МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**Отчет по лабораторной работе № 2**

по дисциплине: ”Распределенная и параллельная обработка данных”

на тему: ***”*** ***Программная реализация и экспериментальное исследование алгоритма планирования ASAP ”***

Выполнил**:** студент группы 10701214 Зубарев А. А.

Принял**:** проф. Прихожий А.А.

Минск 2017

# Лабораторная работа 2. Программная реализация и экспериментальное исследование алгоритма планирования ASAP.

**Цель работы:** Изучение алгоритма ASAP – As Soon As Possible.

## Задание

Спроектировать программу для вычисления шагов управления по заданной таблице смежности графа, количеству операций, типов и таблицы типов.

**Теоретическая часть**

ASAP (As Soon As Possible) – стратегия планирования, которая решает задачу на достижимость. В ней не имеются никаких ограничений и ищет синхронный план минимальной времени длины.

Исходные данные:

Граф предшествования операторов;

Выходные данные:

Шаги управления;

Количество процессоров каждого типа;

### Код программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab2

{

public partial class Form1 : Form

{

private CAsap Asap = new CAsap();

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

Asap.File\_Load("start\_param.txt");

Asap.Planning();

ViewResults();

}

private void ViewResults()

{

// шаги

richTextBox1.Text = "Всего шагов: " + Asap.steps.Count;

for (int i = 0; i < Asap.steps.Count; i++)

{

richTextBox1.Text += "\nНа " + (i + 1) + " шаге запланированы следующие операции: ";

for (int k = 0; k < Asap.steps[i].Count; k++)

{

richTextBox1.Text += " " + (Asap.steps[i][k] + 1);

}

richTextBox1.Text += "\nТипы операций:";

for (int k = 0; k < Asap.countTypes; k++)

{

richTextBox1.Text += "\nТип " + (k + 1) + ": " + Asap.countOperationsForType[i][k];

}

}

// Количество процессоров каждого типа

richTextBox1.Text += "\n\nВсего процессоров: " + Asap.maxproc;

for (int i = 0; i < Asap.countTypes; i++)

{

richTextBox1.Text += "\nТип " + (i + 1) + ": " + Asap.countProcessorsByTypes[i];

}

// типы операций

dataGridView1.ColumnCount = Asap.countTypes + 1;

dataGridView1.RowCount = Asap.countOperations + 1;

dataGridView1.Rows[0].Cells[0].Value = "Operation \\ type";

dataGridView1.ColumnHeadersVisible = false;

dataGridView1.RowHeadersVisible = false;

for (int i = -1; i < Asap.countTypes; i++)

{

dataGridView1.Rows[0].Cells[i + 1].Style.BackColor = Color.FromArgb(110, 110, 110);

if (i > -1)

{

dataGridView1.Rows[0].Cells[i + 1].Value = i + 1;

dataGridView1.Columns[i + 1].Width = 50;

}

}

for (int i = 0; i < Asap.countOperations; i++)

{

dataGridView1.Rows[i + 1].Cells[0].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView1.Rows[i + 1].Cells[0].Value = i + 1;

}

for (int i = 0; i < Asap.countTypes; i++)

{

for (int k = 0; k < Asap.arrayTypes[i].Length; k++)

{

dataGridView1.Rows[Asap.arrayTypes[i][k]].Cells[i + 1].Style.BackColor = Color.FromArgb(82, 97, 160);

}

}

// матрица смежности

dataGridView2.ColumnCount = Asap.countOperations + 1;

dataGridView2.RowCount = Asap.countOperations + 1;

dataGridView2.ColumnHeadersVisible = false;

dataGridView2.RowHeadersVisible = false;

for (int i = 0; i < Asap.countOperations + 1; i++)

{

// по вертикали

dataGridView2.Rows[i].Cells[0].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView2.Rows[i].Cells[0].Value = i;

// по горизонали

dataGridView2.Rows[0].Cells[i].Style.BackColor = Color.FromArgb(110, 110, 110);

dataGridView2.Rows[0].Cells[i].Value = i;

dataGridView2.Columns[i].Width = 50;

