

**LAPORAN TUGAS KECIL 1**

**IF2211 STRATEGI ALGORITMA**

**PENYELESAIAN IQ PUZZLER PRO DENGAN ALGORITMA**

**BRUTE FORCE**



Disusun Oleh:

Sakti Bimasena/13523053

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**JL. GANESA 10, BANDUNG 40132**

**2024**

<b>BAB I</b>	
<b>ALGORITMA BRUTE FORCE</b>	<b>3</b>
<b>BAB II</b>	
<b>SOURCE CODE PROGRAM</b>	<b>5</b>
2.1 Main	5
2.2 InputOutput	6
2.3 Shape	8
2.4 Board	9
2.5 Solver	10
2.6 Gambar	12
<b>BAB III</b>	
<b>UJI COBA</b>	<b>13</b>
<b>LAMPIRAN</b>	<b>19</b>

# **BAB I**

## **ALGORITMA BRUTE FORCE**

Pada program ini, digunakan algoritma brute force untuk menemukan salah satu solusi dari puzzle IQ Puzzler Pro. Langkah-langkah yang digunakan pada program ini adalah:

1. Program menerima file input .txt dari user pada folder test lalu memproses setiap blok puzzle dengan mengubah dan menyimpan semua bentuk transformasinya (rotasi dan pencerminan) ke dalam bentuk list koordinat. Semua list koordinat setiap blok disimpan lagi dalam list of list.
2. Program membentuk matrix char untuk papan permainan yang sesuai dengan panjang dan lebar papan di file input.
3. Program memulai penyelesaian puzzle dengan mentraversal setiap titik koordinat pada papan mulai dari (0,0) lanjut ke (0,1) dst. Program lalu mengambil bentuk blok pertama dan menentukan apakah terdapat posisi pemilihan transformasi dan pada bagian blok yang mana ditaruh pada koordinat tersebut dimana seluruh bagian blok bisa muat di dalam papan dan tidak mengenai blok yang lain.
4. Jika terdapat posisi yang memungkinkan, matrix papan di-update dengan huruf blok tersebut lalu blok tersebut dan transformasi-transformasinya di hapus dari list dengan fungsi bawaan remove dari java. Yang di remove hanya index dari blok tersebut sehingga bisa dikembalikan kepada list jika perlu backtracking.
5. Jika tidak, program lanjut ke blok puzzle selanjutnya.
6. Ketika terdapat saat dimana tidak ada kemungkinan penaruhan blok puzzle manapun yang tersisa pada koordinat tersebut, program memulai backtracking dengan menghapus blok puzzle yang terakhir di taruh dari matrix papan, mengembalikan lagi index blok tersebut pada list blok, lalu mencoba blok selanjutnya pada titik koordinat sebelumnya. Hal ini dapat dilakukan karena algoritma dibuat secara rekursif sehingga dapat dengan mudah kembali ke titik koordinat sebelumnya dan mencoba solusi lainnya.
7. Jika list blok sudah kosong, semua blok sudah ditaruh dan ditemukan solusi. Program menunjukkan solusi tersebut beserta dengan berapa lama dijalankan dan berapa banyak iterasi yang dilakukan.

8. Program dapat menyimpan solusi dalam bentuk file .txt dan .png berwarna pada folder test/Output/
9. Jika tidak ditemukan kemungkinan penaruhan blok manapun yang sesuai dengan peraturan permainan pada suatu titik koordinat, program akan menyatakan bahwa tidak ada solusi.
10. Jika terdapat blok puzzle yang tersisa pada saat papan sudah terisi dengan sempurna, program tidak akan menyatakan solusi itu valid karena masih ada sisa pada list blok.

## BAB II

### SOURCE CODE PROGRAM

#### 2.1 Main

```
1  import java.util.List;
2  import java.util.Scanner;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner input = new Scanner(System.in);
7          String filename;
8          System.out.print("Masukkan nama file .txt input yang berada di folder test: ");
9          filename = "test/" + input.nextLine() + ".txt";
10         InputOutput io = new InputOutput(filename);
11
12         int n = io.getBoardRows();
13         int m = io.getBoardCols();
14         List<Shape> shapes = io.getShapes();
15
16         Board board = new Board(n, m);
17         Solver solver = new Solver(board, shapes);
18
19         long startTime = System.nanoTime();
20
21         solver.solve();
22
23         long endTime = System.nanoTime();
24         long executionTime = (endTime - startTime)/1_000_000;
25
26         InputOutput.writeOutput(board, solver, executionTime);
27     }
28 }
```

