

# Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

## Laporan Tugas Kecil 1

Disusun untuk memenuhi tugas kecil mata kuliah IF2211 Strategi Algoritma



oleh

Sebastian Hung Yansen

13523070

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>BAB I DESKRIPSI MASALAH.....</b>	<b>2</b>
<b>BAB II PENYELESAIAN.....</b>	<b>3</b>
<b>BAB III IMPLEMENTASI.....</b>	<b>4</b>
3.1. Fungsi dan Prosedur.....	4
3.2. Source Code Program.....	5
<b>BAB IV PENGUJIAN.....</b>	<b>11</b>
<b>BAB V KESIMPULAN.....</b>	<b>13</b>
<b>LAMPIRAN.....</b>	<b>14</b>

## BAB I

### DESKRIPSI MASALAH

Algoritma *brute force* adalah pendekatan algoritma yang lurus atau *straightforward* untuk memecahkan suatu persoalan. Algoritma *brute force* memecahkan persoalan secara sederhana, langsung, dan cara yang jelas serta mudah dipahami. Algoritma *brute force* dapat langsung ditulis berdasarkan pernyataan dalam persoalan (*problem statement*) dan konsep yang dilibatkan. Sebagai contoh, misalkan diminta mencari elemen terbesar dalam sebuah *array*. Algoritma *brute force* nya dengan membandingkan setiap elemen *array* dari awal hingga akhir untuk mencari elemen terbesar.

Algoritma *brute force* umumnya tidak “cerdas” dan tidak sangkil. Hal tersebut dikarenakan algoritma *brute force* membutuhkan waktu komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Oleh karena itu, algoritma *brute force* lebih cocok untuk persoalan yang bersifat kecil karena sederhana dan implementasinya yang mudah. Walau begitu, kelebihan algoritma *brute force* sendiri adalah hampir semua persoalan dapat diselesaikan dengan algoritma tersebut.

IQ Puzzler Pro adalah permainan papan dimana pemain harus dapat mengisi seluruh papan dengan *piece* yang telah tersedia. IQ Puzzle Pro sendiri terdiri atas:

1. *Board* (Papan)

Papan menjadi komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan *piece-piece* yang telah disediakan.

2. *Piece*

*Piece* adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap *piece* memiliki bentuknya masing-masing dan semua *piece* harus digunakan untuk menyelesaikan *puzzle*.

Tugas yang diberikan adalah menemukan satu solusi dari permainan IQ Puzzler Pro dengan menggunakan algoritma *Brute Force*. Apabila solusi tidak ditemukan, akan ditampilkan pesan bahwa solusi tidak ditemukan.

## BAB II

### PENYELESAIAN

Berikut adalah langkah-langkah penyelesaian IQ Puzzler Pro dengan algoritma *Brute Force*:

1. Pengguna diminta memasukkan lokasi file .txt. File .txt tersebut mengandung dimensi papan, jumlah *piece* puzzle, jenis papan, dan bentuk-bentuk *piece* yang dilambangkan menggunakan huruf A-Z dalam kapital yang dibaca oleh program. Berdasarkan jenis papan, program akan menyiapkan papan dengan masking untuk menentukan tempat-tempat yang bisa ditempati *piece*.
2. Program akan membaca dan mengelompokkan *piece* berdasarkan hurufnya dan mengubahnya ke matriks. Selain itu, program memeriksa semua bentuk *piece* dengan rotasi dan pencerminan untuk melihat bentuk apa saja yang *piece* tersebut bisa bentuk.
3. Menggunakan rekursif, program kemudian akan memeriksa apakah *piece* bisa ditempatkan. Apabila bisa, maka *piece* akan ditaruh di papan. *Piece* akan ditempatkan lalu program akan memeriksa *piece* berikutnya. Apabila *piece* tidak bisa ditempatkan, program akan mengangkat *piece* sebelumnya dan menempatkannya di tempat lain (backtracking). Apabila semua tempat tetap tidak bisa ditempatkan, maka program akan mencoba *piece* dengan bentuk yang berbeda (yang sudah dirotasi atau dicerminkan).
4. Apabila semua *piece* berhasil ditempatkan, program akan menampilkan papan beserta *piece-piece* yang sudah ditaruh dengan warna-warna berbeda, berapa lama waktu pencarian, dan jumlah kasus yang ditinjau. Apabila tidak ada solusi, program menampilkan pesan bahwa tidak ada solusi.

