LAPORAN TUGAS KECIL III

"Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*"

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan

Mata Kuliah Strategi Algoritma (IF2211)

KELAS 02

Dosen : Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

DISUSUN OLEH:

Yakobus Iryanto Prasethio (13520104)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2021-2022

Daftar Isi

BAB I
Langkah Algoritma *Branch and Bound*

Deskripsi Langkah Algoritma

Algoritma *Branch and Bound* adalah algoritma yang digunakan untuk persoalan optimisasi. Setiap simpul diberi sebuah nilai cost yang merupakan nilai taksiran lintasan termurah ke simpul tujuan yang melalui simpul tersebut. Simpul yang berikutnya di-expand tidak lagi berdasarkan urutan pembuatannya, tetapi simpul yang memiliki cost paling kecil (menggunakan sebuah priority queue).

Langkah – langkah algoritma untuk program 15-Puzzle Program:

1. Program akan menerima sebuah matriks akar (root matrix) dari input user, sebuah file, atau dari sebuah random matrix yang dibuat oleh program. Random matrix yang dibuat pasti dapat dicari solusinya (nilai fungsi KURANG(i) + X selalu genap).

2. Program akan menghitung nilai fungsi KURANG(i) + X yang bertujuan untuk mengecek apakah matriks tersebut dapat diselesaikan atau tidak. Apabila nilai fungsi KURANG(i) + X bernilai genap, maka matriks tersebut dapat memiliki solusi.

3. Program kemudian akan menggunakan matriks akar tersebut sebagai dasar dari algoritma *branch and bound*. Matriks akar tersebut akan dijadikan *e-node*, kemudian akan dibangun matriks anak – anaknya sesuai dengan arah yang memungkinkan.

4. Setiap matriks anak yang dibentuk akan dihitung *cost*-nya, menggunakan fungsi $c(i) = f(i) + g(i)$, dimana $f(i)$ adalah ongkos untuk mencapai simpul i dari akar (kedalaman simpul di dalam pohon) dan $g(i)$ adalah ongkos untuk mencapai simpul tujuan dari simpul i (banyaknya elemen matriks yang berbeda dengan elemen matriks tujuan).

5. Program akan menambahkan anak ke dalam sebuah *priority queue*, dimana *priority*-nya adalah nilai *cost* terkecil akan memiliki *priority* terbesar. Matriks anak yang memiliki *cost* terkecil akan dijadikan *e-node* untuk pembentukan anak selanjutnya.

6. Algoritma akan berhenti ketika matriks *e-node* sama dengan matriks tujuan, sehingga simpul solusi telah ditemukan. Program menghapus semua elemen *priority queue* yang memiliki nilai *cost* lebih besar daripada *cost e-node*.

7. Fungsi algoritma akan mengembalikan sebuah node yang berisi *list* arah yang diambil untuk mencapai simpul tujuan tersebut.

BAB II
Source Program dalam Bahasa Java

Main.java

```java
public class Main {
    public static void main(String[] args) {
        new GUI();
    }
}
```

Input.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class Input {
    // Function to create a random matrix
    public static ArrayList<ArrayList<Integer>> createRandomMatrix() {
        ArrayList<Integer> randomNumber = new ArrayList<~>();
        for (int i = 0; i < 16; i++) {
            randomNumber.add(i);
        }

        Collections.shuffle(randomNumber);

        ArrayList<ArrayList<Integer>> matrix = new ArrayList<~>();
        int index = 0;
        for (int i = 0; i < 4; i++) {
            ArrayList<Integer> matrixRow = new ArrayList<~>();
            for (int j = 0; j < 4; j++) {
                matrixRow.add(randomNumber.get(index));
                index++;
            }
            matrix.add(matrixRow);
        }

        return matrix;
    }
```

```java
// Function to read matrix from a file
public static ArrayList<ArrayList<Integer>> readFromFile(String path) throws FileNotFoundException {
    Scanner input = new Scanner(new File(path));
    ArrayList<ArrayList<Integer>> matrix = new ArrayList<~>();

    for(int i = 0; i < 4; i++) {
        ArrayList<Integer> matrixRow = new ArrayList<~>();
        for (int j = 0; j < 4; j++) {
            if (input.hasNextInt()) {
                matrixRow.add(input.nextInt());
            }
        }
        matrix.add(matrixRow);
    }
    return matrix;
}
```

Node.java

```java
import java.util.ArrayList;

public class Node {
    private static int counter = 0;
    private int id;
    private ArrayList<ArrayList<Integer>> matrix;
    private int cost;
    private ArrayList<String> directions;
    private int depth;

    // Default constructor
    public Node() {
        this.setId(++counter);
        this.matrix = new ArrayList<ArrayList<Integer>>();
        this.cost = 0;
        this.directions = new ArrayList<String>();
        this.depth = 0;
    }

    // User defined constructor
    public Node(ArrayList<ArrayList<Integer>> matrix, int cost, ArrayList<String> directions) {
        this.setId(++counter);
        this.matrix = matrix;
        this.cost = cost;
        this.directions = directions;
        this.depth = this.directions.size();
    }

    // Getter and setter
    public ArrayList<ArrayList<Integer>> getMatrix() { return this.matrix; }

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

    public ArrayList<String> getDirections() { return this.directions; }

    public int getDepth() { return this.depth; }

    public void setId(int id) { this.id = id; }
```

```java
// Function to display the matrix within a node
public void displayMatrix() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            System.out.print(this.matrix.get(i).get(j) + " ");
        }
        System.out.println();
    }
}

// Function to display in-depth information about a node
public void displayIndividualNodes() {
    System.out.println("Id: " + this.id);
    System.out.println("Matrix: ");
    this.displayMatrix();
    System.out.println("Cost: " + this.cost);
    System.out.println("Directions taken: " + this.directions.toString());
    System.out.println("Depth: " + this.depth);
    System.out.println();
}
```

PrioQueue.java

```java
import java.util.ArrayList;

public class PrioQueue {
    private ArrayList<Node> prioQueue;

    // Default Constructor
    public PrioQueue() { this.prioQueue = new ArrayList<Node>(); }

