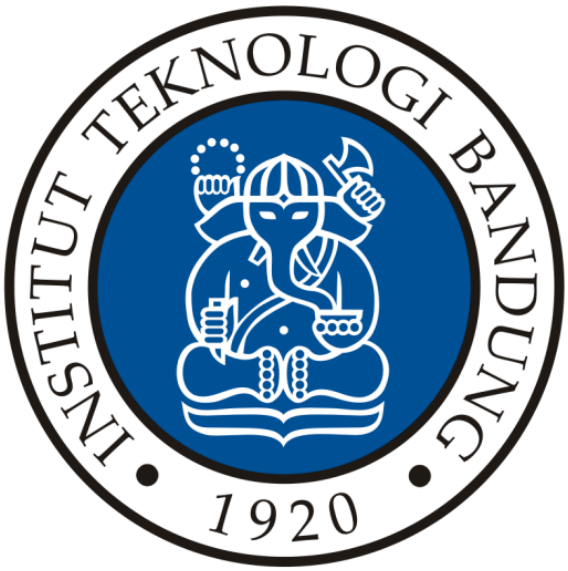
LAPORAN TUGAS KECIL
MATA KULIAH IF2211 STRATEGI ALGORITMA
PROGRAM IQ PUZZLER PRO SOLVER



Disusun oleh :

Rafa Abdussalam Danadyaksa 13523133

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I
DESKRIPSI MASALAH

IQ Puzzler Pro adalah permainan puzzle yang pemain untuk mengisi ruang kosong pada papan permainan dengan blok dengan bentuk yang unik. Permainan ini terdiri dari beberapa komponen yaitu papan dan beberapa blok dengan bentuk yang berbeda-beda. Pemain dapat memutar dan membolak-balik blok sebelum menaruhnya di papan permainan untuk menemukan kombinasi yang sesuai. Permainan dianggap selesai jika seluruh area pada papan permainan sudah terisi dengan blok dan semua blok yang disediakan sudah diletakkan di dalam area papan permainan.

Dalam tugas ini, program akan menggunakan bahasa pemrograman Java dan menggunakan algoritma brute force untuk menyelesaikan permainan IQ Puzzler Pro. Program akan menerima input berupa dimensi papan, banyak block puzzle, jenis kasus (Default, Custom, dan Pyramid), dan bentuk blok puzzle. Setelah solusi ditemukan, program harus dapat menampilkan konfigurasi akhir pada papan, waktu eksekusi program brute force, dan jumlah iterasi yang dilakukan algoritma brute force. Pengguna dapat memilih opsi untuk menyimpan solusi dalam bentuk file txt.

Gambar 1.1 Permainan IQ Puzzler Pro

BAB II
TEORI SINGKAT

2.1 Algoritma Brute Force

Algoritma Brute Force adalah algoritma dengan bersifat lurus (straightforward) dalam memecahkan suatu persoalan. Algoritma ini menggunakan strategi mengevaluasi semua kemungkinan satu per satu, karena hal itu dibutuhkan waktu eksekusi program yang lama dan kurang efisien. Kelebihan dari algoritma ini seperti, sederhana dan mudah dimengerti, dapat diterapkan dalam memecahkan masalah, dan memberikan hasil layak untuk beberapa persoalan.

Dalam program ini Algoritma brute force digunakan dengan mencoba semua kemungkinan penempatan blok-blok pada papan permainan. Setiap blok dilakukan percobaan penempatan pada papan dengan setiap orientasi yang mungkin menggunakan rotasi dan pencerminan sampai ditemukan konfigurasi yang cocok untuk menyelesaikan permainan puzzle.

BAB III
IMPLEMENTASI PROGRAM

3.1 Alur Program

Alur program diawali dari pembacaan file input yang berisi ukuran papan serta blok- blok puzzle. Input awal berbentuk tiga angka yang memastikan dimensi baris, kolom, serta jumlah blok, diikuti dengan jenis permasalahan (DEFAULT, CUSTOM, dan PYRAMID). Setelah itu, program membaca tiap blok puzzle. Blok- blok ini dibaca baris per baris, dengan tiap blok diidentifikasi apakah huruf sama, jika huruf sama maka akan ditambahkan kedalam blok yang sama. Setiap block disimpan di dalam List<String>.

