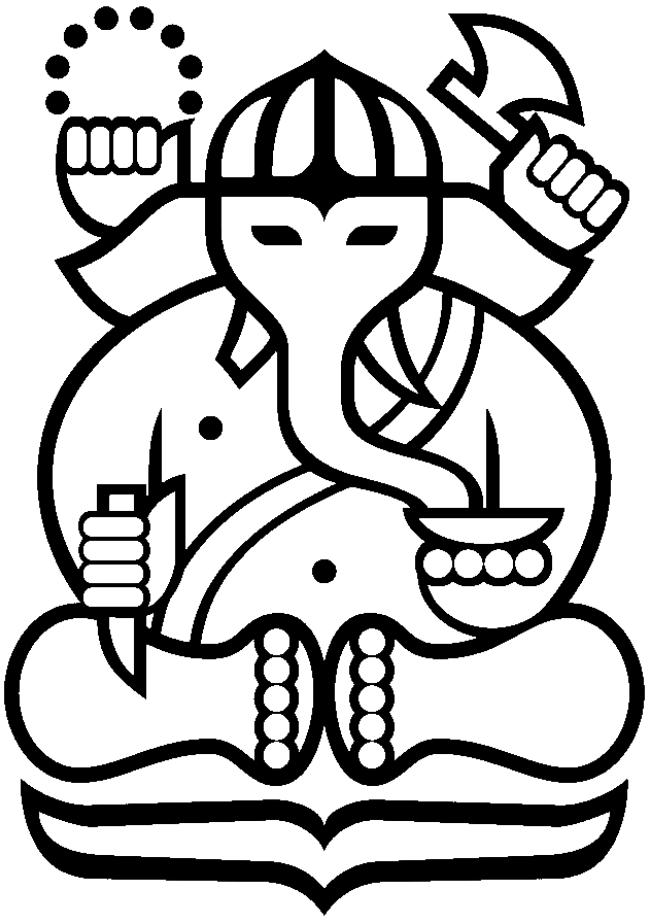
LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

PENYELESAIAN IQ PUZZLER PRO SOLVER DENGAN ALGORITMA BRUTE FORCE



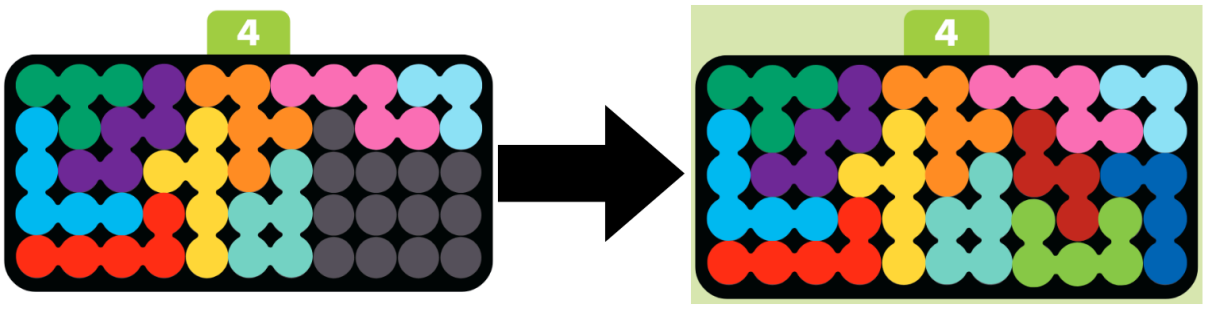
Disusun oleh:
Nathan Jovial Hartono - 13523032

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

2025

I. ALGORITMA

Iq Puzzler Pro merupakan permainan fisik problem-solving dimana pemain diberikan susunan sejumlah potongan beragam bentuk, seperti T, L, ataupun n, untuk pemain mencari solusi dimana board terisi penuh dengan kondisi semua piece diletakkan pada board.



Pada konteks permasalahan tugas kecil, kondisi awal papan main selalu kosong dengan ukuran n x m dimensi dan sejumlah potongan dengan posisi dan ukurannya masing - masing ditentukan oleh user. Solusi ditemukan jika seluruh potongan dapat dimuatkan pada papan main dan papan main terisi penuh tanpa sisa. Setiap akhir proses pencarian akan dihitung iterasi penempatan balok serta waktu operasi. Proses pencarian satu kemungkinan solusi bisa memanfaatkan bruteforce dan menggunakan pendekatan rekursif seperti backtracking.