## 2.2 InputOutput

```
1 public class InputOutput {
2     private int n, m, p;
3     private String type;
4     private List<Shape> shapes;
5
6     private static final String RESET = "\u001B[0m";
7     private static final String[] COLORS = {
8         "\u001B[31m", // Red
9         "\u001B[32m", // Green
10        "\u001B[33m", // Yellow
11        "\u001B[34m", // Blue
12        "\u001B[35m", // Magenta
13        "\u001B[36m", // Cyan
14        "\u001B[91m", // Bright Red
15        "\u001B[92m", // Bright Green
16        "\u001B[93m", // Bright Yellow
17        "\u001B[94m", // Bright Blue
18        "\u001B[95m", // Bright Magenta
19        "\u001B[96m", // Bright Cyan
20        "\u001B[41m", // Red Background
21        "\u001B[42m", // Green Background
22        "\u001B[43m", // Yellow Background
23        "\u001B[44m", // Blue Background
24        "\u001B[45m", // Magenta Background
25        "\u001B[46m", // Cyan Background
26        "\u001B[101m", // Bright Red Background
27        "\u001B[102m", // Bright Green Background
28        "\u001B[103m", // Bright Yellow Background
29        "\u001B[104m", // Bright Blue Background
30        "\u001B[105m", // Bright Magenta Background
31        "\u001B[106m", // Bright Cyan Background
32        "\u001B[1m", // Bold
33        "\u001B[4m" // Underline
34    };
35 }
```

```
1 public InputOutput(String filename){
2     shapes = new ArrayList<>();
3     parseFile(filename);
4 }
5
6 private void parseFile(String filename){
7     try {BufferedReader br = new BufferedReader(new FileReader(filename)) {
8         String[] firstLine = br.readLine().split(" ");
9         n = Integer.parseInt(firstLine[0]);
10        m = Integer.parseInt(firstLine[1]);
11        p = Integer.parseInt(firstLine[2]);
12
13        type = br.readLine().trim();
14
15        String line;
16        List<int[]> coordinates = new ArrayList<>();
17        char shapeLetter = 0;
18        int minCol = Integer.MAX_VALUE;
19        int shapeStartRow = 0;
20        while ((line = br.readLine()) != null){
21            if (line.trim().isEmpty()) continue;
22
23            int firstNonSpaceIdx = -1;
24            for (int i = 0; i < line.length(); i++) {
25                if (line.charAt(i) != ' ') {
26                    firstNonSpaceIdx = i;
27                    break;
28                }
29            }
30            if (firstNonSpaceIdx == -1) continue;
31
32            char firstChar = line.charAt(firstNonSpaceIdx);
33
34            if (shapeLetter == 0 || firstChar != shapeLetter){
35                if (!coordinates.isEmpty()){
36                    for (int[] coord : coordinates){
37                        coord[1] -= minCol;
38                    }
39                    shapes.add(new Shape(shapeLetter, new ArrayList<>(coordinates)));
40                    coordinates.clear();
41                }
42                shapeLetter = firstChar;
43                minCol = Integer.MAX_VALUE;
44                shapeStartRow = coordinates.isEmpty() ? 0 : shapeStartRow + 1;
45            }
46
47            char[] chars = line.toCharArray();
48            for (int col = 0; col < chars.length; col++){
49                if (chars[col] == shapeLetter){
50                    if (shapeStartRow == -1) shapeStartRow = 0;
51                    coordinates.add(new int[] {shapeStartRow, col});
52                    minCol = Math.min(minCol, col);
53                }
54            }
55            shapeStartRow++;
56        }
57
58        if (!coordinates.isEmpty()){
59            for (int[] coord : coordinates){
60                coord[1] -= minCol;
61            }
62            shapes.add(new Shape(shapeLetter, new ArrayList<>(coordinates)));
63        }
64    } catch (IOException e) {
65        e.printStackTrace();
66    }
67 }
68 }
```

```

1 public static void writeOutput(Board board, Solver solver, long time) {
2     System.out.println();
3     if(solver.found){
4         char[][] grid = board.getGrid();
5
6         StringBuilder content = new StringBuilder();
7         for (char[] row : grid) {
8             StringBuilder outputRow = new StringBuilder();
9             for (char cell : row) {
10                 if (cell == ',') {
11                     outputRow.append(cell);
12                 } else {
13                     String color = colorMap.getOrDefault(cell, RESET);
14                     outputRow.append(color).append(cell).append(RESET);
15                 }
16                 content.append(cell);
17             }
18             content.append("\n");
19             System.out.println(outputRow);
20         }
21         content.append("\n");
22         System.out.println();
23         System.out.println("Waktu pencarian: " + time + " ms");
24         content.append("Waktu pencarian: " + time + " ms\n\n");
25         System.out.println();
26         System.out.println("Banyak kasus yang ditinjau: " + solver.getTries());
27         content.append("Banyak kasus yang ditinjau: " + solver.getTries()+ "\n\n");
28         System.out.println();
29
30         Scanner scanner = new Scanner(System.in);
31         String choice;
32         while (true) {
33             System.out.print("Apakah anda ingin menyimpan solusi? (ya/tidak: ");
34             choice = scanner.nextLine().trim().toLowerCase();
35
36             if (choice.equals("ya") || choice.equals("tidak")) {
37                 break;
38             }
39             System.out.println("Input tidak valid! Harap masukkan 'ya' atau 'tidak'.");
40         }
41
42         if (choice.equals("ya")) {
43             System.out.print("Masukkan nama file untuk menyimpan hasil (namafile.txt): ");
44             String fileName = scanner.next();
45
46             String directoryPath = "test/Output/";
47             String extension = ".txt";
48             try (BufferedWriter writer = new BufferedWriter(new FileWriter(directoryPath+fileName+extension))) {
49                 writer.write(content.toString());
50             } catch (IOException e) {
51                 System.out.println("Terjadi kesalahan saat menulis ke file: " + e.getMessage());
52             }
53
54             extension = ".png";
55             Gambar.saveImage(board.getGrid(), fileName+extension);
56             System.out.println("Hasil berhasil disimpan ke dalam file " + fileName);
57         } else if (choice.equals("tidak")) {
58             System.out.println("Solusi tidak disimpan.");
59         }
60
61         scanner.close();

