println();
    }
}

```

```

public void saveMatrix(String filename, boolean solutionFound) {
    try (FileWriter writer = new FileWriter(filename)) {
        if (solutionFound) {
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    char elmt = matrix[i][j];
                    if (elmt == ' ') {
                        writer.write(c: ' ');
                    } else {
                        writer.write(elmt);
                    }
                }
                if (i < rows-1) {
                    writer.write(System.lineSeparator());
                }
            }
        } else {
            writer.write(str: "Tidak ada solusi");
        }
        System.out.println("Solusi disimpan sebagai: " + filename);
    } catch (IOException e) {
        System.out.println(x: "Error dalam menyimpan solusi");
        e.printStackTrace();
    }
}

public void saveMatrixAsImage(String filename) {
    int cellSize = 30;
    int width = cols * cellSize;
    int height = rows * cellSize;

    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics2D result = image.createGraphics();

    result.setColor(Color.WHITE);
    result.fillRect(x:0, y:0, width, height);
    result.setFont(new Font(name:"Arial", Font.BOLD, cellSize - 5));
    FontMetrics metrics = result.getFontMetrics();

    // tulis matriks
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            char cell = matrix[i][j];
            String hexColor = BlockInput.letterColors.getOrDefault(cell, defaultValue:"#000000");
            Color textColor = Color.decode(hexColor);

            result.setColor(textColor);

            int x = j * cellSize + (cellSize - metrics.charWidth(cell)) / 2;
            int y = (i + 1) * cellSize - (cellSize / 4);

            result.drawString(String.valueOf(cell), x, y);
        }
    }
}

```

```

public void saveMatrixAsImage(String filename) {
    int cellSize = 30;
    int width = cols * cellSize;
    int height = rows * cellSize;

    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics2D result = image.createGraphics();

    result.setColor(Color.WHITE);
    result.fillRect(x:0, y:0, width, height);
    result.setFont(new Font(name:"Arial", Font.BOLD, cellSize - 5));
    FontMetrics metrics = result.getFontMetrics();

    // tulis matriks
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            char cell = matrix[i][j];
            String hexColor = BlockInput.letterColors.getOrDefault(cell, defaultValue:"#000000");
            Color textColor = Color.decode(hexColor);

            result.setColor(textColor);

            int x = j * cellSize + (cellSize - metrics.charWidth(cell)) / 2;
            int y = (i + 1) * cellSize - (cellSize / 4);

            result.drawString(String.valueOf(cell), x, y);
        }
    }

    result.dispose();

    // Save
    try {
        ImageIO.write(image, formatName:"png", new File(filename));
        System.out.println("Matriks disimpan sebagai: " + filename);
    } catch (IOException e) {
        System.out.println(x:"Error dalam menyimpan gambar");
        e.printStackTrace();
    }
}

```

4. File: Pyramid.java

```
import java.util.List;
import java.io.FileWriter;
import java.io.IOException;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class Pyramid {
    public char[][][] pyramid;
    public int stacks;
    public int baseRows, baseCols;

    public Pyramid(int baseRows, int baseCols) {
        this.baseRows = baseRows;
        this.baseCols = baseCols;
        this.stacks = Math.min(baseRows, baseCols); // Jumlah stack/tingkat
        this.pyramid = new char[stacks][][];

        // Inisialisasi pyramid
        for (int stack = 0; stack < stacks; stack++) {
            int rows = baseRows - stack;
            int cols = baseCols - stack;
            pyramid[stack] = new char[rows][cols]; // ukuran tiap tingkat
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    pyramid[stack][i][j] = ' ';
                }
            }
        }
    }

    // mengecek apakah block fit di suatu stack
    public boolean BlockFitCheckPyramid(int bottomStack, int topStack, int pivotX, int pivotY, List<int[]> block) {
        for (int stack = bottomStack; stack <= topStack; stack++) {
            for (int[] coord : block) {
                int x = pivotX + coord[0];
                int y = pivotY + coord[1];

                if (x < 0 || x >= pyramid[stack].length || y < 0 || y >= pyramid[stack][0].length || pyramid[stack][x][y] != ' ') {
                    return false;
                }
            }
        }
        return true;
    }
}
```



