МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра СКС

Домашня контрольна робота

з дисципліни

Архітектура комп’ютерів.

Програмне забезпечення

на тему:

**«Складання графіку робіт»**

Виконав студент Перевірив:

групи КВ-64 Петрашенко А.В.

### Подольський Сергій Валентинович

залікова книжка № КВ6415 \_\_\_\_\_\_\_\_\_\_\_\_\_(бали)

#### VІІ семестр

Київ-2009

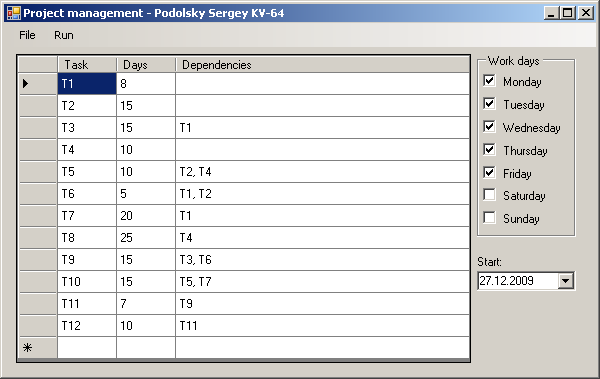
1. **Постановка задачі.**

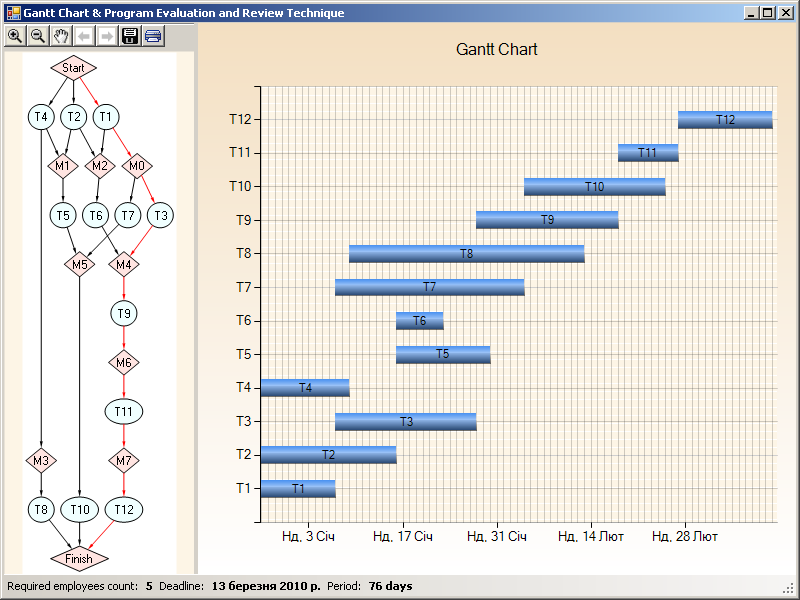
Розробити алгоритм та написати програму планування робіт.

Програма повинна реалізовувати наступні функції:

* побудова діаграми Ганта;
* складання мережевої діаграми PERT;
* знаходження критичного шляху;
* підрахунок необхідної кількості співробітників;
* визначення дат виконання етапів проекту;
* визначення терміну виконання всього проекту;
* врахування робочих та вихідних днів.

1. **Результати роботи програми.**

****

****

1. **Фрагмент тексту програми.**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Windows.Forms;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Windows.Forms.DataVisualization.Charting;

namespace ProjectManagement

{

public partial class FormGanttChart : Form

{

public FormGanttChart()

{

InitializeComponent();

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* GANTT CHART \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

chart.ChartAreas[0].AxisY.IntervalType = DateTimeIntervalType.Days;

//chart.ChartAreas[0].AxisY.Interval = 1;

// Get each task start date and finish date

HashSet<Node> Nodes = Program.formMain.GetNodes();

int maxEmployees = 0;

// While there are uncomlited tasks

DateTime currentDate = Program.formMain.dateTimePicker.Value;

for (; Nodes.Any(new Func<Node, bool>(delegate(Node node) { return node.Days > 0; })); currentDate = currentDate.AddDays(1))

{

// Start unstarted tasks that have all dependencies complited

foreach (Node node in Nodes)

if (!node.HasStarted && node.Dependencies.All(new Func<Node, bool>(delegate(Node dependency) { return dependency.Days == 0; })))

{

node.HasStarted = true;

node.StartDate = currentDate;

}

// Decrese days of uncomplited started tasks and finish them if current day was last

if (Program.formMain.checkboxes[(int)currentDate.DayOfWeek].Checked)

{

// Count of employees required at current day

int employ = 0;

foreach (Node node in Nodes)

if (node.HasStarted && node.Days > 0)

