Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Laporan Tugas Kecil 1 IF2211 Strategi Algoritma

Muhammad Naufal Rayhannida - 10123006

Institut Teknologi Bandung

Jl. Ganesa No.10, Lb. Siliwangi, Kota Bandung, 40132

2025

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.
Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. Board (Papan) – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
2. Blok/Piece – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Blok yang digunakan memiliki bentuk-bentuk yang tidak reguler, sehingga membutuhkan *trial and error* dan intuisi untuk mengisi seluruh area papan. Untuk menyelesaikan permainan ini, akan digunakan algoritma *brute force*.

Algoritma *brute force* adalah algoritma yang mencari solusi dari suatu masalah dengan cara meninjau seluruh kandidat solusi dari masalah. Algoritma ini akan selalu menemukan solusi optimal, namun memiliki kompleksitas waktu yang buruk.

Untuk menyelesaikan permainan IQ Puzzler Pro dengan algoritma *brute force*, prosedur yang akan dilakukan sebagai berikut

1. Ambil blok pertama, apakah dapat disimpan pada posisi pertama (0, 0)?
2. Jika dapat disimpan, blok akan disimpan pada posisi tersebut dan akan lanjut ke blok selanjutnya.
3. Jika tidak dapat disimpan, ulangi langkah pertama dengan setiap permutasi rotasi dan pencerminan pada blok.
4. Jika setiap permutasi blok tidak dapat disimpan, akan dicoba ulang dengan blok selanjutnya.
5. Jika setiap permutasi dari setiap blok tidak dapat disimpan, maka blok yang disimpan sebelumnya akan diubah permutasinya.
6. Langkah 1-5 akan dilakukan sampai papan penuh terisi oleh blok.

BAB 2
IMPLEMENTASI

Bahasa yang digunakan adalah Java.

2.1 Main.java

Method	Deskripsi
`Board solveRec(ArrayList<Board> boards, Block[] blocks, int depth)`	Implementasi algoritma *brute force* dengan backtracking secara recursive.
`Board solve(Board board, Block[] blocks)`	Pemanggil method `solveRec.`
`void main(String[] args)`	Titik masuk untuk program.

2.2 Block.java

Method	Deskripsi
`void iterate(Function3<Character, Integer, Integer> f)`	Untuk mengiterasi setiap karakter pada blok dengan suatu fungsi anonymous.
`int[] size()`	*Getter* untuk besar balok.
`Block rotate(int r)`	Membuat balok baru yang dirotasi sebanyak $r * 90$ deg.
`Block flip(int f)`	Membuat balok baru yang dicerminkan.
`void print()`	Mengeluarkan bentuk balok pada console.

2.3 Board.java

Board adalah abstract class dengan metode-metode berikut

Method	Deskripsi
`Board clone()`	Membuat `Board` baru yang persis sama.
`boolean place(Block b, int x, int y)`	Menyimpan suatu balok pada (x, y), dengan return boolean untuk sukses/gagal.
`boolean isEmpty(int y, int x)`	Mengecek jika (x, y) kosong atau tidak.
`boolean isOccupied(int y, int x)`	Mengecek jika (x, y) dapat disimpan blok atau tidak.
`String toString(Boolean color)`	Membuat representasi string untuk papan
`void print()`	Mengeluarkan papan pada console.
`void toFile(String filepath)`	Menyimpan papan sebagai file .txt
`void toImage(String filepath)`	Menyimpan papan sebagai gambar .png

BAB 3
SOURCE CODE

Source code dapat dilihat pada repository berikut
https://github.com/naufal101006/Tucil1_10123006