```

public boolean addBlockPyramid(int bottomStack, int topStack, int pivotX, int pivotY, List<int[]> block, char alphabet) {
    if (BlockFitCheckPyramid(bottomStack, topStack, pivotX, pivotY, block)) {
        for (int stack = bottomStack; stack <= topStack; stack++) {
            for (int[] coord : block) {
                int x = pivotX + coord[0];
                int y = pivotY + coord[1];
                pyramid[stack][x][y] = alphabet;
            }
        }
        return true;
    }
    return false;
}

public void eraseBlockPyramid(int stack, int pivotX, int pivotY, List<int[]> block) {
    for (int[] coord : block) {
        int x = pivotX + coord[0];
        int y = pivotY + coord[1];
        pyramid[stack][x][y] = ' ';
    }
}

public void printPyramid() {
    for (int stack = 0; stack < stacks; stack++) {
        for (int i = 0; i < pyramid[stack].length; i++) {
            for (int j = 0; j < pyramid[stack][i].length; j++) {
                char cell = pyramid[stack][i][j];
                if (cell == ' ') {
                    System.out.print(s:" ");
                } else {
                    String hexColor = BlockInput.letterColors.getOrDefault(cell, defaultValue:"#FFFFFF");
                    String rgbColor = TransformMatrix.getRGBFromHex(hexColor);
                    System.out.print("\u001B[38;2;" + rgbColor + "m" + cell + "\u001B[0m ");
                }
            }
            System.out.println();
        }
    }
}

public boolean PyramidFullCheck() {
    for (int stack = 0; stack < stacks; stack++) {
        for (int i = 0; i < pyramid[stack].length; i++) {
            for (int j = 0; j < pyramid[stack][i].length; j++) {
                if (pyramid[stack][i][j] == ' ') {
                    return false;
                }
            }
        }
    }
    return true;
}

```

```

public void savePyramidMatrix(String filename, boolean solutionFound) {
    try (FileWriter writer = new FileWriter(filename)) {
        if (solutionFound) {
            for (int stack = 0; stack < stacks; stack++) {
                for (int row = 0; row < pyramid[stack].length; row++) {
                    for (int col = 0; col < pyramid[stack][row].length; col++) {
                        char elmt = pyramid[stack][row][col];
                        if (elmt == ' ') {
                            writer.write(c:' ');
                        } else {
                            writer.write(elmt);
                        }
                    }
                    writer.write(System.lineSeparator());
                }
            }
        } else {
            writer.write(str:"Tidak ada solusi");
        }

        System.out.println("Solusi disimpan sebagai: " + filename);
    } catch (IOException e) {
        System.out.println(x:"Error dalam menyimpan solusi");
        e.printStackTrace();
    }
}

public void savePyramidAsImage(String filename) {
    int size = 30;
    int width = pyramid[0][0].length * size;
    int height = 0;

    for (int stack = 0; stack < stacks; stack++) {
        height += pyramid[stack].length * size;
    }

    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics2D result = image.createGraphics();

    result.setColor(Color.WHITE);
    result.fillRect(x:0, y:0, width, height);
    result.setFont(new Font(name:"Arial", Font.BOLD, size - 5));
    FontMetrics metrics = result.getFontMetrics();

    int Y = 0;

    for (int stack = 0; stack < stacks; stack++) {
        int rows = pyramid[stack].length;
        int cols = pyramid[stack][0].length;

        for (int row = 0; row < rows; row++) {
            for (int col = 0; col < cols; col++) {
                char cell = pyramid[stack][row][col];
                String hexColor = BlockInput.letterColors.getOrDefault(cell, defaultValue:"#000000");
                Color textColor = Color.decode(hexColor);
            }
        }
    }
}

```

```

        result.setColor(textColor);

        int x = col * size + (size - metrics.charWidth(cell)) / 2;
        int y = Y + (row + 1) * size - (size / 4);

        result.drawString(String.valueOf(cell), x, y);
    }
    Y += rows * size;
}

result.dispose();

try {
    ImageIO.write(image, formatName:"png", new File(filename));
    System.out.println("Piramid disimpan sebagai: " + filename);
} catch (IOException e) {
    System.out.println(x:"Error dalam menyimpan gambar");
    e.printStackTrace();
}
}

```

5. File Main.java

