

IF2211 Strategi Algoritma

Penyelesaian IQ Puzzle Pro menggunakan Algoritma Brute-Force

Laporan Tugas Kecil 1

Disusun sebagai pemenuhan tugas kecil 1 IF2211 Strategi Algoritma Tahun
Akademik 2024/2025 Semester 2



Disusun oleh:

Jovandra Otniel P. S. (13523141)

Daftar Isi

BAB I: DESKRIPSI MASALAH	3
BAB II: LANDASAN TEORI	4
BAB III: SPESIFIKASI TUGAS	5
BAB IV: IMPLEMENTASI ALGORITMA	7
BAB V: KODE SUMBER	8
BAB VI: PENGUJIAN ALGORITMA	20
BAB VII: KESIMPULAN DAN SARAN	23
Referensi	23
Lampiran	24

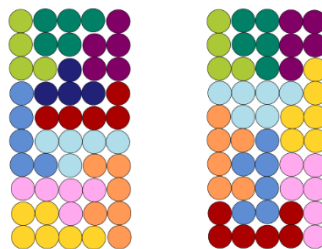
BAB I: DESKRIPSI MASALAH

IQ Puzzle Solver adalah sebuah permainan papan yang terdiri atas sebuah papan persegi panjang berukuran $m \times n$, yang pada awalnya dimulai dalam kondisi kosong. Puzzle ini terdiri dari 2 komponen utama, yaitu potongan dan papan. Papan puzzle menjadi komponen utama yang menampung seluruh potongan puzzle yang diisi oleh pemain. Sedangkan potongan puzzle adalah komponen pengisi papan puzzle yang harus dipakai pengguna hingga papan penuh dan tidak ada potongan yang tersisa.

Dalam permainan ini, puzzle memiliki warna muka depan dan belakang yang sama, sehingga setiap potongan tidak hanya dapat diputar (*rotate*), tetapi juga dapat dicerminkan (dibalik). Hal ini tentunya berbeda dengan puzzle jigsaw, dimana muka depan dan belakang berbeda, sehingga tidak dapat dibalik.

Potongan-potongan puzzle ini memiliki desain yang memungkinkan mereka untuk diputar dan dicerminkan, sehingga satu potongan dapat menghasilkan beberapa orientasi berbeda. Misalnya, sebuah potongan dengan bentuk awal tertentu dapat diubah dengan melakukan rotasi 90 derajat atau dicerminkan secara horizontal, menghasilkan bentuk baru yang tetap mempertahankan struktur dasar namun dengan orientasi yang berbeda. Variasi orientasi ini memberikan sejumlah kemungkinan penempatan yang beraneka, sehingga tata letak akhir papan dapat tercapai melalui berbagai kombinasi potongan.

Aturan permainannya sederhana. Pemain hanya perlu memasukkan semua potongan puzzle ke dalam papan sampai penuh dan tidak ada potongan tersisa. Namun, bagian paling menantang dari permainan ini adalah cara pemain mengatur letak dan orientasi setiap potongan agar semuanya dapat masuk ke dalam papan tersebut.



Gambar 1: IQ Puzzle

(sumber: [wikimedia.org](https://www.wikimedia.org))

BAB II: LANDASAN TEORI

Bab ini membahas dasar teoretis dari dua teknik utama yang sering digunakan untuk menyelesaikan masalah kombinatorial, yaitu algoritma brute force dan metode backtracking. Algoritma brute force merupakan pendekatan yang secara eksplisit menguji setiap kandidat solusi yang mungkin, tanpa mengandalkan optimasi atau strategi pemangkasan. Dalam metode ini, seluruh ruang solusi dicoba secara menyeluruh sehingga, apabila solusi ada, maka ia akan ditemukan. Pendekatan ini, meskipun sederhana, menjamin bahwa tidak ada kemungkinan yang terlewatkan.

Pada implementasi brute force, setiap kandidat solusi dihasilkan dan diuji satu per satu. Proses ini diibaratkan dengan mencoba setiap kombinasi angka untuk membuka sebuah brankas, di mana setiap kombinasi dicoba hingga kombinasi yang benar ditemukan. Namun, seiring bertambahnya kompleksitas masalah, terutama ketika setiap elemen dalam solusi memiliki beberapa kemungkinan transformasi, jumlah kandidat yang harus diuji dapat menjadi sangat besar. Misalnya, ketika sebuah elemen dapat mengalami rotasi atau pencerminan, maka variasi bentuknya bertambah banyak secara pesat. Hal ini menambah kompleksitas ruang solusi yang harus dieksplorasi oleh algoritma brute-force.

