

Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II tahun 2024/2025

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

Disusun oleh:

Abdullah Farhan 13523042



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG 2022**

## A. Algoritma Bruteforce

Algoritma brute force singkatnya ialah cara penyelesaian masalah dengan program yang meninjau semua kasus yang mungkin muncul berdasarkan deskripsi masalah yang diberikan.

Ada banyak approach dalam menyelesaikan suatu masalah dengan algoritma brute force, karena meskipun tujuan akhirnya sama, dalam mencapai tujuan akhir tersebut ada banyak cara yang bisa diambil.

Tahapan utama dalam menyelesaikan IQ Puzzle Pro terdiri atas beberapa tahap diantaranya Pembacaan File, Pencarian Jawaban, Serta Output.

### Pembacaan File

Pada tahap pembacaan file yang ada pada class InputParser dengan nama method yaitu parseInput dan getBlockObject, Program membaca input baris demi baris lalu diubah kedalam Array of Array of character yang ada pada class Block Bernama block.

### Struktur Data dan Spesifikasi Fungsi

```
{ Modul ADT Block }
type Block: < id: character,
             shape: array [0..MAX_SIZE-1, 0..MAX_SIZE-1] of character,
             height: integer,
             width: integer >

{ Modul ADT Board }
type Board: < N, M: integer,
             grid: array [0..MAX_N-1, 0..MAX_M-1] of character >

{ Modul ADT Solver }
type Solver: < board: Board,
             blocks: array [0..MAX_BLOCKS-1] of Block,
             nBlocks: integer,
             iterations: integer >

{ *** Konstruktor Block *** }
procedure CreateBlock(output b: Block, input id: character, input shape: array [0..MAX_SIZE-1, 0..MAX_SIZE-1] of character)
{ I.S.: Sembarang }
{ F.S.: Block terbentuk dengan id dan shape yang diberikan }

{ *** Selektor Block *** }
function getId(b: Block) → character
function getShape(b: Block) → array [0..MAX_SIZE-1, 0..MAX_SIZE-1] of character
function getHeight(b: Block) → integer
function getWidth(b: Block) → integer

{ *** Transformasi Block *** }
function rotate(b: Block) → Block
{ Mengembalikan block yang sudah dirotasi 90 derajat searah jarum jam }

function mirror(b: Block) → Block
{ Mengembalikan block yang sudah dicerminkan secara horizontal (kanan-kiri) }

function getAllTransformations(b: Block) → array [0..7] of Block
{ Mengembalikan semua kemungkinan transformasi dari block (rotasi dan pencerminan) }

{ *** Konstruktor Board *** }
procedure CreateBoard(output b: Board, input N, M: integer)
{ I.S.: Sembarang }
{ F.S.: Board terbentuk dengan ukuran N x M, semua sel berisi '.' }
```

```

    { *** Operasi Board *** }
    function canPlace(b: Board, block: Block, startX, startY: integer) → boolean
    { Mengembalikan true jika block dapat ditempatkan pada posisi (startX, startY) }

    procedure placeBlock(input/output b: Board, input block: Block, input startX, startY:
integer)
    { I.S.: Board terdefinisi, posisi valid untuk menempatkan block }
    { F.S.: board Block ditempatkan pada pada posisi yang ditentukan }

    procedure removeBlock(input/output b: Board, input block: Block, input startX, startY:
integer)
    { I.S.: Block sudah ada di board pada posisi yang ditentukan }
    { F.S.: Block dihapus dari board }

    { *** Konstruktor Solver *** }
    procedure CreateSolver(output s: Solver, input b: Board, input blocks: array [0..MAX_BLOCKS-
1] of Block, input nBlocks: integer)
    { I.S.: Sembarang }
    { F.S.: Solver terbentuk dengan board dan blocks yang diberikan }

    { *** Solver Algorithm *** }
    function solve(input/output s: Solver, input index, startX, startY: integer) → boolean
    { Mencoba menyelesaikan puzzle dengan backtracking. Mengembalikan true jika solusi ditemukan
}

```

## Implementasi Fungsi Utama

```

{ Implementasi solve }
function solve(input/output s: Solver, input index: integer) → boolean
{ Fungsi rekursif untuk mencoba menempatkan blok pada setiap posisi yang valid }
KAMUS LOKAL
    block: Block
    transformations: array [0..7] of Block { maksimal 8 transformasi }
    nTransformations: integer
    row, col: integer
    transformedBlock: Block
    canSolve: boolean

ALGORITMA
    if index = s.nBlocks then
        → true { Basis: semua blok berhasil ditempatkan }

    block ← s.blocks[index]
    transformations ← getAllTransformations(block)
    nTransformations ← length(transformations)

    { Coba setiap transformasi blok }
    i traversal [0..nTransformations-1]
        transformedBlock ← transformations[i]

        { Coba setiap posisi pada papan }
        row traversal [0..s.board.N - transformedBlock.height]
            col traversal [0..s.board.M - transformedBlock.width]
                s.iterations ← s.iterations + 1

                if canPlace(s.board, transformedBlock, row, col) then
                    placeBlock(s.board, transformedBlock, row, col)
                    canSolve ← solve(s, index + 1)

                    if canSolve then
                        → true

                    removeBlock(s.board, transformedBlock, row, col) { Backtrack }

    → false { Tidak ditemukan penempatan valid untuk blok saat ini }

```

```

{ Implementasi rotate }
function rotate(b: Block) → Block
KAMUS LOKAL
    result: Block
    i, j: integer
ALGORITMA
    CreateBlock(result, b.id, [[]])
    i traversal [0..b.height-1]
        j traversal [0..b.width-1]
            result.shape[j][b.height-1-i] ← b.shape[i][j]
    result.height ← b.width
    result.width ← b.height
    → result