    // Getter
    public ArrayList<Node> getPrioQueue() { return this.prioQueue; }

    // Push elements to the priority queue based on its cost
    // The lower the cost, the earlier its placement
    public void push(ArrayList<Node> nodeList, Node elmt) {
        // If the element already exists in the node list, don't add it
        if (Utilities.checkRepeatingNodes(nodeList, elmt)) {
            // If priority queue is still empty, just add the element
            if (this.prioQueue.isEmpty()) {
                this.prioQueue.add(elmt);
            } else {
                int index = -999;
                boolean foundIndex = false;
                // If priority queue has an element, find the correct position for it
                for (int i = 0; i < this.prioQueue.size() && !foundIndex; i++) {
                    if (elmt.getCost() < this.prioQueue.get(i).getCost()) {
                        index = i;
                        foundIndex = true;
                    }
                }
                if (index != -999) {
                    this.prioQueue.add(index, elmt);
                } else {
                    this.prioQueue.add(elmt);
                }
            }
        }
    }
```

```java
// Pop the front element of the queue
public void pop() { this.prioQueue.remove( index: 0); }

// Peek the front element of the queue without popping it
public Node peek() { return this.prioQueue.get(0); }

// Function to get the priority queue length
public int length() { return this.prioQueue.size(); }

// Function to remove elements that have a higher cost than the E-node
public void removeLowerPriority(Node e_node) {
    for (int i = this.length() - 1; i >= 0; i--) {
        if (this.prioQueue.get(i).getCost() > e_node.getCost()) {
            this.pop();
        }
    }
}
```

Utilities.java

```java
import java.util.ArrayList;
import java.util.HashSet;

public class Utilities {
    // Function to add all the elements within an array
    public static int sumArray(ArrayList<Integer> array) {
        int sum = 0;
        for (int i = 0; i < array.size(); i++) {
            sum += array.get(i);
        }
        return sum;
    }

    // Function to find the opposite of a direction (e.g. the opposite of "UP" is "DOWN")
    public static String getReverseDirection(String direction) {
        if (direction.equals("UP")) {
            return "DOWN";
        }
        if (direction.equals("DOWN")) {
            return "UP";
        }
        if (direction.equals("LEFT")) {
            return "RIGHT";
        }
        if (direction.equals("RIGHT")) {
            return "LEFT";
        }
        return "None";
    }
```

```java
// Function to copy a matrix to another matrix
public static ArrayList<ArrayList<Integer>> copyMatrix(ArrayList<ArrayList<Integer>> input) {
    ArrayList<ArrayList<Integer>> output = new ArrayList<~>();
    for (int i = 0; i < 4; i++) {
        ArrayList<Integer> outputRow = new ArrayList<~>();
        for (int j = 0; j < 4; j++) {
            outputRow.add(input.get(i).get(j));
        }
        output.add(outputRow);
    }
    return output;
}

// Function to copy a list to another list
public static ArrayList<String> copyList(ArrayList<String> originalList) {
    return new ArrayList<String>(originalList);
}

// Function to convert a list to a matrix
public static ArrayList<ArrayList<Integer>> convertListToMatrix(ArrayList<Integer> list) {
    int counter = 0;
    ArrayList<ArrayList<Integer>> matrix = new ArrayList<~>();
    for (int i = 0; i < 4; i++) {
        ArrayList<Integer> matrixRow = new ArrayList<~>();
        for (int j = 0; j < 4; j++) {
            matrixRow.add(list.get(counter));
            counter++;
        }
        matrix.add(matrixRow);
    }

    return matrix;
}
```

```java
// Function to display goal states within the GUI
public static String displayGoalStates(ArrayList<ArrayList<Integer>> root, ArrayList<Node> goalState) {
    ArrayList<ArrayList<Integer>> currentMatrix = copyMatrix(root);
    StringBuilder output = new StringBuilder();
    ArrayList<String> direction = goalState.get(0).getDirections();
    for (int i = 0; i < direction.size(); i++) {
        ArrayList<ArrayList<Integer>> printChild = Algorithm.createNewChild(currentMatrix, direction.get(i));
        if (i == direction.size() - 1) {
            output.append("Goal State: \n");
            output.append("Direction taken: " + direction.get(i) + "\n");
            output.append(formatMatrixOutput(printChild));
        } else {
            output.append("Direction taken: " + direction.get(i) + "\n");
            output.append(formatMatrixOutput(printChild));
        }
        currentMatrix = copyMatrix(printChild);
    }
    output.append("Directions Taken: \n");
    output.append(goalState.get(0).getDirections());
    return output.toString();
}
```

```java
// Function to convert a matrix to a list
public static ArrayList<Integer> convertMatrixToList(ArrayList<ArrayList<Integer>> matrix) {
    ArrayList<Integer> list = new ArrayList<~>();
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            list.add(matrix.get(i).get(j));
        }
    }
    return list;
}

// Function to find a repeating node within an arraylist
// If there is a repeating nodes, function will return true
public static boolean checkRepeatingNodes(ArrayList<Node> nodeList, Node elmt) {
    for (Node node : nodeList) {
        if (node.getMatrix().equals(elmt.getMatrix())) {
            return false;
        }
    }
    return true;
}

// Function to find a repeating element within a matrix
// If there is a repeating element, function will return false
public static boolean checkRepeatingElement(ArrayList<ArrayList<Integer>> matrix) {
    HashSet<Integer> setChecker = new HashSet<~>(convertMatrixToList(matrix));
    return (setChecker.size() == matrix.size());
}