```

```

1 } else {
2     StringBuilder content = new StringBuilder();
3     System.out.println("Tidak ada solusi yang ditemukan");
4     content.append("Tidak ada solusi yang ditemukan\n\n");
5     System.out.println();
6     System.out.println("Waktu pencarian: " + time + " ms");
7     content.append("Waktu pencarian: " + time + " ms\n\n");
8     System.out.println();
9     System.out.println("Banyak kasus yang ditinjau: " + solver.getTries());
10    content.append("Banyak kasus yang ditinjau: " + solver.getTries()+ "\n\n");
11    System.out.println();
12    Scanner scanner = new Scanner(System.in);
13    String choice;
14    while (true) {
15        System.out.print("Apakah anda ingin menyimpan solusi? (ya/tidak: ");
16        choice = scanner.nextLine().trim().toLowerCase();
17
18        if (choice.equals("ya") || choice.equals("tidak")) {
19            break;
20        }
21        System.out.println("Input tidak valid! Harap masukkan 'ya' atau 'tidak'.");
22    }
23
24    if (choice.equals("ya")) {
25        System.out.print("Masukkan nama file untuk menyimpan hasil (namafile.txt): ");
26        String fileName = scanner.next();
27
28        String directoryPath = "test/Output/";
29        String extension = ".txt";
30        try (BufferedWriter writer = new BufferedWriter(new FileWriter(directoryPath+fileName+extension))) {
31            writer.write(content.toString());
32        } catch (IOException e) {
33            System.out.println("Terjadi kesalahan saat menulis ke file: " + e.getMessage());
34        }
35
36        System.out.println("Hasil berhasil disimpan ke dalam file " + fileName);
37    } else if (choice.equals("tidak")) {
38        System.out.println("Solusi tidak disimpan.");
39    }
40    scanner.close();
41 }

```

## 2.3 Shape

```
1 public class Shape {
2     private char letter;
3     private List<int[]> normCoords;
4     private List<List<int[]>> transformations;
5
6     public Shape(char letter, List<int[]> coords) {
7         this.letter = letter;
8         this.normCoords = normalize(coords);
9         this.transformations = genTransforms();
10    }
11
12    private List<int[]> normalize(List<int[]> coords){
13        int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE;
14        for (int[] point : coords) {
15            minX = Math.min(minX, point[0]);
16            minY = Math.min(minY, point[1]);
17        }
18        List<int[]> normalized = new ArrayList<>();
19        for (int[] point : coords) {
20            normalized.add(new int[]{point[0] - minX, point[1] - minY});
21        }
22        return normalized;
23    }
24
25    private List<int[]> rotate(List<int[]> coords){
26        List<int[]> result = new ArrayList<>();
27        for (int[] point : coords){
28            result.add(new int[]{point[1], -point[0]});
29        }
30        return normalize(result);
31    }
32
33    private List<int[]> mirror(List<int[]> coords){
34        List<int[]> result = new ArrayList<>();
35        for (int[] point : coords){
36            result.add(new int[]{point[0], -point[1]});
37        }
38        return normalize(result);
39    }
```

```
1 private List<List<int[]>> genTransforms(){
2     List<List<int[]>> results = new ArrayList<>();
3     List<int[]> current = normCoords;
4
5     for(int i = 0; i<4; i++){
6         results.add(current);
7         current = rotate(current);
8     }
9
10    current = mirror(normCoords);
11    for(int i = 0; i<4; i++){
12        results.add(current);
13        current = rotate(current);
14    }
15
16    return results;
17 }
18
```



