

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Peter Wongsoredjo - 13523039^{1,2}

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

13523039@itb.ac.id, peterwongsoredjo@gmail.com

A. Deskripsi Permasalahan

IQ Puzzler Pro adalah permainan papan dan balok, dimana tujuan dari permainan ini adalah untuk mengisi seluruh papan dengan piece yang telah tersedia. Dengan demikian, komponen dari permainan IQ Puzzler Pro terdiri dari dua hal, yaitu *Board* (papan) yang menjadi tempat bagi para pemain untuk harus mengisi seluruh area yang tersedia dengan blok-blok yang ada, serta *Piece* (blok) yang memiliki bentuk unik dan semua piece yang tersedia haruslah digunakan untuk menyelesaikan puzzle.

Permainan dimulai dengan papan yang kosong, dimana blok puzzle harus diletakkan sedemikian sehingga tidak ada blok yang bertumpang tindih. Setiap blok puzzle dapat dirotasikan maupun dicerminkan untuk menyelesaikan puzzle. Permainan dinyatakan selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan.

B. Algoritma Brute Force

Algoritma Brute Force adalah pendekatan yang lurus atau lempeng untuk memecahkan suatu persoalan. Hal ini berarti mencoba seluruh kemungkinan hingga akhirnya menemukan hasil yang benar. Dalam permainan IQ Puzzler Pro, Algoritma Brute Force digunakan untuk mencoba setiap kemungkinan, dengan setiap kemungkinan kombinasi balok / piece yang ada. Algoritma bermula dengan menaruh suatu blok pada papan terlebih dahulu, kemudian dilanjutkan dengan penempatan balok lainnya, hingga terjadi satu antara dua hal, papan terpenuhi dengan seluruh blok puzzle yang ada, atau tidak terdapat ruang untuk diletakkannya satu blok lagi, sehingga memaksa pengguna untuk mundur dan melepas blok sebelumnya, dan mencoba konfigurasi lainnya.

C. Struktur Data

Matrix

Dalam menyelesaikan permainan IQ Puzzler Pro dengan algoritma brute force, kedua komponen utama, yaitu papan dan blok di dalamnya dapat kita analogikan seperti suatu matrix dua dimensi, dimana papan dan juga blok puzzle memiliki elemen yang berupa titik (.) ataupun suatu karakter (char).

```
public class Matrix {  
    protected char[][] ELMT;  
    protected int rows;  
    protected int cols;  
  
    public void CreateMatrix(int y, int x) {  
        rows = y;  
        cols = x;  
        ELMT = new char[y][x];  
    }  
}
```

```

        for (int i = 0; i < y; i++){
            for (int j = 0; j < x; j++){
                ELMT[i][j] = '.';
            }
        }
    }

    public char getElement(int row, int col) {
        return ELMT[row][col];
    }

    public int getRow(){
        return rows;
    }

    public int getCol(){
        return cols;
    }

    public int getLastRowIdx() {
        return getRow() - 1;
    }

    public int getLastColIdx() {
        return getCol() - 1;
    }

    public boolean isRowValid(int i) {
        return i >= 0 && i <= getLastRowIdx();
    }

    public boolean isColValid(int i) {
        return i >= 0 && i <= getLastColIdx();
    }

    public void setElement(int row, int col, char c) {
        if (isRowValid(row) && isColValid(col)) {
            ELMT[row][col] = c;
        }
    }

    public void printMatrix() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(ELMT[i][j] + " ");
            }
            System.out.println();
        }
    }

    public boolean isEqual(Matrix m){
        if (rows != m.getRow() || cols != m.getCol()){
            return false;
        }
        for (int i = 0; i < rows; i++){
            for (int j = 0; j < cols; j++){
                if (ELMT[i][j] != m.getElement(i, j)){

```

```

        return false;
    }
}
return true;
}
}

```

Board (Papan)