```

```

{ Implementasi mirror }
function mirror(b: Block) → Block
KAMUS LOKAL
    result: Block
    i, j: integer
ALGORITMA
    CreateBlock(result, b.id, [[]])
    i traversal [0..b.height-1]
        j traversal [0..b.width-1]
            result.shape[i][b.width-1-j] ← b.shape[i][j]
    result.height ← b.height
    result.width ← b.width
    → result

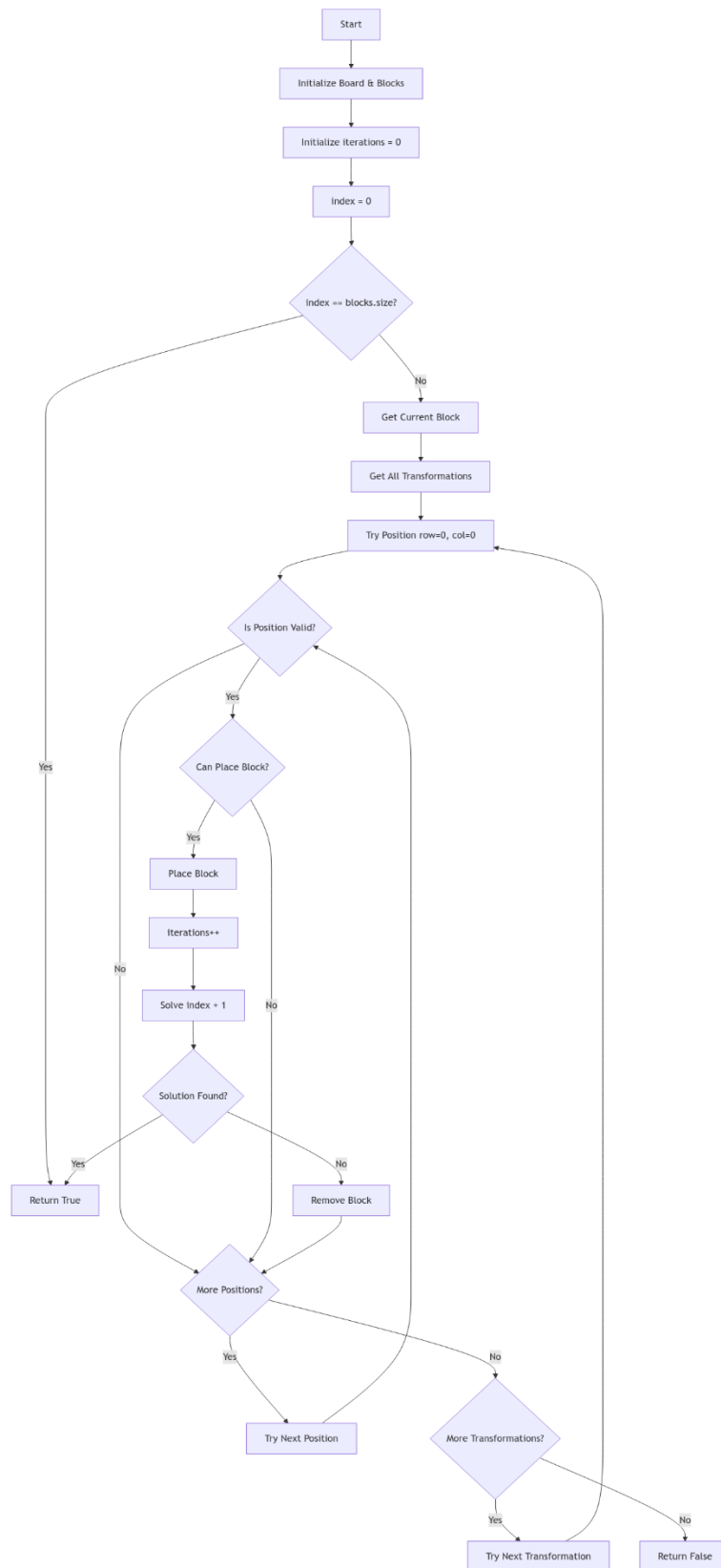
```

```

{ Implementasi canPlace }
function canPlace(b: Board, block: Block, startX, startY: integer) → boolean
KAMUS LOKAL
    i, j: integer
ALGORITMA
    i traversal [0..block.height-1]
        j traversal [0..block.width-1]
            if startX + i ≥ b.N or startY + j ≥ b.M or
               board[startX + i][startY + j] != '.' then
                → false
            if board[startX + i][startY + j] == '*' && block.getShape()[i][j] != '*' then
                → false {Untuk Handle Yang Custom, Jika '*' tidak bisa diisi}
    → true

```

## Flowchart Algorithm



## Intuisi

Ide utama dalam menyelesaikan **IQ Puzzler Pro** adalah menggunakan **backtracking**, yaitu mencoba berbagai kemungkinan untuk setiap blok dan kembali ke langkah sebelumnya jika menemui jalan buntu.

Pada dasarnya, kita mencoba **meletakkan setiap blok satu per satu** dengan semua kemungkinan transformasi (rotasi dan pencerminan) hingga menemukan susunan yang sesuai.

## Pencarian Jawaban

Kita diberikan papan IQ Puzzler Pro yang kosong dan sekumpulan blok. Pendekatan penyelesaiannya adalah sebagai berikut:

### 1. Menempatkan Blok:

- Kita mulai dari blok pertama dan mencoba menempatkannya di berbagai posisi pada papan.
- Setiap blok dapat **diputar dan dicerminkan**, sehingga kita harus mempertimbangkan semua kemungkinan transformasinya. Hasil transformasi yang dipilih yang unik saja, contoh:

Blok : AA

Banyaknya hasil transformasi hanya 2 bukan 8

### 2. Memeriksa Kecocokan:

- Sebelum menempatkan blok, kita menggunakan metode `canPlace()` untuk memeriksa apakah blok tersebut bisa diletakkan di lokasi tertentu tanpa melanggar aturan.

