

Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II tahun 2024/2025
Penyelesaian IQ Puzzler Pro dengan Algoritma
Brute Force



Oleh:

Arlow Emmanuel Hergara

Teknik Informatika

13523161

Algoritma

Algoritma yang digunakan untuk menyelesaikan IQ Puzzler Pro adalah Brute Force dengan cara rekursi. Secara singkat, algoritma ini mencoba untuk menggunakan semua blok yang diberikan. Algoritma akan meletakkan blok pertama di tempat pertama yang ada dari kiri atas. Setelah itu, algoritma akan meletakkan blok berikut pada tempat pertama yang muat dimasuki dari kiri atas. Proses ini mengulang hingga terdapat blok yang tidak muat di mana-mana. Jika hal tersebut terjadi, algoritma akan kembali ke blok sebelumnya ($i - 1$) dan memindahkan blok tersebut ke tempat yang muat berikutnya. Algoritma kemudian mencoba lagi meletakkan blok yang ingin diletakkan sebelumnya (i). Jika blok tetap tidak dapat diletakkan walaupun blok sebelumnya sudah menempati semua kemungkinan tempat yang muat, maka blok sebelumnya ($i - 1$) diputar 90 derajat dan proses peletakkan diulangi lagi. Jika setelah empat kali putaran tetap tidak dapat meletakkan blok yang baru (i), blok dibalik secara horizontal dan proses peletakkan serta rotasi diulang lagi. Jika tetap tidak dapat diletakkan, proses akan mengular ke blok sebelumnya ($i - 2$) yang kemudian diposisikan ulang. Proses ini dilakukan hingga semua blok dapat muat atau blok pertama sudah dicoba diletakkan di semua kemungkinan posisi dan orientasi. Jika setelah proses tersebut selesai tidak terdapat blok yang tersisa dan semua tempat terpenuhi, dikatakan bahwa solusi ditemukan, sebaliknya jika tidak.

Source Code

```
package com.arlow.iqsolverpro.solver;

import com.arlow.iqsolverpro.game.GameInstance;
import com.arlow.iqsolverpro.game.Piece;
import com.arlow.iqsolverpro.game.Plane;

public class BruteforceSolver implements Solver {
    private GameInstance game;
    private Piece[] pieces;
    private int delay;
    private SolveStats result;

    public BruteforceSolver(GameInstance game, int delay) {
        this.game = game;
        this.delay = delay;
        this.result = new SolveStats();

        this.pieces = new Piece[game.availablePieces.size()];
        for (int i = 0; i < game.availablePieces.size(); i++) {
            this.pieces[i] = game.availablePieces.get(i);
        }
    }

    public SolveStats Solve() {
        long startTime = System.currentTimeMillis();

        result.solutionExists = RecursiveSolve(0) && game.board.GetEmptyPositions().length <=
0;
        result.timeTakenInMs = System.currentTimeMillis() - startTime;

        return result;
    }

    private boolean RecursiveSolve(int index) {
        if (index >= pieces.length) {
            return true;
        }
    }
}
```

```

        int[] emptySlots = game.board.GetEmptyPositions();

        for (boolean flipped = false; !flipped; flipped = true) {
            for (int rotations = 0; rotations < 4; rotations++) {
                for (int i = 0; i < emptySlots.length; i++) {
                    pieces[index].position = emptySlots[i];

                    for (Plane plane : game.board.GetPlanesOnPosition(emptySlots[i])) {
                        if (game.PlacePiece(pieces[index], plane)) {
                            result.stepsTaken++;

                            if (delay > 0) {
                                try {
                                    Thread.sleep(delay);
                                } catch (Exception e) {
                                    System.out.println("[WARNING]: solver thread interrupted
during sleep");
                                }
                            }

                            if (RecursiveSolve(index + 1)) {
                                return true;
                            }

                            game.RemovePiece(pieces[index]);
                        }
                    }

                    pieces[index].RotateClockwise();
                }

                pieces[index].FlipHorizontal();
            }

            return false;
        }
    }
}

