

# Laporan Tugas Kecil 1 IF2211 Strategi Algoritma

## IQ Puzzle Pro Solver

Benedict Presley

13523067

[13523067@std.stei.itb.ac.id](mailto:13523067@std.stei.itb.ac.id)

Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2025

<b>Permasalahan</b>	<b>1</b>
<b>Pendekatan Solusi</b>	<b>1</b>
<b>Implementasi</b>	<b>2</b>
Struktur Data	2
Board (Papan)	2
Piece (Blok)	2
Orientasi Blok	3
Memeriksa Apakah Blok Dapat Diletakkan	4
Meletakkan Blok	5
Mengeluarkan Blok	5
Solusi	6
Warna	9
Output To Text	9
Output To Image	9
<b>Pengujian</b>	<b>11</b>
<b>Lampiran</b>	<b>16</b>

# Permasalahan

IQ Puzzler Pro adalah permainan papan yang bertujuan 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. Setiap blok dapat dirotasikan atau direfleksikan.

Tujuan dari program yang dibuat adalah untuk mencari **1 (satu)** solusi dari masukan yang diberikan atau menyatakan bahwa tidak ada solusi.

## Pendekatan Solusi

Paradigma yang digunakan dalam menyelesaikan permasalahan ini adalah paradigma **brute force**. Paradigma brute force adalah pendekatan penyelesaian masalah yang mencoba semua kemungkinan solusi secara sistematis hingga menemukan solusi yang benar atau optimal. Metode ini bekerja dengan mengevaluasi setiap kemungkinan tanpa optimasi atau heuristik, sehingga sering memiliki kompleksitas waktu yang tinggi, terutama pada masalah dengan ruang pencarian besar. Meskipun tidak efisien dalam banyak kasus, brute force menjamin menemukan solusi jika ada. Hal ini dikarenakan sifat algoritma brute force yang mencoba semua kemungkinan yang ada.

Dalam kasus ini, algoritma akan mencoba meletakkan blok satu per satu hingga semua blok berhasil diletakkan. Dalam meletakkan suatu blok, algoritma juga akan memeriksa apakah blok dapat diletakkan pada daerah tersebut, yaitu blok masih berada dalam papan dan blok tidak tumpang tindih dengan blok lain yang sudah diletakkan. Jika blok berhasil diletakkan, algoritma akan mencoba meletakkan blok selanjutnya. Jika blok tidak berhasil diletakkan, algoritma akan mencari posisi lain untuk meletakkan blok tersebut.

# Implementasi

## Struktur Data

```
public class IntBooleanPair {  
    public int first;  
    public boolean second;  
}  
  
public class State {  
    public int placement_row, placement_column, piece_index,  
orientation_index, current_index;  
    public boolean remove;  
}
```

Struktur data IntBooleanPair menyimpan sebuah integer dan sebuah boolean. Struktur data IntBooleanPair akan digunakan untuk membantu dalam implementasi solusi.

Struktur data State digunakan untuk menyimpan kondisi papan secara implisit dalam implementasi solusi.

## Board (Papan)

```
public class Board {  
    private int rows;  
    private int cols;  
    private char[][] data;  
}
```

Board (Papan) adalah salah satu komponen utama dalam puzzle ini. Objek Board digunakan untuk menyimpan kondisi papan saat ini dan mendapatkan informasi mengenai kondisi papan menggunakan metode-metode yang ada.

## Piece (Blok)

```
public class Piece2D {  
    private int rows;
```

```

    private int cols;
    private char[][] data;
    public ArrayList<char[][]> orientations = new
ArrayList<char[][]>();
}

```

Piece (Blok) adalah salah satu komponen utama dalam puzzle ini. Objek Piece2D digunakan untuk menyimpan suatu blok. Piece2D juga menyimpan semua orientasi berbeda dari blok yang disimpan. Orientasi yang dapat dilakukan adalah rotasi dan refleksi.

