

# **TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA**

Penyelesaian **IQ Puzzler Pro** dengan Algoritma Brute Force



Disusun oleh:

Nicholas Andhika Lucas    13523014

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA - KOMPUTASI**

**INSTITUT TEKNOLOGI BANDUNG**

**2025**

## DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
PERMASALAHAN DAN ALGORITMA PENYELESAIAN.....	2
1. Deskripsi Permasalahan.....	2
2. Algoritma Penyelesaian.....	3
3. Pseudocode Algoritma Solver.....	4
BAB III.....	6
SOURCE CODE PROGRAM.....	6
1. Main.java.....	6
2. Solver.java.....	7
3. Input.java.....	8
4. Board.java.....	10
5. Block.java.....	12
BAB IV.....	15
TEST CASE.....	15
1. Test Case 1.....	15
2. Test Case 2.....	15
3. Test Case 3.....	16
4. Test Case 4.....	17
5. Test Case 5.....	17
6. Test Case 6.....	18
7. Test Case 7.....	19
BAB V.....	20
KESIMPULAN.....	20

# BAB I

## PERMASALAHAN DAN ALGORITMA PENYELESAIAN

### 1. Deskripsi Permasalahan



Gambar 1. Permainan IQ Puzzler Pro. Sumber: smartgamesusa.com.

IQ Puzzler Pro adalah suatu 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:

#### a. Board (Papan)

Komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.

#### b. 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. Dalam penyusunan solusi dari permainan ini, blok/*piece* dapat dirotasi atau dicerminkan untuk ditempatkan pada permainan.

Permasalahan yang ingin diselesaikan adalah mencari hanya satu solusi dari permainan tersebut – ketika seluruh papan terisi dengan blok tanpa ada bagian yang kosong –, menggunakan algoritma *brute-force* yang diajarkan pada kelas IF2211 Strategi Algoritma, atau menyatakan tidak ada solusi jika memang tidak ditemukan. Dalam tugas ini, spesifikasi yang diminta adalah permainan yang diawali dengan papan kosong.

## 2. Algoritma Penyelesaian

Penyelesaian permasalahan dalam permainan IQ Puzzler Pro ini menggunakan algoritma *brute force* yang diimplementasikan dalam bahasa Java. Langkah-langkah implementasi algoritmanya adalah sebagai berikut:

1. Program meminta input berupa file .txt yang berisi deskripsi mengenai:
  - a. Dimensi Papan  
Terdiri atas dua buah variabel integer **N** dan **M** yang membentuk papan berdimensi NxM
  - b. Banyak Blok Puzzle  
Dinyatakan dalam variabel integer **P**.
  - c. Jenis Kasus  
Dinyatakan dalam sebuah variabel string **S** yang digunakan untuk mengidentifikasi kasus konfigurasi, hanya mungkin bernilai salah satu diantara DEFAULT/CUSTOM/PYRAMID (namun dalam implementasi tugas ini hanya menggunakan DEFAULT)
  - d. Bentuk Blok Puzzle  
Dilambangkan dengan konfigurasi karakter berhuruf A-Z kapital yang membentuk sebanyak **P** jenis blok puzzle yang berbeda menggunakan **P** huruf berbeda.
2. Program memproses input file yang diberikan dan menyimpan semua nilai pada input sebagai variabel objek.
3. Program memulai pencarian solusi secara rekursif, dengan basis semua blok puzzle telah digunakan dan seluruh kotak pada *board* permainan sudah terisi – dalam artian lain ditemukan solusi pada permainan –. Selain itu, rekursi akan dilakukan terus-menerus untuk mencari kombinasi penyusunan blok puzzle yang dapat memenuhi persyaratan

yang diminta, atau hingga seluruh kemungkinan telah dicari dan tidak menghasilkan solusi.