Selanjutnya, program menginisialisasi papan permainan puzzle kosong dengan dimensi baris serta kolom hasil dari input file. Setelah itu, Solver mencoba menempatkan blok satu per satu. untuk setiap blok, program mencoba penempatan seluruh orientasi yang dilakukan untuk setiap posisi pada papan. Apabila suatu blok dapat ditempatkan, program akan mencoba menempatkan blok selanjutnya secara rekursif. Dalam proses penempatan, setiap saat percobaan penempatan blok, program melakukan pengecekan apakah penempatan valid atau tidak menggunakan **canbePlace**() (tidak tumpang tindih dengan blok lain dan tidak keluar batasan papan). Bila ditemui pemecahan lengkap (seluruh blok sukses ditempatkan). Program akan mengeluarkan output berupa hasil akhir papan, waktu eksekusi, dan jumlah iterasi yang dicoba. Bila sesuatu penempatan tidak menuju ke pemecahan, program hendak membatalkan penempatan tersebut (backtracking) serta berupaya mungkin lain. Setelah program menampilkan solusi, program dapat menawarkan opsi untuk melakukan penyimpanan solusi ke dalam file.

3.2 Program Utama

3.2.1 Main.java

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print(s:"Masukkan Nama File (Namafile): ");
        String filename = scanner.nextLine();
        String filePath = "../test/" + filename + ".txt";

        try {
            File file = new File(filePath);
            Scanner fileScanner = new Scanner(file);

            String[] dimensions = fileScanner.nextLine().split(regex:" ");
            int rows = Integer.parseInt(dimensions[0]);
            int cols = Integer.parseInt(dimensions[1]);
            int jumlahblock = Integer.parseInt(dimensions[2]);

            String Jeniskasus = fileScanner.nextLine();

            if (!Jeniskasus.equals(anObject:"DEFAULT")){
                System.out.println(x:"Jenis kasus yang Valid: DEFAULT");
                System.exit(status:0);
            }
        //    // Debug info
        //    System.out.println("\n=== Debug Info ===");
        //    System.out.println("Board dimensions: " + rows + "x" + cols);
        //    System.out.println("Number of blocks: " + jumlahblock);
        //    System.out.println("Case type: " + Jeniskasus);

            List<List<String>> Blocks = new ArrayList<>();
            List<String> currentBlock = new ArrayList<>();
            String currentLetter = "";

            int countblock = 0;

            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine().trim();

                if (!line.isEmpty()) {
                    String firstLetter = String.valueOf(line.charAt(index:0));

                    if (!firstLetter.equals(currentLetter)) {
                        if (!currentBlock.isEmpty()) {
                            Blocks.add(new ArrayList<>(currentBlock));
                            currentBlock.clear();
                            countblock++;
                        }
                        currentLetter = firstLetter;
                    }

                    currentBlock.add(line);
                }
            }
```

```
            countblock++;
            if (!currentBlock.isEmpty()) {
                Blocks.add(new ArrayList<>(currentBlock));
            }

            fileScanner.close();

            if(countblock != jumlahblock){
                System.out.println(x:"Jumlah blok yang dimasukkan tidak sesuai");
                System.exit(status:0);
            }

        //     // Debug: Print semua blok yang dibaca
        //     System.out.println("\n=== Blocks Read ===");
        //     for (int i = 0; i < Blocks.size(); i++) {
        //         System.out.println("\nBlock " + (i + 1) + ":");
        //         for (String line : Blocks.get(i)) {
        //             System.out.println(line);
        //         }
        //     }

            List<Block> blocks = new ArrayList<>();
            for (List<String> blockLines : Blocks) {
                if (!blockLines.isEmpty()) {
                    char id = blockLines.get(index:0).charAt(index:0);
                    blocks.add(new Block(blockLines, id));
                }
            }

            Board board = new Board(rows, cols);
            Solver solver = new Solver(board, blocks);