Untuk mengatasi masalah waktu komputasi yang tinggi akibat eksplorasi seluruh ruang solusi, teknik backtracking dikemukakan sebagai sebuah strategi penghematan sumber daya eksekutor algoritma (komputer) dan waktu. Backtracking merupakan metode rekursif yang bekerja dengan prinsip “coba dan mundur”. Pada setiap tahap pencarian, jika ditemukan bahwa suatu jalur tidak dapat menghasilkan solusi yang valid, algoritma secara otomatis kembali ke langkah sebelumnya untuk mencoba alternatif lain. Proses ini memungkinkan pemotongan cabang-cabang pencarian yang tidak produktif, sehingga tidak seluruh ruang solusi harus dieksplorasi secara lengkap. Penelitian oleh Korf (1985) menunjukkan bahwa penerapan backtracking dalam pencarian solusi dapat secara signifikan mengurangi jumlah iterasi yang diperlukan, asalkan mekanisme pengembalian diterapkan dengan tepat.

Kombinasi antara brute-force dan backtracking sering kali digunakan untuk memastikan bahwa setiap kemungkinan kandidat solusi tetap diperhitungkan, namun dengan pendekatan yang lebih terstruktur. Teknik ini memungkinkan algoritma untuk menelusuri ruang solusi secara sistematis, dengan tetap memberikan kesempatan untuk mengembalikan dan menguji jalur alternatif jika suatu konfigurasi tidak memenuhi kriteria solusi. Studi oleh Korf (1985) mengemukakan bahwa metode backtracking dapat memotong ruang pencarian secara efektif dengan mengeliminasi jalur-jalur yang tidak mungkin menghasilkan solusi yang valid.

BAB III: SPESIFIKASI TUGAS

- Buatlah program sederhana dalam bahasa **Java** yang mengimplementasikan *algoritma Brute Force* untuk mencari solusi dalam permainan IQ Puzzler Pro.
 - Algoritma brute force yang diimplementasikan harus bersifat “**murni**”, tidak boleh memanfaatkan heuristik.
 - Papan yang perlu diisi mulanya akan selalu kosong.
 - Sebuah blok puzzle bisa saja dirotasi maupun dicerminkan sebelum diletakan pada papan.
 - **Input:** program akan memberikan pengguna sebuah prompt untuk memilih file *test case* berekstensi **.txt**, kemudian program membaca file *test case* tersebut yang berisi 1. **Dimensi Papan** terdiri atas dua buah variabel **N** dan **M** yang membentuk papan berdimensi NxM.
2. **Banyak blok puzzle** direpresentasikan oleh variabel integer **P**.
 3. **Jenis kasus** sebuah variabel string **S** yang digunakan untuk mengidentifikasi kasus konfigurasi, hanya mungkin bernilai salah satu diantara **DEFAULT/CUSTOM/PYRAMID**.
 4. **Bentuk blok puzzle** yang dilambangkan oleh konfigurasi *Character* berupa huruf. Akan ada **P** buah blok puzzle berbeda yang dibentuk oleh **P buah huruf berbeda**. *Character* yang digunakan adalah huruf **A-Z dalam kapital**.

File .txt yang akan dibaca memiliki format sebagai berikut

Contoh Test case: (input.txt)

```
N M P
S
puzzle_1_shape
puzzle_2_shape ...
puzzle_P_shape
```

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

● Output:

```
AGGGD
AABDD
CCBBF
CEEFF
EEEFF
Waktu pencarian: 604 ms
Banyak kasus yang ditinjau: 7387
Apakah Anda ingin menyimpan
solusi? (ya/tidak)
```

1. Tampilkan **konfigurasi blok puzzle** yang berhasil mengisi papan. **Gunakan print berwarna** untuk menunjukkan blok puzzle dengan jelas. Pastikan setiap blok puzzle berbeda memiliki warna berbeda. Beri tahu pengguna apabila puzzle tidak memiliki solusi.
2. **Waktu eksekusi** program dalam *milisecond* (tidak termasuk waktu membaca masukan dan menyimpan solusi, cukup waktu pencarian oleh algoritma).
3. **Banyak kasus** atau jumlah iterasi yang ditinjau oleh algoritma brute force.
4. **Prompt untuk menyimpan solusi** dalam sebuah berkas berekstensi **.txt** (Struktur untuk file output dibebaskan).

