

Laporan Tugas Kecil 1
IF2211 Strategi Algoritma

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

Disusun oleh:

Rendi Adinata
10123083



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

BAB I

DESKRIPSI MASALAH

Pada tugas kecil pertama yang berjudul Penyelesaian *IQ Puzzler Pro* dengan Algoritma *Brute Force* ini, akan diimplementasikan algoritma brute force untuk menyelesaikan permainan *IQ Puzzler Pro*, yakni sebuah permainan papan yang diproduksi oleh Smart Games. Tujuan utama permainan adalah mengisi seluruh papan dengan blok-blok puzzle yang tersedia, dengan tiap blok memiliki bentuk unik dan harus digunakan seluruhnya. Papan awal selalu kosong, dan setiap blok dapat dirotasi maupun dicerminkan untuk memaksimalkan kemungkinan penempatan.

Input program diberikan melalui berkas berekstensi .txt dengan format:

- Baris pertama: dua angka (N dan M) yang menentukan dimensi papan (NxM) serta angka P yang menunjukkan banyaknya blok puzzle.
- Baris kedua: sebuah string yang mengidentifikasi jenis kasus konfigurasi (misalnya DEFAULT, CUSTOM, atau PYRAMID).
- Selanjutnya: representasi bentuk masing-masing blok puzzle dengan karakter (huruf kapital A–Z) yang membedakan setiap blok.

Output yang diharapkan meliputi:

1. Konfigurasi papan yang terisi penuh oleh blok puzzle, dengan setiap blok ditandai dengan warna berbeda (output berwarna).
2. Waktu eksekusi pencarian solusi (dalam milidetik), hanya waktu proses algoritma brute force.
3. Jumlah iterasi atau kasus yang ditinjau oleh algoritma.
4. Opsi bagi pengguna untuk menyimpan hasil output ke dalam berkas .txt.

Projek ini harus menggunakan algoritma brute force murni tanpa memanfaatkan heuristik. Selain itu, terdapat spesifikasi bonus, seperti pembuatan GUI untuk visualisasi papan, penyimpanan output dalam bentuk file gambar, serta penanganan kasus konfigurasi custom dan konfigurasi piramida (3D).

Masalah yang dipecahkan mencakup:

- Menghasilkan solusi *tiling* untuk papan yang telah ditentukan dengan menggunakan seluruh blok yang tersedia.
- Mencari solusi dengan memeriksa setiap kemungkinan transformasi (rotasi dan refleksi) dari tiap blok.
- Menyediakan mekanisme input/output berbasis *file* sehingga pengguna dapat dengan mudah mengganti kasus uji dan menyimpan solusi.

BAB 2

ALGORITMA *BRUTE FORCE*

Pertama-tama, program meminta pengguna memasukkan nama file input dalam format .txt. File input harus mengikuti format sebagai berikut:

N M P

BoardType

<Piece 1>

<Piece 2>

...

<Piece P>

Baris pertama berisi tiga angka, M dan N (dimensi papan $M \times N$) serta P (jumlah puzzle pieces). Baris kedua adalah string, misalnya DEFAULT. Selanjutnya, untuk setiap puzzle piece (dari 1 hingga P), program membaca baris-baris yang menyusun bentuk potongan. Program membuat sebuah list kosong untuk menyimpan baris-baris dari masing-masing puzzle piece. Selama baris yang dibaca tidak kosong dan belum mencapai akhir file, baris tersebut ditambahkan ke list puzzle piece. Selanjutnya, list puzzle piece ini ditambahkan ke list keseluruhan. Kemudian file input ditutup.

Setelah input terbaca, objek Solver diinisialisasi dengan parameter M, N, dan list puzzle pieces. Selanjutnya, metode solve() dipanggil pada objek Solver untuk mencari solusi. Setelah proses pencarian selesai, hasil output (misalnya konfigurasi papan terisi, waktu eksekusi, dan jumlah kasus yang ditinjau) ditampilkan di layar.

Dalam mencari solusi pengisian papan dengan tetromino ini, diimplementasikan algoritma brute force/backtracking. Berikut adalah rincian *method-method* utama yang digunakan:

1. `canPlace(List<Point> shape, int x, int y)`

Method ini mengecek apakah suatu bentuk potongan (yang direpresentasikan sebagai list titik relatif) dapat ditempatkan pada posisi (x, y) di grid.

