Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №2

з курсу «Автоматизація проектування комп'ютерних систем»

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

Тема

Автоматизація аналізу блок-схем алгоритмів.

Мета

Здобуття навичок з розробки та реалізації методів перевірки на помилки блок-схем алгоритмів.

Завдання

1. Представити номер залікової книжки в двійковому вигляді: $7308_{10} = 1110010001100_2$.
2. Реалізувати процедуру пошуку всіх шляхів та циклів у блок-схемі алгоритму, представленому в матричній формі.
3. В залежності від значення розрядів номера залікової книжки виявити наступні помилки в алгоритмі:

n_3	Тип перевірки
1	Виявити нескінчені цикли

4. При наявності помилок локалізувати місце помилки, виділити його у редакторі алгоритму, надати можливість корекції та повторної перевірки.

Опис програми

Для побудови та редагування блок-схем алгоритмів використовується програма з лабораторної роботи №1. Для реалізації процедури пошуку всіх шляхів та циклів у блок-схемі був написаний клас GSAWorker, що містить метод findWaysAndLoops(), що рекурсивно викликає приватний метод step(). З результатів виклику методу findWaysAndLoops() можна виявити нескінчені цикли в алгоритмі. Цикли є нескінченними, якщо в блок-схемі не знайдено ні одного шляху, але цикли є.

Для пошуку всіх шляхів та циклів в алгоритмі необхідно вибрати пункт головного меню “Analyze” -> ”Find All Ways And Loops…”. Результат роботи для блок-схеми алгоритму, що зображений на рисунку 1, показаний на рисунку 2.

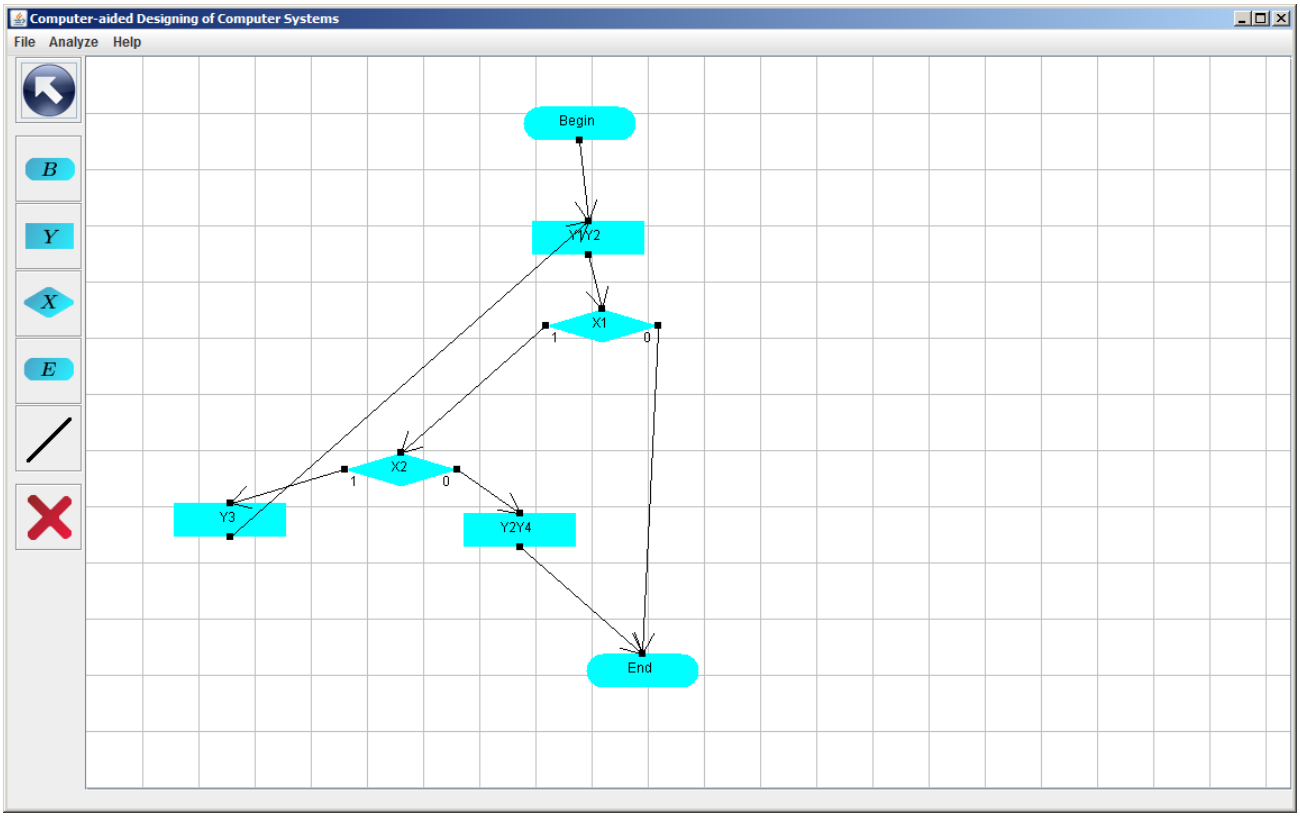

Рисунок 1

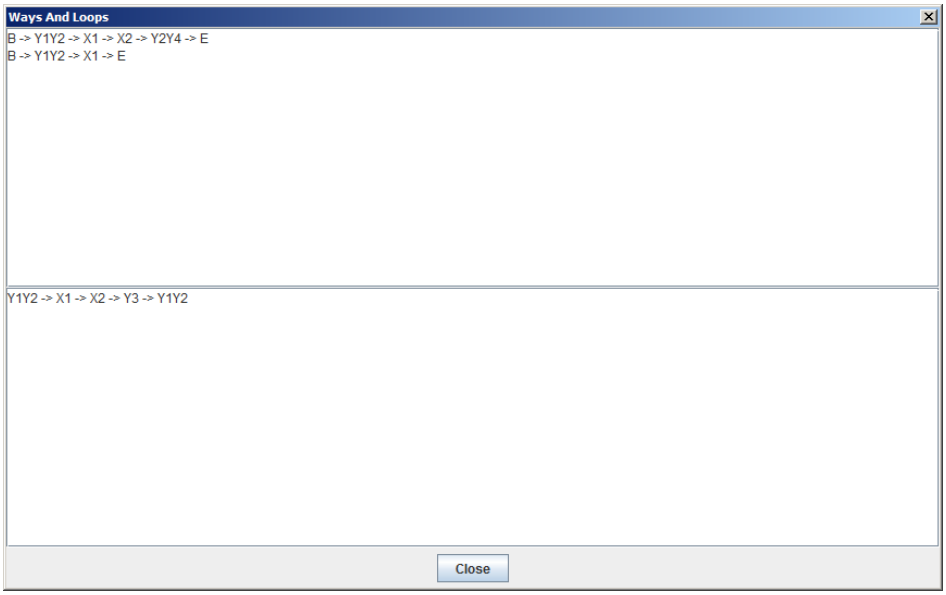

Рисунок 2

Для виявлення нескінченних циклів необхідно вибрати пункт головного меню “Analyze” -> ”Check GSA For Infinite Loops”. Для коректного алгоритму буде виведено повідомлення, що зображено на рисунку 3.

Рисунок 3

Для блок-схеми алгоритму, що містить нескінчені цикли, буде виведено повідомлення, що зображено на рисунку 4, та ці цикли будуть виділені в редакторі (рисунок 5).

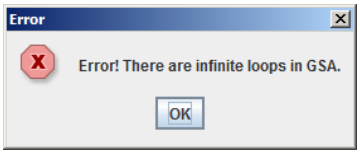

Рисунок 4

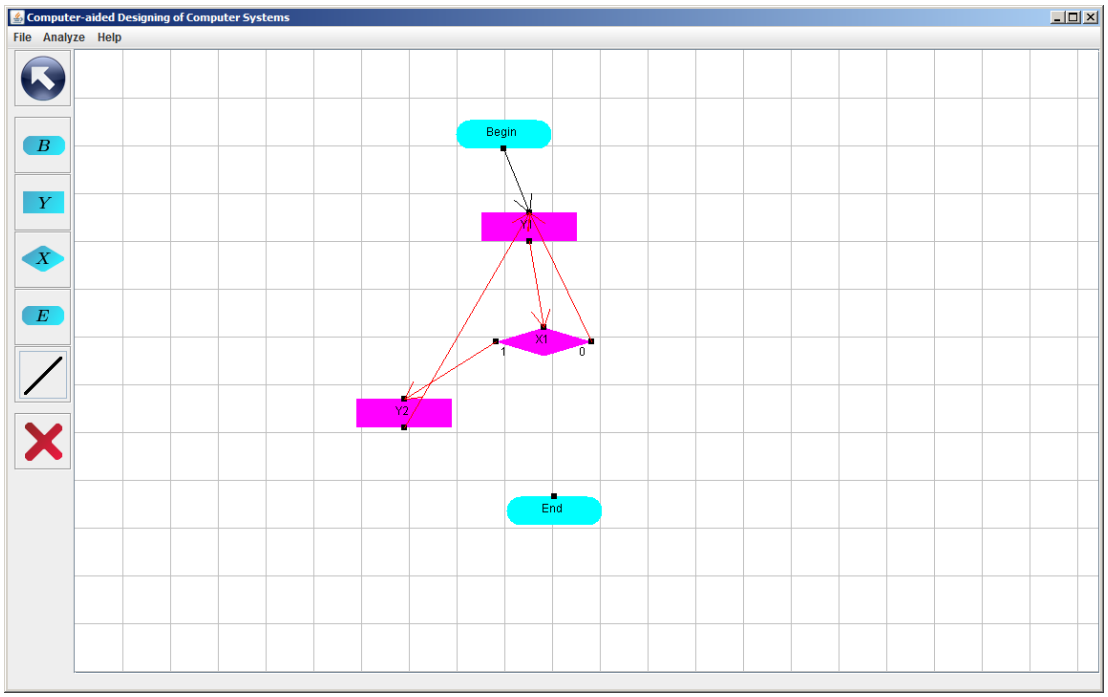

Рисунок 5

Лістинг програми

```
package gsa;

import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 03.10.2010
 * Time: 12:24:35
 * To change this template use File | Settings | File Templates.
 */
public class GSAWorker {

    private int[] nodesType;
    private int[][] connectionMatrix;
    private int[][] signals;
    private int beginNode;
    private int endNode;
    private ArrayList<ArrayList<Integer>> waysNodes;
    private ArrayList<ArrayList<Integer>> loopsNodes;
    private ArrayList<String> ways;
    private ArrayList<String> loops;

    public GSAWorker(int[] nodesType, int[][] connectionMatrix, int[][] signals) {
        this.nodesType = nodesType;
        this.connectionMatrix = connectionMatrix;
        this.signals = signals;
    }

    private void step(int currentNode, ArrayList<Integer> wayNodes, boolean[] visitedNodes, String way) {
        if (currentNode == endNode) {
            wayNodes.add(currentNode);
            way = way + " -> E";
            visitedNodes[currentNode] = true;
            ways.add(way);
            waysNodes.add(wayNodes);
        } else {
            if (!visitedNodes[currentNode]) {
                wayNodes.add(currentNode);
                visitedNodes[currentNode] = true;
                way = way + " -> ";
                if (nodesType[currentNode] == 1) {
                    for (int i = 0; i < signals[currentNode].length; i++) {
                        way = way + "Y" + String.valueOf(signals[currentNode][i]);
                    }
                } else {
                    if (nodesType[currentNode] == 2) {
                        way = way + "X" + String.valueOf(signals[currentNode][0]);
                    }
                }
                boolean findChild = false;
                int i = 0;
                while ((((!findChild) || (nodesType[currentNode] == 2)) && (i < connectionMatrix[currentNode].length)) {
                    if (connectionMatrix[currentNode][i] > 0) {
                        ArrayList<Integer> wayNodesClone = new ArrayList<Integer>();
                        for (int j = 0; j < wayNodes.size(); j++) {
                            wayNodesClone.add(wayNodes.get(j));
                        }
                        boolean[] visitedNodesClone = new boolean[visitedNodes.length];
                        for (int j = 0; j < visitedNodes.length; j++) {
                            visitedNodesClone[j] = visitedNodes[j];
                        }
                        String wayClone = new String(way);
                        findChild = true;
                        step(i, wayNodesClone, visitedNodesClone, wayClone);
                    }
                    i++;
                }
            } else {
                wayNodes.add(currentNode);
                int i = -1;
                boolean found = false;
                while (!found)
                {
                    i++;
                    if (wayNodes.get(i) == currentNode) {
                        found = true;
                    }
                }
                ArrayList<Integer> loopNodes = new ArrayList<Integer> ();
                loopNodes.add(wayNodes.get(i));
                StringBuilder builder = new StringBuilder();
                if (nodesType[wayNodes.get(i)] == 1) {
                    for (int k = 0; k < signals[wayNodes.get(i)].length; k++) {
                        builder.append("Y");
                        builder.append(String.valueOf(signals[wayNodes.get(i)][k]));
                    }
                }
                else {
                    if (nodesType[wayNodes.get(i)] == 2) {
                        builder.append("X");
                        builder.append(String.valueOf(signals[wayNodes.get(i)][0]));
                    }
                }
                for (int j = i + 1; j < wayNodes.size(); j++) {
                    loopNodes.add(wayNodes.get(j));
                    builder.append(" -> ");
                    if (nodesType[wayNodes.get(j)] == 1) {
                        for (int k = 0; k < signals[wayNodes.get(j)].length; k++) {
                            builder.append("Y");
                            builder.append(String.valueOf(signals[wayNodes.get(j)][k]));
                        }
                    }
                    else {
                        if (nodesType[wayNodes.get(j)] == 2) {
                            builder.append("X");
```

```
                        builder.append(String.valueOf(signals[wayNodes.get(j)][0]));
                    }
                }
            }
            loops.add(builder.toString());
            loopsNodes.add(loopNodes);
        }
    }
}

public void findWaysAndLoops() throws BeginEndNodesNotFoundException {
    boolean findBeginNode = false;
    boolean findEndNode = false;
    int i = 0;
    while (((!findBeginNode) || (!findEndNode)) && (i < nodesType.length)) {
        if (nodesType[i] == 0) {
            beginNode = i;
            findBeginNode = true;
        }
        if (nodesType[i] == 3) {
            endNode = i;
            findEndNode = true;
        }
        i++;
    }
    if ((!findBeginNode) && (!findEndNode)) {
        throw new BeginEndNodesNotFoundException(!findBeginNode, !findEndNode);
    }
    waysNodes = new ArrayList<ArrayList<Integer>>();
    loopsNodes = new ArrayList<ArrayList<Integer>>();
    ways = new ArrayList<String>();
    loops = new ArrayList<String>();
    int currentNode = beginNode;
    ArrayList<Integer> wayNodes = new ArrayList<Integer>();
    boolean[] visitedNodes = new boolean[nodesType.length];
    for (boolean e : visitedNodes) {
        e = false;
    }
    wayNodes.add(beginNode);
    visitedNodes[currentNode] = true;
    String way = "B";
    boolean findChild = false;
    i = 0;
    while ((!findChild) && (i < connectionMatrix[currentNode].length)) {
        if (connectionMatrix[currentNode][i] > 0) {
            ArrayList<Integer> wayNodesClone = new ArrayList<Integer>();
            for (int j = 0; j < wayNodes.size(); j++) {
                wayNodesClone.add(wayNodes.get(j));
            }
            boolean[] visitedNodesClone = new boolean[visitedNodes.length];
            for (int j = 0; j < visitedNodes.length; j++) {
                visitedNodesClone[j] = visitedNodes[j];
            }
            String wayClone = new String(way);
            findChild = true;
            step(i, wayNodesClone, visitedNodesClone, wayClone);
        }
        i++;
    }
}

public ArrayList<String> getWays() {
    return ways;
}

public ArrayList<String> getLoops() {
    return loops;
}

public ArrayList<ArrayList<Integer>> getWaysNodes() {
    return waysNodes;
}

public ArrayList<ArrayList<Integer>> getLoopsNodes() {
    return loopsNodes;
}

}
```

```
package gsa;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.10.2010
 * Time: 5:44:47
 * To change this template use File | Settings | File Templates.
 */
public class BeginEndNodesNotFoundException extends Exception {

    private static String BEGIN_TEXT = "Begin node ";
    private static String END_TEXT = "End node ";
    private static String BEGIN_END_TEXT = "Begin and End nodes ";
    private static String COMMON_TEXT1 = "has not been found.";
    private static String COMMON_TEXT2 = "have not been found.";

    private String text;

    public BeginEndNodesNotFoundException(boolean beginNode, boolean endNode) {
        if (beginNode && endNode) {
            text = BEGIN_END_TEXT + COMMON_TEXT2;
        }
        else {
            if (beginNode) {
                text = BEGIN_TEXT + COMMON_TEXT1;
            }
            else {
                if (endNode) {
                    text = END_TEXT + COMMON_TEXT1;
```

```
                }
            }
        }
    }

    @Override
    public String getMessage() {
        return text;
    }

}

package face;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 07.10.2010
 * Time: 0:05:33
 * To change this template use File | Settings | File Templates.
 */
class WaysAndLoopsFrame extends JDialog {

    private JTextArea waysArea;
    private JTextArea loopsArea;
    private JButton closeButton;

    public WaysAndLoopsFrame(MainFrame frame, Rectangle bounds, ArrayList<String> ways, ArrayList<String> loops) {
        super(frame);
        setBounds(bounds);
        setResizable(false);
        setModal(true);
        setTitle("Ways And Loops");
        setLayout(new BorderLayout());
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(2, 1));
        waysArea = new JTextArea();
        waysArea.setEditable(false);
        StringBuilder builder = new StringBuilder();
        if (!ways.isEmpty()) {
            for (int i = 0; i < (ways.size() - 1); i++) {
                builder.append(ways.get(i));
                builder.append("\n");
            }
            builder.append(ways.get(ways.size() - 1));
        }
        waysArea.setText(builder.toString());
        builder = new StringBuilder();
        loopsArea = new JTextArea();
        loopsArea.setEditable(false);
        if (!loops.isEmpty()) {
            for (int i = 0; i < (loops.size() - 1); i++) {
                builder.append(loops.get(i));
                builder.append("\n");
            }
            builder.append(loops.get(loops.size() - 1));
        }
        loopsArea.setText(builder.toString());
        mainPanel.add(new JScrollPane(waysArea));
        mainPanel.add(new JScrollPane(loopsArea));
        closeButton = new JButton(new AbstractAction () {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        closeButton.setText("Close");
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(closeButton);
        add(buttonPanel, BorderLayout.SOUTH);
        add(mainPanel);
    }

}

package face;

import gsa.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.concurrent.Executor;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 12.09.2010
 * Time: 1:57:43
 * To change this template use File | Settings | File Templates.
 */
public class MainFrame extends JFrame {

    private JMenuBar menuBar;
    private JToolBar toolBar;
    private GSAPanel gsaPanel;
    private JLabel statusLabel;
    private JFileChooser chooser;

    private NewAction newAction;
```

```
private OpenAction openAction;
private SaveAction saveAction;
private SaveAsAction saveAsAction;
private CloseAction closeAction;
private ExitAction exitAction;
private FindAllWaysAndLoopsAction findAllWaysAndLoopsAction;
private CheckForInfiniteLoopsAction checkForInfiniteLoopsAction;
private AboutAction aboutAction;

private File openedFile;

public MainFrame(Rectangle bounds) {
    super();
    setBounds(bounds);
    setTitle("Computer-aided Designing of Computer Systems");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    newAction = new NewAction(this);
    openAction = new OpenAction(this);
    saveAction = new SaveAction(this);
    saveAsAction = new SaveAsAction(this);
    closeAction = new CloseAction(this);
    exitAction = new ExitAction(this);
    findAllWaysAndLoopsAction = new FindAllWaysAndLoopsAction(this);
    checkForInfiniteLoopsAction = new CheckForInfiniteLoopsAction(this);
    aboutAction = new AboutAction(this);
    addWindowListener(new WindowHandler(exitAction));
    menuBar = new JMenuBar();
    JMenu fileMenu = new JMenu("File");
    JMenu analyzeMenu = new JMenu("Analyze");
    JMenu helpMenu = new JMenu("Help");
    JMenuItem tempItem = new JMenuItem(newAction);
    tempItem.setText("New");
    fileMenu.add(tempItem);
    tempItem = new JMenuItem(openAction);
    tempItem.setText("Open...");
    fileMenu.add(tempItem);
    tempItem = new JMenuItem(saveAction);
    tempItem.setText("Save");
    fileMenu.add(tempItem);
    tempItem = new JMenuItem(saveAsAction);
    tempItem.setText("Save As...");
    fileMenu.add(tempItem);
    tempItem = new JMenuItem(closeAction);
    tempItem.setText("Close");
    fileMenu.add(tempItem);
    fileMenu.addSeparator();
    tempItem = new JMenuItem(exitAction);
    tempItem.setText("Exit");
    fileMenu.add(tempItem);
    tempItem = new JMenuItem(findAllWaysAndLoopsAction);
    tempItem.setText("Find All Ways And Loops...");
    analyzeMenu.add(tempItem);
    tempItem = new JMenuItem(checkForInfiniteLoopsAction);
    tempItem.setText("Check GSA For Infinite Loops");
    analyzeMenu.add(tempItem);
    tempItem = new JMenuItem(aboutAction);
    tempItem.setText("About...");
    helpMenu.add(tempItem);
    menuBar.add(fileMenu);
    menuBar.add(analyzeMenu);
    menuBar.add(helpMenu);
    setJMenuBar(menuBar);
    setLayout(new BorderLayout());
    toolBar = new JToolBar(JToolBar.VERTICAL);
    toolBar.setFloatable(false);
    toolBar.setRollover(true);
    JButton tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(0);
            statusLabel.setText(" ");
        }
    });
    tempButton.setIcon(new ImageIcon("img/no_action.png"));
    tempButton.setToolTipText("No action");
    toolBar.addSeparator();
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(1);
            statusLabel.setText("Adding Begin Node");
        }
    });
    tempButton.setIcon(new ImageIcon("img/begin_node.png"));
    tempButton.setToolTipText("Begin Node");
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(2);
            statusLabel.setText("Adding Operator Node");
        }
    });
    tempButton.setIcon(new ImageIcon("img/operator_node.png"));
    tempButton.setToolTipText("Operator Node");
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(3);
            statusLabel.setText("Adding Logic Node");
        }
    });
    tempButton.setIcon(new ImageIcon("img/logic_node.png"));
    tempButton.setToolTipText("Logic Node");
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(4);
            statusLabel.setText("Adding End Node");
        }
    });
    tempButton.setIcon(new ImageIcon("img/end_node.png"));
    tempButton.setToolTipText("End Node");
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
```

```
            gsaPanel.getModel().setActionType(6);
            statusLabel.setText("Connecting nodes");
        }
    });
    tempButton.setIcon(new ImageIcon("img/line.png"));
    tempButton.setToolTipText("Connect nodes");
    toolBar.addSeparator();
    tempButton = toolBar.add(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            gsaPanel.getModel().setActionType(5);
            statusLabel.setText("Deleting");
        }
    });
    tempButton.setIcon(new ImageIcon("img/delete_node.png"));
    tempButton.setToolTipText("Delete");
    add(toolBar, BorderLayout.WEST);
    statusLabel = new JLabel(" ");
    add(statusLabel, BorderLayout.SOUTH);
    gsaPanel = new GSAPanel(new GSAModel(), this);
    add(new JScrollPane(gsaPanel));
    chooser = new JFileChooser();
    chooser.setCurrentDirectory(new File("."));
    chooser.addChoosableFileFilter(new GSAFileFilter());
    chooser.setMultiSelectionEnabled(false);
    openedFile = null;
}

private class NewAction extends AbstractAction {

    private MainFrame frame;

    public NewAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (gsaPanel.getModel().isChanged()) {
            closeAction.actionPerformed(e);
        }
        gsaPanel.setModel(new GSAModel());
        gsaPanel.setVisible(true);
        statusLabel.setText(" ");
        frame.repaint();
    }

}

private class OpenAction extends AbstractAction {

    private MainFrame frame;

    public OpenAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        int result = chooser.showOpenDialog(frame);
        if (result == JFileChooser.APPROVE_OPTION) {
            try {
                Scanner input = new Scanner(new BufferedReader(new FileReader(chooser.getSelectedFile())));
                int n = input.nextInt();
                int[] nodesTypeMatrix = new int[n];
                int[][] boundsMatrix = new int[n][];
                for (int i = 0; i < boundsMatrix.length; i++) {
                    boundsMatrix[i] = new int[4];
                }
                int[][] connectivityMatrix = new int[n][];
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    connectivityMatrix[i] = new int[n];
                }
                int[][] signalMatrix = new int[n][];
                for (int i = 0; i < nodesTypeMatrix.length; i++) {
                    nodesTypeMatrix[i] = input.nextInt();
                }
                for (int i = 0; i < boundsMatrix.length; i++) {
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        boundsMatrix[i][j] = input.nextInt();
                    }
                }
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        connectivityMatrix[i][j] = input.nextInt();
                    }
                }
                for (int i = 0; i < signalMatrix.length; i++) {
                    signalMatrix[i] = new int[input.nextInt()];
                }
                for (int i = 0; i < signalMatrix.length; i++) {
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        signalMatrix[i][j] = input.nextInt();
                    }
                }
                try {
                    gsaPanel.getModel().setGSA(nodesTypeMatrix, connectivityMatrix, signalMatrix, boundsMatrix);
                    gsaPanel.setVisible(true);
                    openedFile = chooser.getSelectedFile();
                    input.close();
                    statusLabel.setText(" ");
                    frame.repaint();
                } catch (IllegalNodeException e1) {
                    JOptionPane.showMessageDialog(frame, "Error! Incorrect GSA.",
                            "Error", JOptionPane.ERROR_MESSAGE);
                }
            } catch (FileNotFoundException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't open selected file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
```

```
        }
    }

}

private class SaveAction extends AbstractAction {

    private MainFrame frame;

    public SaveAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (openedFile == null) {
            saveAsAction.actionPerformed(e);
        }
        else {
            openedFile.delete();
            try {
                PrintWriter output = new PrintWriter(new FileWriter(openedFile));
                int[] nodesTypeMatrix = gsaPanel.getModel().getNodesType();
                int[][] boundsMatrix = gsaPanel.getModel().getBoundsMatrix();
                int[][] connectivityMatrix = gsaPanel.getModel().getConnectionMatrix();
                int[][] signalMatrix = gsaPanel.getModel().getSignalMatrix();
                output.println(nodesTypeMatrix.length);
                output.println();
                for (int i = 0; i < nodesTypeMatrix.length; i++) {
                    output.print(nodesTypeMatrix[i]);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < boundsMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        output.print(boundsMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.println();
                for (int i = 0; i < connectivityMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        output.print(connectivityMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.print("\n\n");
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.print(signalMatrix[i].length);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        output.print(signalMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.close();
                gsaPanel.getModel().setChanged(false);
            } catch (IOException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

}

private class SaveAsAction extends AbstractAction {

    private MainFrame frame;

    public SaveAsAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        int result = chooser.showSaveDialog(frame);
        if (result == JFileChooser.APPROVE_OPTION) {
            try {
                if (!chooser.getSelectedFile().getName().endsWith(GSAFileFilter.GSA_EXTENSION)) {
                    chooser.setSelectedFile(new File(chooser.getSelectedFile().getAbsolutePath() + GSAFileFilter.GSA_EXTENSION));
                }
                PrintWriter output = new PrintWriter(new FileWriter(chooser.getSelectedFile()));
                int[] nodesTypeMatrix = gsaPanel.getModel().getNodesType();
                int[][] boundsMatrix = gsaPanel.getModel().getBoundsMatrix();
                int[][] connectivityMatrix = gsaPanel.getModel().getConnectionMatrix();
                int[][] signalMatrix = gsaPanel.getModel().getSignalMatrix();
                output.println(nodesTypeMatrix.length);
                output.println();
                for (int i = 0; i < nodesTypeMatrix.length; i++) {
                    output.print(nodesTypeMatrix[i]);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < boundsMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < boundsMatrix[i].length; j++) {
                        output.print(boundsMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.println();
                for (int i = 0; i < connectivityMatrix.length; i++) {
```

```
                    output.println();
                    for (int j = 0; j < connectivityMatrix[i].length; j++) {
                        output.print(connectivityMatrix[i][j]);
                        output.print(" ");
                    }
                }
                output.print("\n\n");
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.print(signalMatrix[i].length);
                    output.print(" ");
                }
                output.println();
                for (int i = 0; i < signalMatrix.length; i++) {
                    output.println();
                    for (int j = 0; j < signalMatrix[i].length; j++) {
                        output.print(signalMatrix[i][j]);
                        output.print(" ");
                    }
                }
                openedFile = chooser.getSelectedFile();
                output.close();
                gsaPanel.getModel().setChanged(false);
            } catch (IOException e1) {
                JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

}

private class CloseAction extends AbstractAction {

    private MainFrame frame;

    public CloseAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (gsaPanel.getModel().isChanged()) {
            int result = JOptionPane.showConfirmDialog(frame,
                    "GSA has unsaved changes. Do you want to save them before closing?", "Warning",
                    JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE);
            if (result == JOptionPane.CANCEL_OPTION) {
                return;
            }
            else {
                if (result == JOptionPane.YES_OPTION) {
                    saveAction.actionPerformed(e);
                }
            }

        }
        gsaPanel.setVisible(false);
        frame.remove(gsaPanel);
        openedFile = null;
        statusLabel.setText(" ");
        frame.repaint();
    }

}

private class ExitAction extends AbstractAction {

    private MainFrame frame;

    public ExitAction(MainFrame frame) {
        super();
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        if (gsaPanel.getModel().isChanged()) {
            closeAction.actionPerformed(e);
        }
        System.exit(0);
    }

}

private class FindAllWaysAndLoopsAction extends AbstractAction {

    private MainFrame frame;

    private FindAllWaysAndLoopsAction(MainFrame frame) {
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        GSAWorker worker = new GSAWorker(gsaPanel.getModel().getNodesType(),
                gsaPanel.getModel().getConnectionMatrix(), gsaPanel.getModel().getSignalMatrix());
        try {
            worker.findWaysAndLoops();
            Rectangle bounds = new Rectangle(getX() + getWidth() / 8, getY() + getHeight() / 8,
                    getWidth() / 4 * 3, getHeight() / 4 * 3);
            WaysAndLoopsFrame dialog = new WaysAndLoopsFrame(frame, bounds, worker.getWays(), worker.getLoops());
            dialog.setVisible(true);
        } catch (BeginEndNodesNotFoundException e1) {
            JOptionPane.showMessageDialog(frame, e1.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

}

private class CheckForInfiniteLoopsAction extends AbstractAction {

    private MainFrame frame;
```

```
        public CheckForInfiniteLoopsAction(MainFrame frame) {
            super();
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            GSAWorker worker = new GSAWorker(gsaPanel.getModel().getNodesType(),
                    gsaPanel.getModel().getConnectionMatrix(), gsaPanel.getModel().getSignalMatrix());
            try {
                worker.findWaysAndLoops();
                if ((worker.getWays().isEmpty()) && (!worker.getLoops().isEmpty())) {
                    ArrayList<ArrayList<Integer>> loopsNodes = worker.getLoopsNodes();
                    ArrayList<Integer> nodesInLoops = new ArrayList<Integer>();
                    for (int i = 0; i < loopsNodes.size(); i++) {
                        for (int j = 0; j < loopsNodes.get(i).size(); j++) {
                            nodesInLoops.add(loopsNodes.get(i).get(j));
                        }
                    }
                    gsaPanel.getModel().setNodesInInfiniteCycles(nodesInLoops);
                    gsaPanel.repaint();
                    JOptionPane.showMessageDialog(frame, "Error! There are infinite loops in GSA.", "Error"
                            , JOptionPane.ERROR_MESSAGE);
                } else {
                    JOptionPane.showMessageDialog(frame, "Success! GSA has not infinite loops.", "Success"
                            , JOptionPane.PLAIN_MESSAGE);
                }
            } catch (BeginEndNodesNotFoundException e1) {
                JOptionPane.showMessageDialog(frame, e1.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        }

    }

    private class AboutAction extends AbstractAction {

        private MainFrame frame;

        public AboutAction(MainFrame frame) {
            super();
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,
                    "Computer-aided Designing of Computer Systems\nCopyright (c) 2010 Zakhozhyy Ihor",
                    "About", JOptionPane.INFORMATION_MESSAGE);
        }

    }

    private class WindowHandler extends WindowAdapter {

                    private ExitAction exitAction;

                    public WindowHandler(ExitAction exitAction) {
                              this.exitAction = exitAction;
                    }

                    public void windowClosing(final WindowEvent e) {
            final ActionEvent e2 = new ActionEvent(this, EXIT_ON_CLOSE, "close");
            exitAction.actionPerformed(e2);
            super.windowClosing(e);
                    }

          }

}
```

Висновки

В результаті даної лабораторної роботи я здобув навички з розробки методів перевірки на помилки блок-схем алгоритмів. Для побудови блок-схем алгоритмів використовувалася програма з лабораторної роботи №1. Мною був написаний клас, що містив методи для рекурсивного алгоритму пошуку всіх шляхів та циклів на блок-схемі. За допомогою результатів виконання цього пошуку можна виявити нескінчені цикли в алгоритмі. Також мною були написані методи для виділення помилок в редакторі. Для цього я використав засоби бібліотеки Swing.