Bruteforce merupakan algoritma yang mengecek setiap kasus pada suatu permasalahan dengan mengorbankan *time-complexity* tetapi selalu menjamin sebuah hasil yang benar jika tertulis dengan benar. Walaupun selalu mengecek setiap kasus, bruteforce juga dapat dioptimasi dengan pendekatan yang lebih tepat pada permasalahan, salah satunya pendekatan backtracking pada konteks iqpuzzlerpro ini. Backtracking merupakan solusi algoritma yang membangun rantai kasus dan menghapus instansi kasus pada rantai kasus apabila ditentukan tidak cocok. Pendekatan ini cocok untuk konteks permasalahan ini karena penempatan potongan sangat bergantung pada penempatan potongan sebelumnya.

Berikut penjelasan general algoritma yang diterapkan untuk konteks permasalahan ini:

1. Inisiasi variabel *startTime* dan *iter* untuk memproses iterasi dan waktu yang diperlukan. Proses rekursif akan menggunakan fungsi solve dengan parameter *block* collection of *block2D* yang merupakan array dua dimensi, berisi seluruh orientasi potongan yang mungkin, boolean array *used,* dan integer *blockCount.* Parameter *block* digunakan untuk memilih dan mencari piece yang cocok saat bruteforce. Parameter *used* digunakan untuk pengecekan apakah piece telah digunakan, sebuah piece tidak dapat digunakan dua kali. Parameter *blockCount* menghitung jumlah piece yang telah ada di board.

```java
// Solve the board with collection of block with defined orientations
public boolean betterSolve(Block2D[][] block, boolean[] used, int blockCount)
```

2. Proses pencarian akan dimulai dari iterasi papan main pada baris dan kolom ke-0 (0,0) dan dari situ akan dicek apakah potongan tersebut muat pada board. Potongan yang diuji akan diberikan delapan orientasi berbeda sebagai berikut: input/original, flip secara horizontal pada original, rotasi counterclockwise 90° original, flip pada rotasi counterclockwise 90° original, rotasi counterclockwise 180° original, flip pada rotasi counterclockwise 180° original, rotasi counterclockwise 270° original, dan terakhir flip pada rotasi counterclockwise 270° original. Penempatan potongan akan selalu diawali dengan potongan dengan orientasi original. Apabila potongan dengan orientasi tersebut tidak bisa dimuatkan, maka akan menggunakan orientasi selanjutnya.
3. Rekursi bisa return true atau false. Apabila rekursi mengembalikan false maka program akan backtrack dan mencoba semua orientasi dari block tersebut terlebih dahulu dan setelah semua orientasi dicoba akan mencoba menggantikan potongan sebelumnya yang telah terdefinisi pada papan main dengan potongan yang valid.
4. Terdapat beberapa kondisi basis untuk konteks ini. Kondisi basis mengembalikan True saat papan main sudah terisi penuh dan semua potongan telah digunakan. Selain kondisi tersebut, akan mengembalikan False (contoh: kondisi papan main penuh tapi tidak semua potongan digunakan, kondisi papan main tidak penuh tapi semua potongan telah digunakan, dan juga kondisi papan tidak memungkinkan pemuatan potongan tetapi belum terisi penuh)
5. Algoritma awalnya menggunakan angka sebagai representasi sebuah potongan dan disimpan korelasi dengan karakternya pada sebuah array untuk nanti dikeluarkan pada output.txt sebagai character sesuai input.

II. SOURCE CODE

Github Repository
https://github.com/19623248Git/Tucil1_13523032

Block2D.java

Block2D.java merupakan file class untuk objek yang menyimpan data potongan pada input file, seperti data matriks potongan dan juga ukurannya.