}

dataGridView2.Rows[0].Cells[0].Value = "Operation \\ operation";

for (int i = 0; i < Asap.countOperations; i++)

{

for (int k = 0; k < Asap.arrayH[i].Length; k++)

{

if (Asap.arrayH[i][k] == 1)

{

dataGridView2.Rows[i + 1].Cells[k + 1].Style.BackColor = Color.FromArgb(82, 97, 160);

}

}

}

}

private void ClearView()

{

dataGridView1.ColumnCount = 1;

dataGridView1.RowCount = 1;

dataGridView2.ColumnCount = 1;

dataGridView2.RowCount = 1;

}

private void button2\_Click\_1(object sender, EventArgs e)

{

Asap.Clear\_Object();

ClearView();

Asap.File\_Load("start\_param.txt");

Asap.Planning();

ViewResults();

}

}

}

**CAsap.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Lab2

{

class CAsap

{

public int countOperations = 0;

public int countTypes = 0;

public int[][] arrayTypes;

public int[][] arrayH;

public List<List<int>> List\_chains = new List<List<int>>();

private List<List<bool>> List\_chains\_ready\_for\_step = new List<List<bool>>();

public List<List<int>> steps = new List<List<int>>();

public List<List<int>> countOperationsForType = new List<List<int>>();

public List<int> countProcessorsByTypes = new List<int>();

public int maxproc = 0;

public CAsap()

{

maxproc = 0;

}

private string Find\_value(string s)

{

int index = 0;

for (int i = 0; i < s.Length; i++)

{

if (s[i] == ':')

{

index = i + 2;

}

}

string value = "";

s += '\0';

while (s[index] != '\0')

{

value += s[index];

index++;

}

return value;

}

private int[] GetArray(string s)

{

string[] nums = s.Split(' ');

int[] arr = new int[nums.Length];

for (int i = 0; i < nums.Length; i++)

{

arr[i] = int.Parse(nums[i]);

}

return arr;

}

public void File\_Load(string filestr)

{

// получение данных из файла

StreamReader file = new StreamReader(filestr);

string buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

countOperations = Convert.ToInt32(buff);

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

countTypes = Convert.ToInt32(buff);

arrayTypes = new int[countTypes][];

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

{

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

arrayTypes[i] = GetArray(buff);

}

buff = "";

buff = file.ReadLine();

arrayH = new int[countOperations][];

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

{

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

arrayH[i] = GetArray(buff);

}

file.Close();

}

public void Planning()

{

GetChains();

List\_chains\_ready\_for\_step = Create\_clone\_chains\_ready();

WriteSteps();

CountTypesOnSteps();

CountProcessorsEveryType();

maxproc = Maxprocessors(countProcessorsByTypes);

}

private void CountProcessorsEveryType()

{

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

{

int max = 0;

for (int k = 0; k < countOperationsForType.Count; k++)

{

if (max < countOperationsForType[k][i])

{

max = countOperationsForType[i][k];

}

}

countProcessorsByTypes.Add(max);

}

}

private int Maxprocessors(List<int> L)

{

int sum = 0;

for (int i = 0; i < L.Count; i++)

{

sum += L[i];

}

return sum;

}

private void CountTypesOnSteps()

{

for (int i = 0; i < steps.Count; i++)

{

countOperationsForType.Add(new List<int>());

// добавить в список количества операций по типам нули

for (int k = 0; k < countTypes; k++)

{

countOperationsForType[i].Add(0);

for (int t = 0; t < steps[i].Count; t++)

{

if (arrayTypes[k].Contains(steps[i][t]+1))

{

countOperationsForType[i][k]++;

}

}

}

}

}

private void WriteSteps()

{

// step one

steps.Add(new List<int>());

int index\_step = 0;

// добавление на первый шаг все операции на началах чепочек

for (int i = 0; i < List\_chains.Count; i++)

{

steps[index\_step].Add(List\_chains[i][0]);

}

bool exist\_nonplanOperation = false;

// 2 и более шаги

while (true)

{

exist\_nonplanOperation = false;

// проверка на наличие нераспланированных шагов

for (int i = 0; i < List\_chains\_ready\_for\_step.Count; i++)

