TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

Dipersiapkan oleh:

Clarissa Nethania Tambunan 13523016

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
Dr. Ir. Rinaldi Munir, M.T.
Menterico Adrian, S.T, M.T.

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

2025

1. Algoritma Brute Force Program IQ Puzzler Pro

Program IQ Puzzle Pro menggunakan algoritma brute force yang terdapat pada folder src tepatnya di dalam file Puzzler.java dengan berikut adalah notasi pseudocodenya:

```
class Puzzler:
   attributes:
      int N, M
      List<List<String>> blocks
      char[][] board
      int caseCount

   constructor Puzzler(int N, int M, List<List<String>> blocks):
      set N to N
      set M to M
      set blocks to blocks
      initialize board with dimensions N x M with empty spaces
      set caseCount to 0

   method solve() -> boolean:
      return solve(0)

   private method solve(int blockIndex) -> boolean:
      increment caseCount
      if blockIndex equals size of blocks:
         return isBoardFull()

      List<String> block = blocks.get(blockIndex)
      Set<List<String>> allOrientations = getAllUniqueOrientations(block)

      for each orientation in allOrientations:
         for i from 0 to N - size of orientation:
            for j from 0 to M - length of orientation[0]:
               if canPlaceBlock(orientation, i, j):
                  placeBlock(orientation, i, j)
                  if solve(blockIndex + 1):
                     return true
                  removeBlock(orientation, i, j)

      return false

   private method getAllUniqueOrientations(List<String> block) -> Set<List<String>>:
      Set<List<String>> orientations = new HashSet<>()
      orientations.add(block)
      orientations.add(rotate90(block))
      orientations.add(rotate180(block))
      orientations.add(rotate270(block))
      orientations.add(mirrorHorizontal(block))
      orientations.add(mirrorVertical(block))
      orientations.add(mirrorHorizontal(rotate90(block)))
      orientations.add(mirrorVertical(rotate90(block)))
```

```
    orientations.add(mirrorHorizontal(rotate180(block)))
    orientations.add(mirrorVertical(rotate180(block)))
    orientations.add(mirrorHorizontal(rotate270(block)))
    orientations.add(mirrorVertical(rotate270(block)))
    return orientations

private method rotate90(List<String> block) -> List<String>:
    int rows = size of block
    int cols = length of block[0]
    List<String> rotated = new ArrayList<>()
    for j from 0 to cols:
        StringBuilder newRow = new StringBuilder()
        for i from rows - 1 to 0:
            if j < length of block[i]:
                newRow.append(block[i].charAt(j))
            else:
                newRow.append(' ')
        rotated.add(newRow.toString())
    return rotated

private method rotate180(List<String> block) -> List<String>:
    return rotate90(rotate90(block))

private method rotate270(List<String> block) -> List<String>:
    return rotate90(rotate180(block))

private method mirrorHorizontal(List<String> block) -> List<String>:
    List<String> mirrored = new ArrayList<>()
    for each row in block:
        mirrored.add(new StringBuilder(row).reverse().toString())
    return mirrored

private method mirrorVertical(List<String> block) -> List<String>:
    List<String> mirrored = new ArrayList<>(block)
    for i from 0 to size of block / 2:
        String temp = mirrored.get(i)
        mirrored.set(i, mirrored.get(size of block - 1 - i))
        mirrored.set(size of block - 1 - i, temp)
    return mirrored

private method canPlaceBlock(List<String> block, int row, int col) -> boolean:
    for i from 0 to size of block:
        for j from 0 to length of block[i]:
            if block[i].charAt(j) != ' ' and (row + i >= N or col + j >= M or board[row + i][col + j] != ' '):
                return false
    return true

private method placeBlock(List<String> block, int row, int col):
    for i from 0 to size of block:
        for j from 0 to length of block[i]:
            if block[i].charAt(j) != ' ':
```

```
            board[row + i][col + j] = block[i].charAt(j)

    private method removeBlock(List<String> block, int row, int col):
        for i from 0 to size of block:
            for j from 0 to length of block[i]:
                if block[i].charAt(j) != ' ':
                    board[row + i][col + j] = ' '

    private method isBoardFull() -> boolean:
        for i from 0 to N:
            for j from 0 to M:
                if board[i][j] == ' ':
                    return false
        return true

    public method printBoard():
        for i from 0 to N:
            for j from 0 to M:
                char c = board[i][j]
                print Color.getColor(c) + c + Color.reset()
            print new line

    public method getBoard() -> char[][]:
        return board

    public method getCaseCount() -> int:
        return caseCount
```

Berdasarkan notasi pseudocode di atas, berikut adalah langkah-langkah dalam algoritma brute force yang digunakan pada program tersebut.