Method	Detail
`public void rotateCTR()`	Block2D diberikan rotasi counterclockwise 90 derajat
`public void flipH()`	Block2D di-flip secara horizontal
`public Block2D[] getOrientations()`	Semua orientasi Block2D dicari dan dikeluarkan pada sebuah collection of blocks

```
package iqpuzzlerpro;
import java.util.*;


public class Block2D {


   private List<List<Integer>> block;
   private int n_row;
   private int n_col;


   // Default constructor
   public Block2D(){
      this.block = null; // init with empty list
   }


   // Constructor with a given block
   public Block2D(List<List<Integer>> block){
```

```
        this.block = block;
    }


    // Setter
    public void setBlock2D(List<List<Integer>> block){
        this.block = transformBlock2D(block);
    }


    // Transform
    public List<List<Integer>> transformBlock2D(List<List<Integer>> block){


        // Get row size
        this.n_row = block.size();


        // Define n_col INT_MIN first
        this.n_col = Integer.MIN_VALUE;


        for (int i = 0; i < n_row; i++){
              int n = block.get(i).size();
              if (this.n_col < n){
              this.n_col = n;
              }
        }


        List<List<Integer>> buffer2D = new ArrayList<>();


        // define the matrix of 0s first
        for(int i = 0; i < this.n_row; i++){
              buffer2D.add(new ArrayList<>());
              for (int j = 0; j < this.n_col; j++){
              buffer2D.get(i).add(0);
              }
        }


        // re-define the matrix data type
```

```
    for(int i = 0; i < block.size(); i++){
          for(int j = 0; j < block.get(i).size(); j++){
          if(block.get(i).get(j) != 0){
                buffer2D.get(i).set(j,block.get(i).get(j));
          }
          }
    }


    return buffer2D;
}


// Set Block Size
public void setSize(int row, int col){
    this.n_row = row;
    this.n_col = col;
}


// Getter
public List<List<Integer>> getBlock2D(){
    return this.block;
}


// Get Block Size
public int getRow(){
    return this.n_row;
}


public int getCol(){
    return this.n_col;
}


// Print-er
public void printBlock2D(){
    System.out.println(this.block);
}

// Rotate block counterclockwise
```

```java
public void rotateCTR(){
    // mat[i][j] to mat[n-j-1][i]
    // n is this.n_col
    List<List<Integer>> copyBlock = new ArrayList<>();


    // Initialize empty ArrayList
    for(int i = 0; i < this.n_col; i++){
          copyBlock.add(new ArrayList<>());
          for(int j = 0; j < this.n_row;j++){
          copyBlock.get(i).add(j,0);
          }
    }


    // Rotate the block
    for(int i = 0; i < this.n_row; i++){
          for(int j = 0; j < this.n_col; j++){
          copyBlock.get(this.n_col-j-1).set(i,this.block.get(i).get(j));
          }
    }
    this.block = copyBlock;


    // swap row and col
    int swap = this.n_col;
    this.n_col = this.n_row;
    this.n_row = swap;


}


// Flip block
// Flip on the y-axis
public void flipH(){


    List<List<Integer>> copyBlock = new ArrayList<>();


    for (List<Integer> row : this.block) {
          copyBlock.add(new ArrayList<>(row));
```

```
      }


      for (int i = 0; i < this.n_row; i++){
            for (int j = 0; j < this.n_col/2; j++){
            copyBlock.get(i).set(j, this.block.get(i).get(this.n_col - 1 -
j));
            copyBlock.get(i).set(this.n_col - 1 - j,
this.block.get(i).get(j));
            }
      }


      this.block = copyBlock;

   }


   public void flipV(){


      List<List<Integer>> copyBlock = new ArrayList<>();


      for (List<Integer> row : this.block) {
            copyBlock.add(new ArrayList<>(row));
      }


      for (int i = 0; i < this.n_row/2; i++){
            List<Integer> temp = copyBlock.get(i);
            copyBlock.set(i, copyBlock.get(this.n_row - 1 - i));
            copyBlock.set(this.n_row - 1 - i, temp);
      }


      this.block = copyBlock;


   }


   public Block2D[] getOrientations(){
```

```
    Block2D[] block_orientations = new Block2D[8];

    for (int k = 0; k < 8; k++) {
          block_orientations[k] = new Block2D();
    }


    for (int k = 0; k < 8; k+=2) {
          block_orientations[k].setBlock2D(getBlock2D());
          Block2D temp = new Block2D();
          temp.setBlock2D(getBlock2D());
          temp.flipH();
          block_orientations[k+1].setBlock2D(temp.getBlock2D());
          rotateCTR();
    }


    return block_orientations;
  }
```

Board2D.java

Board2D.java merupakan file class untuk objek yang menyimpan data papan main dengan menyimpan atribut seperti ukuran width dan height serta waktu dan iterasi untuk menemukan solusi.

Method	Detail
`public void printCharCorr(char[] char_int_corr)`	Board2D menyimpan data berupa integer yang merepresentasikan pieces, fungsi ini untuk mengeluarkan karakter yang berhubungan dengan integer ini
`public void outFile(char[] char_int_corr)`	Mengeluarkan data atribut Board2D dalam file output.txt
`public boolean isFit(Block2D block,`	Board2D cek apakah Block2D bisa dimuatkan