### 3. Validasi Penempatan:

- Jika blok bisa ditempatkan, kita menambahkannya ke papan menggunakan `placeBlock()`.
- Algoritma kemudian melanjutkan secara **rekursif** untuk mencoba menempatkan blok berikutnya.

### 4. Backtracking:

- Jika menempatkan blok saat ini tidak menghasilkan solusi yang valid (rekursi mengembalikan **false**), kita akan **menghapus blok tersebut** dari papan menggunakan `removeBlock()`.
- Kemudian, kita mencoba kemungkinan posisi lainnya atau transformasi blok berikutnya.

### 5. Terminasi:

- Rekursi berakhir ketika semua blok berhasil ditempatkan dengan benar, yang berarti teka-teki telah terselesaikan.
- Algoritma ini memastikan bahwa solusi yang diperoleh adalah solusi valid pertama yang memenuhi kriteria, tanpa mencari solusi yang lain terlebih dahulu sehingga bisa lebih cepat dari brute force yang asli karena ada “early stopping”.

---

## Kompleksitas

- **Kompleksitas Waktu: Eksponensial** dalam jumlah blok dan transformasinya. Sekitar  $O(8^P \times N \times M)$ , di mana N dan M merupakan dimensi dari papan, dan P adalah banyaknya *Piece*.
- **Kompleksitas Ruang:  $O(\text{jumlah blok yang ditempatkan})$**  karena kita hanya menyimpan status papan saat ini dalam rekursi.

## Output

Pada tahap ini susunan board yang ditemukan akan di cetak berwarna sesuai urutan warna yang ada pada class OutputHandler di bagian static.

## B. Contoh Konkret Langkah-langkah Algoritma

### a. Initial State

Misalkan kita memiliki board 3x3 dan 2 block:

- Block A: bentuk L (2x2)
- Block B: bentuk I (1x2)

### b. Langkah 1: Mencoba Block A

- Generate transformasi Block A (rotasi dan mirror)
- Coba posisi (0,0):

A A .

A . .

. . .

- Jika valid, lanjut ke Block B
- Jika tidak valid atau tidak menghasilkan solusi, backtrack dan coba posisi lain

### c. Langkah 2: Mencoba Block B

- Generate transformasi Block B
- Dengan Block A di posisi (0,0), coba tempatkan B:

A A .    A A .    A A .

A . B   or A B B   or A . .

. . B    . . .    . B B

### d. Langkah 3: Backtracking

Jika tidak ada posisi valid untuk Block B:

- Hapus Block A dari posisi sekarang
- Coba posisi lain atau transformasi lain dari Block A
- Ulangi proses untuk Block B

e. Langkah 4: Solusi Ditemukan

Ketika semua block berhasil ditempatkan dengan valid, solusi ditemukan.

f. Kompleksitas pada Contoh

- Block A: 8 transformasi possible ( $4 \text{ rotasi} \times 2 \text{ mirror}$ )
- Block B: 2 transformasi possible ( $1 \text{ rotasi} \times 2 \text{ mirror}$ )
- Total iterasi maksimal:  $8 \times 9 \text{ posisi} \times 2 \times 9 \text{ posisi} = 1,296$  kemungkinan

### **C. Source Code Program**

1. Solver/Solver.java:



```

1  package Solver;
2
3  import java.util.List;
4
5  import DataStructure.Block;
6  import DataStructure.Board;
7
8  public class Solver {
9      private Board board;
10     private List<Block> blocks;
11     private int iterations;
12
13     public Solver(Board board, List<Block> blocks) {
14         this.board = board;
15         this.blocks = blocks;
16         this.iterations = 0;
17     }
18
19     public boolean solve(int index) {
20         if (index == blocks.size()) {
21             return true;
22         }
23
24         Block block = blocks.get(index);
25         List<Block> transformations = block.getAllTransformations();
26
27         for (Block transformedBlock : transformations) {
28             for (int row = 0; row <= board.getN() - block.getHeight(); row
29 ++)) {
30                 for (int col = 0; col <= board.getM() - block.getWidth();
31 col++) {
32                     iterations++;
33                     if (board.canPlace(transformedBlock, row, col)) {
34                         board.placeBlock(transformedBlock, row, col);
35                         if (solve(index + 1)) {
36                             return true; // Solution found
37                         }
38                     }
39                 }
40             }
41         }
42         return false;
43     }
44
45     public int getIterations() {
46         return iterations;
47     }
48 }