Ditambah dengan penyelesaian tersebut, program akan bertanya apakah ingin menyimpan hasil dalam bentuk .txt dan gambar. Dengan demikian, langkah-langkah di atas membantu dalam menyelesaikan IQ Puzzler Pro dengan algoritma *Brute Force*.

## BAB III

### IMPLEMENTASI

#### 3.1. Fungsi dan Prosedur

Fungsi/Prosedur	Tujuan
<pre>public static void main(String[] args)</pre>	Titik awal eksekusi program, memanggil fungsi lain untuk membaca file, penyelesaian, dan bertanya kepada pengguna untuk penyimpanan solusi
<pre>private static boolean readInputFile(String filePath)</pre>	Membaca file test case untuk mendapat ukuran papan, jumlah <i>piece</i> , jenis papan, dan membaca bentuk puzzle
<pre>private static char[][] convertToMatrix(List&lt;String&gt; shapelines, char letter)</pre>	Mengubah <i>piece</i> menjadi matriks
<pre>private static boolean solve(int pieceIndex)</pre>	Mencoba semua kemungkinan susunan puzzle dengan rekursif dan backtracking. Memanggil fungsi lain untuk memeriksa penempatan <i>piece</i> , menaruh <i>piece</i> , dan mengangkat <i>piece</i> .
<pre>private static List&lt;char[][]&gt; generateTransformations(char[][] piece)</pre>	Menghasilkan semua kemungkinan orientasi <i>piece</i> (rotasi dan pencerminan). Hal tersebut dilakukan dengan memutar <i>piece</i> sebanyak 4 kali, mencerminkan <i>piece</i> , dan menyimpan transformasi-transformasi yang unik.
<pre>private static char[][] copyMatrix(char[][] matrix)</pre>	Meng-copy matriks
<pre>private static char[][] rotate90(char[][] piece)</pre> <pre>private static char[][] mirror(char[][] piece)</pre>	Memutar dan mencerminkan <i>piece</i> . Pencerminan dilakukan secara horizontal
<pre>private static boolean canPlace(char[][] piece, int r, int c)</pre>	Memeriksa apakah <i>piece</i> dapat ditaruh atau tidak. Diperiksa apakah <i>piece</i> tidak keluar batas papan, hanya ditempatkan di tempat yang valid, dan tidak menabrak <i>piece</i> lain.
<pre>private static void placePiece(char[][] piece, int r, int c, char symbol)</pre>	Menempatkan <i>piece</i>
<pre>private static void removePiece(char[][] piece, int r, int c)</pre>	Menghapus <i>piece</i> ketika <i>backtracking</i>
<pre>private static void printBoard()</pre>	Mencetak papan ke terminal beserta <i>piece-piece</i> yang sudah ditempatkan. Setiap <i>piece</i> memiliki warna unik.

<pre>private static void saveSolution(String outputPath)</pre>	Menyimpan solusi sebagai file .txt. File disimpan di folder 'solutions' yang ditempatkan pada tempat file test case
<pre>SaveAsImage.main(board, rows, cols, filePath, filename);</pre> <pre>public static void main(char[][] board, int rows, int cols, String testCasePath, String filename)</pre>	Menyimpan solusi sebagai file .png. Setiap <i>piece</i> memiliki warna unik pada gambar. Gambar disimpan di folder 'images' yang ditempatkan pada tempat file test case
Tambahan	
<pre>private static void printMatrix(char[][] matrix)</pre>	Untuk mencetak <i>piece</i> yang sudah diubah ke matriks