3.1 Main.java

```java
package main;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import main.Board.*;

public class Main {
    private static int iters = 0;

    private static Board solveRec(ArrayList<Board> boards, Block[] blocks,
int depth) {
        Board b =  boards.get(boards.size()-1);
        if (depth >= blocks.length) {
            return b;
        }

        for (int i = 0; i < b.board.length; i++) {
            for (int j = 0; j < b.board[0].length; j++) {
                if (!b.isOccupied(i, j)) {
                    for (int f = 0; f < 2; f++) {
                        for (int r = 0; r < 4; r++) {
                            Board clone = b.clone();
                            boolean canPlace =
clone.place(blocks[depth].flip(f).rotate(r), i, j);

                            if (canPlace) {
                                iters++;
                                boards.add(clone);

                                Board next = solveRec(boards, blocks,
depth+1);

                                if (next != null) {
                                    return next;
                                }
                            }
                        }
                    }
                }
            }
        }

        boards.remove(boards.size()-1);
        return null;
    }

    private static Board solve(Board board, Block[] blocks) {
        ArrayList<Board> list = new ArrayList<Board>();
        list.add(board);

        Board result = solveRec(list, blocks, 0);
        return result;
    }

    public static void main(String[] args) {
        Board board = new RectBoard(0, 0);
        ArrayList<Block> blocks = new ArrayList<Block>();

        try(BufferedReader br = new BufferedReader(new FileReader(args[0])))
{
            StringBuilder sb = new StringBuilder();
            String line = br.readLine();
            int lineIdx = 0;

            int x = -1;
            int y = -1;
            int blockLen = -1;
            int blockArea = 0;
            char currentChar = '\u0000';
            String boardType = "";

            while (line != null) {
                // Size
                if (lineIdx == 0) {
                    String[] parts = line.split(" ");
                    if (parts.length != 3) {
                        throw new IOException("Wrong arguments, check first
line.");
                    }

                    y = Integer.parseInt(parts[0]);
                    x = Integer.parseInt(parts[1]);
                    blockLen = Integer.parseInt(parts[2]);
                // Constructor
                } else if (lineIdx == 1) {
                    switch (line) {
                        case "DEFAULT":
                            boardType = "Default";
                            board = new RectBoard(y, x);
                            // board.print();
                            break;
                        case "CUSTOM":
                            boardType = "Custom";
                            break;
                        default:
                            throw new IOException("Wrong board type, check
second line.");
                    }
                // Block reading
                } else {
                    if (boardType == "Custom" && lineIdx < 2+y) {
                        sb.append(line);
                        sb.append('\n');
                    } else {
                        if (boardType == "Custom" && lineIdx == 2+y) {
                            board = new CustomBoard(y, x, sb.toString());
                            sb.setLength(0);
                        }
                        if (sb.length() == 0 && blocks.size() == 0) {
                            currentChar = line.trim().charAt(0);
                        }

                        if (currentChar != line.trim().charAt(0)) {
                            Block newBlock = new Block(sb.toString());
                            blocks.add(newBlock);
                            blockArea += newBlock.area;

                            sb.setLength(0);
                            currentChar = line.trim().charAt(0);
                        }

                        sb.append(line);
                        sb.append('\n');
                    }
                }

                line = br.readLine();
                lineIdx++;
            }
            Block newBlock = new Block(sb.toString());
            blocks.add(newBlock);
            blockArea += newBlock.area;
            sb = null;

            if (blocks.size() != blockLen) {
                throw new IOException("Wrong block length, check first
line.");
            }

            if (board.area != blockArea) {
                throw new IOException("Block area and board area
mismatch.");
            }

            Block[] blockArr = new Block[blockLen];
            blockArr = blocks.toArray(blockArr);

            long start = System.currentTimeMillis();
            Board solution = solve(board, blockArr);
            long end = System.currentTimeMillis();

            if (solution != null) {
                solution.print();
                System.out.println("Took " + (end-start) + "ms");
                System.out.println("Iterations: " + iters);
                solution.toFile(args[0].replaceAll("(\\w+)\\.txt$",
"$1-result.txt"));
                solution.toImage(args[0].replaceAll("(\\w+)\\.txt$",
"$1-result.png"));
            } else {
                System.out.println("No solution");
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found!");
            System.err.println(e);
        } catch (IOException e) {
            System.out.println("Failed to read!");
            System.err.println(e);
        }
    }
}
```