BAB IV: IMPLEMENTASI ALGORITMA

Program ini dibangun dengan pendekatan modular dengan memisahkan tiga komponen utama: kelas `PuzzlePiece`, kelas `PuzzleBoard`, dan kelas `PuzzleSolver`. Setiap kelas memiliki peran khusus yang saling mendukung dalam mengimplementasikan algoritma brute force dengan backtracking untuk mengisi papan secara lengkap.

Pada kelas `PuzzlePiece`, setiap potongan puzzle individu dimodelkan secara terpisah. Kelas ini menerima bentuk awal potongan sebagai input, yang umumnya berupa deretan string dengan karakter tertentu, dan mengubahnya menjadi kumpulan koordinat relatif. Proses normalisasi dilakukan agar titik acuan—yaitu titik ter kiri-atas—selalu berada pada koordinat (0,0). Selain itu, kelas `PuzzlePiece` memiliki metode untuk menghasilkan semua orientasi unik dari potongan tersebut melalui rotasi (misalnya 90°, 180°, 270°) dan pencerminan. Dengan demikian, setiap potongan dapat direpresentasikan dalam beberapa konfigurasi, yang nantinya akan diuji saat penempatan di papan.

Kelas `PuzzleBoard` bertanggung jawab untuk merepresentasikan papan permainan itu sendiri. Papan disusun sebagai matriks dua dimensi, di mana setiap sel diinisialisasi dengan karakter penanda kekosongan, misalnya titik ('.'). Kelas ini menyediakan fungsi-fungsi untuk mengecek validitas penempatan potongan, seperti memastikan bahwa seluruh koordinat hasil transformasi potongan berada di dalam batas papan dan tidak terjadi tumpang tindih dengan potongan lain yang sudah ditempatkan. Selain itu, kelas `PuzzleBoard` juga menangani tampilan akhir papan, termasuk penerapan warna pada masing-masing potongan untuk memudahkan identifikasi saat output ditampilkan.

Kelas `PuzzleSolver` merupakan inti dari algoritma yang menggabungkan metode brute force dengan backtracking. Kelas ini mengandung sejumlah metode, di antaranya fungsi rekursif untuk mencari sel kosong pada papan dan mencoba menempatkan potongan-potongan yang masih tersedia pada posisi tersebut. Untuk setiap sel kosong yang ditemukan, `PuzzleSolver` akan mengiterasi seluruh potongan yang belum digunakan dan menguji setiap orientasi yang telah dihasilkan oleh kelas `PuzzlePiece`. Apabila penempatan potongan tersebut valid menurut pengecekan pada kelas `PuzzleBoard`, maka potongan akan ditempatkan dan algoritma akan melanjutkan pencarian ke sel kosong berikutnya secara rekursif. Jika suatu titik ditemukan bahwa tidak ada penempatan yang memungkinkan penyelesaian, mekanisme backtracking akan diaktifkan dengan menghapus penempatan terakhir dan mencoba alternatif lain. Selama proses ini, `PuzzleSolver` juga mencatat jumlah iterasi yang dilakukan dan waktu eksekusi, sehingga memberikan gambaran tentang performa algoritma dalam mengeksplorasi ruang solusi.

BAB V: KODE SUMBER

- Main.java

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

public class Main {
    public static void main(String[] args) {
        List<PuzzlePiece> pieces = new ArrayList<>();
        try (Scanner consoleScanner = new Scanner(System.in)) {
            System.out.print("Masukkan path file test case pada folder test (.txt): ");
            String filePath = consoleScanner.nextLine().trim();

            try (Scanner fileScanner = new Scanner(new File("../test/" + filePath))) {
                // Baca dimensi papan dan jumlah blok puzzle
                int boardRows = fileScanner.nextInt();
                int boardCols = fileScanner.nextInt();
                int P = fileScanner.nextInt();
                fileScanner.nextLine(); // konsumsi baris sisa

                // Baca tipe konfigurasi (DEFAULT/CUSTOM/PYRAMID)
                String configType = fileScanner.nextLine().trim();
                System.out.println("Konfigurasi: " + configType);