// Function to format a matrix output
public static String formatMatrixOutput(ArrayList<ArrayList<Integer>> matrix) {
    StringBuilder output = new StringBuilder();
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            output.append(matrix.get(i).get(j) + " ");
        }
        output.append("\n");
    }
    output.append("\n");
    return output.toString();
}
```

Algorithm.java

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

public class Algorithm {
    // Function to find X in KURANG(i) + X
    public static int findX(int i, int j) {
        int x;
        if (i % 2 == 0) {
            if (j % 2 != 0) {
                x = 1;
            } else {
                x = 0;
            }
        } else {
            if (j % 2 == 0) {
                x = 1;
            } else {
                x = 0;
            }
        }
        return x;
    }

    // Function to get the goal matrix
    public static ArrayList<ArrayList<Integer>> getGoalMatrix() {
        ArrayList<ArrayList<Integer>> matrix = new ArrayList<~>();
        int element = 1;
        for(int i = 0; i < 4; i++) {
            ArrayList<Integer> matrixRow = new ArrayList<~>();
            for(int j = 0; j < 4; j++) {
                if (i == 3 && j == 3) {
                    matrixRow.add(0);
                } else {
                    matrixRow.add(element);
                    element++;
                }
            }
            matrix.add(matrixRow);
        }
        return matrix;
```

```java
// Function to count the amount of difference between matrices
public static int getDifference(ArrayList<ArrayList<Integer>> matrix) {
    ArrayList<ArrayList<Integer>> goal = getGoalMatrix();
    int difference = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (!matrix.get(i).get(j).equals(goal.get(i).get(j))) {
                difference++;
            }
        }
    }

    if (difference > 0) {
        return difference - 1;
    } else {
        return 0;
    }
}
```

```java
// Check if goal is reachable using KURANG(i) + X
public static List<Object> isGoalReachable(ArrayList<ArrayList<Integer>> matrix) {
    int addition, counter, result;
    boolean goal;
    ArrayList<Integer> incorrectPos = new ArrayList<~>();
    for (int i = 0; i < 16; i++) {
        incorrectPos.add(0);
    }

    addition = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            int testElmt = matrix.get(i).get(j);
            counter = 0;
            if (testElmt != 0) {
                for (int x = 0; x < 4; x++) {
                    for (int y = 0; y < 4; y++) {
                        if (matrix.get(x).get(y) < testElmt && matrix.get(x).get(y) != 0) {
                            if (x > i) {
                                counter++;
                            } else if (x == i && y > j) {
                                counter++;
                            }
                        }
                    }
                }
                incorrectPos.add( index: testElmt - 1, counter);
            } else {
                addition = findX(i, j);
                for (int x = 0; x < 4; x++) {
                    for (int y = 0; y < 4; y++) {
                        if (x > i) {
                            counter++;
                        } else if (x == i && y > j) {
                            counter++;
                        }
                    }
                }
                incorrectPos.add( index: 15, counter);
        }
    }
}

result = Utilities.sumArray(incorrectPos) + addition;
if (result % 2 == 0) {
    goal = true;
} else {
    goal = false;
}
return Arrays.asList(goal, result);
```

```java
// Function to check which directions are possible to move the empty slot
public static ArrayList<String> checkDirectionPossibilities(ArrayList<ArrayList<Integer>> matrix) {
    ArrayList<String> possibilities = new ArrayList<~>();
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (matrix.get(i).get(j).equals(0)) {
                if (i - 1 >= 0) {
                    possibilities.add("UP");
                }
                if (i + 1 <= 3) {
                    possibilities.add("DOWN");
                }
                if (j - 1 >= 0) {
                    possibilities.add("LEFT");
                }
                if (j + 1 <= 3) {
                    possibilities.add("RIGHT");
                }
            }
        }
    }
    return possibilities;
}
```

```java
// Function to create new child matrix based on its direction
public static ArrayList<ArrayList<Integer>> createNewChild(ArrayList<ArrayList<Integer>> root, String direction) {
    ArrayList<ArrayList<Integer>> newChild = Utilities.copyMatrix(root);
    boolean newMatrixCreated = false;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (root.get(i).get(j).equals(0) && !newMatrixCreated) {
                switch (direction) {
                    case "UP" -> {
                        newChild.get(i).set(j, root.get(i - 1).get(j));
                        newChild.get(i - 1).set(j, 0);
                        newMatrixCreated = true;
                    }
                    case "DOWN" -> {
                        newChild.get(i).set(j, root.get(i + 1).get(j));
                        newChild.get(i + 1).set(j, 0);
                        newMatrixCreated = true;
                    }
                    case "LEFT" -> {
                        newChild.get(i).set(j, root.get(i).get(j - 1));
                        newChild.get(i).set(j - 1, 0);
                        newMatrixCreated = true;
                    }
                    case "RIGHT" -> {
                        newChild.get(i).set(j, root.get(i).get(j + 1));
                        newChild.get(i).set(j + 1, 0);
                        newMatrixCreated = true;
                    }
                }
            }
        }
    }
    return newChild;
}
```

```java
// Branch And Bound Algorithm will return the goalStates and the amount of nodes created
public static List<Object> branchAndBoundAlgorithm(Node root) {
    // Retrieving result from reachableResult function
    List<Object> reachableResult = isGoalReachable(root.getMatrix());
    boolean goal = (boolean)reachableResult.get(0);
    int result = (int)reachableResult.get(1);

    // Declaring list of nodes
    ArrayList<Node> goalStates = new ArrayList<~>();
    ArrayList<Node> allStates = new ArrayList<~>();
    int nodeCount = 0;
    // If goal is reachable, start branch and bound algorithm
    if (goal) {
        System.out.println("Function KURANG(i) + X result is " + result + " which is even");

        // Declaring priority queue and E-node
        PrioQueue liveNodes = new PrioQueue();
        Node e_node;
        boolean finish = false;
        while (!finish) {
            if (liveNodes.getPrioQueue().isEmpty()) {
                e_node = root;
            } else {
                e_node = liveNodes.peek();
                liveNodes.pop();
            }

            if (Utilities.checkRepeatingNodes(allStates, e_node)) {
                allStates.add(e_node);
            }

            // Debugging purposes
            System.out.println("Current e-node: ");
            e_node.displayIndividualNodes();
```

```
            // If the current e_node is the goal state
            if (getDifference(e_node.getMatrix()) == 0) {
                System.out.println("Found solution!");
                // Add the node to the goal state list
                goalStates.add(e_node);
                // Remove nodes with higher cost (lower priority) than the e_node
                liveNodes.removeLowerPriority(e_node);

                // If no more live nodes exists, break the loop
                if (liveNodes.getPrioQueue().isEmpty()) {
                    nodeCount = allStates.size();
                    finish = true;
                }
            } else {
                ArrayList<String> availableDirection = checkDirectionPossibilities(e_node.getMatrix());
                ArrayList<String> directionList = e_node.getDirections();
                for (String direction : availableDirection) {
                    // If e_node directionList is empty, add all directions
                    if (directionList.isEmpty()) {
                        // Create new child matrix based on a direction
                        ArrayList<ArrayList<Integer>> newChildMatrix = createNewChild(e_node.getMatrix(), direction);

                        // Create new direction list for new child
                        ArrayList<String> childDirectionList = Utilities.copyList(directionList);

                        // Calculate differences and cost
                        int difference = getDifference(newChildMatrix);
                        int cost = difference + e_node.getDepth() + 1;

                        // Add direction to the directionList
                        childDirectionList.add(direction);
                        Node newChild = new Node(newChildMatrix, cost, childDirectionList);

                        // Add newChild to liveNodes and allStates list
                        liveNodes.push(allStates, newChild);
                        allStates.add(newChild);
```

```
                    } else {
                        // Skipping the creation of recurring matrices (e.g. after moving "DOWN", skip adding moving "UP")
                        if (!direction.equals(Utilities.getReverseDirection(directionList.get(e_node.getDepth() - 1)))) {
                            // Create new child matrix based on a direction
                            ArrayList<ArrayList<Integer>> newChildMatrix = createNewChild(e_node.getMatrix(), direction);

                            // Create new direction list for new child
                            ArrayList<String> childDirectionList = Utilities.copyList(directionList);

                            // Calculate differences and cost
                            int difference = getDifference(newChildMatrix);
                            int cost = difference + e_node.getDepth() + 1;

                            // Add direction to the directionList
                            childDirectionList.add(direction);
                            Node newChild = new Node(newChildMatrix, cost, childDirectionList);

                            // Add newChild to liveNodes and allStates list
                            liveNodes.push(allStates, newChild);
                            allStates.add(newChild);
                        }
                    }
                }
            }
        }
    // If goal is not reachable, output information to the terminal
    } else {
        System.out.println("Goal is not reachable!");
        System.out.println("Function KURANG(i) + X result is " + result + " which is odd");
    }
    return Arrays.asList(goalStates, nodeCount);
}
```