- Setiap titik dalam bentuk diubah menjadi koordinat absolut dengan menambahkan (x, y).
- Jika titik tersebut berada di luar batas grid atau sudah terisi, method mengembalikan `false`; jika semua titik valid, mengembalikan `true`.

2. `place(List<Point> shape, int x, int y, char letter)`

Setelah diketahui potongan dapat ditempatkan, method ini menandai posisi-posisi yang ditempati pada grid dan matriks solusi dengan karakter identifikasi (huruf) dari potongan tersebut.

3. `remove(List<Point> shape, int x, int y)`

Jika penempatan suatu potongan tidak menghasilkan solusi lengkap, method ini digunakan untuk menghapus potongan dari grid, mengembalikan sel-sel yang diisi ke kondisi kosong.

4. `gridFilled()`

Method ini melakukan pengecekan apakah seluruh sel pada grid telah terisi dengan potongan. Jika tidak ada sel kosong, maka grid dianggap terisi penuh dan solusi valid telah ditemukan.

5. `backtrack(int pieceIdx)`

- Basis Rekursif: Jika indeks potongan (`pieceIdx`) sama dengan jumlah total potongan, maka dilakukan pengecekan apakah grid sudah terisi penuh. Jika ya, variabel `solutionFound` diset ke `true`.
- Rekursi: Untuk potongan pada indeks tertentu, program mendapatkan semua transformasi (rotasi dan refleksi) yang mungkin melalui method `generateTransformations()`. Untuk setiap transformasi, algoritma mencoba menempatkan potongan di setiap posisi di grid. Jika penempatan berhasil (dicek oleh `canPlace()`), maka potongan ditempatkan dengan `place()` dan algoritma memanggil `backtrack(pieceIdx + 1)` untuk potongan berikutnya. Jika solusi tidak ditemukan pada cabang tersebut, `remove()` dipanggil untuk mengembalikan keadaan grid sehingga alternatif lain bisa dicoba.
- Variabel `attempts` dihitung untuk melacak jumlah total percobaan penempatan.

6. `solve()`

Method ini merupakan pembungkus untuk memulai proses backtracking. Dimulai dengan pencatatan waktu awal, kemudian `backtrack(0)` dipanggil. Setelah pencarian selesai, waktu eksekusi dihitung sebagai selisih waktu saat selesai dan waktu awal.

BAB 3

SOURCE CODE PROGRAM

```
import java.util.*;

import java.io.*;

public class TetrominoTilingSolver {

    public static class Point implements Comparable<Point> {

        int x, y;

        public Point(int x, int y) {

            this.x = x;

            this.y = y;

        }

        @Override

        public int compareTo(Point other) {

            if (this.x != other.x)

                return this.x - other.x;

            return this.y - other.y;

        }

    }

    public static List<Point> rotate(List<Point> shape) {

        List<Point> newShape = new ArrayList<>();

        for (Point p : shape) {

            newShape.add(new Point(p.y, -p.x));

        }

        return newShape;

    }

}
```

```

public static List<Point> reflect(List<Point> shape) {
    List<Point> newShape = new ArrayList<>();
    for (Point p : shape) {
        newShape.add(new Point(p.x, -p.y));
    }
    return newShape;
}

public static List<Point> normalize(List<Point> shape) {
    int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE;
    for (Point p : shape) {
        if (p.x < minX) minX = p.x;
        if (p.y < minY) minY = p.y;
    }
    List<Point> normalized = new ArrayList<>();
    for (Point p : shape) {
        normalized.add(new Point(p.x - minX, p.y - minY));
    }
    Collections.sort(normalized);
    return normalized;
}

    public static List<List<Point>>
generateTransformations(List<Point> original) {
    TreeSet<List<Point>> uniqueShapes = new TreeSet<>(new
Comparator<List<Point>>() {
        @Override
        public int compare(List<Point> shape1, List<Point>
shape2) {

```

```

        if (shape1.size() != shape2.size())
            return shape1.size() - shape2.size();
        for (int i = 0; i < shape1.size(); i++) {
            int cmp =
shape1.get(i).compareTo(shape2.get(i));
            if (cmp != 0)
                return cmp;
        }
        return 0;
    });

    List<Point> current = new ArrayList<>(original);
    for (int i = 0; i < 4; i++) {
        List<Point> normalized = normalize(current);
        uniqueShapes.add(normalized);

        List<Point> reflected = normalize(reflect(current));
        uniqueShapes.add(reflected);

        current = rotate(current);
    }

    return new ArrayList<>(uniqueShapes);
}

public static class Piece {
    List<Point> shape;
    char letter;

    public Piece(List<Point> shape, char letter) {

