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 № 4**

по дисциплине: ”Распределенная и параллельная обработка данных”

на тему: ***”*** ***Программная реализация и экспериментальное исследование стратегий многошагового планирования ”***

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

Принял**:** проф. Прихожий А.А.

Минск 2017

# Лабораторная работа 4. Программная реализация и экспериментальное исследование стратегий многошагового планирования.

**Цель работы:** Изучение алгоритма MS – LS.

## Задание

Спроектировать программу для вычисления шагов управления по заданной таблице смежности графа, количеству операций, типов, таблицы типов, количеству процессоров каждого типа и времени выполнения операции каждого типа.

**Теоретическая часть**

Алгоритм многошагового спискового планирования MC-LS (Multi Cycling LS) минимизирует время выполнения плана при заданных ограничениях на объем используемых вычислительных ресурсов (resource-constrained scheduling), учитывая при этом соотношение между временем шага и временами выполнения операций, назначая при необходимости операцию на несколько соседних шагов управления. Ограничения представляются числом доступных процессоров каждого типа. Алгоритм MC-LS может быть построен как на базе стратегии ASAP, так и на базе стратегии ALAP. Остановимся на алгоритме, построенном на базе ASAP.

**Исходные данные:**

1. Граф GH предшествования операций.

2. Времена ti, i ϵ N выполнения операций.

3. Время tstep шага управления.

4. Число pi доступных процессоров типа i=1,…,Types.

**Результирующие данные:**

1. Шаги управления.

2. Отображение операций на шаги управления.

**Описание алгоритма:**

1. Планирование выполняется в цикле, начиная с первого и кончая последним шагом управления. Алгоритм использует список List готовых к планированию операций. На текущий шаг могут назначаться только операции из List.

2. Список List состоит из трех частей. Первая часть List1 включает операции, планирование которых начато, но не завершено на предыдущих шагах управления. Вторая часть List2 включает операции, ставших готовыми к планированию на предыдущих шагах управления, но планирование которых еще не начато. Третья часть List3 включает операции, которые стали готовы к планированию на текущем шаге управления.

3. С каждой операцией r ϵ List ассоциируется время выполнения τr, не покрытое предыдущими шагами управления, на которые операция r частично назначена. В момент начального включения r в List время τr =tr.

4. Алгоритм начинает работу до запуска цикла с поиска операций, не имеющих согласно графу GH операций-предшественников, и включения их в часть List3 с начальной установкой значений τr =tr. Части List1 и List2 списка зануляются.

5. Для каждого шага s управления выполняются следующие действия по планированию:

a. каждая операция r ϵ List1 назначается на текущий шаг s управления; далее, если выполняется неравенство τr <Tstep, то операция r полностью спланирована и удаляется из списка List, в противном случае осуществляется редуцирование времени τr = τr - Tstep с сохранением r в List1;

b. если на планирование операций из List1 потрачена только часть процессоров, то оставшиеся процессоры могут быть использованы для конкурентного планирования операций из List2 и List3;

c. на текущий шаг управления s назначаются операции из List2 и List3, выбираемые по критерию предпочтения в количестве, не превышающем число оставшихся процессоров каждого типа; операция r, для которой τr <Tstep, назначается на шаг s и исключается из List, а операция r, для которой τr >Tstep, назначается на шаг s и перемещается в List1 с пересчетом времени τr = τr - Tstep;

d. оставшиеся в List3 операции остаются не спланированными и перемещаются в List2;

e. в List3 включаются новые еще не спланированные операции, для которых все операции-предшественники оказываются спланированными на предыдущих шагах управления, включая текущий шаг.

6. Алгоритм завершает работу в случае зануления списка List и назначения всех операций на шаги управления.

