Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №2**

*з предмету: «ТПКС»*

*по темі:*

«Автоматизація аналізу блок-схем алгоритмів»

Виконав: студент ФІОТ

групи ІО-92

Петрук В.О.

Київ 2012р.

**Мета:** Здобуття навичок з розробки та реалізації методів перевірки на помилки

блок-схем алгоритмів

**Завдання**

1. Представити номер залікової книжки в двійковому вигляді:

(NNNN)10=(n15n14…n2n1)2 = 100100000000112

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

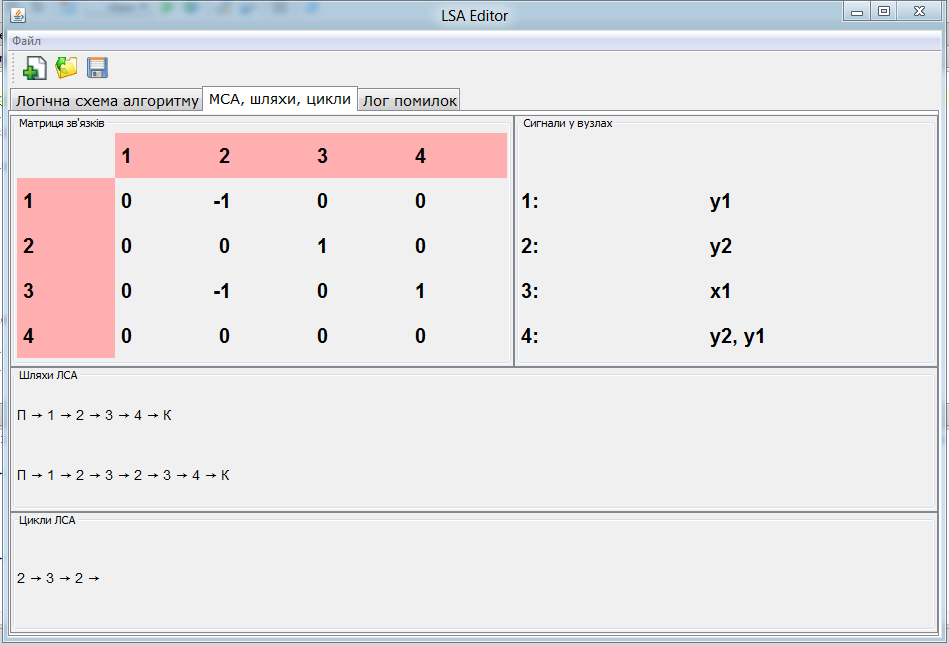
помилки в алгоритмі: n3=0: Виявити усі висячі (не мають вихідних зв’язків, крім кінцевого вузла) та недосяжні вузли (не мають вхідних зв’язків, крім початкового вузла)

4. При наявності помилок локалізувати місце помилки, виділити його у редакторі

алгоритму, надати можливість корекції та повторної перевірки.

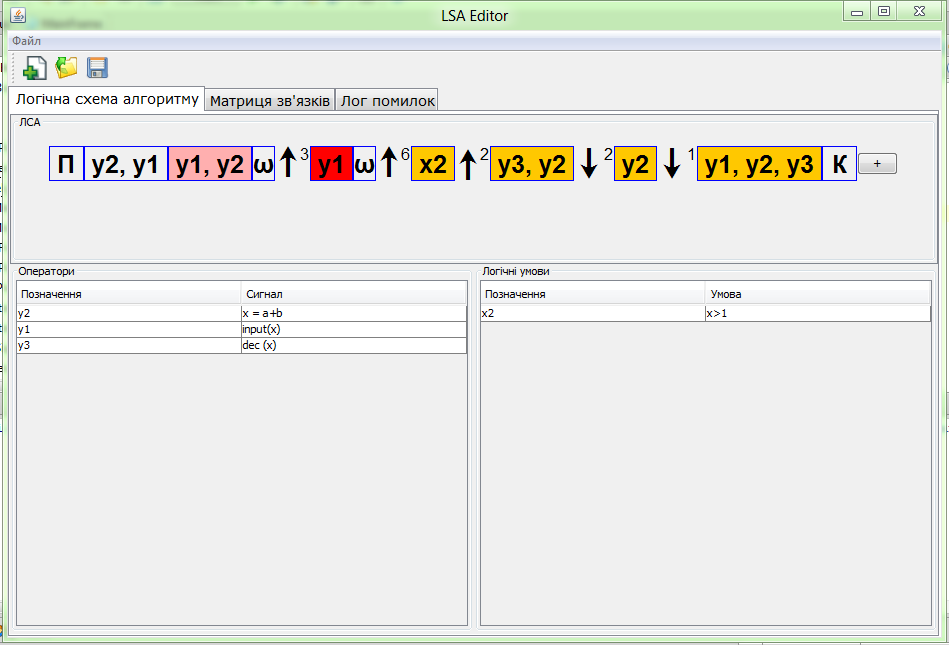
**Опис програми**

Пошук шляхів та циклів у блок-схемі алгоритму виконується за допомогою аналізу матричної форми алгоритму. Всі шляхи та цикли відображаються у вкладці «МСА, шляхи, цикли» програми на панелях «Шлях ЛСА» та «Цикли ЛСА» відповідно (рис. 1).



Пошук висячих та недосяжних вузлів алгоритму також виконується за допомогою аналізу матричної форми алгоритму.

У редакторі алгоритму недосяжні вершини виділяються помаранчевим, висячі – рожевим , а недосяжні та висячі одночасно - червоним кольором (рис. 2).

 Рисунок 2 – Виділення висячих та недосяжних вершин

Підказка, яким саме кольором виділяється та чи інша помилкова вершина показана у вкладці «Лог помилок» (рис. 3). У цій же вкладці відображаються інші помилки алгоритму, а саме:

* відстуність кінцевої вершини алгоритму;
* відсутність елементу повернення (↓і) з індексом, який вказаний у вузлах логічних чи безумовних переходів;
* відсутність логічних або безумовних вузлів з індексами переходу, які відповідать індексу елементу повернення.

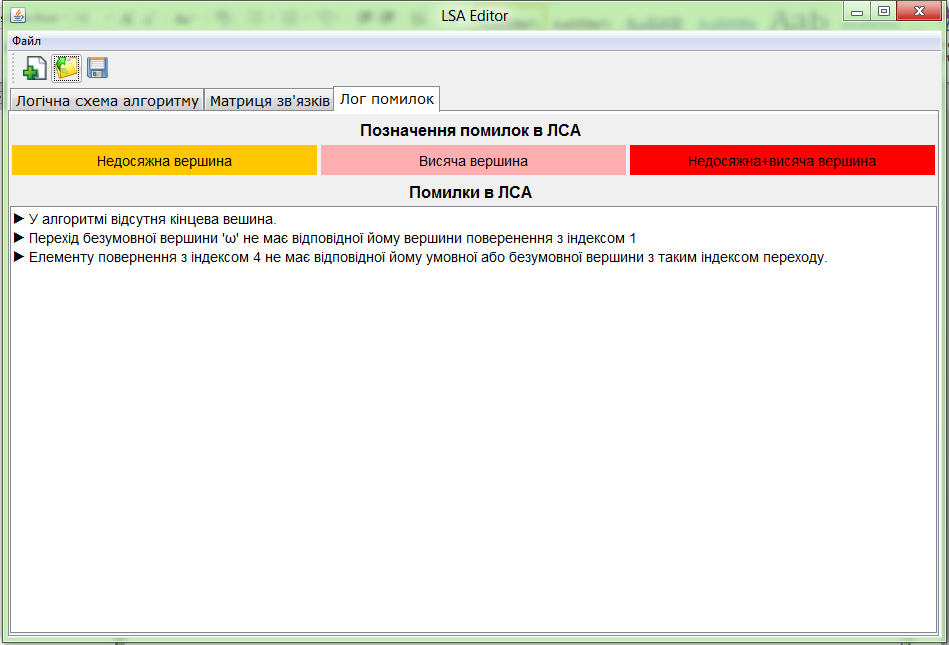


Рисунок 3 – Пенель логу помилок ЛСА

**Код програми**

Представлені лише класи та методи, які безпосередньо були реалізовані або змінені для виконання даної лабораторної роботи.

package lab1.gui;

public class DrawPanel extends JPanel {

/\*\*

\* Прорисовка висячих та недосяжних вершин ЛСА.

\*/

public void drawFailItems() {

// MSA

int[][] unitUnions = lsa.getMsa().getUnitsUnion();

//Масив станів вершин:

// 0 -норм (фон)

// 1 - недосяжна (blue)

// 2 - висяча (orange)

// 3 - недосяжна+висяча (red)

int[] itemArray = new int[unitUnions.length];

//шукаємо недосяжні вершини

for (int i = 1; i < itemArray.length; i++) {

if (isUnreachable(unitUnions, i)) {

itemArray[i] = 1;

nextItemUnreachable(unitUnions, itemArray, i);

}

}

//шукаємо висячі вершини

for (int i = 0; i < unitUnions.length - 1; i++) {

boolean isIsolatedItem = true;

for (int j = 0; j < unitUnions.length; j++) {

if (unitUnions[i][j] != 0) {

isIsolatedItem = false;

break;

}

}

if (isIsolatedItem) {

if (itemArray[i] == 1) {

itemArray[i] = 3;

} else {

itemArray[i] = 2;

}

}

}

//прораховуємо індекси JLable по матриці висячих та недосяжних вершин

int[] labelIndexes = new int[getComponentCount() - 1];//без початкової вершини

int count = 0;

for (int i = 1; i < getComponentCount() - 1; i++) {

JLabel jLabel = (JLabel) getComponent(i);

if ((labelUnitHashMap.get(jLabel).getUnitType().equals(UnitType.O)) ||

(labelUnitHashMap.get(jLabel).getUnitType().equals(UnitType.C))) {

labelIndexes[i] = itemArray[count];

count++;

}

//обчисення індексу наступного елементу

if (jLabel.getText() == "↓") {

i++;

} else {

if ((labelUnitHashMap.get(jLabel).getUnitType().equals(UnitType.C)) ||

(labelUnitHashMap.get(jLabel).getUnitType().equals(UnitType.U))) {

i += 2;

}

}

}

//прорисовка

for (int i = 1; i < labelIndexes.length; i++) {

switch (labelIndexes[i]) {

case 0:

getComponent(i).setBackground(getBackground());

break;

case 1:

getComponent(i).setBackground(Color.orange);

break;

case 2:

getComponent(i).setBackground(Color.pink);

break;

case 3:

getComponent(i).setBackground(Color.red);

break;

}

}

}

/\*\*

\* Перевірка чи вершина недосяжна

\*/

private boolean isUnreachable(int[][] unitUnion, int i) {

for (int j = 0; j < unitUnion.length; j++) {

if (unitUnion[j][i] != 0) {

return false;

}

}

return true;

}

/\*\*

\* Перевірка чи вершина висяча.

\*

\* @param unitUnions

\* @param itemArray

\* @param index

\*/

private void nextItemUnreachable(int[][] unitUnions, int[] itemArray, int index) {

//шукаємо куди переходить даний елемент

int pos = -1;

int spos = -1;//індекс 2ї вершини ( при умовному переході)

for (int i = 0; i < unitUnions.length; i++) {

if (pos == -1) {

if (unitUnions[index][i] != 0) {

pos = i;

}

} else {

if (unitUnions[index][i] != 0) {

spos = i;

break;

}

}

}

if (pos != -1) {

boolean flag = false;

for (int i = 0; i < unitUnions.length; i++) {

if ((unitUnions[i][pos] != 0) && (itemArray[i] != 1)) {

flag = true;

}

}

if (!flag) {

itemArray[pos] = 1;

nextItemUnreachable(unitUnions, itemArray, pos);

}

}

if (spos != -1) {

boolean flag = false;

for (int i = 0; i < unitUnions.length; i++) {

if ((unitUnions[i][spos] != 0) && (itemArray[i] != 1)) {

flag = true;

}

}

if (!flag) {

itemArray[spos] = 1;

nextItemUnreachable(unitUnions, itemArray, spos);

}

}

}

}

package lab1.gui;

import lab1.MSA;

import lab1.Unit;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.util.ArrayList;

public class MSAPanel extends JPanel {

private GridBagConstraints gridBagConstraints = new GridBagConstraints();

private MSA msa;

private JPanel linkMatrixPanel;

private JPanel markingPanel;

private JPanel waysPanel;

private JPanel cyclePanel;

private Font font = new Font("TimesRoman", Font.BOLD, 20);

private ArrayList<ArrayList<Integer>> ways;

private ArrayList<ArrayList<Integer>> cycles;

public MSAPanel() {

setLayout(new GridBagLayout());

linkMatrixPanel = new JPanel();

linkMatrixPanel.setLayout(new GridBagLayout());

linkMatrixPanel.setBorder(new TitledBorder("Матриця зв'язків"));

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 0, 1, 1, 1, 1);

add(new JScrollPane(linkMatrixPanel), gridBagConstraints);

markingPanel = new JPanel();

markingPanel.setLayout(new GridBagLayout());

markingPanel.setBorder(new TitledBorder("Сигнали у вузлах"));

setGridBagParam(1, 0, 1, 1, 1, 1);

add(new JScrollPane(markingPanel), gridBagConstraints);

waysPanel = new JPanel();

waysPanel.setLayout(new GridBagLayout());

waysPanel.setBorder(new TitledBorder("Шляхи ЛСА"));

setGridBagParam(0, 1, 2, 1, 1, 1);

add(new JScrollPane(waysPanel), gridBagConstraints);

cyclePanel = new JPanel();

cyclePanel.setLayout(new GridBagLayout());

cyclePanel.setBorder(new TitledBorder("Цикли ЛСА"));

setGridBagParam(0, 2, 2, 1, 1, 1);

add(new JScrollPane(cyclePanel), gridBagConstraints);

}

public void setMSA(MSA msa) {

this.msa = msa;

repaintMSApanel();

}

private void repaintMSApanel() {

setGridBagParam(1, 1, 1, 1, 1, 1);

setLinkMatrixPanel();

setMarkingPanel();

setWaysPanel();

setCyclePanel();

repaint();

revalidate();

}

/\*\*

\* Створення матриці звязків вершин.

\*/

private void setLinkMatrixPanel() {

linkMatrixPanel.removeAll();

JLabel rowLabel;

JLabel columnLabel;

for (int j = 1; j <= msa.getUnitsUnion().length; j++) {

gridBagConstraints.gridy = 0;

gridBagConstraints.gridx = j;

rowLabel = new JLabel(" " + j + " ");

rowLabel.setFont(font);

rowLabel.setOpaque(true);

rowLabel.setBackground(Color.pink);

linkMatrixPanel.add(rowLabel, gridBagConstraints);

columnLabel = new JLabel(" " + j + " ");

columnLabel.setFont(font);

columnLabel.setOpaque(true);

columnLabel.setBackground(Color.pink);

gridBagConstraints.gridy = j;

gridBagConstraints.gridx = 0;

linkMatrixPanel.add(columnLabel, gridBagConstraints);

}

for (int i = 0; i < msa.getUnitsUnion().length; i++) {

for (int j = 0; j < msa.getUnitsUnion().length; j++) {

gridBagConstraints.gridx = j + 1;

gridBagConstraints.gridy = i + 1;

String beforeSpace;

String aferSpaces;

if (msa.getUnitsUnion()[i][j] == -1) {

beforeSpace = "";

aferSpaces = " ";

} else {

beforeSpace = " ";

aferSpaces = " ";

}

JLabel jLabel = new JLabel(beforeSpace + msa.getUnitsUnion()[i][j] + aferSpaces);

jLabel.setFont(font);

linkMatrixPanel.add(jLabel, gridBagConstraints);

}

}

}

/\*\*

\* Заповнює панель для сигнлаів у вузлах алгоритму..

\*/

private void setMarkingPanel() {

markingPanel.removeAll();

JLabel numbLabel;

JLabel markingLabel;

String markingInItem;

JLabel nullLabel = new JLabel(" ");

nullLabel.setFont(font);

setGridBagParam(0, 0, 1, 1, 1, 1);

markingPanel.add(nullLabel, gridBagConstraints);

for (int i = 0; i < msa.getUnitSignals().size(); i++) {

numbLabel = new JLabel((i + 1) + ": ");

numbLabel.setFont(font);

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = i + 1;

markingPanel.add(numbLabel, gridBagConstraints);

gridBagConstraints.gridx = 1;

markingInItem = "";

for (int j = 0; j < msa.getUnitSignals().get(i).getUnitMarking().length; j++) {

String comma = ", ";

if (j == msa.getUnitSignals().get(i).getUnitMarking().length - 1) {

comma = "";

}

markingInItem += (msa.getUnitSignals().get(i).getUnitMarking()[j] + comma);

}

markingLabel = new JLabel(markingInItem);

markingLabel.setFont(font);

markingLabel.setHorizontalAlignment(JLabel.LEFT);

gridBagConstraints.gridy = i + 1;

markingPanel.add(markingLabel, gridBagConstraints);

}

}

/\*\*

\* Заповнює панель для відображення шляхів ЛСА.

\*/

private void setWaysPanel() {

waysPanel.removeAll();

ways = new ArrayList<ArrayList<Integer>>();

cycles = new ArrayList<ArrayList<Integer>>();

ArrayList<Integer> way1 = new ArrayList<Integer>();

ways.add(way1);

way1.add(0);

serchNext(0, way1, -1);

JLabel wayLabel;

for (int i = 0; i < ways.size(); i++) {

String s = "П → ";

if (msa.getUnitsUnion().length != 0) {

for (Integer integer : ways.get(i)) {

s += ((integer + 1) + " → ");

}

}

s += "К";

System.out.println(s);

wayLabel = new JLabel(s);

wayLabel.setFont(new Font("TimesRoman", Font.PLAIN, 14));

setGridBagParam(0, i + 1, 1, 1, 1, 1);

waysPanel.add(wayLabel, gridBagConstraints);

}

}

/\*\*

\* Шукає, куди відбувається перехід з даного вузла.

\*/

private int[] serchNext(int thisIndex, ArrayList<Integer> thisWay, int secIndex) {

int[] indexes = {-1, -1};

int count = 0;

for (int i = 0; i < msa.getUnitsUnion().length; i++) {

if (msa.getUnitsUnion()[thisIndex][i] == -1) {

count++;

}

}

for (int i = 0; i < msa.getUnitsUnion().length; i++) {

if (msa.getUnitsUnion()[thisIndex][i] == 1) {

indexes[0] = i;

}

if (msa.getUnitsUnion()[thisIndex][i] == -1) {

indexes[1] = i;

}

}

if ((indexes[1] != -1) && (indexes[0] == -1)) {

indexes[0] = indexes[1];

indexes[1] = -1;

}

if (indexes[0] != -1) {

boolean flag = true;

if (indexes[1] != -1) {

if (count <= 1) {

if (secIndex != thisIndex) {

ArrayList<Integer> newWay = (ArrayList<Integer>) thisWay.clone();

ways.add(newWay);

if (indexes[1] != msa.getUnitsUnion().length) {

newWay.add(indexes[1]);

serchNext(indexes[1], newWay, thisIndex);

}

} else {

if ((indexes[0] != msa.getUnitsUnion().length)) {

ArrayList<Integer> newCycle = (ArrayList<Integer>) thisWay.clone();

cycles.add(newCycle);

thisWay.add(indexes[0]);

serchNext(indexes[0], thisWay, -1);

flag = false;

}

}

} else {

ArrayList<Integer> newWay = (ArrayList<Integer>) thisWay.clone();

ways.add(newWay);

}

}

if ((indexes[0] != msa.getUnitsUnion().length) && (flag)) {

thisWay.add(indexes[0]);

serchNext(indexes[0], thisWay, secIndex);

// }

}

}

return indexes;

}

/\*\*

\* Заповнює панель для відображення циклів ЛСА.