                PuzzleBoard puzzleBoard;
                if (configType.equalsIgnoreCase("CUSTOM")) {
                    // Baca custom layout (NxM matrix)
                    String[] layout = new String[boardRows];
                    for (int i = 0; i < boardRows; i++) {
                        layout[i] = fileScanner.nextLine();
                    }
                    // Changed from PuzzleBoardCustom to PuzzleBoard
                    puzzleBoard = new PuzzleBoard(layout);
                } else {
                    // Rectangle board initialization
                    puzzleBoard = new PuzzleBoard(boardRows, boardCols);
                }

                // Baca seluruh baris yang tersisa ke dalam list
                List<String> remainingLines = new ArrayList<>();
                while (fileScanner.hasNextLine()) {
                    String line = fileScanner.nextLine();
                    if (!line.trim().isEmpty()) {
                        remainingLines.add(line);
                    }
                }
            }
        }
    }
}
```



```

    }

    // Baca blok puzzle satu per satu dari list tersebut.
    boolean[] used = new boolean[P];
    int pointer = 0;
    for (int i = 0; i < P; i++) {
        char expectedChar = (char) ('A' + i);
        List<String> shapeLines = new ArrayList<>();
        // Ambil baris selama huruf awal sesuai dengan
expectedChar
        while (pointer < remainingLines.size()
&& !remainingLines.get(pointer).trim().isEmpty() &&
remainingLines.get(pointer).trim().charAt(0) == expectedChar) {
            shapeLines.add(remainingLines.get(pointer));
            pointer++;
        }
        if (shapeLines.isEmpty()) {
            System.out.println("Tidak ditemukan bentuk untuk blok
" + expectedChar);
            return;
        }
        PuzzlePiece piece = new PuzzlePiece(expectedChar,
shapeLines);
        pieces.add(piece);
    }

    // Tampilkan informasi input
    System.out.println("Ukuran papan: " + boardRows + "x" +
boardCols);
    System.out.println("Banyak blok puzzle: " + P);

    // Mulai waktu pencarian algoritma
    long startTime = System.currentTimeMillis();
    boolean solved = PuzzleSolver.solve(puzzleBoard, pieces,
used);
    long endTime = System.currentTimeMillis();
    long searchTimeMs = endTime - startTime;

    if (solved) {
        System.out.println("\nSolusi ditemukan:\n");
        puzzleBoard.printBoardColored(pieces);
        System.out.println("\nWaktu pencarian: " + searchTimeMs +
" ms");
        System.out.println("Banyak kasus yang ditinjau: " +
PuzzleSolver.casesTried);
    } else {
        System.out.println("\nWaktu pencarian: " + searchTimeMs +
" ms");
    }
}

```

```

        System.out.println("Banyak kasus yang ditinjau: " +
PuzzleSolver.casesTried);
        System.out.println("Tidak ada solusi.");
    }

    // Tanyakan apakah solusi ingin disimpan
    System.out.print("\nApakah anda ingin menyimpan solusi?
(ya/tidak): ");
    String response =
consoleScanner.nextLine().trim().toLowerCase();
    if (response.equals("ya")) {
        System.out.print("Masukkan nama file output (.txt): ");
        String outFile = consoleScanner.nextLine();
        puzzleBoard.generateImageResult("../test/" + outFile +
".jpg", pieces);
        PuzzleSolver.saveSolution("../test/" + outFile,
puzzleBoard, searchTimeMs, PuzzleSolver.casesTried);
        System.out.println("Solusi telah disimpan pada folder test
dengan nama " + outFile);
    }

    } catch (FileNotFoundException e) {
        System.out.println("File tidak ditemukan: " + filePath);
    }
    } catch (Exception e) {
        System.out.println("Terjadi kesalahan: " + e.getMessage());
    }
}
}

```

Point.java

```

// Titik Koordinat
public class Point {
    int r, c;
    public Point(int r, int c) {
        this.r = r;
        this.c = c;
    }
}

```

PuzzleBoard.java

```

import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class PuzzleBoard {
    // ANSI color codes untuk output berwarna (maksimal 26 warna)
    static final String[] ANSI_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[37m", "\u001B[90m", "\u001B[31m", "\u001B[32m",
        "\u001B[33m", "\u001B[34m", "\u001B[35m", "\u001B[36m",
        "\u001B[91m", "\u001B[92m", "\u001B[93m", "\u001B[94m",
        "\u001B[95m", "\u001B[96m"
    };
    static final String ANSI_RESET = "\u001B[0m";

    int boardRows, boardCols;
    char[][] board;

    public PuzzleBoard(int rows, int cols) {
        boardRows = rows;
        boardCols = cols;
        board = new char[boardRows][boardCols];
        for (int i = 0; i < boardRows; i++) {
            for (int j = 0; j < boardCols; j++) {
                board[i][j] = '.';
            }
        }
    }