```

```

package com.arlow.iqsolverpro.game;

public class Piece {
    public char key;
    public boolean[][] shape;
    public int position = -1;
    public int pivotX;
    public int pivotY;

    public Piece Clone() {
        Piece clone = new Piece();
        clone.shape = new boolean[shape.length][shape[0].length];
        clone.position = position;
        clone.pivotX = pivotX;
        clone.pivotY = pivotY;

        return clone;
    }

    public void RotateClockwise() {
        boolean[][] newShape = new boolean[shape[0].length][shape.length];

        for (int j = 0; j < shape.length; j++) {
            for (int i = 0; i < shape[j].length; i++) {
                newShape[i][shape.length - 1 - j] = shape[j][i];
            }
        }

        for (int i = 0; i < shape[0].length; i++) {
            if (shape[0][i]) {

```

```

        pivotX = 0;
        pivotY = i;
    }
}

shape = newShape;
}

public void FlipHorizontal() {
    boolean[][] newShape = new boolean[shape.length][shape[0].length];

    for (int i = 0; i < shape.length / 2; i++) {
        newShape[i] = shape[shape.length - 1 - i];
    }

    for (int i = 0; i < shape[0].length; i++) {
        if (shape[0][i]) {
            pivotX = 0;
            pivotY = i;
        }
    }

    shape = newShape;
}
}

```

```

package com.arlow.iqsolverpro.game;

import java.util.ArrayList;
import java.util.List;

public class RectangularBoard implements Board {
    public Piece[] slots;
    public int width;
    public int length;

    public RectangularBoard(int width, int length) {
        this.width = width;
        this.length = length;

        slots = new Piece[width * length];
    }

    @Override
    public int[] GetEmptyPositions() {
        List<Integer> emptySlots = new ArrayList<Integer>();

        for (int i = 0; i < slots.length; i++)
        {
            if (slots[i] == null) emptySlots.add(i);
        }

        int[] emptySlotsArray = new int[emptySlots.size()];

        for (int i = 0; i < emptySlots.size(); i++)
        {
            emptySlotsArray[i] = emptySlots.get(i);
        }

        return emptySlotsArray;
    }

    @Override
    public Plane[] GetPlanesOnPosition(int position) {
        Plane plane = new Plane();
        plane.source = this;

        plane.positionMap = new int[length][width];

        for (int i = 0; i < width * length; i++)
        {
            plane.positionMap[i / width][i % width] = i;
        }

        return new Plane[] {plane};
    }
}

```

```

    }

    @Override
    public Piece GetPieceOnPosition(int position) {
        return slots[position];
    }

    @Override
    public boolean PlacePiece(Piece piece, Plane plane) {
        if (!plane.PieceFits(piece)) {
            return false;
        }

        int offsetX = -1;
        int offsetY = -1;

        for (int j = 0; j < plane.positionMap.length; j++) {
            for (int i = 0; i < plane.positionMap[j].length; i++) {
                if (plane.positionMap[j][i] == piece.position) {
                    offsetX = i;
                    offsetY = j;
                }
            }
        }

        for (int j = 0; j < piece.shape.length; j++) {
            for (int i = 0; i < piece.shape[j].length; i++) {
                int posX = offsetX + i - piece.pivotX;
                int posY = offsetY + j - piece.pivotY;

                if (piece.shape[j][i]) {
                    slots[plane.positionMap[posY][posX]] = piece;
                }
            }
        }

        return true;
    }

    @Override
    public boolean RemovePiece(Piece piece) {
        boolean pieceFound = false;

        for (int i = 0; i < slots.length; i++) {
            if (slots[i] == piece) {
                slots[i] = null;
                pieceFound = true;
            }
        }

        return pieceFound;
    }
}

```

```

package com.arlow.iqsolverpro.game;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

public class GameInstance {
    public enum EventType {
        PiecePlaced,
        PieceRemoved,
        BoardFull
    }

    public List<Piece> availablePieces;
    public Board board;
}

