TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

Benedictus Nelson
13523150

Dosen Pengampu :
Dr. Nur Ula Maulidevi, S.T, M.Sc.
Dr. Ir. Rinaldi Munir, M.T.
Monterico Adrrian, S.T, M.T.

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

DAFTAR ISI

BAB 1 ALGORITMA

3.1 Main Idea

1. **Menyiapkan Papan Kosong (N×M)**
    - Papan diinisialisasi dengan karakter '.' yang menandakan sel kosong.
2. **Membaca Puzzle Piece**
    - Piece dibaca dari file input, yang berisi label (alfabet huruf kapital) dan beberapa baris yang mendefinisikan bentuk piece (angka '1' sel terisi, atau dihasilkan parsing label).
    - Total puzzle piece = P, masing-masing piece punya label (A, B, dst).
3. **Backtracking**
    - Piece ke-i di dalam papan:
        - Mengambil bentuk piece ke-i (transformasi rotasi 0°, 90°, 180°, 270°, dan mirror).
        - Mencoba setiap posisi (r, c) di dalam papan. Jika piece dapat ditempatkan dilanjutkan menempatkan piece ke-(i+1).
        - Jika menempatkan piece ke-(i+1) berakhir, papan kembali seperti semula (remove piece) dan lanjut mencoba posisi lain.
    - Jika berhasil menempatkan seluruh piece (i == P), maka papan diperiksa apakah seluruhnya penuh. Jika penuh, solusi ditemukan.
4. **Pengecekan "Board is Full"**
    - Setelah seluruh piece ditempatkan, cek apakah masih ada sel '.' dan jika tidak ada yang kosong maka puzzle berhasil disusun.

3.2 Langkah Terinci

1. **Parse Input**
    - Baca N, M, P, dan mode (hanya tersedia DEFAULT).
    - Baca puzzle piece.
    - Memastikan total sel puzzle sama dengan N×M.
2. **Siapkan Data Struktural**
    - board = matriks char ukuran N×M, inisialisasi '.'.
    - used = array boolean ukuran P, menandakan apakah piece tertentu sudah ditempatkan.
3. **Fungsi solve()**
    - Memanggil backtrack(0), mencoba menempatkan piece ke-0, lalu ke-1, dst.
4. **Fungsi backtrack(i)**

- Jika i == P, maka cek apa papan penuh. Jika penuh, return true (solusi ditemukan). Jika tidak penuh, return false.
- Ambil puzzle piece ke-i. Dapatkan semua transformasinya (rotasi + mirror).
- Untuk setiap transformasi, dansetiap (r, c) di papan,
  - Jika dapat menempatkan piece di posisi (r, c) tanpa bentrok, piece diletakkan.
  - Rekursi: if (backtrack(i+1)) return true;
  - Jika rekursi bernilai false, remove piece dan lanjutkan.
- Jika semua upaya gagal, return false.

5. **Output**
   - Jika solve() return true, maka cetak papan. Bila false, cetak "No solution found!".
   - Menamampilkan juga waktu eksekusi (selisih sistem waktu sebelum/selesai backtracking) dan jumlah kasus ditinjau (counter increment tiap kali mencoba menempatkan piece di posisi tertentu).

Dengan pendekatan brute force murni ini, **tidak** menggunakan heuristik khusus untuk mempersingkat pencarian. Maaka waktu eksekusi semakin lama jika puzzle dan papan besar, karena banyaknya kemungkinan penempatan.

