LAPORAN TUGAS KECIL 1

IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro Menggunakan Algoritma Brute Force

Dipersiapkan oleh:

M. Rayhan Farrukh 13523035

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Bab 1: Algoritma Brute Force

1.1. Brute Force

Algoritma *Brute Force*, adalah algoritma dimana penyelesaian masalah dilakukan dengan cara *straightforward*, tanpa memikirkan cara untuk membuat penyelesaian masalah tersebut lebih efisien. Artinya, algoritma ini dilakukan dengan mencoba seluruh kemungkinan solusi, dengan tujuan akhir adalah masalah dapat diselesaikan, tidak peduli berapa lama waktu yang dibutuhkan untuk penyelesaian tersebut.

1.2. Heuristik

Heuristik adalah teknik yang dapat digunakan untuk mengurangi atau membatasi kasus-kasus yang perlu dievaluasi dari suatu proses penyelesaian masalah. Teknik heuristik mengabaikan pendekatan matematis, untuk menggunakan pendekatan yang tidak formal, yaitu intuisi manusia atau *common sense*.
Teknik heuristik dapat digunakan untuk membatasi kasus yang perlu dihitung oleh algoritma *brute force*, membuat *brute force* menjadi lebih efisien.

1.3. Algoritma Yang Digunakan

Algoritma *brute force* yang digunakan untuk penyelesaian *puzzle* pada program ini menggunakan teknik *backtracking* dengan rekursi. Teknik yang digunakan serupa dengan teknik yang digunakan untuk penyelesaian *Sudoku* pada [1, p. 46]. Langkah-langkah algoritmanya adalah sebagai berikut:

- Program mulai dari blok *puzzle* pertama yang tersimpan pada sebuah *list* yang berisikan blok *puzzle*.
- Masing-masing blok akan dicoba untuk diletakkan pada *cell-cell* papan.
- Jika blok dapat diletakkan pada *cell* tersebut, letakkan blok lalu lanjut untuk blok berikutnya menggunakan rekursi.
- Jika blok tidak dapat diletakkan, lanjut dengan mengecek transformasi selanjutnya dari blok (rotasi dan pencerminan).
- Pada kasus blok berhasil diletakkan, proses rekursi akan mengevaluasi seluruh blok *puzzle* hingga terakhir untuk kasus tersebut.
- Jika setelah seluruh blok dievaluasi papan penuh, maka *puzzle* dinyatakan telah *solved* dan program berhenti.

- Jika tidak, maka lakukan *backtracking* dan hapus blok terakhir, kemudian lanjutkan dengan mengecek transformasi berikutnya.
- Jika semua transformasi dari semua blok telah dievaluasi, dan papan masih belum terselesaikan, maka konfigurasi *puzzle* tersebut tidak memiliki solusi.

Pseudocode algoritma:

```
FUNGSI recursive_solver(listBlok, indexBlok):
    IF indexBlok > indexBlokTerakhir THEN
        RETURN cekApakahPuzzleSelesai()

    blokSekarang ← listBlok[indexBlok]

    FOR setiap baris papan:
        FOR setiap kolom papan:
            FOR setiap rotasi blok: // (0 - 270 derajat)
                FOR setiap pencerminan blok: // (normal, vertikal, horizontal)
                    IF blok dapat diletakkan THEN
                        letakkanBlok()

                        // Cek kasus ini sampai selesai
                        IF recursive_solver(listBlok, indexBlok+1) = TRUE THEN
                            RETURN TRUE // puzzle selesai
                        ELSE
                            backtrackDanHapusBlok()

    RETURN FALSE // Puzzle tidak ada solusi
```

Algoritma ini mengecek semua kemungkinan penempatan blok pada papan tanpa melakukan heuristik untuk mengurangi jumlah iterasi. Oleh karena itu, algoritma ini sangat tidak efisien.

Kompleksitas waktu *worst case* untuk algoritma ini adalah $O(N \times M \times 12^P)$, dengan N x M adalah dimensi papan, P adalah jumlah blok *puzzle*, dan 12 adalah seluruh kemungkinan transformasi satu blok *puzzle*. Oleh karena itu, algoritma *brute force* ini sangat tidak bagus digunakan bahkan untuk jumlah data relatif tidak terlalu besar.