3.2 Block.java

```
package main;

import main.utils.Function3;

public class Block {
    public Character[][] block;
    public int area;

    public Block(String s) {
        String[] parts = s.split("\\r?\\n");
        int maxLen = -1;

        for (String p : parts) {
            maxLen = Math.max(p.length(), maxLen);
        }

        this.block = new Character[parts.length][maxLen];

        for (int i = 0; i < this.block.length; i++) {
            for (int j = 0; j < this.block[i].length; j++) {
                this.block[i][j] = parts[i].length() > j ?
(parts[i].charAt(j) != ' ' ? parts[i].charAt(j) : '.') : '.';
                if (Character.isAlphabetic(this.block[i][j])) {
                    this.area++;
                }
            }
        }
    }
```

```java
public void iterate(Function3<Character, Integer, Integer> f) {
    for (int i = 0; i < this.block.length; i++) {
        for (int j = 0; j < this.block[i].length; j++) {
            f.apply(this.block[i][j], i, j);
        }
    }
}

public int[] size() {
    int[] L = {this.block.length, this.block[0].length};
    return L;
}

public Block rotate(int r) {
    StringBuilder sb = new StringBuilder();

    switch(r) {
        case 0:
            return this;
        case 1:
            for (int j = this.block[0].length-1; j >= 0; j--) {
                for (int i = 0; i < this.block.length; i++) {
                    sb.append(this.block[i][j]);
                }
                sb.append('\n');
            }
            break;
        case 2:
            for (int i = this.block.length-1; i >= 0; i--) {
                for (int j = this.block[0].length-1; j >= 0; j--) {
                    sb.append(this.block[i][j]);
                }
                sb.append('\n');
            }
            break;
        case 3:
            for (int j = 0; j < this.block[0].length; j++) {
                for (int i = this.block.length-1; i >= 0; i--) {
                    sb.append(this.block[i][j]);
                }
                sb.append('\n');
            }
            break;
    }
    return new Block(sb.toString());
}
```

```
    public Block flip(int f) {
        StringBuilder sb = new StringBuilder();

        switch(f) {
            case 0:
                return this;
            case 1:
                for (int i = 0; i < this.block.length; i++) {
                    for (int j = this.block[0].length-1; j >= 0; j--) {
                        sb.append(this.block[i][j]);
                    }
                    sb.append('\n');
                }
                break;
        }
        return new Block(sb.toString());
    }

    public void print() {
        for (int i = 0; i < this.block.length; i++) {
            StringBuilder sb = new StringBuilder();

            for (int j = 0; j < this.block[i].length; j++) {
                sb.append(this.block[i][j]);
            }

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

3.3 Board.java

```
package main.Board;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import main.Block;
```

```java
public abstract class Board {
    public Character[][] board;
    public int area = 0;
    private static final Map<Character, String> consoleColorMap = new
HashMap<>();
    private static final Map<Character, Integer> imageColorMap = new
HashMap<>();
    static {
        consoleColorMap.put('A', "\u001B[31m");
        consoleColorMap.put('B', "\u001B[32m");
        consoleColorMap.put('C', "\u001B[33m");
        consoleColorMap.put('D', "\u001B[34m");
        consoleColorMap.put('E', "\u001B[35m");
        consoleColorMap.put('F', "\u001B[36m");
        consoleColorMap.put('G', "\u001B[38;2;140;184;165m");
        consoleColorMap.put('H', "\u001B[91m");
        consoleColorMap.put('I', "\u001B[92m");
        consoleColorMap.put('J', "\u001B[93m");
        consoleColorMap.put('K', "\u001B[94m");
        consoleColorMap.put('L', "\u001B[95m");
        consoleColorMap.put('M', "\u001B[96m");
        consoleColorMap.put('N', "\u001B[38;2;191;105;29m");
        consoleColorMap.put('O', "\u001B[38;2;38;171;164m");
        consoleColorMap.put('P', "\u001B[38;2;156;201;66m");
        consoleColorMap.put('Q', "\u001B[38;2;128;43;126m");
        consoleColorMap.put('R', "\u001B[38;2;194;164;89m");
        consoleColorMap.put('S', "\u001B[38;2;186;123;145m");
        consoleColorMap.put('T', "\u001B[38;2;115;55;37m");
        consoleColorMap.put('U', "\u001B[38;2;118;194;172m");
        consoleColorMap.put('V', "\u001B[38;2;142;21;194m");
        consoleColorMap.put('W', "\u001B[38;2;76;81;181m");
        consoleColorMap.put('X', "\u001B[38;2;77;153;61m");
        consoleColorMap.put('Y', "\u001B[38;2;94;35;73m");
        consoleColorMap.put('Z', "\u001B[38;2;39;161;81m");