BAB 2 PROGRAM

Main.java

```
import javax.swing.JFileChooser;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
            JFileChooser fileChooser = new JFileChooser();
            int returnVal = fileChooser.showOpenDialog(null);
            if (returnVal != JFileChooser.APPROVE_OPTION) {
                System.out.println("Tidak ada file yang dipilih");
                return;
            }

            File chosenFile = fileChooser.getSelectedFile();

            try (Scanner sc = new Scanner(chosenFile)) {
                int N = sc.nextInt();
                int M = sc.nextInt();
                int P = sc.nextInt();
                sc.nextLine();

                String mode = sc.nextLine().trim();

                if (!mode.equalsIgnoreCase("DEFAULT")) {
                    System.out.println("Mode yang didukung hanya DEFAULT.");
                    return;
                }

                List<String> remainingLines = new ArrayList<>();
                while (sc.hasNextLine()) {
                    String line = sc.nextLine();
                    if (!line.trim().isEmpty()) {
                        remainingLines.add(line.trim());
                    }
                }

                if (remainingLines.isEmpty()) {
                    throw new IllegalArgumentException("Tidak ada data puzzle piece setelah mode");
                }

                PuzzlePiece[] pieces = parseDefaultPieces(remainingLines, P);
                if (pieces.length < P) {
                    throw new IllegalArgumentException("Kurang data untuk puzzle piece" +
pieces.length + ".");
                }

                PuzzleSolver solver = new PuzzleSolver(N, M, pieces);
```

```
            long startTime = System.currentTimeMillis();
            boolean solved = solver.solve();
            long endTime = System.currentTimeMillis();
            if (solved) {
                System.out.println("Solusi:");
                solver.printBoard();
            } else {
                System.out.println("Tidak ada solusi.");
            }
            System.out.println("Waktu pencarian: " + (endTime - startTime) + " ms");
            System.out.println("Banyak kasus yang ditinjau: " + solver.getAttemptCount());

            try (Scanner in = new Scanner(System.in)) {
                System.out.println("Apakah anda ingin menyimpan solusi? (ya/tidak)");
                String answer = in.nextLine().trim();
                if (answer.equalsIgnoreCase("ya")) {
                    System.out.println("Solusi disimpan.");
                } else {
                    System.out.println("Solusi tidak disimpan.");
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("File tidak ditemukan: " + chosenFile.getAbsolutePath());
        }
    }

    private static PuzzlePiece[] parseDefaultPieces(List<String> lines, int P) {
        PuzzlePiece[] pieces = new PuzzlePiece[P];
        int count = 0;
        int index = 0;
        while (count < P && index < lines.size()) {
            String firstLine = lines.get(index);
            char label = firstLine.charAt(0);
            List<String> pieceLines = new ArrayList<>();
            pieceLines.add(firstLine);
            index++;

            while (index < lines.size() && lines.get(index).charAt(0) == label) {
                pieceLines.add(lines.get(index));
                index++;
            }

            int maxLen = 0;
            for (String s : pieceLines) {
                if (s.length() > maxLen) {
                    maxLen = s.length();
                }
            }
            int rows = pieceLines.size();
            int[][] shape = new int[rows][maxLen];
            for (int i = 0; i < rows; i++) {
                String s = pieceLines.get(i);
                while (s.length() < maxLen) {
                    s += " ";
                }
```

```
                for (int j = 0; j < maxLen; j++) {
                    shape[i][j] = (s.charAt(j) == label) ? 1 : 0;
                }
            }
            PuzzlePiece piece = new PuzzlePiece(label);
            piece.addShape(shape);
            pieces[count++] = piece;
        }
        return pieces;
    }
}
```

PuzzlePiece.java

```
import java.util.ArrayList;
import java.util.List;

public class PuzzlePiece {
    private char label;
    private List<int[][]> shapes;

    public PuzzlePiece(char label) {
        this.label = label;
        this.shapes = new ArrayList<>();
    }

    public char getLabel() {
        return label;
    }

    public List<int[][]> getShapes() {
         return shapes;
        }

     public void addShape(int[][] shape) {
         this.shapes.add(shape);
     }
```

```
}
```