### Код программы

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Lab4\_MS\_LS

{

public partial class Form1 : Form

{

CMS\_LS LSA = new CMS\_LS();

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

LSA.Clear\_Object();

ClearView();

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

LSA.Planning();

ViewResults();

}

private void ViewResults()

{

// шаги

richTextBox1.Text = "Всего шагов: " + (LSA.steps.Count - 1);

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

{

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

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

{

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

}

richTextBox1.Text += "\nСостояние списка: ( ";

for (int t = 0; t < LSA.List\_operations\_on\_steps[i][0].Count; t++)

{

if (t == LSA.List\_operations\_on\_steps[i][0].Count - 1)

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][0][t] + 1) + " ";

else

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][0][t] + 1) + ", ";

}

richTextBox1.Text += "| ";

for (int t = 0; t < LSA.List\_operations\_on\_steps[i][1].Count; t++)

{

if (t == LSA.List\_operations\_on\_steps[i][1].Count - 1)

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][1][t] + 1) + " ";

else

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][1][t] + 1) + ", ";

}

richTextBox1.Text += "| ";

for (int t = 0; t < LSA.List\_operations\_on\_steps[i][2].Count; t++)

{

if (t == LSA.List\_operations\_on\_steps[i][2].Count - 1)

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][2][t] + 1) + " ";

else

richTextBox1.Text += (LSA.List\_operations\_on\_steps[i][2][t] + 1) + ", ";

}

richTextBox1.Text += ")\n";

}

// типы операций

dataGridView1.ColumnCount = LSA.countTypes + 1;

dataGridView1.RowCount = LSA.countOperations + 1;

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

dataGridView1.ColumnHeadersVisible = false;

dataGridView1.RowHeadersVisible = false;

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

{

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

dataGridView1.Rows[0].Cells[i + 1].Style.ForeColor = Color.FromArgb(255, 255, 255);

if (i > -1)

{

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

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

}

}

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

{

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

dataGridView1.Rows[i + 1].Cells[0].Style.ForeColor = Color.FromArgb(255, 255, 255);

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

}

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

{

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

{

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

}

}

// матрица смежности

dataGridView2.ColumnCount = LSA.countOperations + 1;

dataGridView2.RowCount = LSA.countOperations + 1;

dataGridView2.ColumnHeadersVisible = false;

dataGridView2.RowHeadersVisible = false;

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

{

// по вертикали

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

dataGridView2.Rows[i].Cells[0].Style.ForeColor = Color.FromArgb(255, 255, 255);

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

// по горизонтали

dataGridView2.Rows[0].Cells[i].Style.ForeColor = Color.FromArgb(255, 255, 255);

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 < LSA.countOperations; i++)

{

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

{

if (LSA.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;

}

}

}

**CMS\_LS.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Lab4\_MS\_LS

{

class CMS\_LS

{

public int countOperations = 0;

public int countTypes = 0;

public int[][] arrayTypes;

public int[] countProcessorsByTypes;

public int[] TimeCalculationByTypes;

public int[][] arrayH;

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

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

public List<List<List<int>>> List\_operations\_on\_steps = new List<List<List<int>>>();

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

private List<List<int>> Duration\_calculation\_by\_type = new List<List<int>>();

public CMS\_LS() { }

public void Clear\_Object()

{

countOperations = 0;

countTypes = 0;

arrayH = null;

arrayTypes = null;

countProcessorsByTypes = null;

List\_chains.Clear();

List\_chains\_ready\_for\_step.Clear();

steps.Clear();

List\_operations\_on\_steps.Clear();

}

private string Find\_value(string s)

{

int index = s.IndexOf(':') + 2;

return s.Substring(index); // извлечение подстроки с указанной позиции и до конца

}

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);

//step 1 чтение кол-ва операций

string buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

countOperations = Convert.ToInt32(buff);

//step 2 чтение кол-ва типов операций

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

countTypes = Convert.ToInt32(buff);

//step 3 чтение операций каждого типа

arrayTypes = new int[countTypes][];

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

{

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

arrayTypes[i] = GetArray(buff);

