Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота № 3**

з дисципліни «Комп’ютерне моделювання»

Виконав студент групи ІО-01

*Редько Олександр*

# Програма

## Лістинг коду

package lab23;

import java.text.DecimalFormat;

import java.util.Arrays;

public class Main3 {

public static Device[] arrayToDevices(int[][] ary) {

Device[] dev = new Device[ary.length];

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

dev[i] = new Device(ary[i]);

}

return dev;

}

public static double[] formM(int [] frequency){

double [] res=new double[frequency.length];

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

res[i]=frequency[0]/frequency[i];

}

return res;

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

int[] frq = { 3500,// processor

600,// Ram

1600,// Northern bridge

850,// GP

865,// southern bridge

250,// dc

30, // COM

100 };// ISA

double[] M = formM(frq);

double[][] P = { { 0.68, 0, 0.32, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0.57, 0.18, 0, 0.1, 0.15, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0.62, 0, 0, 0.27, 0.11, 0 },

{ 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1 },

{ 0, 0, 0, 0, 0.79, 0, 0.21, 0 } };

int[][] mydevices = { { 1, 3, 0 }, { 0, 0, 1 }, { 0, 0, 1 },

{ 0, 0, 1 }, { 0, 0, 1 }, { 0, 0, 1 }, { 0, 0, 1 }, { 0, 0, 1 } };

System.out.println(Arrays.toString(M));

Tree.setDevNumb(8);

Tree.setM(M);

Tree.setP(P);

System.out.println("Tree building...");

Tree mytree = new Tree(arrayToDevices(mydevices), null, 0, 0);

mytree.buildTree();

mytree.printTree();

Tree.printSet();

System.out.println("");

double[][] graph = mytree.treeToGraph();

System.out.println("We have graph with " + graph.length + " vershin");

//Tree.printMatrix(graph);

double[][] left = mytree.formEquations(graph);

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

Tree.printMatrix(left);

double[] right = mytree.formEquation();

double[] solution = Gauss.gauss(left, right);

//System.out.println(Arrays.toString(solution));

double[] result = mytree.devicesChanse(solution);

System.out.println("Total number of states:" + (Tree.count + 1));

System.out.println("Results");

DecimalFormat formatter = new DecimalFormat("##0.####");

for (int i = 0; i < Tree.devNumb; ++i) {

System.out.println("Device №" + (i + 1) + " has working chance:"

+ formatter.format(result[i] \* 100) + "%");

}

Gauss.write(result);

}

}

package lab23;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

/\*\*

\* @author RED

\*

\*/

public class Tree {

// глобальний лічильник станів

public static int count = -1;

// номер стану

private int num;

//

// //number of parent state

// private int pnumb;

// час переходу з батьківського стану

private double ptime;

// ймовірність переходу з батьківського стану

private double p;

// Вектор-стан

Device[] vector;

// вектор працюючих девайсів

boolean[] work;

// батьківський стан

private Tree parent;

// дочірні cтани

private ArrayList<Tree> children;

// набір існуючих станів

private static MySet set = new MySet();;

// вектор середнього часу роботи пристроїв

static double[] M;

// ймовырносты переходів

static double[][] P;

// Число пристроїв

static int devNumb;

// ----------------------------------------------------------------------

// first constructor

public Tree(Device[] vector, Tree parent, double p, double pt) {

this.num = 1;

this.p = p;// p

this.ptime = pt;// pt;

this.vector = vector;

this.parent = parent;

children = new ArrayList<Tree>();

this.work = new boolean[devNumb];

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

work[i] = this.vector[i].isWork();

}

}

// /second constructor

public Tree(Tree tree, Tree parent, double p, double pt) {

this.num = tree.getNumb();

this.p = p;// p

this.ptime = pt;// pt;

this.vector = tree.getVector();

this.parent = parent;

children = new ArrayList<Tree>();

this.work = tree.getWork();

}

// set number of the devices

public static void setDevNumb(int number) {

devNumb = number;

}

// reset counter

public static void resetCount() {

count = 0;

}

// set vector of Medium times

public static boolean setM(double[] times) {

boolean did = true;

if (times.length == devNumb)

M = times;

else {

did = false;

System.out.println("Incorrect times vector");

}

return did;

}

// встановлення матриці ймовірностей переходів

public static boolean setP(double[][] PP) {

boolean did = true;

if (PP.length == devNumb) {

P = PP;

} else {

did = false;

System.out.println("Incorrect matrix");

}

return did;

}

// add new element to set

public static boolean addToSet(Tree tree) {

return set.add(tree);

}

// get батьківський стан

Tree getPerent() {

return parent;

}

// повертає вектор-стан

public Device[] getVector() {

return this.vector;