            System.out.println(x:"\n=== Start Solver ===");
            long startTime = System.nanoTime();
            boolean solved = solver.solve();
            long endTime = System.nanoTime();
            long TotalTime = (endTime - startTime) / 1000000;

            if (solved && board.isFull()) {
                System.out.println(x:"\nSolusi ditemukan:");
                board.printBoard();
                System.out.println("\nWaktu pencarian: " + TotalTime + " ms");
                System.out.println("Banyak kasus yang ditinjau: " + solver.getIterationCount());

                System.out.print(s:"\nApakah anda ingin menyimpan solusi? (y/n): ");
                String savefile = scanner.nextLine();

                if (savefile.equalsIgnoreCase(anotherString:"y")) {
                    System.out.print(s:"Masukkan nama file (Hasil.txt): ");
                    String FileName = scanner.nextLine();
                    saveSolution(board, solver.getIterationCount(), TotalTime, FileName);
                    System.out.println("Solusi berhasil disimpan ke " + FileName);
                }
            } else {
                System.out.println(x:"\nTidak ditemukan solusi.");
                System.out.println("Waktu pencarian: " + TotalTime + " ms");
                System.out.println("Banyak kasus yang ditinjau: " + solver.getIterationCount());
            }
```

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

        scanner.close();
    }

    private static void saveSolution(Board board, long iterations, long TotalTime, String fileName) {
        try {
            FileWriter writer = new FileWriter("../test/" + fileName + ".txt" );
            writer.write(board.toString() + "\n");
            writer.write("Waktu pencarian: " + TotalTime + " ms\n");
            writer.write("Banyak kasus yang ditinjau: " + iterations + "\n");
            writer.close();
        } catch (IOException e) {
            System.out.println("Gagal menyimpan solusi: " + e.getMessage());
        }
    }
}
```

3.2.2 Block.java

```
public class Block {
    private char[][] shape;
    private char id;
    private List<char[][]> allOrientations;

    public Block(List<String> blockLines, char id) {
        this.id = id;
        this.shape = parser(blockLines);
        this.allOrientations = Orientations();
    }

    private char[][] parser(List<String> blockLines) {
        int maxRows = blockLines.size();
        int maxCols = 0;
        for (String line : blockLines) {
            maxCols = Math.max(maxCols, line.length());
        }
```

```
        char[][] blockShape = new char[maxRows][maxCols];
        for (int i = 0; i < maxRows; i++) {
            for (int j = 0; j < maxCols; j++) {
                blockShape[i][j] = '.';
            }
        }

        for (int i = 0; i < blockLines.size(); i++) {
            String line = blockLines.get(i);
            for (int j = 0; j < line.length(); j++) {
                blockShape[i][j] = line.charAt(j);
            }
        }
        return blockShape;
    }

    private List<char[][]> Orientations() {
        List<char[][]> orientations = new ArrayList<>();

        orientations.add(shape);

        for (int rotation = 1; rotation <= 4; rotation++) {
            char[][] rotated = rotateBlock(shape, rotation);
            orientations.add(rotated);
        }

        for (int mirror = 1; mirror <= 2; mirror++) {
            char[][] mirrored = mirror(shape);
            orientations.add(mirrored);

            for (int rotation = 1; rotation <= 4; rotation++) {
                char[][] rotatedMirrored = rotateBlock(mirrored, rotation
                orientations.add(rotatedMirrored);
            }
        }

        return orientations;
    }
```

```java
private char[][] rotateBlock(char[][] block, int times) {
    for (int t = 0; t < times; t++) {
        int rows = block.length;
        int cols = block[0].length;
        char[][] rotated = new char[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                rotated[j][rows-1-i] = block[i][j];
            }
        }
        block = rotated;
    }

    return block;
}

private char[][] mirror(char[][] block) {
    int rows = block.length;
    int cols = block[0].length;
    char[][] mirrored = new char[rows][cols];

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            mirrored[i][cols-1-j] = block[i][j];
        }
    }
    return mirrored;
}
```

```java
public char[][] getShape() {
    return shape;
}

public List<char[][]> getAllOrientations() {
    return allOrientations;
}

public char getId() {
    return id;
}
```

