БЕЛАРУСКІ НАЦЫЯНАЛЬНЫ ТЭХНІЧНЫ УНІВЕРСІТЭТ

Факультэт інфармацыйных тэхналогій і робататэхнікі

Кафедра «Праграмнае забеспячэнне вылічальнай тэхнікі і аўтаматызаваных сыстэм»

СПРАВАЗДАЧА

аб лабараторнай рабоце №2

Дысцыпліна

«Размеркаваная і паралельная апрацоўка дадзеных»

Тэма

«Праграмная рэалізацыя і эксперэметальнае даследванне алгарытма планавання ASAP»

Выканаў: студэнт гр. 10701213 Будкоўскі Г.Л.

Праверыла: Адамовіч М.М.

Мінск 2016

**План лабараторнай работы:**

1. Вывучэнне задачы планавання паралельных вылічэнняў.

2. Вывучэнне алгарытму ASAP - As Soon As Possible.

3. Распрацоўка архітэктуры праграмы.

4. Распрацоўка структур дадзеных для адлюстроўвання аперацый, графа непасрэдных папярэдніх аперацый, паслядоўна-паралельнага плану.

5. Праграмная рэалізацыя алгарытму.

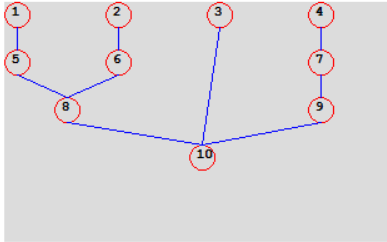
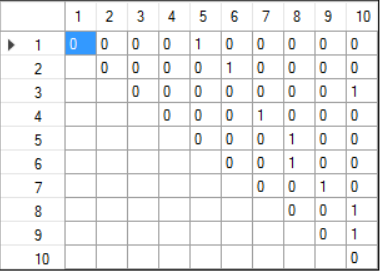
6. Наладка праграмы.

7. Правядзенне вылічальнага эксперыменту на пэўных зыходных дадзеных (узгадніць з выкладчыкам).

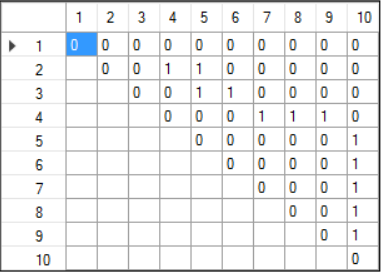
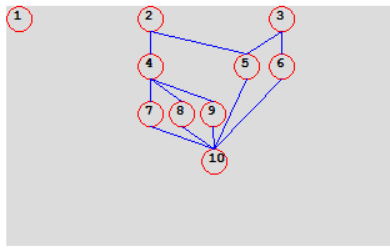
8. Здача лабараторнай працы.

**Ход выканання:**

1. Вывучыў задачу планавання.
2. Азнаёміўся з алгарытмам ASAP.
3. Была разпрацаваная патрэбная архітектура.
4. Структура графа супадае з класам “Tree”.
5. Зыходны код прыведзены ў дадатку А.
6. Быў праведзены тэставы запуск матрыцы ў параўнанні з зыходнай. Вынікі супадаюць.

1. Быў праведзены эксперымент з наступнымі дадзенымі.

Дадатак А

**Зыходны код.**

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

{

var size = Int32.Parse(this.textBox1.Text);

this.dataGridView1.ColumnCount = this.dataGridView1.RowCount = size;

for (int b = 1; b <= size; b++)

{

this.dataGridView1.Columns[b - 1].HeaderText = b.ToString();

this.dataGridView1.Rows[b - 1].HeaderCell.Value = b.ToString();

this.dataGridView1[b - 1, b - 1].Value = "0";

this.dataGridView1[b - 1, b - 1].ReadOnly = true;

this.dataGridView1.Columns[b - 1].Width = 25;

}

foreach (DataGridViewRow t in dataGridView1.Rows)

{

foreach (DataGridViewCell t1 in t.Cells)

{

if (t1.ColumnIndex > t.Index)

t1.Value = "0";

else

{

t1.ReadOnly = true;

// // t1.Style.BackColor = Color.LightGray;

// t1.Style.ForeColor = Color.DarkGray;

}

}

}

if (this.checkBox1.Checked)

{

this.dataGridView1[4, 0].Value = "1";

this.dataGridView1[5, 1].Value = "1";

this.dataGridView1[9, 2].Value = "1";

this.dataGridView1[6, 3].Value = "1";

this.dataGridView1[7, 4].Value = "1";

this.dataGridView1[7, 5].Value = "1";

this.dataGridView1[8, 6].Value = "1";

this.dataGridView1[9, 7].Value = "1";

this.dataGridView1[9, 8].Value = "1";

}

else

{

this.dataGridView1[3, 1].Value = "1";

this.dataGridView1[4, 1].Value = "1";

this.dataGridView1[4, 2].Value = "1";

this.dataGridView1[5, 2].Value = "1";

this.dataGridView1[6, 3].Value = "1";

this.dataGridView1[7, 3].Value = "1";

this.dataGridView1[8, 3].Value = "1";

this.dataGridView1[9, 4].Value = "1";

this.dataGridView1[9, 5].Value = "1";

this.dataGridView1[9, 6].Value = "1";

this.dataGridView1[9, 7].Value = "1";

this.dataGridView1[9, 8].Value = "1";