    // Konstruktor alternatif untuk konfigurasi CUSTOM
    public PuzzleBoard(String[] layout) {
        boardRows = layout.length;
        boardCols = layout[0].length();
        board = new char[boardRows][boardCols];
        for (int i = 0; i < boardRows; i++) {
            for (int j = 0; j < boardCols; j++) {
                // 'X' artinya sel kosong, '#' artinya sel terisi
                board[i][j] = (layout[i].charAt(j) == 'X') ? '.' : '#';
            }
        }
    }

    public int getRows() {

```

```

        return boardRows;
    }
    public int getCols() {
        return boardCols;
    }
    public char getCell(int r, int c) {
        return board[r][c];
    }
    public void setCell(int r, int c, char val) {
        board[r][c] = val;
    }

    public void printBoardColored(List<PuzzlePiece> pieces) {
        Map<Character, String> colorMap = new HashMap<>();
        for (PuzzlePiece piece : pieces) {
            int index = piece.id - 'A';
            colorMap.put(piece.id, ANSI_COLORS[index % ANSI_COLORS.length]);
        }
        for (int i = 0; i < boardRows; i++) {
            for (int j = 0; j < boardCols; j++) {
                char ch = board[i][j];
                if (ch != '.') {
                    String color = colorMap.getOrDefault(ch, "");
                    System.out.print(color + ch + ANSI_RESET);
                } else {
                    System.out.print(ch);
                }
            }
            System.out.println();
        }
    }

    // New method to generate an image result (JPG) with colored cells like an
    // IQ puzzle.
    public void generateImageResult(String outputFile, List<PuzzlePiece>
pieces) {
        // Create a mapping from piece id to a Color using a predefined
        // palette.
        Map<Character, Color> colorMap = new HashMap<>();
        Color[] palette = {
            Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW,
            Color.ORANGE, Color.MAGENTA, Color.CYAN, Color.PINK,
            new Color(128, 0, 0), new Color(0, 128, 0), new Color(0, 0, 128),
            new Color(128, 128, 0), new Color(128, 0, 128), new Color(0, 128,
128),
            Color.LIGHT_GRAY, Color.GRAY, Color.DARK_GRAY, new Color(255, 105,
180),

```

```

        new Color(255, 20, 147), new Color(221, 160, 221), new Color(102,
205, 170),
        new Color(95, 158, 160), new Color(186, 85, 211), new Color(240,
128, 128),
        new Color(144, 238, 144), new Color(173, 216, 230)
    };
    for (PuzzlePiece piece : pieces) {
        int index = piece.id - 'A';
        colorMap.put(piece.id, palette[index % palette.length]);
    }

    int cellSize = 50; // The size of each puzzle cell in pixels.
    int imgWidth = boardCols * cellSize;
    int imgHeight = boardRows * cellSize;

    BufferedImage image = new BufferedImage(imgWidth, imgHeight,
BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = image.createGraphics();

    // Fill background with white.
    g2d.setColor(Color.WHITE);
    g2d.fillRect(0, 0, imgWidth, imgHeight);

    // Draw each cell.
    for (int i = 0; i < boardRows; i++) {
        for (int j = 0; j < boardCols; j++) {
            char ch = board[i][j];
            int x = j * cellSize;
            int y = i * cellSize;
            if (ch != '.' && ch != '#') {
                // Draw the puzzle piece with its corresponding color.
                Color color = colorMap.get(ch);
                if (color == null) {
                    color = Color.LIGHT_GRAY;
                }
                g2d.setColor(color);
                g2d.fillRect(x, y, cellSize, cellSize);
            } else {
                // For barriers ('#') draw black; for empty cells draw
white.

                if (ch == '#') {
                    g2d.setColor(Color.BLACK);
                    g2d.fillRect(x, y, cellSize, cellSize);
                } else {
                    g2d.setColor(Color.WHITE);
                    g2d.fillRect(x, y, cellSize, cellSize);
                }
            }
        }
    }
}

```

```

        // Draw cell border.
        g2d.setColor(Color.BLACK);
        g2d.drawRect(x, y, cellSize, cellSize);
    }
}

g2d.dispose();