## Orientasi Blok

```

public void generate_orientations(){
    char[][] temp = new char[rows][cols];
    for(int i = 0; i < rows; ++i){
        for(int j = 0; j < cols; ++j){
            temp[i][j] = data[i][j];
        }
    }
    for(int i = 0; i < 4; ++i){
        if(unique(temp)){
            orientations.add(temp);
        }
        temp = rotate(temp);
    }
    for(int i = 0; i < rows; ++i){
        for(int j = 0; j < cols; ++j){
            temp[i][j] = data[i][j];
        }
    }
    temp = flip(temp);
    for(int i = 0; i < 4; ++i){
        if(unique(temp)){
            orientations.add(temp);
        }
        temp = rotate(temp);
    }
}
}

```

Fungsi `generate_orientations()` digunakan untuk membangkitkan semua kemungkinan orientasi dari suatu puzzle. Fungsi `rotate()` digunakan untuk merotasikan blok sebesar 90 derajat berlawanan arah jarum jam. Fungsi `flip()` digunakan untuk merefleksitkan blok pada sumbu vertikal. Fungsi `unique()` memeriksa apakah suatu orientasi belum pernah terbentuk sebelumnya, hal ini digunakan untuk memastikan tidak ada variasi blok yang dicoba lebih dari satu kali pada saat algoritma brute force dijalankan.

## Memeriksa Apakah Blok Dapat Diletakkan

```
private boolean can_put_piece(char[][] piece, int piece_row,
int piece_col, int r, int c){
    if(!board.is_empty(r, c) && piece[0][0] != ' '){
        return false;
    }

    if(r + piece_row - 1 >= N || c + piece_col - 1 >= M){
        return false;
    }

    for(int i = r; i < r + piece_row; ++i){
        for(int j = c; j < c + piece_col; ++j){
            if(piece[i - r][j - c] == ' '){
                continue;
            }
            if(!board.is_empty(i, j)){
                return false;
            }
        }
    }
    return true;
}
```

Board `board` adalah matriks yang mengandung informasi papan saat ini. `char[][] piece` adalah suatu matriks yang berisi blok yang akan diletakkan. `piece_row` dan `piece_col` masing-masing menyatakan banyak baris dan kolom dari blok yang akan diletakkan. `r` dan `c` menyatakan posisi sudut kiri atas tempat meletakkan blok.

Jika blok dipastikan akan melewati batas papan, blok tidak dapat diletakkan. Jika blok tidak melewati papan, perlu dilakukan pemeriksaan untuk memastikan setiap posisi yang ditempati oleh blok belum ditempati oleh blok lain.

## Meletakkan Blok

```
private void put_piece(char[][] piece, int piece_row, int
piece_col, int r, int c){
    for(int i = r; i < r + piece_row; ++i){
        for(int j = c; j < c + piece_col; ++j){
            if(piece[i - r][j - c] == ' '){
                continue;
            }
            board.set(i, j, piece[i - r][j - c]);
        }
    }
}
```

Board board adalah matriks yang mengandung informasi papan saat ini. char[][] piece adalah suatu matriks yang berisi blok yang akan diletakkan. piece\_row dan piece\_col masing-masing menyatakan banyak baris dan kolom dari blok yang akan diletakkan. r dan c menyatakan posisi sudut kiri atas tempat meletakkan blok.

Fungsi ini dipanggil setelah blok dipastikan dapat diletakkan, menggunakan fungsi can\_put\_piece(). Setiap posisi yang ditempati blok akan ditandai pada board.

## Mengeluarkan Blok

```
private void remove_piece(char[][] piece, int piece_row, int
piece_col, int r, int c){
    for(int i = r; i < r + piece_row; ++i){
        for(int j = c; j < c + piece_col; ++j){
            if(piece[i - r][j - c] == ' '){
                continue;
            }
            board.set(i, j, ' ');
        }
    }
}
```

Board board adalah matriks yang mengandung informasi papan saat ini. char[][] piece adalah suatu matriks yang berisi blok yang akan diletakkan. piece\_row dan piece\_col masing-masing menyatakan banyak baris dan kolom dari blok yang akan diletakkan. r dan c menyatakan posisi sudut kiri atas tempat meletakkan blok.

Fungsi ini adalah kebalikan dari put\_piece(). Setiap posisi yang ditempati oleh blok akan dikosongkan dengan cara menandai dengan ' '.

## Solusi

```
private IntBooleanPair solve_iterative(){
    IntBooleanPair total = new IntBooleanPair(0, false);

    ArrayDeque<State> stack = new ArrayDeque<>();

    State start = new State(0, 0, 0, 0, 0, false);
    stack.push(start);

    while(!stack.isEmpty()){
        State now = stack.pop();

        if(!now.remove){
            if(now.piece_index != now.current_index){
                char[][] to_place =
pieces.get(now.piece_index).orientations.get(now.orientation_in
dex);

                int to_place_row = to_place.length;
                int to_place_col = to_place[0].length;

                put_piece(to_place, to_place_row, to_place_col,
now.placement_row, now.placement_column);
            }

            if(now.current_index >= P){
                boolean possible = (!board.empty_spaces()) &&
(board.count_unique() == P);
                if(possible){
                    total.first++;
                }
            }
        }
    }
}
```

```

        total.second = true;
        return total;
    }
    continue;
}

ArrayList<char[][]> orientations =
pieces.get(now.current_index).orientations;

boolean leaf = true;
for(int r = 0; r < N; ++r){
    for(int c = 0; c < M; ++c){
        if(!board.is_empty(r, c)){
            continue;
        }

        int iter = -1;
        for(char[][] current : orientations){

            iter++;
            int current_row = current.length;
            int current_col = current[0].length;

            if(can_put_piece(current, current_row,
current_col, r, c)){
                leaf = false;

                State rem = new State(r, c,
now.current_index, iter, now.current_index, true);
                stack.push(rem);

                State next = new State(r, c,
now.current_index, iter, now.current_index + 1, false);
                stack.push(next);
            }
        }
    }
}

if(leaf){

```



```

        total.first++;
    }
}
else{
    char[][] to_remove =
pieces.get(now.piece_index).orientations.get(now.orientation_in
dex);

    int to_remove_row = to_remove.length;
    int to_remove_col = to_remove[0].length;

    remove_piece(to_remove, to_remove_row,
to_remove_col, now.placement_row, now.placement_column);
}
}

return total;
}

```

Solusi brute force diimplementasikan secara iteratif dengan menggunakan stack untuk mensimulasikan sifat rekursif. Setiap State memiliki suatu boolean remove yang menyatakan apakah kondisi saat ini untuk meletakkan blok atau mengeluarkan blok. Meletakkan blok dapat dianggap sebagai perpindahan dari suatu state ke state baru dan mengeluarkan blok dapat dianggap sebagai perpindahan dari suatu state ke state sebelumnya yang memanggil state tersebut.

Untuk berpindah dari suatu state ke state baru, algoritma akan memeriksa setiap petak yang ada pada board. Bila petak tersebut kosong, dilakukan pemeriksaan apakah piece dapat diletakkan menggunakan `can_put_piece()`. Jika bisa diletakkan, 2 state baru akan dimasukkan ke dalam stack. State yang dimasukkan lebih dulu adalah state untuk mengeluarkan blok, state yang kedua adalah state untuk meletakkan blok dan berpindah ke blok baru. Karena sifat Stack adalah LIFO (Last In First Out), maka state yang akan dikunjungi lebih dulu adalah state meletakkan blok dan berpindah ke blok baru, setelah itu state ini akan mengunjungi state baru lainnya atau berhenti jika semua blok telah diletakkan. Jika tidak bisa diletakkan, algoritma akan mencoba orientasi lain.

Setelah semua blok berhasil diletakkan, algoritma akan memeriksa apakah masih terdapat petak yang kosong menggunakan `empty_spaces()` dan apakah semua blok berhasil diletakkan menggunakan `count_unique()`. Jika kedua syarat terpenuhi, maka

solusi dianggap benar dan algoritma berhenti. Selain itu, algoritma akan mencoba mencari solusi baru.

Banyaknya percobaan yang dilakukan dihitung sebagai banyaknya leaves yang ada pada pohon rekursi yang terbentuk dari pencarian menggunakan algoritma brute force hingga suatu solusi ditemukan atau semua kemungkinan telah dicoba.

## Warna

Constants.java menyimpan nilai-nilai RGB dan ANSI dari 26 warna berbeda yang akan digunakan untuk melakukan output pada CLI dan juga output to image.

