

IF2211 Strategi Algoritma
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Laporan Tugas Kecil 1

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2024/2025



Disusun oleh:

Theo Kurniady (13523154)

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

**BANDUNG
2024**

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
DESKRIPSI MASALAH	3
BAB II	4
SPESIFIKASI TUGAS	4
BAB III	6
TEORI SINGKAT	6
BAB IV	7
CARA KERJA ALGORITMA	7
BAB V	8
SOURCE PROGRAM	8
BAB VI	13
PENGETESAN PROGRAM	13
BAB VII	
KESIMPULAN DAN SARAN	17
a. Kesimpulan	17
b. Saran	17
BAB VIII	18
DAFTAR PUSTAKA	18
1. Sumber Pustaka	18
2. Lampiran	18

BAB I

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.

Tugas anda adalah menemukan cukup satu solusi dari permainan **IQ Puzzler Pro** dengan menggunakan *algoritma Brute Force*, atau menampilkan bahwa solusi tidak ditemukan jika tidak ada solusi yang mungkin dari puzzle.



Gambar Permainan IQ Puzzler Pro

(Sumber: <https://www.smartgames.eu/uk/one-player-games/iq-puzzler-pro>)

BAB II

SPESIFIKASI TUGAS

- Membuat program sederhana dalam bahasa Java yang mengimplementasikan *algoritma Brute Force* untuk mencari salah satu solusi permainan IQ Puzzler Pro.
- Algoritma brute force yang diimplementasikan harus bersifat “murni”, tidak boleh memanfaatkan heuristik.
- Papan yang perlu diisi mulanya akan selalu kosong.
- Sebuah blok puzzle bisa saja dirotasi maupun dicerminkan sebelum diletakan pada papan.
- **Input:**
 1. **N** dan **M** merepresentasikan baris dan kolom dari papan permainan
 2. **P** merepresentasikan jumlah blok yang ada.
 3. **S** merepresentasikan berbagai bentuk papan permainan, yaitu DEFAULT (persegi), CUSTOM (bentuk bebas, bonus), dan PYRAMID (bentuk piramida, bonus). Blok direpresentasikan dalam bentuk karakter alfabet.
 4. Input yang dibaca dari sebuah berkas berekstensi .txt yang memiliki format sebagai berikut:

```
N M P
S
puzzle_1_shape
puzzle_2_shape
...
puzzle_P_shape
```

```
Contoh :
5 5 8
DEFAULT
AA
A
BB
B
CC
C
DD
D
EE
EE
E
FF
FF
F
GGG
```

- **Output:**

1. Papan yang telah diisi dengan semua blok sesuai aturan. (Bila tidak memungkinkan, program memberi error message.)
2. Jumlah gerakan yang dilakukan program untuk mendapat hasil.
3. Waktu yang diperlukan untuk mencapai hasil.
4. Hasil dapat disimpan di dalam file dengan format .txt dan .png (gambar).

AABBD
AEBDD
EECCF
EECFF
GGGFF

Waktu pencarian : 5700 ms

Banyak kasus yang ditinjau : 3339

Apakah anda ingin menyimpan solusi? (ya/tidak)

ya

Board saved to test/output.txt

Board saved as image: test/output.png

BAB III

TEORI SINGKAT

Algoritma brute force merupakan metode pencarian solusi dengan mencoba setiap kemungkinan yang ada secara sistematis hingga menemukan jawaban yang benar atau paling sesuai. Pendekatan ini cukup sederhana dan mudah dipahami karena tidak memerlukan pemahaman mendalam mengenai masalah yang dihadapi. Meskipun demikian, brute force memiliki kelemahan signifikan, yaitu konsumsi waktu dan sumber daya yang sangat besar, terutama jika ruang solusi yang harus dieksplorasi sangat luas atau memiliki tingkat kompleksitas yang tinggi.

IQ Puzzler Pro adalah permainan puzzle yang mengharuskan pemain menempatkan blok berbentuk unik ke dalam papan dengan berbagai kemungkinan susunan. Karena setiap blok memiliki orientasi dan posisi yang beragam, pemecahan puzzle ini bisa dianggap sebagai masalah pencarian kombinatorial. Permainan dapat diselesaikan dengan memenuhi setiap bagian kosong di dalam papan. Papan dapat berbentuk persegi, piramida, ataupun bebas / custom.