{

if (List\_chains\_ready\_for\_step[i].Contains(false))

{

exist\_nonplanOperation = true;

break;

}

}

// если нераспланированных шагов не осталось, то выйти

if (!exist\_nonplanOperation)

{

break;

}

// добавление нового шага

steps.Add(new List<int>());

index\_step++;

int index\_false = 0;

for (int i = 0; i < List\_chains\_ready\_for\_step.Count; i++)

{

// поиск индекса в цепи, где значение = false

index\_false = Find\_index(List\_chains\_ready\_for\_step[i]);

// если есть значение индекса и предшественники операции по индексу

// уже распланированы

if (index\_false != -1 && Check\_predecessors(List\_chains[i][index\_false]))

{

// против дублирования обработки одной и той же операции

if (!steps[index\_step].Contains(List\_chains[i][index\_false]))

{

steps[index\_step].Add(List\_chains[i][index\_false]);

}

}

}

// изменение на true всех операций, которые распределены на текущем шаге

for (int i = 0; i < List\_chains\_ready\_for\_step.Count; i++)

{

index\_false = Find\_index(List\_chains\_ready\_for\_step[i]);

if (index\_false != -1)

{

List\_chains\_ready\_for\_step[i][index\_false] = true;

}

}

}

}

private bool Check\_predecessors(int operation)

{

//проверить всех предшественников, все ли они распранированы

List<bool> predesessors = new List<bool>();

bool check = false;

for (int i = 0; i < List\_chains.Count; i++)

{

int index = Find\_index(List\_chains[i], operation);

index--;

if (index != -2)

{

predesessors.Add(List\_chains\_ready\_for\_step[i][index]);

}

}

// проверить в списке на наличие хоть одного нераспланированного

check = !predesessors.Contains(false);

return check;

}

private int Find\_index(List<bool> L)

{

int index = -1;

for(int i = 0; i < L.Count; i++)

{

if (L[i] == false)

{

index = i;

return index;

}

}

return index;

}

private int Find\_index(List<int> L, int operation)

{

int index = -1;

for (int i = 0; i < L.Count; i++)

{

if (L[i] == operation)

{

index = i;

return index;

}

}

return index;

}

private List<List<bool>> Create\_clone\_chains\_ready()

{

List<List<bool>> ready = new List<List<bool>>();

for (int i = 0; i < List\_chains.Count; i++)

{

ready.Add(new List<bool>());

for (int k = 0; k < List\_chains[i].Count; k++)

{

if (k == 0)

{

ready[i].Add(true);

}

else

{

ready[i].Add(false);

}

}

}

return ready;

}

private void GetChains()

{

// построить цепи зависимостей

List<int> List\_use = new List<int>();

List<int> Chain = new List<int>();

int operation = 0;

while (List\_use.Count < countOperations)

{

operation = 0;

while (List\_use.Contains(operation))

{

operation++;

}

Chain.Clear();

Chain.Add(operation);

List\_use.Add(operation);

while (true)

{

// найти в строке таблицы смежности следующее направление

bool index = false;

for (int i = 0; i < arrayH[operation].Length; i++)

{

if (arrayH[operation][i] == 1)

{

index = true;

operation = i;

break;

}

}

if (index)

{

Chain.Add(operation);

if (!List\_use.Contains(operation))

{

List\_use.Add(operation);

}

}

else

{

break;

}

}

List\_chains.Add(new List<int>(Chain));

}

}

private int Find\_types(int num)

{

int type = 0;

for (type = 0; type < countTypes; type++)

{

if (arrayTypes[type].Contains(num))

{

return type;

}

}

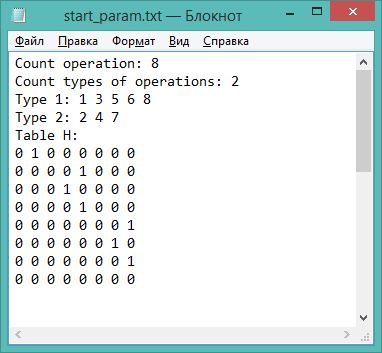
return -1;

}

}

}

### Скриншоты результатов



## Вывод

Изучен стратегия планирования ASAP.