Bab 2: Kode Program

2.1. Kelas Main

Kelas Main adalah kelas utama yang akan dijalankan saat menge-run program. Kelas ini berisi user interface pada CLI, dan bertanggung jawab memanggil fungsi-fungsi utama dari kelas-kelas lainnya untuk memulai penyelesaian puzzle.

Source Code dapat dilihat pada *repository*.

2.2. Kelas Block

Kelas Block berfungsi untuk menyimpan masing-masing blok atau piece dari *puzzle*, serta *method-method* yang digunakan untuk mentranslasikan masukan dari *text file* ke dalam bentuk matriks karakter. Selain itu, kelas ini juga memiliki method-method untuk mentransformasikan bentuk blok untuk kegunaan penyelesaian puzzle.

Source Code:

```
import java.util.ArrayList;
import java.util.Arrays;

public class Block {
    public char id;
    public char[][] shape;
    public int rows;
    public int cols;

    public Block(char id, ArrayList<String> shape){
        this.id = id;
        this.shape = filterEmptyColumns(padAndFormatShape(shape));
        this.rows = this.shape.length;
        this.cols = this.shape[0].length;
    }

    private static ArrayList<char[]> padAndFormatShape(ArrayList<String> shape){
        ArrayList<char[]> formattedShape = new ArrayList<>();

        int row = shape.size();
        int maxCols = 0;
        for (String line : shape) {
            maxCols = Math.max(maxCols, line.length());
```

```
        }

        for (int i = 0; i < row; i++) {
            char[] currRow = new char[maxCols];
            Arrays.fill(currRow, ' ');  // Pre populate
            String currentLine = shape.get(i);

            for (int j = 0; j < currentLine.length(); j++) {
                currRow[j] = currentLine.charAt(j);
            }

            formattedShape.add(currRow);
        }
        return formattedShape;
    }

    private static char[][] filterEmptyColumns(ArrayList<char[]> formattedShape) {
        int rows = formattedShape.size();
        int cols = formattedShape.get(0).length;
        boolean[] keepColumn = new boolean[cols];
        int finalCols = 0;

        for (int j = 0; j < cols; j++) {
            for (int i = 0; i < rows; i++) {
                if (formattedShape.get(i)[j] != ' ') {
                    keepColumn[j] = true;
                    finalCols++;
                    break;
                }
            }
        }
        char[][] filteredShape = new char[rows][finalCols];
        for (int i = 0; i < rows; i++) {
            int newCol = 0;
            for (int j = 0; j < cols; j++) {
                if (keepColumn[j]) {
                    filteredShape[i][newCol++] = formattedShape.get(i)[j];
                }
            }
        }

        return filteredShape;
    }
}
```

2.3. Kelas Board

Kelas *Board* berfungsi sebagai tempat untuk menyimpan *state* papan *puzzle* yang digunakan. Kelas ini juga menyimpan fungsi-fungsi logika atau algoritma *brute force* utama yang digunakan dalam penyelesaian *puzzle*.