`int pivotRow, int pivotCol)`	dalam atribut matriks Board2D
`public void placeBlock(Block2D block, int pivotRow, int pivotCol)`	Block2D dimuatkan dalam atribut matriks Board2D
`public void removeBlock(Block2D block, int pivotRow, int pivotCol)`	Block2D dilepaskan dari atribut matriks Board2D
`public void findCorner()`	Board2D mencari pojok matriks yang kosong
`public boolean isFull()`	Board2D cek apabila atribut matriks sudah terisi semua
`public boolean betterSolve(Block2D[][] block, boolean[] used, int blockCount)`	Solver untuk Board2D dengan collection of Block2D beserta orientasinya

```java
package iqpuzzlerpro;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class Board2D {

	// Define board as a contiguous linear array of width * height
	// Reflects the true definition of list in memory
	private List<List<Integer>> board;
	private int width;
	private int height;
	private int pivotRow = 0;
	private int pivotCol = 0;
	private long iter = 0;
	private String mode;
	private double  elapsedTime = 0;
	private String outPath;

	// Default constructor
	public Board2D(){
	this.board = null;
	}
```

```java
// Constructor with a given board
public Board2D(List<List<Integer>> board){
this.board = board;
}

/*
Setter with defined board
DO NOT USE THIS!! NOT SUPPORTED
*/
public void setBoard2D(List<List<Integer>> board){
this.board = board;
}

// Setter with defined board size
public void setBoard2D(int width, int height){

if (width <= 0 || height <= 0){
      throw new IllegalArgumentException(
            "Width and height must be positive"
      );
}

this.width = width;
this.height = height;

this.board = new ArrayList<>(height);
for(int i = 0; i < height; i++){
      this.board.add(new ArrayList<>());
      for(int j = 0; j < width; j++){
            this.board.get(i).add(0);
      }
}

}

// Set Board2D element at certain 2D index
public void setElmt(int row, int col, int value){

if (board == null){
      throw new IllegalArgumentException(
            "Board2D is not initialized"
      );
}
```

```
if (row < 0 || row >= height || col < 0 || col >= width){
      throw new IllegalArgumentException(
            "row or column idx is out of bounds"
      );
}

this.board.get(row).set(col,value);

}

// Set Board2D height
public void setHeight(int value){
this.height = value;
}

// Set Board2D width
public void setWidth(int value){
this.width = value;
}

// Getter
public List<List<Integer>> getBoard2D(){
return this.board;
}

public long getIter(){
return this.iter;
}

// Get Board2D element at certain 2D index
public int getElmt(int row, int col){

if (board == null){
      throw new IllegalArgumentException(
            "Board2D is not initialized"
      );
}

if (row < 0 || row >= height || col < 0 || col >= width){
      throw new IllegalArgumentException(
            "row or column idx is out of bounds"
      );
```

```
	}

	return this.board.get(row).get(col);

	}

	// Get Board2D height
	public int getHeight(){
	return(this.height);
	}

	// Get Board2D width
	public int getWidth(){
	return(this.width);
	}

	// Fit attributes
	public void fit(int[] nmp, String mode){

	setBoard2D(nmp[1], nmp[0]);

	if(mode.toLowerCase().equals("custom") ||
mode.toLowerCase().equals("custom")){
		System.out.println("Mode has not been implemented, converting
to default mode");
	}
	else if(!mode.toLowerCase().equals("default")){
		System.out.println("Mode type not found, converting to default
mode!");
	}
	mode = "default";
	this.mode = mode;
	}

	// Print-er
	public void printBoard2D(){
	System.out.println(this.board);
	}

	public void prettierPrintBoard2D(){
	System.out.println("--------------------------------");
	for(int i = 0; i < this.height; i++){
		System.out.println(board.get(i));
```

```
	}
	System.out.println("---------------------------------");
	}

	public void printCharCorr(char[] char_int_corr) {
	String[] colors = {
		"\u001B[38;5;196m", // Red
		"\u001B[38;5;220m", // Yellow
		"\u001B[38;5;51m",  // Cyan
		"\u001B[38;5;214m", // Gold
		"\u001B[38;5;57m",  // Dark Purple
		"\u001B[38;5;46m",  // Green
		"\u001B[38;5;201m", // Pink
		"\u001B[38;5;39m",  // Light Blue
		"\u001B[38;5;172m", // Dark Orange
		"\u001B[38;5;27m",  // Deep Blue
		"\u001B[38;5;129m", // Magenta
		"\u001B[38;5;178m", // Light Yellow
		"\u001B[38;5;88m",  // Brownish Red
		"\u001B[38;5;207m", // Hot Pink
		"\u001B[38;5;44m",  // Deep Turquoise
		"\u001B[38;5;136m", // Olive
		"\u001B[38;5;93m",  // Purple
		"\u001B[38;5;83m",  // Teal
		"\u001B[38;5;160m", // Dark Red
		"\u001B[38;5;94m",  // Dark Gold
		"\u001B[38;5;219m", // Light Pink
		"\u001B[38;5;140m", // Lavender
		"\u001B[38;5;202m", // Orange
		"\u001B[38;5;165m", // Violet
		"\u001B[38;5;21m",  // Blue
		"\u001B[38;5;255m"  // White
	};

	String reset = "\u001B[0m"; // Reset color

	for (int i = 0; i < this.height; i++) {
		for (int j = 0; j < this.width; j++) {
			int boardValue = board.get(i).get(j) - 1;
			String color = colors[boardValue % colors.length]; //
Cycle through 26 colors
			System.out.print(color + char_int_corr[boardValue] +
reset);
```

```
            }
            System.out.println();
      }
      }

      // File output
      public void outFile(char[] char_int_corr){
      try (BufferedWriter writer = new BufferedWriter(new