Dengan pengetahuan algoritma brute force, permasalahan yang membutuhkan pemikiran seperti IQ Puzzler Pro dapat dicari solusinya. Dengan brute force, semua kemungkinan susunan diuji, disertai dengan backtracking untuk melanjutkan pencarian. Algoritma ini bagus untuk permainan puzzle sederhana, terutama bila ruang permainan juga kecil. Walaupun, waktunya yang lama dalam menyelesaikan permasalahan yang besar, algoritma brute force relatif mudah dibuat sehingga dapat menjadi algoritma backup.

BAB IV

CARA KERJA ALGORITMA

1. **Membaca File** : File dengan format .txt akan dibaca oleh program. Baris pertama dari file berisi 3 angka, yang dibaca dan dimasukkan ke dalam variabel N (baris), M (kolom), P (jumlah piece). Baris kedua dari file berisi jenis papan, yakni antara DEFAULT (persegi, tidak mampu custom dan piramida). Di bawah baris kedua adalah blok yang akan digunakan untuk memenuhi papan permainan. Blok direpresentasikan dengan alfabet. Bila scanner mendeteksi karakter yang berbeda dari sebelumnya, program akan mengubah blok dengan alfabet sebelumnya menjadi sebuah matriks yang dimasukkan ke dalam List of Matrix. Proses diulang hingga blok terakhir. Jika total bit blok lebih dari ukuran area papan, maka solusi tidak dapat ditemukan.
2. **Mencari Setiap Orientasi Blok**: Setiap blok mampu memiliki 8 bentuk yang berbeda. Untuk mencari seluruh kemungkinan bentuk, dapat dilakukan rotasi blok / matriks sebanyak 4 kali dan membalikkan blok dilanjutkan dengan melakukan rotasi sebanyak 4 kali lagi. Seluruh orientasi dimasukkan ke dalam List of Matrix.
3. **Pencarian Solusi melalui Algoritma Brute Force** : Pencarian solusi berputar pada fungsi Solve. Basis dari rekursi ini adalah ketika isComplete(Board) atau papan yang sudah penuh dengan blok, alias tidak memiliki titik atau ruang kosong. Blok diambil mulai dari paling depan list of matrix sambil dihitung indexnya. Blok tersebut akan dicari semua orientasinya. Untuk tiap orientasinya, blok akan melalui setiap titik di papan. Bila dapat ditaruh, maka blok akan langsung ditaruh dan indeks blockList bertambah sehingga bisa memakai blok selanjutnya. Bila sudah sampai ujung dan tidak bisa, maka blok akan berubah menjadi orientasi selanjutnya dan bergerak dari awal lagi hingga dapat ditaruh atau habis. Bila habis semua gerakan, maka akan dilakukan akan dilakukan removePiece, yaitu proses backtracking. Program akan melanjutkan sisa kemungkinan blok sebelumnya dan mengulang proses terus menerus hingga mencapai basis.
4. **Penyimpanan Solusi** : Bila sebuah permainan berhasil diselesaikan atau didapatkan sebuah solusi, tombol save akan muncul, yang berguna untuk menyimpan hasil papan ke file. Fitur ini akan langsung menyimpan papan ke dalam dua format, yaitu .txt dan .png. Kedua file disimpan di /test.