GUI.java

```java
import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class GUI {
    // Creating frame objects
    private JFrame frame;

    // Creating label objects
    private JLabel intro1;
    private JLabel intro2;
    private JLabel intro3;
    private JLabel intro4;
    private JLabel errorLabel;
    private JLabel resetLabel;
    private JLabel timeTaken;

    // Creating button objects
    private JButton button_file_input;
    private JButton button_random_input;
    private JButton button_start;
    private JButton button_reset;

    // Creating text objects
    private JTextField matrixTextField;

    // Creating text area
    private JTextArea outputArea;
    private JTextArea exampleMatrix;

    // Creating scroll bar
    private JScrollPane scrollVertical;

    // Creating panel objects
    private JPanel mainPanel;
    private JPanel leftPanel;
    private JPanel matrixInputPanel;
```

```java
// Create array of text fields
private ArrayList<JTextField> listOfTextFields;

// Required ArrayLists for the algorithm
private ArrayList<ArrayList<Integer>> matrix;
private ArrayList<Node> goalStates;

// Global integer
private long startTime;
private long endTime;

// Global boolean
private boolean inputExists;
private boolean usingFile;
private boolean usingRandom;

public GUI() {
    this.listOfTextFields = new ArrayList<JTextField>();
    this.matrix = new ArrayList<ArrayList<Integer>>();

    this.inputExists = false;
    this.usingFile = false;
    this.usingRandom = false;

    frame = new JFrame();

    mainPanel = new JPanel();
    mainPanel.setLayout(new GridBagLayout());

    // Create a panel on the left side
    leftPanel = new JPanel();
    leftPanel.setLayout(new GridBagLayout());
```

```java
// Creating large text area on the left
outputArea = new JTextArea();
outputArea.setBorder(BorderFactory.createLineBorder(Color.BLACK));
outputArea.setEditable(false);
GridBagConstraints cTextArea = new GridBagConstraints();
cTextArea.fill = GridBagConstraints.BOTH;
cTextArea.gridx = 0;
cTextArea.gridy = 0;
cTextArea.gridheight = 12;
cTextArea.ipadx = 400;
leftPanel.add(outputArea, cTextArea);

scrollVertical = new JScrollPane(outputArea, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED)
GridBagConstraints cScrollPanel = new GridBagConstraints();
cScrollPanel.fill = GridBagConstraints.BOTH;
cScrollPanel.gridx = 0;
cScrollPanel.gridy = 1;
cScrollPanel.gridheight = 12;
cScrollPanel.ipadx = 400;
cScrollPanel.ipady = 400;
leftPanel.add(scrollVertical, cScrollPanel);

GridBagConstraints cLeftPane = new GridBagConstraints();
cLeftPane.fill = GridBagConstraints.BOTH;
cLeftPane.gridx = 0;
cLeftPane.gridy = 0;
cLeftPane.gridheight = 12;
cLeftPane.insets = new Insets( top: 20,  left: 20,  bottom: 20,  right: 10);
mainPanel.add(leftPanel, cLeftPane);
```

```
// Objects on the right
// Label Objects
intro1 = new JLabel( text: "Welcome to 15-Puzzle Problem Solver using Branch and Bound Algorithm");
GridBagConstraints cLabel1 = new GridBagConstraints();
cLabel1.gridx = 1;
cLabel1.gridy = 0;
cLabel1.insets = new Insets( top: 20,  left: 10,  bottom: 0,  right: 20);
mainPanel.add(intro1, cLabel1);

intro2 = new JLabel( text: "Please insert your matrix number (Empty slots are indicated by 0)");
GridBagConstraints cLabel2 = new GridBagConstraints();
cLabel2.gridx = 1;
cLabel2.gridy = 1;
cLabel2.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(intro2, cLabel2);

intro3 = new JLabel( text: "Example:");
GridBagConstraints cLabel3 = new GridBagConstraints();
cLabel3.gridx = 1;
cLabel3.gridy = 2;
cLabel3.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(intro3, cLabel3);

intro4 = new JLabel( text: "Or you can get the matrix using these");
GridBagConstraints cLabel4 = new GridBagConstraints();
cLabel4.gridx = 1;
cLabel4.gridy = 5;
cLabel4.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(intro4, cLabel4);
```

```
errorLabel = new JLabel( text: "Error: No Error Found!");
GridBagConstraints cErrorLabel = new GridBagConstraints();
cErrorLabel.gridx = 1;
cErrorLabel.gridy = 8;
cErrorLabel.insets = new Insets( top: 5,  left: 10,  bottom: 5,  right: 20);
mainPanel.add(errorLabel, cErrorLabel);

resetLabel = new JLabel( text: "Click the reset button to use another method");
GridBagConstraints cResetLabel = new GridBagConstraints();
cResetLabel.gridx = 1;
cResetLabel.gridy = 10;
cResetLabel.insets = new Insets( top: 5,  left: 10,  bottom: 5,  right: 20);
mainPanel.add(resetLabel, cResetLabel);

timeTaken = new JLabel( text: "Time elapsed: 0 ms");
GridBagConstraints cTimeTaken = new GridBagConstraints();
cTimeTaken.gridx = 0;
cTimeTaken.gridy = 12;
cTimeTaken.gridwidth = 2;
cTimeTaken.insets = new Insets( top: 5,  left: 20, bottom: 20,  right: 20);
mainPanel.add(timeTaken, cTimeTaken);
```

```
// Text area for example matrix
exampleMatrix = new JTextArea();
exampleMatrix.setBorder(BorderFactory.createLineBorder(Color.BLACK));
exampleMatrix.setEditable(false);
exampleMatrix.setText(
        "1 2 3 4\n" + "5 6 0 8\n" + "9 10 7 11\n" + "13 14 15 12"
);
GridBagConstraints cExampleMatrix = new GridBagConstraints();
cExampleMatrix.gridx = 1;
cExampleMatrix.gridy = 3;
cExampleMatrix.insets = new Insets( top: 10,  left: 10,  bottom: 10,  right: 20);
mainPanel.add(exampleMatrix, cExampleMatrix);

// Matrix input area
matrixInputPanel = new JPanel();
matrixInputPanel.setLayout(new GridLayout( rows: 4, cols: 4,  hgap: 2,  vgap: 2));

// Create text objects
for (int i = 0; i < 16; i++) {
    matrixTextField = new JTextField();
    matrixInputPanel.add(matrixTextField);
    this.listOfTextFields.add(matrixTextField);
}
GridBagConstraints cMatrixArea = new GridBagConstraints();
cMatrixArea.fill = GridBagConstraints.BOTH;
cMatrixArea.gridx = 1;
cMatrixArea.gridy = 4;
cMatrixArea.insets = new Insets( top: 10,  left: 10,  bottom: 10,  right: 20);
mainPanel.add(matrixInputPanel, cMatrixArea);
```

```
// Buttons
// File input button
button_file_input = new JButton( text: "Input matrix from file");
GridBagConstraints cButtonFile = new GridBagConstraints();
cButtonFile.gridx = 1;
cButtonFile.gridy = 6;
cButtonFile.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(button_file_input, cButtonFile);

// Random input button
button_random_input = new JButton( text: "Use random matrix");
GridBagConstraints cButtonRandom = new GridBagConstraints();
cButtonRandom.gridx = 1;
cButtonRandom.gridy = 7;
cButtonRandom.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(button_random_input, cButtonRandom);

// Start button
button_start = new JButton( text: "Start");
GridBagConstraints cButtonStart = new GridBagConstraints();
cButtonStart.gridx = 1;
cButtonStart.gridy = 9;
cButtonStart.insets = new Insets( top: 2,  left: 10,  bottom: 2,  right: 20);
mainPanel.add(button_start, cButtonStart);

// Reset button
button_reset = new JButton( text: "Reset");
GridBagConstraints cButtonReset = new GridBagConstraints();
cButtonReset.gridx = 1;
cButtonReset.gridy = 11;
cButtonReset.insets = new Insets( top: 2,  left: 10,  bottom: 20,  right: 20);
mainPanel.add(button_reset, cButtonReset);

// Action listener
button_file_input.addActionListener(e -> fileButtonPressed());
button_random_input.addActionListener(e -> randomButtonPressed());
button_start.addActionListener(e -> startButtonPressed());
button_reset.addActionListener(e -> resetButtonPressed());
```

```
    // Add Frame
    frame.add(mainPanel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("15 Puzzle Problem Solver");
    frame.pack();
    frame.setVisible(true);
}

// Action when file button is pressed
public void fileButtonPressed() {
    if (!inputExists) {
        JFileChooser fileChooser = new JFileChooser(System.getProperty("user.dir"));
        int status = fileChooser.showOpenDialog(frame);
        if (status == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            if (file == null) {
                return;
            }

            String fileName = fileChooser.getSelectedFile().getPath();
            System.out.println("Path: " + fileName);
            String convertedFileName = fileName.replace( target: "\\",  replacement: "\\\\");
            System.out.println("New Path: " + convertedFileName);
            try {
                matrix = Input.readFromFile(convertedFileName);
                if (Utilities.checkRepeatingElement(matrix)) {
                    printErrorMessage( text: "Not all matrix elements are distinct!");
                } else {
                    inputExists = true;
                    usingFile = true;
                    int counter = 0;
                    for (int i = 0; i < 4; i++) {
                        for (int j = 0; j < 4; j++) {
                            listOfTextFields.get(counter).setText(matrix.get(i).get(j).toString());
                            listOfTextFields.get(counter).setEditable(false);
                            counter++;
                        }
                    }
                }
            } catch (Exception e) {
                printErrorMessage( text: "File not found!");
            }
        }
    } else {
        printErrorMessage( text: "Choose only one method to get the matrix!");
    }
}
```

```java
// Action when random button is pressed
public void randomButtonPressed() {
    if (!inputExists) {
        boolean reachableMatrix = false;
        while (!reachableMatrix) {
            matrix = Input.createRandomMatrix();
            List<Object> reachableResult = Algorithm.isGoalReachable(matrix);
            boolean goal = (boolean)reachableResult.get(0);
            if (goal) {
                reachableMatrix = true;
            }
        }

        inputExists = true;
        usingRandom = true;
        int counter = 0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                listOfTextFields.get(counter).setText(matrix.get(i).get(j).toString());
                listOfTextFields.get(counter).setEditable(false);
                counter++;
            }
        }
    } else {
        printErrorMessage( text: "Choose only one method to get the matrix! Click the reset button to use another method");
    }
}
```

```java
// Action when reset button is pressed
public void resetButtonPressed() {
    usingFile = false;
    usingRandom = false;
    inputExists = false;
    for (int i = 0; i < 16; i++) {
        listOfTextFields.get(i).setText(null);
        listOfTextFields.get(i).setEditable(true);
    }
    outputArea.setText(null);
    timeTaken.setText("Time elapsed: 0 ms");
    printErrorMessage( text: "No Error Found!");
}

// Function to display text in the text area
public void printTextOutput(String text) {
    outputArea.setText(text);
}

// Function to display error messages
public void printErrorMessage(String text){
    errorLabel.setText("Error: " + text);
}
```