FileWriter(this.outPath))) {
            for (int i = 0; i < height; i++) {
                  for (int j = 0; j < width; j++) {
                  int boardValue = board.get(i).get(j) - 1;
                  writer.write(char_int_corr[boardValue]); // Write
character without color codes
                  }
                  writer.newLine();
            }
            writer.newLine();
            writer.write("Number of iterations: " + getIter());
            writer.newLine();
            writer.write("Elapsed time: " + this.elapsedTime + " seconds");
            System.out.println("Output file has been created!");
      }
      catch (IOException e) {
            System.err.println("Error creating the file!");
            // e.printStackTrace();
      }
      }


      // Check if block is fit in board
      public boolean isFit(Block2D block, int pivotRow, int pivotCol){

      // block.printBlock2D();

      if (pivotRow < 0 || pivotCol < 0 || pivotRow + block.getRow() >
getHeight() || pivotCol + block.getCol() > getWidth()) {
            // System.out.println("( " + (pivotRow) + " , " + (pivotCol) +
" )");
            // System.out.println("fail #1");
            return false;
      }
```

```
      for (int i = 0; i < block.getRow(); i++){
            for (int j = 0; j < block.getCol(); j++){
                  if(i+pivotRow >= getHeight() || j+pivotCol >= getWidth()
|| i+pivotRow < 0 || j+pivotCol < 0){
                  // System.out.println("( " + (i+pivotRow) + " , " +
(j+pivotCol) + " )");
                  // System.out.println("fail #2");
                  return false;
                  }
                  if(this.board.get(i+pivotRow).get(j+pivotCol)!=0 &&
block.getBlock2D().get(i).get(j)!=0){
                  // System.out.println("( " + (i+pivotRow) + " , " +
(j+pivotCol) + " )");
                  // System.out.println("fail #3");
                  return false;
                  }
            }
      }
      return true;
      }

      // Place block in board
      public void placeBlock(Block2D block, int pivotRow, int pivotCol){
      for (int i = 0; i < block.getRow(); i++){
            for (int j = 0; j < block.getCol(); j++){
                  int element = block.getBlock2D().get(i).get(j);
                  if (element!=0){
                  this.board.get(i+pivotRow).set(j+pivotCol,element);
                  }
            }
      }
      }

      // Remove block in board
      public void removeBlock(Block2D block, int pivotRow, int pivotCol){
      for (int i = 0; i < block.getRow(); i++){
            for (int j = 0; j < block.getCol(); j++){
                  if(block.getBlock2D().get(i).get(j)!=0){
                  this.board.get(i+pivotRow).set(j+pivotCol,0);
                  }
            }
      }
      }
```

```
	// Find Top Left Empty Corner
	public void findCorner(){
	for(int i = 0; i <  this.height; i++){
		for(int j = 0; j < this.width; j++){
			// this.iter+=1;
			if(this.board.get(i).get(j) == 0){
			this.pivotRow = i;
			this.pivotCol = j;
			return;
			}
		}
	}
	}


	// Check if board is full
	public boolean isFull(){
	for(int j = 0; j <  this.width; j++){
		for(int i = 0; i < this.height; i++){
			if(this.board.get(i).get(j) == 0){
			return false;
			}
		}
	}
	return true;
	}

	// Solve the board with collection of block with defined orientations
	public boolean betterSolve(Block2D[][] block, boolean[] used, int
blockCount) {

	// Solved condition
	if (blockCount == block.length && isFull()) {
		return true;
	}

	if (blockCount > block.length) {
		return false;
	}

	findCorner();
```

```
	int pR = this.pivotRow;
	int pC = this.pivotCol;

	for (int i = pR; i < this.height; i++) {
		for (int j = pC; j < this.width; j++) {
			for(int k = 0; k < block.length; k++){
			for(int p = 0; p < block[0].length; p++){

				if(used[k]) continue;

				this.iter += 1;

				if (isFit(block[k][p], i, j)) {

					placeBlock(block[k][p], i, j);
					used[k] = true;

					if (betterSolve(block, used, blockCount + 1))
{
					return true;
					}

					removeBlock(block[k][p], i, j);
					used[k] = false;
				}
			}
			}
		}
		if(pC!=0){
			pC = 0;
		}
	}

	return false;
	}

	// fit and solve
	public void fitSolve(int[] nmp, String mode, Block2D[] blocks, char[]
char_int_corr, String path){

	this.outPath = path + "/output.txt";

	fit(nmp, mode);
```

```
	boolean used[] = new boolean[blocks.length];
	Block2D[][] block_w_orients = new Block2D[blocks.length][8];

	for(int i = 0; i < blocks.length; i++){
		used[i] = false;
		block_w_orients[i] = blocks[i].getOrientations();
	}

	long startTime = System.nanoTime();
	long endTime;

	// depecrated mode
	// if(solve(blocks, used, 0))

	if(betterSolve(block_w_orients, used, pivotCol)){
		endTime = System.nanoTime();
		this.elapsedTime = (endTime - startTime) / 1_000_000_000.0;
		System.out.println("Solution found!");
		printCharCorr(char_int_corr);
		System.out.println("Number of iterations: " + getIter());
		System.out.println("Elapsed time: " + this.elapsedTime + "
seconds");
		outFile(char_int_corr);
	}
	else{
		System.out.println("Solution not found :(");
		endTime = System.nanoTime();
		this.elapsedTime = (endTime - startTime) / 1_000_000_000.0;
		System.out.println("Elapsed time: " + this.elapsedTime + "
seconds");
	}
	}


}
```