BAB V

SOURCE PROGRAM

Solve (Rekursi)

```
public class Solve {
    public static int moveCount = 0;
    public static boolean isSolved = false;

    public static boolean Solve(char[][] board, List<char[][]> pieces, int index, MainController controller) {
        if (isComplete(board)) return true;
        else{
            char[][] piece = pieces.get(index);
            char pieceLabel = (char) ('A' + index);

            List<char[][]> orientations = getAllOrientations(piece);

            for (char[][] orientation : orientations) {
                for (int row = 0; row < board.length; row++) {
                    for (int col = 0; col < board[0].length; col++) {
                        if (canPlacePiece(board, orientation, row, col)) {
                            placePiece(board, orientation, row, col, pieceLabel);
                            moveCount++;

                            Platform.runLater(() -> controller.updateMoveCount(moveCount, timeElapsed:0));

                            // clearScreen();
                            // MatrixUtils.printMatrix(board);

                            Platform.runLater(() -> controller.updateBoard(board));

                            if (Solve(board, pieces, index + 1, controller)) return true;

                            removePiece(board, orientation, row, col);
                            // clearScreen();
                            // MatrixUtils.printMatrix(board);
                            Platform.runLater(() -> controller.updateBoard(board));
                        }
                    }
                }
            }
            return false;
        }
    }
}
```



```

public static boolean isComplete(char[][] board) {
    for (char[] row : board) {
        for (char cell : row) {
            if (cell == '.') {
                return false;
            }
        }
    }
    return true;
}

```

```

public static List<char[][]> getAllOrientations(char[][] piece) {
    List<char[][]> orientations = new ArrayList<>();

    orientations.add(piece);
    Arrays.stream(new int[]{0, 1, 2, 3}).forEach(i -> {
        char[][] rotated = MatrixUtils.rotateMatrix(orientations.get(orientations.size() - 1));
        orientations.add(rotated);
    });

    char[][] flipped = MatrixUtils.mirrorHorizontal(piece);
    orientations.add(flipped);
    Arrays.stream(new int[]{0, 1, 2, 3}).forEach(i -> {
        char[][] rotated = MatrixUtils.rotateMatrix(orientations.get(orientations.size() - 1));
        orientations.add(rotated);
    });

    return orientations;
}

```

```

public static boolean canPlacePiece(char[][] board, char[][] piece, int startRow, int startCol) {
    if (startRow + piece.length > board.length || startCol + piece[0].length > board[0].length) {
        return false;
    }

    for (int i = 0; i < piece.length; i++) {
        for (int j = 0; j < piece[i].length; j++) {
            if (piece[i][j] != '.' && board[startRow + i][startCol + j] != '.') {
                return false;
            }
        }
    }

    return true;
}

```

```

public static void removePiece(char[][] board, char[][] piece, int startRow, int startCol) {
    for (int i = 0; i < piece.length; i++) {
        for (int j = 0; j < piece[i].length; j++) {
            if (piece[i][j] != '.') {
                board[startRow + i][startCol + j] = '.';
            }
        }
    }
}

```

```

public static void placePiece(char[][] board, char[][] piece, int startRow, int startCol, char pieceLabel) {
    for (int i = 0; i < piece.length; i++) {
        for (int j = 0; j < piece[i].length; j++) {
            if (piece[i][j] != '.') {
                board[startRow + i][startCol + j] = pieceLabel;
            }
        }
    }
}

```

Main Controller

```

@FXML
private void solvePuzzle() {
    String filename = fileInput.getText().trim();
    if (filename.isEmpty()) {
        resultLabel.setText(value: "Please select a file.");
        return;
    }

    PuzzleData data = ReadFile.parseFile(filename);
    int N = data.getN();
    int M = data.getM();
    List<char[][]> blockList = data.getBlockList();

    int sum = 0;

    for (char[][] piece : blockList) {
        sum += PuzzleData.countCharsInPiece(piece);
    }

    if (sum != N * M) {
        resultLabel.setText(value: "No Solution Possible.");
    } else {
        char[][] board = Solve.createEmptyBoard(N, M);
        Solve.moveCount = 0;
    }
}

```

```

        final long startTime = System.currentTimeMillis();

        new Thread(() -> {
            boolean solved = Solve.Solve(board, blockList, index:0, this);
            long elapsedTime = System.currentTimeMillis() - startTime;

            Platform.runLater(() -> {
                if (solved) {
                    resultLabel.setText(value:"Solution Found!");
                    solutionBoard = board;
                    updateBoard(board);
                    saveButton.setDisable(value:false);
                } else {
                    resultLabel.setText(value:"No Solution Possible.");
                }

                updateMoveCount(Solve.moveCount, elapsedTime);
            });
        }).start();
    }
}

```

```

public void updateMoveCount(int moves, long timeElapsed) {
    Platform.runLater(() -> {
        moveLabel.setText("Moves: " + moves);
        timeLabel.setText("Time: " + timeElapsed + " ms");
    });
}