## 2.4 Board

```
1 public class Board {
2     private int rows, cols;
3     private char[][] grid;
4
5     public Board(int rows, int cols){
6         this.rows = rows;
7         this.cols = cols;
8         this.grid = new char[rows][cols];
9
10        for (int i = 0; i < rows; i++){
11            Arrays.fill(grid[i], '.');
12        }
13    }
14
15    public boolean canPlaceShape(Shape shape, int x, int y, int rotation) {
16        List<int[]> shapeCoords = shape.getCoords(rotation);
17
18        for (int[] coordinates : shapeCoords){
19            int newRow = coordinates[0] + x;
20            int newCol = coordinates[1] + y;
21
22            if (newRow < 0 || newRow >= rows || newCol < 0 || newCol >= cols){
23                return false;
24            }
25
26            if (!isEmpty(newRow, newCol)){
27                return false;
28            }
29        }
30        return true;
31    }
```

```
1 public boolean placeShape(Shape shape, int x, int y, int rotation) {
2     if (canPlaceShape(shape, x, y, rotation)){
3         char letter = shape.getLetter();
4         List<int[]> shapeCoords = shape.getCoords(rotation);
5
6         for (int[] coordinates : shapeCoords){
7             int newRow = coordinates[0] + x;
8             int newCol = coordinates[1] + y;
9
10            grid[newRow][newCol] = letter;
11        }
12        return true;
13    }
14
15    return false;
16 }
17
18 public void removeShape(Shape shape, int x, int y, int rotation) {
19     for (int[] offset : shape.getCoords(rotation)) {
20         int newX = x + offset[0];
21         int newY = y + offset[1];
22         grid[newX][newY] = '.';
23     }
24 }
```

## 2.5 Solver

```
1 private boolean solveRecursive(List<Shape> remainingShapes, int x, int y) {
2     if (remainingShapes.isEmpty()) {
3         return true;
4     }
5
6     if (x >= board.getRows()) {
7         return false;
8     }
9
10    int nextX = x, nextY = y + 1;
11    if (nextY >= board.getCols()) {
12        nextX = x + 1;
13        nextY = 0;
14    }
15
16    if (!board.isEmpty(x, y)) {
17        return solveRecursive(remainingShapes, nextX, nextY);
18    }
19
20    for (int i = 0; i < remainingShapes.size(); i++) {
21        Shape shape = remainingShapes.get(i);
22
23        for (int r = 0; r < shape.getTransformationCount(); r++) {
24            tries++;
25            if (board.canPlaceShape(shape, x, y, r)) {
26                board.placeShape(shape, x, y, r);
27                remainingShapes.remove(i);
28
29                if (solveRecursive(remainingShapes, nextX, nextY)) {
30                    return true;
31                }
32                board.removeShape(shape, x, y, r);
33                remainingShapes.add(i, shape);
34            }
35        }
36    }
37
38    return false;
39 }
```

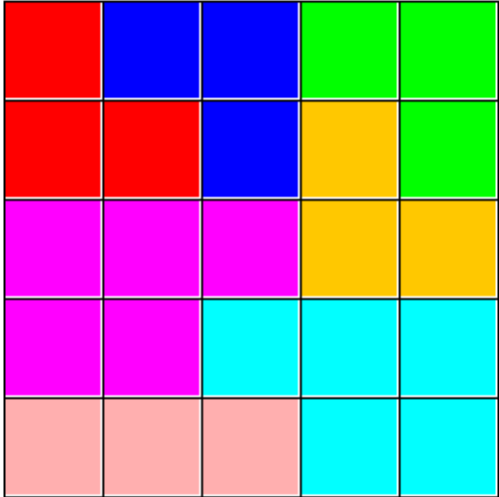
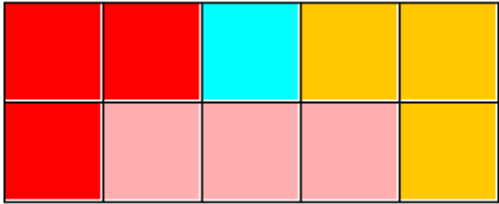