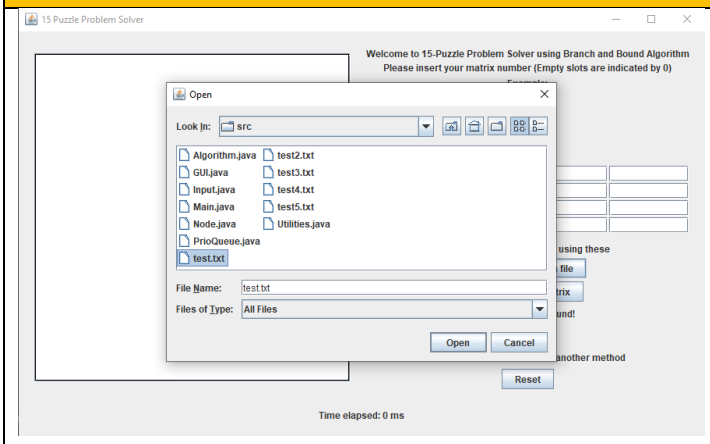
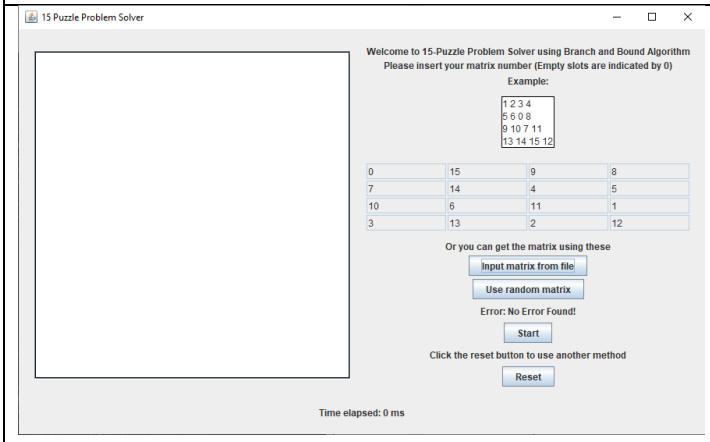
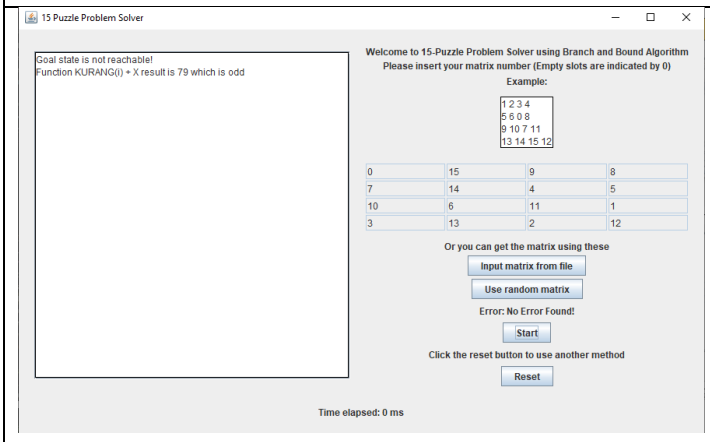
```java
// Action when start button is pressed
@SuppressWarnings("unchecked")
public void startButtonPressed() {
    boolean correct = true;
    // If user didn't use a file or create a random matrix, then read the entry from the text fields
    if (!usingFile && !usingRandom) {
        try {
            correct = false;
            ArrayList<Integer> listOfNumbers = new ArrayList<Integer>();
            for (JTextField textFields : listOfTextFields) {
                textFields.setEditable(false);
                String strNumber = textFields.getText();
                Integer number = Integer.valueOf(strNumber);
                listOfNumbers.add(number);
            }
            System.out.println(listOfNumbers);
            matrix = Utilities.convertListToMatrix(listOfNumbers);
            if (Utilities.checkRepeatingElement(matrix)) {
                printErrorMessage( text: "Not all matrix elements are distinct!");
            } else {
                correct = true;
            }
        } catch (Exception e) {
            printErrorMessage( text: "Please input the matrix using one method!");
            listOfTextFields.get(0).setEditable(true);
        }

    }
```