4. Sebuah blok yang diambil dari input file memiliki banyak kemungkinan posisi karena dapat dirotasi dan/atau dicerminkan. Program akan membuat daftar seluruh kemungkinan orientasi dari blok tersebut yang disimpan dalam sebuah *list*.
5. Program mulai melakukan pencarian posisi yang valid untuk posisi blok pertama pada titik awal (0,0) di papan. Program berusaha mencari variasi orientasi blok yang dapat mengisi papan, dimulai dari orientasi pertama yang terdaftar. Blok tidak dapat ditempatkan apabila terletak di luar batasan indeks papan atau terjadi *overlap* dengan blok lain. Ketika suatu blok tidak berhasil ditempatkan, maka program akan mencari variasi orientasi lain dari blok tersebut yang dapat ditempatkan. Jika masih tidak dapat ditemukan, maka program akan mencoba menempatkan blok tersebut di posisi selanjutnya, dimulai dari *traversal* posisi kolom, kemudian *traversal* posisi baris.
6. Setelah blok pertama ditempatkan, maka akan dilakukan pencarian posisi yang valid untuk penempatan blok selanjutnya, mengikuti langkah-langkah rekursi yang disebutkan pada langkah 4-5. Jika blok selanjutnya berhasil ditempatkan, langkah ini akan diulangi.
7. Apabila suatu blok sama sekali tidak mungkin untuk ditempatkan pada orientasi atau posisi manapun, program melakukan *backtracking* untuk kembali ke blok sebelumnya untuk mencoba kombinasi yang lain.
8. Program berakhir jika ditemukan satu solusi dari permainan atau tidak ditemukannya solusi. Dicantumkan juga waktu pencarian oleh algoritma, banyak langkah yang ditelusuri, serta opsi untuk menyimpan hasil solusi ke dalam suatu file .txt apabila terdapat solusi dari permasalahan yang diberikan.

### 3. Pseudocode Algoritma Solver

```
function solvePuzzle(board:Matrix of character, blocks: List of Block,  
blockIndex:int, stepsTaken:List of Integer)-> boolean  
    { Basis }  
    if (blockIndex = size(blocks) then  
        -> isBoardFull(board)
```

```

{ Rekursi }
currentBlock <- blocks[blockIndex]
orientations <- GenerateAllOrientations(currentBlock)

i traversal [0..board.length - 1]
  j traversal [0..board[0].length - 1]
    for each orientation in orientations:
      stepsTaken <- stepsTaken[0] + 1
      if canPlaceBlock(board, getShape(orientation), i, j)
      then
        placePiece(board, getShape(orientation), i, j,
          getID(orientation))
        idxPiece <- idxPiece + 1

        if solvePuzzle(board, blocks, blockIndex + 1,
          stepsTaken) then
          -> true
        else
          removeBlock(board, getShape(orientation),
            i, j)

-> false

```

## BAB III

### *SOURCE CODE PROGRAM*

Berikut terlampir implementasi algoritma yang disebutkan pada bab pertama, menggunakan bahasa pemrograman Java. Pada implementasi tugas ini, dimanfaatkan 5 kelas berupa Main, Solver, Input, Board, Block, dan ParsedData.

#### 1. Main.java

```
import java.io.IOException;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        Input input = new Input();
        Input.ParsedData parsedData;

        try {
            parsedData = input.readFile();
            if (parsedData == null) {
                System.out.println("Failed to read input data.");
                return;
            }
        } catch (IOException e) {
            System.out.println("Error reading input file: " + e.getMessage());
            return;
        }

        // Initialize data
        int rows = parsedData.rows;
        int cols = parsedData.cols;
        List<Block> blocks = parsedData.blocks;
        Board board = new Board(rows, cols);

        // Track time and steps taken
        long startTime = System.nanoTime();
        int[] stepsTaken = {0};

        Solve solver = new Solve();
        boolean solved = solver.solvePuzzle(board.getBoard(), blocks, 0, stepsTaken);
        long endTime = System.nanoTime();

        // Print results
        if (solved) {
            System.out.println("Solusi:");
            board.printBoard();
            System.out.println();
            System.out.println("Waktu pencarian: " + (endTime - startTime) / 1_000_000.0 + " (ms)");
            System.out.println();
        }
    }
}
```

```

        System.out.println("Banyak kasus yang ditinjau: " + stepsTaken[0]);
        System.out.println();
        Solve.saveSolution(board.getBoard());
    } else {
        System.out.println("Tidak ada solusi.");
        System.out.println();
        System.out.println("Waktu pencarian: " + (endTime - startTime) / 1_000_000.0 + " (ms)");
        System.out.println();
        System.out.println("Banyak kasus yang ditinjau: " + stepsTaken[0]);
    }
}
}