```

```

private List<GameEventListener> listeners = new LinkedList<GameEventListener>();
private List<GameEventListener> listenerAddList = new LinkedList<GameEventListener>();
private List<GameEventListener> listenerRemoveList = new LinkedList<GameEventListener>();

public GameInstance(Piece[] pieces, Board board) {
    this.availablePieces = new ArrayList<Piece>(Arrays.asList(pieces));
    this.board = board;
}

public boolean PlacePiece(Piece piece, Plane plane) {
    if (board.PlacePiece(piece, plane)) {
        availablePieces.remove(piece);
        NotifyListeners(EventType.PiecePlaced, piece);

        if (board.GetEmptyPositions().length < 1) {
            NotifyListeners(EventType.BoardFull, null);
        }

        return true;
    }

    return false;
}

public boolean RemovePiece(Piece piece) {
    if (board.RemovePiece(piece)) {
        availablePieces.add(piece);
        NotifyListeners(EventType.PieceRemoved, piece);

        return true;
    }

    return false;
}

public boolean AddListener(GameEventListener listener) {
    if (listeners.contains(listener)) return false;

    listenerAddList.add(listener);

    return true;
}

public boolean RemoveListener(GameEventListener listener) {
    if (!listeners.contains(listener)) return false;

    listenerRemoveList.add(listener);

    return true;
}

private void NotifyListeners(EventType type, Piece piece) {
    for (int i = 0; i < listeners.size(); i++) {
        GameEventListener current = listeners.get(i);

        if (current.targetEventType == type) {
            current.Notify(piece);
        }
    }

    while (!listenerRemoveList.isEmpty()) {
        listeners.remove(listenerRemoveList.get(listenerRemoveList.size() - 1));
        listenerRemoveList.remove(listenerRemoveList.size() - 1);
    }

    while (!listenerAddList.isEmpty()) {
        listeners.add(listenerAddList.get(listenerAddList.size() - 1));
        listenerAddList.remove(listenerAddList.size() - 1);
    }
}
}

```

Testing

Input	Output	Valid
2 2 3 DEFAULT A BB C	AC BB Solving Results Solution found: Yes Time taken: 5ms Steps taken: 3	Ya
5 5 6 DEFAULT AAA A BB CCC C C D D EE E FF F FF	AAABB DACCC DEE.C FEF.C FFF.. Solving Results Solution found: No Time taken: 27ms Steps taken: 6	Ya
11 5 12 DEFAULT AAAA A B B BB B CCC C D DD EE EE E FFF F GGG G	AAAA. ABCCC .BC.D KBBDD .BEEJ .EE.J .EFFF GGGFL HHG.L .HH.L ..HI. Solving Results Solution found: No Time taken: 48ms Steps taken: 16	Tidak

HH HH H I III I JJJ JJ KK KK K L L LLL		
8 8 12 DEFAULT AAAAA BBB B B CCCC C DDDD D EEE E E FFF FF GG GG HHH HH I II II J JJJ J K KK KK L L LLL	AAAAABBB CCCC.B.B CDDDDDEEE GGDFFFE. GG..FFEI HHH.JJII ..HHJIIK LLLJJ..K Solving Results Solution found: No Time taken: 34ms Steps taken: 14	Tidak

4 4 5 DEFAULT AA A BB B CC C DD D EE EE	AABB A.B. CCDD CEED Solving Results Solution found: No Time taken: 5ms Steps taken: 6	Tidak
5 5 7 DEFAULT AA AA BB BB CC CC DD DD EE EE FF FF GG GG	AABBG AABBG CC.DD CC.DD EE.FF Solving Results Solution found: No Time taken: 7ms Steps taken: 10	Tidak
10 10 5 DEFAULT AAA A B B BB CCCC C DD D DDD EEE E E	AAABCCCC.. A..B.CDD.D EEEBB..DDD E.E..... Solving Results Solution found: No Time taken: 4ms Steps taken: 5	Ya

Repository

[Arlow5761/Tucil1_13523161](#)

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	V	
2	Program berhasil dijalankan	V	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan		V
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	V	
5	Program memiliki <i>Graphical User Interface</i> (GUI)	V	
6	Program dapat menyimpan solusi dalam bentuk file gambar		V
7	Program dapat menyelesaikan kasus konfigurasi <i>custom</i>		V
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		V
9	Program dibuat oleh saya sendiri	V	