PuzzleSolver.java

```
import java.util.List;
import java.util.ArrayList;

/*
    Konstruktor:
      I.S.: Parameter rows, cols, dan array pieces
      F.S.: Board puzzle diinisialisasi sebagai matriks rows x cols yang
terisi karakter '.'

    Main
      - solve: I.S.: Board kosong, pieces belum ditempatkan
               F.S.: (papan terisi penuh)
      - printBoard: I.S.: solusi
                    F.S.: board dicetak ke konsol

      - canPlace: cek apa matrix shape bisa ditempatkan di board di
posisi tertentu
      - placePiece: naro shape di board
      - removePiece: hapus shape dri board
*/
public class PuzzleSolver {
    private int rows, cols;
    private char[][] board;
    private PuzzlePiece[] pieces;
    private boolean[] used;
    private long attemptCount;

    // ANSI color codes
    private static final String ANSI_RESET  = "\u001B[0m";
    private static final String ANSI_RED    = "\u001B[31m";
```

```
    private static final String ANSI_GREEN  = "\u001B[32m";
    private static final String ANSI_YELLOW = "\u001B[33m";
    private static final String ANSI_BLUE   = "\u001B[34m";
    private static final String ANSI_PURPLE = "\u001B[35m";
    private static final String ANSI_CYAN   = "\u001B[36m";
    private static final String ANSI_WHITE  = "\u001B[37m";
    /**
     * Konstruktor PuzzleSolver.
     * I.S.: Parameter rows, cols, dan array pieces terdefinisi
     * F.S.: Board puzzle diinisialisasi ukuran rows x cols, terisi
dengan '.'
     */
    public PuzzleSolver(int rows, int cols, PuzzlePiece[] pieces) {
        this.rows = rows;
        this.cols = cols;
        this.pieces = pieces;
        this.board = new char[rows][cols];
        this.used = new boolean[pieces.length];
        this.attemptCount = 0;
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                board[r][c] = '.';
            }
        }
    }

    /**
     * brute force
     * I.S.: board kosong & semua pieces ada
     * F.S.: true jika solusi ditemukan (board terisi penuh), false jika
tidak
     */
    public boolean solve() {
        return backtrack(0);
    }

    // backtracking
    private boolean backtrack(int pieceIndex) {
        if (pieceIndex == pieces.length)return boardIsFull();
```

```
        PuzzlePiece piece = pieces[pieceIndex];
        for (int[][] shape :
getAllTransformations(piece.getShapes().get(0))) {
            for (int r = 0; r < rows; r++) {
                for (int c = 0; c < cols; c++) {
                attemptCount++; //hitung jumlah percobaan
                    if (canPlace(shape, r, c)) {
                        placePiece(shape, r, c, piece.getLabel());
                        used[pieceIndex] = true;
                        if (backtrack(pieceIndex + 1)) return true;
                        removePiece(shape, r, c);
                        used[pieceIndex] = false;
                    }
                }
            }
        }
        return false;
    }

    /**
     * cek board apa udah penuh
     * I.S.: board puzzle terdefinisi
     * F.S.: true jika tidak ada sel kosong ('.'), false jika ada
     */
    private boolean boardIsFull() {
        for (int r = 0; r < rows; r++)
            for (int c = 0; c < cols; c++)
                if (board[r][c] == '.') return false;
        return true;
    }

    /**
     * cek apa shape dapat ditempatkan pada board di posisi (startRow,
startCol).
     * I.S.: matrix shape dan posisi board terdefinisi
     * F.S.: return true jika shape dapat ditempatkan tanpa tumpang
tindih, false sebaliknya.
     */
    private boolean canPlace(int[][] shape, int startRow, int startCol) {
```

```
        int rCount = shape.length, cCount = shape[0].length;
        if (startRow + rCount > rows || startCol + cCount > cols) return
false;
        for (int r = 0; r < rCount; r++) {
            for (int c = 0; c < cCount; c++) {
                if (shape[r][c] == 1 && board[startRow + r][startCol + c]
!= '.')
                    return false;
            }
        }
        return true;
    }

    /**
     * penempatan shape pada board di posisi (startRow, startCol)
     * I.S.: matrix shape dan posisi board
     * F.S.: board diperbarui, sel-sel yang sesuai
     */
    private void placePiece(int[][] shape, int startRow, int startCol,
char label) {
        int rCount = shape.length, cCount = shape[0].length;
        for (int r = 0; r < rCount; r++) {
            for (int c = 0; c < cCount; c++) {
                if (shape[r][c] == 1) {
                    board[startRow + r][startCol + c] = label;
                }
            }
        }
    }

    /**
     * hapus shape dari board pada posisi (startRow, startCol).
     * I.S.: matrix shape dan posisi board
     * F.S.: Sel-sel yang sebelumnya diisi dengan label dikembalikan ke
'.'
     */
    private void removePiece(int[][] shape, int startRow, int startCol) {
        int rCount = shape.length, cCount = shape[0].length;
        for (int r = 0; r < rCount; r++) {
```

```
            for (int c = 0; c < cCount; c++) {
                if (shape[r][c] == 1) {
                    board[startRow + r][startCol + c] = '.';
                }
            }
        }
    }

    //transformation
    private List<int[][]> getAllTransformations(int[][] shape) {
        List<int[][]> transformations = new ArrayList<>();
        List<int[][]> rotations = getRotations(shape);
        for (int[][] rot : rotations) {
            if (!contains(transformations, rot))
                transformations.add(rot);
            int[][] mirrored = mirror(rot);
            if (!contains(transformations, mirrored))
                transformations.add(mirrored);
        }
        return transformations;
    }

    private List<int[][]> getRotations(int[][] shape) {
        List<int[][]> rotations = new ArrayList<>();
        rotations.add(shape);
        int[][] rotated = shape;
        for (int i = 0; i < 3; i++) {
            rotated = rotate90(rotated);
            if (!contains(rotations, rotated))
                rotations.add(rotated);
        }
        return rotations;
    }

    private int[][] rotate90(int[][] shape) {
        int r = shape.length, c = shape[0].length;
        int[][] res = new int[c][r];
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
```

```
                res[j][r - 1 - i] = shape[i][j];
            }
        }
        return res;
    }

    private int[][] mirror(int[][] shape) {
        int r = shape.length, c = shape[0].length;
        int[][] res = new int[r][c];
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                res[i][c - 1 - j] = shape[i][j];
            }
        }
        return res;
    }

    private boolean contains(List<int[][]> list, int[][] candidate) {
        for (int[][] t : list) {
            if (areSame(t, candidate)) return true;
        }
        return false;
    }

    private boolean areSame(int[][] a, int[][] b) {
        if (a.length != b.length || a[0].length != b[0].length) return
false;
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[0].length; j++) {
                if (a[i][j] != b[i][j]) return false;
            }
        }
        return true;
    }

    /**
     * Mencetak board puzzle ke konsol
     * I.S.: Board (solusi) telah terbentuk
     * F.S.: Board dicetak ke konsol
```

```
     */
    public void printBoard() {
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                char ch = board[r][c];
                String color = getColor(ch);
                System.out.print(color + ch + ANSI_RESET);
            }
            System.out.println();
        }
    }

    private String getColor(char ch) {
        switch (Character.toUpperCase(ch)) {
            case 'A': return ANSI_RED;
            case 'B': return ANSI_GREEN;
            case 'C': return ANSI_YELLOW;
            case 'D': return ANSI_BLUE;
            case 'E': return ANSI_PURPLE;
            case 'F': return ANSI_CYAN;
            case 'G': return ANSI_WHITE;
            case 'H': return ANSI_RED;
            case 'I': return ANSI_GREEN;
            case 'J': return ANSI_YELLOW;
            case 'K': return ANSI_BLUE;
            case 'L': return ANSI_PURPLE;
            case 'M': return ANSI_CYAN;
            case 'N': return ANSI_WHITE;
            case 'O': return ANSI_RED;
            case 'P': return ANSI_GREEN;
            case 'Q': return ANSI_YELLOW;
            case 'R': return ANSI_BLUE;
            case 'S': return ANSI_PURPLE;
            case 'T': return ANSI_CYAN;
            case 'U': return ANSI_WHITE;
            case 'V': return ANSI_RED;
            case 'W': return ANSI_GREEN;
            case 'X': return ANSI_YELLOW;
            case 'Y': return ANSI_BLUE;
```

```
            case 'Z': return ANSI_PURPLE;
            default:  return ANSI_RESET;
        }
    }

    /**
     * Mengembalikan jumlah attempts yang telah dilakukan
     * I.S.: variabel attemptCount
     * F.S.: return nilai attemptCount
     */
    public long getAttemptCount() {
        return attemptCount;
    }
}
```