```

## 2. Solver.java

```

import java.io.FileWriter;
import java.io.IOException;

import java.time.LocalDateTime;
import java.time.Format.DateTimeFormatter;
import java.util.List;
import java.util.Scanner;

public class Solve {
    public boolean solvePuzzle(char[][] board, List<Block> blocks, int blockIndex, int[] stepsTaken)
    {
        // Base case: All blocks placed, return true
        if (blockIndex == blocks.size()) {
            return Board.isBoardFull(board);
        }

        Block currentBlock = blocks.get(blockIndex);
        List<Block> orientations = Block.generateAllOrientation(currentBlock);

        // Try placing all block positions on the board
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                for (Block orientation : orientations) {

                    stepsTaken[0]++;

                    if (Board.canPlaceBlock(board, orientation.getShape(), i, j)) {
                        Board.placeBlock(board, orientation.getShape(), i, j, orientation.getID());
                        // place the block with the ID of char A-Z

                        // Recursion call to place next block
                        if (solvePuzzle(board, blocks, blockIndex + 1, stepsTaken)) {
                            return true;
                        } else {

                            // Backtrack if placing next block fail

```



```

        Board.removeBlock(board, orientation.getShape(), i, j);
    }
}

}

}

// No valid placement found for current block
return false;
}

public static void saveSolution(char[][] board) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Apakah anda ingin menyimpan solusi? (Y/N): ");
    String choice = scanner.nextLine().trim().toLowerCase();

    if (!choice.equals("y")) {
        return;
    }

    // Format output file name
    LocalDateTime time = LocalDateTime.now();
    DateTimeFormatter timeFormat = DateTimeFormatter.ofPattern("HHmmss_ddMMyyyy");
    String timestamp = time.format(timeFormat);
    String filePath = "../test/output/solusi_" + timestamp + ".txt";

    try (FileWriter writer = new FileWriter(filePath)) {
        for (char[] row : board) {
            for (char cell : row) {
                writer.write(cell);
            }
            writer.write("\n");
        }
        System.out.println("Solusi tersimpan dengan nama: solusi_" + timestamp + ".txt");
    } catch (IOException e) {
        System.out.println("Gagal menyimpan solusi.");
    }
}
}

```

### 3. Input.java

```

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

public class Input {
    private static Scanner in = new Scanner(System.in);

    public static class ParsedData {
        public int rows, cols;
    }
}

```

```

public List<Block> blocks;
public String config;

public ParsedData(int rows, int cols, List<Block> blocks, String config) {
    this.rows = rows;
    this.cols = cols;
    this.blocks = blocks;
    this.config = config;
}
}

public ParsedData readFile() throws IOException {
    System.out.print("Masukkan nama file: ");
    String fileName = in.next();

    try (Scanner fileScanner = new Scanner(new File("../test/input/" + fileName))) {

        // Read first line N M P
        String[] firstLine = fileScanner.nextLine().split(" ");
        int rows = Integer.parseInt(firstLine[0]);
        int cols = Integer.parseInt(firstLine[1]);
        int blockCount = Integer.parseInt(firstLine[2]);

        // Read second line "DEFAULT/CUSTOM/PYRAMID"
        String config = fileScanner.nextLine();

        // Set map (A-Z) to read blocks
        Map<Character, List<int[]>> blockMap = new HashMap<>();
        int rowIdx = 0;
        Character currentBlockId = null;

        // Read blocks
        while (fileScanner.hasNextLine()) {
            String line = fileScanner.nextLine();
            char blockId = line.charAt(0);

            // Reset row count on new block
            if (currentBlockId == null || currentBlockId != blockId) {
                currentBlockId = blockId;
                rowIdx = 0;
            }

            // Add new coordinates to the block
            for (int colIdx = 0; colIdx < line.length(); colIdx++) {
                if (line.charAt(colIdx) == blockId) {
                    blockMap.computeIfAbsent(blockId, k -> new ArrayList<>()).add(new
int[] {rowIdx, colIdx});
                }
            }
            rowIdx++;
        }

        // Error if initialized block count is not the same as unique number of blocks
        if (blockMap.size() != blockCount) {
            System.out.println("Error: Expected " + blockCount + " blocks, but found " +

```

```

blockMap.size());
        return null;
    }

    // Convert map to blocks
    List<Block> blocks = new ArrayList<>();
    for (Map.Entry<Character, List<int[]>> entry : blockMap.entrySet()) {
        blocks.add(new Block(entry.getKey(), entry.getValue()));
    }

    return new ParsedData(rows, cols, blocks, config);

} catch (FileNotFoundException e) {
    System.out.println("File tidak ditemukan: " + e.getMessage());
    return null;
}
}
}

```

#### 4. Board.java

```

import java.util.Arrays;
import java.util.List;

public class Board {
    private int rows, cols;
    private char[][] board;

    public int getRowLength() {
        return this.rows;
    }

    public int getCollLength() {
        return this.cols;
    }

    public char[][] getBoard() {
        return this.board;
    }

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.board = new char[rows][cols];

        // Set empty space on board with '.'
        for (int i = 0; i < this.rows; i++) {
            Arrays.fill(this.board[i], '.');
        }
    }

    public void printBoard() {
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++) {