## Output To Text

```
public void save_board_text(String file_name) throws
IOException {
    if (file_name == null || file_name.isEmpty()) {
        throw new IllegalArgumentException("File name cannot be
null or empty");
    }

    try (BufferedWriter writer = new BufferedWriter(new
FileWriter(file_name))) {
        for (int i = 0; i < rows; i++) {
            writer.write(data[i]);
            writer.newLine();
        }
    }
}
```

Fungsi save\_board\_text() akan menyimpan kondisi board saat ini pada file .txt dengan nama file\_name. Fungsi memanfaatkan BufferedWriter dari Java untuk melakukan write to file.

## Output To Image

```
public void save_board_image(String file_name) throws
IOException {
```

```

        if (file_name == null || file_name.isEmpty()) {
            throw new IllegalArgumentException("File name cannot be
null or empty");
        }

        int scale = 64;

        BufferedImage image = new BufferedImage(scale * rows, scale
* cols, BufferedImage.TYPE_INT_RGB);

        for(int i = 0; i < rows; ++i){
            for(int j = 0; j < cols; ++j){
                for(int k = 0; k < scale; ++k){
                    for(int l = 0; l < scale; ++l){
                        char ch = data[i][j];
                        image.setRGB(scale * i + k, scale * j + l,
Constants.get_color(ch));
                    }
                }
            }
        }

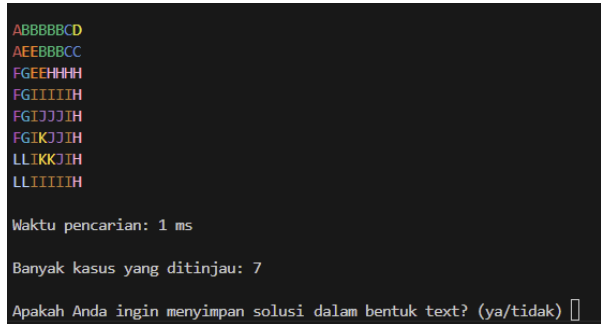
        File output_file = new File(file_name);
        ImageIO.write(image, "jpg", output_file);
    }

```

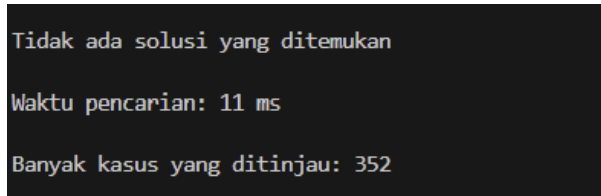
Fungsi `save_board_image()` akan menyimpan kondisi board saat ini pada file `.jpg` dengan nama `file_name`. Fungsi memanfaatkan `BufferedImage` dari Java untuk menyimpan dalam bentuk gambar. Setiap satu petak pada board akan menjadi suatu persegi 64 x 64 dengan warna yang disesuaikan dengan `get_color()`.

# Pengujian

## Test Case 1: DEFAULT (Terdapat Solusi)

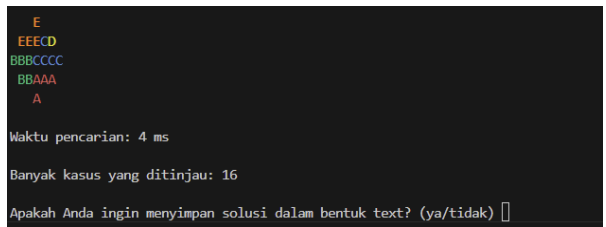
Masukan	Keluaran
8 8 12 DEFAULT A A BBBBB BBB C CC D EE EE FFFF GGGG HHHH H H H H H IIII I  I I  I I  I IIII JJJ JJ J KK K LL LL	

## Test Case 2: DEFAULT (Tidak Terdapat Solusi)

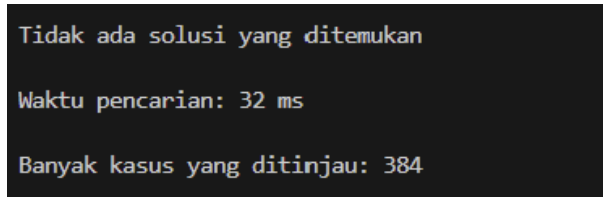
Masukan	Keluaran
6 6 5 DEFAULT A A AAAAA A	

A BBBBBB CC CCC DD DDD EE EEE E EEE EE	
--	--

### Test Case 3: CUSTOM (Terdapat Solusi)

Masukan	Keluaran
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	

### Test Case 4: CUSTOM (Tidak Terdapat Solusi)

Masukan	Keluaran
6 6 5 DEFAULT A A AAAAA A A BBBBBB CC CCC	

DD DDD EE EEE E EEE EE	
--	--

#### Test Case 5: Ukuran Board Tidak Valid

Masukan	Keluaran
2 0 3 DEFAULT AAAA AA B CCC	Invalid: Ukuran papan harus lebih dari 0.