\*/

private void setCyclePanel() {

cyclePanel.removeAll();

for (int i = 0; i < cycles.size(); i++) {

for (int j = 0; j < cycles.get(i).size() - 1; j++) {

boolean flag = false;

for (int k = j + 1; k < cycles.get(i).size(); k++) {

if (cycles.get(i).get(j).equals(cycles.get(i).get(k))) {

//видалаяємо все після k

for (int l = k; l < cycles.get(i).size(); l++) {

cycles.get(i).remove(cycles.get(i).size() - 1);

}

//видаляємо все до j

for (int l = 0; l < j; l++) {

cycles.get(i).remove(0);

}

flag = true;

}

}

if (flag) {

break;

}

}

}

JLabel cycleLabel;

for (int i = 0; i < cycles.size(); i++) {

String s = "";

for (Integer integer : cycles.get(i)) {

s += ((integer + 1) + " → ");

}

System.out.println(s);

cycleLabel = new JLabel(s);

cycleLabel.setFont(new Font("TimesRoman", Font.PLAIN, 14));

setGridBagParam(0, i + 1, 1, 1, 1, 1);

cyclePanel.add(cycleLabel, gridBagConstraints);

}

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Unit;

import lab1.UnitType;

import javax.swing.\*;

import java.awt.\*;

/\*\*

\* Перелік тестів, для контролю правильності введеного алгоритму.

\*/

enum TestType {

IS\_END, IS\_RETURN\_AFTER\_END, CONDITION\_INDEX\_NOT\_CORRESPONDS\_RETURN\_INDEX,

UNCONDITION\_INDEX\_NOT\_CORRESPONDS\_RETURN\_INDEX, FOREVER\_ALONE\_RETURN\_INDEX

}

/\*\*

\* Панель відображення помилок логічного алгоритму.

\*/

public class LogPanel extends JPanel {

private JTextArea jTextArea;

private GridBagConstraints gridBagConstraints;

private Font font = new Font("TimesRoman", Font.PLAIN, 14);

private Font fontTitle = new Font("TimesRoman", Font.BOLD, 16);

public LogPanel() {

setLayout(new GridBagLayout());

gridBagConstraints = new GridBagConstraints();

Dimension labelSize = new Dimension(300, 25);

JLabel errorLabel = new JLabel("Позначення помилок в ЛСА");

JLabel unreachableItemLabel = new JLabel("Недосяжна вершина");

JLabel isolatedItemLabel = new JLabel("Висяча вершина");

JLabel UIItemLabel = new JLabel("Недосяжна+висяча вершина");

JLabel otherErrorLabel = new JLabel("Помилки в ЛСА");

jTextArea = new JTextArea();

jTextArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(jTextArea);

unreachableItemLabel.setPreferredSize(labelSize);

isolatedItemLabel.setPreferredSize(labelSize);

UIItemLabel.setPreferredSize(labelSize);

unreachableItemLabel.setHorizontalAlignment(JLabel.CENTER);

unreachableItemLabel.setVerticalAlignment(JLabel.CENTER);

isolatedItemLabel.setHorizontalAlignment(JLabel.CENTER);

isolatedItemLabel.setVerticalAlignment(JLabel.CENTER);

UIItemLabel.setHorizontalAlignment(JLabel.CENTER);

UIItemLabel.setVerticalAlignment(JLabel.CENTER);

unreachableItemLabel.setOpaque(true);

isolatedItemLabel.setOpaque(true);

UIItemLabel.setOpaque(true);

unreachableItemLabel.setBackground(Color.orange);

isolatedItemLabel.setBackground(Color.pink);

UIItemLabel.setBackground(Color.red);

errorLabel.setFont(fontTitle);

unreachableItemLabel.setFont(font);

isolatedItemLabel.setFont(font);

UIItemLabel.setFont(font);

otherErrorLabel.setFont(fontTitle);

jTextArea.setFont(font);

setGridBagParam(0, 0, 3, 1, 1, 1);

add(errorLabel, gridBagConstraints);

setGridBagParam(0, 1, 1, 1, 1, 1);

add(unreachableItemLabel, gridBagConstraints);

setGridBagParam(1, 1, 1, 1, 1, 1);

add(isolatedItemLabel, gridBagConstraints);

setGridBagParam(2, 1, 1, 1, 1, 1);

add(UIItemLabel, gridBagConstraints);

setGridBagParam(0, 2, 3, 1, 1, 1);

add(otherErrorLabel, gridBagConstraints);

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 3, 3, 1, 1, 100);

add(scrollPane, gridBagConstraints);

}

/\*\*

\* Перевіряє ЛСА даного проекту на помилки.