```

```

        System.out.print(getColor(board[i][j]) + board[i][j] + "\u001B[0m ");
    }
    System.out.println();
}

/** Helper function for brute force algorithm */
public static boolean canPlaceBlock(char[][] board, List<int[]> orientation, int row, int col) {
    for (int[] coordinate : orientation) {
        int x = coordinate[0] + col;
        int y = coordinate[1] + row;

        // Check if out of bounds or overlap with existing blocks
        if (x < 0 || y < 0 || x >= board[0].length || y >= board.length || board[y][x] != '.') {
            return false;
        }
    }
    return true;
}

public static void placeBlock(char[][] board, List<int[]> orientation, int row, int col, char
blockID) {
    for (int[] coordinate : orientation) {
        int x = coordinate[0] + col;
        int y = coordinate[1] + row;
        board[y][x] = blockID; // Place the block on the board
    }
}

public static void removeBlock(char[][] board, List<int[]> orientation, int row, int col) {
    for (int[] coordinate : orientation) {
        int x = coordinate[0] + col;
        int y = coordinate[1] + row;
        board[y][x] = '.'; // Reset the cell to . (empty)
    }
}

/** OTHER UTILS */
public static String getColor(char blockID) {
    int ascii = blockID; // Convert A-Z to ASCII (65-90)
    int r = (ascii * 47) % 256; // Prime numbers help spread colors
    int g = (ascii * 83) % 256;
    int b = (ascii * 191) % 256;

    return String.format("\u001B[38;2;%d;%d;%dm", r, g, b); // Return ANSI RGB code
}

public static boolean isBoardFull(char[][] board) {
    for (char[] row : board) {
        for (char cell : row) {
            if (cell == '.') {
                return false;
            }
        }
    }
}

```

```

        return true;
    }
}

```

## 5. Block.java

```

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Block {
    private char id; // Nama blok puzzle (A-Z)
    private List<int[]> shape; // Array (row, col), offset dari origin 0,0

    public char getID() {
        return id;
    }

    public List<int[]> getShape() {
        return shape;
    }

    public Block(char id, List<int[]> shape) {
        this.id = id;
        this.shape = new ArrayList<>(shape);
    }

    /** Transformative Functions */
    // Rotate 90 degrees clockwise
    public Block rotate() {
        List<int[]> rotateShape = new ArrayList<>();
        // Formula: (x,c)→(c,-x)
        for (int[] pos : shape) {
            int newRow = pos[1];
            int newCol = -pos[0];
            rotateShape.add(new int[]{newRow, newCol});
        }
        return new Block(id, rotateShape);
    }

    // Mirror block horizontally
    public Block mirrorHorizontal() {
        List<int[]> mirrorShape = new ArrayList<>();
        // Formula: (x,c)→(x,-c)
        for (int[] pos : shape) {
            int newRow = pos[0];
            int newCol = -pos[1];
            mirrorShape.add(new int[]{newRow, newCol});
        }
        return new Block(id, mirrorShape);
    }
}

```

```

}

// Mirror block vertically
public Block mirrorVertical() {
    List<int[]> mirrorShape = new ArrayList<>();
    // Formula: (x,c)-(-x,c)
    for (int[] pos : shape) {
        int newRow = -pos[0];
        int newCol = pos[1];
        mirrorShape.add(new int[]{newRow, newCol});
    }
    return new Block(id, mirrorShape);
}

// Normalize coordinates to (0,0)
public static List<int[]> normalizeShape(List<int[]> shape) {
    int minRow = Integer.MAX_VALUE;
    int minCol = Integer.MAX_VALUE;

    // Find min row/col
    for (int[] pos : shape) {
        minRow = Math.min(minRow, pos[0]);
        minCol = Math.min(minCol, pos[1]);
    }

    // Shift coordinate
    List<int[]> normalizedShape = new ArrayList<>();
    for (int[] pos : shape) {
        normalizedShape.add(new int[]{pos[0] - minRow, pos[1] - minCol});
    }

    return normalizedShape;
}

// Find if shapes are equal
private static boolean isShapeEqual(List<int[]> shape1, List<int[]> shape2) {
    if (shape1.size() != shape2.size()) return false;

    Set<String> set1 = new HashSet<>();
    Set<String> set2 = new HashSet<>();

    for (int[] pos : shape1) {
        set1.add(pos[0] + "," + pos[1]); // Store coordinates in set as "row,col"
    }
    for (int[] pos : shape2) {
        set2.add(pos[0] + "," + pos[1]);
    }

    return set1.equals(set2); // Fast O(1) comparison
}

// Find if a shape is already in a block to avoid duplicates
private static boolean isShapeInBlock(List<Block> source, Block target) {
    for (Block b : source) {
        if (isShapeEqual(b.getShape(), target.getShape())) {