}

// повертає список дочірних станів

public ArrayList<Tree> getChildren() {

return this.children;

}

// додає стан до списку дочірних станів новий елемент

public boolean addState(Device[] vector, double p, double pt) {

return children.add(new Tree(vector, this, p, pt));

}

// another realisation of addState()

public boolean addState(Tree tree, double p, double pt) {

return children.add(new Tree(tree, this, p, pt));

}

// return parent working time

public double getTime() {

return this.ptime;

}

// return p

public double getP() {

return this.p;

}

// print tree

public void printTree() {

if (parent != null) {

System.out.print("M" + this.parent.getNumb());

}

System.out.println(" >-----t(" + this.ptime + ")\*p(" + p + ") "

+ "---->M" + this.num + ":" + vectorToString());

for (Iterator<Tree> i = this.children.iterator(); i.hasNext();) {

i.next().printTree();

}

}

// build tree of states

public void buildTree() {

if (!Tree.set.isAvaible(this)) {

Tree.count++;

this.num = Tree.count;

Tree.addToSet(this);

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

if (work[i]) {

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

if (P[i][j] > 0) {

if (i != j) {

Device[] clone = cloneVector();

int tasks = clone[i].uploadDevice();

clone[j].addToQueue(tasks);

if (!Tree.set.isAvaible(clone)) {

addState(clone, P[i][j], M[i]);

} else {

addState(set.get(clone), P[i][j], M[i]);

}

} else {

addState(this, P[i][j], M[i]);

}

}

}

}

}

for (Iterator<Tree> i = (Iterator<Tree>) children.iterator(); i

.hasNext();) {

i.next().buildTree();

}

} else {

this.num = Tree.set.get(this.vector).getNumb();

}

}

// return string representation of devices vector

private String vectorToString() {

String s = "(";

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

s += vector[i].toMyString();

if (i != vector.length - 1)

s += ",";

}

s += ")";

return s;

}

// return number of the working devices

public int getWorkNumb() {

int numb = 0;

for (int i = 0; i < Tree.devNumb; ++i) {

if (work[i])

numb++;

}

return numb;

}

// return number of linked devices

public int getLinksNumber(int device) {

int res = 0;

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

if (P[device][i] > 0)

res++;

}

return res;

}

// compare two tree object

public static boolean compare(Tree t1, Tree t2) {

for (int i = 0; i < Tree.devNumb; ++i) {

if (!Device.compare(t1.getVector()[i], t2.getVector()[i]))

return false;

}

return true;

}

//

public boolean[] getWork() {

return this.work.clone();

}

// return id number of current state

public int getNumb() {

return this.num;

}

// clone device vector

public Device[] cloneVector() {

Device[] dev = new Device[vector.length];

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

dev[i] = vector[i].clone();

}

return dev;

}

// print set of the our states

public static void printSet() {

Tree[] wood = set.toArray();

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

System.out.println(wood[i].num + (wood[i].vectorToString()));

}

// print vector of the our devices

public static void printVector(Device[] dev) {

String s = "(";

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

s += dev[i].toMyString();

if (i != dev.length - 1)

s += ",";

else

s += ")";

}

System.out.println(s);

}

// return graph that based on this states tree

public double[][] treeToGraph() {

Tree[] list = Tree.set.toArray();

double[][] graf = new double[list.length][list.length];

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

for (Iterator<Tree> j = list[i].getChildren().iterator(); j

.hasNext();) {

Tree child = j.next();

int colum = child.getNumb();

if (i != colum) {

graf[i][colum] = child.getP() \* child.getTime();

}

}

}

return graf;

}

// using for to print our graph

public static void printMatrix(double[][] matrix) {

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

System.out.println( Arrays.toString(matrix[i]));

}

}

// this method return left part of system of equations

public double[][] formEquations(double[][] graph) {

double[][] equations = new double[set.size()][set.size()];

for (int i = 0; i <= Tree.count; ++i) {

for (int j = 0; j < (Tree.count + 1); ++j) {

equations[i][i] += graph[i][j];

if(graph[j][i]>0)

equations[i][j] = (-graph[j][i]);

}

}

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

equations[Tree.count][i] = 1;

return equations;

}

// return the right part of system of equations

public double[] formEquation() {

double[] res = new double[set.size()];

res[res.length - 1] = 1;

return res;

}

//checking working device

public boolean isWork(int device) {

return (vector[device].isWork());

}

//form resulting chances for each device

public double[] devicesChanse(double [] result){

double [] chances=new double[devNumb];

Tree [] states=set.toArray();

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

for(int j=0;j<devNumb;++j)

if(states[i].isWork(j)){

chances[j]+=result[i];

}

}

return chances;

}

}