```

2. DataStructure/Block.java:

```

1 public class Block {
2     private char id;
3     private char[][] shape;
4
5     public Block(char id, char[][] shape) {
6         this.id = id;
7         this.shape = shape;
8     }
9
10    public char getId() { return id; }
11    public char[][] getShape() { return shape; }
12    public int getHeight() { return shape.length; }
13    public int getWidth() { return shape[0].length; }
14
15    public Block rotate() {
16        int rows = shape.length, cols = shape[0].length;
17        char[][] rotated = new char[cols][rows];
18        for (int i = 0; i < rows; i++)
19            for (int j = 0; j < cols; j++)
20                rotated[j][rows - 1 - i] = shape[i][j];
21        return new Block(id, rotated);
22    }
23
24    public Block mirror() {
25        int rows = shape.length, cols = shape[0].length;
26        char[][] mirrored = new char[rows][cols];
27        for (int i = 0; i < rows; i++)
28            for (int j = 0; j < cols; j++)
29                mirrored[i][cols - 1 - j] = shape[i][j];
30        return new Block(id, mirrored);
31    }
32
33    public void displayBlock() {
34        for (char[] row : shape) {
35            for (char cell : row) {
36                System.out.print(cell + " ");
37            }
38            System.out.println();
39        }
40    }
41
42    public boolean equals(Object obj) {
43        if (this == obj) return true;
44        if (obj == null || getClass() != obj.getClass()) return false;
45        Block block = (Block) obj;
46        return Arrays.deepEquals(this.shape, block.shape);
47    }
48
49    @Override
50    public int hashCode() {
51        return Arrays.deepHashCode(this.shape);
52    }
53
54    public List<Block> getAllTransformations() {
55        Set<Block> uniqueTransformations = new HashSet<>();
56        List<Block> transformations = new ArrayList<>();
57
58        Block current = this;
59        for (int i = 0; i < 4; i++) {
60            if (uniqueTransformations.add(current)) {
61                transformations.add(current);
62            }
63
64            Block mirrored = current.mirror();
65            if (uniqueTransformations.add(mirrored)) { // untuk delete duplikasi
66                transformations.add(mirrored);
67            }
68
69            current = current.rotate();
70        }
71        return transformations;
72    }
73
74    // untuk debugging aja
75    @Override
76    public String toString() {
77        StringBuilder sb = new StringBuilder();
78        sb.append("Block{");
79        sb.append("id=").append(id).append(", ");
80        sb.append("shape=");
81        for (char[] row : shape) {
82            sb.append(Arrays.toString(row)).append(", ");
83        }
84        sb.delete(sb.length() - 2, sb.length());
85        sb.append("}");
86        return sb.toString();
87    }
88 }

```

### 3. DataStructure/Board.java:

```

1 package DataStructure;
2 // package Board;
3 import java.util.*;
4
5 public class Board {
6     private int N, M;
7     private char[][] board;
8
9     public Board(int N, int M) {
10         this.N = N;
11         this.M = M;
12         this.board = new char[N][M];
13         for (char[] row : board) {
14             Arrays.fill(row, '.');
15         }
16     }
17     public char getCell(int i, int j) {
18         return board[i][j];
19     }
20     public int getN() { return this.N; }
21     public int getM() { return this.M; }
22
23     public void setCell (int i, int j, char c) {
24         board[i][j] = c;
25     }
26     public boolean isOutOfBounds(int x, int y) {
27         if (x < 0 || y < 0 || x >= N || y >= M) {
28             return true;
29         }
30         return false;
31     }
32
33     public boolean canPlace(Block block, int startX, int startY) {
34         try {
35             for (int i = 0; i < block.getHeight(); i++) {
36                 for (int j = 0; j < block.getWidth(); j++) {
37                     if (block.getShape()[i][j] != '.') {
38                         if (startX + i >= N || startY + j >= M || board[
startX + i][startY + j] != '.') {
39                             return false;
40                         }
41                         if (board[startX + i][startY + j] == '*' && block.
getShape()[i][j] != '*') {
42                             System.out.println("
Error: Cannot place block on '*' cell.");
43                             return false;
44                         }
45                     }
46                 }
47             }
48         } catch (ArrayIndexOutOfBoundsException e) {
49             return false;
50         }
51         return true;
52     }
53
54     public void placeBlock(Block block, int startX, int startY) {
55         for (int i = 0; i < block.getHeight(); i++) {
56             for (int j = 0; j < block.getWidth(); j++) {
57                 if (block.getShape()[i][j] != '.') {
58                     board[startX + i][startY + j] = block.getId();
59                 }
60             }
61         }
62     }
63
64     public void removeBlock(Block block, int startX, int startY) {
65         for (int i = 0; i < block.getHeight(); i++) {
66             for (int j = 0; j < block.getWidth(); j++) {
67                 if (block.getShape()[i][j] != '.') {
68                     board[startX + i][startY + j] = '.';
69                 }
70             }
71         }
72     }
73
74     public void displayBoard() {
75         for (char[] row : board) {
76             for (char cell : row) {
77                 System.out.print(cell + " ");
78             }
79             System.out.println();
80         }
81     }
82
83     public void setGrid(char[][] grid) {
84         this.board = grid;
85     }
86 }
87