\*

\* @param lsa

\*/

public void checkLSAforErrors(LSA lsa) {

jTextArea.setText("");

for (TestType testType : TestType.values()) {

switch (testType) {

case IS\_END:

jTextArea.setText(isEndTest(lsa));

break;

case CONDITION\_INDEX\_NOT\_CORRESPONDS\_RETURN\_INDEX:

jTextArea.setText(jTextArea.getText() + condCorrespondReturn(lsa));

break;

case UNCONDITION\_INDEX\_NOT\_CORRESPONDS\_RETURN\_INDEX:

jTextArea.setText(jTextArea.getText() + uncondCorrespondReturn(lsa));

break;

case FOREVER\_ALONE\_RETURN\_INDEX:

jTextArea.setText(jTextArea.getText() + foreverAloneReturnIndex(lsa));

break;

}

}

}

/\*\*

\* Первіряє, чи є кінцева вершина в алгоритмі.

\*

\* @param lsa

\* @return Повідомлення про помилку відсутності кінцевої вершини.

\*/

private String isEndTest(LSA lsa) {

for (Unit unit : lsa.getLsaItems()) {

if (unit.getUnitType().equals(UnitType.E)) {

return "";

}

}

return "► У алгоритмі відсутня кінцева вешина.\n";

}

/\*\*

\* Перевірка, чи індексам умовних переходів відповідають індекси поверення.

\*

\* @param lsa

\* @return

\*/

private String condCorrespondReturn(LSA lsa) {

String s = "";

for (Unit unit : lsa.getLsaItems()) {

if (unit.getUnitType().equals(UnitType.C)) {

boolean flag = true;

for (Unit unit1 : lsa.getLsaItems()) {

if ((unit1.getUnitType().equals(UnitType.R)) && (unit1.getIndex() == unit.getIndex())) {

flag = false;

break;

}

}

if (flag) {

s += ("► Перехід умовної вершини '" + unit.getUnitMarking()[0] + "' не має відповідної йому вершини " +

"поверенення з індексом " + unit.getIndex() + "\n");

}

}

}

return s;

}

/\*\*

\* Перевірка, чи індексам безумовних переходів відповідають індекси поверення.

\*

\* @param lsa

\* @return

\*/

private String uncondCorrespondReturn(LSA lsa) {

String s = "";

for (Unit unit : lsa.getLsaItems()) {

if (unit.getUnitType().equals(UnitType.U)) {

boolean flag = true;

for (Unit unit1 : lsa.getLsaItems()) {

if ((unit1.getUnitType().equals(UnitType.R)) && (unit1.getIndex() == unit.getIndex())) {

flag = false;

break;

}

}

if (flag) {

s += ("► Перехід безумовної вершини 'ω' не має відповідної йому вершини " +

"поверенення з індексом " + unit.getIndex() + "\n");

}

}

}

return s;

}

/\*\*

\* Перевірка чи існують иепеходи на вершини повернення.

\*

\* @param lsa

\* @return

\*/

private String foreverAloneReturnIndex(LSA lsa) {

String s = "";

for (Unit unit : lsa.getLsaItems()) {

if (unit.getUnitType().equals(UnitType.R)) {

boolean flag = true;

for (Unit unit1 : lsa.getLsaItems()) {

if (((unit1.getUnitType().equals(UnitType.C)) || (unit1.getUnitType().equals(UnitType.U))) &&

(unit1.getIndex() == unit.getIndex())) {

flag = false;

break;

}

}

if (flag) {

s += ("► Елементу повернення з індексом " + unit.getIndex() + " не має відповідної йому умовної " +

"або безумовної вершини з таким індексом переходу.\n");

}

}

}

return s;

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

}

**Висновок**

Реалізована процедура пошуку шляхів та циклів алгоритму, що представлений у матричній формі. Результати роботи процедури (шляхи та цикли) показуються в графічному інтерфейсі користувача.

Розроблено алгоритм пошуку помилок (висячі та недосяжні вершини) в ЛСА, представленому в матричній формі. Виявлені помилки відображаються у редакторі алгоритму та автоматично оновлюються при редагуванні елементів алгоритму.