НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №3

з дисципліни **«**Комп’ютерне моделювання**»**

Виконав:

студент групи ІО-42, ФІОТ

Коваленко В’ячеслав Сергійович

ЗК : 4209

м. Київ 2016 р.

Завдання



Лістинг програми

public class Main {

private static List<Unit> allUnitList = new ArrayList<>();

private static String[] currentSequence = {"210", "001", "001", "001", "001", "001", "001", "001", "001", "001", "001"};

private static List<Vertex> vertexList = new ArrayList<>();

private static final int COUNT = currentSequence.length;

private static double[][] mainMatrix;

private static double[] lastCol;

private static double[] resultPossibilities;

private static double[] result = new double[COUNT];

public static void main(String[] args) {

initialSetup();

buildTree(currentSequence);

printVertexList(0, 0);

generateMatrix();

printPossibilities();

}

private static void initialSetup() {

int index = 0;

Unit cpu = new Unit(1, index++);

Unit northBridge = new Unit(1, index++);

Unit ram = new Unit(20., index++);

Unit graphicProcessor = new Unit(100., index++);

Unit southBridge = new Unit(50., index++);

Unit networkAdapter = new Unit(100., index++);

Unit audioProcessor = new Unit(100., index++);

Unit cmd = new Unit(200., index++);

Unit isa = new Unit(20., index++);

Unit cod = new Unit(200., index++);

Unit com = new Unit(100., index++);

allUnitList.add(cpu);

allUnitList.add(northBridge);

allUnitList.add(ram);

allUnitList.add(graphicProcessor);

allUnitList.add(southBridge);

allUnitList.add(networkAdapter);

allUnitList.add(audioProcessor);

allUnitList.add(cmd);

allUnitList.add(isa);

allUnitList.add(cod);

allUnitList.add(com);

cpu.setCoUnits(northBridge.getIndex());

cpu.setCoUnitsPossibilities(1.);

northBridge.setCoUnits(southBridge.getIndex(), ram.getIndex(), graphicProcessor.getIndex(), cpu.getIndex());

northBridge.setCoUnitsPossibilities(0.7, 0.1, 0.1, 0.1);

ram.setCoUnits(northBridge.getIndex());

ram.setCoUnitsPossibilities(1.0);

graphicProcessor.setCoUnits(northBridge.getIndex());

graphicProcessor.setCoUnitsPossibilities(1.0);

southBridge.setCoUnits(cmd.getIndex(), networkAdapter.getIndex(), isa.getIndex(), cod.getIndex(), audioProcessor.getIndex(), northBridge.getIndex());

southBridge.setCoUnitsPossibilities(0.2, 0.2, 0.2, 0.2, 0.1, 0.1);

networkAdapter.setCoUnits(southBridge.getIndex());

networkAdapter.setCoUnitsPossibilities(1.0);

audioProcessor.setCoUnits(cpu.getIndex());

audioProcessor.setCoUnitsPossibilities(1.0);

cmd.setCoUnits(southBridge.getIndex());

cmd.setCoUnitsPossibilities(1.0);

isa.setCoUnits(southBridge.getIndex(), com.getIndex());

isa.setCoUnitsPossibilities(0.8, 0.2);

cod.setCoUnits(southBridge.getIndex());

cod.setCoUnitsPossibilities(1.0);

com.setCoUnits(cpu.getIndex());

com.setCoUnitsPossibilities(1.0);

}

private static void buildTree(String[] currentSequence) {

ArrayList<Sequence> currentSequenceList = new ArrayList<>();

ArrayList<Sequence> nextSequenceList = new ArrayList<>();

currentSequenceList.add(new Sequence(currentSequence, "M0"));

vertexList.add(new Vertex(vertexList.size(), currentSequence));

boolean flagSuccess = true;

while (flagSuccess) {

if (currentSequenceList.size() != 0) {

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

int currentVertexPosition = getVertexPositionWithTitle(currentSequenceList.get(i).getTitle());

// make copy of current sequence

String[] temporarySequence = new String[COUNT];

System.arraycopy(currentSequenceList.get(i).getSequence(), 0, temporarySequence, 0, COUNT);

// generate next vertexes

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

if (!temporarySequence[j].equals("001")) {

List<Integer> nextUnitsIndexes = allUnitList.get(j).getNextUnitList();

List<Double> nextUnitsPossibilities = allUnitList.get(j).getNextUnitPossibilityList();

for (int k = 0; k < nextUnitsIndexes.size(); k++) {

String[] newSequence = generateNextSequence(temporarySequence, j, nextUnitsIndexes.get(k));

int tempPosition = getPositionInVertexList(newSequence);

if (tempPosition == -1) {

vertexList.add(new Vertex(vertexList.size(), newSequence));

vertexList.get(vertexList.size() - 1).addInVertex(currentSequenceList.get(i).getTitle(), nextUnitsPossibilities.get(k));

nextSequenceList.add(new Sequence(newSequence, vertexList.get(vertexList.size() - 1).getTitle()));

if (currentVertexPosition != -1) {

vertexList.get(currentVertexPosition).addOutVertex(vertexList.get(vertexList.size() - 1).getTitle(), nextUnitsPossibilities.get(k));