### 3.2. Source Code Program

#### 1. File IQPuzzlerSolver.java

```
src > J IQPuzzlerSolver.java > ...
1  import java.io.*;
2  import java.util.*;
3
4
5  public class IQPuzzlerSolver {
6      private static int rows, cols, pieceCount;
7      private static char[][] board;
8      private static char[][] maskBoard; // For Masking the board
9      private static List<char[][]> pieces = new ArrayList<>();
10     private static long iterations = 0;
11     // COLORS //
12     private static final String[] COLORS = {
13         "\u001B[31m", // Red
14         "\u001B[32m", // Green
15         "\u001B[33m", // Yellow
16         "\u001B[34m", // Blue
17         "\u001B[35m", // Magenta
18         "\u001B[36m", // Cyan
19         "\u001B[38;5;208m", // Orange
20         "\u001B[38;5;214m", // Light Orange
21         "\u001B[38;5;165m", // Purple
22         "\u001B[38;5;200m", // Pink
23         "\u001B[38;5;118m", // Bright Lime Green
24         "\u001B[38;5;75m", // Sky Blue
25         "\u001B[38;5;220m", // Gold
26         "\u001B[38;5;130m", // Brown
27         "\u001B[38;5;255m", // Light Gray
28         "\u001B[38;5;21m", // Deep Blue
29         "\u001B[91m", // Bright Red
30         "\u001B[92m", // Bright Green
31         "\u001B[93m", // Bright Yellow
32         "\u001B[94m", // Bright Blue
33         "\u001B[95m", // Bright Magenta
34         "\u001B[96m", // Bright Cyan
35         "\u001B[90m", // Dark Gray
36         "\u001B[97m", // White
37         "\u001B[38;5;196m", // Dark Red
```

```

38     "\u001B[38;5;40m", // Dark Green
39 );
40 private static final String RESET = "\u001B[0m";
41
42 Run | Debug
43 public static void main(String[] args) {
44     Scanner scanner = new Scanner(System.in);
45     System.out.print(s:"Masukkan path test case(.txt): ");
46     String filePath = scanner.nextLine();
47     System.out.println(x:"");
48     if (!readInputFile(filePath)) {
49         System.out.println(x:"Gagal membaca file test case atau file tidak ada.\n");
50         scanner.close();
51         return;
52     }
53     long startTime = System.currentTimeMillis();
54     boolean solved = solve(pieceIndex:0);
55     long endTime = System.currentTimeMillis();
56
57     if (solved) {
58         printBoard();
59     } else {
60         System.out.println(x:"Tidak ditemukan solusi.\n");
61     }
62
63     System.out.println("Waktu pencarian: " + (endTime - startTime) + " ms\n");
64     System.out.println("Banyak kasus yang ditinjau: " + iterations + "\n");
65
66     if (solved) {
67         System.out.print(s:"Apakah anda ingin menyimpan solusi sebagai txt? (ya/tidak): ");
68         if (scanner.nextLine().equalsIgnoreCase(anotherString:"ya")) {
69             saveSolution(filePath + "_solution.txt", filePath);
70         }
71         System.out.print(s:"Apakah anda ingin menyimpan solusi sebagai gambar? (ya/tidak): ");

```

```

72         if (scanner.nextLine().equalsIgnoreCase(anotherString:"ya")) {
73             System.out.print(s:"Masukkan nama file: ");
74             String filename = scanner.nextLine();
75             SaveAsImage.main(board, rows, cols, filePath, filename);
76         }
77     }
78     scanner.close();
79 }
80
81 private static boolean readInputFile(String filePath) {
82     try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
83         String[] dimensions = br.readLine().split(regex:" ");
84         rows = Integer.parseInt(dimensions[0]);
85         cols = Integer.parseInt(dimensions[1]);
86         pieceCount = Integer.parseInt(dimensions[2]);
87         maskBoard = new char[rows][cols];
88         board = new char[rows][cols];
89         String boardType = br.readLine(); // DEFAULT, CUSTOM
90         if (boardType.equals(anObject:"DEFAULT")) {
91             for (char[] row : maskBoard) Arrays.fill(row, val:'X');
92         } else if (boardType.equals(anObject:"CUSTOM")) {
93             for (int i = 0; i < rows; i++) {
94                 maskBoard[i] = br.readLine().toCharArray();
95             }
96         }
97         for (char[] row : board) Arrays.fill(row, val:'.');
98
99         int piecesRead = 0;
100         String currentLetter = "";
101         List<String> shapeLines = new ArrayList<>();
102         String line;
103
104         while ((line = br.readLine()) != null) {
105             line = line.stripTrailing(); // Menghapus space tambahan