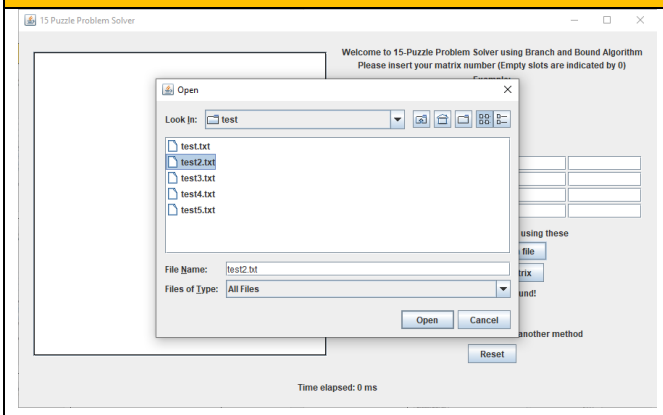
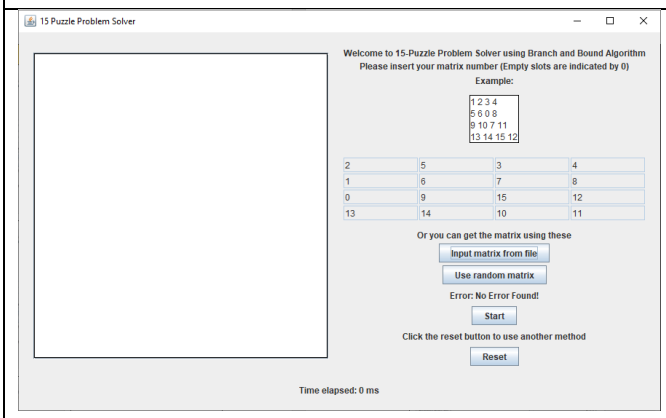
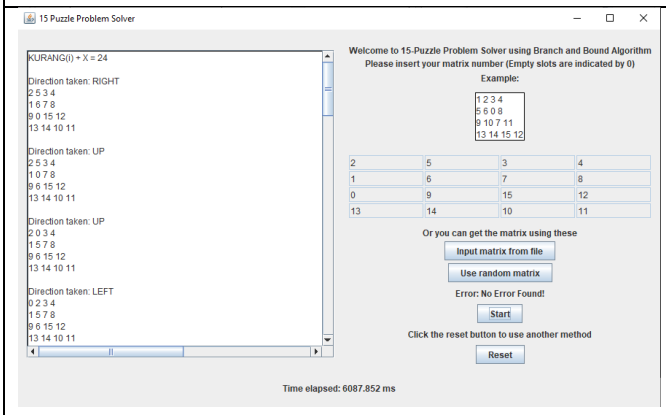
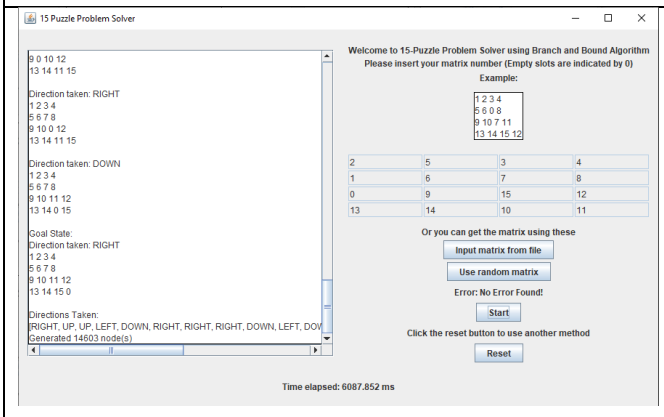
```java
    if (correct) {
        // Create a root node to start the algorithm
        startTime = System.nanoTime();
        ArrayList<String> direction = new ArrayList<String>();
        Node root = new Node(matrix,  cost: 0, direction);
        int result = (int)Algorithm.isGoalReachable(matrix).get(1);
        List<Object> algorithmResult = Algorithm.branchAndBoundAlgorithm(root);
        goalStates = (ArrayList<Node>) algorithmResult.get(0);
        int nodeCount = (int)algorithmResult.get(1);
        endTime = System.nanoTime();
        // If goalStates is empty, then function KURANG(X) + i result is odd
        if (goalStates.isEmpty()) {
            StringBuilder output = new StringBuilder();
            output.append("Goal state is not reachable!\n");
            output.append("Function KURANG(i) + X result is ").append(result).append(" which is odd");
            printTextOutput(output.toString());
        } else {
            String text = "KURANG(i) + X = " + result + "\n" + "\n";
            text += Utilities.displayGoalStates(matrix, goalStates);
            text += "\nGenerated " + nodeCount + " node(s)";
            printTextOutput(text);
            long difference = (endTime - startTime);
            float timeElapsed = (float)difference / 1000000;
            timeTaken.setText("Time elapsed: " + timeElapsed + " ms");
        }
    }
}
```