} else System.out.println("Oops, something went wrong");

} else {

vertexList.get(tempPosition).addInVertex(currentSequenceList.get(i).getTitle(), nextUnitsPossibilities.get(k));

vertexList.get(currentVertexPosition).addOutVertex(vertexList.get(tempPosition).getTitle(), nextUnitsPossibilities.get(k));

}

}

}

}

}

currentSequenceList.clear();

currentSequenceList.addAll(nextSequenceList);

nextSequenceList.clear();

} else flagSuccess = false;

}

}

private static int getPositionInVertexList(String[] currentSequence) {

int position = -1;

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

if (Arrays.equals(vertexList.get(i).getSequence(), currentSequence)) position = i;

}

return position;

}

private static int getVertexPositionWithTitle(String title) {

int position = -1;

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

if (vertexList.get(i).getTitle().equals(title)) position = i;

}

return position;

}

private static String[] generateNextSequence(String[] temporarySequence, int previous, int next) {

String[] nextSequence = new String[COUNT];

System.arraycopy(temporarySequence, 0, nextSequence, 0, COUNT);

// subtract one task from previous state

int previousState = Integer.parseInt(nextSequence[previous].substring(0, 2));

if (previousState == 1) nextSequence[previous] = "001";

else if (previousState == 11) nextSequence[previous] = "010";

else {

nextSequence[previous] = String.valueOf((previousState - 10) \* 10);

}

// plus one task in next state

int nextState = Integer.parseInt(nextSequence[next].substring(0, 2));

if (nextState == 0) nextSequence[next] = "010";

else if (nextState > 0) nextSequence[next] = String.valueOf((nextState + 10) \* 10);

return nextSequence;

}

private static void printVertexList(int start, int end) {

System.out.println("Vertex quantity = " + vertexList.size());

if (start != end) {

System.out.println("\nList from M" + start + " to M" + end);

for (int i = start; i < end + 1; i++) {

System.out.println(vertexList.get(i));

}

}

}

private static void generateMatrix() {

int length = vertexList.size();

mainMatrix = new double[length][length];

lastCol = new double[length];

resultPossibilities = new double[length];

// fill result column and first row in matrix

lastCol[0] = 1.0;

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

if (i != 0) lastCol[i] = 0.;

mainMatrix[0][i] = 1;

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

mainMatrix[j][i] = 0.;

}

}

// fill main matrix

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

for (int j = 0; j < vertexList.get(i).getOutPossibilities().size(); j++) {

mainMatrix[i + 1][i] -= vertexList.get(i).getOutPossibilities().get(j);

}

for (int j = 0; j < vertexList.get(i).getInPossibilities().size(); j++) {

int colNumber = Integer.parseInt(vertexList.get(i).getInVertexList().get(j).substring(1));

mainMatrix[i + 1][colNumber] += vertexList.get(i).getInPossibilities().get(j);

}

}

// calculate all possibilities

Matrix A = new Matrix(mainMatrix);

double denominator = A.det();

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

double[][] temp = getNewMatrixWithCol(i, mainMatrix);

Matrix B = new Matrix(temp);

double coef = B.det();

resultPossibilities[i] = coef / denominator;

}

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

result[i] = 0;

}

for (int j = 0; j < vertexList.size(); j++) {

for (int i = 0; i < vertexList.get(j).getSequence().length; i++) {

if (Double.parseDouble(vertexList.get(j).getSequence()[i]) > 1) {

result[i] += resultPossibilities[j];

}

}

}

}

private static double[][] getNewMatrixWithCol(int i, double[][] mainMatrix) {

double[][] newMatrix = new double[mainMatrix.length][mainMatrix[0].length];

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

for (int k = 0; k < mainMatrix.length; k++) {

newMatrix[j][k] = mainMatrix[j][k];

}

}

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

newMatrix[j][i] = lastCol[j];

}

return newMatrix;

}

private static void printPossibilities() {

System.out.println();

System.out.println("Results : ");

double commonTime = 0.;

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

commonTime += result[i] \* 100;

}

System.out.println("CPU = " + result[0] \* 100);

System.out.println("NorthBridge = " + result[1] \* 100);

System.out.println("RAM = " + result[2] \* 100);

System.out.println("GraphicProcessor = " + result[3] \* 100);

System.out.println("SouthBridge = " + result[4] \* 100);

System.out.println("NetworkAdapter = " + result[5] \* 100);

System.out.println("AudioProcessor = " + result[6] \* 100);

System.out.println("CMD = " + result[7] \* 100);

System.out.println("ISA = " + result[8] \* 100);

System.out.println("COD = " + result[9] \* 100);

System.out.println("COM = " + result[10] \* 100);

System.out.println("\nCommon time : " + commonTime);

}

}