        imageColorMap.put('A', 0xff800000);
        imageColorMap.put('B', 0xff008000);
        imageColorMap.put('C', 0xff808000);
        imageColorMap.put('D', 0xff000080);
        imageColorMap.put('E', 0xff800080);
        imageColorMap.put('F', 0xff008080);
        imageColorMap.put('G', 0xff8cb8a5);
        imageColorMap.put('H', 0xffff0000);
        imageColorMap.put('I', 0xff00ff00);
        imageColorMap.put('J', 0xffffff00);
        imageColorMap.put('K', 0xff0000ff);
        imageColorMap.put('L', 0xffff00ff);
        imageColorMap.put('M', 0xff00ffff);
        imageColorMap.put('N', 0xffbf691d);
        imageColorMap.put('O', 0xff26aba4);
        imageColorMap.put('P', 0xff9cc942);
        imageColorMap.put('Q', 0xff802b7e);
        imageColorMap.put('R', 0xffc2a459);
        imageColorMap.put('S', 0xffba7b91);
        imageColorMap.put('T', 0xff733725);
        imageColorMap.put('U', 0xff76c2ac);
        imageColorMap.put('V', 0xff8e15c2);
        imageColorMap.put('W', 0xff4c51b5);
        imageColorMap.put('X', 0xff4d993d);
        imageColorMap.put('Y', 0xff5e2349);
        imageColorMap.put('Z', 0xff27a151);
    }

    public abstract Board clone();
    public abstract boolean place(Block b, int x, int y);
    public boolean isEmpty(int y, int x) {
        if (y < this.board.length && x < this.board[0].length) {
            return this.board[y][x] == '.';
        }
        return false;
    }

    public boolean isOccupied(int y, int x) {
        if (y < this.board.length && x < this.board[0].length) {
            return Character.isLetterOrDigit(this.board[y][x]);
        }
        return false;
    }

    public String toString(Boolean color) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < this.board.length; i++) {

            for (int j = 0; j < this.board[i].length; j++) {
                char c = this.board[i][j];
                if (color) {
                    sb.append(consoleColorMap.containsKey(c) ?
consoleColorMap.get(c) : "\u001B[0m");
                    sb.append(c);
                    sb.append("\u001B[0m");
                } else {
                    sb.append(c);
                }
            }

            sb.append("\n");
```

```java
        }
        return sb.toString();
    }

    public void print() {
        System.out.println(toString(true));
    }

    public void toFile(String filepath) {
        try (FileWriter f = new FileWriter(filepath)) {
            f.write(toString(false));
        } catch (IOException e) {
            System.err.println(e);
        }
    }

    public void toImage(String filepath) {
        int PIXELS_PER_CHAR = 50;
        try {
            BufferedImage bi = new BufferedImage(board[0].length *
PIXELS_PER_CHAR, board.length * PIXELS_PER_CHAR,
BufferedImage.TYPE_INT_ARGB);

            for (int i = 0; i < board.length; i++) {
                for (int j = 0; j < board[0].length; j++) {
                    for (int px = 0; px < PIXELS_PER_CHAR; px++) {
                        for (int py = 0; py < PIXELS_PER_CHAR; py++) {
                            char c = this.board[i][j];
                            bi.setRGB(j*PIXELS_PER_CHAR+px,
i*PIXELS_PER_CHAR+py, imageColorMap.containsKey(c) ? imageColorMap.get(c) :
0x00000000);
                        }
                    }
                }
            }