BAB III
Screenshot Input dan *Output*

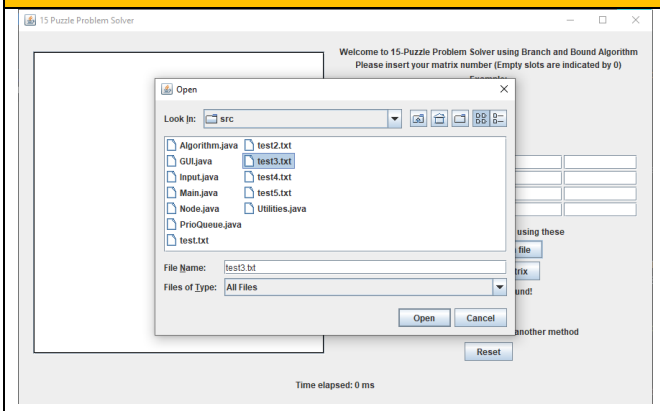
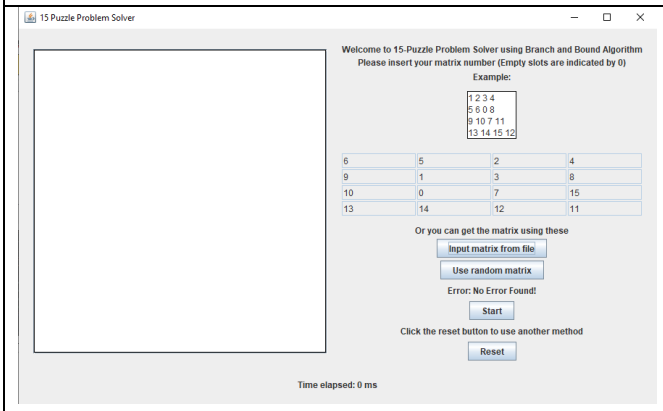
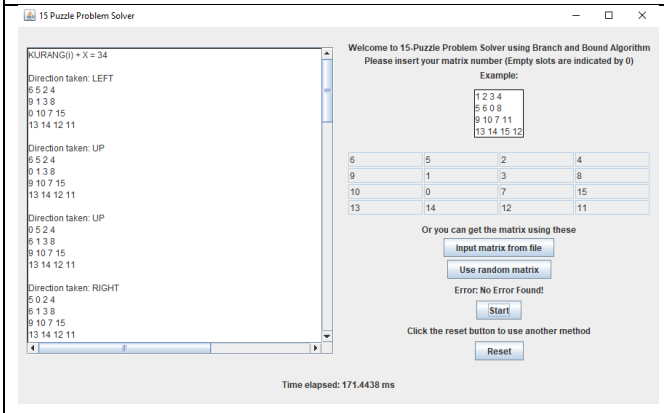
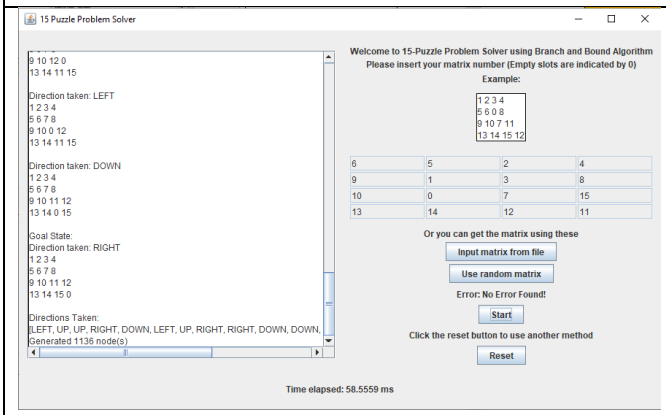
3.1.Menggunakan file test.txt

Gambar	Keterangan
	Memilih file **test.txt**
	Matriks yang terdapat di file **test.txt** muncul di preview matriks
	Hasil output program muncul di sebelah kiri. Algoritma tidak bisa dijalankan karena hasil nilai fungsi KURANG(i) + X ganjil.