```

#### 4. IO/InputParser.java:

```
1 public class InputParser {
2     private int N, M, P;
3     private String caseType
4 ;
5     private List<List<
6 String>> blocks;
7     private Board board;
8
9     public InputParser() {
10         blocks = new
11 ArrayList<>();
12         board = new Board(N
13 , M);
14     }
```

```
1 private char getFirstValidChar(String str) {
2     for (char c : str.toCharArray()) {
3         if (c != ' ' && c != '\n' && c != '\r' && c != '\0') {
4             return c;
5         }
6     }
7     throw new RuntimeException("Tidak dapat menemukan karakter blok yang valid.");
8 }
9
10 public int getN() { return N; }
11 public int getM() { return M; }
12 public int getP() { return P; }
13 public String getCaseType() { return caseType; }
14 public List<List<String>> getBlocks() { return blocks; }
15
16 public Board getBoard() {
17     return board;
18 }
```

```

1 public void parseInput(String filePath) throws IOException {
2     BufferedReader br = new BufferedReader(new FileReader(filePath));
3     String[] dimensions = br.readLine().split(" ");
4     if (dimensions.length < 3) {
5         br.close();
6         throw new IllegalArgumentException("
7         Error: Dimensi papan tidak ditemukan dalam file atau tidak lengkap.");
8     }
9     try {
10         N = Integer.parseInt(dimensions[0]);
11         M = Integer.parseInt(dimensions[1]);
12         P = Integer.parseInt(dimensions[2]);
13     } catch (NumberFormatException e) {
14         br.close();
15         throw new IllegalArgumentException("
16         Error: Nilai N, M, atau P tidak ditemukan dalam file atau bukan bilangan bulat.");
17     }
18     if (P > 26) {
19         br.close();
20         throw new IllegalArgumentException("Error: Jumlah blok (" + P + "
21         ) melebihi batas maksimum (26).");
22     }
23     caseType = br.readLine().replace("\r", "").replace("\0", "");
24     if (!caseType.equals("DEFAULT") && !caseType.equals("CUSTOM")) {
25         br.close();
26         throw new IllegalArgumentException("Error: Case type tidak valid.");
27     }
28     Board board = new Board(N, M);
29     if (caseType.equals("CUSTOM")) {
30         char[][] initialGrid = new char[N][M];
31         for (int i = 0; i < N; i++) {
32             String line = br.readLine().replace("\r", "").replace("\0", "");
33             for (int j = 0; j < M; j++) {
34                 initialGrid[i][j] = (line.charAt(j) == '.' ? '*' : '.');
35             }
36         }
37         board.setGrid(initialGrid);
38     }
39     this.board = board;
40
41     List<String> lines = new ArrayList<>();
42     String line;
43     while ((line = br.readLine()) != null) {
44         line = line.replace("\r", "").replace("\0", "");
45         if (!line.isEmpty()) {
46             lines.add(line);
47         }
48     }
49     br.close();
50
51     if (lines.isEmpty()) {
52         throw new IOException("Error: Tidak ada data blok dalam file.");
53     }
54
55     char currentBlockId = getFirstValidChar(lines.get(0));
56     List<String> currentBlock = new ArrayList<>();
57
58     for (String l : lines) {
59         l = l.replace("\r", "").replace("\0", "");
60         if (l.isEmpty()) continue;
61
62         for (int i = 0; i < l.length(); i++) {
63             if (l.charAt(i) != currentBlockId && l.charAt(i) != ' ') {
64                 if (!currentBlock.isEmpty()) {
65                     blocks.add(new ArrayList<>(currentBlock));
66                     currentBlock.clear();
67                 }
68                 currentBlockId = l.charAt(i);
69             }
70         }
71         currentBlock.add(l);
72     }
73     if (!currentBlock.isEmpty()) {
74         blocks.add(currentBlock);
75     }
76     if (blocks.size() != P) {
77         throw new IOException("Jumlah blok yang terbaca tidak sesuai dengan P");
78     }
79 }

```

```

1  public List<Block> getBlockObjects() {
2      List<Block> blockObjects = new ArrayList<>();
3      for (List<String> blockShape : blocks) {
4          int height = blockShape.size();
5          int max_width = blockShape.stream().mapToInt(String::length).max().orElse(0);
6          char[][] shape = new char[height][max_width];
7          char blockId = getBlockId(blockShape);
8
9          for (int i = 0; i < height; i++) {
10             for (int j = 0; j < max_width; j++) {
11                 if (j < blockShape.get(i).length() && blockShape.get(i).charAt(j) ==
12 ' ' ) {
13                     shape[i][j] = '.';
14                 }
15                 else if (j < blockShape.get(i).length() && blockShape.get(i).charAt(j)
16 ) == blockId) {
17                     shape[i][j] = blockShape.get(i).charAt(j);
18                 } else {
19                     shape[i][j] = '.';
20                 }
21             }
22             blockObjects.add(new Block(blockId, shape));
23         }
24         return blockObjects;
25     }
26
27     private char getBlockId(List<String> blockShape) {
28         for (String row : blockShape) {
29             for (char c : row.toCharArray()) {
30                 if (c != ' ' && c != '.') {
31                     return c;
32                 }
33             }
34         }
35         throw new RuntimeException("Tidak dapat menemukan karakter blok");
36     }