```

import java.util.*;

public class Main {

    private static Pyramid pyramid;
    private static TransformMatrix matrix;
    private static List<List<String>> blocks;
    private static List<List<int[]>> blockCoordinates;
    private static boolean puzzleSolved = false;
    private static long iterations = 0;
    private static long startTime;

```

```

public static void main(String[] args) {
    // Baca input blocks
    blocks = BlockInput.ReadInputFromFile();
    if (blocks == null) {
        System.out.println(x:"Periksa file input.");
        return;
    }

    // Ambil koordinat dari tiap blok
    blockCoordinates = new ArrayList<>();
    for (List<String> block : blocks) {
        blockCoordinates.add(BlockInput.BlockToCoordinates(block));
    }

    pyramid = new Pyramid(BlockInput.M, BlockInput.N);
    matrix = new TransformMatrix(BlockInput.M, BlockInput.N);

    // bruteforce
    startTime = System.currentTimeMillis();
    if (BlockInput.Type.equals(anObject:"DEFAULT")) {
        solvePuzzle(blockIndex:0);
    } else {
        solvePuzzlePyramid(blockIndex:0);
    }

    long endTime = System.currentTimeMillis();
    long runtime = endTime - startTime;

    // solusi ditemukan (print)
    if (puzzleSolved) {
        System.out.println(x:"Solusi");
        System.out.println();

        if (BlockInput.Type.equals(anObject:"DEFAULT")) {
            matrix.printMatrix();
        } else {
            pyramid.printPyramid();
        }
    } else {
        System.out.println(x:"Tidak ditemukan solusi.");
    }

    System.out.println("\nWaktu eksekusi: " + runtime + " ms\n");
    System.out.println("Banyak kasus yang ditinjau: " + iterations);

    Scanner saveScanner = new Scanner(System.in);
    boolean userInputCorrect = false;
    boolean userInputImgCorrect = false;

```

```

while (!userInputCorrect) {
    System.out.print(s:"Apakah anda ingin menyimpan solusi? (ya/tidak): ");

    String userInput = saveScanner.nextLine().trim().toLowerCase();

    //simpan txt
    if (userInput.equals(anObject:"ya")) {
        System.out.print(s:"Masukkan nama file untuk menyimpan solusi: ");
        String saveFilename = "../test/" + saveScanner.nextLine().trim();
        if (BlockInput.Type.equals(anObject:"DEFAULT")) {
            matrix.saveMatrix(saveFilename + ".txt", puzzleSolved);
        } else {
            pyramid.savePyramidMatrix(saveFilename + ".txt", puzzleSolved);
        }
        userInputCorrect = true;

        //simpan gambar (png)
        if (puzzleSolved) {
            while (!userInputImgCorrect) {
                System.out.print(s:"Apakah anda ingin menyimpan solusi sebagai gambar? (ya/tidak): ");
                String userInputImg = saveScanner.nextLine().trim().toLowerCase();

                if (userInputImg.equals(anObject:"ya")) {
                    System.out.print(s:"Masukkan nama file untuk menyimpan solusi gambar: ");
                    String saveFilenameImg = "../test/" + saveScanner.nextLine().trim();
                    if (BlockInput.Type.equals(anObject:"DEFAULT")) {
                        matrix.saveMatrixAsImage(saveFilenameImg + ".png");
                    } else {
                        pyramid.savePyramidAsImage(saveFilenameImg + ".png");
                    }

                    userInputImgCorrect = true;
                }

                else if (userInputImg.equals(anObject:"tidak")) {
                    System.out.println(x:"Solusi tidak disimpan sebagai gambar.");
                    userInputImgCorrect = true;
                } else {
                    System.out.println(x:"Input tidak valid. Harap masukkan 'ya' atau 'tidak'.");
                }
            }
        }
    }

    else if (userInput.equals(anObject:"tidak")) {
        System.out.println(x:"Solusi tidak disimpan.");
        userInputCorrect = true;
    } else {
        System.out.println(x:"Input tidak valid. Harap masukkan 'ya' atau 'tidak'.");
    }
}
}

```