Board atau papan yang digunakan sebagai daerah penempatan blok puzzle, diibaratkan sebagai suatu matrix, yang memiliki dua fungsi dasar yaitu, untuk mengecek apabila seluruh area pada papan sudah terpenuhi, dan untuk mencetak dirinya sendiri dengan setiap blok memiliki warna yang berbeda satu dengan lainnya (ConsoleColors tidak dimasukkan di dalam dokumen dan tersedia di repository).

```

public class Board extends Matrix {
    public void CreateBoard(int x, int y) {
        super.CreateMatrix(x, y);
    }

    public boolean isBoardFull(){
        for (int i = 0; i < getRow(); i++){
            for(int j = 0; j < getCol(); j++){
                if (getElement(i, j) == '.'){
                    return false;
                }
            }
        }
        return true;
    }

    public void printColoredMatrix() {
        for (int i = 0; i < getRow(); i++) {
            for (int j = 0; j < getCol(); j++) {
                char pieceChar = getElement(i, j);
                if (pieceChar == '#') {
                    System.out.print(" ");
                } else if (pieceChar != '.') {
                    System.out.print(getColorForChar(pieceChar) + pieceChar + " " + ConsoleColors.RESET);
                } else {
                    System.out.print(pieceChar + " ");
                }
            }
            System.out.println();
        }
    }

    private String getColorForChar(char pieceChar) {
        if (pieceChar == '.') return ConsoleColors.RESET;

        int colorIndex = (pieceChar - 'A') % ConsoleColors.COLORS.length;
        return ConsoleColors.COLORS[colorIndex];
    }
}

```

Piece (Blok)

Blok puzzle dalam permainan, dapat kita ibaratkan juga seperti suatu matrix, dimana dengan bentuknya yang unik, bagian yang tidak terisi dalam matrix blok, akan terisi dengan '.'. Blok puzzle ini dapat juga dirotasikan dan juga dicerminkan dengan menggunakan fungsi berikut.

```
import java.util.List;

public class Piece extends Matrix {
    public void CreatePiece(List<String> piecelines) {
        int rows = piecelines.size();
        int maxCols = 0;
        for (String row : piecelines) {
            maxCols = Math.max(maxCols, row.length());
        }

        super.CreateMatrix(rows, maxCols);

        for (int i = 0; i < rows; i++) {
            String row = piecelines.get(i);
            for (int j = 0; j < maxCols; j++) {
                if (j < row.length()) {
                    ELMT[i][j] = row.charAt(j);
                } else {
                    ELMT[i][j] = '.';
                }
            }
        }
    }

    public void rotate90(Piece pIn) {
        int newRow = pIn.getCol();
        int newCol = pIn.getRow();

        super.CreateMatrix(newRow, newCol);
        for (int i = 0; i < pIn.getRow(); i++) {
            for (int j = 0; j < pIn.getCol(); j++) {
                ELMT[j][newCol - 1 - i] = pIn.getElement(i, j);
            }
        }
    }

    public void mirror(Piece pIn) {
        super.CreateMatrix(pIn.getRow(), pIn.getCol());

        for (int i = 0; i < pIn.getRow(); i++) {
            for (int j = 0; j < pIn.getCol(); j++) {
                ELMT[i][pIn.getCol() - 1 - j] = pIn.getElement(i, j);
            }
        }
    }
}
```

D. Implementasi Program

Implementasi algoritma brute force pada permainan dalam program ini menggunakan pendekatan rekursif dengan melakukan backtracking untuk mencoba seluruh konfigurasi sebelum akhirnya menemukan solusi untuk permainan. Langkah penyelesaian pada algoritma brute force dilakukan seperti berikut.

Cari Sel Kosong Pertama

```
private int[] findFirstEmptyCell(){
    int[] cell = new int[2];
    for (int i = 0; i < board.getRow(); i++) {
        for (int j = 0; j < board.getCol(); j++) {
            if (board.getElement(i, j) == '.') {
                cell[0] = i;
                cell[1] = j;
                return cell;
            }
        }
    }
    return null;
}
```

Fungsi findFirstEmptyCell berfungsi untuk mencari sel kosong (.) pertama pada board, sel kosong ini yang nantinya akan menjadi titik awal dimana Piece akan dicoba untuk di tempatkan. Pencarian sel kosong pertama dilakukan secara sekuensial dari kiri ke kanan, mulai dari baris paling atas. Fungsi nantinya akan mengembalikan array dari integer, dengan elemen pertama pada integer berupa baris sel kosong, dan elemen kedua berupa kolom.