BAB 3 EKSPERIMEN

No.	Input	Output
1.	5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	Solusi: AGGGC AABCC EEBBF EEDFF EDDFF Waktu pencarian: 78 ms Banyak kasus yang ditinjau: 1313207 Apakah anda ingin menyimpan solusi? (ya/tidak)
2.	4 4 2 DEFAULT AAAA AAAA BBBB BBBB	Solusi: AAAA AAAA BBBB BBBB Waktu pencarian: 0 ms Banyak kasus yang ditinjau: 10 Apakah anda ingin menyimpan solusi? (ya/tidak)

3.	10 10 11 DEFAULT AAA AAA BBB BBB CCC CCC DDD DDD EEE EEE EEE FFF FFF FFF GGG GGG GGG GGG HHH HHH HHH HHH IIII IIII IIII IIII JJJ JJJ JJJ KKK KKK KKK	Solusi: AAABBBGGGG AAABBBGGGG CCCDDDGGGG CCCDDDHHHH EEEFFFHHHH EEEFFFHHHH EEEFFFIIII JJJKKKIIII JJJKKKIIII JJJKKKIIII Waktu pencarian: 39680 ms Banyak kasus yang ditinjau: 2377201791
4.	6 6 5 CUSTOM A AAAA BB BBBB CCCCC CCCCC DDDDD DDDDD E EEEEEE	Mode yang didukung hanya DEFAULT.

5.	4 4 3 DEFAULT A AA B BB C CC CC	Tidak ada solusi. Waktu pencarian: 18 ms Banyak kasus yang ditinjau: 88384
6.	6 6 8 DEFAULT A AA B BB C CC CC D DD DD E EE EE F FF FF G GG GG H HH HH	Solusi: AEEEFF AAEEFF DDDCBF DDCCBB HHCCGG HHHGGG Waktu pencarian: 13110 ms Banyak kasus yang ditinjau: 1063576006 Apakah anda ingin menyimpan solusi? (ya/tidak)

7.	5 6 6 DEFAULT A AA A B BB BBB C C C CC D DDD DD E E EE F F F F F	Solusi: AAACCC FAFCEE FDFBCE FDDBBE DDDBBB Waktu pencarian: 235 ms Banyak kasus yang ditinjau: 5966542 Apakah anda ingin menyimpan solusi? (ya/tidak)

LAMPIRAN

Link Repository

https://github.com/BenedictusNelson/Tucil1_13523150.git

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	✓	