// Save the image as a JPG file.
try {
    ImageIO.write(image, "jpg", new File(outputFile));
    System.out.println("Hasil image telah disimpan di folder test
dengan nama " + outputFile);
} catch (IOException e) {
    System.out.println("Error menyimpan file gambar: " +
e.getMessage());
}
}
}

```

PuzzlePiece.java

```

import java.util.*;
import java.util.function.Consumer;

// Kelas untuk merepresentasikan blok puzzle
public class PuzzlePiece {
    char id; // misalnya 'A', 'B', dst.
    // Setiap orientasi berupa list titik relatif (setelah dinormalisasi
    sehingga titik terkiri-atas = (0,0))
    List<List<Point>> orientations;

    // Konstruktor menerima blok puzzle dalam bentuk list baris (string) yang
    merupakan konfigurasi
    public PuzzlePiece(char id, List<String> shapelines) {
        this.id = id;
        // Dapatkan list titik dari konfigurasi
        List<Point> baseShape = new ArrayList<>();
        for (int i = 0; i < shapelines.size(); i++) {
            String line = shapelines.get(i);
            for (int j = 0; j < line.length(); j++) {
                if (line.charAt(j) == id) {
                    baseShape.add(new Point(i, j));
                }
            }
        }
        // Normalisasi agar titik terkiri-atas = (0,0)
    }
}

```

```

        baseShape = normalize(baseShape);
        // Hasilkan seluruh orientasi unik (rotasi dan cermin)
        orientations = generateOrientations(baseShape);
    }

    // Normalisasi: geser semua titik sehingga minimal r dan c = 0
    private List<Point> normalize(List<Point> pts) {
        int minR = Integer.MAX_VALUE, minC = Integer.MAX_VALUE;
        for (Point p : pts) {
            if (p.r < minR) minR = p.r;
            if (p.c < minC) minC = p.c;
        }
        List<Point> norm = new ArrayList<>();
        for (Point p : pts) {
            norm.add(new Point(p.r - minR, p.c - minC));
        }
        return norm;
    }

    // Menghasilkan semua orientasi (rotasi dan cermin) secara unik
    private List<List<Point>> generateOrientations(List<Point> base) {
        Set<String> seen = new HashSet<>();
        List<List<Point>> result = new ArrayList<>();

        // Fungsi untuk menambahkan orientasi jika belum ada (menggunakan
signature)
        Consumer<List<Point>> addIfUnique = (List<Point> shape) -> {
            List<Point> norm = normalize(shape);
            String sig = signature(norm);
            if (!seen.contains(sig)) {
                seen.add(sig);
                result.add(norm);
            }
        };

        // Tambahkan rotasi dari bentuk dasar
        List<Point> current = base;
        for (int i = 0; i < 4; i++) {
            addIfUnique.accept(current);
            current = rotate90(current);
        }

        // Mendapatkan cermin (flip horizontal) dari bentuk dasar
        List<Point> mirrored = mirror(base);
        current = mirrored;
        for (int i = 0; i < 4; i++) {
            addIfUnique.accept(current);
            current = rotate90(current);
        }
    }

```

```

        return result;
    }

    // Membuat signature string dari list titik (digunakan untuk mengecek
    // duplikasi)
    private String signature(List<Point> pts) {
        List<String> list = new ArrayList<>();
        for (Point p : pts) {
            list.add(p.r + "_" + p.c);
        }
        Collections.sort(list);
        return String.join(",", list);
    }

    // Rotasi 90° searah jarum jam
    private List<Point> rotate90(List<Point> pts) {
        int maxR = 0, maxC = 0;
        for (Point p : pts) {
            if (p.r > maxR) maxR = p.r;
            if (p.c > maxC) maxC = p.c;
        }
        List<Point> rotated = new ArrayList<>();
        for (Point p : pts) {
            rotated.add(new Point(p.c, maxR - p.r));
        }
        return rotated;
    }

    // Mirror secara horizontal (flip terhadap sumbu vertikal)
    private List<Point> mirror(List<Point> pts) {
        int maxC = 0;
        for (Point p : pts) {
            if (p.c > maxC) maxC = p.c;
        }
        List<Point> mirrored = new ArrayList<>();
        for (Point p : pts) {
            mirrored.add(new Point(p.r, maxC - p.c));
        }
        return mirrored;
    }
}

```

PuzzleSolver.java