Source Code:

```
public class Board {
     public long executeTime = 0;
    public int casesCount = 0;
    public char[][] board;
    public boolean solved = false;
    int rows;
    int cols;

    public Board(int N, int M){
        this.board = new char[N][M];
        // Populate dengan ' '
        for (char[] row : this.board) {
            Arrays.fill(row, ' ');
        }
        this.rows = N;
        this.cols = M;
    }

    public void solver(List<Block> puzzleBlocks) {
        long startTime = System.currentTimeMillis();
        if (recPuzzleSolver(puzzleBlocks, 0)) {
            System.out.println("Found a solution!\n");
            Output.printBlock(board);
            solved = true;
        }
        else {
            System.out.println("No solutions found for the given configuration of blocks");
        }
        this.executeTime = System.currentTimeMillis() - startTime;
        System.out.print("\nExecution time: ");
        Utils.printColorStr(Output.fm.format(executeTime) + " ms", 10, true);
        System.out.print("Cases evaluated: ");
        Utils.printColorStr("" + Output.fm.format(casesCount) + "\n", 10, true);
```

```
    }

    public boolean recPuzzleSolver(List<Block> puzzleBlocks, int blockIndex){
        if (blockIndex >= puzzleBlocks.size()) {
            return this.isSolved(); // Basis
        }

        Block currBlock = puzzleBlocks.get(blockIndex);

        for (int row = 0; row < rows; row++){
            for (int col = 0; col < cols;col++){

                for (int rot = 0; rot < 4; rot++){
                    char[][] rotatedBlock = currBlock.rotateBlock(rot);
                    for(int mirror = 0; mirror < 3; mirror++){
                        char[][] mirroredBlock = Block.mirrorBlock(rotatedBlock, mirror);

                        casesCount++;
                        if (isPlaceable(mirroredBlock, row, col)){
                            placeBlock(mirroredBlock, row, col);

                            // Rekursi
                            if (recPuzzleSolver(puzzleBlocks, blockIndex+1)){
                                return true;
                            } else {
                                casesCount++;
                                                    removeBlock(mirroredBlock, row, col,
puzzleBlocks.get(blockIndex).id);
                            }
                        }
                    }
                }
            }
        }

        return false;
    }
}
```

2.4. Kelas Input

Kelas *Input* berisikan metode bagi program untuk membaca dan menginterpretasikan isi dari *file* input yang dimasukkan, serta memvalidasi input sesuai batasan-batasan tertentu yang dibuat agar program

tidak pernah keluar atau berhenti karena terjadi *error*. Dalam penggunaannya, kelas ini menyimpan konfigurasi yang dibuat di dalam *file* input, seperti konfigurasi dari papan jika menggunakan mode *custom*, serta blok-blok *puzzle* yang disimpan sebagai sebuah *list* dari objek *Block*.

Source Code:

```
public class Input {
    public static Input handleFileInput() {
        Scanner scanner = new Scanner(System.in);
        try {
            Utils.printColorStr("Enter input filename (.txt): ", 190, false);
            String filepath = "IO/input/" + scanner.nextLine();
            System.out.print("Checking " + filepath);

            File inputFile = new File(filepath);
            BufferedReader reader = new BufferedReader(new FileReader((inputFile)));