```

## 5. IO/OutputHandler.java:

```

public class OutputHandler {
    // ANSI color codes for terminal output
    private static final String RESET = "\u001B[0m";
    private static final Map<Character, String> BLOCK_COLORS = new HashMap<>();
    private static final Map<Character, Color> IMAGE_COLORS = new HashMap<>();
    private static final int CELL_SIZE = 50; // Size of each cell in pixel
    }

    static {
        BLOCK_COLORS.put('A', "\u001B[31m");
        BLOCK_COLORS.put('B', "\u001B[34m");
        BLOCK_COLORS.put('C', "\u001B[32m");
        BLOCK_COLORS.put('D', "\u001B[35m");
        BLOCK_COLORS.put('E', "\u001B[33m");
        BLOCK_COLORS.put('F', "\u001B[36m");
        BLOCK_COLORS.put('G', "\u001B[35m");
        BLOCK_COLORS.put('H', "\u001B[94m");
        BLOCK_COLORS.put('I', "\u001B[92m");
        BLOCK_COLORS.put('J', "\u001B[93m");
        BLOCK_COLORS.put('K', "\u001B[91m");
        BLOCK_COLORS.put('L', "\u001B[96m");
        BLOCK_COLORS.put('M', "\u001B[90m");
        BLOCK_COLORS.put('N', "\u001B[97m");
        BLOCK_COLORS.put('O', "\u001B[37m");
        BLOCK_COLORS.put('P', "\u001B[30m");
        BLOCK_COLORS.put('Q', "\u001B[38;5;208 ");
        BLOCK_COLORS.put('R', "\u001B[38;5;202 ");
        BLOCK_COLORS.put('S', "\u001B[38;5;129 ");
        BLOCK_COLORS.put('T', "\u001B[38;5;93m");
        BLOCK_COLORS.put('U', "\u001B[38;5;27m");
        BLOCK_COLORS.put('V', "\u001B[38;5;28m");
        BLOCK_COLORS.put('W', "\u001B[38;5;226 ");
        BLOCK_COLORS.put('X', "\u001B[38;5;87m");
        BLOCK_COLORS.put('Y', "\u001B[38;5;141 ");
        BLOCK_COLORS.put('Z', "\u001B[38;5;203 ");
        }

        IMAGE_COLORS.put('A', new Color(255, 0, 0));
        IMAGE_COLORS.put('B', new Color(0, 0, 255));
        IMAGE_COLORS.put('C', new Color(0, 255, 0));
        IMAGE_COLORS.put('D', new Color(128, 0, 128));
        IMAGE_COLORS.put('E', new Color(255, 255, 0));
        IMAGE_COLORS.put('F', new Color(0, 255, 255));
        IMAGE_COLORS.put('G', new Color(255, 192, 203));
        IMAGE_COLORS.put('H', new Color(173, 216, 230));
        IMAGE_COLORS.put('I', new Color(144, 238, 144));
        IMAGE_COLORS.put('J', new Color(255, 228, 181));
        IMAGE_COLORS.put('K', new Color(220, 20, 60));
        IMAGE_COLORS.put('L', new Color(70, 130, 180));
        IMAGE_COLORS.put('M', new Color(47, 79, 79));
        IMAGE_COLORS.put('N', new Color(245, 245, 245));
        IMAGE_COLORS.put('O', new Color(169, 169, 169));
        IMAGE_COLORS.put('P', new Color(0, 0, 0));
        IMAGE_COLORS.put('Q', new Color(255, 140, 0));
        IMAGE_COLORS.put('R', new Color(255, 69, 0));
        IMAGE_COLORS.put('S', new Color(255, 105, 180));
        IMAGE_COLORS.put('T', new Color(148, 0, 211));
        IMAGE_COLORS.put('U', new Color(25, 25, 112));
        IMAGE_COLORS.put('V', new Color(34, 139, 34));
        IMAGE_COLORS.put('W', new Color(255, 255, 102));
        IMAGE_COLORS.put('X', new Color(0, 206, 209));
        IMAGE_COLORS.put('Y', new Color(216, 191, 216));
        IMAGE_COLORS.put('Z', new Color(250, 128, 114));
    }

    public static void displayColoredBoard (Board board) {
        for (int i = 0; i < board.getM(); i++) {
            for (int j = 0; j < board.getN(); j++) {
                char cell = board.getCell(i, j);
                if (cell != '.') {
                    System.out.print(BLOCK_COLORS.getOrDefault(cell, RESET) + cell + " " + RESET);
                } else {
                    System.out.print(cell + " ");
                }
            }
            System.out.println();
        }
    }
}

```

```

public static void saveBoardAsImage(Board board, String filePath, long time, long cases) throws
IOException {
    int width = board.getM() * CELL_SIZE;
    int height = board.getN() * CELL_SIZE + 100;

    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = image.createGraphics();

    g2d.setColor(Color.WHITE);
    g2d.fillRect(0, 0, width, height);

    for (int i = 0; i < board.getN(); i++) {
        for (int j = 0; j < board.getM(); j++) {
            char cell = board.getCell(i, j);
            if (cell != '.') {
                g2d.setColor(IMAGE_COLORS.getOrDefault(cell, Color.GRAY));
                g2d.fillRect(j * CELL_SIZE, i * CELL_SIZE, CELL_SIZE, CELL_SIZE);
            }

            g2d.setColor(Color.BLACK);
            g2d.drawRect(j * CELL_SIZE, i * CELL_SIZE, CELL_SIZE, CELL_SIZE);
        }
    }

    g2d.setFont(new Font("Arial", Font.BOLD, CELL_SIZE/2));
    for (int i = 0; i < board.getN(); i++) {
        for (int j = 0; j < board.getM(); j++) {
            char cell = board.getCell(i, j);
            if (cell != '.') {
                g2d.setColor(Color.BLACK);
                String label = String.valueOf(cell);
                FontMetrics metrics = g2d.getFontMetrics();
                int x = j * CELL_SIZE + (CELL_SIZE - metrics.stringWidth(label)) / 2;
                int y = i * CELL_SIZE + ((CELL_SIZE + metrics.getHeight()) / 2) - metrics.getDescent();
                g2d.drawString(label, x, y);
            }
        }
    }

    g2d.setFont(new Font("Arial", Font.BOLD, 24));
    g2d.setColor(Color.BLACK);
    String timeLabel = "Time: " + time + " ms";
    String iterationsLabel = "Iterations: " + cases;
    g2d.drawString(timeLabel, 10, board.getN() * CELL_SIZE + 30);
    g2d.drawString(iterationsLabel, 10, board.getN() * CELL_SIZE + 60);

    g2d.dispose();

    File outputFile = new File(filePath);
    ImageIO.write(image, "png", outputFile);
    System.out.println("Image saved to " + filePath);
}

