Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №4**

З дисципліни «Комп’ютерне моделювання»

Виконав: Перевірив:

Студент групи ІО-21 доц. Марковський О.П.

Коноз А.О.

Дата здачі\_\_\_\_\_\_\_\_\_\_\_\_\_

Захищено з балом\_\_\_\_\_

Київ 2014

**Лістинг коду**

**public** **class** Lab4 {

**public** **static** **void** main(String[] args) {

NodeSystem g = **new** NodeSystem();

g.config();

g.buildTree();

g.calcProbability(g.buildBalansSystem(g.buildDuskrGraf(g.buildUnceasingGraf(), 0.1)));

g.calcLoadingNode();

}

}

import java.text.DecimalFormat;

import java.text.NumberFormat;

import java.util.ArrayList;

import Jama.Matrix;

//import sun.launcher.resources.launcher;

import static java.lang.Math.\*;

public class NodeSystem {

// ArrayList<Level>levels = new ArrayList<Level>();

ArrayList<State> uniqueStates = new ArrayList<State>();

ArrayList<Transition> uniqueArcs = new ArrayList<Transition>();

public int nodeCount = 1;

public int conversationCount = 1;

Level previousLevel;

private double[] prop;

public void config() {

// id, queve, resource, vacant processors, name, avarege time

Node cpu = new Node(1, 4, 2, 0, "CPU", 0.00025);

Node nBridge = new Node(2, 0, 0, 1, "NordBridge", 0.005);

Node sBridge = new Node(3, 0, 0, 1, "SouthBridge", 1.0);

Node ram = new Node(4, 0, 0, 1, "Ram", 0.00066);

Node gpu = new Node(5, 0, 0, 1, "GPU", 0.001);

Node isa = new Node(6, 0, 0, 1, "ISA", 0.03);

Node lpt = new Node(7, 0, 0, 1, "LPT", 22.0);

//----------------CPU connections-----------------------

cpu.addConnections(cpu, nBridge);

cpu.addConnectionsProbabilities(0.85, 0.15);

//----------------North bridge connections--------------

nBridge.addConnections(cpu, sBridge, ram, gpu);

nBridge.addConnectionsProbabilities(0.1, 0.15, 0.5, 0.25);

//----------------RAM connections-----------------------

ram.addConnections(nBridge);

ram.addConnectionsProbabilities(1.0);

//----------------South bridge connections--------------

sBridge.addConnections(isa);

sBridge.addConnectionsProbabilities(1.0);

//----------------Graphic processor connections---------

isa.addConnections(lpt);

isa.addConnectionsProbabilities(1.0);

//----------------LPT connections-----------------------

lpt.addConnections(cpu);

lpt.addConnectionsProbabilities(1.0);

//----------------ISA connections-----------------------

gpu.addConnections(cpu);

gpu.addConnectionsProbabilities(1.0);

State defaultState = new State(cpu, nBridge, sBridge, ram, gpu, lpt, isa);

uniqueStates.add(defaultState);

Level defaultLevel = new Level(0, this);

defaultLevel.add(defaultState);

previousLevel = defaultLevel;

}

public void buildTree() {

Level nextLevel;

int i = 0;

previousLevel.printLevel();

int s = 0;

s = s + previousLevel.size();

do {

nextLevel = new Level(i + 1, this);

previousLevel.buildNextLevel(nextLevel);

i++;

previousLevel = nextLevel;

s = s + previousLevel.size();

// previousLevel.printLevel();

} while (previousLevel.isEnd());

// System.out.println("E N D");

System.out.println("Number of States: " + uniqueStates.size());

System.out.println("Number of conversations: " + uniqueArcs.size());

}

public int findThisState(State state) {

int p = -1;

for (State stt : uniqueStates) {

if (stt.equals(state)) {

p = stt.id;

break;

}

}

return p;

}

public int findThisConversation(Transition transition) {

int p = -1;

for (Transition arc : uniqueArcs) {

if (arc.equals(transition)) {

p = arc.id;

break;

}

}

return p;

}

public double[][] buildUnceasingGraf() {

double[][] matrixTransit = new double[uniqueStates.size()][uniqueStates

.size()];

for (Transition transition : uniqueArcs) {

if (transition.tallNode != transition.headNode) {

matrixTransit[transition.headNode][transition.tallNode] = transition.lambda

\* transition.probability;

}

}

return matrixTransit;

}

public double[][] buildDuskrGraf(double[][] graf, double dt) {

double[][] res = new double[graf.length][graf[0].length];

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

for (int j = 0; j < res[0].length; j++) {

if (graf[i][j] != 0)

res[i][j] = 1 - exp(-1.0 \* graf[i][j] \* dt);

}

}

return res;

}

public void calcProbability(double[][] balansMatrix) {

double[] cnst = new double[balansMatrix.length];

cnst[0]=1.0;

prop = new double[cnst.length];

Matrix balansSystem = new Matrix(balansMatrix);

Matrix balansVector = new Matrix(cnst,cnst.length);

Matrix result = balansSystem.solve(balansVector);

double [][] res =result.getArray();

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

prop[i] = res[i][0];

}

}

public void calcLoadingNode() {

double[] loading = new double[8];

for (State d : uniqueStates) {

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

if (d.devices[i].resource!=0){

if (d.devices[i].resource==1){

loading[d.devices[i].id]+=prop[d.id];

}else{

if(d.devices[i].resource==2){

loading[0]+=prop[d.id];

}

}

}

}

}

double sum = 0;

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

sum +=loading[i];

}

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

loading[i] = (loading[i]/sum);

}

NumberFormat f = new DecimalFormat("###0.000000000");

System.out.println("CPU: loading of 1 processor: "+f.format(loading[1]) + " loading of 2 processors: " + f.format(loading[0]));

System.out.println("Nord Bridge: "+f.format(loading[2]));

System.out.println("South Bridge: "+f.format(loading[3]));

System.out.println("RAM: "+f.format(loading[4]));

System.out.println("GPU: "+f.format(loading[5]));

System.out.println("LPT: "+f.format(loading[6]));

System.out.println("ISA: "+f.format(loading[7]));

}

public double[][] buildBalansSystem(double[][] graf) {

int n = graf.length;

double[][] res = new double[n][n];

for (int i = 0; i < res[0].length; i++) {

res[0][i] = 1.0;

}

for (int i = 1; i < n; i++) {

double s = 0.0;

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

if (graf[i][j] != 0)

s += graf[i][j];

}

s \*= -1.0;

res[i][i] = s;

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

if (graf[j][i] != 0)

res[i][j] = graf[j][i];

}

}

return res;

}

}

**import** java.util.ArrayList;

**public** **class** Level {

**private** ArrayList<State>states = **new** ArrayList<State>();

**public** **int** index;

**public** **int** indexItems=0;

**private** NodeSystem graf;

**public** **void** add(State state) {

//if (graf.findThisState(state)==-1){

states.add(state);

//}

}

**public** **void** buildNextLevel(Level nextLevel){

**for** (State state : states) {

**if** (state.printFlag)

state.buildNextState(nextLevel,graf);

}

}

**public** **void** printLevel(){

**for** (State state : states) {

state.printState();

System.***out***.print(" ");

}

System.***out***.println();

}

**public** Level(**int** index,NodeSystem graf) {

**this**.index = index;

**this**.graf = graf;

}

**public** **boolean** isEnd(){

**boolean** flag = **false**;

**for** (State state : states) {

flag = flag | state.printFlag;

}

**return** flag;

//return states.size()!=0;

}

**public** **int** size(){

**return** states.size();

}

}

**public** **class** State {

Node[] devices;

**boolean** printFlag=**true**;

**int** id;

**public** State(State state) {

devices = **new** Node[state.devices.length];

**int** i = 0;

**for** (Node device : state.devices) {

devices[i] =(Node) device.clone();

i++;

}

}

**public** State(Node... dev) {

devices = dev;

}

**public** Node findById(**int** id) {

Node res = **null**;

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

**if** (devices[i].id == id) {

res = devices[i];

**break**;

}

}

**return** res;

}

**public** **void** buildNextState(Level nextLevel,NodeSystem graf) {

**for** (Node deviceThis : devices) {

**if** (deviceThis.isNextState()) {

**int**[] outDev = deviceThis.outDevice;

**double** [] outProp = deviceThis.outProbability;

State nextState;

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

nextState = **new** State(**this**);

nextState.findById(deviceThis.id).removeTask();

nextState.findById(outDev[i]).addTask();

**int** markNode = graf.findThisState(nextState);

**if** (markNode==-1){

nextState.id = graf.nodeCount;

graf.nodeCount++;

graf.uniqueStates.add(nextState);

}**else**{

nextState.printFlag = **false**;

nextState.id = markNode;

}

**if** (nextState.printFlag)

nextLevel.add(nextState);

Transition transition = **new** Transition();

transition.headNode = **this**.id;

transition.tallNode = nextState.id;

transition.lambda = deviceThis.lambda;

transition.probability = outProp[i];

**int** markArc = graf.findThisConversation(transition);

**if** (markArc==-1){

transition.id = graf.conversationCount;

graf.conversationCount++;

graf.uniqueArcs.add(transition);

}**else**{

transition.id = markArc;

transition = **null**;

}

}

}

}

}

**public** **void** printState() {

**if** (printFlag){

System.***out***.print("M"+id+"{");

**for** (Node device : devices) {

System.***out***.print(device.toString());

System.***out***.print(",");

}

System.***out***.print("}");

}**else**{

System.***out***.print("M"+id);

}

}

@Override

**public** **boolean** equals(Object obj) {

State s = (State)obj;

**boolean** flag =**true**;

**for** (Node device : devices) {

Node objDevice = s.findById(device.id);

**if** (objDevice.queue != device.queue || objDevice.core!=device.core || objDevice.resource!=device.resource){

flag = **false**;

**break**;

}

}

**return** flag;

}

}

**public** **class** Transition {

**int** id;

**double** lambda;

**double** probability;

// headNode -> tallNode

**int** headNode;

**int** tallNode;

@Override

**public** **boolean** equals(Object obj) {

Transition t = (Transition) obj;

**return** t.headNode == **this**.headNode && t.tallNode == **this**.tallNode;

}

@Override

**public** String toString(){

**return** id+" "+headNode+" ---- "+lambda+"\*"+probability+"---->"+tallNode;

}

}