```

ReadFile

```
public static PuzzleData parseFile(String filename) {
    int N = 0, M = 0, P = 0;
    List<char[][]> blockList = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
        String line = br.readLine();
        if (line != null) {
            String[] dimensions = line.split(regex:" ");
            N = Integer.parseInt(dimensions[0]);
            M = Integer.parseInt(dimensions[1]);
            P = Integer.parseInt(dimensions[2]);
        }

        br.readLine();

        blockList = parseBlocks(filename);
    } catch (IOException e) {
        e.printStackTrace();
    }

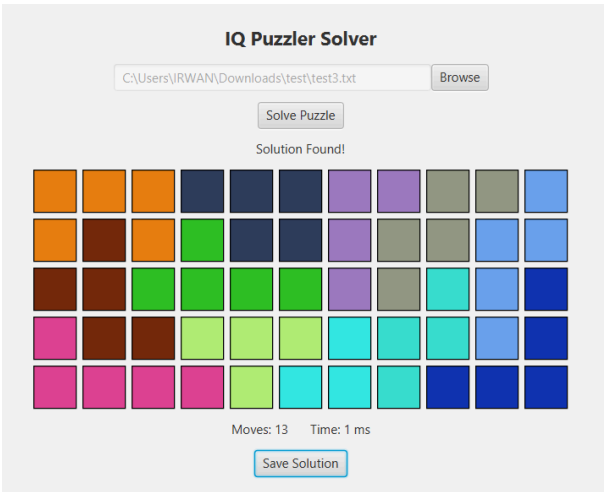
    return new PuzzleData(N, M, P, blockList);
}
```

BAB VI

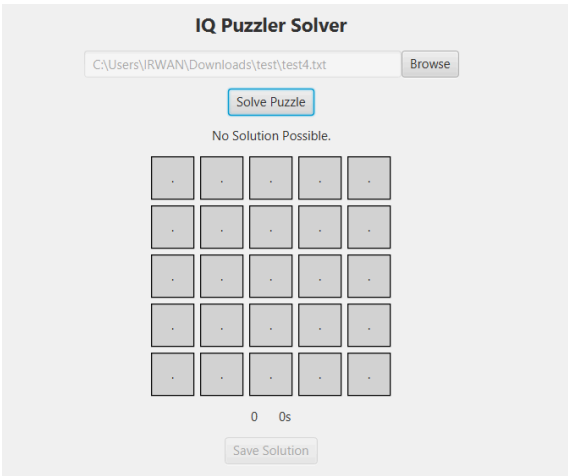
PENGETESAN PROGRAM

Input	Output
5 5 8 DEFAULT AA A BB B CC C DD D EE EE E FF FF F GGG	
4 4 5 DEFAULT AA A BB B CC C DDD D EE E	

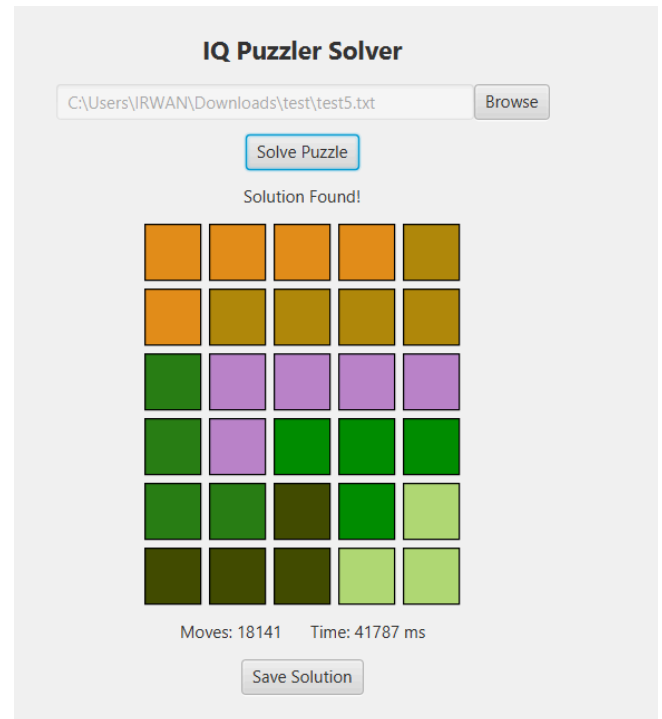
5 11 12
DEFAULT
AAA
A A
BBB
BB
CC
C
C
DD
DD
D
E
EE
E
E
F
FF
FF
G
GGGG
H
HHHH
III
I
J
JJ
K
KK
K
L
L
LLL