            File output = new File(filepath);
            ImageIO.write(bi, "png", output);
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

3.4 RectBoard.java

```
package main.Board;

import main.Block;

public class RectBoard extends Board {
    public RectBoard(int y, int x) {
        this.board = new Character[y][x];
        this.area = x*y;
        for (int i = 0; i < this.board.length; i++) {
            for (int j = 0; j < this.board[i].length; j++) {
                this.board[i][j] = '.';
            }
        }
    }

    public RectBoard clone() {
        RectBoard clone = new RectBoard(this.board.length,
this.board[0].length);

        for (int i = 0; i < this.board.length; i++) {
            for (int j = 0; j < this.board[i].length; j++) {
                clone.board[i][j] = this.board[i][j];
            }
        }

        return clone;
    }

    public boolean place(Block b, int y, int x) {
        boolean canPlace = true;
        int[] blockSize = b.size();

        for (int i = 0; i < blockSize[0]; i++) {
            for (int j = 0; j < blockSize[1]; j++) {
                if ((!isEmpty(y+i, x+j) && b.block[i][j] != '.') ||
!canPlace) {
                    canPlace = false;
                    break;
                }
            }
        }

        if (canPlace) {
            b.iterate((c, i, j) -> {
                if (c != '.') {
                    this.board[y+i][x+j] = c;
                }
            });
        }

        return canPlace;
    }

    public boolean isEmpty(int y, int x) {
        if (y < this.board.length && x < this.board[0].length) {
            return this.board[y][x] == '.';
        }
        return false;
    }
}
```

3.5 CustomBoard.java

```
package main.Board;

import java.io.IOException;

import main.Block;

public class CustomBoard extends Board {
    public CustomBoard(int y, int x) {
        this.board = new Character[y][x];

        for (int i = 0; i < this.board.length; i++) {
            for (int j = 0; j < this.board[i].length; j++) {
                this.board[i][j] = '.';
            }
        }
    }

    public CustomBoard(int y, int x, String layout) throws IOException {
        this.board = new Character[y][x];
        String[] parts = layout.split("\\r?\\n");

        if (parts.length != y || parts[0].length() != x) {
            throw new IOException("Wrong board size, check first line.");
        }

        for (int i = 0; i < this.board.length; i++) {
            for (int j = 0; j < this.board[i].length; j++) {
                if (parts[i].charAt(j) == '.') {
                    this.board[i][j] = ' ';
                } else if (parts[i].charAt(j) == 'X') {
```

```java
                    this.board[i][j] = '.';
                    this.area++;
                }
            }
        }
    }

    public CustomBoard clone() {
        CustomBoard clone = new CustomBoard(this.board.length,
this.board[0].length);
        clone.area = this.area;

        for (int i = 0; i < this.board.length; i++) {
            for (int j = 0; j < this.board[i].length; j++) {
                clone.board[i][j] = this.board[i][j];
            }
        }

        return clone;
    }

    public boolean place(Block b, int y, int x) {
        boolean canPlace = true;
        int[] blockSize = b.size();

        for (int i = 0; i < blockSize[0]; i++) {
            for (int j = 0; j < blockSize[1]; j++) {
                if ((!isEmpty(y+i, x+j) && b.block[i][j] != '.') ||
!canPlace) {
                    canPlace = false;
                    break;
                }
            }
        }

        if (canPlace) {
            b.iterate((c, i, j) -> {
                if (c != '.') {
                    this.board[y+i][x+j] = c;
                }
            });
        }

        return canPlace;
    }
}
```

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	✓	