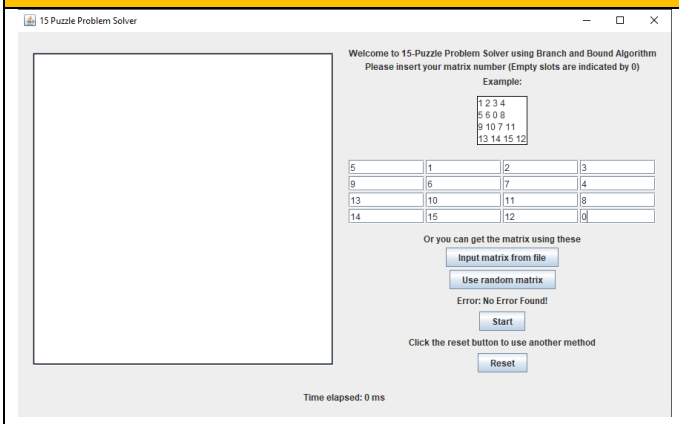
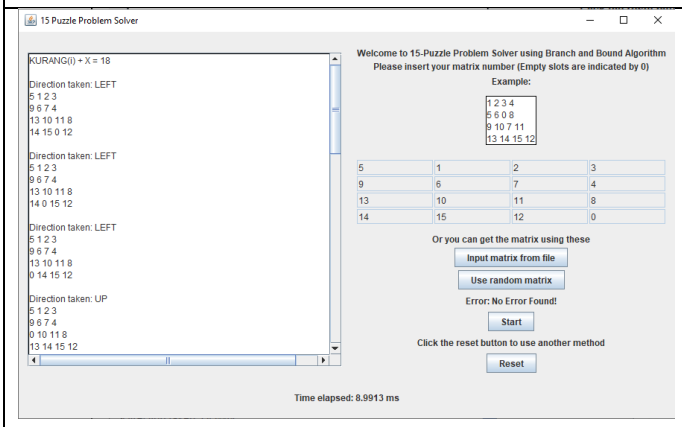
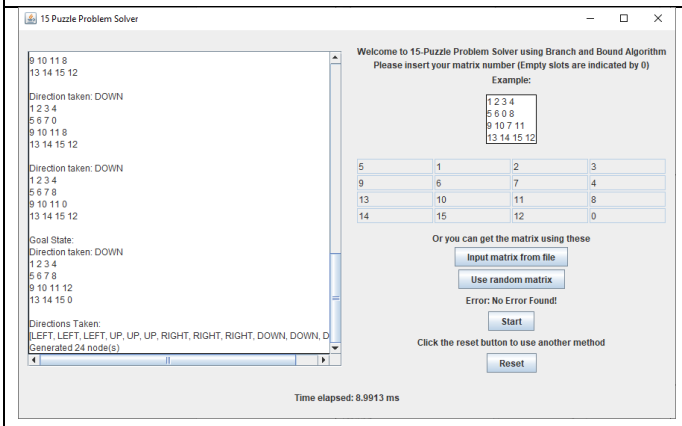
3.2.Menggunakan file test2.txt

Gambar	Keterangan
	Memilih file **test2.txt**
	Matriks yang terdapat di file **test2.txt** muncul di preview matriks
	Hasil output program muncul di sebelah kiri. Bagian atas menunjukkan hasil nilai fungsi KURANG(i) + X.
	Urutan arah yang diambil dari matriks awal ke matriks tujuan ditunjukkan di sebelah kiri. Jumlah simpul yang dibangkitkan juga ditunjukkan.

3.3.Menggunakan file test3.txt

Gambar	Keterangan
	Memilih file **test3.txt**
	Matriks yang terdapat di file **test3.txt** muncul di preview matriks
	Hasil output program muncul di sebelah kiri. Bagian atas menunjukkan hasil nilai fungsi KURANG(i) + X.
	Urutan arah yang diambil dari matriks awal ke matriks tujuan ditunjukkan di sebelah kiri. Jumlah simpul yang dibangkitan juga ditunjukkan.

3.4.Menggunakan input matriks secara langsung

Gambar	Keterangan
	Mengisi input matriks secara langsung
	Hasil output program muncul di sebelah kiri. Bagian atas menunjukkan hasil nilai fungsi KURANG(i) + X.
	Urutan arah yang diambil dari matriks awal ke matriks tujuan ditunjukkan di sebelah kiri. Jumlah simpul yang dibangkitan juga ditunjukkan.

BAB IV
Berkas Data Uji

test.txt (not solvable)
0 15 9 8 7 14 4 5 10 6 11 1 3 13 2 12
test2.txt (solvable)
2 5 3 4 1 6 7 8 16 9 15 12 13 14 10 11
test3.txt (solvable)
6 5 2 4 9 1 3 8 10 0 7 15 13 14 12 11
test4.txt (solvable)
2 3 7 4 1 6 11 8 5 10 12 15 9 13 14 0
test5.txt (not solvable)
15 10 12 2 3 4 14 0 13 9 8 5 7 6 11 1

BAB V
Checklist Program

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	✓	

BAB VI
Link Kode Program

Kode program dapat dilihat pada halaman github berikut

https://github.com/YakobusIP/Tucil3_13520104.git

BAB VII
Daftar Referensi

Munir, Rinaldi. 2022. Algoritma *Branch & Bound* (Bagian 1). Diakses pada 2 April 2022, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf