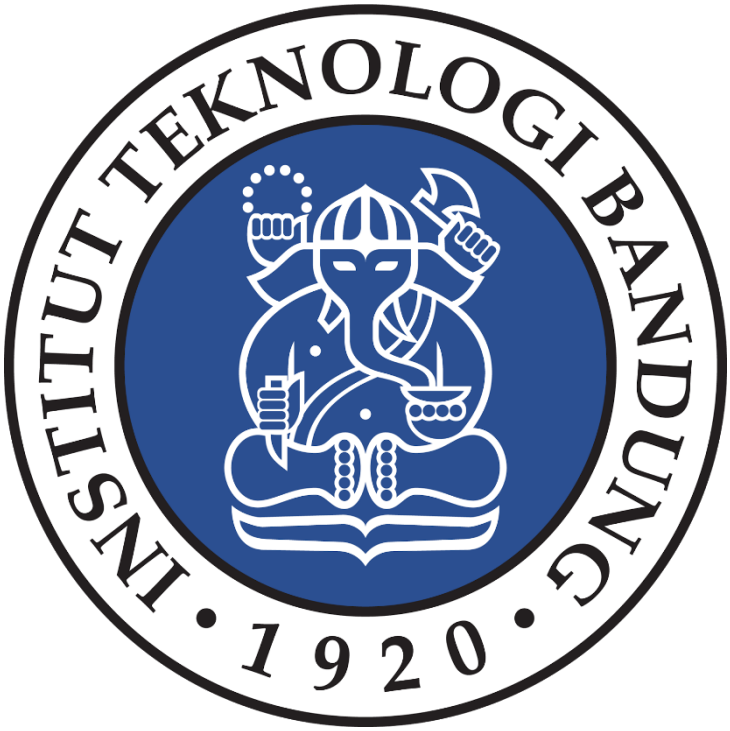
Laporan Tugas Kecil 3

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Mata Kuliah IF2211 Strategi Algoritma



Disusun oleh:

Muhammad Fahmi Irfan

13520152

K-02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 2 TAHUN 2021/2022

Bab 1 : Deskripsi Singkat

Program ini merupakan program untuk menyelesaikan permainan 15-Puzzle, yaitu permainan yang terdiri dari 15 kotak yang berisi angka dari 1 sampai 15 yang berada pada kotak 4x4. Permainan ini bertujuan untuk menyusun kotak-kotak tersebut agar tersusun berurutan seperti gambar pada di bawah ini.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

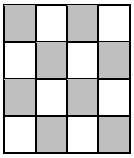
Gambar 1. Susunan akhir persoalan 15-Puzzle

Untuk menyelesaikan permainan tersebut, program ini menggunakan algoritma Branch and Bound.

Bab 2 : Cara Kerja Algoritma Branch and Bound

Berikut merupakan implementasi Algoritma Branch and Bound pada program ini.

1. Buat sebuah *node root* yang berisi matriks 4x4 yang merupakan kondisi awal puzzle (kotak kosong akan dianggap sebagai kotak 16)
2. Hitung Kurang(i) untuk tiap i dari 1 sampai 16, yaitu banyaknya kotak $j < i$ yang berada "setelah" kotak i (kotak x berada setelah kotak y didefinisikan sebagai keadaan di mana kotak x berada di posisi kotak x' yang seharusnya, kotak y berada di posisi kotak y' yang seharusnya, dan $x' > y'$) hitung $\Sigma_{i=1}^{16} Kurang(i) + X$, dengan X bernilai 0 jika kotak kosong berada pada daerah yang diarsir di bawah ini, dan 1 jika sebaliknya.



Gambar 2. Tabel untuk menentukan nilai X

3. Jika $\Sigma_{i=1}^{16} Kurang(i) + X$ bernilai ganjil, maka persoalan tidak bisa diselesaikan, hentikan algoritma. Jika genap, persoalan dapat diselesaikan.
4. Buat sebuah *priority queue* yang berisi node, di mana anggota *queue* yang memiliki cost paling kecil akan diproses lebih dulu.
5. Cari nilai *cost* untuk *node* yang sedang diproses. Nilai *cost* untuk node x dapat dicari menggunakan persamaan berikut
$$cost(x) = f(x) + \hat{g}(x)$$
Dengan $f(x)$ merupakan panjang lintasan dari root ke node x dan $\hat{g}(x)$ merupakan banyaknya kotak yang tidak menempati posisi yang benar.
6. Masukkan *node* ke *priority queue.*
7. Ambil satu *node* dalam *priority queue*.
8. Cek apakah *node* merupakan solusi. Dapat dicek dengan apakah $\hat{g}(x) = 0$ atau apakah $cost(x) = f(x)$ jika tidak menyimpan nilai $\hat{g}(x)$
9. Jika *node* solusi, tetapkan suatu batas B sehingga untuk *node* yang *cost*-nya lebih dari B tidak akan diproses. Bandingkan *cost node* ini dengan solusi yang sudah ada sebelumnya jika ada.

Jika *cost node* ini lebih kecil, maka *node* ini dapat menjadi solusi sementara. Jika sebelumnya belum ada solusi, maka *node* ini dapat menjadi solusi sementara.

10. Lakukan *expand* pada *node* yan sedang diproses, yaitu mencari langkah yang mungkin dari keadaan *puzzle* pada *node* tersebut, lalu keadaan *puzzle* setelah melakukan satu langkah tersebut akan menjadi *child* dari *node* yang sedang diproses. Kemudian, buang *node* dari *priority queue.*
11. Lakukan langkah 5 dan 6 untuk semua *child* yang baru saja dibuat, lalu kembali ke langkah 7 sampai tidak ada *node* yang dapat diambil dari *priority queue*.
12. *Node* solusi pada program ini hanya menyimpan satu langkah sebelum mencapai *node* tersebut, sehingga untuk mendapatkan semua solusi, lakukan *tracking* dari *node* solusi ke *node* akar dan simpan tiap-tiap langkah dalam suatu *list*. Kemudian, *reverse list* tersebut untuk mendapatkan langkah dari keadaan awal *puzzle* sampai *puzzle* selesai.

Bab 3 : Kode Program

Program dibagi menjadi empat program, yaitu *GUI.java, Tree.java, Node.java,* dan *Move.java.*

1. *File enum Move.java*
   File enum Move.java berisi kode berikut.

```
public enum Move {
    UP,
    DOWN,
    LEFT,
    RIGHT,
    NULL
}
```

2. *File class Node.java*
   File class Node.java berisi kode berikut.

```
import java.util.*;

public class Node {
    private List<Node> child;
    private Move move;
    private int[][] puzzle;
    private int n;
    private int depth;
    private int cost;
    private Node parent;
    private boolean isActive;
    private boolean isBounded;

    public Node(int[][] puzzle){
        System.out.println("Generating Root Node...");
        this.child = new ArrayList<Node>();
        this.move = Move.NULL;
        this.n = 4;
```

```
        this.depth = 0;
        this.puzzle = new int[n][n];
        for(int i=0; i<n; i++) System.arraycopy(puzzle[i], 0,
this.puzzle[i], 0, n);
        this.cost = this.costCounter();
        this.parent = null;
        this.isActive = false;
        this.isBounded = false;
    }

    public Node(Move m, int n, int d, int[][] puzzle, Node parent){
        System.out.println("Generating Node with depth " + d);
        this.child = new ArrayList<Node>();
        this.move = m;
        this.n = n;
        this.depth = d;
        this.puzzle = new int[n][n];
        for(int i=0; i<n; i++) System.arraycopy(puzzle[i], 0,
this.puzzle[i], 0, n);
        this.move(move);
        this.parent = parent;
        this.isActive = false;
        this.isBounded = false;
        System.out.println(getVoidIndex() + " " + getCost());
    }

    public Move getMove(){
        return this.move;
    }

    public int getCell(int idx){
        return this.puzzle[idx/this.n][idx%this.n];
    }

    public int getCell(int x, int y){
        return this.puzzle[x][y];
    }

    public int getCost(){
        return this.cost;
    }

    public int[][] getPuzzle(){
        return this.puzzle;
    }
    public int getDepth(){
        return this.depth;
    }
```

```
    public List<Node> getChildren(){
        return this.child;
    }
    public int Kurang(int i){
        int idx = 0;
        int res = 0;
        while(puzzle[idx/n][idx%n] != i){
            idx++;
        }
        while(idx<n*n){
            if(puzzle[idx/n][idx%n] < i) {res++;}
            idx++;
        }
        //System.out.println("Kurang " + i + " = " + res);
        return res;
    }
    public int kurangPlusX(){
        int c = 0;
        for(int i=0; i<n*n; i++){
            if(this.puzzle[i/n][i%n] == this.n*this.n && (i/n +
i%n)%2 == 1){
                c++;
                break;
            }
        }
        for(int i=1; i<=n*n; i++){
            c+=Kurang(i);
        }
        return c;
    }
    public boolean possibleChecker(){
        return kurangPlusX()%2 == 0;
    }

    public boolean getIsActive(){
        return this.isActive;
    }

    public Node getParent(){
        return this.parent;
    }

    public void setIsActive(boolean val){
        this.isActive = val;
    }

    public boolean getIsBounded(){
```

```
        return this.isBounded;
    }

    public void setBounded(){
        this.isBounded = true;
    }

    public boolean isSolution(){
        return this.depth == this.cost;
    }
    private int costCounter(){
        int c = this.depth;
        for(int i=0; i<this.n*this.n; i++){
            if(this.puzzle[i/this.n][i%this.n] != this.n*this.n &&
i+1 != this.puzzle[i/this.n][i%this.n]){
                c++;
            }
        }
        return c;
    }

    public int getVoidIndex(){
        int idx = 0;
        while(this.puzzle[idx/this.n][idx%this.n] != this.n*this.n){
            idx++;
        }
        return idx;
    }

    private boolean isValidMove(Move move){
        int voidX = getVoidIndex()/n;
        int voidY = getVoidIndex()%n;
        if(move == Move.UP){
            return voidX>0;
        }else if(move == Move.DOWN){
            return voidX<n-1;
        }else if(move == Move.LEFT){
            return voidY>0;
        }else if(move == Move.RIGHT){
            return voidY<this.n-1;
        }else return false;
    }

    public void move(Move move){
        int voidX = getVoidIndex()/n;
        int voidY = getVoidIndex()%n;
        if(move == Move.UP){
            int tmp = this.puzzle[voidX][voidY];
```

```
            this.puzzle[voidX][voidY] = this.puzzle[voidX-1][voidY];
            this.puzzle[voidX-1][voidY] = tmp;
            System.out.println(puzzle[voidX][voidY] + " " +
puzzle[voidX-1][voidY]);
        }else if (move == Move.DOWN){
            int tmp = this.puzzle[voidX][voidY];
            this.puzzle[voidX][voidY] = this.puzzle[voidX+1][voidY];
            this.puzzle[voidX+1][voidY] = tmp;
            System.out.println(puzzle[voidX][voidY] + " " +
puzzle[voidX+1][voidY]);
        }else if (move == Move.LEFT){
            int tmp = this.puzzle[voidX][voidY];
            this.puzzle[voidX][voidY] = this.puzzle[voidX][voidY-1];
            this.puzzle[voidX][voidY-1] = tmp;
            System.out.println(puzzle[voidX][voidY] + " " +
puzzle[voidX][voidY-1]);
        }else if (move == Move.RIGHT){
            int tmp = this.puzzle[voidX][voidY];
            this.puzzle[voidX][voidY] = this.puzzle[voidX][voidY+1];
            this.puzzle[voidX][voidY+1] = tmp;
            System.out.println(puzzle[voidX][voidY] + " " +
puzzle[voidX][voidY+1]);
        }
        this.cost = this.costCounter();
    }

    private Move oppositeMove(Move m){
        if(m == Move.DOWN) return Move.UP;
        if(m == Move.LEFT) return Move.RIGHT;
        if(m == Move.RIGHT) return Move.LEFT;
        if(m == Move.UP) return Move.DOWN;
        return Move.NULL;
    }
    private void generateChild(){
        Move[] moveset = Move.values();
        for(Move m : moveset){
            if(isValidMove(m) && this.move != oppositeMove(m)){
                Node nd = new Node(m, this.n, this.depth+1,
this.puzzle, this);
                this.child.add(nd);
            }
        }
    }

    public void expand(){
        System.out.println("Expanding Node...");
        if(this.isActive){
            generateChild();
```

```
        }
    }
}
```

3. *File class Tree.java*
   File class Tree.java berisi kode berikut.

```
import java.util.*;
import java.lang.Math;
import java.io.*;

public class Tree {
    private Node root;
    private boolean isPossible;
    private PriorityQueue<Node> pq;
    private List<Move> solution;
    private int costUpperBound;
    private int numOfNodesGenerated;
    private long start;
    private long end;

    public Tree(Node root){
        this.root = root;
        this.isPossible = this.root.possibleChecker();
        System.out.println("this is run");
        this.pq = new PriorityQueue<Node>(new NodeComparator());
        this.costUpperBound = Integer.MAX_VALUE;
        this.solution = new ArrayList<Move>();
        this.numOfNodesGenerated = 1;
    }

    public void start(){
        start = System.currentTimeMillis();
        if(isPossible){
            System.out.println("Generating Tree..");
            Node bestLastNode = null;
            pq.add(this.root);
            while(pq.peek() != null){
                Node nodeCheck = pq.poll();
                System.out.print(nodeCheck.getCost() + " ");
                System.out.print(nodeCheck.getMove() + " ");
                System.out.print(nodeCheck.getVoidIndex() + " ");
                if(nodeCheck.isSolution()){
                    this.costUpperBound =
Math.min(this.costUpperBound,nodeCheck.getCost());
                    if(bestLastNode == null || bestLastNode.getCost()
> nodeCheck.getCost()){
                        bestLastNode = nodeCheck;
```

```
                        System.out.println("New bestLastNode!");
                    }
                }else{
                    if(nodeCheck.getCost() > costUpperBound){
                        nodeCheck.setBounded();
                    }else{
                        nodeCheck.setIsActive(true);
                        nodeCheck.expand();
                        for(Node c : nodeCheck.getChildren() ){
                            pq.add(c);
                            numOfNodesGenerated++;
                        }
                        nodeCheck.setIsActive(false);

                    }
                }
            }

            Node SolutionNode = bestLastNode;
            while(SolutionNode != this.root){
                solution.add(SolutionNode.getMove());
                System.out.println(SolutionNode.getMove() + " added
to solution");
                SolutionNode = SolutionNode.getParent();
            }
            Collections.reverse(solution);
        }
        end = System.currentTimeMillis();
    }

    public int getNumOfNodesGenerated(){
        return this.numOfNodesGenerated;
    }
    public Node getRoot(){
        return this.root;
    }

    public boolean getIsPossible(){
        return this.isPossible;
    }

    public List<Move> getSolution(){
        return this.solution;
    }

    public long getTimeElapsed(){
        return this.end - this.start;
    }
```

```java
public static void main(String[] args){
    int[][] intarray = new int[4][4];
    try{
        FileReader fr = new FileReader(args[0]);

        // Declaring loop variable
        int i;
        int tmp=0;
        int x = 0;
        int y = 0;
        // Holds true till there is nothing to read
        while ((i = fr.read()) != -1){

            // Print all the content of a file
            if(i >= '0' && i <= '9') {
                tmp = tmp*10 + (i - '0');
            }
            else if(tmp!=0){
                System.out.println(x + " " + y + " " + tmp);
                intarray[x][y] = tmp;
                tmp = 0;
                y++;
                if(y==4){
                    x++;
                    y=0;
                }
            }
        }
        if(tmp!=0){
            intarray[x][y] = tmp;
        }
        System.out.print((char)i);
        fr.close();

        Node root = new Node(intarray);
        Tree tree = new Tree(root);
        tree.start();

        System.out.print(tree.getSolution());
    }catch(FileNotFoundException e){
        System.out.println("An error occurred.");
        e.printStackTrace();
    }catch(IOException e){
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
    public void SaveFile(String filepath, String text){

        try{
            File myObj = new File(filepath);
            if (myObj.createNewFile()) {
              System.out.println("File created: " + myObj.getName());
            } else {
              System.out.println("File already exists.");
            }
            PrintWriter out = new PrintWriter(filepath);
            out.println(text);
            out.close();
        }catch(IOException e){
            System.out.println("Error has occured");
        }
    }
}

class NodeComparator implements Comparator<Node> {

    public int compare(Node n1, Node n2)
    {
        if(n1.getCost() < n2.getCost()){
            return -1;
        }else if (n1.getCost() > n2.getCost()){
            return 1;
        }
        return 0;
    }
}
```

4. *File class GUI.java*
   File class GUI.java berisi kode berikut.

```
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.BorderFactory.*;

import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.CardLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.File;
import java.awt.Image;
import java.awt.Font;
```

```java
import java.awt.Dimension;
import java.io.*;

import java.util.*;

public class GUI {

    public Tree puzzleTree;
    public PuzzlePanel pnl_puzzle;
    public JTextField textfield;
    public GUI(){
        JFrame frame = new JFrame();
        frame.setTitle("Boom Boom Bakudan 15 Puzzle Game");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.setSize(420,560);

        ImageIcon logo = new ImageIcon("src/assets/logo.png");
        frame.setIconImage(logo.getImage());
        frame.getContentPane().setBackground(new Color(0x9c6b65));

        BorderLayout layout = new BorderLayout();
        frame.setLayout(layout);
        layout.layoutContainer(frame.getContentPane());

        JLabel label1 = new JLabel();
        label1.setText("Puzzle Location: ");
        label1.setForeground(new Color(0x752c18));
        textfield = new JTextField(20);
        JButton btn = new JButton("Browse!");
        btn.setBackground(new Color(0x752c18));
        btn.setForeground(new Color(0xd5beaa));
        btn.setSize(5,5);
        btn.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e){
                JFileChooser fileChooser = new JFileChooser();
                int option = fileChooser.showOpenDialog(frame);
                if(option == JFileChooser.APPROVE_OPTION){
                    File file = fileChooser.getSelectedFile();
                    textfield.setText(file.getAbsolutePath());
                }else{
                    System.out.println("Open command canceled");
                }
            }
        });
        JPanel pnl_browseFolder = new JPanel();
        pnl_browseFolder.setOpaque(false);
```

```
        pnl_browseFolder.add(label1);
        pnl_browseFolder.add(textfield);
        pnl_browseFolder.add(btn);

        JPanel pnl_info = new JPanel();
        pnl_info.setLayout(new CardLayout());
        pnl_info.setOpaque(false);

        JButton btn_start = new JButton("Start!");
        btn_start.setBackground(new Color(0x752c18));
        btn_start.setForeground(new Color(0xd5beaa));

        JLabel lbl_process = new JLabel("Processing...");
        lbl_process.setVerticalAlignment(JLabel.CENTER);
        lbl_process.setHorizontalAlignment(JLabel.CENTER);

        pnl_info.add("start", btn_start);
        pnl_info.add("process", lbl_process);

        JPanel pnl_north = new JPanel();
        pnl_north.setLayout(new GridLayout(2,1));
        pnl_north.setOpaque(false);
        pnl_north.add(pnl_browseFolder);
        pnl_north.add(pnl_info);

        JPanel pnl_center = new JPanel();
        pnl_center.setOpaque(false);

        Dimension southDimension = new Dimension(400,100);
        JPanel pnl_south = new JPanel();
        pnl_south.setOpaque(false);
        pnl_south.setPreferredSize(southDimension);

        JTextArea txtarea_solution = new JTextArea();
        //txtarea_solution.setBounds(20, 20, 20, 20);
        //DefaultCaret caret =
(DefaultCaret)txtarea_solution.getCaret();
        //caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
        //txtarea_solution.setPreferredSize(southDimension);
        JScrollPane scrollpane = new JScrollPane(txtarea_solution);
        scrollpane.setPreferredSize(southDimension);
        pnl_south.add(scrollpane);

        pnl_puzzle = new PuzzlePanel();
        pnl_center.add(pnl_puzzle);
        btn_start.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent ev){
```

```
                SwingWorker<Void,String> worker = new
SwingWorker<Void,String>(){
                    @Override
                    protected Void doInBackground() throws Exception
{
                        File puzzleFile = new
File(textfield.getText());
                        if(puzzleFile.exists()){
                            txtarea_solution.setText("");
                            CardLayout cl = (CardLayout)
pnl_info.getLayout();
                            cl.next(pnl_info);
                            lbl_process.setText("Processing...");
                            pnl_puzzle.generatePuzzle(textfield.getTe
xt());
                            pnl_puzzle.animateSolution(lbl_process);
                            cl.first(pnl_info);
                            txtarea_solution.setText(pnl_puzzle.gener
ateSolutionText());
                        }else{
                            JOptionPane.showMessageDialog(frame,
"File not found!");
                        }
                        return null;
                    }
                };
                worker.execute();
            }

        });

        frame.add(pnl_north, BorderLayout.NORTH);
        frame.add(pnl_center, BorderLayout.CENTER);
        frame.add(pnl_south, BorderLayout.SOUTH);
        frame.setVisible(true);
    }
    public static void main(String[] args){
        new GUI();
    }
}

class PuzzleElement extends JLabel{
    public int[] darkfont = {6,15};

    PuzzleElement(int i){
        if(i == 16){
            this.setSize(50,50);
            this.setBackground(new Color(0xd5beaa));
```

```
            this.setOpaque(true);
        }else{
            boolean dark = false;
            for(int df : darkfont) if(i == df) dark = true;
            Color darkFont = new Color(0x6b5f55);
            Color lightFont = new Color(0xd5beaa);
            Border border = BorderFactory.createLineBorder(new
Color(0xd5beaa),1);

            this.setSize(50,50);
            this.setText("" + i);
            this.setIcon(new ImageIcon(new
ImageIcon("src/assets/klee-" + i +
".png").getImage().getScaledInstance(80, 80, Image.SCALE_DEFAULT)));
            this.setBackground(new Color(0x752c18));
            this.setForeground(dark ? darkFont : lightFont);
            this.setHorizontalTextPosition(JLabel.CENTER);
            this.setVerticalTextPosition(JLabel.CENTER);
            this.setVerticalAlignment(JLabel.CENTER);
            this.setHorizontalAlignment(JLabel.CENTER);
            this.setFont(new Font("Serif", Font.BOLD, 30));
            this.setBorder(border);
            this.setBounds(0,0,50,50);
            this.setOpaque(true);

        }
    }
}

class PuzzlePanel extends JPanel{
    public JPanel[] pnl_puzzleCard = new JPanel[16];
    public JLabel[] lbl_puzzlePiece = new JLabel[16];
    public Tree puzzleTree;
    public int[][] pieces;

    PuzzlePanel(){
        this.setOpaque(false);
        this.setLayout(new GridBagLayout());
        for(int i=1; i<=16; i++){
            GridBagConstraints c = new GridBagConstraints();
            c.gridx = (i-1)%4;
            c.gridy = (i-1)/4;
            c.weightx = 0;
            c.weighty = 1;

            pnl_puzzleCard[i-1] = new JPanel();
            pnl_puzzleCard[i-1].setLayout(new CardLayout());
```

```
            CardLayout cl = (CardLayout)(pnl_puzzleCard[i-
1].getLayout());
            for(int j=1; j<=16; j++){
                pnl_puzzleCard[i-1].add("" + j, new
PuzzleElement(j));
            }
            this.add(pnl_puzzleCard[i-1],c);
            cl.show(pnl_puzzleCard[i-1], "" + i);
        }
    }

    public void generateArray(String filepath){
        this.pieces = new int[4][4];
        try{
            FileReader fr = new FileReader(filepath);
            // Declaring loop variable
            int i;
            int tmp=0;
            int x = 0;
            int y = 0;
            // Holds true till there is nothing to read
            while ((i = fr.read()) != -1){

                // Print all the content of a file
                if(i >= '0' && i <= '9') {
                    tmp = tmp*10 + (i - '0');
                }
                else if(tmp!=0){
                    System.out.println(x + " " + y + " " + tmp);
                    this.pieces[x][y] = tmp;
                    tmp = 0;
                    y++;
                    if(y==4){
                        x++;
                        y=0;
                    }
                }
            }
            if(tmp!=0){
                System.out.println(x + " " + y + " " + tmp);
                this.pieces[x][y] = tmp;
            }
            System.out.print((char)i);
            fr.close();
        }catch(FileNotFoundException e){
            System.out.println("An error occurred.");
            e.printStackTrace();
        }catch(IOException e){
```

```
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }

    public void generatePuzzle(String filepath){
        generateArray(filepath);
        Node root = new Node(pieces);
        puzzleTree = new Tree(root);
        for(int i=0; i<4; i++){
            for(int j=0; j<4; j++){
                CardLayout cl =
(CardLayout)(pnl_puzzleCard[4*i+j].getLayout());
                cl.show(pnl_puzzleCard[4*i+j], "" + pieces[i][j] );
            }
        }
    }


    public void animateSolution(JLabel lbl_move){
        puzzleTree.start();
        List<Move> solutions = puzzleTree.getSolution();
        for(Move m : solutions){
            System.out.println(m);
            animateMove(m);
            lbl_move.setText(m.toString());
            try{
                Thread.sleep(1000);
            }catch(InterruptedException ex){
                Thread.currentThread().interrupt();
            }
        }
    }

    public int getVoidIdx(){
        int idx = 0;
        while(this.pieces[idx/4][idx%4] != 16){
            idx++;
        }
        return idx;
    }

    public void swapLabel(int x1, int y1, int x2, int y2){

        int temp = pieces[y1][x1];
        pieces[y1][x1] = pieces[y2][x2];
        pieces[y2][x2] = temp;
```

```
        CardLayout cl1 =
(CardLayout)(pnl_puzzleCard[y1*4+x1].getLayout());
        cl1.show(pnl_puzzleCard[y1*4+x1],"" + pieces[y1][x1]);
        CardLayout cl2 =
(CardLayout)(pnl_puzzleCard[y2*4+x2].getLayout());
        cl2.show(pnl_puzzleCard[y2*4+x2],"" + pieces[y2][x2]);
    }
    public void animateMove(Move m){
        int void_idx = getVoidIdx();
        int voidx = void_idx%4;
        int voidy = void_idx/4;
        if(m == Move.UP){
            swapLabel(voidx, voidy, voidx, voidy-1);
        }else if(m == Move.DOWN){
            swapLabel(voidx, voidy, voidx, voidy+1);
        }else if(m == Move.RIGHT){
            swapLabel(voidx, voidy, voidx+1, voidy);
        }else if(m == Move.LEFT){
            swapLabel(voidx, voidy, voidx-1, voidy);
        }
    }
    public String generateSolutionText(){
        Node root = puzzleTree.getRoot();
        String text = "1. Initial Puzzle Matrix: \n";
        for(int i=0; i<4; i++){
            for(int j=0; j<4; j++){
                text += root.getCell(i, j) + " ";
            }
            text += '\n';
        }
        System.out.println("1 success");
        text += "2. Kurang(i) values: \n";
        for(int i=1; i<=16; i++){
            text += "Kurang(" + i + ") = " + root.Kurang(i) + "\n";
        }
        System.out.println("2 success");
        text += "3. Sum of Kurang(i) + X = " + root.kurangPlusX() +
"\n";
        System.out.println("3 success");
        if(root.possibleChecker()){
            text += "Solution Steps: \n";
            Node x = root;
            for(Move m : puzzleTree.getSolution()){
                text += m.toString() + '\n';
                x.move(m);
                for(int i=0; i<4; i++){
                    for(int j=0; j<4; j++){
                        text += x.getCell(i, j) + " ";
```

```
                    }
                    text += '\n';
                }
            }
            System.out.println("4 success");
            text += "Steps : " + puzzleTree.getSolution().size();
            text += "\nTime elapsed to generate solution: " +
puzzleTree.getTimeElapsed() + " ms\n";
            text += "Number of generated nodes = " +
puzzleTree.getNumOfNodesGenerated() + "\n";
            System.out.println("all success");
        }else{
            text += "Puzzle cannot be solved";
        }
        return text;
    }
}

/*
class PuzzleWorker extends SwingWorker<> {

}*/
```

Bab 4 : Masukan dan Luaran Program

Masukan program ini berupa *file* berekstensi *.txt*. Perlu diperhatikan bahwa pada program ini, arah UP, DOWN, LEFT, dan RIGHT mengacu pada pergerakan kotak kosong (jika kotak kosong ke atas, langkahnya ialah UP dan seterusnya)

1. *File solvable_1.txt*
   File solvable_1.txt berisi teks berikut.

   ```
   1 2 3 4
   5 6 16 8
   9 10 7 11
   13 14 15 12
   ```

   Dengan masukan *file* tersebut, keluarannya ialah seperti berikut.

Gambar 3. Hasil program ketika menjalankan *solvable_1.txt*

Textbox pada bagian bawah GUI berisi teks berikut.

1. Initial Puzzle Matrix:
1 2 3 4
5 6 16 8
9 10 7 11
13 14 15 12
2. Kurang(i) values:
Kurang(1) = 0
Kurang(2) = 0
Kurang(3) = 0
Kurang(4) = 0
Kurang(5) = 0
Kurang(6) = 0
Kurang(7) = 0
Kurang(8) = 1
Kurang(9) = 1
Kurang(10) = 1
Kurang(11) = 0
Kurang(12) = 0
Kurang(13) = 1
Kurang(14) = 1
Kurang(15) = 1
Kurang(16) = 9
3. Sum of Kurang(i) + X = 16
Solution Steps:
DOWN
1 2 3 4
5 6 7 8
9 10 16 11
13 14 15 12
RIGHT
1 2 3 4

```
5 6 7 8
9 10 11 16
13 14 15 12
DOWN
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Steps : 3
Time elapsed to generate solution: 9 ms
Number of generated nodes = 10
```

2. *File solvable_2.txt*

   File solvable_2.txt berisi teks berikut.

```
16 1 2 3
6 7 8 4
5 9 10 11
13 14 15 12
```

Dengan masukan *file* tersebut, keluaran program ialah seperti berikut.

Gambar 4. Hasil program ketika menjalankan *solvable_2.txt*

Textbox pada bagian bawah GUI berisi teks berikut.

```
1. Initial Puzzle Matrix:
16 1 2 3
6 7 8 4
5 9 10 11
```

13 14 15 12
2. Kurang(i) values:
Kurang(1) = 0
Kurang(2) = 0
Kurang(3) = 0
Kurang(4) = 0
Kurang(5) = 0
Kurang(6) = 2
Kurang(7) = 2
Kurang(8) = 2
Kurang(9) = 0
Kurang(10) = 0
Kurang(11) = 0
Kurang(12) = 0
Kurang(13) = 1
Kurang(14) = 1
Kurang(15) = 1
Kurang(16) = 15
3. Sum of Kurang(i) + X = 24
Solution Steps:
RIGHT
1 16 2 3
6 7 8 4
5 9 10 11
13 14 15 12
RIGHT
1 2 16 3
6 7 8 4
5 9 10 11
13 14 15 12
RIGHT
1 2 3 16
6 7 8 4
5 9 10 11
13 14 15 12
DOWN
1 2 3 4
6 7 8 16
5 9 10 11
13 14 15 12
LEFT
1 2 3 4
6 7 16 8
5 9 10 11
13 14 15 12
LEFT
1 2 3 4
6 16 7 8
5 9 10 11
13 14 15 12
LEFT

```
1 2 3 4
16 6 7 8
5 9 10 11
13 14 15 12
DOWN
1 2 3 4
5 6 7 8
16 9 10 11
13 14 15 12
RIGHT
1 2 3 4
5 6 7 8
9 16 10 11
13 14 15 12
RIGHT
1 2 3 4
5 6 7 8
9 10 16 11
13 14 15 12
RIGHT
1 2 3 4
5 6 7 8
9 10 11 16
13 14 15 12
DOWN
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Steps : 12
Time elapsed to generate solution: 91 ms
Number of generated nodes = 28
```

3. *File solvable_3.txt*
   File solvable_3.txt berisi teks berikut.

```
1 6 2 4
5 16 3 8
9 7 15 11
13 14 10 12
```

Dengan masukan *file* tersebut, keluaran program ialah seperti berikut.

Gambar 5. Hasil program ketika menjalankan *solvable_3.txt*

Textbox pada bagian bawah GUI berisi teks berikut.

```
1. Initial Puzzle Matrix:
1 6 2 4
5 16 3 8
9 7 15 11
13 14 10 12
2. Kurang(i) values:
Kurang(1) = 0
Kurang(2) = 0
Kurang(3) = 0
Kurang(4) = 1
Kurang(5) = 1
Kurang(6) = 4
Kurang(7) = 0
Kurang(8) = 1
Kurang(9) = 1
Kurang(10) = 0
Kurang(11) = 1
Kurang(12) = 0
Kurang(13) = 2
Kurang(14) = 2
Kurang(15) = 5
Kurang(16) = 10
3. Sum of Kurang(i) + X = 28
Solution Steps:
LEFT
1 6 2 4
16 5 3 8
```

9 7 15 11
13 14 10 12
DOWN
1 6 2 4
9 5 3 8
16 7 15 11
13 14 10 12
DOWN
1 6 2 4
9 5 3 8
13 7 15 11
16 14 10 12
RIGHT
1 6 2 4
9 5 3 8
13 7 15 11
14 16 10 12
RIGHT
1 6 2 4
9 5 3 8
13 7 15 11
14 10 16 12
UP
1 6 2 4
9 5 3 8
13 7 16 11
14 10 15 12
LEFT
1 6 2 4
9 5 3 8
13 16 7 11
14 10 15 12
DOWN
1 6 2 4
9 5 3 8
13 10 7 11
14 16 15 12
LEFT
1 6 2 4
9 5 3 8
13 10 7 11
16 14 15 12
UP
1 6 2 4
9 5 3 8
16 10 7 11
13 14 15 12
UP
1 6 2 4
16 5 3 8
9 10 7 11

```
13 14 15 12
RIGHT
1 6 2 4
5 16 3 8
9 10 7 11
13 14 15 12
UP
1 16 2 4
5 6 3 8
9 10 7 11
13 14 15 12
RIGHT
1 2 16 4
5 6 3 8
9 10 7 11
13 14 15 12
DOWN
1 2 3 4
5 6 16 8
9 10 7 11
13 14 15 12
DOWN
1 2 3 4
5 6 7 8
9 10 16 11
13 14 15 12
RIGHT
1 2 3 4
5 6 7 8
9 10 11 16
13 14 15 12
DOWN
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Steps : 18
Time elapsed to generate solution: 11520 ms
Number of generated nodes = 9178
```

4. *File unsolvable_1.txt*
   File unsovable_1.txt berisi teks berikut.

```
5 1 3 4
9 2 7 8
10 6 15 11
13 16 14 12
```

Dengan masukan *file* tersebut, keluaran program ialah seperti berikut.