```

```

        return true;
    }
}
return false;
}

// Generate all possible orientation of blocks
public static List<Block> generateAllOrientation(Block block) {
    List<Block> uniqueOrientation = new ArrayList<>();
    Block currentShape = block;

    // Generate All Rotation
    for (int i = 0; i < 4; i++) {
        if (!isShapeInBlock(uniqueOrientation, currentShape)){
            uniqueOrientation.add(currentShape);
        }
        currentShape = currentShape.rotate();
        currentShape = new Block(block.id, normalizeShape(currentShape.getShape()));
    }

    // Generate Horizontal Mirror
    currentShape = block.mirrorHorizontal();
    currentShape = new Block(block.id, normalizeShape(currentShape.getShape()));
    for (int i = 0; i < 4; i++) {
        if (!isShapeInBlock(uniqueOrientation, currentShape)){
            uniqueOrientation.add(currentShape);
        }
        currentShape = currentShape.rotate();
        currentShape = new Block(block.id, normalizeShape(currentShape.getShape()));
    }

    // Generate Vertical Mirror
    currentShape = block.mirrorVertical();
    currentShape = new Block(block.id, normalizeShape(currentShape.getShape()));
    for (int i = 0; i < 4; i++) {
        if (!isShapeInBlock(uniqueOrientation, currentShape)){
            uniqueOrientation.add(currentShape);
        }
        currentShape = currentShape.rotate();
        currentShape = new Block(block.id, normalizeShape(currentShape.getShape()));
    }

    return uniqueOrientation;
}

// Print block
public void printShape() {
    System.out.println("Block " + id + ":");
    for (int[] pos : shape) {
        System.out.println("(" + pos[0] + ", " + pos[1] + ")");
    }
}
}
}

```

## BAB IV

### *TEST CASE*

#### 1. Test Case 1

Input (.txt)	Output (CLI)
<pre> test &gt; input &gt; ≡ tc1.txt 1    5 5 7 2    DEFAULT 3    A 4    AA 5    B 6    BB 7    C 8    CC 9    D 10   DD 11   EE 12   EE 13   E 14   FF 15   FF 16   F 17   GGG </pre>	<pre> Masukkan nama file: tc1.txt Solusi: A A B B G C A D B G C C D D G E E E F F E E F F F  Waktu pencarian: 28.5669 (ms)  Banyak kasus yang ditinjau: 783  Apakah anda ingin menyimpan solusi? (Y/N): y Solusi tersimpan dengan nama: solusi_112650_24022025.txt </pre>

#### 2. Test Case 2

Input (.txt)	Output (CLI)
--------------	--------------



<pre>test &gt; input &gt; tc2.txt 1 6 6 11 2 DEFAULT 3 A 4 AA 5 B 6 BB 7 C 8 CC 9 D 10 DD 11 E 12 EE 13 F 14 FF 15 G 16 GG 17 HHHHHH 18 I 19 II 20 JJJJ 21 KK</pre>	<pre>Masukkan nama file: tc2.txt Solusi: A A B B C C D A E B F C D D E E F F G G K K I I G J J J J I H H H H H H  Waktu pencarian: 24.4374 (ms)  Banyak kasus yang ditinjau: 2179  Apakah anda ingin menyimpan solusi? (Y/N): y Solusi tersimpan dengan nama: solusi_112654_24022025.txt</pre>
---	--

### 3. Test Case 3

Input (.txt)	Output (CLI)
--------------	--------------

<pre>test &gt; input &gt; ≡ tc3.txt 1    2 4 3 2    DEFAULT 3    A 4    B 5    BB 6    CC 7    CC</pre>	<pre>Masukkan nama file: tc3.txt Solusi: A B C C B B C C  Waktu pencarian: 10.9916 (ms)  Banyak kasus yang ditinjau: 8  Apakah anda ingin menyimpan solusi? (Y/N): y Solusi tersimpan dengan nama: solusi_112659_24022025.txt</pre>
---	---