```

```

106         if (line.isEmpty()) continue; // Melewati barisan kosong
107
108         char firstChar = line.trim().charAt(index:0); // Memeriksa huruf
109         if (currentLetter.isEmpty() || firstChar == currentLetter.charAt(index:0)) {
110             shapelines.add(line);
111         } else {
112             // Simpan piece sebelum, tambah piece lain
113             pieces.add(convertToMatrix(shapelines, currentLetter.charAt(index:0)));
114             piecesRead++;
115             shapelines.clear();
116             shapelines.add(line);
117         }
118         currentLetter = String.valueOf(firstChar);
119     }
120
121     // Piece terakhir
122     if (!shapelines.isEmpty()) {
123         pieces.add(convertToMatrix(shapelines, currentLetter.charAt(index:0)));
124         piecesRead++;
125     }
126
127     return piecesRead == pieceCount;
128 } catch (Exception e) {
129     return false;
130 }
131 }
132
133 private static char[][] convertToMatrix(List<String> shapelines, char letter) {
134     int h = shapelines.size();
135     int w = shapelines.stream().mapToInt(String::length).max().orElse(0);
136
137     // Membuat matrix kosong
138     char[][] shape = new char[h][w];
139     for (int i = 0; i < h; i++) {
140         Arrays.fill(shape[i], val: ' ');
141     }
142
143     String line = shapelines.get(i);
144     for (int j = 0; j < line.length(); j++) {
145         if (line.charAt(j) == letter) {
146             shape[i][j] = letter; // Matrix diisi dengan huruf
147         }
148     }
149     return shape;
150 }
151
152 private static boolean solve(int pieceIndex) {
153     if (pieceIndex >= pieces.size()) return true;
154
155     char[][] piece = pieces.get(pieceIndex);
156     List<char[][]> transformations = generateTransformations(piece);
157
158     for (char[][] transformedPiece : transformations) {
159         for (int r = 0; r <= rows - transformedPiece.length; r++) {
160             for (int c = 0; c <= cols - transformedPiece[0].length; c++) {
161                 if (canPlace(transformedPiece, r, c)) {
162                     char shapeSymbol = piece[0][0];
163                     int symbolPicker = 1;
164                     while (shapeSymbol == ' ') { // Kalau tile atas kiri kosong, cari tile lain
165                         shapeSymbol = piece[0][symbolPicker]; // e.g: .A
166                         symbolPicker++; // AA
167                     } // where dot = empty space
168                     placePiece(transformedPiece, r, c, shapeSymbol);
169                     iterations++;
170                     if (solve(pieceIndex + 1)) return true;
171                     removePiece(transformedPiece, r, c);
172                 }
173             }
174         }
175     }
176     return false;