{

employ++;

if (--node.Days == 0)

node.FinishDate = currentDate;

}

// Check for loops

if (employ == 0) throw new Exception("Loop detected");

// Check if max emloyees required at current day

if (employ > maxEmployees) maxEmployees = employ;

}

}

employees.Text = maxEmployees.ToString();

deadline.Text = currentDate.ToLongDateString();

period.Text = (currentDate - Program.formMain.dateTimePicker.Value).Days.ToString() + " days";

// Set Y axis maximum an minimum dates

chart.ChartAreas[0].AxisY.Minimum = Program.formMain.dateTimePicker.Value.AddDays(-1).ToOADate();

chart.ChartAreas[0].AxisY.Maximum = currentDate.ToOADate();

// Add DataPoints to chart

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

{

Node node = Nodes.ElementAt(i);

chart.Series[0].Points.AddXY(i, node.StartDate, node.FinishDate.AddDays(1));

chart.Series[0].Points[i].Label = chart.Series[0].Points[i].AxisLabel = node.ID;

chart.Series[0].Points[i].ToolTip = node.StartDate.ToShortDateString() + " - " + node.FinishDate.ToShortDateString();

}

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* Program Evaluation and Review Technique \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

// Get union of all nodes dependencies

List<Node> allDependencies = new List<Node>();

// List of checkpoints

List<Checkpoint> checkpoints = new List<Checkpoint>();

for (int i = 0, m = 0; i < Nodes.Count; i++)

{

Node node = Nodes.ElementAt(i);

allDependencies.AddRange(node.Dependencies);

if (node.checkpoint == null)

{

// Set checkpoint to node

Checkpoint checkpoint = new Checkpoint();

checkpoint.ID = (checkpoint.Dependencies = node.Dependencies).Count == 0 ? "Start" : Checkpoint.template + m++.ToString();

node.checkpoint = checkpoint;

// Check other nodes for same dependencies

for (int j = i + 1; j < Nodes.Count; j++)

{

Node otherNode = Nodes.ElementAt(j);

if (otherNode.Dependencies.SetEquals(node.Dependencies))

otherNode.checkpoint = checkpoint;

}

checkpoints.Add(checkpoint);

}

}

// Create finish checkpoint

Checkpoint finish = new Checkpoint();

finish.ID = "Finish";

finish.Dependencies = new HashSet<Node>(Nodes.Except(allDependencies));

checkpoints.Add(finish);

// Get critical way

List<object> critical = new List<object>();

critical.Add(finish);

for (Checkpoint checkpoint = finish; checkpoint.Dependencies.Count > 0; )

{

DateTime latestDate = checkpoint.Dependencies.Max(new Func<Node, DateTime>(delegate(Node node) { return node.FinishDate; }));

Node latestNode = checkpoint.Dependencies.First(new Func<Node, bool>(delegate(Node next) { return next.FinishDate == latestDate; }));

critical.Add(checkpoint = latestNode.checkpoint);

critical.Add(latestNode);

}

// Create graph

Microsoft.Glee.Drawing.Graph graph = new Microsoft.Glee.Drawing.Graph("PERT");

// Add nodes

foreach (Node node in Nodes)

{

graph.AddNode(node.ID);

graph.FindNode(node.ID).Attr.Fillcolor = Microsoft.Glee.Drawing.Color.Azure;

}

// Add checkpoints

foreach (Checkpoint checkpoint in checkpoints)

{

graph.AddNode(checkpoint.ID);

Microsoft.Glee.Drawing.Node graphNode = graph.FindNode(checkpoint.ID);

graphNode.Attr.Fillcolor = Microsoft.Glee.Drawing.Color.MistyRose;

graphNode.Attr.Shape = Microsoft.Glee.Drawing.Shape.Diamond;

}

// Connect each node to its dependency checkpoint

foreach (Node node in Nodes)

{

Microsoft.Glee.Drawing.Edge edge = new Microsoft.Glee.Drawing.Edge(node.checkpoint.ID, null, node.ID);

if (critical.Contains(node))

edge.Attr.Color = Microsoft.Glee.Drawing.Color.Red;

graph.Edges.Add(edge);

}

// Connect each checkpoint to all dependency nodes

foreach (Checkpoint checkpoint in checkpoints)

foreach (Node node in checkpoint.Dependencies)

{

Microsoft.Glee.Drawing.Edge edge = new Microsoft.Glee.Drawing.Edge(node.ID, null, checkpoint.ID);

if (critical.Contains(node) && critical.Contains(checkpoint))

edge.Attr.Color = Microsoft.Glee.Drawing.Color.Red;

graph.Edges.Add(edge);

}

// Bind the graph to the viewer

gViewer.Graph = graph;

gViewer.OutsideAreaBrush = Brushes.OldLace;

}

}

}