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 № 3**

по дисциплине: ”Распределенная и параллельная обработка данных”

на тему: ***”*** ***Программная реализация и экспериментальное исследование стратегии спискового планирования ”***

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

Принял**:** проф. Прихожий А.А.

Минск 2017

# Лабораторная работа 3. Программная реализация и экспериментальное исследование стратегии спискового планирования.

**Цель работы:** Изучение алгоритма LS – List Scheduling.

## Задание

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

**Теоретическая часть**

Алгоритм спискового планирования LS минимизирует число шагов управления и время выполнения плана при заданных ограничениях на объем используемых вычислительных ресурсов (resource-constrained scheduling). Ограничения представляются числом доступных процессоров каждого типа. Название алгоритма подчеркивает тот факт, что в процессе своей работы алгоритм активно использует список List готовых к планированию операций. Алгоритм LS может быть построен в двух вариантах: на базе стратегии ASAP и на базе стратегии ALAP. Ниже дается описание алгоритма LS, построенного на базе ASAP. ё

**Исходные данные:**

* Граф GH предшествования операций;
* Число pi доступных процессоров типа i = 1 ,…, Types.

**Результирующие данные:**

* Шаги управления;
* Отображение операций на шаги управления.

**Описание алгоритма:**

1. Планирование выполняется в цикле, начиная с первого шага и кончая последним шагом управления. В текущий шаг, который формируется на очередной итерации цикла, могут быть включены только те операции, которые находятся в списке List.
2. Список List готовых к планированию операций изменяется динамически. Он состоит из двух частей. Первая часть включает операции, оставшиеся не спланированными с предыдущей итерации цикла планирования и перенесенные на текущий шаг планирования. Вторая часть включает операции, которые стали готовы к планированию благодаря тому, что все их операции предшественники стали спланированными в результате формирования предыдущего шага управления на предыдущей итерации цикла.
3. Начальное состояние списка List формируется перед запуском цикла планирования. Первая часть списка является пустой, во вторую часть включаются операции, не имеющие операций предшественников в графе GH.
4. Когда первая и вторая части списка List становятся пустыми, то это значит, что все операции назначены на шаги управления. Алгоритм LS завершает работу.
5. Для каждого шага управления выполняются следующие действия по планированию операций: d. из списка List, равно как из первой, так и из второй части, должны быть выбраны операции, назначаемые на текущий шаг управления; e. если найдется хотя бы один тип i{1,…,Types} такой, что для подмножества LiList операций rLi, typer=i выполняется соотношение |Li|>pi, то между операциями возникает конкуренция на включение в текущий шаг управления из-за нехватки процессоров типа i; f. для предпочтительного выбора операций используются критерии выбора. Наиболее часто используемым является критерий принадлежности операции критическому пути на графе GH предшествования операций; g. на текущий шаг управления назначаются операции из списка List в количестве, не превышающем число имеющихся процессоров каждого типа; h. оставшиеся в списке List операции остаются не спланированными, перемещаются в первую часть списка и становятся претендентами на включение в следующий шаг плана; i. во вторую часть списка List включаются новые ранее не спланированные операции, для которых все операции-предшественники оказываются спланированными на текущем шаге.

### Код программы

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Lab3\_LS\_ASAP

{

public partial class Form1 : Form

{

CLS\_ASAP LSA = new CLS\_ASAP();

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;

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 += ")";

}

// 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;

}

}

}

**CLS\_ASAP.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Lab3\_LS\_ASAP

{

class CLS\_ASAP

{

public int countOperations = 0;

public int countTypes = 0;

public int[][] arrayTypes;

public int[] countProcessorsByTypes;

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

public CLS\_ASAP() {}

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 = 0;

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

{

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

{

index = i + 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 таблицы смежности

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();

}

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>());

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

// step zero

int index\_step = 0;

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

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

// добавление всех готовых к распределению в правую часть списка

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

{

List\_operations\_on\_steps[index\_step][1].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 не распланированных 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 < 2; 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);

}

}

List<List<int>> ret\_2 = new List<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 левую часть списка

List\_operations\_on\_steps[index\_step][0] = 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++)

{

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

{

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

{

int index\_operation = List\_chains[k].IndexOf(steps[index\_step][i]);

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

}

}

}

// 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) &&

Check\_predecessors(value\_index\_false))

{

List\_operations\_on\_steps[index\_step][1].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++)

{

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 распланированы

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;

}

}

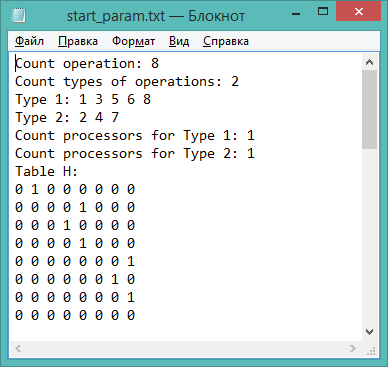
return index;

}

}

}

### Скриншоты результатов



## Вывод

Изучен стратегия планирования LS–ASAP.