```



```

176     }
177
178     private static List<char[][]> generateTransformations(char[][] piece) {
179         Set<String> uniqueTransformations = new HashSet<>();
180         List<char[][]> transformations = new ArrayList<>();
181         for (int i = 0; i < 4; i++) {
182             piece = rotate90(piece);
183             if (uniqueTransformations.add(Arrays.deepToString(piece))) {
184                 transformations.add(copyMatrix(piece));
185             }
186             char[][] mirrored = mirror(piece);
187             if (uniqueTransformations.add(Arrays.deepToString(mirrored))) {
188                 transformations.add(copyMatrix(mirrored));
189             }
190         }
191         return transformations;
192     }
193
194     private static char[][] copyMatrix(char[][] matrix) {
195         char[][] copy = new char[matrix.length][matrix[0].length];
196         for (int i = 0; i < matrix.length; i++) {
197             System.arraycopy(matrix[i], 0, copy[i], 0, matrix[i].length);
198         }
199         return copy;
200     }
201     // Modify pieces
202     private static char[][] rotate90(char[][] piece) {
203         int h = piece.length, w = piece[0].length;
204         char[][] rotated = new char[w][h];
205         for (int i = 0; i < h; i++) {
206             for (int j = 0; j < w; j++) {
207                 rotated[j][h - 1 - i] = piece[i][j];
208             }
209         }
210         return rotated;

```

```

211     }
212
213     private static char[][] mirror(char[][] piece) {
214         int h = piece.length, w = piece[0].length;
215         char[][] mirrored = new char[h][w];
216         for (int i = 0; i < h; i++) {
217             for (int j = 0; j < w; j++) {
218                 mirrored[i][w - 1 - j] = piece[i][j];
219             }
220         }
221         return mirrored;
222     }
223
224     private static boolean canPlace(char[][] piece, int r, int c) {
225         for (int i = 0; i < piece.length; i++) {
226             for (int j = 0; j < piece[i].length; j++) {
227                 if (piece[i][j] != '.' ) {
228                     if (r + i >= rows || c + j >= cols || board[r + i][c + j] != '.' || maskBoard[r + i][c + j] != 'X') {
229                         return false;
230                     }
231                 }
232             }
233         }
234         return true;
235     }
236
237     private static void placePiece(char[][] piece, int r, int c, char symbol) {
238         for (int i = 0; i < piece.length; i++) {
239             for (int j = 0; j < piece[i].length; j++) {
240                 if (piece[i][j] != '.' ) {
241                     board[r + i][c + j] = symbol;
242                 }
243             }
244         }
245     }

```

```

246     private static void removePiece(char[][] piece, int r, int c) {
247         for (int i = 0; i < piece.length; i++) {
248             for (int j = 0; j < piece[i].length; j++) {
249                 if (piece[i][j] != '.' ) {
250                     board[r + i][c + j] = '.';
251                 }
252             }
253         }
254     }
255
256     private static void printBoard() { // Print board
257         Map<Character, String> colorMap = new HashMap<>();
258         for (char c = 'A'; c <= 'Z'; c++) {
259             colorMap.put(c, COLORS[(c - 'A') % COLORS.length]);
260         }
261         for (int i = 0; i < rows; i++) {
262             for (int j = 0; j < cols; j++) {
263                 char cell = board[i][j];
264                 if (cell == '.') {
265                     System.out.print((maskBoard[i][j] == 'X' ? "." : " ") + " "); // Menyembunyikan . dan/atau X pada board
266                 } else {
267                     System.out.print(colorMap.get(cell) + cell + RESET + " ");
268                 }
269             }
270             System.out.println();
271         }
272     }
273
274     private static void saveSolution(String outputPath, String testCasePath) { // File .txt
275         File testCaseDir = new File(testCasePath).getParentFile();
276         File solutionsDir = new File(testCaseDir, child:"solutions");

```

```

283     if (!solutionsDir.exists()) {
284         solutionsDir.mkdir(); // Create folder if it doesn't exist
285     }
286
287     File outputFile = new File(solutionsDir, new File(outputPath).getName());
288
289     try (PrintWriter writer = new PrintWriter(outputFile)) {
290         for (int i = 0; i < rows; i++) {
291             for (int j = 0; j < cols; j++) {
292                 char cell = board[i][j];
293                 writer.print(cell == '.' ? " " : cell);
294                 writer.print(s" ");
295             }
296             writer.println();
297         }
298         System.out.println("File berhasil disimpan di " + outputFile.getAbsolutePath());
299     } catch (IOException e) {
300         System.out.println(x:"Gagal menyimpan solusi.");
301     }
302 }
303