Penyesuaian Titik Mulai Piece

```
private int[] adjustStartingPosition(Piece piece, int startRow, int startCol) {
    for(int j = 0; j < piece.getCol(); j++){
        if(piece.getElement(0, j) == '.'){
            if(startCol > 0){
                startCol--;
            }
        } else {
            break;
        }
    }
    return new int[]{startRow, startCol};
}
```

Untuk suatu Piece ataupun bentuk transformasi Piece tersebut, yang bermulai (elemen paling kiri atas) dengan ruang hampa (.), diperlukan penyesuaian, dikarenakan elemen tersebut dapat ditanggihkan di atas Piece lainnya pada board. Dengan melakukan pemindahan kolom permulaan ke kiri sesuai dengan ruang hampa (.) yang berkesinambungan di kanan titik pada baris pertama, kita dapat menyesuaikan Piece agar dalam penempatannya dapat sesuai dengan Piece lainnya yang sudah terdapat dalam Papan permainan. Fungsi mengembalikan array dengan elemen pertama berupa baris awal penempatan Piece, dan elemen kedua berupa kolom awal penempatan Piece.

Validasi Penempatan Blok

```
private boolean isPlacementValid(Piece piece, int startRow, int startCol){
    // Geser dulu wak
    int[] adjustedPosition = adjustStartingPosition(piece, startRow, startCol);
    startRow = adjustedPosition[0];
    startCol = adjustedPosition[1];

    // Cek nabrak dan kalo keluar board
    for(int i = 0; i < piece.getRow(); i++){
        for(int j = 0; j < piece.getCol(); j++){
            if(piece.getElement(i, j) == '.'){
                continue;
            }
            if(!board.isRowValid(startRow + i) || !board.isColValid(startCol + j)){
                return false;
            }
            if(board.getElement(startRow + i, startCol + j) != '.' && board.getElement(startRow + i,
startCol + j) != '#'){
                return false;
            }
            if(board.getElement(startRow + i, startCol + j) == '#'){
                return false;
            }
        }
    }
    return true;
}
```

Setelah menemukan sel kosong pertama dan menyesuaikan titik permulaan penempatan Piece, selanjutnya program akan memeriksa apabila Piece dapat ditempatkan pada lokasi tersebut. Dilakukan iterasi sebanyak dimensi Piece, dengan memastikan apabila:

1. Potongan berada dalam batas papan, baik secara default (dimensi papan), maupun dengan papan custom, dengan tidak memperbolehkan penempatan jika char Piece bertemu dengan '#'
2. Tidak ada potongan yang tumpang tindih dengan potongan lain

Apabila Piece mampu untuk memenuhi syarat tersebut, maka fungsi akan mengembalikan true yang berarti Piece dapat ditempatkan pada daerah tersebut.

Penempatan Blok pada Papan

```
private void placePiece(Piece piece, int startRow, int startCol){
    for(int i = 0; i < piece.getRow(); i++){
        for(int j = 0; j < piece.getCol(); j++){
            if(piece.getElement(i, j) != '.'){
                board.setElement(startRow + i, startCol + j, piece.getElement(i, j));
            }
        }
    }
}
```

Suatu potongan (Piece) yang telah divalidasi, kemudian akan ditempatkan pada papan permainan. Setiap karakter yang bukan merupakan . dalam potongan akan dipindahkan ke papan menggunakan fungsi setElement pada board.