}

this.dataGridView1.RowHeadersDefaultCellStyle.Padding = new Padding(1);

if (size > 10) this.dataGridView1.ScrollBars = ScrollBars.Both;

else

{

this.dataGridView1.Height = dataGridView1.Rows.GetRowsHeight(DataGridViewElementStates.None) +

dataGridView1.ColumnHeadersHeight + 2;

dataGridView1.Width = dataGridView1.Columns.GetColumnsWidth(DataGridViewElementStates.None) +

dataGridView1.RowHeadersWidth + 2;

}

}

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

{

var g = new List<Tree>();

BuildGraph(g);

DrawGraph(g);

int max = (from s in g where s.level == 0 select s).Count();

var levels = ( from s in g select s.level).Distinct();

int curr = 0;

foreach (var f in levels)

{

curr += (from s in g where s.level == f select s).Count();

}

this.label11.Text = $"Max = {max}, levels = {levels.Count()}, MAXCPU = {max\*levels.Count()}, Curr = {curr} , % = {curr/ ((double)max \* levels.Count())}";

}

private void DrawGraph(List<Tree> trees)

{

this.richTextBox2.Clear();

var graphics = this.pictureBox1.CreateGraphics();

graphics.Clear(this.pictureBox1.BackColor);

int r = 20;

int level\_max = (from s in trees select s.level).Max() + 1;

int h = this.pictureBox1.Height/level\_max - r/2;

var arr = trees.FindAll(g => g.level == 0);

int x\_l = this.pictureBox1.Width/(arr.Count + 1);

x\_l = x\_l + ((x\_l - r)/2);

foreach (var c in arr)

{

int x = x\_l\*arr.IndexOf(c);

int y = c.level\*(h);

c.X = x;

c.Y = y;

DrawItem(graphics, c);

}

for (int i = 1; i < level\_max; i++)

{

var ar = trees.FindAll(g => g.level == i - 1);

foreach (var c in ar)

{

if (c.list.Count > 0)

{

foreach (var item in c.list)

{

if (item.level == c.level + 1)

{

int y = item.level\*(h);

if (!item.isDraw)

{

if (item.input\_count == 1 && CheckOverFalse(trees, c.X, y)) item.X = c.X;

else

{

int t = CalcXForItem(trees, item);

item.X = t;

}

item.Y = y;

DrawItem(graphics, item);

}

}

DrawLine(graphics, c, item);

}

}

}

}

for (int i = 0; i <= level\_max; i++)

{

var ar = trees.FindAll(g => g.level == i);

if (ar.Count == 0) continue;

this.richTextBox2.Text += (i + 1).ToString() + " чынам выканаць заданні \n";

ar.ForEach((s) =>

{

this.richTextBox2.Text += (s.N + " ");

});

this.richTextBox2.Text += ("\n");

}

}

public bool CheckOverFalse(List<Tree> l, int x,int y)

{

return l.FindAll(a => a.X == x && a.Y == y).Count == 0;

}

public void DrawLine(Graphics g,Tree c, Tree item,int r=20)

{

if(item.isDraw)

g.DrawLine(Pens.Blue, c.X + r / 2, c.Y + r, item.X + r / 2, item.Y);

else

{

item.drawAction += new Tree.MethodContainer(() =>

{

g.DrawLine(Pens.Blue, c.X + r / 2, c.Y + r, item.X + r / 2, item.Y);

});

}

}

public int CalcXForItem(List<Tree> c, Tree item)

{

int x = 0;

var d = from s in c where s.list.Contains(item) select s.X;

var g = from s in c where s.list.Contains(item) select s;

foreach (var b in d)

{

x += b;

}

x /= d.Count();

x += g.FirstOrDefault().list.IndexOf(item)\*25;

return x;

}

public void DrawItem(Graphics g,Tree c,int r= 20)

{

g.DrawEllipse(Pens.Red, c.X, c.Y, r, r);

var f = new Font(FontFamily.GenericMonospace, 8, FontStyle.Bold);

g.DrawString((c.N).ToString(), f, Brushes.Black, c.X+r/5, c.Y);

c.isDraw = true;

}

public void BuildGraph(List<Tree> graph)

{

int size = this.dataGridView1.ColumnCount;

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

{

var c = new Tree(i + 1, GetBranchDependeces(i));

graph.Add(c);

}

for (int i = 1; i < size; i++) {

FillLevel(graph,i);

}

}

void FillLevel(List<Tree> l,int level)

{

List<Tree> arr\_level = l.FindAll(g => g.level == level - 1);

foreach (var item in arr\_level)

{

foreach (DataGridViewCell cell in this.dataGridView1.Rows[item.N - 1].Cells)

{

if (cell.GetIntValue() == 1)

{

var dest = l.ElementAt(cell.ColumnIndex);

dest.hasRef++;

if (dest.input\_count == dest.hasRef) dest.level = level;

item.list.Add(dest);

}

}

}

}

int GetBranchDependeces(int column)

{

int count = 0;

for (int i = 0; i < this.dataGridView1.RowCount; i++)

{

DataGridViewCell cell = this.dataGridView1[column, i];

if (column > cell.RowIndex)

if (Int32.Parse(cell.Value.ToString()) == 1)

count++;

}

return count;

}

}

}