Kpi-best

Міністерство освіти і науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота № 4

з курсу “Системне програмне забезпечення – 1”

Виконав студент 3 курсу групи ІО – 01

Редько О.М.

Перевірив: старший викладач кафедри ОТ Сімоненко А.В.

Київ – 2013

**Варіант:** 03 mod 13 = 3

3. Визначити вірогідність появи конфліктних призначень при зміні заповненості матриці "1" зв'язності від 1% до 100% для заданої розмірності від 10 до 30 з кроком 5.

**Лістинг програми**

#pragma once

#include <string>

class Lab4

{

public:

static int \*\*MoveL(int matrix[][], int i);

static int \*\*MoveC(int matrix[][], int j);

static int getSumL(int i);

static int getSumC(int j);

static int getMinL();

static int getMaxC(int i);

static int getNext1(int i, int indexJ);

static bool isConflict();

static void put1(int i, int j);

static void init();

static void main(std::wstring args[]);

static void print();

private:

static int N;

static int count1;

static int matrix[N][N];

static double res[N \* N][2];

static int reduced;

};

#include "Lab4.h"

int \*\*Lab4::MoveL(int matrix[][], int i)

{

for (int j = 0; j < sizeof(matrix) / sizeof(matrix[0]); j++)

{

int temp = matrix[reduced][j];

matrix[reduced][j] = matrix[i][j];

matrix[i][j] = temp;

}

return matrix;

}

int \*\*Lab4::MoveC(int matrix[][], int j)

{

for (int i = 0; i < sizeof(matrix) / sizeof(matrix[0]); i++)

{

int temp = matrix[i][reduced];

matrix[i][reduced] = matrix[i][j];

matrix[i][j] = temp;

}

return matrix;

}

int Lab4::getSumL(int i)

{

int s = 0;

for (int j = 0; j < sizeof(matrix) / sizeof(matrix[0]); j++) {

s += matrix[i][j];

}

return s;

}

int Lab4::getSumC(int j)

{

int s = 0;

for (int i = 0; i < sizeof(matrix) / sizeof(matrix[0]); i++)

{

s += matrix[i][j];

}

return s;

}

int Lab4::getMinL()

{

int index = reduced;

int s = getSumL(reduced);

if (s == 0)

{

s = N \* N;

}

for (int i = reduced + 1; i < sizeof(matrix) / sizeof(matrix[0]); i++)

{

int tempSum = getSumL(i);

if ((s > tempSum) && (tempSum > 0))

{

s = tempSum;

index = i;

}

}

return index;

}

int Lab4::getMaxC(int i)

{

int index = reduced;

int s;

if (matrix[i][reduced] != 1)

{

s = 0;

}

else

{

s = getSumC(reduced);

}

for (int j = reduced + 1; j < sizeof(matrix) / sizeof(matrix[0]); j++)

{

if (matrix[i][j] == 1)

{

int tempSum = getSumC(j);

if (s < tempSum)

{

s = tempSum;

index = j;

}

}

}

return index;

}

int Lab4::getNext1(int i, int indexJ)

{

for (int j = indexJ + 1; j < sizeof(matrix) / sizeof(matrix[0]); j++)

{

if (matrix[i][j] == 1)

{

return j;

}

}

return -1;

}

bool Lab4::isConflict()

{

for (int i = 0; i < sizeof(matrix) / sizeof(matrix[0]); i++)

{

int sum = 0;

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

{

sum += matrix[i][j];

}

if ((sum != i + 1) && (sum != 0) && (sum != 1))

{

return true;

}

}

return false;

}

void Lab4::put1(int i, int j)

{

if (matrix[i][j] != 1)

{

matrix[i][j] = 1;

}

else

{

if (j + 1 < N)

{

put1(i, j + 1);

}

else

{

if (i + 1 < N)

{

put1(i + 1, 0);

}

else

{

put1(0, 0);

}

}

}

}

void Lab4::init()

{

reduced = 0;

matrix = RectangularArrays::ReturnRectangularIntArray(N, N);

for (int i = 0; i < count1; i++)

{

put1(static\_cast<int>(Math::random() \* N), static\_cast<int>(Math::random() \* N));

}

}

void Lab4::main(std::wstring args[])

{

for (int i = 0; i < N \* N; i++)

{

count1 = i + 1;

res[i][1] = static\_cast<double>(count1)\*100 / (N\*N);

for (int j = 0; j < 100; j++)

{

for (int z = 0; z < 100; z++)

{

init();

while (reduced < sizeof(matrix) / sizeof(matrix[0]))

{

matrix = MoveL(matrix, getMinL());

matrix = MoveC(matrix, getMaxC(reduced));

reduced++;

}

if (isConflict())

{

res[i][0]++;

}

}

}

res[i][0] /= 100;

std::cout << res[i][1] + std::wstring(L"; ") + res[i][0] + std::wstring(L";") << std::endl;

}

}

void Lab4::print()

{

for (int i = 0; i < sizeof(matrix) / sizeof(matrix[0]); i++)

{

for (int j = 0; j < sizeof(matrix) / sizeof(matrix[0]); j++)

{

std::cout << matrix[i][j] + std::wstring(L" ");

}

std::cout << std::endl;

}

}

public class RectangularArrays

{

public:

static int\*\* ReturnRectangularIntArray(int Size1, int Size2) {

int\*\* Array;

if (Size1 > -1)

{

Array = new int\*[Size1];

if (Size2 > -1)

{

for (int Array1 = 0; Array1 < Size1; Array1++)

{

Array[Array1] = new int[Size2];

}

}

}

return Array;

}

};

**Графіки залежності ймовірності виникнення конфліктного призначення від наповненості матриці**