```

```

        this.shape = shape;

        this.letter = letter;
    }
}

public static Piece parsePiece(List<String> pieceLines) {
    List<Point> shape = new ArrayList<>();
    Character letter = null;

    for (int i = 0; i < pieceLines.size(); i++) {
        String line = pieceLines.get(i);

        for (int j = 0; j < line.length(); j++) {
            char ch = line.charAt(j);

            if (ch != ' ') {
                shape.add(new Point(i, j));

                if (letter == null)
                    letter = ch;
            }
        }
    }

    return new Piece(shape, letter == null ? '?' : letter);
}

public static class Solver {
    int M, N;

    boolean[][] grid;

    int attempts = 0;

    boolean solutionFound = false;

    char[][] solutionGrid;
}

```



```

        List<List<List<Point>>> piecesTransformations = new
ArrayList<>();

        List<Character> pieceLetters = new ArrayList<>();

        long duration;

        public Solver(int M, int N, List<List<String>>
piecesInput) {

            this.M = M;

            this.N = N;

            grid = new boolean[M][N];

            solutionGrid = new char[M][N];

            for (int i = 0; i < M; i++) {

                Arrays.fill(solutionGrid[i], '.');

            }

            for (List<String> pieceLines : piecesInput) {

                Piece piece = parsePiece(pieceLines);

                List<List<Point>> transformations =
generateTransformations(piece.shape);

                piecesTransformations.add(transformations);

                pieceLetters.add(piece.letter);

            }

        }

        public boolean canPlace(List<Point> shape, int x, int y)
{

            for (Point p : shape) {

                int nx = x + p.x;

                int ny = y + p.y;

                if (nx < 0 || nx >= M || ny < 0 || ny >= N ||
grid[nx][ny])

```

```

        return false;
    }

    return true;
}

public void place(List<Point> shape, int x, int y, char
letter) {
    for (Point p : shape) {
        grid[x + p.x][y + p.y] = true;
        solutionGrid[x + p.x][y + p.y] = letter;
    }
}

public void remove(List<Point> shape, int x, int y) {
    for (Point p : shape) {
        grid[x + p.x][y + p.y] = false;
        solutionGrid[x + p.x][y + p.y] = '.';
    }
}

public boolean gridFilled() {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            if (!grid[i][j])
                return false;
        }
    }

    return true;
}

```

```

public void backtrack(int pieceIdx) {
    if (pieceIdx == piecesTransformations.size()) {
        if (gridFilled()) {
            solutionFound = true;
        }
        return;
    }

    List<List<Point>> transformations =
piecesTransformations.get(pieceIdx);

    for (List<Point> shape : transformations) {
        for (int x = 0; x < M; x++) {
            for (int y = 0; y < N; y++) {
                attempts++;

                if (canPlace(shape, x, y)) {
                    place(shape, x, y,
pieceLetters.get(pieceIdx));

                    backtrack(pieceIdx + 1);

                    if (solutionFound)
                        return;

                    remove(shape, x, y);
                }
            }
        }
    }
}

public void solve() {
    long start = System.currentTimeMillis();

    backtrack(0);
}

```

```

        long end = System.currentTimeMillis();

        duration = end - start;

    }

    public boolean isSolutionFound() {

        return solutionFound;

    }

    public char[][] getSolutionGrid() {

        return solutionGrid;

    }

    public long getDuration() {

        return duration;

    }

    public int getAttempts() {

        return attempts;

    }

}

public static void main(String[] args) {

    Scanner inputScanner = new Scanner(System.in);

    System.out.println("Masukkan nama file: ");

    String fileName = inputScanner.nextLine();

    File inputFile = new File(fileName);

    List<String> colors = Arrays.asList(

        "\033[38;2;255;0;0m",      // Red

        "\033[38;2;0;255;0m",      // Green