3.2.3 Solver.java

```java
public class Solver {
    private Board board;
    private List<Block> blocks;
    private long iterationCount;

    public Solver(Board board, List<Block> blocks) {
        this.board = board;
        this.blocks = blocks;
        this.iterationCount = 0;
    }

    public boolean solve() {
        return solvepuzzle(blockIndex:0);
    }

    private boolean solvepuzzle(int blockIndex) {
        if (blockIndex >= blocks.size()) {
            return true;
        }

        Block block = blocks.get(blockIndex);
        List<char[][]> orientations = block.getAllOrientations();

        List<Block> allBlock = new ArrayList<>();
        for(char[][] orientation : orientations) {
            List<String> orientationStrings = allblocktolist(orientation);
            allBlock.add(new Block(orientationStrings, block.getId()));
        }
```

```
                for (Block orientedBlocks : allBlock) {
                    iterationCount++;

                    if (board.canbePlace(orientedBlocks, row, col)) {
                        board.placeBlock(orientedBlocks, row, col);

                        if (solvepuzzle(blockIndex + 1)) {
                            return true;
                        }
                        board.removeBlock(orientedBlocks, row, col);
                    }
                }
            }
        }

        return false;
    }

    private List<String> allblocktolist (char[][] orientation) {
        List<String> result = new ArrayList<>();
        for (char[] row : orientation) {
            result.add(new String(row));
        }
        return result;
    }

    public long getIterationCount() {
        return iterationCount;
```

3.2.4 Board.java

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

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            grid[i][j] = '.';
        }
    }
}

public boolean canbePlace(Block block, int startRow, int startCol) {
    char[][] shape = block.getShape();

    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[0].length; j++) {
            if (shape[i][j] == block.getId()) {
                int newRow = startRow + i;
                int newCol = startCol + j;
                if (!isInBounds(newRow, newCol) || !isCellEmpty(newRow, newCol)) {
                    return false;
                }
            }
        }
    }
    return true;
}

public void placeBlock(Block block, int startRow, int startCol) {
    char[][] shape = block.getShape();

    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[0].length; j++) {
            if (shape[i][j] == block.getId()) {
                grid[startRow + i][startCol + j] = block.getId();
            }
        }
    }
}

public void removeBlock(Block block, int startRow, int startCol) {
    char[][] shape = block.getShape();

    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[0].length; j++) {
            if (shape[i][j] == block.getId()) {
                grid[startRow + i][startCol + j] = '.';
            }
        }
    }
}
```

```java
public boolean isFull() {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == '.') {
                return false;
            }
        }
    }
    return true;
}
```

BAB IV
EKSPERIMEN

4.1 test1.txt

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

```
Block 5:
EE
EE
E

Block 6:
FF
FF
F

Block 7:
GGG

=== Start Solver ===

Solusi ditemukan:
ABBCC
AABCD
EEEDD
EEFFF
GGGFF

Waktu pencarian: 505 ms
Banyak kasus yang ditinjau: 2070532
```

4.2 test2.txt

```
3 3 3
CUSTOM
AAA
A
A
BB
B
C
```

```
PS C:\Users\rafaa\ITB\Tucil stima\Tucil1_13523133\src> java Main
Masukkan Nama File (Namafile): test2
Jenis kasus yang Valid: DEFAULT
```

4.3 test3.txt

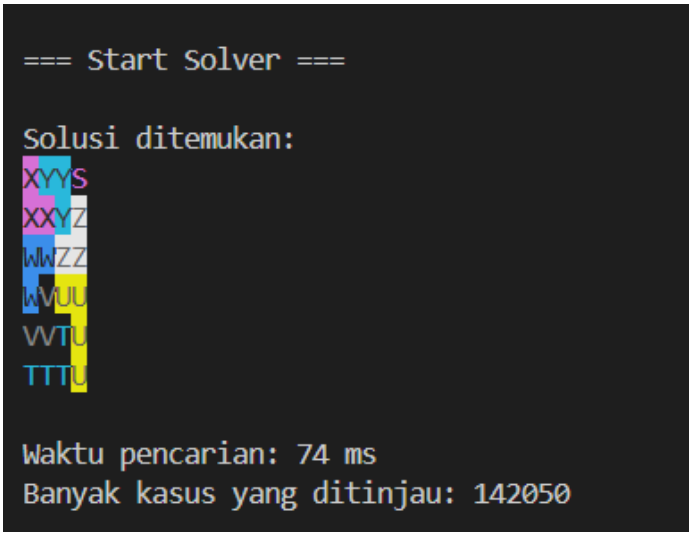
```
4 5 5
DEFAULT
A
AA
B
```