```

## 2. File SaveAsImage.java

```

src > J SaveAsImage.java > ...
1  import java.awt.*;
2  import java.awt.image.BufferedImage;
3  import java.io.File;
4  import java.io.IOException;
5  import java.util.HashMap;
6  import java.util.Map;
7  import javax.imageio.ImageIO;
8
9  public class SaveAsImage {
10     private static final Color[] IMAGE_COLORS = {
11         Color.RED,
12         Color.GREEN,
13         Color.YELLOW,
14         Color.BLUE,
15         Color.MAGENTA,
16         Color.CYAN,
17         Color.ORANGE,
18         new Color(r:255,g:178,b:102), // Color.LIGHT_ORANGE,
19         new Color(r:76,g:0,b:153), // Color.PURPLE,
20         Color.PINK,
21         new Color(r:102,g:255,b:102), // Color.BRIGHT_LIME_GREEN,
22         new Color(r:153,g:204,b:255), // Color.SKY_BLUE,
23         new Color(r:255, g:215, b:0), //Color.GOLD,
24         new Color(r:102,g:51,b:0), // Color.BROWN,
25         Color.LIGHT_GRAY,
26         new Color(r:0,g:0,b:153), // Color.DEEP_BLUE,
27         new Color(r:255,g:102,b:102), // Color.BRIGHT_RED,
28         new Color(r:153,g:255,b:153), // Color.BRIGHT_GREEN,
29         new Color(r:255,g:255,b:153), // Color.BRIGHT_YELLOW,
30         new Color(r:204,g:229,b:255), // Color.BRIGHT_BLUE,
31         new Color(r:229,g:204,b:255), // Color.BRIGHT_MAGENTA,
32         new Color(r:204,g:255,b:255), // Color.BRIGHT_CYAN,
33         Color.DARK_GRAY,
34         Color.WHITE,
35         new Color(r:153,g:0,b:0), // Color.DARK_RED,
36         new Color(r:0,g:102,b:0)// Color.DARK_GREEN,
37     };

```

```

39 public static void main(char[][] board, int rows, int cols, String testCasePath, String filename) {
40     int tileSize = 50;
41     int width = cols * tileSize;
42     int height = rows * tileSize;
43
44     BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
45     Graphics2D graphics = image.createGraphics();
46     graphics.setColor(Color.WHITE);
47     graphics.fillRect(x:0, y:0, width, height);
48
49     Map<Character, Color> colorMap = new HashMap<>();
50     int colorIndex = 0;
51
52     for (char[] row : board) {
53         for (char tile : row) {
54             if (tile != '.' && !colorMap.containsKey(tile)) {
55                 colorMap.put(tile, IMAGE_COLORS[colorIndex % IMAGE_COLORS.length]);
56                 colorIndex++;
57             }
58         }
59     }
60
61     for (int r = 0; r < rows; r++) {
62         for (int c = 0; c < cols; c++) {
63             char tile = board[r][c];
64             if (tile != '.') {
65                 graphics.setColor(colorMap.get(tile));
66                 graphics.fillRect(c * tileSize, r * tileSize, tileSize, tileSize);
67
68                 graphics.setColor(Color.BLACK);
69                 graphics.drawRect(c * tileSize, r * tileSize, tileSize, tileSize);
70                 graphics.setColor(Color.WHITE);
71                 graphics.drawString(String.valueOf(tile), c * tileSize + 20, r * tileSize + 30);
72             }
73         }
74     }