            String firstLine = reader.readLine();

            if (!firstLine.matches("\\d+\\s+\\d+\\s+\\d+\\s*$")){
                reader.close();
                throw new IOException("Invalid line: " + firstLine);
            }

            int[] NMP = Arrays.stream(firstLine.split(" "))
                                    .mapToInt(Integer::parseInt)
                                    .toArray();

            if (NMP[2] < 1 || NMP[2] > 26) {
                reader.close();
                throw new IllegalStateException("Puzzle blocks must be between 1 - 26");
            }

            System.out.print(".");  // Loading

            String secondLine = reader.readLine();
            if (!"DEFAULT".equals(secondLine) && !"CUSTOM".equals(secondLine)){
                reader.close();
                throw new IllegalStateException("Invalid mode: " + secondLine);
            }

            boolean isCustom = "CUSTOM".equals(secondLine);
            ArrayList<String> board = new ArrayList<>();
            if (isCustom){board = Utils.parseCustomBoard(reader, NMP[0], NMP[1]);}
```

```
            ArrayList<String> blockInputs = new ArrayList<>();
            String line;
            while ((line = reader.readLine()) != null) {
                if (Utils.containsInvalidChars(line, false)){
                            throw new IllegalStateException("Found invalid characters when
evaluating blocks!");
                }
                if (!line.trim().isEmpty()) {
                    blockInputs.add(line);
                }
            }
            reader.close();

            System.out.print(".");  // Loading

            if (blockInputs.size() < 1){
                throw new IllegalStateException("No blocks found!");
            }
            ArrayList<Block> listOfPuzzleBlocks = Utils.parseBlocks(blockInputs);

            if (listOfPuzzleBlocks.size() != NMP[2]){
                if (listOfPuzzleBlocks.size() > NMP[2]){
                    int lastIdx;
                          System.out.println("Warning! Amount of blocks found is more than
declared, some blocks won't be included");
                    while ((lastIdx = listOfPuzzleBlocks.size()) > NMP[2]){
                        listOfPuzzleBlocks.remove(lastIdx-1);
                    }
                } else {
                        throw new IllegalStateException("Amount of blocks found doesnt match
what is declared!"); }
            }

            System.out.println(".");  // Loading

            Input result = new Input(NMP, secondLine, listOfPuzzleBlocks, isCustom, board);
            int boardSize = isCustom ? Board.getEffectiveCells(board) : result.N * result.M;
            int blocksEffectiveCells = result.sumEffectiveCells();
            if ( boardSize != blocksEffectiveCells ) {
                   throw new IllegalStateException("Size of board does not equal the sum of
puzzle block's size!");
            }
```

```
            return result;
        }
         catch (FileNotFoundException e) {
            Utils.clearScreen();
            Utils.printColorStr("Error: File not found!", 9, true);
        } catch (IllegalStateException e) {
            Utils.clearScreen();
            Utils.printColorStr("Error: " + e.getMessage(), 9, true);
        } catch (IOException e) {
            Utils.clearScreen();
            Utils.printColorStr("Error: IO failed!", 9, true);
        }
        return null;
    }
}
```