}

//step 4 чтение количества процессоров каждого типа

countProcessorsByTypes = new int[countTypes];

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

{

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

countProcessorsByTypes[i] = Convert.ToInt32(buff);

}

//step 5 чтение времени выполнения операции каждого типа

TimeCalculationByTypes = new int[countTypes];

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

{

buff = "";

buff = file.ReadLine();

buff = Find\_value(buff);

TimeCalculationByTypes[i] = Convert.ToInt32(buff);

}

//step 6 чтение таблицы смежности

buff = "";

buff = file.ReadLine();

arrayH = new int[countOperations][];

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

{

buff = "";

buff = file.ReadLine();

arrayH[i] = GetArray(buff);

}

file.Close();

}

public void Planning()

{

GetChains();

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

WriteSteps();

}

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++)

{

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) // если найдено следующее значение в цепи

{

Chain.Add(operation);

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

{

List\_use.Add(operation);

}

}

else

{

break;

}

}

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

}

}

private void WriteSteps()

{

// переменная для инициализации List\_operations\_on\_steps

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

ret.Add(new List<int>()); // выполняющиеся

ret.Add(new List<int>()); // не распределенные на предыдущем шаге

ret.Add(new List<int>()); // готовые к распределению

Duration\_calculation\_by\_type.Add(new List<int>()); // операции

Duration\_calculation\_by\_type.Add(new List<int>()); // оставшаяся продолжительность выполнения

// step zero

int index\_step = 0;

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

List\_operations\_on\_steps.Add(ret);

// добавление всех готовых к распледелению в правую часть списка

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

{

List\_operations\_on\_steps[index\_step][2].Add(List\_chains[i][0]);

}

bool exist\_nonplanOperation = false;

// 1 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++;

// список всех операциий, которые готовы к планированию

List<int> ready\_operations = new List<int>();

for (int k = 0; k < 3; k++)

{

for (int i = 0; i < List\_operations\_on\_steps[index\_step - 1][k].Count; i++)

{

ready\_operations.Add(List\_operations\_on\_steps[index\_step - 1][k][i]);

}

}

// планировка операций на текущий шаг

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

{

List<int> res = GetOperationsByType(i, ready\_operations);

foreach (int a in res)

{

steps[index\_step].Add(a);

}

}

// уменьшение продолжительности вычисления операций

for (int i = 0; i < Duration\_calculation\_by\_type[0].Count; i++)

{

Duration\_calculation\_by\_type[1][i]--;

}

List<List<int>> ret\_2 = new List<List<int>>();

ret\_2.Add(new List<int>());

ret\_2.Add(new List<int>());

ret\_2.Add(new List<int>());

List\_operations\_on\_steps.Add(ret\_2); // добавление пустого объекта

// перемещение всех нераспланированных на этом шаге операций в правую часть списка

List\_operations\_on\_steps[index\_step][1] = minus(ready\_operations, steps[index\_step]);

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

for (int i = 0; i < steps[index\_step].Count; i++)

{

if (Duration\_calculation\_by\_type[1][i] == 0)

{

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

{

if (List\_chains[k].Contains(Duration\_calculation\_by\_type[0][i]))

{

int index\_operation = List\_chains[k].IndexOf(Duration\_calculation\_by\_type[0][i]);

List\_chains\_ready\_for\_step[k][index\_operation] = true;

}

}

}

}

for (int i = 0; i < Duration\_calculation\_by\_type[0].Count; i++)

{

if (Duration\_calculation\_by\_type[1][i] == 0)

{

Duration\_calculation\_by\_type[0].RemoveAt(i);

Duration\_calculation\_by\_type[1].RemoveAt(i);

i--;

}

else

{

List\_operations\_on\_steps[index\_step][0].Add(Duration\_calculation\_by\_type[0][i]);

}

}

// поиск всех операций, готовых к распределению

int index\_false = 0;

int value\_index\_false = -1;

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 операция не содержится в левой части списка,