Menghapus Piece yang Tidak Sesuai pada Papan

```
private void removePiece(Piece piece, int startRow, int startCol){
    for(int i = 0; i < piece.getRow(); i++){
        for(int j = 0; j < piece.getCol(); j++){
            if(piece.getElement(i, j) != '.' && board.getElement(startRow + i, startCol + j) != '#') {
                board.setElement(startRow + i, startCol + j, '.');
            }
        }
    }
}
```

Apabila kombinasi konfigurasi yang bermulaikan suatu Piece tidak dapat dilanjutkan / menyelesaikan permainan (dead end), maka untuk mencoba konfigurasi pada Piece lainnya, Piece awal akan dihapus dari board, dengan menghapus semua karakter Piece dari papan, mengembalikannya menjadi '.'. Dengan papan custom, harus dipastikan bahwa ketika menghapus suatu piece dengan dimensi MxN, tidak mengubah '#' menjadi '.', sehingga ditambahkan kondisional pada fungsi.

Mencoba Seluruh Kombinasi Transformasi Blok

```
public List<Piece> getTransformations(Piece original){
    List<Piece> transformations = new ArrayList<Piece>();
    transformations.add(original);

    Piece temp = new Piece();
    temp.rotate90(original);
    for(int i = 0; i < 3; i++){
        if (!containsEqualPiece(transformations, temp)) {
            transformations.add(temp);
        }
        Piece temp2 = new Piece();
        temp2.rotate90(temp);
        temp = temp2;
    }

    Piece mirrored = new Piece();
    mirrored.mirror(original);
    if (!containsEqualPiece(transformations, mirrored)) {
        transformations.add(mirrored);
    }

    Piece mirroredrotated = new Piece();
    mirroredrotated.rotate90(mirrored);
    for(int i = 0; i < 3; i++){
        if (!containsEqualPiece(transformations, mirroredrotated)) {
            transformations.add(mirroredrotated);
        }
        Piece mirroredrotated2 = new Piece();
        mirroredrotated2.rotate90(mirroredrotated);
        mirroredrotated = mirroredrotated2;
    }

    return transformations;
}
```

```

private boolean containsEqualPiece(List<Piece> list, Piece piece) {
    for (Piece p : list) {
        if (p.isEqual(piece)) {
            return true;
        }
    }
    return false;
}

```

Program akan mencoba seluruh bentuk potongan yang mungkin melalui rotasi dan juga pencerminan. Untuk setiap Pieces, semua bentuk transformasi yang dimilikinya akan dimasukkan ke dalam List transformations sehingga nantinya dapat diiterasikan oleh program. Setiap transformasi yang terdapat pada List transformations dipastikan unik satu dari lainnya, untuk menghindari adanya transformasi yang menghasilkan bentuk yang sama dari transformasi lainnya. Hal ini dikarenakan ketika bermain, transformasi yang sama, tidak akan memberikan kombinasi konfigurasi keseluruhan yang berbeda.

Backtracking dan DFS

```

public boolean solve(int pieceIndex) {
    if (pieceIndex >= totalPieces) {
        if (board.isBoardFull()) {
            endTime = System.currentTimeMillis();
            board.printColoredMatrix();
            System.out.println("Waktu Pencarian: " + (endTime - startTime) + " ms");
            System.out.println("Total kasus yang ditinjau: " + casesTried);
            System.out.println("Solusi ditemukan! Apakah Anda ingin menyimpan solusi? (y/n)");
            Scanner scanner = new Scanner(System.in);
            String response = scanner.next().toLowerCase();
            if (response.equals("y")) {
                saveBoardToFile("solution.txt");
            }
            else if (response.equals("n")) {
            }
            else {
                System.out.println("Input tidak valid, solusi tidak disimpan.");
            }
            return true;
        }
        return false;
    }

    int[] firstEmptyCell = findFirstEmptyCell();
    if (firstEmptyCell == null) {
        return false;
    }

    int startingRow = firstEmptyCell[0];
    int startingCol = firstEmptyCell[1];

    for (int i = 0; i < pieces.size(); i++) {
        Piece currentPiece = pieces.get(i);
        List<Piece> transformations = getTransformations(currentPiece);

        for (Piece transformation : transformations) {
            casesTried++;