1. Setelah main.java berhasil mendapatkan informasi nilai masukan dari variabel N, M, dan P serta menerima pilihan DEFAULT yang artinya menggunakan aturan permainan seperti biasa yaitu papan harus penuh atau tidak ada solusi dan juga telah selesai menerima masukan blok yang diakhiri kata ‘stop’ untuk melakukan perhentian dari masukan blok apabila jumlah blok sudah terpenuhi sesuai dengan nilai variabel P, di dalam Puzzler.java akan dilakukan inisiasi papan N x M dengan cara mengisinya dengan spasi kosong (‘ ‘) yang diisi pada board.
2. Kemudian, fungsi solve() dipanggil untuk memulai proses penyelesaian dengan memanggil metode rekursif solve(int blockIndex) dengan indeks blok awal adalah 0.
   Lalu, fungsi solve(int blockIndex) adalah fungsi yang melakukan rekursif untuk mencari solusi dengan cara menempatkan setiap blok pada papan. Jika semua blok telah ditempatkan dan ketika dilakukan pengecekan oleh fungsi isBoardFull dan terbukti benar maka akan dikembalikan *true*. Jika tidak, maka akan dilakukan penempatan blok yang berde secara berulang kali hingga mendapatkan solusi yang benar. Jika solusi ditemukan, maka akan dikembalikan true tetapi jika tidak,akan dilakukan penghapusan blok ataupun mengubah posisi blok ke tempat lain dan apabila sampai akhir rekursi tetapi tidak berhasil mengisi papan hingga penuh maka tidak mengeluarkan tampilan papan.
3. Setelah menemukan solusi dari blok-blok tersebut yang mengisi papan dengan penuh maka pada printBoard akan dipanggil fungsi dari file Color.java untuk mewarnai teks huruf kapital dari blok agar setiap huruf yang ditampilkan memiliki warna yang unik artinya setiap huruf yang berbeda memiliki warnanya sendiri agar dapat dilihat blok yang sudah dimasukkan tadi melalui file main.java

Namun, setelah dilakukan pengecekan pada algoritma tersebut sebelumnya dapat berjalan tetapi program tersebut tidak dapat menampilkan papan yang berisikan blok sebagai solusi meskipun program tersebut sudah dapat membaca masukan secara manual dan membaca file berekstensi .txt sehingga saya masih harus belajar untuk dapat mengimplementasikan algoritma brute force tersebut pada program IQ Puzzle Pro. Kemudian pada File.java digunakan sebagai fungsi yang membaca file berekstensi .txt serta pada file Color.java untuk melakukan pewarnaan secara acak pada huruf blok.

2. Source Code Program

Source Code di file Color.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

public class Color {
    private static final Map<Character, String> colorMap = new
HashMap<>();
    private static final String[] colors = {
        "\u001B[30m", // Black
        "\u001B[31m", // Red
        "\u001B[32m", // Green
        "\u001B[33m", // Yellow
        "\u001B[34m", // Blue
        "\u001B[35m", // Magenta
        "\u001B[36m", // Cyan
        "\u001B[90m", // Bright Black
        "\u001B[91m", // Bright Red
        "\u001B[92m", // Bright Green
        "\u001B[93m", // Bright Yellow
        "\u001B[94m", // Bright Blue
        "\u001B[95m", // Bright Magenta
        "\u001B[96m"  // Bright Cyan
    };

    static {
        assignRandomColors();
    }

    private static void assignRandomColors() {
        Random random = new Random();
        for (char c = 'A'; c <= 'Z'; c++) {
            String color;
            do {
```

```
                color = colors[random.nextInt(colors.length)];
            } while (colorMap.containsValue(color));
            colorMap.put(c, color);
        }
    }

    public static String getColor(char c) {
        return colorMap.getOrDefault(c, "\u001B[0m");
    }

    public static String reset() {
        return "\u001B[0m";
    }
}
```

Source Code di file File.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class File {
    public static List<String> readFile(String fileName) {
        List<String> lines = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                lines.add(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
            return null;
        }
        return lines;
    }

    public static void writeFile(String fileName, List<String> lines) {
```

```
        try (FileWriter writer = new FileWriter(fileName)) {
            for (String line : lines) {
                writer.write(line + System.lineSeparator());
            }
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());
        }
    }
}
```