FileHandling.java

FileHandling.java merupakan file class untuk objek yang membaca dan menyimpan data dari input.txt.

Method	Detail
`public void inputPath()`	Input path untuk file sesuai konvensi dari tugas
`public void handle()`	Filehandling setelah diberikan path file

```
package iqpuzzlerpro;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class FileHandling {

	private String path;
	private int[] nmp;
	private String mode;
	private Block2D[] blocks;
	private char[] char_int_corr;

	// debug printing for file handling
	public void debugPrint(){
	System.out.println("");
	System.out.println("n:_" + this.nmp[0] + "_");
	System.out.println("m:_" + this.nmp[1] + "_");
	System.out.println("p:_" + this.nmp[2] + "_");
	System.out.println("mode:_" + this.mode + "_");
	for (int i = 0; i  < this.nmp[2]; i++){
		System.out.println("int: " + (i+1) + " char: " +
this.char_int_corr[i]);
		this.blocks[i].printBlock2D();
	}
	}
```

```
      public void inputPath(){
      String currentDir = System.getProperty("user.dir");
      System.out.println("\nCurrent directory: " + currentDir);
      String expectedDir = currentDir + "/test/[dir_name]/input.txt";
      System.out.println("\nExpected directory: " + expectedDir);

      try (Scanner myObj = new Scanner(System.in)) {
            System.out.print("\nInput dir_name: ");
            String dir_name = myObj.nextLine();
            this.path = currentDir + "/test/" + dir_name;
            File f = new File(this.path);
            if (f.exists()){
                  System.out.println("\nInput Directory Exists!");
            }
            else{
                  System.out.println("\n"  +  this.path + "Doesn't
Directory Exists!");
            }
      }
      }

      // Get path
      public String getPath(){
      return this.path;
      }

      // file handling
      public void handle() {
      try (BufferedReader reader = new BufferedReader(new
FileReader(Paths.get(this.path, "input.txt").toString()))) {
            // Variables
            this.nmp = new int[3];
            int nmp_itr = 0;
            boolean nmp_read = false;

            this.mode = "";
            boolean mode_read = false;

            List<List<Integer>> buffer2D = null;
            this.blocks = null;
            char unique = (char) (1);
            int[] yx = {0, -1};
```

```
            this.char_int_corr = null;
            int idx_cic = 0;
            int indent = 0;

            int data;
            while ((data = reader.read()) != -1) {
                  if (data == '\r') continue; // Skip carriage return in
Windows

                  if (!nmp_read) {
                  if (Character.isDigit(data)) {
                        while (data != ' ' && data != '\n' && data != -1) {
                              this.nmp[nmp_itr] = this.nmp[nmp_itr] * 10 +
(data - '0');
                              data = reader.read();
                              if (data == '\r') data = reader.read(); //
Handle Windows line endings
                        }
                        nmp_itr++;
                        if (nmp_itr != 3 && data == '\n') {
                              System.err.println("Only three numbers at the
beginning of the file");
                              return;
                        }
                  }
                  if (nmp_itr == 3) {
                        nmp_read = true;
                        this.char_int_corr = new char[this.nmp[2] == 0 ? 1
: this.nmp[2]];
                        this.blocks = new Block2D[this.nmp[2]];
                        for (int i = 0; i < this.nmp[2]; i++) {
                              this.blocks[i] = new Block2D();
                        }
                  }
                  } else if (!mode_read) {
                  if (data != '\n') {
                        this.mode += (char) data;
                  } else {
                        mode_read = true;
                  }
                  } else {
                  if ((char) data != unique && data != '\n' && data != ' ')
{
```

