НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №4

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

Виконав:

студент 3 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Перевірив:

Марковський О.П.

Київ – 2014 р.

Варіант завдання: задач 6, ядер процесора 2, конфігурація системи – ВП

Лістинг коду програми

package com.cm.lab4;

public class Test {

public static void main(String[] args) {

// TODO Auto-generated method stub

long startTime = System.currentTimeMillis();

Graf g = new Graf();

g.config();

g.buildTree();

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

g.calcLoadingDevice();

long timeSpent = System.currentTimeMillis() - startTime;

System.out.println("программа выполнялась " + timeSpent + " миллисекунд");

}

}package com.cm.lab4;

public class Device {

int id;

int queue=0;

int resource=0;

int core=1;

String name;

double lambda;

Device [] outDevice;

Double[] outProbability;

public Device(int id,int strtQueue, int strtResourse, int strtCore, String name, double lambda) {

this.id = id;

this.queue = strtQueue;

this.resource = strtResourse;

this.core = strtCore;

this.name = name;

this.lambda = lambda;

// TODO Auto-generated constructor stub

}

@Override

public String toString(){

return "<"+queue+","+resource+","+core+">";

}

public void linkOutDevice(Device ...devices){

outDevice = devices;

}

public void linkOutProbability(Double ...probabilityies){

outProbability = probabilityies;

}

public void addTask(){

if (core>=1){

core--;

resource++;

}else{

queue++;

}

}

public void removeTask(){

resource--;

core++;

while(core!=0 && queue!=0){

resource++;

queue--;

core--;

}

}

public boolean isNextState(){

if (resource!=0) return true;

else return false;

}

}package com.cm.lab4;

//import java.awt.Toolkit;

import java.util.ArrayList;

import Jama.Matrix;

import org.apache.commons.math3.linear.\*;

import static java.lang.Math.\*;

public class Graf {

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

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

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

public int nodeCount = 1;

public int arcCount = 1;

Level previousLevel;

private double[] prop;

public void config() {

Device proc = new Device(1, 4, 2, 0, "Процесор", 0.25);

Device nourthBridge = new Device(2, 0, 0, 1, "Північний міст", 10.0);

Device southBridge = new Device(3, 0, 0, 1, "Південний міст", 65.0);

Device ram = new Device(4, 0, 0, 1, "Ram", 0.5);

Device videoProc = new Device(5, 0, 0, 1, "Відео процесор", 68.0);

proc.linkOutDevice(proc, nourthBridge);

proc.linkOutProbability(0.85, 0.15);

nourthBridge.linkOutDevice(proc, southBridge, ram);

nourthBridge.linkOutProbability(0.1, 0.4, 0.5);

ram.linkOutDevice(nourthBridge);

ram.linkOutProbability(1.0);

southBridge.linkOutDevice(videoProc);

southBridge.linkOutProbability(1.0);

videoProc.linkOutDevice(proc);

videoProc.linkOutProbability(1.0);

State head = new State(proc, nourthBridge, southBridge, ram, videoProc);

/\*

proc.linkOutDevice(nourthBridge);

proc.linkOutProbability(1.0);

nourthBridge.linkOutDevice(proc, nourthBridge);

nourthBridge.linkOutProbability(0.85, 0.15);

State head = new State(proc, nourthBridge);

\*/

uniqueStates.add(head);

Level headLevel = new Level(0, this);

headLevel.add(head);

previousLevel = headLevel;

}

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();

} while (previousLevel.isEnd());

for (Transition d : uniqueArcs) {

//System.out.println(d);

}

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

System.out.println("Cтанів " + uniqueStates.size());

System.out.println("Переходів " + 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 findThisArc(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 calcPropability(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];

//System.out.printf("%2.10f\n",prop[i]);

}

}

public void calcLoadingDevice() {

double[] loading = new double[6];

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{

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

}

}

}

}

System.out.println("Одне Ядро "+String.format("%3.15f", loading[1]));

System.out.println("Два Ядра "+String.format("%3.15f", loading[0]));

System.out.println("Північний міст "+String.format("%3.15f", loading[2]));

System.out.println("Південний міст "+String.format("%3.15f", loading[3]));

System.out.println("Оперативна пам\*ять "+String.format("%3.15f", loading[4]));

System.out.println("Відео процесор "+String.format("%3.15f", loading[5]));

}

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;

}

}package com.cm.lab4;

import java.util.ArrayList;

public class Level {

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

public int index;

public int indexItems=0;

private Graf graf;

public void add(State state) {

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

states.add(state);

//}

}

public void buildNextLevel(Level nextLevel){

for (State state : states) {

state.buildNextState(nextLevel,graf);

}

}

public void printLevel(){

for (State state : states) {

state.printState();

System.out.print(" ");

}

System.out.println();

}

public Level(int index,Graf 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();

}

}package com.cm.lab4;

import java.util.ArrayList;

public class State {

Device[] devices;

boolean printFlag=true;

int id;

public State(State state) {

devices = new Device[state.devices.length];

int i = 0;

for (Device device : state.devices) {

devices[i] = new Device(device.id, device.queue, device.resource,

device.core, device.name, device.lambda);

i++;

}

for (Device device : devices) {

switch (device.id) {

case 1:

//proc

device.linkOutDevice(findById(1), findById(2));

device.linkOutProbability(state.findById(1).outProbability);

//device.linkOutDevice(findById(2));

//device.linkOutProbability(state.findById(1).outProbability);

break;

case 2:

//nb

device.linkOutDevice(findById(1), findById(4), findById(3));

device.linkOutProbability(state.findById(2).outProbability);

//device.linkOutDevice(findById(1),findById(2));

//device.linkOutProbability(state.findById(2).outProbability);

break;

case 3:

//sb

device.linkOutDevice(findById(5));

device.linkOutProbability(state.findById(3).outProbability);

break;

case 4:

//ram

device.linkOutDevice(findById(2));

device.linkOutProbability(state.findById(4).outProbability);

break;

case 5:

//vp

device.linkOutDevice(findById(1));

device.linkOutProbability(state.findById(5).outProbability);

break;

default:

break;

}

}

}

public State(Device... dev) {

devices = dev;

}

public Device findById(int id) {

Device 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,Graf graf) {

for (Device deviceThis : devices) {

if (deviceThis.isNextState()) {

Device[] 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].id).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.findThisArc(transition);

if (markArc==-1){

transition.id = graf.arcCount;

graf.arcCount++;

graf.uniqueArcs.add(transition);

}else{

transition.id = markArc;

transition = null;

}

}

}

}

}

public void printState() {

if (printFlag){

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

for (Device 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 (Device device : devices) {

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

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

flag = false;

break;

}

}

return flag;

}

}package com.cm.lab4;

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;

}

}