Source Code di Puzzler.java

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

public class Puzzler {
    private int N, M;
    private List<List<String>> blocks;
    private char[][] board;
    private int caseCount;

    public Puzzler(int N, int M, List<List<String>> blocks) {
        this.N = N;
        this.M = M;
        this.blocks = blocks;
        this.board = new char[N][M];
        this.caseCount = 0;
        // Initialize the board with empty spaces
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                board[i][j] = ' ';
            }
        }
    }

    public boolean solve() {
        return solve(0);
    }
```

```
    private boolean solve(int blockIndex) {
        caseCount++;
        if (blockIndex == blocks.size()) {
            return isBoardFull();
        }
        List<String> block = blocks.get(blockIndex);
        Set<List<String>> allOrientations =
getAllUniqueOrientations(block);
        for (List<String> orientation : allOrientations) {
            for (int i = 0; i <= N - orientation.size(); i++) {
                for (int j = 0; j <= M - orientation.get(0).length(); j++)
{
                    if (canPlaceBlock(orientation, i, j)) {
                        placeBlock(orientation, i, j);
                        if (solve(blockIndex + 1)) {
                            return true;
                        }
                        removeBlock(orientation, i, j);
                    }
                }
            }
        }
        return false;
    }

    private Set<List<String>> getAllUniqueOrientations(List<String> block)
{
        Set<List<String>> orientations = new HashSet<>();
        orientations.add(block);
        orientations.add(rotate90(block));
        orientations.add(rotate180(block));
        orientations.add(rotate270(block));
        orientations.add(mirrorHorizontal(block));
        orientations.add(mirrorVertical(block));
        orientations.add(mirrorHorizontal(rotate90(block)));
        orientations.add(mirrorVertical(rotate90(block)));
        orientations.add(mirrorHorizontal(rotate180(block)));
        orientations.add(mirrorVertical(rotate180(block)));
        orientations.add(mirrorHorizontal(rotate270(block)));
        orientations.add(mirrorVertical(rotate270(block)));
```

```
        return orientations;
    }

    private List<String> rotate90(List<String> block) {
        int rows = block.size();
        int cols = block.get(0).length();
        List<String> rotated = new ArrayList<>();
        for (int j = 0; j < cols; j++) {
            StringBuilder newRow = new StringBuilder();
            for (int i = rows - 1; i >= 0; i--) {
                if (j < block.get(i).length()) {
                    newRow.append(block.get(i).charAt(j));
                } else {
                    newRow.append(' ');
                }
            }
            rotated.add(newRow.toString());
        }
        return rotated;
    }

    private List<String> rotate180(List<String> block) {
        return rotate90(rotate90(block));
    }

    private List<String> rotate270(List<String> block) {
        return rotate90(rotate180(block));
    }

    private List<String> mirrorHorizontal(List<String> block) {
        List<String> mirrored = new ArrayList<>();
        for (String row : block) {
            mirrored.add(new StringBuilder(row).reverse().toString());
        }
        return mirrored;
    }

    private List<String> mirrorVertical(List<String> block) {
        List<String> mirrored = new ArrayList<>(block);
        for (int i = 0; i < block.size() / 2; i++) {
```

```
            String temp = mirrored.get(i);
            mirrored.set(i, mirrored.get(block.size() - 1 - i));
            mirrored.set(block.size() - 1 - i, temp);
        }
        return mirrored;
    }

    private boolean canPlaceBlock(List<String> block, int row, int col) {
        for (int i = 0; i < block.size(); i++) {
            for (int j = 0; j < block.get(i).length(); j++) {
                if (block.get(i).charAt(j) != ' ' && (row + i >= N || col
+ j >= M || board[row + i][col + j] != ' ')) {
                    return false;
                }
            }
        }
        return true;
    }

    private void placeBlock(List<String> block, int row, int col) {
        for (int i = 0; i < block.size(); i++) {
            for (int j = 0; j < block.get(i).length(); j++) {
                if (block.get(i).charAt(j) != ' ') {
                    board[row + i][col + j] = block.get(i).charAt(j);
                }
            }
        }
    }

    private void removeBlock(List<String> block, int row, int col) {
        for (int i = 0; i < block.size(); i++) {
            for (int j = 0; j < block.get(i).length(); j++) {
                if (block.get(i).charAt(j) != ' ') {
                    board[row + i][col + j] = ' ';
                }
            }
        }
    }