```



```

int[] adjustedPosition = adjustStartingPosition(transformation, startingRow, startingCol);
int adjustedRow = adjustedPosition[0];
int adjustedCol = adjustedPosition[1];

if (isPlacementValid(transformation, adjustedRow, adjustedCol)) {
    placePiece(transformation, adjustedRow, adjustedCol);
    pieces.remove(i);

    if (solve(pieceIndex + 1)) {
        return true;
    }

    removePiece(transformation, adjustedRow, adjustedCol);
    pieces.add(i, currentPiece);
}
}
}

return false;
}

```

Metode Backtracking dan DFS pada fungsi solve menjadi inti dari algoritma brute force yang berbasis rekursi untuk mencari solusi permainan. Rekursi dengan DFS berarti mengeksplorasi konfigurasi kemungkinan hingga bagian paling dalam sebelum berpindah ke kemungkinan lain. Rekursi bermula dengan basis, dimana fungsi memeriksa apabila semua potongan sudah ditempatkan ($\text{pieceIndex} \geq \text{totalPieces}$). Jika kondisi terpenuhi, maka fungsi akan mengecek apabila papan sudah penuh atau belum. Jika papan penuh, maka solusi ditemukan. Fungsi akan mencatat waktu eksekusi, mencetak papan dengan seluruh Piece, dan mencetak total kasus yang telah diperiksa, dan pengguna akan diberi pilihan untuk menyimpan solusi. Jika papan belum penuh, maka algoritma akan melakukan backtrack, karena masih terdapat kesalahan dalam konfigurasi Pieces.

Rekursi dilakukan dengan menjalankan kembali seluruh langkah sebelumnya, dengan mencari sel kosong pertama, melakukan iterasi semua potongan dan transformasi, menyesuaikan posisi peletakan potongan, validasi dan akhirnya penempatan potongan. Piece yang telah diletakkan akan dihilangkan dari List Pieces, dan kemudian rekursi akan dipanggil, dengan board yang baru, dengan List Pieces tanpa Piece sekarang. Apabila kemudian rekursi tidak memiliki solusi, Piece sekarang akan dihilangkan, dan dikembalikan ke dalam List Pieces. Jika tidak ada cara untuk menyusun potongan dan tidak ada solusi, algoritma mengembalikan false, dan program harus kembali ke langkah sebelumnya untuk mencoba kemungkinan lain.

Bonus: Custom Board

FileScanner.java:

```

if(mode.equals("CUSTOM")){
    Board customBoard = new Board();
    customBoard.CreateMatrix(boardRows, boardCols);

    for(int i = 0; i < boardRows; i++){
        String line = reader.readLine();
        for(int j = 0; j < boardCols; j++){
            char c = line.charAt(j);
            if(c == '.'){
                customBoard.setElement(i, j, '#');
            }
        }
    }
}

```

```

    }
    else if(c == 'X'){
        customBoard.setElement(i, j, '.');
    }
    else{
        customBoard.setElement(i, j, c);
    }
}

board = customBoard;
}
else{
    Board defaultBoard = new Board();
    defaultBoard.CreateMatrix(boardRows, boardCols);
    board = defaultBoard;
}