```
BB
C
CC
D
DD
```

```
=== Debug Info ===
Board dimensions: 4x5
Number of blocks: 5
Case type: DEFAULT
Jumlah blok yang dimasukkan tidak sesuai
```

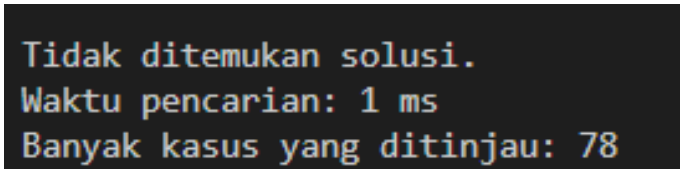
4.4 test4.txt

```
6 4 8
DEFAULT
X
XX
Y
YY
Z
ZZ
W
WW
V
VV
U
UUU
T
TTT
S
```



4.5 test5.txt

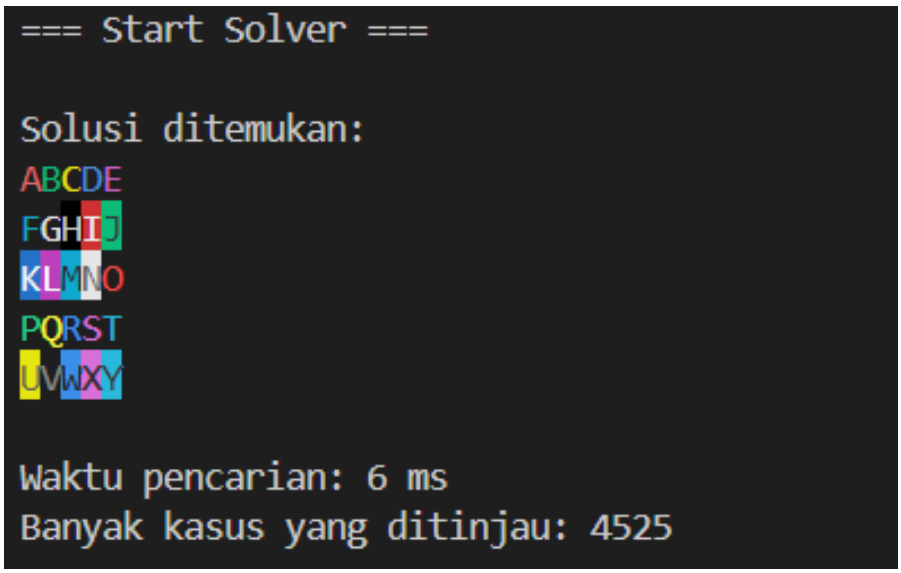
```
3 3 2
DEFAULT
ZZZ
ZZ
Z
YY
```



4.6 test6.txt

```
5 5 25
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
```



4.7 test7.txt

```
3 3 3
DEFAULT
AA
A
AA
B
```

```
BB
B
CC
```

```
Masukkan Nama File (Namafile): test7

=== Start Solver ===

Tidak ditemukan solusi.
Waktu pencarian: 20 ms
Banyak kasus yang ditinjau: 12015
```

4.8 test8.txt

```
5 3 5
DEFAULT
B
CC
C
CC
D
DD
D
E
E
F
FF
```

```
=== Start Solver ===

Solusi ditemukan:
BCC
EEC
DCC
DDF
DFF

Waktu pencarian: 44 ms
Banyak kasus yang ditinjau: 34511
```

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil yang diperoleh dapat disimpulkan bahwa Permainan IQ Puzzler pro dapat diselesaikan dengan memanfaatkan algoritma brute force dengan mengevaluasi setiap kemungkinan satu per satu, namun algoritma ini tidak efisien dan memakan banyak memori.

5.2 Link Repository

Pranala Repository
https://github.com/RafaAbdussalam/Tucil1_13523133

Lampiran

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf

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 .tx	✔	
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	✔	