```

private static boolean BoardFullCheck() {
    for (int i = 0; i < matrix.rows; i++) {
        for (int j = 0; j < matrix.cols; j++) {
            if (matrix.matrix[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}

private static void solvePuzzle(int blockIndex) {
    if (blockIndex >= blockCoordinates.size()) {
        if (BoardFullCheck()) {
            puzzleSolved = true;
        }
        return;
    }

    List<int[]> currentBlock = blockCoordinates.get(blockIndex);
    char currentLetter = BlockInput.blockLetters.get(blockIndex);

    // Coba setiap posisi dan rotasi
    for (int rotation = 0; rotation < 4; rotation++) {
        List<int[]> rotatedBlock = Rotate.RotateBlocks(currentBlock, rotation);

        for (int x = 0; x < matrix.rows; x++) {
            for (int y = 0; y < matrix.cols; y++) {
                iterations++;
                if (matrix.BlockFitCheck(x, y, rotatedBlock)) {
                    matrix.addBlock(x, y, rotatedBlock, currentLetter);

                    solvePuzzle(blockIndex + 1);

                    if (puzzleSolved) {
                        return;
                    }

                    // backtrack jika gagal
                    matrix.eraseBlock(x, y, rotatedBlock);
                }
            }
        }
    }
}

```

```

private static void solvePuzzlePyramid(int blockIndex) {
    if (blockIndex >= blockCoordinates.size()) {
        if (pyramid.PyramidFullCheck()) {
            puzzleSolved = true;
        }
        return;
    }

    List<int[]> currentBlock = blockCoordinates.get(blockIndex);
    char currentLetter = BlockInput.blockLetters.get(blockIndex);

    // coba setiap posisi dan rotasi 3D
    for (int rotation = 0; rotation < 8; rotation++) {
        List<int[]> rotatedBlock = Rotate.RotateBlocksBy45(currentBlock, rotation);

        for (int bottomStack = 0; bottomStack < pyramid.stacks; bottomStack++) {
            for (int endStack = bottomStack; endStack < pyramid.stacks; endStack++) {
                for (int x = 0; x < pyramid.pyramid[bottomStack].length; x++) {
                    for (int y = 0; y < pyramid.pyramid[bottomStack][x].length; y++) {
                        iterations++;
                        if (pyramid.BlockFitCheckPyramid(bottomStack, endStack, x, y, rotatedBlock)) {
                            pyramid.addBlockPyramid(bottomStack, endStack, x, y, rotatedBlock, currentLetter);
                            solvePuzzlePyramid(blockIndex + 1);

                            if (puzzleSolved) {
                                return;
                            }
                            // Backtrack
                            pyramid.eraseBlockPyramid(bottomStack, x, y, rotatedBlock);
                        }
                    }
                }
            }
        }
    }
}
}

```

BAB III

SCREENSHOT HASIL TEST

Pada program ini, input dan output terletak pada folder test.

1. Input: input2.txt

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

Output: terminal, sol2.txt, sol2.png

```
Masukkan nama file: input2.txt
Solusi

A G G G C
A A B C C
E E B B F
E E D F F
E D D F F

Waktu eksekusi: 18 ms

Banyak kasus yang ditinjau: 418857
```

```
AGGGC
AABCC
EEBBF
EEDFF
EDDFF
```

```
A G G G C
A A B C C
E E B B F
E E D F F
E D D F F
```


2. Input: input1.txt

```
3 3 6
PYRAMID
A
AA
BB
CC
DDD
E
EE
F
```

Output: terminal

```
Masukkan nama file: input1.txt
Solusi

F
C C
E E
A B B
A A F
D D D

Waktu eksekusi: 28 ms

Banyak kasus yang ditinjau: 409394
```

3. Input: input3.txt

```
2 3 3
DEFAULT
AA
CC
BBB
```

Output: terminal

```

Masukkan nama file: input3.txt
Tidak ditemukan solusi.