```

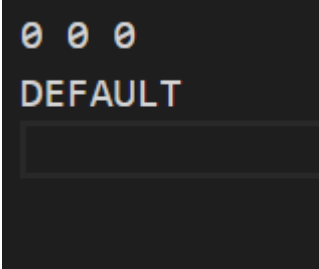
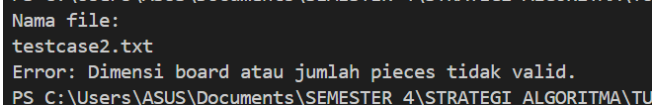
Sebagai bentuk pengerjaan bonus, pengguna dapat memberikan board yang khusus dengan memberikan input CUSTOM. Pada mode input custom, papan dapat memiliki bentuk lain, dengan memberikan '.' sebagai daerah yang tidak dapat ditempati suatu Piece, dan 'X' sebagai daerah yang dapat ditempati. Program akan mengganti '.' dengan '#' sebagai penanda dari daerah yang tidak dapat dimainkan, dan 'X' dengan '.', sebagai daerah yang dapat ditempati, seperti pada kode awal. Dengan adanya '#' sebagai penanda daerah yang tidak dapat ditempati, program harus beradaptasi, lebih tepatnya pada fungsi validasi blok, dan menghilangkan blok, untuk tidak menempati maupun menghilangkan '#' dari papan, sehingga terdapat kondisional tambahan yang memperhatikan '#' pada board. Dengan demikian, Piece dapat diletakkan pada daerah dengan '.' pada board.

E. Eksperimen

Case 1: Kasus 5x5 DEFAULT (ada solusi)

Test Case	Output
<pre> 5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG </pre>	<pre> Nama file: testcase1.txt A B C C A A B D C E E E D D E E F F F G G G F F Waktu Pencarian: 4 ms Total kasus yang ditinjau: 725 Solusi ditemukan! Apakah Anda ingin menyimpan solusi? (y/n) y Solusi disimpan ke file solution.txt PS C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUCIL 1\Tuci </pre>

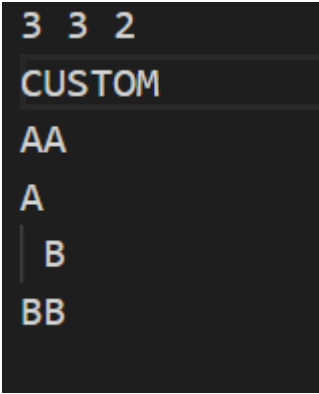
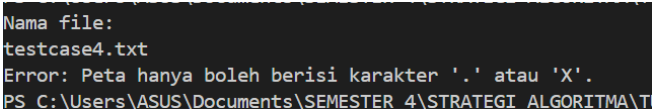
Case 2: Dimensi Tidak Valid (Tidak ada solusi)

Test Case	Output
	

Case 3: Kasus 4x4 DEFAULT, Piece Melebihi Board (Tidak ada Solusi)

Test Case	Output
	

Case 4: Tidak Terdapat Custom Board (Tidak ada solusi)

Test Case	Output
	

Case 5: Kasus 4x6 DEFAULT (ada solusi)

Test Case	Output
<pre>4 6 7 DEFAULT A AA B BBB BB C D DD EEEE FF FF F GG</pre>	<pre>Nama file: testcase5.txt A C B B D D A A B B B D F F F B G G F F E E E E Waktu Pencarian: 8 ms Total kasus yang ditinjau: 26 Solusi ditemukan! Apakah Anda ingin menyimpan solusi? (y/n) n PS: C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUCTI</pre>

Case 6: CUSTOM Board Tidak Mencukupi (Tidak ada solusi)

Test Case	Output
<pre>5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX ..XXXX. ...X... A AAA BB BBB CCCC C D EEE E</pre>	<pre>PS C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUGAS 1> Nama file: testcase6.txt Solusi tidak ditemukan. Total kasus yang ditinjau: 5582 PS C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUGAS 1></pre>

Case 7: CUSTOM Board (Ada solusi)

Test Case	Output
<pre>5 5 6 CUSTOM XX.XX XX.XX ..X.. XX.XX XX.XX AA AA BB C C C DD E E FF FF</pre>	<pre>PS C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUGAS 1> Nama file: testcase7.txt A A B B A A D D C E C F F C E F F Waktu Pencarian: 0 ms Total kasus yang ditinjau: 17 Solusi ditemukan! Apakah Anda ingin menyimpan solusi? (y/n) n PS C:\Users\ASUS\Documents\SEMESTER 4\STRATEGI ALGORITMA\TUGAS 1></pre>

F. Lampiran

Repository Github: https://github.com/PeterWongsoredjo/Tucil1_13523039

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	✓	