#### 4. Test Case 4

Input (.txt)	Output (CLI)
<pre>test &gt; input &gt; ≡ tc4.txt 1    4 6 6 2    DEFAULT 3    AAAA 4    BB 5    B 6    C 7    DDD 8    D 9    EE 10   E 11   FF</pre>	<pre>Masukkan nama file: tc4.txt Tidak ada solusi.  Waktu pencarian: 10171.8242 (ms)  Banyak kasus yang ditinjau: 108254064</pre>

#### 5. Test Case 5

Input (.txt)	Output (CLI)
--------------	--------------

<pre>test &gt; input &gt; ≡ tc5.txt 1 4 4 7 2 DEFAULT 3 AA 4 BB 5 CC 6 DD 7 EE 8 FF 9 F 10 GG 11 G</pre>	<pre>Masukkan nama file: tc5.txt Solusi: A B C D A B C D E F F G E F G G  Waktu pencarian: 12.9733 (ms)  Banyak kasus yang ditinjau: 113  Apakah anda ingin menyimpan solusi? (Y/N): y Solusi tersimpan dengan nama: solusi 113158 24022025.txt</pre>
--	---

## 6. Test Case 6

Input (.txt)	Output (CLI)
<pre>test &gt; input &gt; ≡ tc6.txt 1 4 4 6 2 DEFAULT 3 AA 4 A 5 BB 6 C 7 CC 8 DD 9 D 10 EE 11 E 12 FF</pre>	<pre>Masukkan nama file: tc6.txt Solusi: A A B F A C B F D C C E D D E E  Waktu pencarian: 17.689 (ms)  Banyak kasus yang ditinjau: 561  Apakah anda ingin menyimpan solusi? (Y/N): y Solusi tersimpan dengan nama: solusi 112837 24022025.txt</pre>

## 7. Test Case 7

Input (.txt)	Output (CLI)
<pre>test &gt; input &gt; ≡ tc7.txt 1    3 3 3 2    DEFAULT 3    XX 4    X 5    Y 6    Y 7    Y 8    Z 9    ZZ</pre>	<pre>Masukkan nama file: tc7.txt Solusi: X X Y X Z Y Z Z Y  Waktu pencarian: 13.7182 (ms)  Banyak kasus yang ditinjau: 23  Apakah anda ingin menyimpan solusi? (Y/N): y</pre>

## BAB V

### KESIMPULAN

Pada tugas kecil ini, saya berusaha mengimplementasikan salah satu materi yang diajarkan pada kelas, yaitu algoritma *brute force*. Algoritma ini tidak mementingkan efisiensi atau mempertimbangkan faktor seperti waktu maupun memori untuk menjalankan program, tetapi hanya memperdulikan hasil yang didapatkan. Oleh karena itu, algoritma *brute force* memungkinkan pencarian seluruh kemungkinan sebanyak mungkin dengan cara yang lebih ‘sederhana’ dan ‘*straight-forward*’ demi menemukan solusi yang diinginkan.

Algoritma ini diimplementasikan dalam suatu *board* game sederhana IQ Puzzler Pro, suatu permainan papan yang diproduksi oleh perusahaan Smart Games. Permainan ini mampu diselesaikan apabila seluruh papan permainan dapat diisi penuh dengan blok puzzle yang tersedia tanpa menyisakan tempat kosong. Saya mengimplementasikan algoritma *brute-force* menggunakan *backtracking* dalam mencari satu solusi saja yang dapat menyelesaikan permainan ini, berdasarkan persoalan yang diberikan. Sederhananya, blok puzzle akan diusahakan untuk ditempatkan pada papan permainan, mencoba segala jenis orientasi blok maupun posisi, yang jika masih saja tidak berhasil, akan dilakukan *backtrack* untuk mencoba blok lain dan menggunakan kombinasi selanjutnya.

Berdasarkan hasil uji coba yang dilakukan pada bab sebelumnya, dapat disimpulkan bahwa algoritma *brute force* ini cocok digunakan untuk program berskala kecil, karena mampu mencari segala kemungkinan cara dalam waktu yang cepat, mengingat kecilnya memori yang dibutuhkan. Namun, ketika mencapai kasus seperti *test case* 4 yang tidak memiliki solusi, program memakan waktu lebih lama karena mengeksplorasi banyak cara, hingga mencapai 100 juta kemungkinan.

## LAMPIRAN

Pranala *repository*:

[https://github.com/andhikalucas/Tucil1\\_13523014](https://github.com/andhikalucas/Tucil1_13523014)

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	✓	