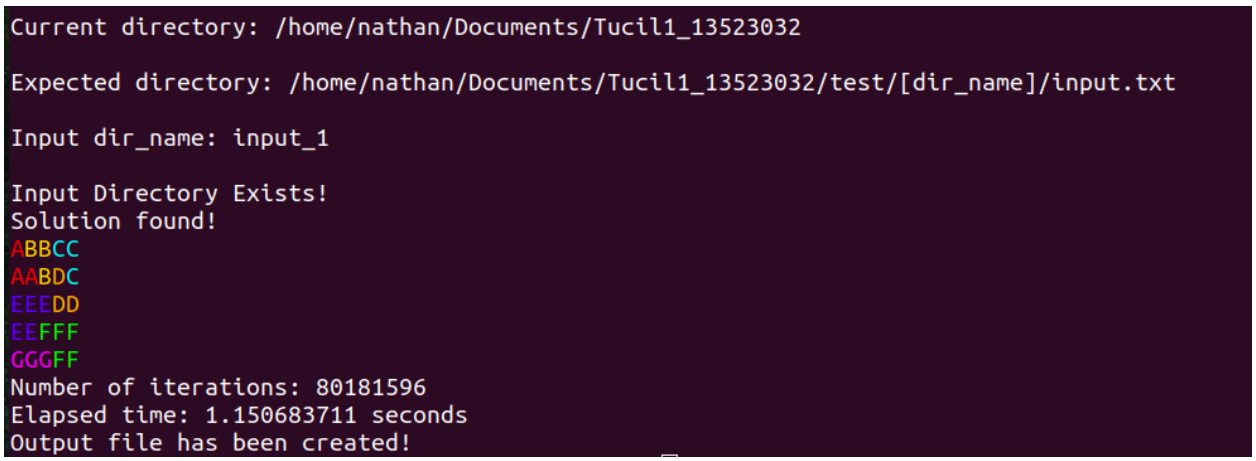
```
                        unique = (char) data;
                        if (buffer2D != null) {
                              buffer2D.remove(yx[0]);
                              this.blocks[idx_cic -
1].setBlock2D(buffer2D);
                        }
                        this.char_int_corr[idx_cic++] = unique;
                        yx[0] = 0;
                        yx[1] = -1;
                        buffer2D = new ArrayList<>();
                        buffer2D.add(new ArrayList<>());
                        for (int i = 0; i < indent; i++) {
                              buffer2D.get(yx[0]).add(0);
                        }
                        indent = 0;
                  }
                  if ((char) data == unique) {
                        yx[1]++;
                        for (int i = 0; i < indent; i++) {
                              buffer2D.get(yx[0]).add(0);
                        }
                        indent = 0;
                        buffer2D.get(yx[0]).add(idx_cic);
                  } else if (data == '\n') {
                        yx[0]++;
                        yx[1] = -1;
                        indent = 0;
                        buffer2D.add(new ArrayList<>());
                  } else if (data == ' ') {
                        yx[1]++;
                        indent++;
                  }
                  }
            }
            this.blocks[idx_cic - 1].setBlock2D(buffer2D);
            // debugPrint();
      } catch (IOException e) {
            System.err.println("input file not found in directory: " +
this.path);
      }
      }

      // Getter
```

```
        public int[] getNmp(){
        return this.nmp;
        }

        public String getMode(){
        return this.mode;
        }

        public Block2D[] getBlocks(){
        return this.blocks;
        }

        public char[] getCharIntCorr(){
        return this.char_int_corr;
        }

        // Setter
        public void setNmp(int[] nmp){
        this.nmp = nmp;
        }

        public void setMode(String mode){
        this.mode = mode;
        }

        public void setBlocks(Block2D[] blocks){
        this.blocks = blocks;
        }

        public void setCharIntCorr(char[] char_int_corr){
        this.char_int_corr = char_int_corr;
        }
}
```

III. EKSPERIMEN

Test Case #1:

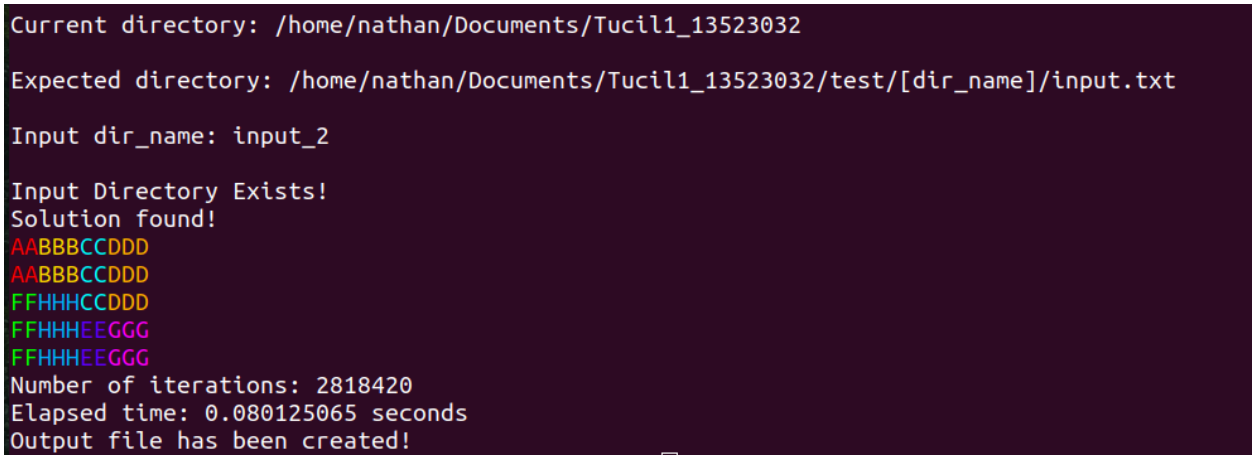
input.txt	output.txt
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	ABBCC AABDC EEEDD EEFFF GGGFF Number of iterations: 80181596 Elapsed time: 1.134567297 seconds



Test Case #2:

input.txt	output.txt
5 10 8	AABBBCCDDD

DEFAULT AA AA BBB BBB CC CC CC DDD DDD DDD EE EE FFF FFF GG GG GG HHH HHH HHH	AABBBCCDDD FFHHHCCDDD FFHHHEEGGG FFHHHEEGGG Number of iterations: 2818420 Elapsed time: 0.080250685 seconds



Test Case #3:

input.txt	output.txt
4 6 6 DEFAULT A	ABEEEC ABBBEC AADFCC

A AA B B BB C C CC D D DD E E EE F F FF	DDDFFF Number of iterations: 1199622 Elapsed time: 0.03122176 seconds

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_3

Input Directory Exists!
Solution found!
ABEEEC
ABBBEC
AADFCC
DDDFFF
Number of iterations: 1199622
Elapsed time: 0.03046016 seconds
Output file has been created!
```

Test Case #4:

input.txt	output.txt
8 8 12 DEFAULT AA A EEEE BB B	AAEEEEBB AIIIJJJB HIIIJJJF HIIIJJJF HKKKLLLF HKKKLLLF DKKKLLLC

III III III JJJ JJJ JJJ H H H H F F F F KKK KKK KKK LLL LLL LLL C CC GGGG D DD	DDGGGGCC Number of iterations: 4198 Elapsed time: 0.001571982 seconds

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_5

Input Directory Exists!
Solution found!
AAEEEEBB
AIIIJJJB
HIIIJJJF
HIIIJJJF
HKKKLLLF
HKKKLLLF
DKKKLLLC
DDGGGGCC
Number of iterations: 4198
Elapsed time: 0.001509563 seconds
Output file has been created!
```

Test Case #5:

input.txt	output.txt
4 4 5 DEFAULT AAAAA BBBBB CCCCC DDDDD EEEEE	-

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_6

Input Directory Exists!
Solution not found :(
Elapsed time: 1.4511E-4 seconds
```

Test Case #6:

input.txt	output.txt
6 6 9 DEFAULT A AAA B BBB CC CC DD DD E E EEE F FF FF G GG G G HHH	ABBBCC AAABCC DDFFFI HDDFFE HHGGGE HGGEEE Number of iterations: 936677314 Elapsed time: 12.46992077 seconds

H I	

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_7

Input Directory Exists!
Solution found!
ABBBCC
AAABCC
DDFFFI
HDDFFE
HHGGGE
HGGEEE
Number of iterations: 936677314
Elapsed time: 12.388498339 seconds
Output file has been created!
```

Test Case #7:

input.txt	output.txt
4 8 6 DEFAULT E EE EEE EEEE A AAA B BBB C CCC D DDD F FF FFF	EFFFADDD EEFFAADB EEEFACBB EEEECCCB Number of iterations: 72315417 Elapsed time: 0.889931667 seconds

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_9

Input Directory Exists!
Solution found!
EFFFADDD
EEFFAADB
EEEFACBB
EEEECCCB
Number of iterations: 72315417
Elapsed time: 0.892993583 seconds
Output file has been created!
```

Test Case #8:

input.txt	output.txt
5 5 1 DEFAULT A	-

```
Current directory: /home/nathan/Documents/Tucil1_13523032

Expected directory: /home/nathan/Documents/Tucil1_13523032/test/[dir_name]/input.txt

Input dir_name: input_10

Input Directory Exists!
Solution not found :(
Elapsed time: 0.001103666 seconds
```

IV. 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	✓	