```

```

        "\033[38;2;0;0;255m",      // Blue
        "\033[38;2;255;255;0m",    // Yellow
        "\033[38;2;255;165;0m",    // Orange
        "\033[38;2;128;0;128m",    // Purple
        "\033[38;2;0;255;255m",    // Cyan
        "\033[38;2;255;192;203m",  // Pink
        "\033[38;2;165;42;42m",    // Brown
        "\033[38;2;75;0;130m",     // Indigo
        "\033[38;2;255;215;0m",    // Gold
        "\033[38;2;173;216;230m",  // Light Blue
        "\033[38;2;0;128;0m",      // Dark Green
        "\033[38;2;128;128;128m",  // Gray
        "\033[38;2;192;192;192m",  // Silver
        "\033[38;2;0;0;128m",      // Navy
        "\033[38;2;128;0;0m",      // Maroon
        "\033[38;2;0;128;128m",    // Teal
        "\033[38;2;139;69;19m",    // Saddle Brown
        "\033[38;2;46;139;87m",    // Sea Green
        "\033[38;2;255;99;71m",    // Tomato
        "\033[38;2;186;85;211m",   // Medium Orchid
        "\033[38;2;30;144;255m",    // Dodger Blue
        "\033[38;2;154;205;50m",    // Yellow Green
        "\033[38;2;218;112;214m",  // Orchid
        "\033[38;2;107;142;35m"    // Olive Drab
    );

    try (Scanner sc = new Scanner(inputFile)) {
        int M = sc.nextInt();
        int N = sc.nextInt();
        int P = sc.nextInt();
    }

```

```

        sc.nextLine();

        sc.nextLine();

        List<String> remainingLines = new ArrayList<>();
        while (sc.hasNextLine()) {
            remainingLines.add(sc.nextLine());
        }

        List<List<String>> pieces = new ArrayList<>();
        int index = 0;
        for (int i = 0; i < P; i++) {
            List<String> pieceLines = new ArrayList<>();
            while (index < remainingLines.size() &&
remainingLines.get(index).trim().isEmpty()) {
                index++;
            }
            if (index >= remainingLines.size()) {
                break;
            }
            String firstLine = remainingLines.get(index);
            String trimmedFirst = firstLine.trim();
            if (trimmedFirst.isEmpty()) {
                index++;
                continue;
            }
            char pieceChar = trimmedFirst.charAt(0);
            pieceLines.add(firstLine);
            index++;
        }
    }
}

```

```

        while (index < remainingLines.size()) {

            String nextLine = remainingLines.get(index);

            String trimmedNext = nextLine.trim();

            if (trimmedNext.isEmpty()) {

                index++;

                continue;

            }

            if (trimmedNext.charAt(0) == pieceChar) {

                pieceLines.add(nextLine);

                index++;

            } else {

                break;

            }

        }

        pieces.add(pieceLines);

    }nSystem.out.println();

    Solver solver = new Solver(M, N, pieces);

    solver.solve();

    if (solver.isSolutionFound()) {

        char[][] solutionGrid = solver.getSolutionGrid();

        Set<Character> uniqueLetters = new HashSet<>();

        for (char[] row : solutionGrid) {

            for (char c : row) {

                if (c != '.' &&
!uniqueLetters.contains(c)) {

                    uniqueLetters.add(c);

                }

            }

        }
    }

```

```

    }

    List<Character> letters = new
ArrayList<>(uniqueLetters);

    List<String> shuffledColors = new
ArrayList<>(colors);

    Collections.shuffle(shuffledColors);

    while (shuffledColors.size() < letters.size()) {
        Collections.shuffle(colors);
        shuffledColors.addAll(colors);
    }

    Map<Character, String> colorMap = new
HashMap<>();

    for (int i = 0; i < letters.size(); i++) {
        colorMap.put(letters.get(i),
shuffledColors.get(i));
    }

    String reset = "\033[0m";
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            char c = solutionGrid[i][j];

            String color = colorMap.getOrDefault(c,
reset);

            System.out.print(color + c + reset + "
");

        }

        System.out.println();
    }
}

```



```

        System.out.println("Waktu pencarian: " +
solver.getDuration() + " ms");

        System.out.println("Banyak kasus yang ditinjau: " +
solver.getAttempts());

        System.out.println("Simpan solusi ke file?
(yes/no)");

        String choice =
inputScanner.nextLine().trim().toLowerCase();

        if (choice.equals("yes") || choice.equals("y")) {

            System.out.println("Masukkan nama file: ");

            String outFile = inputScanner.nextLine();

            try (PrintWriter writer = new
PrintWriter(outFile)) {

                for (int i = 0; i < M; i++) {

                    for (int j = 0; j < N; j++) {

                        writer.print(solutionGrid[i][j]);

                    }

                    writer.println();

                }

                System.out.println("Solusi disimpan di "
+ outFile);

            } catch (IOException e) {

                System.out.println("Gagal menyimpan: " +
e.getMessage());

            }

        } else {

            System.out.println("Tidak ada solusi yang
ditemukan.");

        }