Waktu eksekusi: 2 ms

Banyak kasus yang ditinjau: 2472
Apakah anda ingin menyimpan solusi? (ya/tidak): tidak
Solusi tidak disimpan.

```

4. Input: input4.txt

```

5 26 26
DEFAULT
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE
FFFFF
GGGGG
HHHHH
IIIII
JJJJJ
KKKKK
LLLLL
MMMMM
NNNNN
OOOOO
PPPPP
QQQQQ
RRRRR
SSSSS
TTTTT
UUUUU
VVVVV
WWWWW
XXXXX
YYYYY
ZZZZZ

```

Output: terminal, sol4.png

```

Masukkan nama file: input4.txt
Solusi

A A A A A B B B B B C C C C C D D D D D E E E E E Z
F F F F F G G G G G H H H H H I I I I I J J J J J Z
K K K K K L L L L L M M M M M N N N N N O O O O O Z
P P P P P Q Q Q Q Q R R R R R S S S S S T T T T T Z
U U U U U V V V V V W W W W W X X X X X Y Y Y Y Y Z

Waktu eksekusi: 1 ms

```

```

A A A A A B B B B B C C C C C D D D D D E E E E E Z
F F F F F G G G G G H H H H H I I I I I J J J J J Z
K K K K K L L L L L M M M M M N N N N N O O O O O Z
P P P P P Q Q Q Q Q R R R R R S S S S S T T T T T Z
U U U U U V V V V V W W W W W X X X X X Y Y Y Y Y Z

```

5. Input: input5.txt

```
6 6 9
DEFAULT
A
A
AAAAA
C
CC
DD
D
D
EE
EE
B
F
FF
G
HH
      I
      I
      I
      I
      I
IIIIII
```

Output: terminal

```
Masukkan nama file: input5.txt
Solusi

B A C F F I
G A C C F I
A A A A A I
D H H E E I
D D D E E I
I I I I I I

Waktu eksekusi: 252753 ms

Banyak kasus yang ditinjau: 20291068698
```

6. Input: input100.txt (tidak ada)

Output: terminal

```
Masukkan nama file: input100.txt
File tidak ditemukan. Pastikan nama file ditambah dengan .txt
Periksa file input.
```

7. Input: input7.txt

```
4 4 5
AAAA
BBBB
CCCC
DDDD
```

Output: terminal

```
Masukkan nama file: input7.txt
Error: Tipe harus merupakan 'DEFAULT' atau 'PYRAMID'.
Periksa file input.
```

8. Input: input8.txt

```
1 0 0
DEFAULT
AAAAA|
```

Output: terminal

```
Masukkan nama file: input8.txt
Error: Dimensi matriks harus lebih besar dari 0.
Periksa file input.
```

9. Input: input9.txt

```
3 3 A
PYRAMID
AAA
BBB
CC
CC
D
```

Output: terminal

```
Masukkan nama file: input9.txt
Error: Baris pertama harus berisi 3 bilangan bulat.
Periksa file input.
```

10. Input10 : input10.txt (kosong)

Output:

```
Masukkan nama file: input10.txt
Error: Baris pertama harus berisi 3 bilangan bulat.
Periksa file input.
```

11. Input: input11.txt

```
2 2 1
DEFAULT
aa
aa
```

Output:terminal

```
Masukkan nama file: input11.txt
Error: Terdapat karakter yang tidak valid pada file. Hanya A-Z dan spasi yang diperbolehkan.
Periksa file input.
```

12. Input: input12.txt

```
5 5 5
DEFAULT
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE
FFFFF
```

Output:terminal

```
Masukkan nama file: input12.txt
Error: Jumlah blok tidak sesuai dengan jumlah blok (5).
Periksa file input.
```

LINK REPOSITORY

https://github.com/henry204xx/Tucil1_13523108

CHECKLIST

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 <i>Graphical User Interface</i> (GUI)		✓
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	
7	Program dapat menyelesaikan kasus konfigurasi <i>custom</i>		✓
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)	✓	
9	Program dibuat oleh saya sendiri	✓	