```

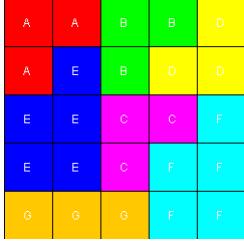
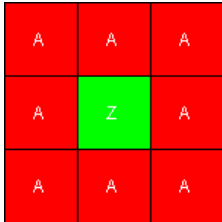
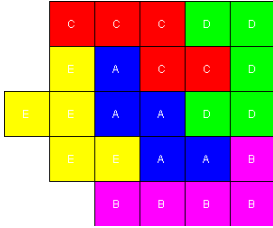
```

74     }
75
76     graphics.dispose();
77
78     try {
79         File testCaseDir = new File(testCasePath).getParentFile();
80         File imagesDir = new File(testCaseDir, child:"images");
81         if (!imagesDir.exists()) {
82             imagesDir.mkdir();
83         }
84
85         File outputFile = new File(imagesDir, filename + ".png");
86         ImageIO.write(image, formatName:"png", outputFile);
87         System.out.println("Gambar berhasil disimpan di " + outputFile.getAbsolutePath());
88     } catch (IOException e) {
89         System.out.println(x:"Gagal menyimpan gambar");
90     }
91 }
92
93 }
94

```

## BAB IV

### PENGUJIAN

Test Case	Solusi	Gambar	Keterangan
<pre>test &gt; testcase.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>Menemukan path test case(,txt): C:\Mahasiswa\Semester 4\STIMA\Tut11\10220 ... waktu pencarian: 18 ms banyak kasus yang ditinjau: 102</pre>		Test case berdasarkan contoh pada spesifikasi
<pre>test &gt; testcase2.txt 1 2 2 1 2 DEFAULT 3 A 4 AA 5 AA</pre>	<pre>Menemukan path test case(,txt): C:\Mahasiswa\Semester 4\STIMA\Tut11\10220 Tidak ditemukan solusi. waktu pencarian: 0 ms banyak kasus yang ditinjau: 0</pre>	-	Test case <i>piece</i> melebihi petak
<pre>test &gt; testcase3.txt 1 3 3 2 2 DEFAULT 3 AAA 4 A A 5 AAA 6 Z</pre>	<pre>Menemukan path test case(,txt): C:\Mahasiswa\Semester 4\STIMA\Tut11\10220 ... waktu pencarian: 1 ms banyak kasus yang ditinjau: 2</pre>		Test case <i>piece</i> berbentuk persegi kosong
<pre>test &gt; testcase4.txt 1 5 6 5 2 CUSTOM 3 .XXXXX 4 .XXXXX 5 XXXXXX 6 .XXXXX 7 ..XXXX 8  AA 9 AA 10 A 11 BB 12 B 13 B 14 B 15  CCC 16 CC 17 DD 18 D 19 DD 20 E 21 EE 22 EE</pre>	<pre>Menemukan path test case(,txt): C:\Mahasiswa\Semester 4\STIMA\Tut11\10220 ... waktu pencarian: 47 ms banyak kasus yang ditinjau: 102</pre>		Test case No. 8 Junior dari IQ Puzzler Pro 48 Free Challenges



## BAB V

### KESIMPULAN

Berdasarkan implementasi penyelesaian IQ Puzzler Pro dengan algoritma *brute force*, dapat disimpulkan bahwa algoritma *brute force* dapat digunakan untuk penyelesaian masalah tersebut. Walaupun begitu, terdapat algoritma-algoritma lain di luar sana yang lebih efektif untuk menyelesaikan masalah-masalah yang serupa. *Brute force* memakan waktu yang cukup lama, terutama apabila papan berukuran besar dan jumlah piece sangat banyak karena algoritma tersebut memeriksa segala kemungkinan.

Meskipun algoritma *brute force* cenderung kurang efisien, algoritma tersebut tetap menjadi pilihan algoritma yang baik untuk masalah-masalah sederhana. Untuk masalah yang lebih kompleks, dibutuhkan pendekatan algoritma yang lebih tepat, canggih, dan pintar untuk efisiensi waktu dan penyelesaiannya itu sendiri.

## LAMPIRAN

### 1. Repository

[https://github.com/SebastianHYH/Tucil1\\_13523070](https://github.com/SebastianHYH/Tucil1_13523070)

### 2. Tabel Pemeriksaan

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	✓	