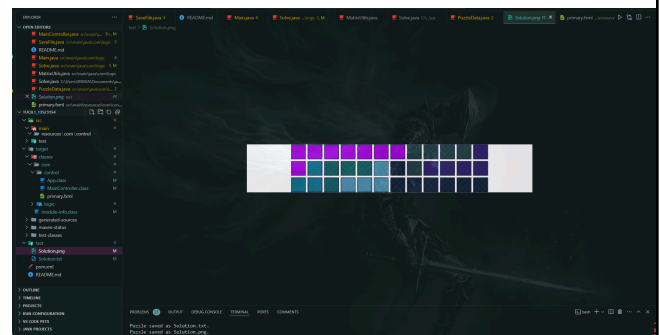
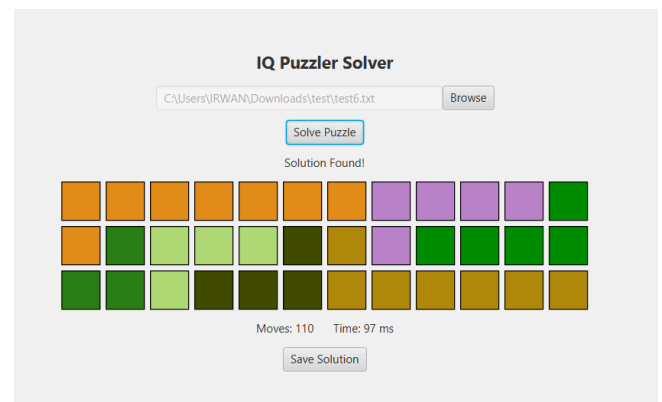
2 2 2
DEFAULT
A
A
AA
BB



6 5 7
 DEFAULT
 AAAAA
 A
 BBBB
 B
 CC
 C
 DDDD
 D
 EEE
 E
 FFF
 F
 GGG
 G



3 12 8
 DEFAULT
 AAAAAAA
 A
 BBBB
 B
 CCC
 C
 DDDDDD
 D
 EEE
 E
 FF
 F
 GGGG
 G



















Contoh Save sebagai gambar

9 3 6
DEFAULT
AAA
A A
BBB
B
CCCC
C C
DDDD
D
EEE
E
FF
F

IQ Puzzler Solver

Solution Found!

Moves: 6579 Time: 14671 ms

BAB VII

KESIMPULAN DAN SARAN

a. Kesimpulan

Algoritma *brute force* dapat digunakan untuk mencari salah satu solusi yang bisa dicapai pada permainan IQ Puzzler Pro. Kecepatan program bergantung pada ukuran data yang diinput, waktu cukup lama mulai terasa ketika matriks yang diinput tidak urut, hingga yang terburuknya ketika ada lebih dari 10 blok dan blok pertama dan akhir ditukar posisinya. Hal ini dikarenakan algoritma *brute force* memiliki kompleksitas waktu yang relatif buruk dan harus melakukan segala cara dengan urut.

b. Saran

Program dapat dijadikan acuan untuk pengembangan permainan berbasis pola lainnya. Program juga tidak ramah karena membuat laptop saya ngelag karena memory yang tidak kuat.

BAB VIII

DAFTAR PUSTAKA

1. Sumber Pustaka

[1] R. Munir, "Bahan Kuliah IF2211 Strategi Algoritma Algoritma Brute Force." Accessed: Feb. 21, 2025. [Online]. Available: [informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-\(2025\)-Bag1.pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf)

2. Lampiran

1. Github: https://github.com/TKurr/Tucil1_13523154

2.

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 <i>Graphical User Interface</i> (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	
7	Program dapat menyelesaikan kasus konfigurasi <i>custom</i>		✓
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	