```

```

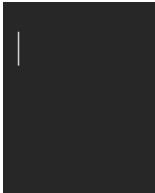
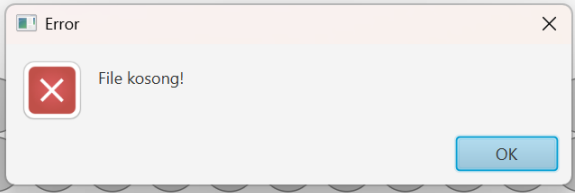
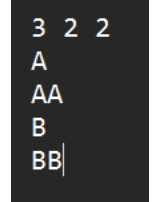
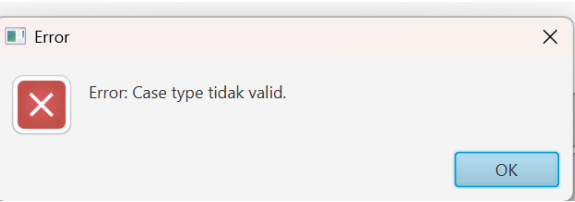
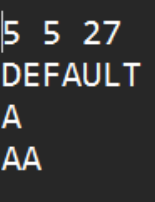
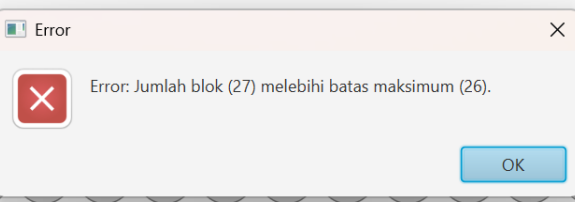
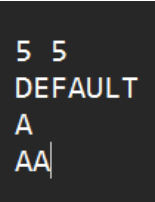
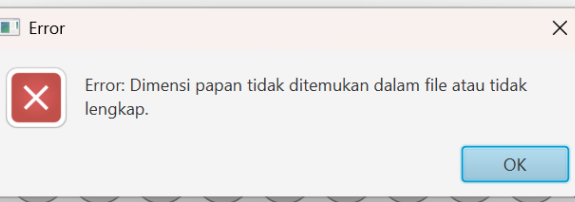

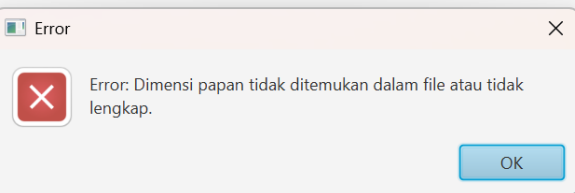
1 public static void saveBoardAsText(Board board, String filePath, long time, long cases)
  throws IOException {
2     try (FileWriter writer = new FileWriter(filePath)) {
3         for (int i = 0; i < board.getN(); i++) {
4             for (int j = 0; j < board.getM(); j++) {
5                 char cell = board.getCell(i, j);
6                 writer.write(cell + " ");
7             }
8             writer.write("\n");
9         }
10        writer.write("Time execution: " + time + " ms\n");
11        writer.write("Iterations: " + cases + "\n\n");
12    }
13    System.out.println("Text file saved to " + filePath);
14 }
15 }

```

## D. Testing



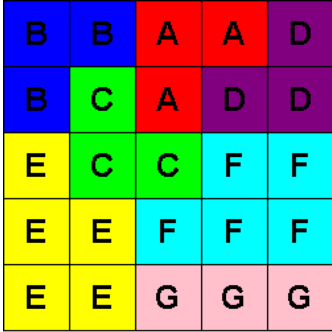
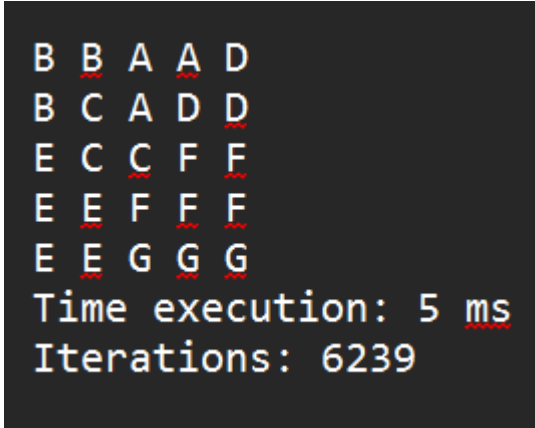
Testcase Input validasi:

Nama Testcase	Input	Output
Testcase-1 (File kosong)		
Testcase-2 (File tidak memiliki CaseType)		
Testcase-3 (P >26):		
Testcase-4 (P Tidak ada)		
Testcase-5		

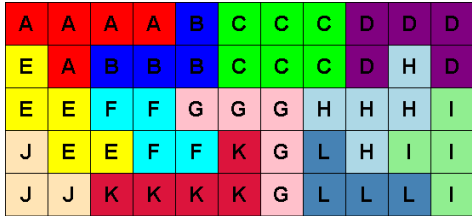
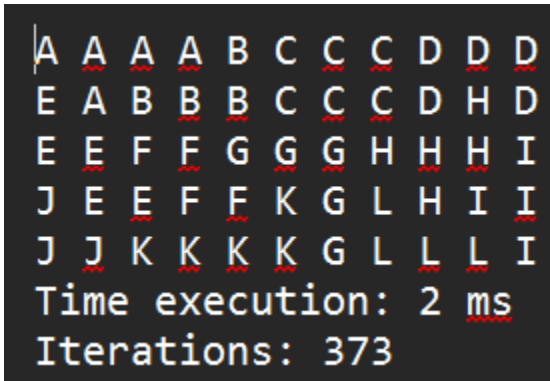
Testcase Tambahan:

Testcase-6:

Input	Hasil Gambar	Hasil txt
-------	--------------	-----------

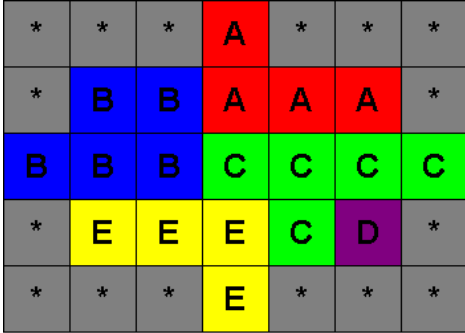
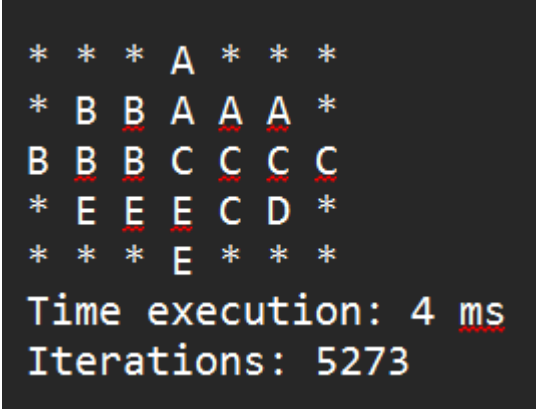
5 5 7 DEFAULT BB B AA A C CC D DD EE EE E FF FF F GGG	 <p>Time: 2 ms Iterations: 6239</p>	
---	--	--

Testcase-7 (Setidaknya ada 1 karakter yang rata kiri) :

Input	Hasil Gambar	Hasil txt
5 11 12 DEFAULT AAAA A B BBB CCC CCC DDD D D E EE EE FF FF GGG G G H HHH H I II	 <p>Time: 1 ms Iterations: 373</p>	

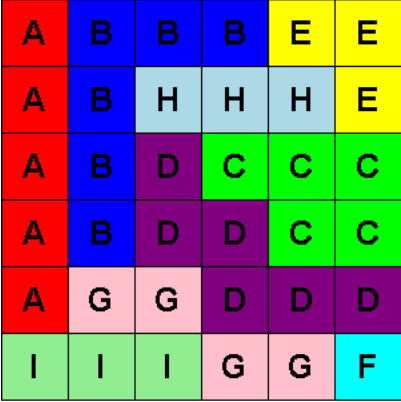
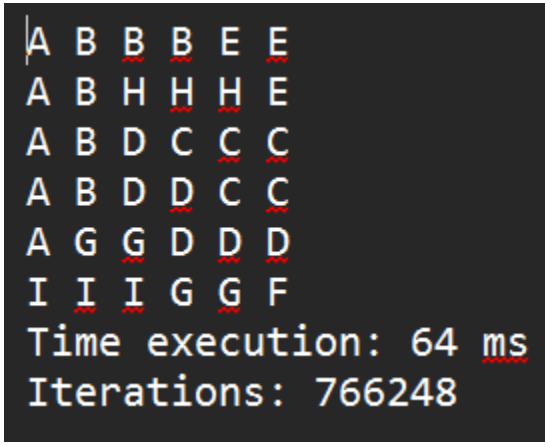
I J JJ KKKK K L LLL		
---------------------------------------	--	--

Testcase ke-8 (Konfigurasi Custom):

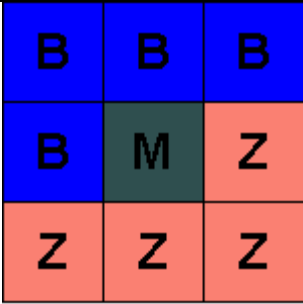
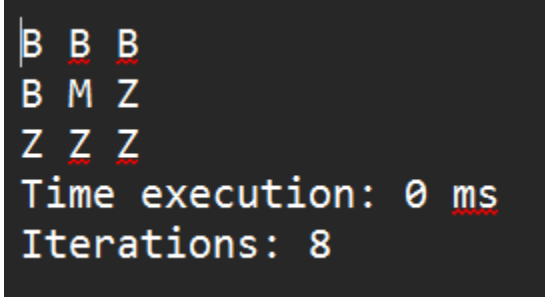
Input	Hasil Gambar	Hasil txt
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	 <p>Time: 4 ms Iterations: 5273</p>	

Testcase-9:

Input	Hasil Gambar	Hasil txt
-------	--------------	-----------

6 6 9 DEFAULT A A A A A BBB B B B CCC CC D D DD DD E EE F GG GG HHH III	 <p>Time: 64 ms Iterations: 766248</p>	
--	---	--

Testcase-10 (Tidak Urut):

Input	Hasil Gambar	Hasil txt
3 3 3 DEFAULT BBB B M Z ZZZ	 <p>Time: 0 ms Iterations: 8</p>	

## **E. Extra**

### **Analisis Trade-off Pendekatan Brute Force**

#### **1. Kelebihan**

##### **1. Garansi Solusi Optimal**

- Algoritma menjamin menemukan solusi jika ada
- Tidak ada risiko melewatkan kemungkinan solusi
- Cocok untuk puzzle yang membutuhkan solusi eksak

##### **2. Memory Efficient**

- Hanya membutuhkan space untuk menyimpan state board saat ini
- Tidak perlu menyimpan history states
- Space complexity  $O(N \times M)$  dimana  $N, M$  adalah dimensi board

#### **2. Kekurangan**

##### **1. Performa Waktu**

- Kompleksitas waktu eksponensial  $O(8^P \times N \times M)$
- Tidak efisien untuk puzzle besar
- Runtime meningkat drastis dengan penambahan pieces

##### **2. Scalability Issues**

- Sulit menangani board ukuran besar
- Tidak praktis untuk puzzle dengan banyak pieces
- Runtime bisa tidak predictable

**Alamat Github** : [https://github.com/Farhanabd05/Tucil1\\_13523042](https://github.com/Farhanabd05/Tucil1_13523042)

**Spek Laptop (soalnya beda spek laptop, beda runtime)** :

- Windows 11
- RAM 12 GB
- Intel Core i5
- SSD 512 GB

**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 custom	✓	
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	