```

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

```



```

public class PuzzleSolver {
    public static int casesTried = 0;
    // Metode rekursif untuk pencarian solusi dengan backtracking (brute force
    murni dengan backtracking)
    public static boolean solve(PuzzleBoard puzzleBoard, List<PuzzlePiece>
pieces, boolean[] used) {
        int[] pos = findEmpty(puzzleBoard);
        if (pos == null) {
            // Papan penuh, solusi ditemukan
            return true;
        }
        int r = pos[0], c = pos[1];

        // Coba setiap blok puzzle yang belum terpakai
        for (int i = 0; i < pieces.size(); i++) {
            if (!used[i]) {
                PuzzlePiece piece = pieces.get(i);
                // Coba setiap orientasi dari blok
                for (List<Point> orient : piece.orientations) {
                    // Karena kita ingin menutupi sel (r, c), coba setiap
titik pada orientasi sebagai "anchor"
                    for (Point anchor : orient) {
                        int offsetR = r - anchor.r;
                        int offsetC = c - anchor.c;
                        casesTried++; // hitung tiap kali kita mencoba
penempatan

                        if (canPlace(puzzleBoard, orient, offsetR, offsetC)) {
                            placePiece(puzzleBoard, piece.id, orient, offsetR,
offsetC);

                            used[i] = true;
                            if (solve(puzzleBoard, pieces, used)) {
                                return true;
                            }
                            // Backtrack
                            removePiece(puzzleBoard, orient, offsetR,
offsetC);

                            used[i] = false;
                        }
                    }
                }
            }
        }
        return false;
    }

    // Cari sel kosong pertama pada papan (diindeks dengan '.')
    public static int[] findEmpty(PuzzleBoard puzzleBoard) {

```

```

        for (int i = 0; i < puzzleBoard.getRows(); i++) {
            for (int j = 0; j < puzzleBoard.getCols(); j++) {
                if (puzzleBoard.getCell(i,j) == '.') {
                    return new int[]{i, j};
                }
            }
        }
        return null;
    }

    // Cek apakah blok puzzle dapat diletakkan pada papan pada posisi tertentu
    // Tanda '#' diabaikan, '.' dianggap kosong, karakter lain dianggap sudah
    terisi
    public static boolean canPlace(PuzzleBoard puzzleBoard, List<Point>
orient, int offsetR, int offsetC) {
        for (Point p : orient) {
            int r = offsetR + p.r;
            int c = offsetC + p.c;
            if (r < 0 || r >= puzzleBoard.getRows() || c < 0 || c >=
puzzleBoard.getCols())
                return false;
            char cell = puzzleBoard.getCell(r, c);
            if (cell == '#') {
                // Skip sel yang sudah terisi
                continue;
            }
            if (cell != '.')
                return false;
        }
        return true;
    }

    // Letakkan blok puzzle pada papan pada posisi tertentu
    // Karakter '#' diabaikan, karakter '.' diisi dengan karakter id
    public static void placePiece(PuzzleBoard puzzleBoard, char id,
List<Point> orient, int offsetR, int offsetC) {
        for (Point p : orient) {
            int r = offsetR + p.r, c = offsetC + p.c;
            if (puzzleBoard.getCell(r, c) == '.')
                puzzleBoard.setCell(r, c, id);
        }
    }

    // Menghapus blok puzzle dari papan pada posisi tertentu
    // Karakter '#' diabaikan, karakter '.' diisi kembali dengan karakter '.'
    public static void removePiece(PuzzleBoard puzzleBoard, List<Point>
orient, int offsetR, int offsetC) {
        for (Point p : orient) {

```

```

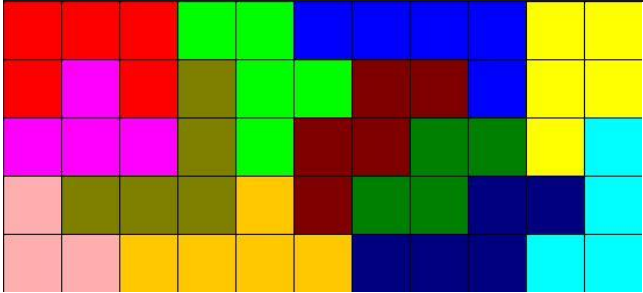
        int r = offsetR + p.r, c = offsetC + p.c;
        if (puzzleBoard.getCell(r, c) != '#')
            puzzleBoard.setCell(r, c, '.');
    }
}