#### Test Case 6: Banyak Blok Tidak Sesuai

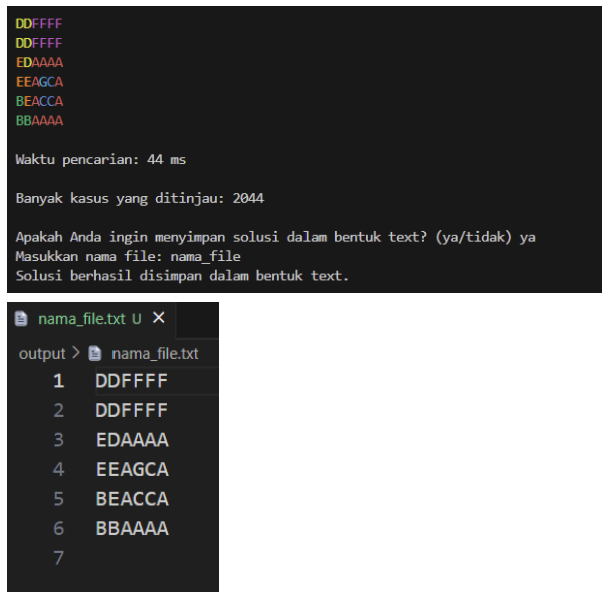
Masukan	Keluaran
6 6 9 DEFAULT AA BBBB CC DD DDDDDD EEE FFF FFFFFF GG G H HH I JJJ	Invalid: Banyak bentuk blok tidak sesuai.

#### Test Case 7: Terdapat Karakter Berulang

Masukan	Keluaran
5 5 7 DEFAULT A	Invalid: Kode blok tidak boleh sama.


AA B BB C CC D DD EE EE E FF FF F AAA	
--	--

#### Test Case 8: Output To Text

Masukan	Keluaran
6 6 7 DEFAULT AAAA A A A A AAAA B BB C CC DDD DD EE EE FFFF FFFF G	 <pre> DDFFFF DDFFFF EDAAAA EEAGCA BEACCA BBAAAA  Waktu pencarian: 44 ms  Banyak kasus yang ditinjau: 2044  Apakah Anda ingin menyimpan solusi dalam bentuk text? (ya/tidak) ya Masukkan nama file: nama_file Solusi berhasil disimpan dalam bentuk text.  nama_file.txt U x output &gt; nama_file.txt 1 DDFFFF 2 DDFFFF 3 EDAAAA 4 EEAGCA 5 BEACCA 6 BBAAAA 7 </pre>

#### Test Case 9: Output To Image

Masukan	Keluaran
---------	----------

6 6 7 DEFAULT AAAA A A A A AAAA B BB C CC DDD DD EE EE FFFF FFFF G	<pre> DDFFFF EDAAAA EEAGCA BEACCA BBAAAA  Waktu pencarian: 44 ms  Banyak kasus yang ditinjau: 2044  Apakah Anda ingin menyimpan solusi dalam bentuk text? (ya/tidak) ya Masukkan nama file: nama_file Solusi berhasil disimpan dalam bentuk text.  Apakah Anda ingin menyimpan solusi dalam bentuk image? (ya/tidak) ya Masukkan nama file: nama_file Solusi berhasil disimpan dalam bentuk image. </pre> 
--	---

#### Test Case 10: Jenis Papan Tidak Valid

Masukan	Keluaran
2 5 3 DEFAULT AAAA AA B CCC	<pre> Invalid: Tipe papan tidak diketahui. </pre>



## 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	✓	

Tautan Repositori: [https://github.com/BP04/Tucil1\\_13523067](https://github.com/BP04/Tucil1_13523067)