2.5. Kelas Output

Kelas *Output* berisikan kode-kode yang berguna untuk menampilkan hasil penyelesaian *puzzle* oleh program. *Output* yang tersedia pada program terdiri dari *output* terminal, *text file*, serta gambar.

Source Code untuk output gambar*:*

```
public class Output {
    public static final DecimalFormat fm = new DecimalFormat("#,###");
    static {
        DecimalFormatSymbols symbols = fm.getDecimalFormatSymbols();
        symbols.setGroupingSeparator('.');
        fm.setDecimalFormatSymbols(symbols);
    }

    public static void handleImageOutput(Board board){

        Scanner scanner = new Scanner(System.in);
        String filepath = "IO/results/";
        createDirectory(filepath);
        while (true) {
            Utils.printColorStr("Enter image filename (.png): ", 190, false);
            filepath = filepath + scanner.nextLine();
            if (filepath.equals("IO/results/") || !filepath.endsWith(".png")){
                Utils.clearScreen();
                 Utils.printColorStr("Filename can't be empty, and must be of type png!", 9,
true);
                Utils.enterToContinue();
```

```
                Utils.clearScreen();
            } else {break;}
        }

        try {
            System.out.print("\nWriting to " + filepath + "...");
            BufferedImage image = createImage(board);
            ImageIO.write(image, "png", new File(filepath));
            Utils.clearScreen();
            System.out.println("Image written to " + filepath);
        } catch (IOException e) {
            Utils.printColorStr("Error creating image!",9,true);
        }
    }

    public static BufferedImage createImage(Board board) {
        BufferedImage image;
        Graphics2D g2d;
        int height;
        int width;
        Font result;

        if (!board.solved) {
            width = 100;
            height = 100;
                            image = new BufferedImage(width + 300, height + 200,
BufferedImage.TYPE_INT_ARGB);
            g2d = image.createGraphics();
            result = new Font("Arial", Font.BOLD, 20);
            g2d.setFont(result);
            g2d.drawString("No solutions found for the given", 20, 40);
            g2d.drawString("configuration of blocks", 20,  80);
        } else {
            char[][] bord = board.board;
            int rows = board.rows;
            int cols = board.cols;

            int cellWidth = 100;
            int cellHeight = 100;
            width = cols * cellWidth;
            height = rows * cellHeight;


            image = new BufferedImage(width, height + 80, BufferedImage.TYPE_INT_ARGB);
```

```
            g2d = image.createGraphics();
            Font font = new Font("Arial", Font.BOLD, 38);
            FontMetrics fontMetrics = g2d.getFontMetrics();
            g2d.setFont(font);

            g2d.setColor(Color.DARK_GRAY);
            g2d.fillRect(0, 0, width, height + 400);

            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    char currLetter = bord[i][j];
                    if (currLetter == '.') {
                        continue;
                    }

                    int colorIndex = currLetter - 'A';
                    g2d.setColor(new Color(
                        ansiToRGB[colorIndex][0],
                        ansiToRGB[colorIndex][1],
                        ansiToRGB[colorIndex][2]
                        ));

                    int circleDiameter = 80;
                    int xCircle = j * cellWidth + (cellWidth - circleDiameter) / 2;
                    int yCircle = i * cellHeight + (cellHeight - circleDiameter) / 2;
                    g2d.fillOval(xCircle, yCircle, circleDiameter, circleDiameter);

                    g2d.setColor(Color.WHITE); // warna huruf
                    String text = String.valueOf(currLetter);
                    int textWidth = fontMetrics.stringWidth(text);
                    int textHeight = fontMetrics.getAscent();
                    int xText = xCircle + (circleDiameter - textWidth) / 2 - 6;
                    int yText = yCircle + (circleDiameter + textHeight) / 2 + 8;
                    g2d.drawString(text, xText, yText);
                    }
                }
                result = new Font("Arial", Font.BOLD, 30);
            }

        g2d.setFont(result);
        g2d.drawString("Execute time: " + fm.format(board.executeTime) + " ms", 20, height +
30);
        g2d.drawString("Cases evaluated: " + fm.format(board.casesCount), 20, height + 70);
```

```
        g2d.dispose();
        return image;
    }
}
```

2.6. Kelas Utils

Kelas *Utils* adalah kelas yang menyimpan fungsi-fungsi utilitas umum yang tidak cocok dimasukkan ke kelas-kelas lainnya. Kelas ini dibuat agar kode dari kelas-kelas lainnya tidak terlalu panjang.

Source Code dapat dilihat pada *repository*.

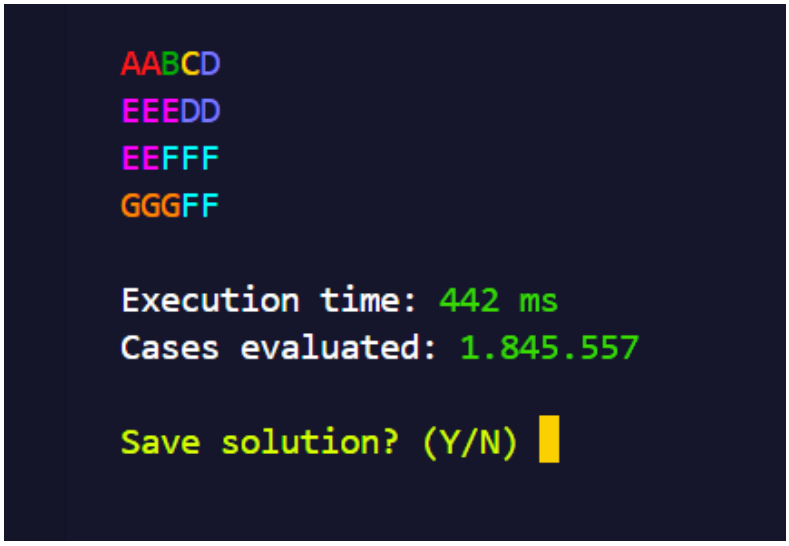
Bab 3: Pengujian

1. Kasus Uji 1

a. Input

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

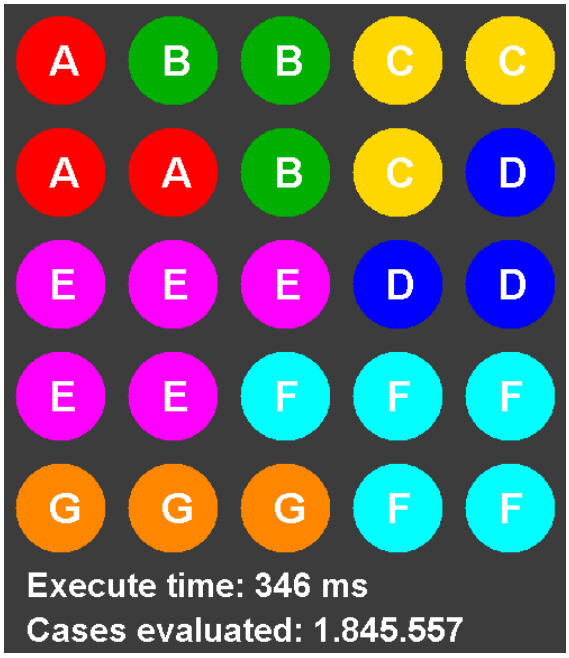
b. Output Terminal



c. **Output File**

```
1   ABBCC
2   AABCD
3   EEEDD
4   EEFFF
5   GGGFF
6
7   Execution time: 441
8   Cases evaluated: 1.845.557
```

d. **Output Gambar**



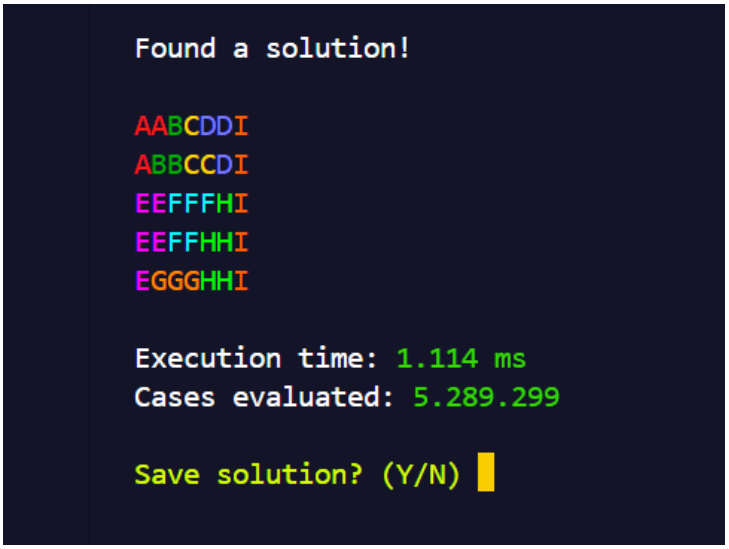
2. Kasus Uji 2

a. **Input**

```
5 7 9
DEFAULT
AA
A
B
BB
C
CC
```

D
DD
EE
EE
E
FF
FF
F
GGG
HHH
HH
IIII

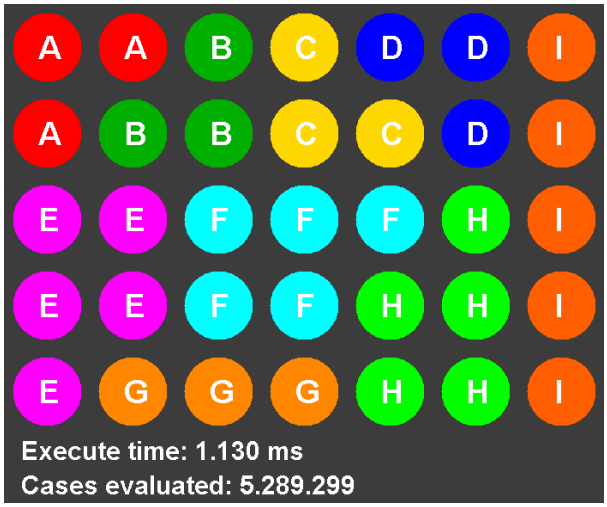
b. Output Terminal



c. **Output File**

```
1  AABCDDI
2  ABBCCDI
3  EEFFFHI
4  EEFFHHI
5  EGGGHHI
6
7  Execution time: 1.139
8  Cases evaluated: 5.289.299
```

d. **Output Gambar**



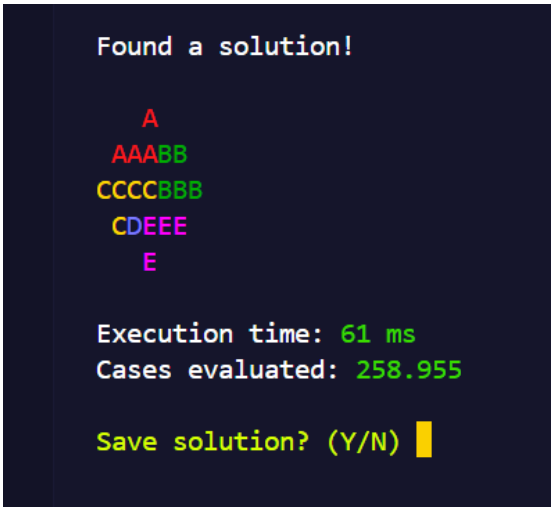
3. Kasus Uji 3

a. **Input**

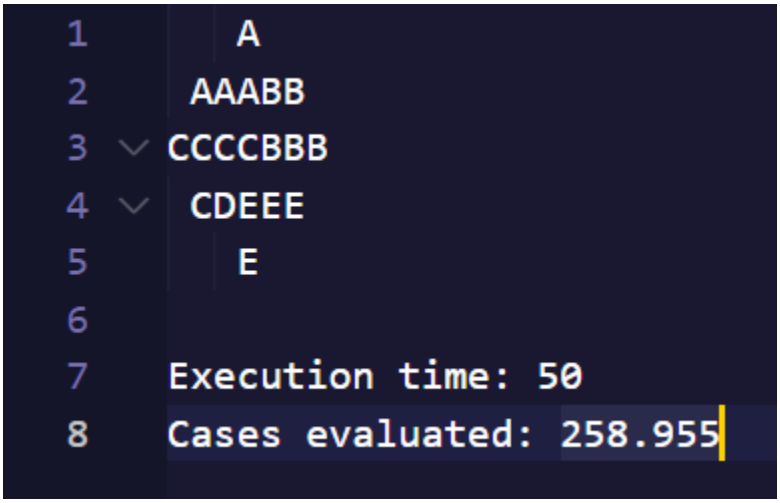
```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
```

```
BB
BBB
CCCC
 C
D
EEE
E
```

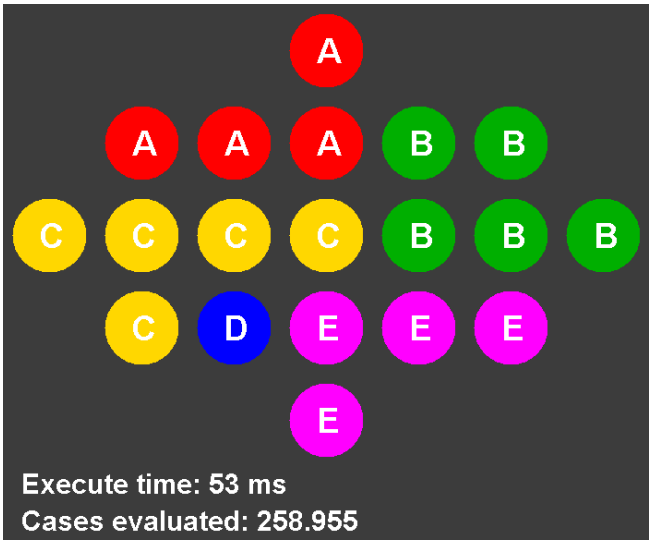
b. Output Terminal



c. Output File



b. Output Gambar



4. Kasus Uji 4

a. Input

b.

```
5 7 5
CUSTOM
XXXXXXX
...X...
.XXXXX.
...X...
.XXXXX.
A
AAA
BB
BBB
CCCC
 C
D
EEE
E
```

c. Output Terminal

```
Starting solver...

No Solutions found for the given configuration of blocks

Execution time: 3 ms
Cases evaluated: 10.524

Save solution? (Y/N)
```

d. Output File

```
1   No solutions found for the given configuration of blocks
2
3   Execution time: 3
4   Cases evaluated: 10.524
```

e. Output Gambar

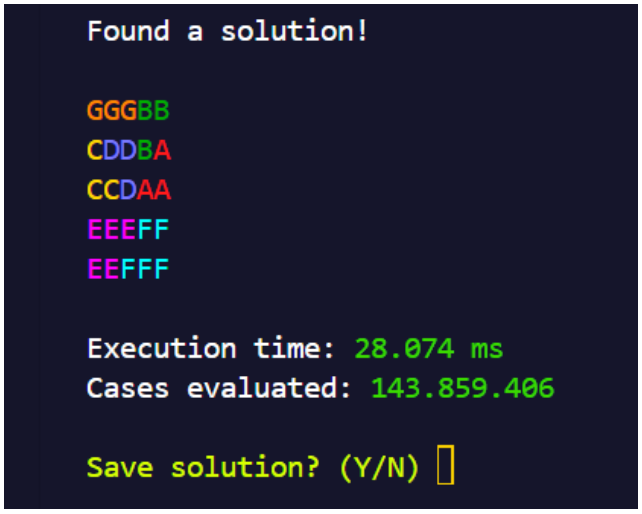
5. Kasus Uji 5

a. Input

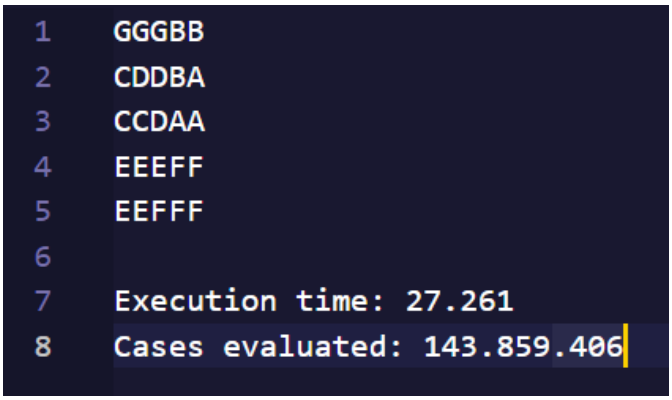
```
5 5 7
DEFAULT
GGG
B
BB
C
CC
E
EE
EE
D
```

DD
FF
FF
F
A
AA

b. Output Terminal



c. Output File



d. Output Gambar

6. Kasus Uji 6

a. Input

```
5 5 10
CUSTOM
GGG
B
BB
C
CC
E
EE
EE
D
DD
FF
FF
F
A
AA
```

b. Output Terminal

```
Error: Custom board can only contain '.' or 'X', also ensure the dimensions of the board are correct

Press enter to continue...
```

7. Kasus Uji 7

a. Input

```
5 5 13
DEFAULT
GGG
EE
BB
C
FF
CC
E
EE
D
DD
FF
B
F
A
AA
LLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLL
LLLLLLLLLLLLLLLLLLLLL
```

b. Output Terminal

```
Error: Size of board does not equal the sum of puzzle block's size!

Press enter to continue...
```

Referensi

[1]. R. Munir, "Algoritma Brute Force (Bagian 1)" Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf. [Accessed: Feb. 21, 2025].

[2]. R. Munir, "Algoritma Brute Force (Bagian 2)" Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/03-Algoritma-Brute-Force-(2025)-Bag2.pdf. [Accessed: Feb. 21, 2025].

[3]. Oracle, "Javadoc Tool," Java SE Documentation, 2015. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html. [Accessed: Feb. 21, 2025].

[4]. Brilliant.org, "Recursive Backtracking," Brilliant. [Online]. Available: https://brilliant.org/wiki/recursive-backtracking/. [Accessed: Feb. 21, 2025].

[5]. Oracle, "Drawing on an Image," Java 2D API, 2015. [Online]. Available: https://docs.oracle.com/javase/tutorial/2d/images/drawonimage.html. [Accessed: Feb. 21, 2025].

[6]. Oracle, "Working with Images," Java 2D API, 2015. [Online]. Available: https://docs.oracle.com/javase/tutorial/2d/images/index.html. [Accessed: Feb. 21, 2025].

Lampiran

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

Repository:

https://github.com/grwna/Tucil1_13523035