// Simpan solusi ke file output
public static void saveSolution(String fileName, PuzzleBoard puzzleBoard,
long searchTimeMs, long casesTried) {
    try (PrintWriter writer = new PrintWriter(new File(fileName))) {
        for (int i = 0; i < puzzleBoard.getRows(); i++) {
            for (int j = 0; j < puzzleBoard.getCols(); j++) {
                writer.print(puzzleBoard.getCell(i,j));
            }
            writer.println();
        }
        writer.println();
        writer.println("Waktu pencarian: " + searchTimeMs + " ms");
        writer.println("Banyak kasus yang ditinjau: " + casesTried);
    } catch (IOException e) {
        System.out.println("Gagal menyimpan file: " + fileName);
    }
}
}

```

BAB VI: PENGUJIAN ALGORITMA

Test Case (Input)	Hasil (Output)
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<p>Solusi ditemukan:</p> <p> ABBCC AABCD EEEDD EFFFF GGGFF </p> <p>Waktu pencarian: 3 ms Banyak kasus yang ditinjau: 494</p> 
6 6 10 DEFAULT A AA A BBB B CC C D DD E EE E FFF GG G G HH III II JJ JJ	<p>Solusi ditemukan:</p> <p> ABBCCD AABCD AEBFFF GEEHHI GEJJII GGJJII </p> <p>Waktu pencarian: 1 ms Banyak kasus yang ditinjau: 82</p> 

5 11 12 DEFAULT AAA AA B BB BB C CCCC DDD DD E EEEE FFF F GG G G H HH II II I J JJ J K KK K K L L LLL	Solusi ditemukan: AAABBBCCCDD AFALBBIIICDD FFFLBIIJJDDG HLLLIIJJKKG HHEEEKKKGG Waktu pencarian: 108 ms Banyak kasus yang ditinjau: 1261460 
2 2 1 DEFAULT A	Waktu pencarian: 1 ms Banyak kasus yang ditinjau: 1 Tidak ada solusi.
5 11 12 DEFAULT AAA AA BBB BB CC C C DD DD D E	Solusi ditemukan: AAABBBCCDDE AFAGBBCCDDE FFGGGGCDKEL HFFIIJJKKEL HHHHIIJJLLLL Waktu pencarian: 5 ms Banyak kasus yang ditinjau: 652

EE E E F FF FF G GGGG H HHHH III I J JJ K KK K L L LLL	
2 2 3 DEFAULT AAA BBB CCC	<p>Waktu pencarian: 1 ms Banyak kasus yang ditinjau: 18 Tidak ada solusi.</p>
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	<p>Solusi ditemukan:</p> <pre> ###A### #BBAAA# BBBCCCC #EEECD# ###E### </pre> <p>Waktu pencarian: 9 ms Banyak kasus yang ditinjau: 14326</p> 

BAB VII: KESIMPULAN DAN SARAN

Program IQPuzzleSolver berhasil menyelesaikan berbagai kombinasi puzzle dalam 1 papan. Dalam kasus ini, permainan mempunyai solusi jika dan hanya jika seluruh potongan mampu mengisi semua sel (bagian) pada papan. Algoritma brute-force memecahkan teka-teki puzzle secara intuitif dan backtracking membantu algoritma dalam menghindari jalan buntu. Namun, algoritma ini kurang efisien untuk teka-teki dengan skala besar, seperti papan 100 x 100. Waktu komputasi semakin lama seiring meluasnya ukuran papan dan banyaknya jumlah potongan.

Perancangan aplikasi IQPuzzleSolver dapat dijadikan acuan untuk memecahkan masalah yang serupa. Namun, algoritma aplikasi ini dapat ditingkatkan lagi dengan menggunakan metode heuristik supaya jumlah rekursi dapat diminimalkan.

Referensi

- [1] R. Munir, "Algoritma Brute Force (Bagian 1)," 2025. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf). [Diakses 24 Februari 2025].
- [2] B. V. Indriyono, R. Wardani dan T. Susanti, "Analysis of the Concept of Solving the Sudoku Game Using Backtracking and Brute Force Algorithms," *International Journal of Artificial Intelligence & Robotics (IJAIR)*, vol. 6, no. 1, pp. 40-47, 2024.
- [3] L. Cloudeka, "Bagaimana Cara Kerja Algoritma Brute Force?," Cloudeka, 29 November 2023. [Online]. Available: <https://www.cloudeka.id/id/berita/web-sec/cara-kerja-algoritma-brute-force/>. [Diakses 24 Februari 2025].

Lampiran

Link repository GitHub: https://github.com/jovan196/Tucil1_13523141

Tabel 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 Graphical User Interface (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	✓	