    private boolean isBoardFull() {
```

```
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }

    public void printBoard() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                char c = board[i][j];
                System.out.print(Color.getColor(c) + c + Color.reset());
            }
            System.out.println();
        }
    }

    public char[][] getBoard() {
        return board;
    }

    public int getCaseCount() {
        return caseCount;
    }
}
```

Source Code di file main.java

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

public class main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Apakah ingin input melalui file test case
berekstensi .txt? (ya/tidak): ");
        String useFileInput = scanner.nextLine().trim().toLowerCase();
```

```
        if (useFileInput.equals("ya")) {
            System.out.print("Masukkan nama file (dengan ekstensi .txt):
");
            String fileName = scanner.nextLine().trim();
            String filePath = "../test/" + fileName;
            List<String> fileInput = File.readFile(filePath);
            if (fileInput == null) {
                System.out.println("Gagal membaca file. Input akan
dilakukan secara manual.");
                useFileInput = "tidak";
            } else {
                processInput(fileInput);
                return;
            }
        }
        if (useFileInput.equals("tidak")) {
            List<String> manualInput = new ArrayList<>();
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine().trim();
                manualInput.add(line);
                if (line.equals("stop")) {
                    break;
                }
            }
            processInput(manualInput);
        } else {
            System.out.println("Input tidak valid. Program akan
berhenti.");
        }

        scanner.close();
    }

    private static void processInput(List<String> input) {
        Scanner scanner = new Scanner(String.join("\n", input));
        int N = 0, M = 0, P = 0;
        while (true) {
            String line = scanner.nextLine().trim();
            String[] parts = line.split(" ");
            if (parts.length == 3) {
```

```
                try {
                    N = Integer.parseInt(parts[0]);
                    M = Integer.parseInt(parts[1]);
                    P = Integer.parseInt(parts[2]);
                    break;
                } catch (NumberFormatException e) {
                    System.out.println("Input nilai N, M, atau P tidak
valid. Mohon diinput ulang.");
                }
            } else {
                System.out.println("Input nilai N, M, atau P tidak valid.
Mohon diinput ulang.");
            }
        }
        while (true) {
            String S = scanner.nextLine().trim();
            if (S.equals("DEFAULT")) {
                break;
            } else {
                System.out.println("Input tidak valid! Hanya terdapat
pilihan DEFAULT.");
            }
        }

        List<List<String>> listBlok = new ArrayList<>();
        List<String> currBlok = new ArrayList<>();
        char prevChar = ' ';
        int countBlok = 0;
        boolean validInput = true;
        boolean stopMarkerFound = false;
        while (countBlok < P) {
            if (!scanner.hasNextLine()) {
                System.out.println("Jumlah blok tidak sesuai. Mohon
diinput ulang.");
                validInput = false;
                break;
            }
            String inputLine = scanner.nextLine().trim();
            if (inputLine.equals("stop")) {
                stopMarkerFound = true;
```

```
                break;
            }
            if (inputLine.isEmpty()) {
                System.out.println("Jumlah blok tidak sesuai. Mohon
diinput ulang.");
                validInput = false;
                break;
            }
            StringBuilder cleanedInput = new StringBuilder();
            for (char c : inputLine.toCharArray()) {
                if (Character.isUpperCase(c)) {
                    cleanedInput.append(c);
                } else {
                    cleanedInput.append(' '); // Semua input pada blok
selain huruf kapital dianggap spasi (' ')
                }
            }
            char currChar = '\0';
            for (char c : cleanedInput.toString().toCharArray()) {
                if (Character.isUpperCase(c)) {
                    currChar = c;
                    break;
                }
            }
            if (currChar == '\0') {
                System.out.println("Blok tidak valid! Mohon diinput
ulang.");
                validInput = false;
                break;
            }
            if (currBlok.isEmpty() || currChar != prevChar) {
                if (!currBlok.isEmpty()) {
                    if (!isBlokValid(currBlok)) {
                        System.out.println("Blok tidak valid! Mohon
diinput ulang.");
                        validInput = false;
                        break;
                    }
                    listBlok.add(new ArrayList<>(currBlok));
                    countBlok++;
```

```
                    if (countBlok == P) {
                        break;
                    }
                }
                currBlok.clear();
            }
            currBlok.add(cleanedInput.toString());
            prevChar = currChar;
        }
        if (!validInput || !stopMarkerFound) {
            System.out.println("Jumlah blok kelebihan! Mohon diinput
ulang.");
            return;
        }
        if (!currBlok.isEmpty() && countBlok < P) {
            if (!isBlokValid(currBlok)) {
                System.out.println("Blok tidak valid! Mohon diinput
ulang.");
                return;
            }
            listBlok.add(new ArrayList<>(currBlok));
            countBlok++;
        }

        Puzzler puzzler = new Puzzler(N, M, listBlok);
        long startTime = System.currentTimeMillis();
        boolean solved = puzzler.solve();
        long endTime = System.currentTimeMillis();
        if (solved) {
            System.out.println("Papan berhasil diisi penuh.");
            puzzler.printBoard();
        } else {
            System.out.println("Tidak ada solusi.");
        }
        System.out.println("Waktu pencarian: " + (endTime - startTime) + "
ms");
        System.out.println("Banyak kasus yang ditinjau: " +
puzzler.getCaseCount());
        System.out.print("Apakah anda ingin menyimpan solusi? (ya/tidak):
");
```

```
        String saveSolution = scanner.nextLine().trim().toLowerCase();
        if (saveSolution.equals("ya")) {
            System.out.print("Masukkan nama file untuk menyimpan solusi
(dengan ekstensi .txt): ");
            String outputFileName = scanner.nextLine().trim();
            List<String> outputLines = new ArrayList<>();
            outputLines.add("Papan berhasil diisi dengan benar.");
            for (int i = 0; i < N; i++) {
                StringBuilder line = new StringBuilder();
                for (int j = 0; j < M; j++) {
                    line.append(puzzler.getBoard()[i][j]);
                }
                outputLines.add(line.toString());
            }
            outputLines.add("Waktu pencarian: " + (endTime - startTime) +
" ms");
            outputLines.add("Banyak kasus yang ditinjau: " +
puzzler.getCaseCount());
            File.writeFile(outputFileName, outputLines);
            System.out.println("Solusi telah disimpan ke " +
outputFileName);
        }

        scanner.close();
    }

    public static boolean isBlokValid(List<String> blok) {
        int rows = blok.size();
        int cols = blok.get(0).length();

        for (int i = 0; i < rows; i++) {
            String row = blok.get(i);
            for (int j = 0; j < row.length(); j++) {
                char currChar = row.charAt(j);
                if (!Character.isUpperCase(currChar)) continue;

                boolean bersisian = false;
                if (i > 0 && j < blok.get(i - 1).length() && blok.get(i -
1).charAt(j) == currChar)
                    bersisian = true; // mengecek bagian atas huruf
```

```
                if (i < rows - 1 && j < blok.get(i + 1).length() &&
blok.get(i + 1).charAt(j) == currChar)
                    bersisian = true; // mengecek bagian bawah huruf
                if (j > 0 && row.charAt(j - 1) == currChar)
                    bersisian = true; // mengecek bagian kiri huruf
                if (j < row.length() - 1 && row.charAt(j + 1) == currChar)
                    bersisian = true; // mengecek bagian kanan huruf
                if (!bersisian) return false;
            }
        }
        return true;
    }
}
```

Fungsi pengecekan blok apakah valid atau tidak saya pakai agar memastikan bahwa masukan blok yang terdiri dari huruf tersebut dapat saling terhubung dan juga pada fungsi sebelumnya untuk hanya menyimpan blok yang sudah valid terhubung dan hanya menyimpan huruf kapital sehingga tidak menyimpan simbol lain

3. Tangkapan Layar Input dan Output

Tampilan input dengan file berekstensi .txt yaitu,

```
Apakah ingin input melalui file test case berekstensi .txt? (ya/tidak): ya
Masukkan nama file (dengan ekstensi .txt): tc1.txt
Papan berhasil diisi penuh.
```

Program tidak berhasil menunjukkan papan berisi blok yang disebutkan sudah ditemukan solusinya.

Tampilan input secara manual yaitu,

```
Apakah ingin input melalui file test case berekstensi .txt? (ya/tidak): tidak
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
stop
Papan berhasil diisi penuh.
```

Program dengan input manual juga tidak berhasil menampilkan papan yang berisikan blok.

4. Tautan Repository

https://github.com/4clarissaNT4/Tucil1_13523016

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	✓	
5	Program menyimpan solusi dalam berkas .txt		✓
6	Program memiliki *Graphical User Interface* (GUI)		✓
7	Program dapat menyimpan solusi dalam bentuk file gambar		✓
8	Program dapat menyelesaikan kasus konfigurasi *custom*		✓
9	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
10	Program dibuat oleh saya sendiri	✓	