```

```
        } catch (FileNotFoundException e) {  
            System.out.println("File tidak ditemukan: " +  
e.getMessage());  
        } finally {  
            inputScanner.close();  
        }  
    }  
}
```

BAB 4

INPUT DAN OUTPUT PROGRAM

Test case 1: Default dan terdapat solusi

Masukan	Keluaran
3 3 2 DEFAULT AAA A A AAA B	<pre> Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase1.txt A A A A B A A A A Waktu pencarian: 0 ms Banyak kasus yang ditinjau: 6 Simpan solusi ke file? (yes/no) no </pre>

Test case 2: Default dan terdapat solusi

Masukan	Keluaran
5 5 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<pre> Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase2.txt A A B B D A E B D D E E C C F E E C F F G G G F F Waktu pencarian: 8 ms Banyak kasus yang ditinjau: 147937 Simpan solusi ke file? (yes/no) no </pre>

--	--

Test case 3: Default dan terdapat solusi

Masukan	Keluaran
4 6 3 DEFAULT AA AA AA AAAAAA BBBB B B CCC CCC	<pre> Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase3.txt A A A A A A A A B B B B A A B C C C A A B C C C Waktu pencarian: 0 ms Banyak kasus yang ditinjau: 26 Simpan solusi ke file? (yes/no) no </pre>

Test case 4: Default dan tidak terdapat solusi

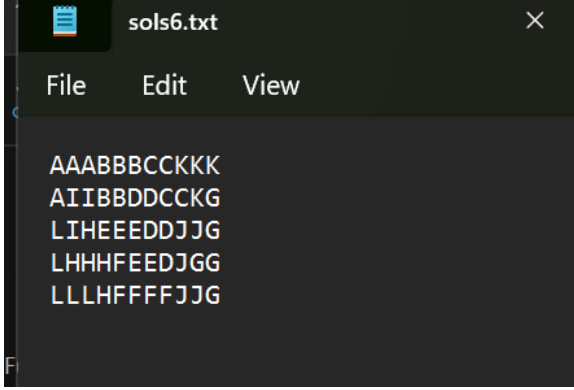
Masukan	Keluaran
5 7 3 DEFAULT AA AA BB BBB C CC D DD	<pre> Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase4.txt Tidak ada solusi yang ditemukan. </pre>

Test case 5: Default dan tidak terdapat solusi

Masukan	Keluaran
5 5 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GG	Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase5.txt Tidak ada solusi yang ditemukan.

Test case 6: Default dan tidak terdapat solusi

Masukan	Keluaran
5 11 12 DEFAULT AA A A BB BBB C CC C D DD	Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase6.txt A A A B B B C C K K K A I I B B D D C C K G L I H E E E D D J J G L H H H F E E D J G G L L L H F F F F J J G Waktu pencarian: 7468 ms Banyak kasus yang ditinjau: 1337979109 Simpan solusi ke file? (yes/no) yes Masukkan nama file: sols6.txt Solusi disimpan di sols6.txt

DD E E EE E FF F F F GGGG G H HH HH I II JJJ JJ K KK K L L LLL	
---	--

Test case 7: Default dan tidak terdapat solusi

Masukan	Keluaran
5 5 6 DEFAULT A A AA BB BB CCCCC C C	Masukkan nama file: C:\Users\rendy\Documents\IF\Java\tcase7.txt Tidak ada solusi yang ditemukan.

DDDD EEE EEE F F FFFF F F F	
---	--

REFERENSI

- https://www.w3schools.com/java/java_type_casting.asp diakses pada 21 Februari 2025 pukul 18.37 WIB
- <https://www.geeksforgeeks.org/dsa-tutorial-learn-data-structures-and-algorithms/> diakses pada 21 Februari 2025 pukul 19.04 WIB
- <https://docs.oracle.com/en/java/> diakses pada 23 Februari 2025 pukul 20.30
- <https://stackoverflow.com/questions/20696511/where-does-the-saved-file-go-j-ava> diakses pada 24 Februari 2025 pukul 08.24 WIB

LAMPIRAN

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	✓	

pranala repository: https://github.com/xinuza/Tucil1_10123083.git