// то добавить в правую часть списка

if (index\_false != -1)

{

value\_index\_false = List\_chains[i][index\_false];

if (!List\_operations\_on\_steps[index\_step][0].Contains(value\_index\_false) &&

!List\_operations\_on\_steps[index\_step][1].Contains(value\_index\_false) &&

!List\_operations\_on\_steps[index\_step][2].Contains(value\_index\_false) &&

Check\_predecessors(value\_index\_false))

{

List\_operations\_on\_steps[index\_step][2].Add(value\_index\_false);

}

}

}

}

}

List<int> minus(List<int> a, List<int> b)

{

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

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

{

if (!b.Contains(a[i]))

{

res.Add(a[i]);

}

}

return res;

}

class Priority\_Operation

{

public int priority;

public int operation;

public Priority\_Operation(int priority, int operation)

{

this.priority = priority;

this.operation = operation;

}

}

// класс-компаратор по убыванию

class CompInv<T> : IComparer<T>

where T : Priority\_Operation

{

// Реализуем интерфейс IComparer<T>

public int Compare(T x, T y)

{

if (x.priority < y.priority)

return 1;

if (x.priority > y.priority)

return -1;

else return 0;

}

}

private List<int> GetOperationsByType(int type, List<int> Ready)

{

List<int> on\_step = new List<int>();

List<int> Ready\_2 = new List<int>();

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

{

// если операция в списке не того типа, то удалить из списка

if (arrayTypes[type].Contains((Ready[i]) + 1))

{

Ready\_2.Add(Ready[i]);

}

}

Ready = Ready\_2;

// определить приоритет каждой операции по длине оставшейся цепи

List<Priority\_Operation> List\_priority\_operations = new List<Priority\_Operation>();

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

{

List\_priority\_operations.Add(new Priority\_Operation(MaxLengthChain(Ready[i]), Ready[i]));

}

// сортировка по убыванию

CompInv<Priority\_Operation> cp = new CompInv<Priority\_Operation>();

List\_priority\_operations.Sort(cp);

try

{

for (int i = 0; i < countProcessorsByTypes[type]; i++)

{

if (!Duration\_calculation\_by\_type[0].Contains(List\_priority\_operations[i].operation))

{

on\_step.Add(List\_priority\_operations[i].operation);

Duration\_calculation\_by\_type[0].Add(List\_priority\_operations[i].operation);

Duration\_calculation\_by\_type[1].Add(TimeCalculationByTypes[type]);

}

else

{

on\_step.Add(List\_priority\_operations[i].operation);

}

}

}

catch (IndexOutOfRangeException) // словить, если число процессоров больше, чем доступных операций

{

return on\_step;

}

catch (ArgumentOutOfRangeException) // словить, если число процессоров больше, чем доступных операций

{

return on\_step;

}

return on\_step;

}

public int MaxLengthChain(int operation)

{

int max = 0;

for (int i = 0; i < List\_chains.Count; i++) // пройтись по всем цепям

{

if (List\_chains[i].Contains(operation)) // если цепь содержит операцию

{

// длина от операции до конца цепи

int len = List\_chains[i].Count - (List\_chains[i].IndexOf(operation) + 1);

if (max < len)

max = len;

}

}

return max;

}

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 >= 0)

{

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;

}

}

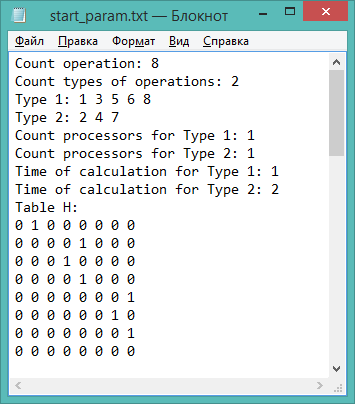
return index;

}

}

}

### Скриншоты результатов



## Вывод

Изучен стратегия планирования MS-LS–ASAP.