Gambar 6. Hasil program ketika menjalankan *unsolvable_1.txt*

Textbox pada bagian bawah GUI berisi teks berikut.

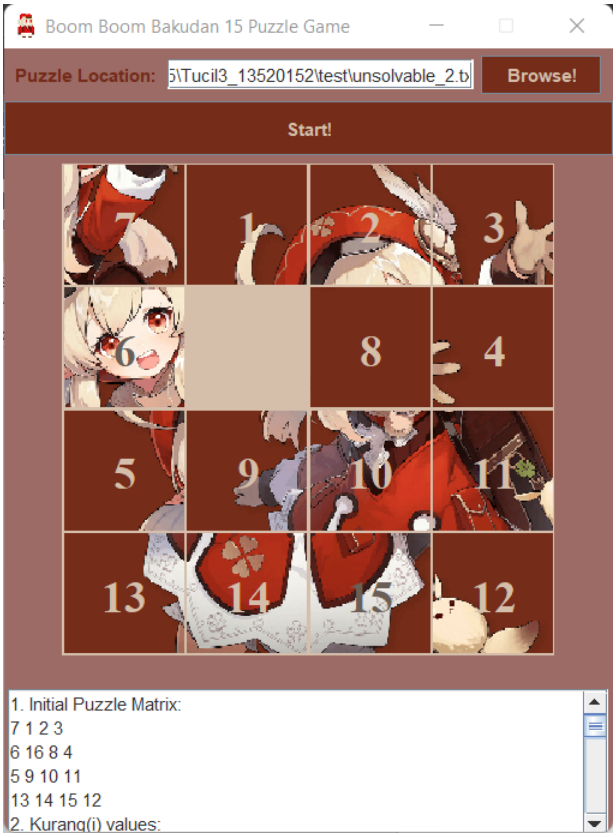
```
1. Initial Puzzle Matrix:
5 1 3 4
9 2 7 8
10 6 15 11
13 16 14 12
2. Kurang(i) values:
Kurang(1) = 0
Kurang(2) = 0
Kurang(3) = 1
Kurang(4) = 1
Kurang(5) = 4
Kurang(6) = 0
Kurang(7) = 1
Kurang(8) = 1
Kurang(9) = 4
Kurang(10) = 1
Kurang(11) = 0
Kurang(12) = 0
Kurang(13) = 1
Kurang(14) = 1
Kurang(15) = 4
Kurang(16) = 2
3. Sum of Kurang(i) + X = 21
Puzzle cannot be solved
```

5. *File unsolvable_2.txt*

   File unsolvable_2.txt berisi teks berikut.

```
7 1 2 3
6 16 8 4
5 9 10 11
13 14 15 12
```

Dengan masukan *file* tersebut, keluaran program ialah seperti berikut.



Gambar 7. Hasil program ketika menjalankan *unsolvable_2.txt*

Textbox pada bagian bawah GUI berisi teks berikut.

```
1. Initial Puzzle Matrix:
7 1 2 3
6 16 8 4
5 9 10 11
13 14 15 12
2. Kurang(i) values:
Kurang(1) = 0
Kurang(2) = 0
Kurang(3) = 0
Kurang(4) = 0
Kurang(5) = 0
Kurang(6) = 2
Kurang(7) = 6
Kurang(8) = 2
Kurang(9) = 0
Kurang(10) = 0
Kurang(11) = 0
Kurang(12) = 0
Kurang(13) = 1
Kurang(14) = 1
```

Kurang(15) = 1 Kurang(16) = 10 3. Sum of Kurang(i) + X = 23 Puzzle cannot be solved

Bab 5 : Kesimpulan

Permainan 15-Puzzle dapat diselesaikan dengan algoritma Branch and Bound. Akan tetapi, algoritma ini tidak cukup efektif untuk 15-Puzzle yang membutuhkan lebih dari 30 langkah, karena tiap langkahnya dapat membangkitkan setidaknya dua *node*, sehingga terdapat lebih dari 2^{30} *node* yang dapat dibangkitkan.

Bab 6 : Alamat Github

Program ini dapat diakses pada tautan github berikut.

DeeGeeDow/Tucil3_13520152: Boom Boom Bakudan 15 Puzzle Solver is a program that solve 15 Puzzle with Branch and Bound Algorithm (github.com)

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat	✓	