## 2.6 Gambar

```
1 public static void saveImage(char[][] grid, String filename) {
2     int rows = grid.length;
3     int cols = grid[0].length;
4
5     int width = cols * CELL_SIZE + 2 * PADDING;
6     int height = rows * CELL_SIZE + 2 * PADDING;
7
8     BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
9     Graphics2D g = image.createGraphics();
10
11     g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
12
13     g.setColor(Color.WHITE);
14     g.fillRect(0, 0, width, height);
15
16     for(int r = 0; r < rows; r++){
17         for(int c = 0; c < cols; c++){
18             int x = PADDING + c * CELL_SIZE;
19             int y = PADDING + r * CELL_SIZE;
20
21             g.setColor(Color.BLACK);
22             g.drawRect(x, y, CELL_SIZE, CELL_SIZE);
23
24             if (grid[r][c] != '.'){
25                 g.setColor(getColorForShape(grid[r][c]));
26                 g.fillRect(x + 1, y + 1, CELL_SIZE - 2, CELL_SIZE - 2);
27             }
28         }
29     }
30
31     g.dispose();
32
33     String directory = "test/Output/";
34     File outputFile = new File(directory + filename);
35     try {
36
37         ImageIO.write(image, "png", outputFile);
38         System.out.println("Grid saved as " + filename);
39     } catch (Exception e) {
40         e.printStackTrace();
41     }
42 }
```

# BAB III

## UJI COBA

Input	Output
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	 <div> <p>             ABBCC              AABDC              EEEDD              EEFFF              GGGFF           </p> <p>Waktu pencarian: 5 ms</p> <p>Banyak kasus yang ditinjau: 6235</p> </div>
2 5 4 DEFAULT A AA GGG D DD F	 <div> <p>             AAFDD              AGGGD           </p> <p>Waktu pencarian: 1 ms</p> <p>Banyak kasus yang ditinjau: 2407</p> </div>

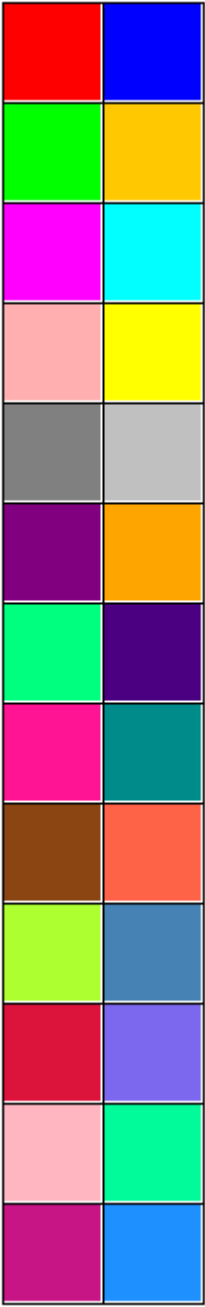
3 3 3 DEFAULT AAA BBBBB C	<div>Tidak ada solusi yang ditemukan</div> <div>Waktu pencarian: 1 ms</div> <div>Banyak kasus yang ditinjau: 1048</div>
11 5 12 DEFAULT AAA A A BB BB B C CC C DD DD E E E EE FF FFF G GG HHH H H I III J JJ JJ KKK KK LL L L	

AAEE  
ACGE  
ACGE  
FCDE  
FFJD  
IFJJ  
IFJH  
IIHH  
IBLL  
KKBLL  
KKBL

Waktu pencarian: 26862 ms

Banyak kasus yang ditinjau: 1366266194

13 2 26  
DEFAULT  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z





	<div> <div> AB CD EF GH IJ KL MN OP QR ST UV WX YZ </div> <div> Waktu pencarian: 0 ms  Banyak kasus yang ditinjau: 26 </div> </div>
3 3 4 DEFAULT AA  BB CC DDD	<div> </div> <div> <div> AAB CCB DDD </div> <div> Waktu pencarian: 0 ms  Banyak kasus yang ditinjau: 5 </div> </div>
3 3 5 DEFAULT A A A A B C D E	<div> </div>

	<pre>.AB ACA DAE  Waktu pencarian: 0 ms Banyak kasus yang ditinjau: 5</pre>
--	---

## LAMPIRAN

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

Pranala Github: [https://github.com/sbimasena/Tucil1\\_13523053](https://github.com/sbimasena/Tucil1_13523053)